DEVICEGUARD: EXTERNAL DEVICE-ASSISTED SYSTEM AND DATA SECURITY

Yipan Deng

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science & Applications

Danfeng Yao, Chair
Dennis G. Kafura
Ali R. Butt

May 2nd, 2011
Blacksburg, VA

Keywords: Host Security, System Security, Data Security, Smartphone

DEVICE GUARD: EXTERNAL DEVICE-ASSISTED SYSTEM AND DATA SECURITY

Yipan Deng

ABSTRACT

This thesis addresses the threat that personal computer faced from malware when the personal computer is connected to the Internet. Traditional host-based security approaches, such as anti-virus scanning protect the host from virus, worms, Trojans and other malwares. One of the issues of the host-based security approaches is that when the operating system is compromised by the malware, the antivirus software also becomes vulnerable.

In this thesis, we present a novel approach through using an external device to enhance the host security by offloading the security solution from the host to the external device. We describe the design of the DeviceGuard framework that separate the security solution from the host and offload it to the external device, a Trusted Device. The architecture of the DeviceGuard consists of two components, the DeviceGuard application on the Trusted Device and a DeviceGuard daemon on the host.

Our prototype based on Android Development Phone (ADP) shows the feasibilities and efficiency of our approach to provide security features including system file and user data integrity monitoring, secure signing and secure decryption. We use Bluetooth as the communication protocol between the host and the Trusted Device. Our experiment results indicates a practical Bluetooth throughput at about 2M Bytes per second is sufficient for short range communication between the host and the Trusted Device; Message digest with SHA-512, digital signing with 1024 bits signature and secure decryption with AES 256 bits on the Trusted device takes only the scale of 10^1 and 10^3 ms for 1K bytes and 1M bytes respectively which are also shows the feasibility and efficiency of the DeviceGuard solution.

We also investigated the use of embedded system as the Trusted Device. Our solution takes advantage of the proliferation of devices, such as Smartphone, for stronger system and data security.

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Dr. Danfeng Yao, for his guidance, patience, support and assistance in all phases of my graduate study. I appreciate the opportunity she provided me to work on this project and I'm grateful for his continuous help and advice. Her intelligent, careful, responsive and diligent attitude towards her work is impressive. Besides coursework and research, Dr. Yao's caring for me and all the students in the research group brought many warm memories.

I would like to thank all the professors and staffs at Virginia Tech who guided me in the class, for the research and in my graduate life.

I would like to thank all my friends for sharing many good times with me.

In the end, I would like to deeply appreciate my parents, my grandparents, my siblings and all my close relatives. They always support me, guide me, and care for me on my path to my dreams. I could not accompany and remain with them for over 2 years because of the long distance on the earth between us. But I can always feel their love. I hope I can make you happy and proud.

ATTRIBUTIONS

My committee chair Dr. Danfeng Yao, supervised and helped me throughout the project by providing great ideas and numerous guidance.

Dr. Kafura provided many great suggestions at the early stage of the project.

All photos by author, unless otherwise stated in the citation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1. 1 MOTIVATION

It has been decades since personal computer come into existence; people now are more and more relying on personal computer both in work and daily life. The trend to store business and personal information digitally in the computer make personal computers a valuable asset nowadays. The advancement in high speed network make personal computers connected in the virtual world, brings conveniences in obtaining information and alternative ways of communication.

On the other hand, as personal computer stays on-line, it becomes the target of malware. Simply put, virus, worms, Trojan and malicious code can all be categorized in to malware. In Symantec's Internet Security Report for 2010[1], we have seen 93% increase of malware comparing to 2009; There are about 3 million attacks recorded by Symantec; As much as 286 millions of unique malware were detected in 2010. Consistently, According to a recent white paper on security trend for 2011 from Imperva, a leading data security company, Malware threats will continue to increase in 2011[2]. Business and personal data and system integrity are at risk in such kind of environment. It also arise the attention in end users of the personal computing towards the security and privacy concern on using the computing. One the top 10 security trend in 2011 as predicted by Imperva, data in the form of file will be in the central of security breaches[2].

Traditional host-based approach, such as the anti-virus software running on the personal computer, actively protects the host from malware. One of the issue of this type of approach is that when the system is compromised by the malware, the anti-virus software is also vulnerable thus cannot guarantee to provide the expected protection to the host. According to [3], in order to implement new components in the malware and upgrade the malware, malware creators not only teamed up to use the established business model, but also start to use the state-of-art development tool and architecture to evolve the malware, which make it difficult for anti-virus software to catch up and provide the remedies; Besides, there is exists a delay for anti-virus software to detect the newly released malware before they can be analyzed and treated. This trend continues to be an issue in this battle, thus finding a way to mitigate this issue is becoming even more important.

Trusted Platform Module (TPM) was introduced to personal computing by providing the primary scope of assuring the platform integrity. TPM, as the public specification, is standardized by Trusted Computing Group (TCG) to store the cryptographic keys in the TPM chip and provide inter-operative interfaces to achieve the goal of securing the platform. The key features of TPM chip (or TMP Security Device) including secure generation of cryptographic key, access control, remote attestation and storage sealing, etc.[4] With TPM in place, together with proper integration to the existing security solution, it can provide additional protection to the system and anti-virus software. Nevertheless, due to various reasons, currently only few personal computer vendors including Lenovo, Fujitsu and HP had integrated the TPM chip in their latest products. As a result, we are trying to seek alternative approaches to address the problem we identified earlier, either a TPM like approach or otherwise.

Past few years have seen a rapid grows in the Smartphone market, Smartphone as a type of handheld device is increasing to become dominant in the mobile industry. According to Berg Insight [5], comparing to 2009, Smartphone shipment grew 74 percent in 2010 to 295 million units. The forecast for the Smartphone shipment in 2015 will reach 1.2 billion units. Smartphone, is physically held and trusted by the users, possess substantial computing capabilities and certain memory space. It also has rich communication interfaces to communicate with other devices and the Internet, such as GSM/CDMA, Wi-Fi, Bluetooth, Infrared etc.

A wild idea come out of our mind: Why not use the Smartphone as an external device to offload some security features from the Host to provide additional protections to the Host without additional cost? To turn users' Smartphone as a Trusted Device in their daily computerized life. As a matter of fact, Smartphone is just a type of Trusted Device; any device that has the following characteristics will meet our requirement and falls into the definition of Trusted Device.

1) The device should be physically held by user.
2) The device should possess certain computation capability.
3) The device should possess certain memory and disk space.
4) The device should possess certain communication channel to other devices

As the following sections will reveal, Android Dev Phone 2 and a Linux Damp Box are selected to be two different types of Trusted Devices in this project. They all meet the minimum requirements of the definition of the Trusted Device.

With Trusted Device in place, we are going to present our DeviceGuard framework which based on the Trusted Device to enhance the host security by offloading the system and data integrity monitoring from the host to the Trusted Device.

1. 2 CONTRIBUTIONS

The contributions of this research including:

1) We presented a novel approach through using an external device to enhance the host security.

2) We described the design of the DeviceGuard framework that separates the traditional host-based security solution from the host and offload it to the external device, a Trusted Device.

3) Our prototype based on Android Development Phone (ADP) shows the feasibilities and efficiency of our approach to provide security features including system file and user data integrity monitoring, secure signing and secure decryption.

4) Our solution takes advantage of the proliferation of devices, such as Smartphone, for stronger system and data security.

1. 3 THESIS ORGANIZATION

The rest of the thesis will be organized as follows: CHAPTER 2 provides background on host security researches, Bluetooth and Android Dev Phone. CHAPTER 3 explains the threat model we focused in this paper and our security goals; CHAPTER 4 describes our design on the DeviceGuard framework, including the overall architecture and detailed operations in our applications; CHAPTER 5 provides the details of implementation of the DeviceGuard framework and solution cases; CHAPTER 6 presents our experiment setup and results; CHAPTER 7 discusses pitfalls in our framework and potential remedies, etc.; CHAPTER 8 wraps up with the conclusion and future work.

CHAPTER 2 BACKGROUND

This chapter provides background information which is related to our research. Section 2.1 presents a review on relevant literatures to the research in this thesis. Section 2.2 covers information on Bluetooth technologies, including Bluetooth specification, Bluetooth Security and Bluetooth protocol stacks. Section 2.3 provides background on Android Dev Phone which is relevant to this research.

2. 1 LITERATURE REVIEW

2.1.1 Introduction

The goal of the research presented in this thesis is to exploit a solution that can separate the security functionality from the host in order to mitigate the malware compromise. Prior to achieve this goal, we have done a review of relevant literatures, including malware taxonomy, boot-time checking, runtime checking and device-assisted security by the literatures.

2.1.2 Malware Taxonomy

Malware, or malicious software, can be defined as a generic term which put virus, worms, Trojans and other malicious codes into this single category, the malware[6]. While in[7], Joanna gave the bellowing definition to Malware that:

"Malware is a piece of code which changes the behavior of either the operating system kernel or some security sensitive applications, without a user consent and in such a way that it is then impossible to detect those changes using a documented features of the operating system or the application (e.g. API)."[7]

Malware is classified into four types, with Type 0 Malware defined as a type of malware that does not interact with the operating system using undocumented methods to reach its malicious goal; Type I Malware as a type of malware that will modify the relatively constant system resources, such as the BIOS code or running kernel's in-memory code section. They usually attack the system by hooking to different level of the system architecture from high level function modules to interrupt handlers; Type II Malware is relatively different from type I

malware, this type of malware will only modify the system resources which are dynamic by nature, such as the data section; and a Type III Malware that can control the whole operating system, such as the Blue Pill[8], which uses the hardware virtualization technologies.[7]

For the type 0 malware and type I malware, we could also roughly classify them into a user space malware and kernel space malware or root-kit. As the research in [9] shows, kernel malware is becoming one the biggest threat in the malware family.

Although the malware classification still follow the above over the time, but according to [3], the evolution of malware is accelerating in terms of the malware creation is becoming a business and state-of-the-art software engineering practice is introduced in this process for malware evolving. The prediction for malware in 2011 from Imperva also indicates an increase in this battle.[2]

In our research, we looks into the user space malware and kernel space malware which does not compromise the kernel functionalities. More specifically, our framework will enable the detection of the malware that tampers with the file system on the host and the malware that steals the private key on the host.

2.1.3 Boot-Time Checking

Boot-time checking is mainly used to perform a verification of the kernel image before the kernel is launched. One way to achieve this goal is to use the TPM [4] to keep a secret signature of the kernel image, verify the current image with this signature before it is loaded into the memory. While the TPM-based solution requires the commodity computer equipped with the TPM chip, our DeviceGuard solution doesn't not rely on any specialized chip to functioning.

2.1.4 Run-time Checking

Run-time checking for system and data security is one of the most common approaches in facing the threats from various types malware. There are a bunch of researches had been done in this direction.

ATP [10], the anti tampering program, is a tool that enables post-attack analysis for the compromises in file tempering. Tripwire[11] extended the research in ATP to further working on the file integrity checking during run-time. While both ATP and Tripwire are solutions that runs

as software on the un-trusted host, our DeviceGuard solution will only require a minimum daemon runs of the host to provide similar file integrity monitoring to the host.

In [12], Patil et al. indicated that the traditional integrity check and IDS is less efficient when performing scheduled task during the run-time not only because they cannot find the compromised between checks in time but also the performance downgrade will be introduced when the scheduled task is ongoing . They introduced a new file system which enables an on-access check in real time. Similarly, this approach not only runs on the un-trusted host but also require major modifications to the file system to achieve the security goal.

Copilot [13], a coprocessor-based kernel runtime integrity monitor, is designed for detecting malicious modifications to the kernel. Since it uses a plug-in device to the PCI and thus granted the access to the memory and file system, there are no additional modifications to the operating system, it provides a transparent protection to the kernel runtime integrity. Copilot works in a way that requires to plug-in a PCI card and physically connected to a monitoring computer via a serial cable.

SVFS [14], a Secure Virtual File System, is proposed to use the virtual machine technology so that all the sensitive file will be stored in the virtual file system and all the access is subject to the access control. This research is more on preventing the system and data integrity instead of performing a run-time checking.

2.1.5 Device-Assisted Security

Device-assisted security is an emerging direction that takes advantage of the proliferation of the devices people are publicly available, such as Smartphone. The use of external device showcases the potential opportunity in explore the device assisted security.

In Bump in the Ether[15], the author designed a framework for securing sensitive user input by offloading the user input to a phone with Bluetooth as the underlying communication channel. Another work in Bumpy[16], which is similar to Bump in the Ether[15] in terms of leverage the encryption of user input with another device, the author addressed the threats from the malware for password and sensitive data. The system they introduced, without using a VMM or hypervisor, can exclude the operating system and application in the sensitive information input. The difference between our DeviceGuard solution and the Bumpy lies in that Bumpy requires a version 1.2 TPM and a chipset which enables late launch, or the ability to

establish a Dynamic root of trust, while our approach does not bind to TPM as the specialized chip in the host. Other researches related to using a device for enhancing the system and data security can be found in [17-19].

Lockheed Martin together with IronKey introduced the 'PC on a stick' flash drive, providing a secure and portable device where user can plug the device in any host computer and boot directly from the device. It will bypass the hard disk on the host computer, running user's own operating system, files, applications from the flash drive[20]. This work differs from our work in the sense that the user is using a virtual environment within the USB device while still consuming the computation capability in the host. Our DeviceGuard solution will totally offload the security features to run on an external device, which provides another level of isolation from the malware threats.

