Hardware-Aided Privacy Protection and Cyber Defense for IoT

Ruide Zhang

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

Doctor of Philosophy
in
Computer Science and Application

Apr 22, 2020
Falls Church, Virginia

Keywords: Internet of things, Electromyogram signal, Digital signal processing, Machine learning, Cognitive radio network, Remote attestation, Trusted execution environment, Program analysis, Side channel, Symbolic execution, Compartmentalization

Hardware-Aided Privacy Protection and Cyber Defense for IoT

Ruide Zhang

ABSTRACT

With recent advances in electronics and communication technologies, our daily lives are immersed in an environment of Internet-connected smart things. Despite the great convenience brought by the development of these technologies, privacy concerns and security issues are two topics that deserve more attention. On one hand, as smart things continue to grow in their abilities to sense the physical world and capabilities to send information out through the Internet, they have the potential to be used for surveillance of any individuals secretly. Nevertheless, people tend to adopt wearable devices without fully understanding what private information can be inferred and leaked through sensor data. On the other hand, security issues become even more serious and lethal with the world embracing the Internet of Things (IoT). Failures in computing systems are common, however, a failure now in IoT may harm people's lives. As demonstrated in both academic research and industrial practice, a software vulnerability hidden in a smart vehicle may lead to a remote attack that subverts a driver's control of the vehicle.

Our approach to the aforementioned challenges starts by understanding privacy leakage in the IoT era and follows with adding defense layers to the IoT system with attackers gaining increasing capabilities. The first question we ask ourselves is 'what new privacy concerns do IoT bring'. We focus on discovering information leakage beyond people's common sense from even seemingly benign signals. We explore how much private information we can extract by designing information extraction systems. Through our research, we argue for stricter access control on newly coming sensors. After noticing the importance of data collected by IoT, we trace where sensitive data goes. In the IoT era, edge nodes are used to process sensitive data. However, a capable attacker may compromise edge nodes. Our second research focuses on applying trusted hardware to build trust in large-scale networks under this circumstance. The application of trusted hardware protects sensitive data from compromised edge nodes. Nonetheless, if an attacker becomes more powerful and embeds malicious logic into code for trusted hardware during the development phase, he still can secretly steal private data. In our third research, we design a static analyzer for detecting malicious logic hidden inside code for trusted hardware. Other than the privacy concern of data collected, another important aspect of IoT is that it affects the physical world. Our last piece of research work enables a user to verify the continuous execution state of an unmanned vehicle. This way, people can trust the integrity of the past and present state of the unmanned vehicle.

Hardware-Aided Privacy Protection and Cyber Defense for IoT

Ruide Zhang

GENERAL AUDIENCE ABSTRACT

The past few years have witnessed a rising in computing and networking technologies. Such advances enable the new paradigm, IoT, which brings great convenience to people's life. Large technology companies like Google, Apple, Amazon are creating smart devices such as smartwatch, smart home, drones, etc. Compared to the traditional internet, IoT can provide services beyond digital information by interacting with the physical world by its sensors and actuators. While the deployment of IoT brings value in various aspects of our society, the lucrative reward from cyber-crimes also increases in the upcoming IoT era. Two unique privacy and security concerns are emerging for IoT. On one hand, IoT brings a large volume of new sensors that are deployed ubiquitously and collect data 24/7. User's privacy is a big concern in this circumstance because collected sensor data may be used to infer a user's private activities. On the other hand, cyber-attacks now harm not only cyberspace but also the physical world. A failure in IoT devices could result in loss of human life. For example, a remotely hacked vehicle could shut down its engine on the highway regardless of the driver's operation. Our approach to emerging privacy and security concerns consists of two directions. The first direction targets at privacy protection. We first look at the privacy impact of upcoming ubiquitous sensing and argue for stricter access control on smart devices. Then, we follow the data flow of private data and propose solutions to protect private data from the networking and cloud computing infrastructure. The other direction aims at protecting the physical world. We propose an innovative method to verify the cyber state of IoT devices.

To my family and fiancée

Acknowledgments

Luck has been on my side throughout my life. Words can't express how lucky I am to have received so much supports and encouragement from my family, teachers, colleagues, and friends. I would love to express my greatest gratitude to all of them.

First and foremost, I would like to express my deepest appreciation to my advisor Dr. Wenjing Lou. She supports me to pursue my research interest even if it is not my academic background. And she always encourages me to seek for the truth in science. I am constantly inspired by her rigorous research attitude and professional dedication. Besides, I am super grateful for all the visions she shares with me and they reform the way I look at the world.

I would also like to thank Dr. Ning Zhang. He has guided me through researching in system security and pointed out important directions in this area. He has shared his rich experiences as a security researcher and has helped me to survive in system security research. I sincerely thank him for being a knowledgeable mentor and a cheerful friend.

I would like to thank my committee members Dr. Na Meng, Dr. Ing-Ray Chen, Dr. Tom Hou for their invaluable advice and insightful comments on my research.

I thank Dr. Wenhai Sun, Dr. Changlai Du, Dr. Yan Zheng, Dr. Qiben Yan, Dr. Kun Sun, Ning Wang, Assad Moini for their helpful suggestions in our research works.

I also thank my colleagues at the complex network and security research (CNSR) laboratory at Virginia Tech, Dr. Bing Wang, Dr. Yao Zheng, Dr. Tao Jiang, Dr. Xindi Ma, Dr. Jin Li, Dr. Teng Li, Dr. Yimin Chen, Dr. Yaxing Chen, Dr. Qinlong Huang, Yang Xiao, Di Zhang, Wei Chang, Haomeng Xie, Shanghao Shi, Yang Hu, Jingru Wang for their active discussion.