2. 2 BLUETOOTH

Bluetooth is a proprietary short-range wireless communication technology for exchanging data from Bluetooth enabled devices. The radio technology Bluetooth used, frequency-hopping spread spectrum, enables the data segment being transmitted up to 79 bands. Since Bluetooth can create a personal area network (PAN) among the devices, it exhibits very high level of security for short-range communication. As a cable replacement technology to USB and serial cables, which originally created by Ericsson in 1994, it is especially favored by embedded and portable devices.

Bluetooth Protocol Stack is available under different operating systems, but the implementation is independent among them. In Linux community, BlueZ is now the official Linux Bluetooth Protocol Stack. Since Linux kernel version 2.4.6, BlueZ is part of the official kernel[21]. Figure 2-1 gives an overview of the BlueZ protocol stack diagram.

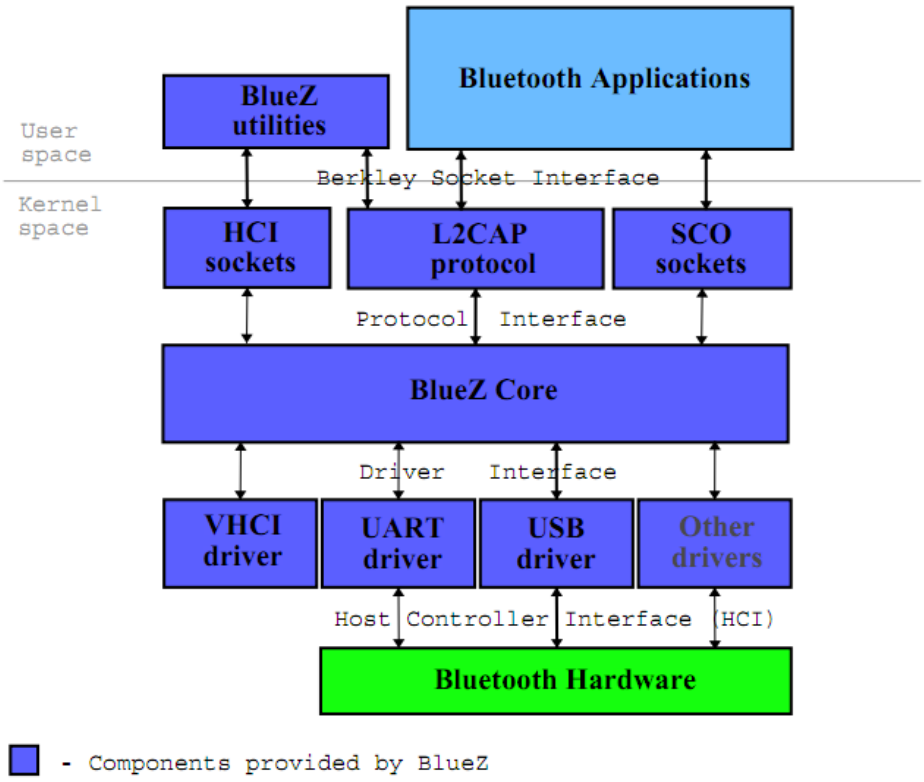

Figure 2-1 BlueZ Protocol Stack Overview Diagram[22]

BlueCove is a Java library for Bluetooth (JSR-82 implementation) interfaces with Bluetooth protocol stacks in Windows, Mac OS X and Linux. Particularly, BlueCove-GPL is an additional module licensed under GPL to support BlueCove runtime on BlueZ.[23] One of the significant contributions of BlueCove is that it empowers developers to develop Bluetooth related application under the general framework of Java environment. Figure 2-2 shows the overview diagram of BlueCove-enabled Bluetooth Protocol Stack.

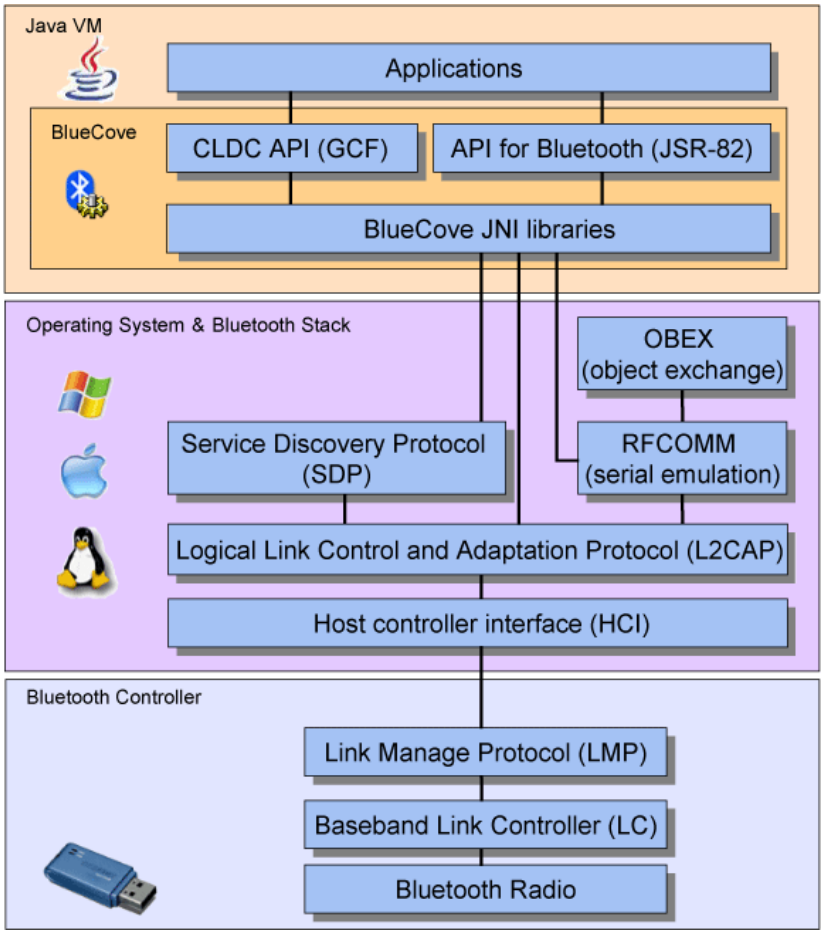

Figure 2-2 BlueCove-enabled Protocol Stack Diagram[23]

2. 3 ANDROID DEV PHONE

Android Dev Phone is released by Google to enable developers to run and debug Android applications directly on the device, which is normally not supported by the Smartphone as consumer products from various carriers. The Android Dev Phones are carrier-independent, and available for purchase by developers through their Android Market publisher accounts. Thus Android Dev Phone supports developers to modify and rebuild the Android operating system, and flash it onto the phone at will. We will introduce more detail of the device in sub-section 2.3.1; background of the Android operating system and framework API in sub-section 2.3.2, and available communication interfaces in sub-section 2.3.3.

2.3.1 Device

Since the release of Android version 1.1, in order to facilitate developer in contributing to the Android Market by publishing useful and funny applications, Google provide unlocked and carrier independent Android Dev Phone to qualifying developers. Almost every year, Google releases a newer version of Android Dev Phone during the Google IO conference which attracts innumerous Android developers to share most recent advancement in related technologies. Android Dev Phone™ is followed by Android Dev Phone 2™and Nexus One™. [24]

The device we are using is Android Dev Phone 2™, equips with Android 1.6 natively. Detailed specification will be introduced in 6.1 on the Trusted Device.

2.3.2 Operating System and Framework API

Android is a software stack for mobile devices that includes an operating system, middleware and key applications.[25]

"The Android open-source software stack consists of Java applications running on a Java-based, object-oriented application framework on top of Java core libraries running on a Dalvik virtual machine featuring JIT compilation. Libraries written in C include the surface manager, OpenCore media framework, SQLite relational database management system, OpenGL ES 2.0 3D graphics API, WebKit layout engine, SGL graphics engine, SSL, and Bionic libc. The Android operating system, including the Linux kernel, consists of roughly 12 million lines of code including 3 million lines of XML, 2.8 million lines of C, 2.1 million lines of Java, and 1.75 million lines of C++. " [25]

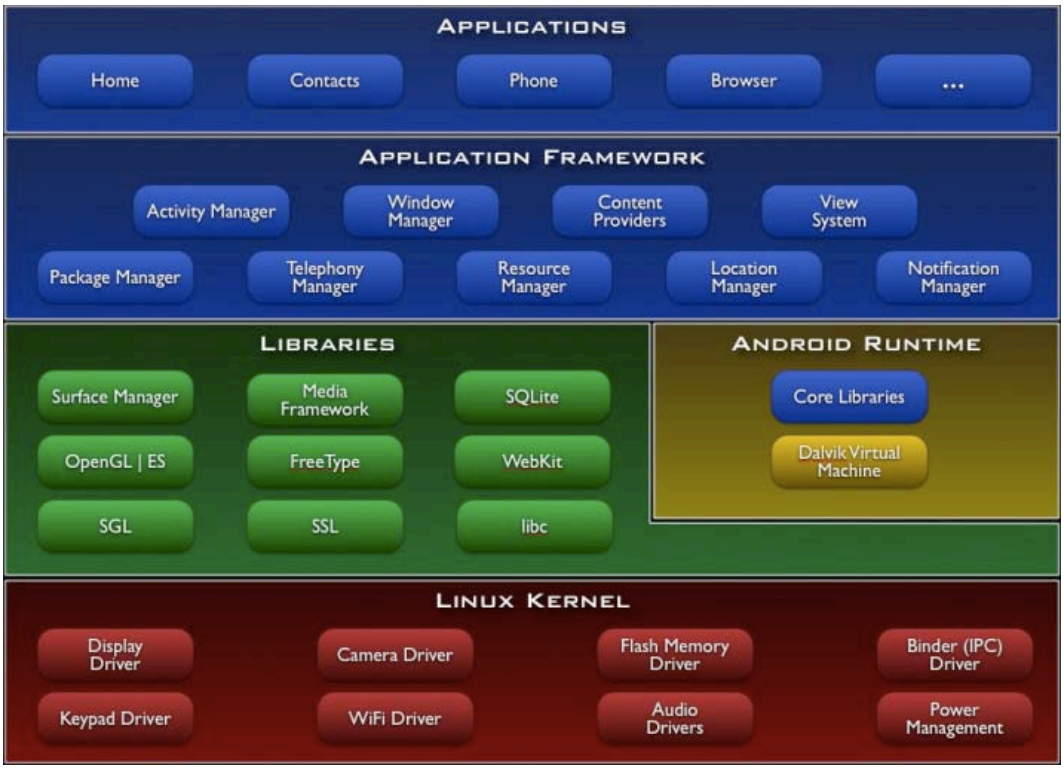

Figure 2-3 Android Architecture

The architecture of Android in Figure 2-3 shows the major components in the Android operating system, including: Linux Kernel, Android Runtime, Libraries, Application Framework and the application on the top.

As an effort to make the open sourced Android easily accessed by developers, it also continues to provide rich framework APIs which covers almost every technology that currently existing in the Smartphone. These efforts turn Android to be one of the most popular Smartphone operating system in the market.

2.3.3 Communication Interface

ADP supports rich communication interfaces including wireless 802.11 b/c, Bluetooth and the Android debug bridge based on USB. *"Starts from Android 2.0, the Android platform includes support for the Bluetooth network stack, which allows a device to wirelessly exchange data with other Bluetooth devices. The application framework provides access to the Bluetooth functionality through the Android Bluetooth APIs. These APIs let applications wirelessly connect to other Bluetooth devices, enabling point-to-point and multipoint wireless features."* [25]

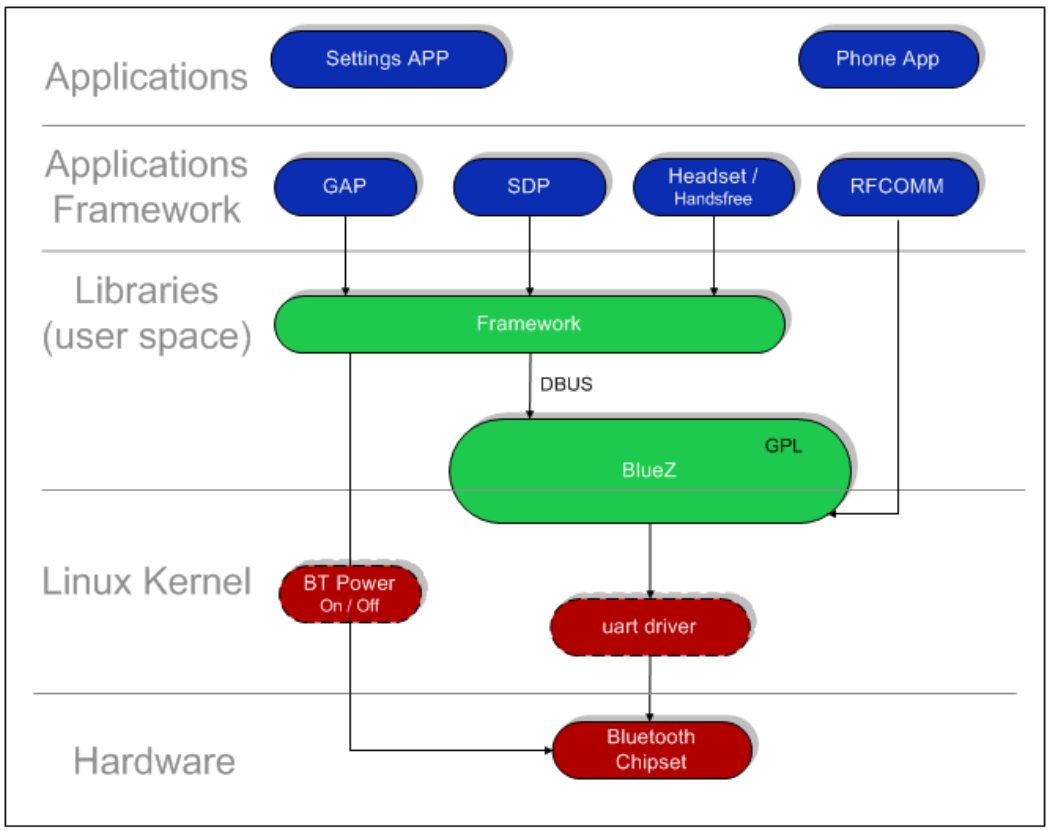

Figure 2-4 Android Blueetooth Diagram[26]

As shown in the Figure 2-4, Android Bluetooth Diagram presents the different level of the Bluetooth stack which internally based on the BlueZ protocol stack. BlueZ is used because Android borrows the kernel from Linux, where officially integrated BlueZ as the default Bluetooth protocol stack. The applications framework build upon BlueZ provides a uniformed Java API for accessing the Bluetooth resources.

CHAPTER 3 THREAT MODEL AND SECURITY GOALS

DeviceGuard, focused on providing a novel approach by using a Trust Device to enhance system and data integrity, but this is not a one for all solution. In this chapter, we will introduce the threat model that we are going to address in section 3.1 and the assumptions that we make for this research in section 3.2.

3. 1 THREAT MODEL

The thread model behind this project stems from the observation of increasingly affect resulted from malware and worms. Although anti-virus software can scan and capture some of the malware and worms, under the severe condition when anti-virus software is also compromised, it is no longer been trusted to provide guaranteed service and protection to the personal computer.

The adversary (Malware, etc.) in the compromised host might either try to temper the system or data integrity and even the anti-virus software installed in the compromised host to hide their trace. By separating the security solution from the security object, the personal computer, we are able to mitigate the influence of malware and worms on host to the anti-virus software.

Since we still have a DeviceGuard daemon in the Host, more specifically running in user space, the adversary may try to compromise the daemon in order to stay undercover. We will discuss in CHAPTER 7 with more details on security threats in our solutions and potential remedies.

More specifically, according to the malware taxonomy in [7], there are four types of malwares. We will discuss the security threats from each type of the malware below:

- Type 0 Malware: This type of malware dose not interact with the operating system using undocumented methods to reach its malicious goal[7]. Although they are not aiming at compromising the system, they are still malicious enough to be able to delete user's personal files or important user data, etc.

- Type I Malware: This type of malware will modify the relatively constant system resources, such as the BIOS code or running kernel's in-memory code section. They usually attack the system by hooking to different level of the system architecture from high level function modules to interrupt handlers.[7] It could be easy to detect the system and data security compromise by simply perform the scanning techniques, but to detect Type I malware will require the verification on the integrity of constant system resources that we mentioned earlier.

- Type II Malware: Different from type I malware, this type of malware will only modify the system resources which are dynamic by nature, such as the data section[7]. Since it is difficult to verify the signature of the dynamic system resources, actually there is no consistent signature for that, we cannot detect the type II malware as we did to detect type I malware. But similarly, type II malware is also able to compromise the system and data security, thus detecting the compromises still a goal for us in this case.

- Type III Malware: It is a type of malware that can control the whole operating system, such as the Blue Pill[8], which uses the hardware virtualization technologies[7]. We are not going to deal with this type of malware in this thesis.

3. 2 ASSUMPTIONS

In addressing the data and system security problem aroused on the personal computer, we have the following assumptions in our solution to clearly define a research boundary:

1) The Host which vulnerable to compromises is NOT trusted. We can imagine it has already been compromised by the malware of certain type.

2) The system kernel of the Host operating system is trusted. As we are still rely on the system kernel to provide the basic functional for the DeviceGuard daemon to obtain system information and file system manipulation

3) The DeviceGuard daemon on the Host is trusted. Only when the daemon is trusted, can we make sure the DeviceGuard solution is trusted.

3. 3 SECURITY GOALS

In our DeviceGuard framework, we are targeting a few security goals to evaluate how our framework works to address the problem we discussed in this research. In general, DeviceGuard framework will be enable the Trusted Device to provide additional system and data security to the un-trusted Host. More specifically, we are targeting at the malware that tempers with the file system on the Host and the malware that steals the private key on the Host. We will present in later chapters for the use cases in our DeviceGuard framework to demonstrate these security goal in details.

CHAPTER 4 OVERVIEW OF DESIGN

In this chapter, we will describe the detailed design of our solution, including the overall architecture communication protocol and application design.

4. 1 ARCHITECTURE

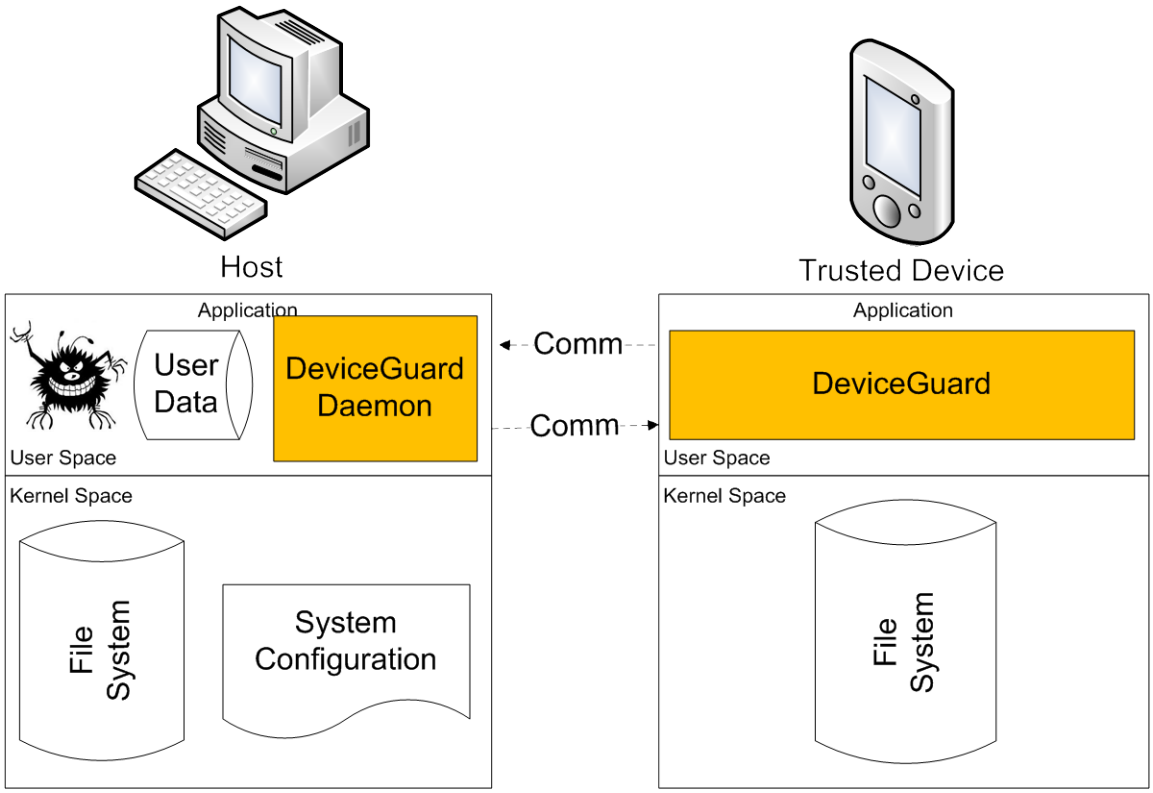

Figure 4-1 DeviceGuard Architecture

In Figure 4-1, we present the architecture of our DeviceGuard framework. Host as a personal computer is the target machine which is vulnerable to compromises. The Trusted Device is an external device which offloads part of the traditional host-based security solution from the Host. It will provide security enhancement to the Host and using a specified communication interface for communication, such as the Bluetooth, between the Host and the Trusted Device.

On the Trusted Device, where the DeviceGuard framework mainly focused on, we designed and implemented the communication protocol and security features in the

DeviceGuard application that will assist the host security. Besides that, the communication interface and a method for secure communicate upon the specific communication interface is also devised.

On the host side, since we still need an agent to assist the DeviceGuard on the Trusted Device to execute certain command in order to reach our security goal, we designed and implemented a DeviceGuard daemon. The daemon interfaces the agreed and configured communication channel between the Trusted Device and the Host. It proactively listens to the communication channel for commands from the Trusted Device, executes the command as requested. The DeviceGuard daemon is designed to be lightweight, with limited logic within such that our assumption on the trustworthy of this daemon is reasonable.

The DeviceGuard framework is in part based on a Challenge-Response mechanism to proactively assist the data and system security on the Host. In order to demonstrate our DeviceGuard solution, we pick up three different cases where the DeviceGuard could be applied to assist the system and data security in the practice.

4. 2 CASE 1: FILE INTEGRITY MONITORING

File Integrity Monitoring demonstrates a case using external device to monitor and challenge the file integrity from potential compromises. Figure 4-2 shows the application architecture and protocols.

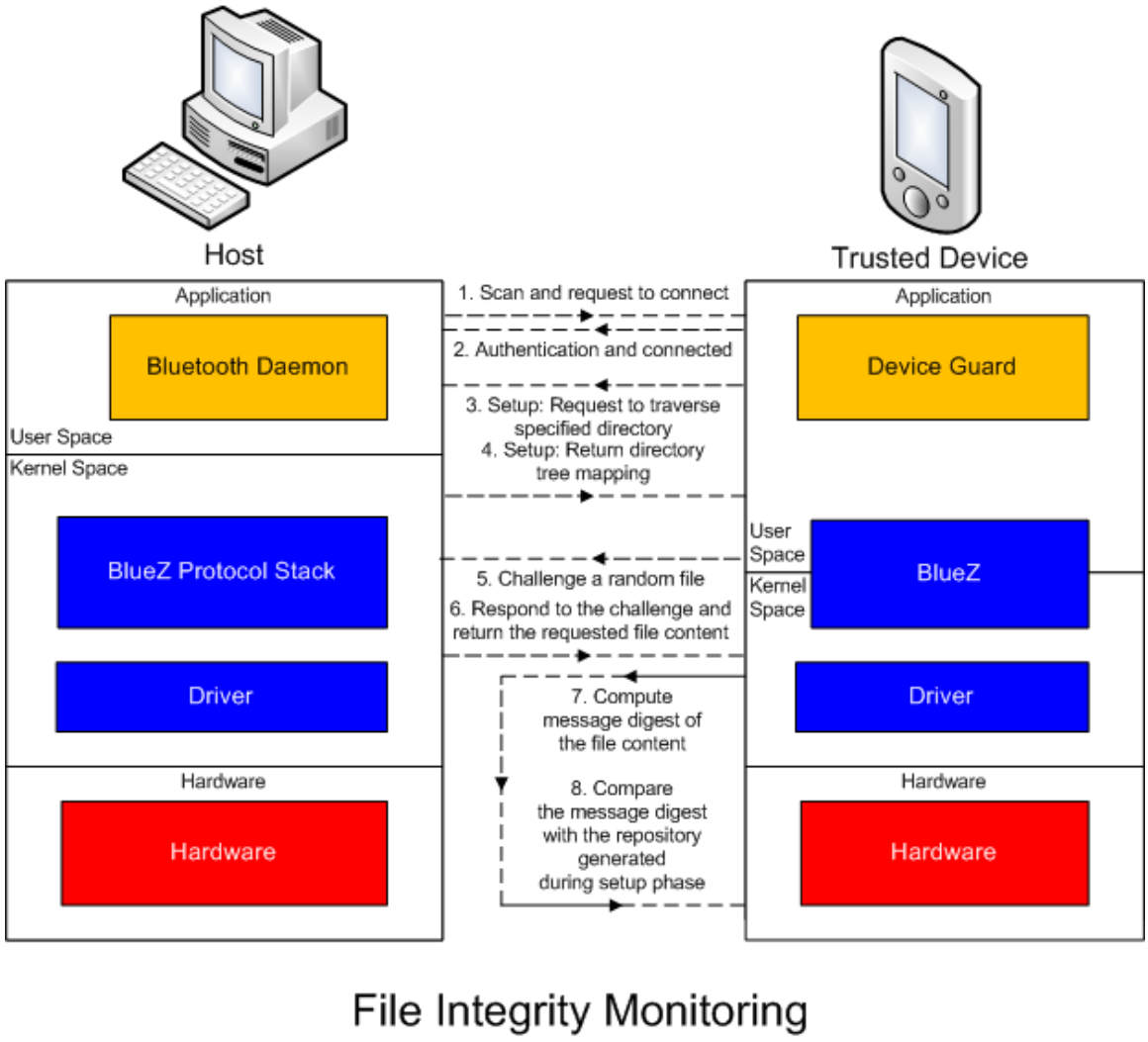

Figure 4-2 File Integrity Monitoring Architecture and Protocol

On the host side, a DeviceGuard daemon is deployed to listen to command of the Trusted Device, execute the command as requested. Similarly, on the Trusted Device, we also deployed a DeviceGuard application to monitor the Host via the DeviceGuard daemon on the Host.

Both the Host and the Trusted Device are using BlueZ, the Linux official Bluetooth protocol stack[21] as the Bluetooth protocol stack to interface the Bluetooth chipset. As Figure 4-2 shows, there are some slightly differences in terms of Bluetooth components in the architecture. On the Host, all BlueZ related components are running in kernel mode (Since Linux kernel 2.4.6, BlueZ is native in Linux kernel), while on the Trusted Device, with Android 2.1 as

the operating system, the RFCOMM and SDP components are brought up to the user space in an Application Framework layer. Although there exist the above mentioned differences, the BlueZ core remain the same, it can still provide the same base for the Bluetooth communication.