Lastly, I would like to thank my family for being supportive of all my decisions. And I would like to thank my fiancée. She sacrifices her career opportunity in west coast to come with me to the east coast and takes care of me wholeheartedly. The delicious food she makes and her cheerfulness inspires me and supports me through the tough days.

Funding Acknowledgments

The works presented in this dissertation were supported in part by US National Science Foundation under grants CNS-1446478, CNS-1443889, CNS-1837519, CNS-1916902 and CNS-1916926, by Office of Naval Research under grant N00014-19-1-2621, and by Virginia Commonwealth Cyber Initiative (CCI).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation - The Emergence of IoT

The term, Internet of Things, was first proposed by Kevin Ashton in 1999 [1]. He foresaw a world where every physical object has its own identities and is ubiquitously connected. With recent advances in computing and communication technologies, this future is coming closer and closer. According to Gartner, around 25 billion smart devices are expected to join the Internet of Things by the year 2020 [2]. These smart devices collect data with their sensors and forward them to the cloud for processing. The cloud performs analytics on the received data and sends action commands back to the smart devices.

Although IoT shows great potential and a bright future, privacy concerns and security issues are even more critical. Attacks on upcoming ubiquitous sensors are concerning and a failure in the computing system can now lead to severe consequences to the physical world. For instance, hackers snoop on security cameras [3] and vulnerability in the software stack of a smart vehicle can disable a driver's control on the car [4]. In the meantime, the unique properties of IoT compared to traditional computing systems brings additional challenges. For example, IoT has always-on sensing capabilities that could lead to surveillance of individuals; IoT relies on edge computing and cloud computing for its data processing where service providers may be honest but curious; IoT has limited resources and real-time constraints that restricts it from modern cyber defense mechanisms. To systematically solve unique privacy and security challenges in IoT, we present the architecture of IoT and describe our research efforts tackling corresponding challenges in the following.

1.2 Security and Privacy Challenges in IoT

IoT is composed of three layers, things layer, network layer and cloud layer as shown in Fig. 1.1. The things layer consists of smart devices equipped with various sensors and actuators. They obtain data from and interact with the physical world. The network layer transmits the gathered data from the smart devices to the cloud. Internet service providers, such as Verizon and AT&T, control the network layer [5]. The cloud layer stores and processes the received data and it sends out action commands back to smart devices. Amazon EC2 and iCloud are examples of the cloud layer [6]. To systematically tackle the security and privacy of IoT, we span our research on all three layers of IoT and increases the attacker's power gradually. We start by *understanding the privacy leakage in IoT* at the things layer. Through this investigation, we realize the importance of private data collected by smart devices. We assume an unprivileged attacker here. Because smart things lack access control on new sensors, even an unpriviledged attacker has access to the sensor data. Then we trace the data flow of private data and research on *defending against compromised network components* of the network layer. Here we assume a more powerful attacker who is capable of compromising service providers and launching data breach attacks at runtime. We use trusted hardware to defend against compromised network components. The next stop of private data is the cloud layer. Data analytics is performed on the private data at this layer. Confidential cloud computing with trusted hardware can protect the private data during the analytical phase. However, an insider attacker may hide malicious logic inside the code for trusted hardware during the development phase. So we put our attention to *harden trusted hardware ecosystem.* Finally, we revisit the things layer. This time, instead of sensing capabilities, we focus on the interaction with the physical world aspect of smart devices. We propose to *protect the physical world* by remote verification of the integrity of smart devices. We assume the attacker here is extremely powerful and can compromise a smart device and recover the normal state of the device without being noticed. The following listings identify the unique challenges brought by IoT and position our works. We introduce our research contributions in the next section.

Understanding Privacy Leakage in IoT:

- New types of sensors: IoT devices introduce new sensors at the things layer. They are deployed ubiquitously and collect data 24/7. During data collection, the enforcement of collecting the specified type of data and the access control to the collected data are important from the privacy perspective. However, with a large amount of upcoming different types of sensors, the privacy implication of each new type of data is not known. It is hard to tell which type of sensor data does not have a privacy concern since modern data analytics can dig out hidden facts. [7] and [8] are among the first works working in this direction. They show that seemingly benign accelerometer data on a smartwatch could be used to leak PIN code. Our work [9] explores the privacy implication of EMG signals by designing and building a prototype to infer passwords. Through this work, we argue for stricter access control when introducing a new type

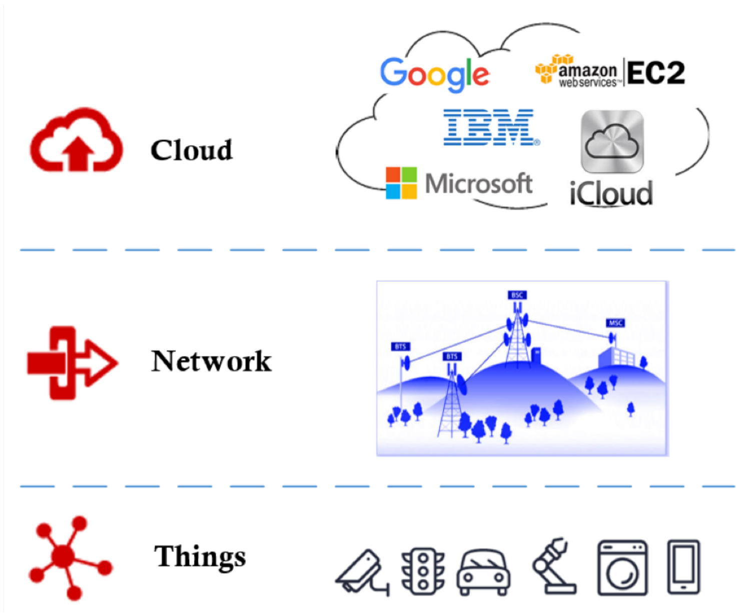


Figure 1.1: IoT architecture.

of sensor.