After the Host and the Trusted Device are paired and connected, a two phase procedure is designed as blow:

1) Setup Phase:

a. The Trusted Device sends a TRAVERSE command to the DeviceGuard daemon on the Host indicating under which directory the system and data the user are interested to get it protected.

b. The Host will traverse the specified directory after receiving the TRAVERSE command and return the tree mapping results to the Trusted Device in a TREEMAPPING message. The tree mapping results include the total number of files that have been traversed, what each file's location is, the size of each file and their contents.

b. Upon receiving the tree mapping results on the Trusted Device, the DeviceGuard will automatically generates the message digest for each file and store it in the repository with other metadata, such as file lookup number, file name and file URL. We will discuss later in 5. 2 more details on the local repository design issues on the Trusted Device.

2) Challenge-Response Phase:

a. The DeviceGuard on the Trusted Device is able to randomly pick up a file in the repository or simply follows the user inputs. Then it will compose the message and send a CHALLENGE command to the Host to challenge the file integrity.

b. After receiving the CHALLENGE command on the Host, the DeviceGuard daemon will try to find the requested file in the Host followed by preparing the file content and sending a RESPONSE message back to the Trusted Device. The RESPONSE message will include the exact content of the file which is challenged. We can actually performed the message digest generation in the DeviceGuard daemon, but given that we are concerned about the security on the Host and don't want to store the repository on the Host, we are deferring the message digest generation in the Trusted Device.

c. When the Trusted Device receives the challenge RESPONSE, a temporary message digest will be generated and compared to the one which is previously generated and stored in the repository during the setup phase. The comparison result will tell whether the challenged file is compromised or not.

4. 3 CASE 2: DIGITAL SIGNING

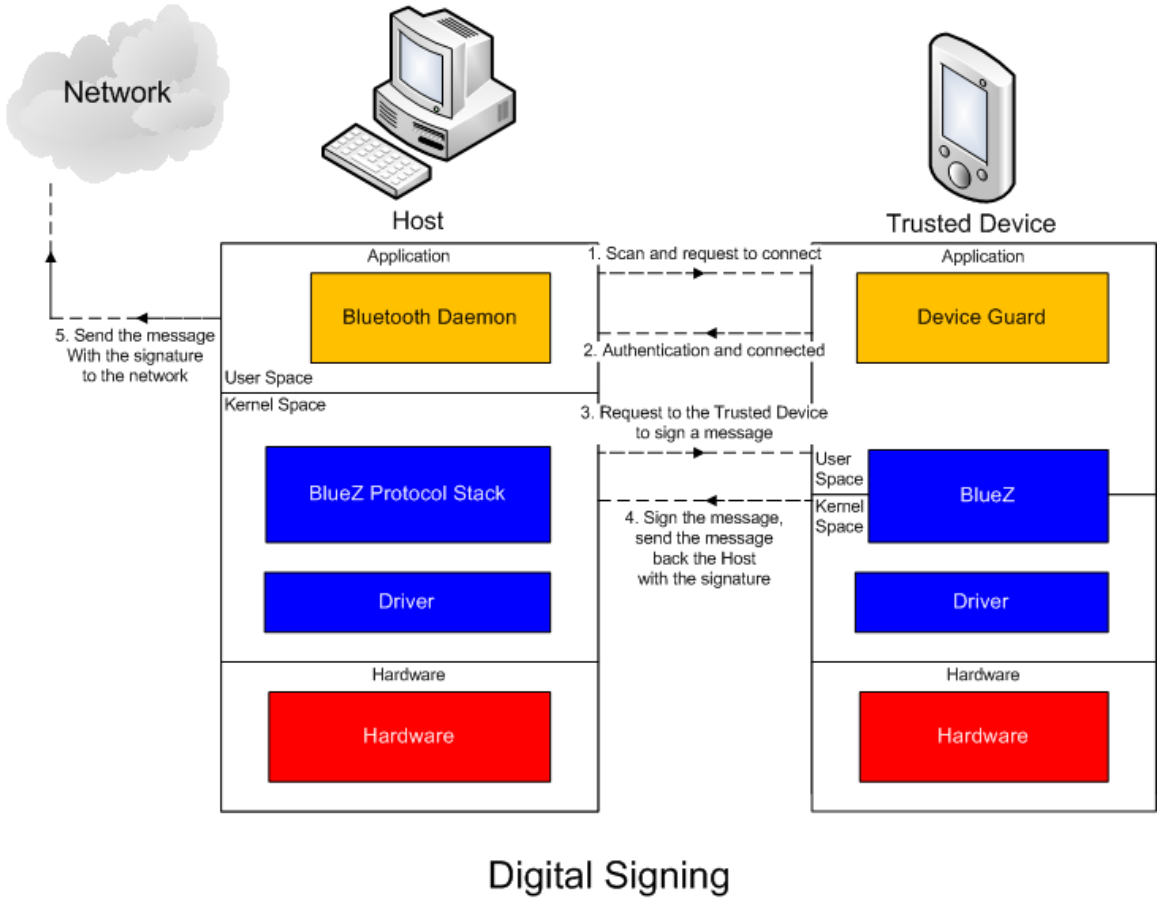

Figure 4-3 Digital Signing Architecture and Protocol

Digital Signing application demonstrates a case to use the Trusted Device to keep the secret key in public key cryptography. Since the Host is vulnerable to the attacks from the malware thus it is not trusted, this make the private key on the host also vulnerable to compromises. By storing the private key on the Trusted Device, we can make sure there is no similar compromise that will affect the secrecy of the private key. Whenever the Host needs to send a singed message to a recipient in the network, we can simply forward the message to the

Trusted Device for a signature before propagate to the internet. Figure 4-3 shows the application architecture and protocols after the basic pairing and connection establishment is done.

a. The DeviceGuard daemon on the host prepares the message and sends to the Trusted Device. It expects the DeviceGuard solution on the Trusted Device to sign the message with the private key.

b. When the Trusted Device receives the SIGN message from the Host, it will understand this is a request to using the signing service on the Trusted Device. The DeviceGuard will then sign the message with the private key it hold at hand to generate a digital signature and respond the SIGNATURE message back to the Host.

c. The Host receives the signature as a proof that the message send to the network is signed. The message together with the signature will be then send to the recipient in the network.

d. The recipient in the network receives the message, with the publicly available public key of the Host to verify the authenticity of the message received. It will be able to easily verify whether the message is actually send from the Host or from a spoof.

4. 4 CASE 3: SECURE DECRYPTION

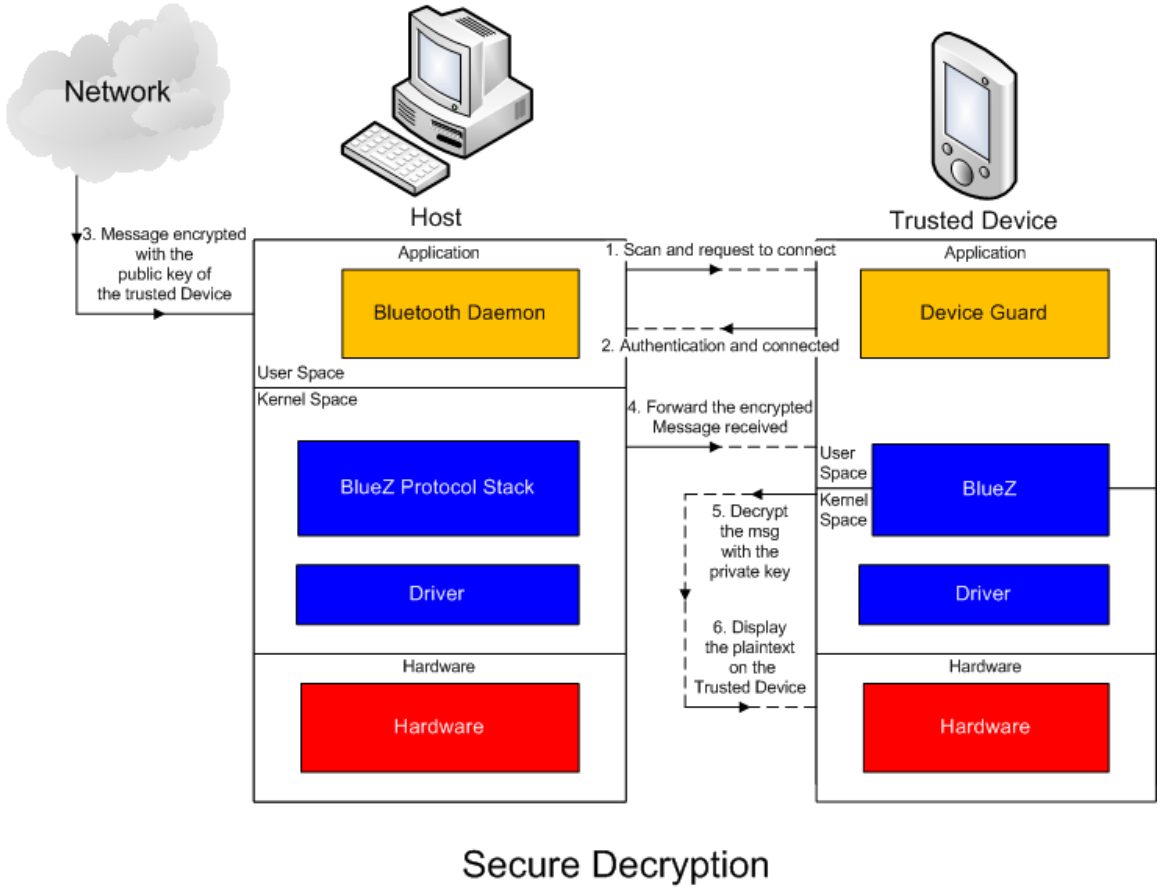

Figure 4-4 Secure Decryption Architecture and Protocol

Secure Decryption is yet another case to demonstrate the public key cryptography using an external device to protect the private key of the host. To protect the message received in the un-trusted Host, the private key is instead stored in the Trusted Device for decrypting the message which is encrypted with the public key in the key pair. By saying secure decryption, firstly, the private key is secure, it will not suffer from compromises on the Host; secondly, the encrypted message is confidential, it is difficult for the malware on the host to know the message content since the message is decrypted and displayed on the Trusted Device. Figure 4-4 shows the basic application architecture and protocols.

a. A sender in the network sends a confidential message to the Host. To ensure the confidentiality of the message, the message is encrypted with the public key of the Host before

sending it to the Host. Then only the party who possess the private key in the pair will be able to decrypt and know the exact message content.

b. After receiving the encrypted message on the Host, the Host knows it doesn't have the private key to decrypt this cipher text. Then it forwards the encrypted message to the Trusted Device for accessing the secure decryption service. This is the only way to decrypt the encrypted message.

c. The Trusted Device decrypts the message with the private key it keeps to recover the plaintext in the message and display it on the Trusted Device securely. The isolation provided in this case demonstrates the feasibility and prospect in hosting the secure decryption service.

CHAPTER 5 IMPLEMENTATION

In this chapter, we will elaborate the implementation details in this project. Section 5.1 gives a brief overview of the common issue that is related to the implementation both for DeviceGuard solution in the Trusted Device and DeviceGuard daemon in the Host. Section 5.2 introduces the implementation related consideration and details for the DeviceGuard application in the Trusted Device. Section 5.3 covers the implementation details of the DeviceGuard daemon in the Host.

5. 1 OVERVIEW OF IMPLEMENTATION ISSUES

In CHAPTER 4, we have already discussed the overall architectures of the DeviceGuard framework and the communication protocols under different cases. There still left some other considerations and design when move forward to the implementation of the DeviceGuard solution.

One of the considerations is to define the Trusted Device and choose a publicly available device in our experiment to demonstrate our solution. Smartphone caught our attention in our selection process as it is widely used by users of the personal computer. As a matter of fact, a Trusted Device does not necessarily to be a Smartphone, which means any device that have the following characteristics will meet our requirements and fall into the definition of the Trusted Device.

- The device should be physically held by user.
- The device should possess certain computation capability.
- The device should possess certain memory and disk space.
- The device should possess certain communication channel to other devices

Among various choices of a Smartphone, Android Development Phone 2 (ADP2) and a Linux Damp Box stands out and are selected to be two different types of Trusted Devices in this project. They all meet the minimum requirements of the definition of the Trusted Device and are available to be purchased at the early stage of the project.

The next consideration is to find the communication interface which is available and suitable for our DeviceGuard solution to communicate between the host and the trusted device.

The communication interface had better to be widely supported by the personal computer to reduce additional cost that may arise.

USB was considered in the first place. We were imagining that if an external device can be simply plugged into the personal computer to perform the malware scanning or system and data integrity checking, then it is really cool. Unfortunately, after an extensive research on the USB practices and the specifications, it turned out that when the USB is connected to the personal computer, the USB controller in the Host will only allow communication initiated from the Host, but not bi-directional. This drives up to find alternative communication interfaces.

After a careful evaluation, we found Bluetooth exhibits a great potential in fulfilling our required communication patterns. Bluetooth supports bi-directional communication between two devices; Bluetooth supports SDP, Service Discoverable Protocol, which allows each service to be bind to a UUID; Bluetooth also supports encryption to the communication based on the passkey agreed by the communicating devices, which ensures the confidentiality of the communication. Besides, Bluetooth is publically available in almost all the devices including Smartphone, even for the devices without embedded Bluetooth chip, USB Bluetooth dongle is available at low cost.

The third consideration is how to design the internal communication protocol and message data structure for our DeviceGuard solution. Based on different security features we supported in DeviceGuard, the internal communication protocol will differ a little bit. But overall we will have an interactive communication protocol. The message data structure is tightly couples with the communication protocol since we will need to define a header in the Bluetooth message payload to distinguish different commands before the DeviceGuard can perform designated operations.

The generic DeviceGuard message data structure is defined in Figure 5-1. The first byte in the message indicates the message type, follows by four bytes of the message size field and the remaining bytes as the message payload.

DeviceGuard Message Data Type Definition

Figure 5-1 DeviceGuard Message Data Type Definition

5. 2 DEVICEGUARD IMPLEMENTATION

5.2.1 Application Structure

The application is designed to follow the MVC design pattern.

1) View

The DeviceGuard provides a graphic user interface to assist user using the security features. To comply with MVC design pattern, we separated and implemented a layer to take care of the user actions.

2) Control

In the Control layer, we implemented the Bluetooth communication procedures and the security features that we provided in DeviceGuard solution. It provides interfaces for the View layer to push or pull data and listens to events that will trigger user interface updating. It also makes no assumption of how the underlying repository is implemented.

3) Model

We designed a local repository in the Trusted Device where it can store necessary metadata to assist the security features that are provided in the DeviceGuard solution.

Take the case in system and data integrity monitoring for example, since we will provide the integrity checking for system files and user data on the Host, we will need a mechanism and a local repository to aid in determine whether the system files and user data is compromised.

As we discussed earlier, we will have a setup phase to traverse the user interested information on the Host and then build the repository on the Trusted Device. Since the local storage is limited in the Trusted Device, we have to carefully design the data structure that will be used in the repository to conserve the precious storage resource while still maintaining premium performance in manipulating the repository.

The first version of the repository is designed as a two-dimensional array to facilitate an efficient random checking. One of the shortcomings is that the two-dimensional array will not be able to expand at low cost. We then investigated the possibility of using a Hash map in the repository, it will allow dynamic resizing of the repository while offers constant time (O(1)) lookup performance. But the problem in this data structure is that it cannot support randomization of the entry for a random challenge to the system and data integrity. Due to this reason, we decided to keep the original design of the repository for the moment.

5.2.2 Key Functions

1) Service Registration

Service registration involves assigning a UUID to the service interface, so that only one device who knows about this UUID will be able to connect to this device.

2) Device discovery and pairing

Bluetooth requires the communicating device be able to discover the remote device before they can pair with each other and setup the connection.

3) Case based security feature

a. Local Repository construction and manipulation

As we mentioned in the previous section, we have decided to design the local repository using the simple two-dimensional array. During the setup phase, based on the traverse results, the DeviceGuard will learn the total entries of the system file and data to be monitored; with this information, the repository is initialized.

The next step is to populate the repository with subsequent data send from the DeviceGuard daemon, including the path to the system file and data, contents of the system file and data. Since we are trying to reduce the repository size but can still maintain enough information for identifying a compromise in a later time, we decided to store the message digest of the system file and data, instead of the content. The storage efficiency will be examined in our evaluation section.

b. Digital Signing Service

Since we stored the private key in the Trusted Device, the digital signing services is able to accept the message from the Host to be signed. This interface takes the message as the input and signature as the output which will send back to the daemon on the Host.

c. Secure Decryption Service

Similarly, with the private key stored in the Trusted Device, all the encrypted message send from the Host will be decrypted with the private key and directly displayed on the screen of the Trusted Device. This interface includes accepting the cipher text, performing decryption, and pushing the plain text to the GUI.

4) Bi-directional Bluetooth communication.

The bi-directional Bluetooth communication is setup by directly manipulate the RFCOMM communication channel, a serial port emulation protocol. Different from using other special purpose profiles, RFCOMM enables a simple reliable data stream to the user. The throughput of RFCOMM communication between the two devices is measured to evaluate the feasibility to use Bluetooth as the communication interface in the DeviceGuard solution. It will be further discussed in 6.2.1.

5. 3 DEVICEGUARD DAEMON IMPLEMENTATION

5.3.1 Application Structure

The DeviceGuard Daemon is running as a background daemon in the Host, where it will listen to the Bluetooth communication channel for allowed connection from other device. The daemon is designed to only contain limited logic which simply serves as a relay for the DeviceGuard on the Trusted Device.

5.3.2 Key Functions

1) Service Registration.

   Same as the service registration for the DeviceGuard, in the DeviceGuard also involves assigning a UUID to the service interface, so that only one device who knows about this UUID will be able to connect to this device.

2) Case based functionality

   a. Directory Traversal.

      As part of the case in system files and data integrity monitoring, to prepare the data for populating the repository in the DeviceGuard on the Trusted Device, we need to traverse all the system files and user data with which user are interested to put under surveillance.

   b. Obtaining Digital Signature for message

      This interface will forward the message generated on the Host to the Trusted Device for obtaining a signature with the private key.

   c. Forwarding Encrypted Message for Decryption

      The encrypted message received from another party in the Internet will be directly forwarded to the Trusted Device for decrypting and displaying on the device.

3) Bluetooth Message Parsing

   In supporting the DeviceGuard communication protocol, we wrap the internal message data structure in the payload of the Bluetooth message. Handling the

message is the key to maintain a reliable interaction between DeviceGuard daemon on the Host and the DeviceGuard on the Trusted Device.

4) Bluetooth Message Construction and Response

The message data structure consists of a header with both the message type and message size inside, followed by the actual payload for DeviceGuard.

5) Daemon message loop

Daemon message loop is necessary since if only the daemon stays will it be able to listen to the incoming command from the DeviceGuard. As such the DeviceGuard solution will be able to function as we expected.

CHAPTER 6 EVALUATION

In this section, we introduce the experiment setup of our DeviceGuard solution, both on the host and the trusted device, followed by the evaluation of the results obtained on the Trusted Device. The experiments we are going to present in this chapter including Bluetooth throughput, message digest performance, signing performance, decryption performance and footprint of the solution. The evaluation goals include:

1) To examine the feasibility and efficiency of using Bluetooth to facilitate the communication between the Trusted Device and the Host;
2) To examine the feasibility and efficiency for constructing a repository for file integrity checking in the Trusted Device;
3) To examine the feasibility and efficiency of providing signing service in the Trusted Device;
4) To examine the feasibility and efficiency of performing the secure decryption in the Trusted Device.

6. 1 EXPERIMENT SETUP

6.1.1 Host

In our experiment, we use a Lenovo T410 laptop as the Host as below in Figure 6-1. The specification includes the 2.40 GHz dual core Intel Core i5 processors, 2 Giga bytes memory and a Bluetooth chipset compatible with Bluetooth version 2.1 specification.

Ubuntu 10.04 LTS is installed as the host operating system. We use GCC version 4.4.3 as the default compiler to compiler our DeviceGuard Daemon on the host. Besides, in Bluetooth protocol stack selection, we choose BlueZ version 4.66, the Linux official Bluetooth protocol stack, in our experiment.

Figure 6-1 Host: Lenovo T410 Laptop with ADP2

6.1.2 Trusted Device

Figure 6-2 Trusted Device: Android Dev Phone 2

Android Dev Phone 2 (ADP2) is been selected as the Trusted Device in our experiment since it meets our criteria of our definition for the Trusted Device, which possess certain computing capability and memory space besides the rich communication interfaces. ADP2 is a Smartphone device from Google which provides to the application developers for Android

Smartphone. Inside ADP2, we have a processor running up to 576 MHz, 512 Mega bytes Read-Only memory, 288 Mega bytes random access memory, and an additional 1 Giga bytes MicroSD storage. Besides, the Bluetooth chipset on ADP2 is compatible with Bluetooth version 2.0 specification.

The native Android operating system in ADP2 is on version 1.6. In order to use the Bluetooth API that appears in later version of the Android operating system, we upgraded it to version 2.1. Android provides a developing platform based on Java virtual machine, more specifically, it supports Java SDK version 1.6. In our experiments, we used Java Cryptographic libraries to provide security features, such as java.security.* and java.crypto.*. The Eclipse Compiler for JAVA (ECJ), an open source incremental compiler used by the Eclipse JDT, based on IBM VisualAge Java compiler, is used to compile our DeviceGuard application running on the Trusted Device. Besides, BlueZ version 4.47 is the Bluetooth Protocol Stack on Android as we mentioned earlier.

In the rest of this chapter, we evaluate our solution from different perspectives, the results shows in the figures are average of 5 runs consistently.

6. 2 EXPERIMENT RESULTS AND EVALUATION

The experiments covers Bluetooth Throughput, examines the feasibility and efficiency of communicating over Bluetooth for repository setup and challenge response; the required sample size verses cheating effort to figure out how difficult it would be for the malware to fool the DeviceGuard in the device; Message Digest performance on the device, which will show the efficiency for performing bulk of file message digesting on the device where computation resources are limited; Digital Signing performance, the efficiency in the case of signing feature is offloaded to the device, both DSA and RSA with different signature size are examined; Secure Decryption with AES and RSA under different key sizes is also examined to prove the efficiency and feasibility of using the device to fulfill the security feature; last but not least, the storage usage improvement is also discussed in this section.

6.2.1 Bluetooth Throughput

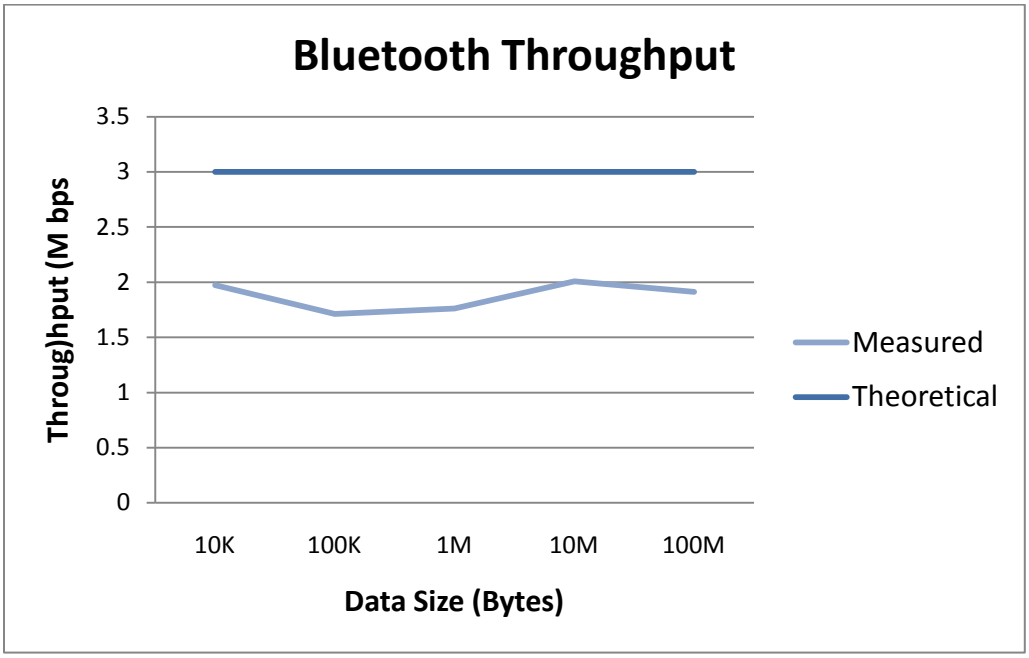

Figure 6-3 Bluetooth Throughput

Figure 6-3 shows the Bluetooth throughput in our solution. Theoretically, the Bluetooth throughput is designed to guarantee a 3M bps throughput. Our experiment shows the measured throughput can only reach about 2M bps. The overhead includes encryption in the Bluetooth channel, non-optimum sending and receiving buffer configuration during communication. With 2M bps throughput, we will be able to transfer 1G bytes in about an hour during the setup phase in the File System Integrity Checking Case. For other use cases, since it doesn't prohibit a data intensive pattern, 2M bps throughput is sufficient enough in our DeviceGuard solution.

6.2.2 Required Sample Size Vs. Cheating Effort

Since the challenge that initiates from the Trusted Device could be randomly generated, the malware in the host have no idea which file is going to be challenged. In the adverse condition where the malware is able to intercept the challenge to the DeviceGuard daemon and fake the response message to the device. To fool the DeviceGuard on the device, the malware must response with the exact clean copy of the file been challenged. As the research presented in the un-cheatable Grid Computing[27], the cost for cheating the random challenge is relatively high.