Defending against Compromised Network Components:

- Edge computing: More data is processed at edge nodes at the network layer. IoT requires service providers to provide its networking functionalities. But networking service providers may be compromised. Confidential computing is a promising tool to solve this concern. To provide confidential computing, there exist two directions of research. [10] represents the cryptographic way to achieve confidential computing. While [11] and [12] design confidential data analytics platforms using trusted hardware. Cryptographic solutions have the advantage of theoretical guarantees while trusted-hardware-based solutions have the benefit of performance. Our work [13] follows the direction of trusted hardware and offers a privacy-preserving networking architecture. The simple adoption of trusted hardware to the network layer is not scalable. We design a scalable scheme to set up trusted execution environments (TEE) on edge nodes. And we apply this scheme to the cognitive radio context attestation scenario and show its efficacy.

Hardening Trusted Hardware Ecosystem

- Trusted hardware: Trusted hardware is gaining popularity for confidential machine learning (ML) in cloud computing. IoT broadly applies ML for data processing and confidential ML can protect private data at the cloud layer. Intel SGX is one of the most promising trusted hardware technologies. It provides a secure enclave on the cloud providers' servers, and consumers can trust their private data on this secure enclave.

However, an insider attack within an organization is a long-lasting threat [14] and Intel SGX relies on the benignity of enclave programs themselves. A malicious insider may hide data leakage code inside the ML enclave code of Intel SGX during the development phase. Information flow analysis [15, 16] is a conventional line of research to discover information leakage in programs. However, it is not suitable for ML enclave programs. This is because the output of an ML enclave program always correlates with its input. Our work [17], PrivacyScope, solves this challenge and provides a static analysis tool to automatically find out information leakage on ML enclave programs. PrivacyScope is also extendable to other TEE technologies.

Protecting the Physical World:

- Interaction with the physical world: More IoT devices at the things layer can affect the physical world and we need to ensure their integrity. Remote attestation of runtime property is a promising solution. [18, 19, 20] provide remote attestations of runtime control flow and data flow properties. However, because of computational power and energy consumption limitations, IoT devices cannot afford heavy runtime overhead. Also, the runtime overhead could break the real-time constraints of an IoT device. Failing to meet real-time requirements leads to a malfunctioning device. Besides, existing runtime property remote attestations focus on a snapshot of the system state, while in the IoT scenario, only a continuous benign running state of a device can guarantee the integrity of a mission. Our work, Conattest, aims at providing remote attestation of memory view flow as a security service under such harsh circumstances. Conattest applies light-weight instrumentation on IoT devices and can provide continuous runtime attestation capability for a device. Conattest allows an IoT user to verify and trust on the past and present execution state integrity of an IoT device.

1.3 Research Contribution in Understanding Privacy Leakage in IoT

The first work of this dissertation focuses on exploring the privacy leakage of IoT. More specifically, we are interested in discovering what kind of seemingly benign sensor data on gesture control devices can lead to serious privacy leakage. Gesture control devices have recently emerged to be the next great IoT gadgets due to its unique ability to enable computer interaction with day-to-day gestures. While these gesture control devices are bringing revolutions to our interaction with the cyber world, it is also important to consider potential privacy leakages from these always-on wearable devices. Especially, the coarse access control on the current IoT system could lead to possible abuse of sensor data.

Although the always-on gesture sensors are frequently quoted as a privacy concern, there hasn't been any study on information leakage of these devices. In this work, we present our study on side-channel information leakage of the most popular gesture control device, Myo. Using signals recorded from the electromyography (EMG) sensor and accelerometers on Myo, we can recover sensitive information such as passwords typed on a keyboard and PIN sequence entered through a touchscreen. EMG signal records subtle electric current of muscle contractions. We design novel algorithms based on dynamic cumulative sum and wavelet transform to determine the exact time of finger movements. Furthermore, we adopt the Hudgins feature set in support vector machine to classify recorded signals segments into individual fingers or numbers. We also apply coordinate transformation techniques to recover fine-grained spatial information with low-fidelity outputs from the sensor in keystroke recovery.

We evaluated the information leakage using data collected from a group of volunteers. Our results show that there is severe privacy leakage from these commodity wearable sensors. Our system recovers complex passwords constructed with lower case letters, upper case letters, numbers, and symbols with a mean success rate of 91%.

1.4 Research Contribution in Defending against Compromised Network Components

The second part of this dissertation focuses on protecting sensitive data from compromised edge nodes. More specifically, we are interested in designing a privacy-preserving attestation framework and apply it to the cognitive radio network (CRN) scenario. Spectrum shortage is a global concern and CRN is envisioned to be one of the key technologies for overcoming this challenge. However, the proper operation of a CRN heavily depends on the compliance of cognitive radios (CRs). Although remote attestation of a CR's radio context is a promising solution, the current remote attestation that requires the target's configuration to be publicly available to the verifier poses a fundamental challenge to the operational security of spectrum users, especially military primary users.