6.2.3 Message Digest Performance

Table 6-1 Digest Performance (ms)

	Data Size (bytes)				
Algorithms	1K	10K	100K	1M	10M
MD5	4.5	2.25	14.75	53.75	499.75
SHA1	5.75	47	252.25	2357.25	22861.25
SHA-224	5.5	8	12.5	80.75	756.75
SHA-256	9.25	8.5	12.5	80.5	756.75
SHA-384	16.25	132	576.75	5681	51714
SHA-512	22	74.25	524	5217	51246.5

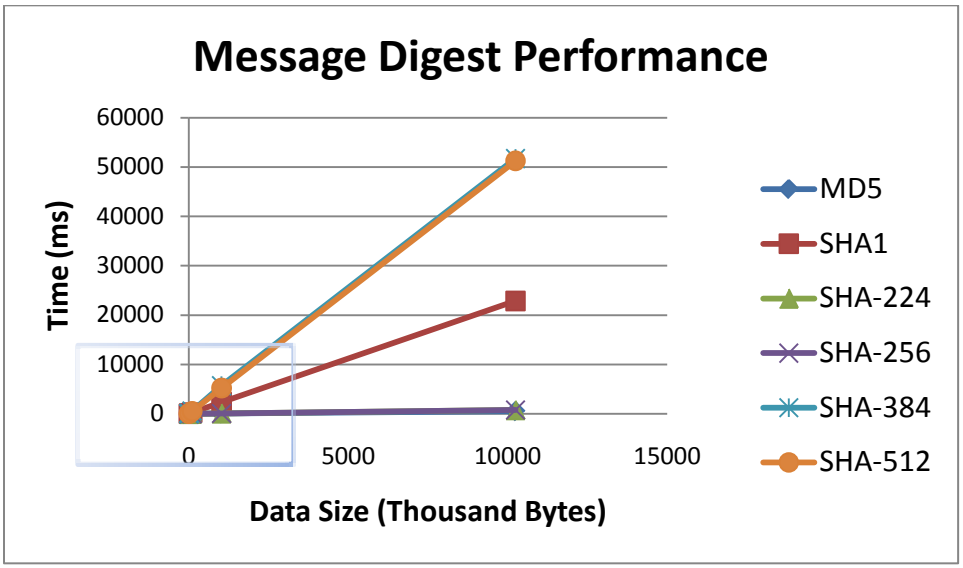

Figure 6-4 Message Digest Performance

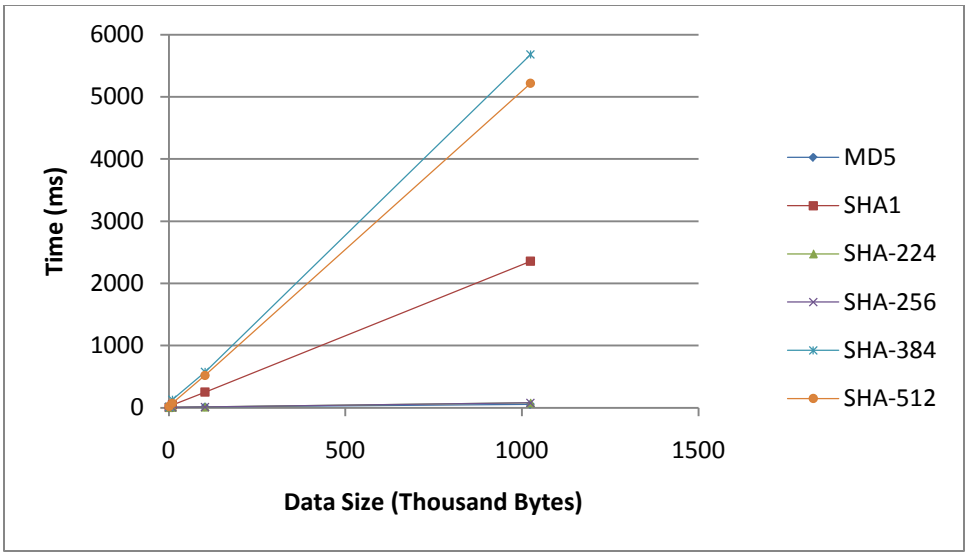

Figure 6-5 Enlarged Rectangular Area in the Above Figure

We measured the message digest performance with the Java Cryptographic library on Android 2.1. The results in Figure 6-4 shows MD5, SHA-224 and SHA-256 are the most stable algorithm exhibits almost linear curve when the data size increases. Table 6-1 shows the exact data points in companion with Figure 6-4 for better understanding the results. Interestingly, SHA-224 and SHA-256 use a longer digest size than MD5, but performs almost the same as MD5. Since SHA1 uses 160 bits digest size, slightly larger than 128 bits used in MD5, it performs worth than MD5. For SHA-384 and SHA-512, use longer digest size, as we can expect it performs the worst but given the longer digest size, it is more secure than all the other algorithms shown in the figure. Based on the above observation, we will be able to evaluate the requirement and choose the algorithm that can provide the best tradeoff in algorithm complexity and performance.

6.2.4 Digital Signing Performance

Table 6-2 Digital Signing Performance (ms)

	Data Size (bytes)						
Algorithm/ Signature Size	1K	5K	10K	50K	100K	500K	1M
DSA-512 bit	10.75	19.25	29.5	120.25	232.5	1138.75	2339.75
DSA-1024 bit	16.75	25.5	37.5	126.25	240.5	1143.75	2282

RSA-256 bit	8.333333	22.66667	37.33333	170.6667	353.6667	1717	3225.667
RSA-512 bit	11	23.33333	40.33333	180.3333	351.3333	1753	3210.667
RSA-1024 bit	25.66667	39.66667	53.66667	197.6667	364	1631	3249.667

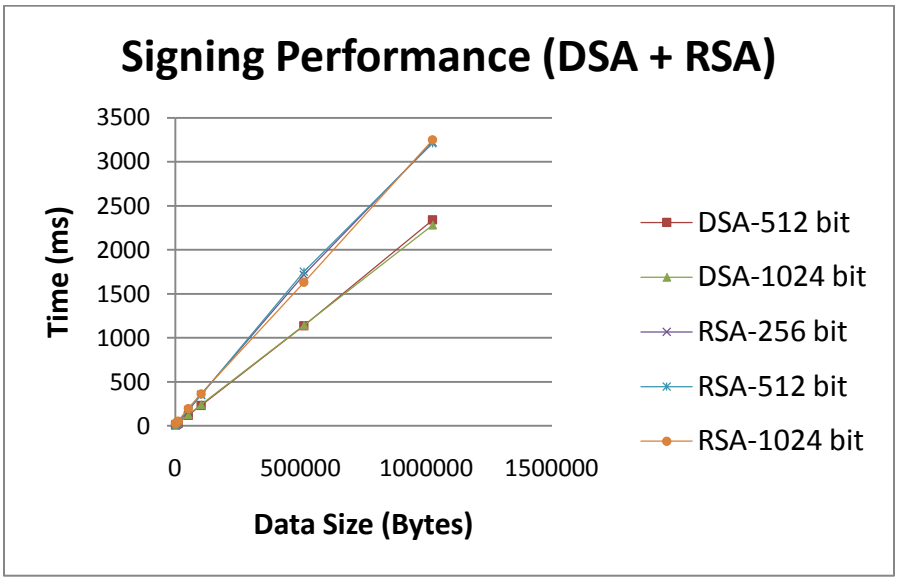

Figure 6-6 Signing Performance (DSA + RSA)

Table 6-2 and Figure 6-6 shows the signing performance by using DSA and RSA respectively with different key sizes (256 bits, 512 bits, 1024 bits). DSA out performed RSA no matter what key sized been used, and DSA with 256 bits key size performs the best.

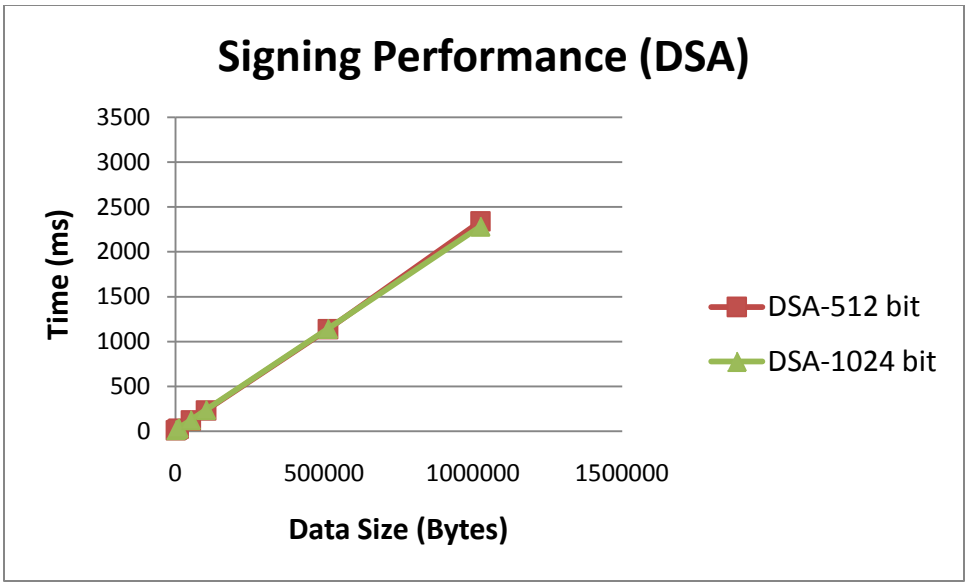

Figure 6-7 Signing Performance (DSA)

Figure 6-7 compares the performance of choosing DSA in singing the message. The performance of DSA with 512 bits and 1024 bits performs the almost the same.

Figure 6-8 Signing Performance (RSA)

Figure 6-8 shows that no matter what key size we choose, either 256 bits, or 512 bit, or 1024 bits, the RSA performs almost the same.

6.2.5 Decryption Performance

Table 6-3 Decryption Performance with AES (ms)

	1K			
AES	1K	10K	100K	1M
128 bits	6	46	480	5232
192 bits	6	53	552	6767
256 bits	7	58	742	6900

Table 6-4 Decryption Performance with RSA (ms)

	Data Size (bytes)			
RSA	1K	10K	100K	1M
512 bits	96	960	9600	96000
1024 bits	304	3040	30400	304000

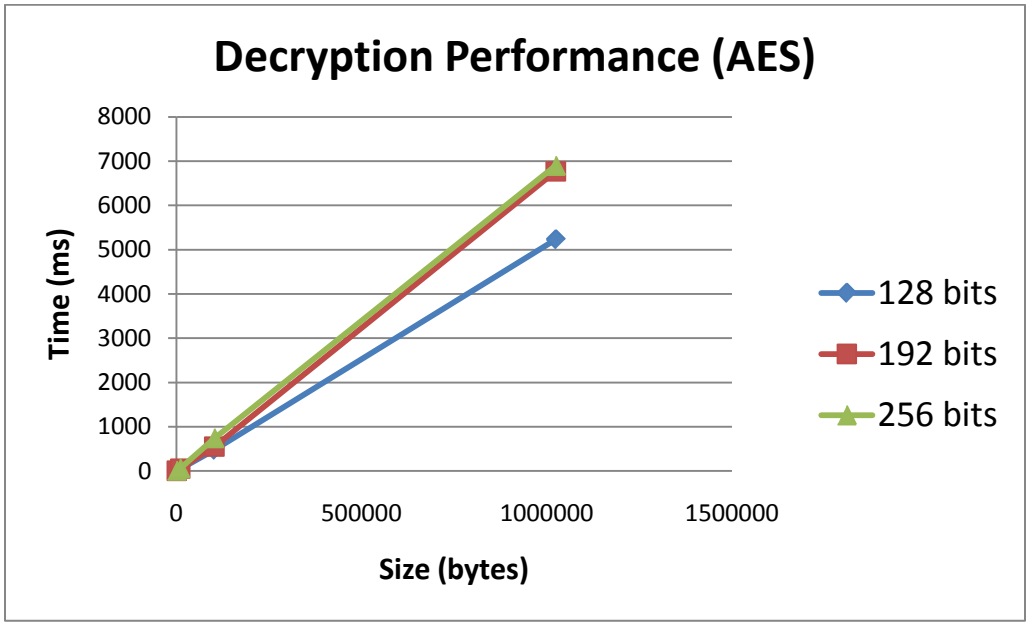

Figure 6-9 Decryption Performance on Device with AES

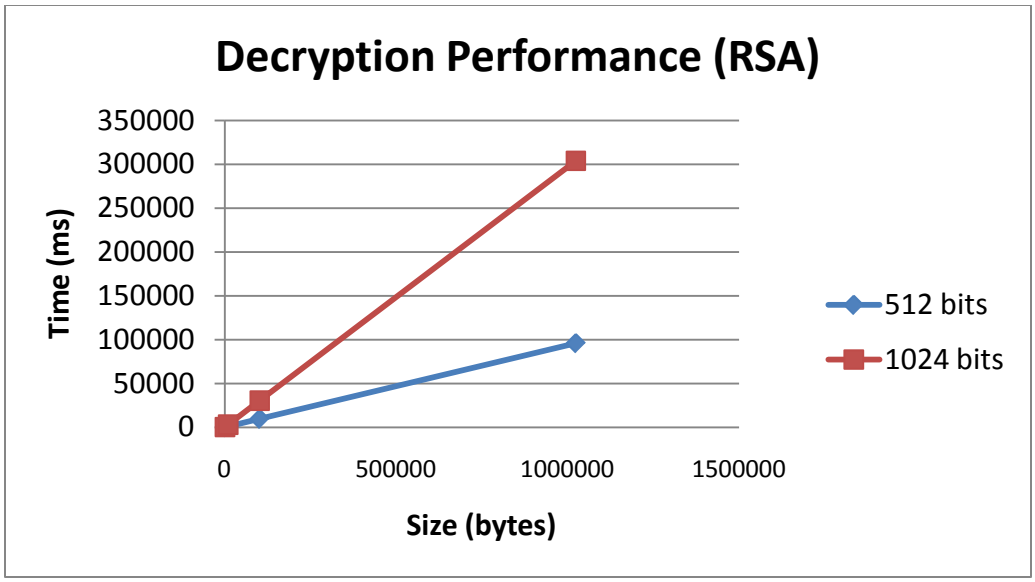

Figure 6-10 Decryption Performance on Device with RSA

In the public key encryption, the decryption performance in Figure 6-9 and Figure 6-10 exhibits linear characteristics. The complexity of the algorithm increases with the increase of key size from 512 bits to 1024 bits. Table 6-3 and Table 6-4 show the exact data points that represented in Figure 6-9 and Figure 6-10.

In general, the performance of Symmetric key encryption performs better than public key encryption in decrypting the cipher-text due the innate complexity in the algorithms. The complexity of AES increases with the increase of key size from 128 bits, to 256 bits.