To protect a device's configuration information, we propose PriRoster, a privacy-preserving remote attestation mechanism, that effectively separates the need to know the operational configuration from the capability to execute the verification process correctly at the verifier. PriRoster hides sensitive device and/or radio configuration information from untrusted intermediate verifiers in a public network and enables a range of new applications such as efficient network-wide radio context attestation. Trusted execution environment (TEE) such as Intel SGX is used in our design to provide confidential processing. However, the naive application of TEE suffers from not only poor system scalability, but also information side-channel leakage. We develop trust transfer protocol to significantly enhance system scalability, and the protection against information side-channel attack is accomplished by automatically incorporating obliviousness primitive into the attestation program.

We build a prototype of the proposed PriRoster system using Raspberry Pi, USRP, Intel

NUC, and AWS cloud. The feasibility of our proposed framework is demonstrated by system benchmarks and the effectiveness of the proposed oblivious appraisal functions are verified by recording memory access pattern via code instrumentation.

1.5 Research Contribution in Hardening Trusted Hardware Ecosystem

The third work of this dissertation focuses on hardening TEE ecosystem for IoT users. We research on complimenting the ecosystem of current trusted hardware by designing a new program analysis tool. IoT data analytics is having a profound impact on many sectors of the economy by transforming raw data into actionable intelligence. However, increased use of sensitive business and private personal data with no or limited privacy safeguards has raised great concerns among individuals and government regulators. To address the growing tension between the need for data utility and the demand for data privacy, trusted execution environment (TEE) is being used in academic research as well as industrial application as a powerful primitive to enable confidential computation on the private data with only the result disclosed but not the original private data. While much of the current research has been focusing on protecting the TEE against attacks (e.g. side-channel information leakage), the security and privacy of the applications executing inside a TEE enclave has received little attention. The general attitude is that the application is running inside a trusted computing base (TCB), and therefore can be trusted. This assumption may not be valid when it comes to unverified third-party applications.

In this work, we present PrivacyScope, a static code analyzer designed to detect leakage of private data by an application code running in a TEE. PrivacyScope accomplishes this by analyzing the application code and identifying violations of a property called nonreversibility. We introduce nonreversibility since the classical noninterference property falls short of detecting private data leakage in certain scenarios, e.g., in machine learning (ML) programs where the program output is always related to (private) input data. Given its strict reliance on observable state, the noninterference falls short of detecting private data leakage in these situations. By design, PrivacyScope detects both explicit and implicit information leakage. The nonreversibility property is formally defined based on the noninterference property. Additionally, we describe the algorithms for PrivacyScope as extensions to the run-time semantics of a general language. To evaluate the efficacy of our approach and proof-of-feasibility prototype, we apply PrivacyScope to detect data leakage in select open-source ML code modules including linear regression, k-means clustering and collaborative filtering. Also, PrivacyScope can detect intentional data leakage code injected by a programmer. We responsibly disclosed all the discovered vulnerabilities leading to disclosure of private data in the open-source ML program we analyzed.

1.6 Research Contribution in Protecting the Physical World

The fourth work of this dissertation focuses on the trustworthiness of the IoT devices. We research on designing a continuous attestation scheme for IoT devices. As the proliferation of IoT like drones, PLC, autonomous cars, the security of these systems is important. Since now IoT can affect the physical world and bring physical damage to human beings if controlled by malicious actors. The verification of if a IoT is running at a benign state is of utmost importance. To know an IoT is compromised ahead of time can save people from cyber attacks. Remote attestation is a crucial security service particularly targets for this purpose. However, remote attestation on a static snapshot of software under attest cannot capture runtime attacks, for example, ROP. And recent research on remote attestations of runtime property like control flow sequence or data flow sequence leads to high performance overhead at IoT side and high computational complexity at the verifier side. This heavy burden on IoT could break the real-time constraint of IoT and make the system unresponsive. And the large volume of runtime records makes the time span of the attestation report as short as seconds. While a task for an IoT may span minutes or even hours, for example, a drone food delivery. Besides, existing attestation of runtime property either cannot or needs manual annotation for defending against data-oriented attacks.

In this work, we propose a new runtime property, View Flow Integrity (VFI), and ConAttest to attest on it. Instead of attesting on the sequence of fine-grained control flow, ConAttest compartmentalizes the IoT software into the user-defined amount of segments. Each segment has its memory view during runtime. ConAttest attesting on the sequence of view switch between segments. Through this way, continuous attestation span as wide as hours is achieved, the real-time constraint for IoT is satisfied and data-oriented attacks are mitigated. ConAttest uses ARM TrustZone to record VFI and to transmit VFI report from IoT to the verifier. We create a prototype of ConAttest using a real-world drone and demonstrate its efficacy.

1.7 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 explores the privacy indication of EMG signals. Chapter 3 presents our work on designing a large-scale privacy-preserving radio context attestation for the cognitive radio network. In Chapter 4, we focus on building a static analyzer for finding information leakage in code for trusted hardware. Chapter 5 introduces our continuous runtime view switch flow attestation scheme for unmanned vehicles. At the end, Chapter 6 summarizes the research contributions and future research directions.

Chapter 2

Understanding Privacy Leakage in IoT

2.1 Privacy Impact of Wearables in Augmented Reality

Augmented reality (AR) technology is a variation of virtual reality (VR). Unlike VR, AR promises to enhance our perception of and interactions with the real world, while VR completely immerse users inside a simulated one. AR has been researched extensively in academic world since 1960s. However, previous research mainly focused on the construction and application aspects of AR, there is little study on the security and privacy implications. With recent advancements in wireless networking and embedded devices, AR is no longer a fancy equipment in sci-fi movies. Early generation AR products are already available commercially. Microsoft Hololens was just released in the start of 2016 [21]. People's enthusiasm on AR can be seen through the popular AR game Pokemon Go sweeping the world. Apart from this, Goldman Sachs Group has announced its prediction of an 80 billion dollars market by 2025 for AR and VR [22].