Since the supported key size in RSA and AES are different, we cannot provide side by side performance comparison in this section.

6.2.6 Disk Space Usage on the Trusted Device

On the Trusted Device, we only need to store the message digest of each file that we are monitoring, which is 160 bits per entry. For each entry, we also need to store the file path which indicates the file location on the host, 256 bytes maximum. To sum up, the total size for each entry will require 276 bytes maximum.

6.2.7 Storage Usage Improvement

By using the message digest in the Trusted Device for challenge-response, we don't need to store the whole file system and user data in the device. Given the constraint in the device, this makes the solution feasible in practice.

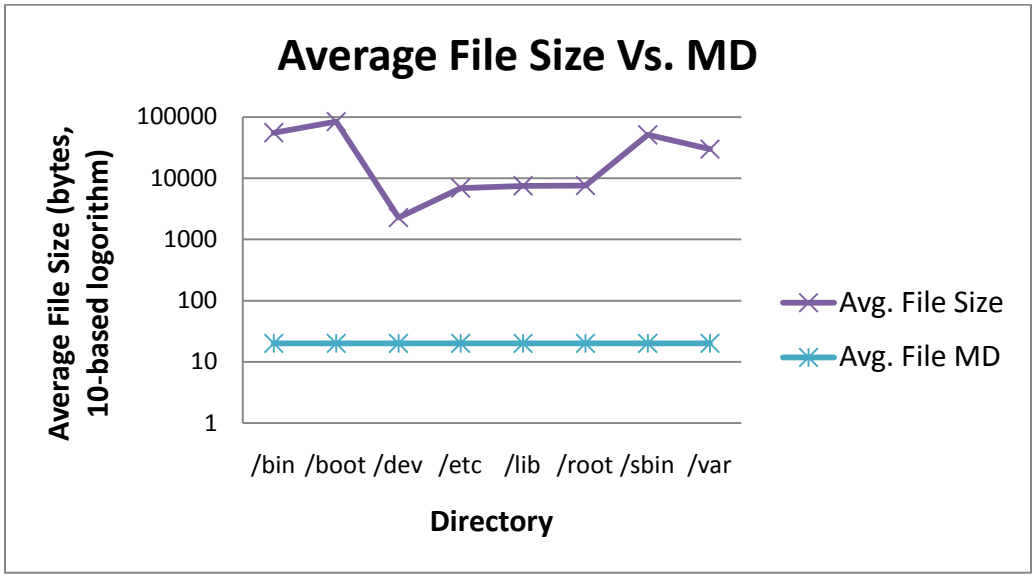

Figure 6-11 Average File Size Vs. Message Digest Size

Figure 6-11 shows the average file size and message digest size in Linux system directories. The average size for the message digest is constant.

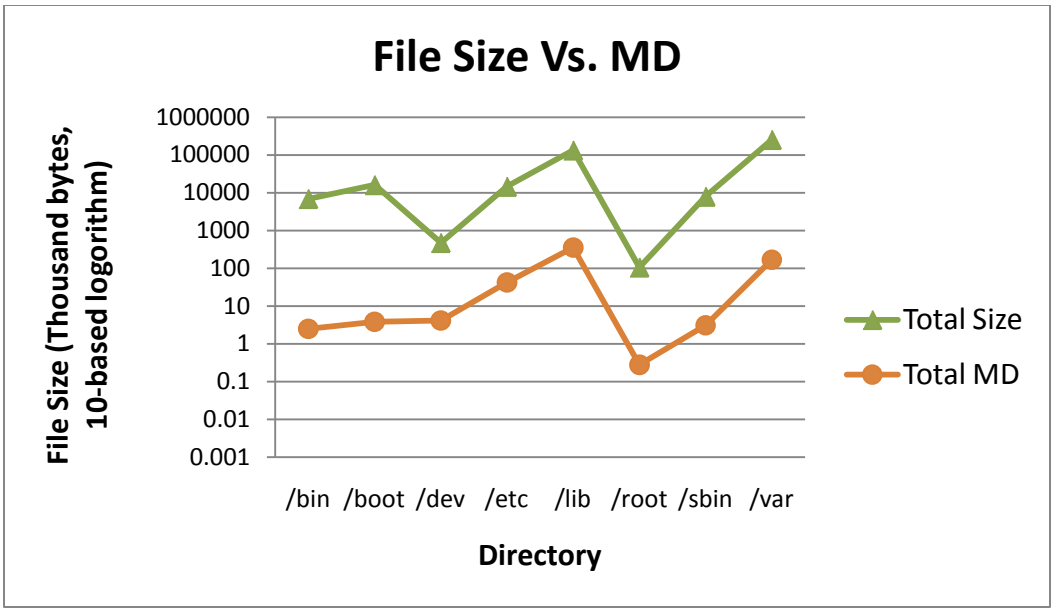

Figure 6-12 Total Size of Files Vs. Total Message Digest Size

Figure 6-12 shows the benefit in disk space of storing message digest on the trusted device comparing to storing the contents of all the files. Our message digest based repository only needs to use as little as 10^{-3} scale of the original data size. Given the limited storage on the Trusted Device, the results prove the feasibility of constructing a local repository on the device.

6.2.8 Experiment summary

The experiment results as shown in the previous subsections examine the feasibility of our DeviceGuard solution from several different perspectives, including:

1) The feasibility and efficiency of using the Bluetooth to facilitate the communication between the Trusted Device and the Host. With a 2Mega bytes per second practical throughput as we measured in the experiment, it is fairly enough for the short range data exchange.

2) The cheating effort required for the malware to cheat the DeviceGuard. In order to cheat the DeviceGuard, the malware will need to prepare fake data where the cost is relatively high both in term of the space and compromising effort. This makes the malware difficult to execute the compromise while still maintain it undercover.

3) The efficiency for constructing a repository for file integrity checking in the Trusted Device. In terms of the space, for the message digest typically with SHA-1 or MD5,

only 160bits is needed for each entry in the repository; while in terms of the time, our results show the message digest can be generated in the time scale of 10^1 for each kilo bytes of data.

4) The efficiency of providing signing service in the Trusted Device. Digital signing with 1024 bits signature on the Trusted Device takes only the scale of 10^1 and 10^3 ms for 1K bytes and 1M bytes respectively.

5) The efficiency of performing the secure decryption in the Trusted Device. As show in the results, secure decryption with AES 256 bits on the Trusted Device takes only the scale of 10^1 and 10^3 ms for 1K bytes and 1M bytes respectively, and secure decryption with RSA 1024 bits takes the scale of 10^2 and 10^5 ms for 1K bytes and 1M bytes respectively.

Our DeviceGuard solution takes advantage of the proliferation of devices, such as Smartphone, with minimum or no extra cost to assist the host for stronger system and data security.

CHAPTER 7 DISCUSSION AND REMIDIES

In the previous chapters we have already gradually elaborate the problem that we addressed in this project and how we tackle it by offloading security features to an external device to enhance the security on the Host. The experiment result demonstrates the effectiveness of our approach in providing security enhancement on the Host. But on the other hand, the approach that we proposed is not perfect. We have noticed some of the topics that can be further enhanced in the future work and would like to have a discussion in this chapter and provide a preliminary research summary.

7. 1 DISCUSSION ON CURRENT APPROACH

The DeviceGuard architecture separates the security solution from the machine enables the Trusted Device to assist the system and data security on the Host. Although we have stated some assumptions for this research, in the real world environment those assumptions might still need to be addressed. It can be treated as the pitfalls for a robust solution which could be the target of the attack and thus possess potential breach in practices. In this subsection, we are going to discuss the pitfalls in our solution and what are the potential remedies, etc.

The Kernel is Compromised.

Under the current architecture of DeviceGuard, the DeviceGuard daemon running on the Host to respond to challenges from the Trusted Device relying on the reliable functioning of kernel, especially the file system in the kernel space. Although we have already taken into account the kernel security in our assumptions, we still need to be aware of kernel breach in practice. “Kernel malware is malicious software that runs fully or partially at the most privileged execution level, ring 0, having full access to memory, all CPU instructions, and all hardware.”[9]

One way to offset the kernel compromise is to taking advantage of the recent advance in virtual machine technologies. Hypervisor or Virtual Machine Monitor runs below the guest OS provide another abstraction layer which also can isolate and limited the privilege of the guest OS, provides a great potential to host the DeviceGuard daemon which are able to provide a solid runtime environment for the DeviceGuard solution. We will discuss in 7.3 the Hypervisor-based DeviceGuard in detail.

DeviceGuard Daemon is Compromised

One of our assumptions in the DeviceGuard solution is that DeviceGuard daemon in the Host is not compromised. In other words, it should not be either replaced by a fake DeviceGuard daemon or simply been disabled by malware.

This assumption is reasonable in the sense that we are more focused on the solution to demonstrate the feasibility of our idea of the DeviceGuard. It is the weak point that could be challenged. But given the kernel is not compromised, running the daemon in the privilege mode will prevent normal privilege malware compromising the daemon.

On the other hand, in the case of the DeviceGuard daemon been disabled by the malware, the problem would be easily detected when user performed a regular checking of system integrity. Thus user can deduce there might be something wrong with the system, and a more sophisticated inspection of the system would be triggered.

DeviceGuard Daemon Failed to Identify the Trusted Device.

In CHAPTER 4, we mentioned that Host and Trusted Device are securely connected in part by identifying the Bluetooth service id, where we use the Universal Unique identifier (UUID), before they get connected with passkey agreed by them. There is possibility that either the UUID on the Host side or on the Trusted Device side is been unintentionally or maliciously modified, then there is a mismatch in identify each other for connection.

Since the Trusted Device is owned by the end user and not connected to the internet, we could assume it is secure in this setting. In considering the UUID been maliciously modified by malware on the Host, this can be actually categorized in the threat of the DeviceGuard daemon is been compromised in 0. Thus, we will not expand the discussion here at this point.

Bluetooth Communication is Intercepted.

Bluetooth is chosen to provide the bi-directional communication channels in our current solution. For almost all the communication, there is potential threats stem from the interception, Bluetooth is not an exception. Communication interception is usually in the form of Man-In-The-Middle (MITM) attack. In Bluetooth, the MITM attack is to happen for an attacking device stays between the Host and the Trusted Device. The attacking device will hide its existence to make the Host and Trusted Device believe they are simply just talking to each other, but in reality they are all communicating with the attacking device for relaying the packets.

Bithidproxy [28] demonstrated the Bluetooth MITM attack by using the Human Interface Device(HID) attack to setup the connection between two devices, a Host and a HID device. In this attack, the attacker will wait until the Host to power down, then it will simply clone the Host's Bluetooth device information to connect to the HID device. On the other hand, the attacker will then clone the Bluetooth device information of the HID device in another interface, thus the Host will later reconnect to this fake HID interface after it is restarted. After this attack, all the communication traffic between the Host and the HID device will be monitored by the attacker.

In this research, since we are focusing on the malware inside the Host, not on the Bluetooth inception or Man-In-The-Middle attack from another device, thus the *Bithidproxy*-like [28] attacks will not be a problem in current solution. Besides, the attacked demonstrated in *Bithidproxy* [28] is only effective in the HID device setting, while our solution is using the RFCOMM communication channel which will be un-affected.

The Bluetooth Service is Disabled.

In our DeviceGuard solution, we are relying on the Bluetooth service to carry on bi-directional communication between the Host and the Trusted Device. If Bluetooth Service on the Host is been disabled by the malware, the Trusted Device will unable to sending command and receiving reply from the DeviceGuard daemon on the Host to perform either system integrity checking or scanning.

We could make another assumption here to state that if the Bluetooth communication is broken between the Host and the Trusted Device, we can deduce there must be something wrong on the Host (If we are sure the Bluetooth service on the Trusted Device is in good condition). From this standpoint, referring to 0 that the malware are trying to hide their existence, we could find there is contradiction in this case. Thus, we can basically free from finding remedies for this potential pitfall.

The Bluetooth Device is Turned Off From the Discoverable Mode

Bluetooth device in discoverable mode enables other devices to find it and retrieve the public services on it. If the Bluetooth device on the Host is turned off from the discoverable mode, then unless the Trusted Device already know the Host's Bluetooth physical address and paired beforehand, the Trusted Device will not be able to aware of the Host's existence.

Similar to what discussed in subsection 0, given the malware's intention is to hide its trace of existence, this problem will not be further discussed in this research.

7. 2 DEVICEGUARD WITH A LINUX BOX

As we mentioned earlier in CHAPTER 1, the Trusted Device need not necessarily be a Smartphone or any device that are natively connected to the Internet, we choose Smartphone as our test-bed for the DeviceGuard solution simply because it is available and meet our requirements for a Trusted Device. To demonstrate this idea, we are trying to expend the DeviceGuard by using the ADVANTECH Box-PC ARK-1360 to serve as the Trusted Device as shown in Figure 7-1. It has the following specifications which highly matches our criteria for the external Trusted Device:

1) ARK-1360 is a small fan-less box which would be easily carried by user.
2) ARK-1360 possessed built-in Intel® Atom™ processor Z510P 1.1 GHz processor which can easily perform the cryptographic computation.
3) ARK-1360 is equipped with 1G Bytes memory and 4G Bytes Solid-state Drive (SSD), this enables storing runtime repository and local storage of the operating system and repository.
4) ARK-1360 has rich communication interfaces, including RS-232 Serial Ports, USB 2.0 compatible USB Ports, PS/2 Port, 10/100/1000Base-T Gigabit Ethernet and 54MB 802.11b/g/n Wireless.

Figure 7-1 Trusted Device: ADVANTECH Box-PC ARK-1360

In order to communicate with the Host in Figure 6-1 using the same Bluetooth communication channel, we plugged in a BTA-3210 USB 2.0 Micro Bluetooth Dongle, which supports Bluetooth specification version 2.1 EDR with 10 meters effective range. In Figure 7-1, this tiny device is plugged in USB1 port with Blue light sparkling.

Ubuntu 10.04 LTS Server edition is installed in ARK-1360, it provides a minimum runtime environment for our experiments.

Experiment Setup

The experiment setup for the DeviceGuard with a Linux Box is not different from the setup we used in the ADP2 based solution, except for that we replace ADP2 with ADVANTECH Box-PC and plugged-in a USB Bluetooth dongle to facilitate the similar Bluetooth communication interfaces.

Idea and Approaches

On ADP2, the Android SDK only supports Java API for application development, as a result, the Bluetooth communication API is also supported by Java API. This also means the DeviceGuard solution on ADP2 is a Java application. In order to smoothly transplant the

DeviceGuard solution from ADP2 to ADVANTECH Linux Box, we will need to either find a similar Java API for Bluetooth communication or rewrite the Bluetooth communication part of the DeviceGuard in C++ by directly interfacing the BlueZ protocol stack. After some research, we identified that BlueCove [23], a JSR-82 implementation of Java Library for Bluetooth, is available to be used for providing similar Java API for accessing the underlying Bluetooth protocol stack BlueZ.

Technical Obstacles

In transplanting the DeviceGuard solution from the Android Dev Phone to the Linux box, although the overall architecture is almost the same, there are few issues needs to be properly addressed due to the slightly different in the user interface and the communication authentication.

Firstly, command line interface is used since there is no display for the Linux box. The transition from the GUI interface to command line interface brought up the issues in the communication sequence to accommodate the changes.

Secondly, when the Bluetooth communication is paired and connected on the Android Dev Phone, there is a GUI passkey agent that will pop up a dialog to assist the passkey entry and then take care of the authentication within the Bluetooth protocol stack. On the Linux Box, since GUI is not available and there is no command line passkey agent that can be used readily, this bring up a major obstacle in getting the Bluetooth communication online between the Trusted Device and the Host. The solution to this issue is then solved by implementing a separate passkey agent to take over the default passkey agent in dealing with the authentication.

7. 3 HYPERVISOR-BASED DEVICEGUARD

In our DeviceGuard project, we have demonstrated our solution to enhance the host security with the assistance of an external device. DeviceGuard application running on the external device to challenge data/system integrity or generating digital signatures for the host by offloading some security features to the external device. A DeviceGuard daemon is deployed on the host, acting as a proxy to execute commands from the DeviceGuard in the external device.

One of the limitations of our solution lies in the DeviceGuard daemon, which runs in user space, bares vulnerability of attacks both from the kernel or user space malwares. To justify the current solution, we need to assume that kernel is trusted and the daemon is not compromised. This assumption faces challenges in providing a robust solution to the problem we addressed in this research. Thus, we are looking at further research to eliminate this assumption by pushing the DeviceGuard daemon down to the kernel space, ideally to the hypervisor.

Begins with an expectation of strengthening the DeviceGuard solution with lower level support either in kernel or hypervisor, we started to look into a hypervisor-based approach with Dr. Xinyuan Wang in GMU. By pushing the DeviceGuard daemon to hypervisor, we will be able to eliminate the limitations that we mentioned above. Then neither malware running in the user space nor in the kernel space will be able to compromise the daemon because hypervisor have higher privileges.

Hypervisor-based Security Research

Hypervisor, or Virtual Machine Monitor is introduced as a hardware abstraction to virtualizes the hardware resources and provide isolation to the Guest OS build upon it. According to Robert P. Goldberg's classification for virtual machine, there are two types of hypervisors, type I as the native hypervisor directly runs on top the hardware and provide management for the hardware resources and type II to be a hosted hypervisor runs within an existing operating system[29]. We will only focus on the type I hypervisor in this study.

There are several prior researches are based the type I hypervisor, such as using the hypervisor to facilitate a VMM-based memory shadowing to prevent the threat from the kernel rootkits[30]; a thin hypervisor-based prevention of persistent rootkits[31]; implementing a virtual machine introspection based architecture for the intrusion detection in the operating system[32]; and yet another Virtual-Machine-based Intrusion Detection on File-aware Block Level Storage[33].

There are quite a few VMMs in the research, including Xen[34], QEMU[35], SecVisor[36] and BitVisor[37], etc. Xen as a representation of type I hypervisor and open sourced, it becomes the most popular hypervisor in the hypervisor-based research [34]. SecVisor[36] and BitVisor[37] are both light-weight hypervisors with SecVisor[36] more focused on the OS kernel's code integrity while BitVisor[37] is focus on the I/O interception-based protection.

BitVisor

BitVisor [37], a thin hypervisor for enforcing security on I/O devices, stands out among various hypervisors that are available in the community. BitVisor introduced the para-passthrough hypervisor architecture by only intercepting I/O accesses of interest and pass-through all other I/O accesses.

Idea and Approach

Our original idea is to develop a para pass-through Bluetooth device driver in the BitVisor. Then we can intercept all the incoming and outgoing Bluetooth traffic in the hypervisor, combined with the prior work that our collaborator Dr. Xinyuan Wang had done in reconstructing the low level file system semantics in the Virtual Machine Monitor (VMM) [38]. It will enable us to provide a more solid inspection/checking on the critical system file or user data on the Guest OS, our Host. This approach will significantly minimize the influence of malware to the DeviceGuard daemon and the OS kernel.

CHAPTER 8 CONCLUSION

8. 1 CONCLUSION

In this thesis, in observing the increasing security threats from the malware in the networked personal computing environment, we proposed a novel approach to mitigate the system and data security on the personal computer by separating the security solution from the host to and external device. The conclusions for this research can be summarized in the following aspects:

1) We identified the system and data security challenges and proposed the external device-assisted approach to mitigate the issues in the traditional host-based solution. As predicted by Imperva, not only the threats from the malware will increase in 2011, but also the data security will be the central of security combat in 2011[2]. Our work is seconded by this prediction in terms of heading the very important direction in the system and data security research.

2) We researched the external device-assisted DeviceGuard solution, described the design of our DeviceGuard framework, defined the Trusted Device and demonstrated the feasibility of using Bluetooth as the communication interface to facilitate the secure communication between the host and the trusted device.

3) We prototyped our DeviceGuard solution based on a Android Development Phone (ADP), which serves as the Trusted Device, to examine the feasibility and effectiveness of our external device-assisted framework. There different use cases that representing the real world security scenario is examined with the results show a highly feasible and effectiveness of our approach to provide security features including system file and user data integrity monitoring, secure signing and secure decryption.

4) We exploited our solution to take the advantage of the proliferation of devices, such as Smartphone as a widely used device in our daily life, for stronger system and data security. As the mobile industry continues to grow and the Smartphone market prosperous, we see a great opportunity in utilizing these devices in providing additional security enhancement to our personal computer.

8. 2 FUTURE WORK

As an exciting security research direction to separate the security solution from the host and offload it to an external device, we have demonstrated the feasibility and effectiveness of our solution in this thesis. But as we mentioned in CHAPTER 7, this solution still left many issues to be addressed in the future work.

- Explore the potential in the hypervisor-based environment to host the DeviceGuard daemon in responding to the command from the trusted device. With the daemon in the hypervisor, we will be able to eliminate the assumption that the daemon on the Host is trusted.
- Explore additional communication interfaces to better facilitate the communication at lower level, especially in hypervisor. Since the hypervisor is designed to provide a lightweight abstraction, to enable the handling of Bluetooth communication we will need to provide a Bluetooth protocol stack which will make the hypervisor complicated.
- Make a case to develop a browser plug-in which will be able to integrate our DeviceGuard framework to provide digital signing for sensitive information. The browser plug-in will intercept and hijack the communication before message is sending out by explicitly or implicitly request the user to obtain a signature from the Trusted Device.

REFERENCES

[1] Symantec, "Symantec Internet Security Threat Report," April 2011 2011.

[2] Imperva, "Security Trend for 2011," *Imperva Application Defense Center,* 2010.

[3] A. Lee, "The Evolution of Malware," *Virus Bulletin Conference,* 2007.

[4] TCG, "TPM Main Specification Level 2 Version 1.2, Revision 116," *http://www.trustedcomputinggroup.org,* 2011.

[5] B. Insight, "Shipments of smartphones grew 74 percent in 2010," *BergInsight.com,* Mar. 10th 2011.

[6] T. Nash, "An Undirected Attack Against Critical Infrastructure," *US-CERT,* 2005.

[7] J. Rutkowska, "Introducing Stealth Malware Taxonomy," *COSEINC Advanced Malware Labs,* Nov. 2006 2006.

[8] J. Rutkowska, "Subverting Vista Kernel For Fun And Profit," *Black Hat USA 2006,* 2006.

[9] K. Kasslin, "Kernel Malware: The Attack from Within," *Association of Anti-Virus Asia Researchers (AVAR) International Conference 2006,* 2006.

[10] D. Vincenzetti and M. Cotrozzi, "ATP anti tampering program," in *Proceedings of USENIX: 4th UNIX Security Symposium, 4-6 Oct. 1993,* Berkeley, CA, USA, 1993, pp. 79-89.

[11] G. H. Kim and E. H. Spafford, "The design and implementation of tripwire: a file system integrity checker," presented at the Proceedings of the 2nd ACM Conference on Computer and communications security, Fairfax, Virginia, United States, 1994.

[12] S. Patil, *et al.*, "FS: An In-Kernel Integrity Checker and Intrusion Detection File System," presented at the Proceedings of the 18th USENIX conference on System administration, Atlanta, GA, 2004.

[13] J. Nick L. Petroni, *et al.*, "Copilot - a coprocessor-based kernel runtime integrity monitor," presented at the Proceedings of the 13th conference on USENIX Security Symposium - Volume 13, San Diego, CA, 2004.

[14] X. Zhao, *et al.*, "Towards Protecting Sensitive Files in a Compromised System," presented at the Proceedings of the Third IEEE International Security in Storage Workshop, 2005.

[15] J. M. McCune, *et al.*, "Bump in the ether: a framework for securing sensitive user input," presented at the Proceedings of the annual conference on USENIX '06 Annual Technical Conference, Boston, MA, 2006.

[16] A. P. Jonathan M. McCune, and Michael K. Reiter, "Safe Passage for Passwords and Other Sensitive Data," *Network and Distributed System Security Symposium (NDSS),* February 2009 2009.

[17] Y.-Y. Chen, *et al.*, "A Mobile Ticket System Based on Personal Trusted Device," *Wireless Personal Communications*, vol. 40, pp. 569-578, 2007.

[18] O. Alina, "Securing a Remote Terminal Application with a Mobile Trusted Device," 2004, pp. 438-447.

[19] J. Zic, Nepal, S., "Implementing a portable trusted environment," *Proceedings of Future of Trust in Computing*, pp. 17-29, 2009.

[20] L. Martin, ""“PC on a Stick” Secures Desktop and Data for Telecommuters, On-the-Go Workforce," *http://www.lockheedmartin.com/news/press_releases/2010/01-18-IronClad.html*, 2010.

[21] BlueZ.org, "BlueZ FAQ: What is BlueZ?."

[22] M. K. Jan Beutel, "Linux BlueZ Howto," Nov. 14th 2001.

[23] BlueCove, "BlueCove Project."

[24] Google, "Android Developer," *http://developer.android.com/index.html*.

[25] Google, "What is Andorid?," *http://developer.android.com/guide/basics/what-is-android.html*.

[26] E. Yueh, "Android Bluetooth Introduction," *http://www.slideshare.net/erinyueh/android-bluetooth-introduction*, 2009.

[27] W. Du, *et al.*, "Uncheatable Grid Computing," presented at the Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), 2004.

[28] M. Ossmann, "Bluetooth Keyboards: who owns your keystrokes?," 2010.

[29] G. Robert P., " Architectural Principles for Virtual Computer Systems," *Harvard University*, pp. 22-26, 1973.

[30] R. Riley, *et al.*, "Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing," presented at the Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection, Cambridge, MA, USA, 2008.

[31] Y. Chubachi, *et al.*, "Hypervisor-based prevention of persistent rootkits," presented at the Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 2010.

[32] T. G. a. M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," *In Proceedings of the Internet Society's 2003 Symposium on Network and Distributed System Security*, pp. 191-206, Feb. 2003 2003.

[33] Z. Youhui, *et al.*, "Virtual-Machine-based Intrusion Detection on File-aware Block Level Storage," in *Computer Architecture and High Performance Computing, 2006. SBAC-PAD '06. 18TH International Symposium on*, 2006, pp. 185-192.

[34] P. Barham, *et al.*, "Xen and the art of virtualization," presented at the Proceedings of the nineteenth ACM symposium on Operating systems principles, Bolton Landing, NY, USA, 2003.

[35] F. Bellard, "QEMU Internals," *Technical Report*, *http://qemu.weilnetz.de/qemu-tech.html*, 2006.

[36] A. Seshadri, *et al.*, "SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes," presented at the Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, Stevenson, Washington, USA, 2007.

[37] T. Shinagawa, *et al.*, "BitVisor: a thin hypervisor for enforcing i/o device security," presented at the Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, Washington, DC, USA, 2009.

[38] X. Jiang, *et al.*, "Stealthy malware detection and monitoring through VMM-based "out-of-the-box" semantic view reconstruction," *ACM Trans. Inf. Syst. Secur.*, vol. 13, pp. 1-28, 2010.