As the AR systems are making their way to people's lives, we believe that it is now a pressing issue to study new security and privacy issues arise with AR. In order to identify new security problem, we ask ourselves: What new security and privacy concerns arise with AR systems? We observe that unlike most of today's desktop and smartphone applications, to provide their intended functionality, complex AR applications will require various, always-on sensing. It is crucial for AR systems to balance the access required for functionality with the risk of an application stealing data or misusing that access. The current common access control for mobile application is to ask for user permission for accessing specific sensor data. Some permissions, such as access to video or voice recording, can be easily identified as sensitive, while others can be subtle and difficult to know the implicit privacy risk. For example, two recent papers [7, 8] leveraged insensitive accelerometer sensors on smartwatch to infer PIN sequence which a user keyed in on an ATM machine.

In this paper, we identify a new type of side channel information leakage from the electromyography (EMG) based gesture devices used in AR systems. EMG signals are subtle electric currents detectable from the skin due to muscle movements. When worn on the arm of the user, the signatures of current can be used to identify the hand gesture, such as holding a fist or waving hands. When the users are interacting with the AR system, gesture control input devices are used legitimately. However, in this work, we show that it is possible for malware listening in the background of AR system to infer sensitive secrets from coarse grained EMG signals, when the users are interacting with other computing systems in the physical world. To demonstrate the feasibility of attack, we use the leading EMG-based gesture control device, Myo, as the platform for the study. We consider two scenarios which happen almost every day in our daily lives.

The first one is tapping in PIN sequence to unlock screen on a mobile device. Nowadays, the authentication system on mobile devices like iPhone relies on PIN sequence. If one can steal the PIN sequence, he is able to access all information (e.g. photo, text ...) on the mobile device. Previous research [7, 8] has designed attacks for ATM based on the fact the user is moving his hand during the input process. However, when it comes to unlocking screen, people often tap with both thumbs rather than a single one, so their hands keep still. The idea is to know which number each thumb is taping from EMG sensor data. Our case study on this scenario shows that EMG signals can significantly reduce the search space for smart device PIN recovery.

The second scenario we consider is typing passwords on a keyboard. If one can deduce the password a victim type on a computer, he potentially may access the resources on the computer and even the victim's bank account. [7] and [23] has provided methods to deduce words a user has input on a keyboard. But modern passwords are seldom words but a combination of signs, letters, and numbers. Therefore, recovering password needs accurate recovery of each symbol typed without the help of a dictionary. The idea is to combine the knowledge of which finger a user moves through raw EMG data and the track of user's hand movement to recover the keystroke a user typed. Our experiments show promising results of recovery complex passwords with high probability. Furthermore, we observe that, even though the assumption of having a prior model of user's typing habit is widely used, it could be unrealistic in certain scenarios. We also perform further investigations to assess the possibility of employing unsupervised learning to develop user-specific models from his own typing. Even though the accuracy is lower than the supervised counterpart, it remains a serious threat to user privacy.

There are several challenges in our attack. First, it is challenging to detect the exact starting points for input events on the keyboard or touch screen device. Because of the pushing and releasing phase of a single keystroke or tapping being so close, the EMG signals appears as a whole rather than distinct signals. Besides, there are eight EMG signal channels, and not a single channel can have enough information to detect all of the starting points. Second, to track hand movement with low-fidelity sensors could be troublesome. The white noise due to the imperfection of sensor would make the estimation of direction and distance easily go wrong. Third, how to determine which finger has moved through the raw EMG sensor data segment is not clear.

To solve these challenges, we design and implement four subsystems based on the key insight of the signals. We borrow ideas from digital signal processing field and machine learning field to help build up the subsystems. Input event subsystem is capable of obtaining the starting points of the noisy EMG signal which solve the first challenge. The second challenge is tackled down by the coordinate transformation subsystem which projects the track of hand movement to the keyboard plane. Number classification subsystem and finger classification system are designed to deal with the third challenge. The former one is able to infer the PIN sequences from EMG signal segments around the starting points, while the latter one can get the exact finger user is moving.

We summarize our main contributions as follows:

- We are among the first to study privacy leakage from EMG signal in gesture-control devices, which is poised to be an essential component in next generation human-computer interaction in AR.
- We design novel algorithms based on dynamic cumulative sum and wavelet transform to determine the exact time of finger movements. Furthermore, we adopt Hudgins feature set in support vector machine to classify recorded signals segments into individual fingers or numbers. We also apply coordinate transformation techniques to recover fine-grained spatial information with low-fidelity outputs from the sensor in key stroke recovery.
- Based on the experiments with one of the most popular gesture-control device, Myo, we show that it is possible to recover sensitive user secrets, such as PIN sequence for unlocking smart devices and complex passwords typed on physical keyboards, using the coarsed-grained information EMG and accelerometer from sensor.

2.2 Background

2.2.1 Gesture Controlled Device - Myo

Myo [24] is a gesture control device that is designed to be worn on the forearms of a user. It's light-weighted with only 93 grams. Fig. 1.2 shows a user wearing Myo arm bands while typing on a keyboard. Multiple sensors are included on Myo to provide seamless human-computer interaction. Myo is connected to computer desktop or mobile devices using bluetooth. It is powered by an ARM Cotex M4 processor which enables it to be used for a full day with one charge. Within the slick design, it houses high-resolution medical grade sensors including eight EMG sensors and one three-axis accelerometer. Fig. 1.1 show some samples of EMG signals, which are recorded when a user is stroking letter 's' on the keyboard. Although Myo

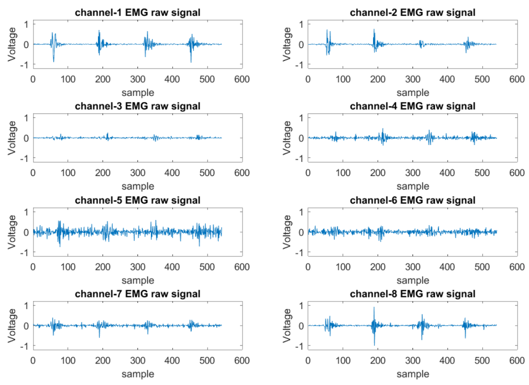


Figure 2.1: Real-life EMG signals collected by Myo

Figure 2.2: A user interacting with AR system using Myo

Figure 2.3: Myo device and its collected signal.

enables many applications because of its high-resolution sensors, we observe that there is a potential privacy leakage caused by them. There is no control of the access to the data streams generated by these sensors. Based on this observation, we believe this vulnerability can be leveraged to record passwords and PIN sequences.

2.2.2 The Myoelectric Signals

Body movement is a result of muscle contraction[25]. A skeletal muscle is comprised of individual cells, or fibres, that are grouped into functional units called motor units. A single motor nerve can innervate the muscle fibres of a motor unit to make them contract together when receiving an electrical stimulus, called an action potential. The electrical stimulus is sent from the motor cortex of the brain to the muscle fibres via the motor nerve. When the motor unit fibres receive an action potential, they also generate action potentials by themselves, which are transient electrical signals that are conducted along the muscle fibre membranes. The motor unit action potential (MUAP) is the summation of the electrical stimulus in the single fibres of the motor unit and it can be elicited by a single action potential sent to a motor unit, which will lead to a transient contraction of the associated muscle fibres. Since muscle contraction results in electrical activity near the skin surface, it is possible to place sensors, called electrodes, onto the skin to detect the electrical activity. The area that an electrode is in direct contact with is referred to as the detection surface[26]. Physiological data recorded by a surface electrode is called a surface EMG. Any portion of a muscle may contain muscle fibers belonging to 20-50 motor units. During a muscle contraction, multiple motor units are repeatedly stimulated. These stimulations typically occur asynchronously to

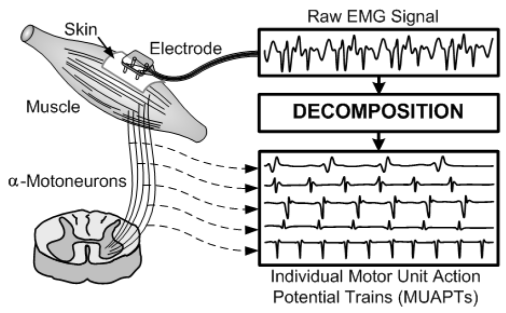


Figure 2.4: EMG signals and their decomposition into MUAPTs.

facilitate smooth movements and delay muscle fatigue. This excitation pattern results in a sequence of MUAPs called a motor unit action potential train (MUAPT). Fig. 2.4 shows that the myoelectric signal represents the temporal and spatial summation of MUAPTs within the pickup region of the recording electrode [27]. As we can see in Fig. 2.4, EMG is a composite of different MUAPTs. The key insight here is that, when people is doing different motions, each MUAPT will contribute differently. This shows the possibility of classifying different finger actions.

2.3 Information leakage from EMG signal Overview

In this section, we first make clear the assumptions. Then we demonstrate the steps of how our supervised implementations could infer passwords and PIN sequences. Besides, at the end of each part, we demonstrate the potential advantages of information leakage based on our methods.

2.3.1 Threat Model

For our supervised attacks, we assume an application has been installed beforehand on a user's computer or mobile device depending on to which Myo armbands are connecting. This application can access insensitive EMG and accelerometer sensor data and the application can interact with a remote server. We also assume our application includes an initialization phase. During the initialization phase, a user is instructed to do a series of tapping actions. Applications with these abilities are common. For example, a health monitoring application would serve all the needs.

2.3.2 Steps in deducing passwords

We implement a realistic system to clarify the possibility of using side-channel provided by gesture control device to recover passwords. Fig. 2.5 presents the system framework for deducing passwords. In the system, we first detect the starting points of each keystroke. Then we extract the EMG signal around the starting point. With machine learning schemes, we map the EMG signal to finger. At the same time, we extract the trace of hand movement relative to keyboard. Combining the finger and trace of hand, we deduce the exact keystroke a user has input. The methods applied in the system is elaborated below.

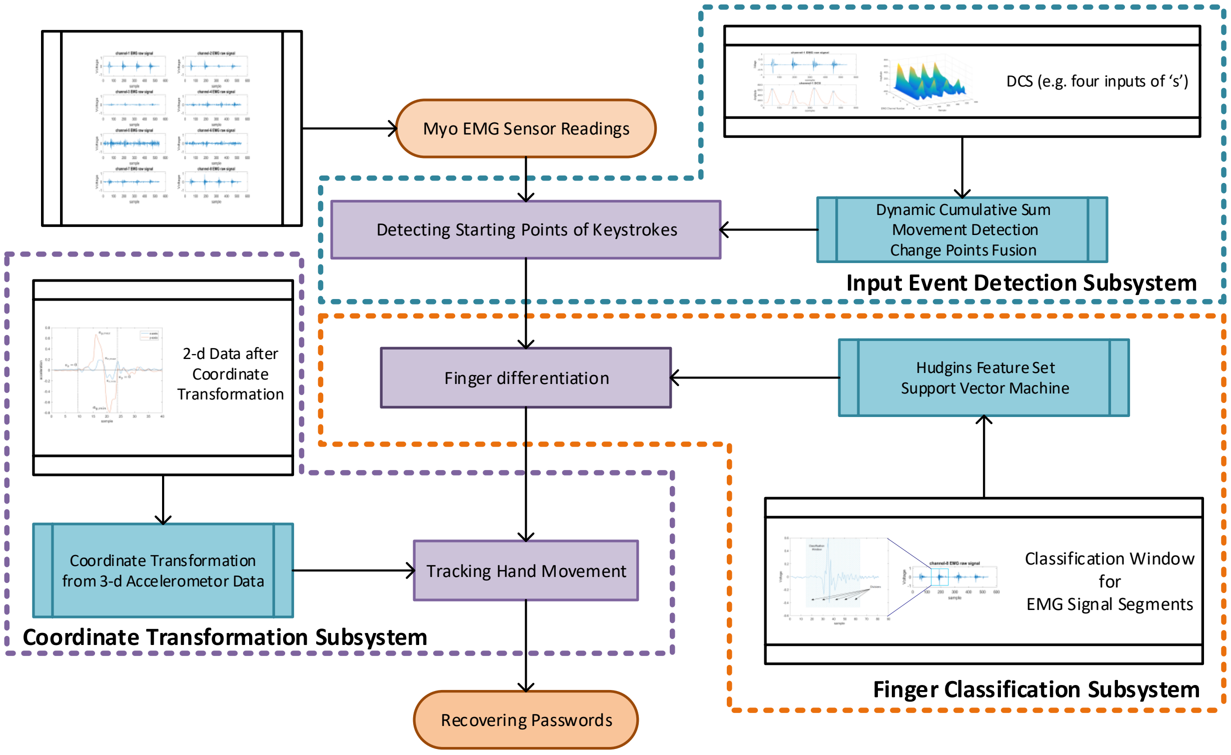

DCS (e.g. four inputs of 's')
Myo EMG Sensor Readings
Detecting Starting Points of Keystrokes
Dynamic Cumulative Sum
Movement Detection
Change Points Fusion
Input Event Detection Subsystem
2-d Data after Coordinate Transformation
Finger differentiation
Hudgins Feature Set
Support Vector Machine
Coordinate Transformation from 3-d Accelerometor Data
Tracking Hand Movement
Coordinate Transformation Subsystem
Classification Window for EMG Signal Segments
Finger Classification Subsystem
Recovering Passwords

Figure 2.5: System overview of deducing passwords.

Detecting keystrokes. The sensor data for a user's finger movements need to be separated first when he types on a keyboard. By the methods implemented in the input event detection subsystem, we are able to separate the signals. The methods include dynamic cumulative sum (DCS) in [28, 29, 30, 31, 32] and our new algorithms, movement detection algorithm and change points fusion algorithm.

Finger differentiation. With the timestamps from the previous step, we now direct them into the finger classification subsystem. Finger classification subsystem uses Hudgins feature set [33] and adopt supervised machine learning method, Support vector machine (SVM) [34], to generate classifier which is .

Track hand movement. Utilizing the starting points from the first step, we employ the coordinate transformation subsystem to obtain the distance and direction of hand movement relative to keyboard between consecutive keystrokes.

Recovering passwords. Combining the information from the second and third steps, we can now infer which key the user has typed each time. Furthermore, by recording the key sequences when a user is typing passwords, we successfully recover a user's passwords.

Compared with previous works on keystroke inferring methods, our work has multiple advantages as follows:

- Non-intrusive. In [35, 36, 37], they have to deliberately put external devices like microphone or touch screen device close to the keyboard. Otherwise, they cannot access signals with enough signal noise ratio to do the inference. However, in our scheme, Myo is on the user's forearm. So we do not have to set up any specified scenario.
- No access to highly sensitive sensors. In [7], they assume the application can gain access to the audio recorder of the mobile device and in [38, 39], they assume the application can obtain the camera data. In our method, we only need access to accelerometer and EMG sensor, which are pretty common for any gesture control applications.
- Capability of recovering non-contextual inputs. In [40, 7], linguistic models or dictionaries are employed to infer the words. Their methods cannot recover non-contextual inputs like passwords. Nonetheless, with our scheme, we can achieve letter-granularity precision which is necessary for recovering passwords.
- High accuracy. We adopt accelerometer and EMG sensor data which can generate high entropy. And this leads to the high letter-granularity accuracy of our scheme.

2.3.3 Steps in recovering PIN Sequences

The other scenario we consider is that the victim is holding his touch screen device with both hands when unlocking screen. In this case, our application only require the access of EMG sensor. At first, we detect the starting points of each tapping action. Then we extract the EMG signal around the starting point and map the EMG signal to number. The methods applied in the demonstration is elaborated below:

Detecting thumb movement. We slightly modify the parameters in our input even detection subsystem to adapt to this scenario because the patterns of EMG signal are similar to the keyboard scenario. The outcome of this step is the starting point for every thumb movement.

Classifying thumb movement. With the starting point of every thumb movement, we direct EMG signal segments into the number classification subsystem. SVM and Hudgins feature set are adopted in the number classification subsystem.

Previous works [41, 42, 43] utilize data through the accelerometer embedded in the targeted device which is the touch screen device in our case. They cannot be used if the touch screen device is free of malware. On the contrary, the method we adopt does not need direct access to the target device. In addition, our attack in this scenario is also non-intrusive and does not need access to highly sensitive sensors.

2.4 Information Extraction System Design

In this part, we discuss the detailed technologies applied in our supervised attacks. We start with introducing the modeling of EMG signal and then we introduce the four subsystems to accomplish the demonstrations.

2.4.1 EMG Signal Modeling

Input event detection subsystem is designed for detecting the starting points of the motions. Detection of finger movement is based on the analysis and characterization of forearm EMG during an action. In our case, the recorded electromyographic signals can be modeled by a random process

$$x(t) = \sum_{i=1}^{n} C_i(t) + \sum_{i=1}^{n} R_i(t) + n(t) \tag{2.1}$$

This equation is a composite of multiple types of signals collected by the EMG sensor like activity burst and noise. $\sum_{i=1}^{n} C_i(t)$ are our target signals which are caused by the pushing actions while $\sum_{i=1}^{n} R_i(t)$ are caused by the releasing actions. The superscript n means the number of keystrokes performed. Both the pushing and releasing actions follow a pattern of short potentials which appear with the acts of fingers. At last, $n(t)$ is the white noise caused by multiple factors like environmental conditions or thermal noise.

2.4.2 Input Event Detection

Detection of human movement by simple threshold methods and simple energy comparison between neighbor signal windows has been presented in [44, 45, 46] and in [47]. However, those methods are not suitable for cases where signals appear dynamic and noisy. Also, to obtain the starting point of movement visually as in [48] is neither accurate nor efficient in our case. In addition, no unique database can be set up for any person [28]. Fortunately, the generalized likelihood ratio test in [49, 50] can be utilized to build DCS which can be used to detect human movement [28, 29, 30, 31, 32]. Nonetheless, their method cannot be directly applied to our case because our EMG signals has eight channels and requires distinguishing between two similar patterns (i.e., pushing and releasing of keys). In order to construct an

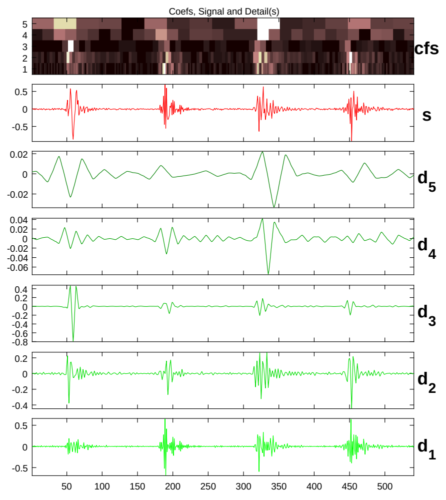


Figure 2.6: Multicscale decomposition using orthogonal wavelet.

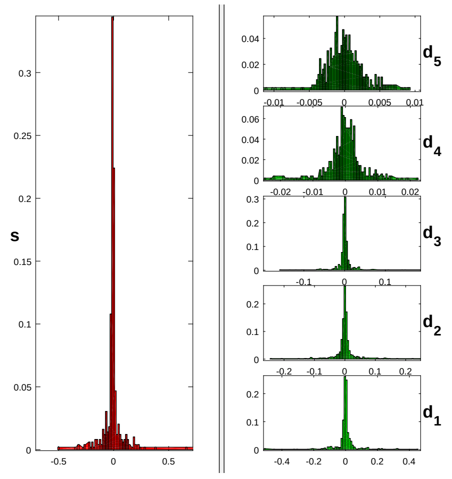


Figure 2.7: 5-levels detailed WT decompositions histograms.

Figure 2.8: Analysis of EMG signals.

accurate movement detection algorithm, we first obtain the dynamic cumulative sum (DCS) of the collected EMG signals. Then we design novel algorithms to obtain the starting point of each target action from DCS. The key insight is that, DCS will reach maximum during the motion as proved in [51].

Dynamic Cumulative Sum

DCS is an improvement of CUSUM (or cumulative sum control chart) [52]. However, CUSUM is only suitable for situations where the priori knowledge of what change will happen to the signal after the point of change is known. Thus we adopt DCS which suits circumstances where the priori knowledge is not required. One prerequisite of applying DCS is that signals must follow Gaussian distribution. We will show that in the following part of this section. Basically, DCS calculate local cumulative sum of likelihood ratios between segments before and after time point t_m. Let us assume the two segments are $S_b^{(t_m)}$ (before t_m) and $S_a^{(t_m)}$ (after t_m) and the width of these two segments is W. $S_b^{t_m} : x_{i;i=t_m-W,\ldots,t_m-1}$ follows a pdf $f_{\theta_b}(x_i)$ and $S_a^{t_m} : x_{i;i=t_m+1,\ldots,t_m+W}$ follows a pdf $f_{\theta_a}(x_i)$. The parameters $\hat{\theta}_b$ and $\hat{\theta}_a$ are estimated using $S_b^{(t_m)}$ and $S_a^{(t_m)}$. The DCS is defined as the sum of the logarithm of likelihood ratios from the beginning of the signal to the time t_m:

$$DCS^{(t_m)}(S_a^{(t_m)}, S_b^{(t_m)}) = \sum_{i=1}^{t_m} Ln \frac{f_{\hat{\theta}_a}{}^{(t_m)}(x_i)}{f_{\hat{\theta}_b}{}^{(t_m)}(x_i)} \tag{2.2}$$

where, the θ can be estimated by the variance of each segments.

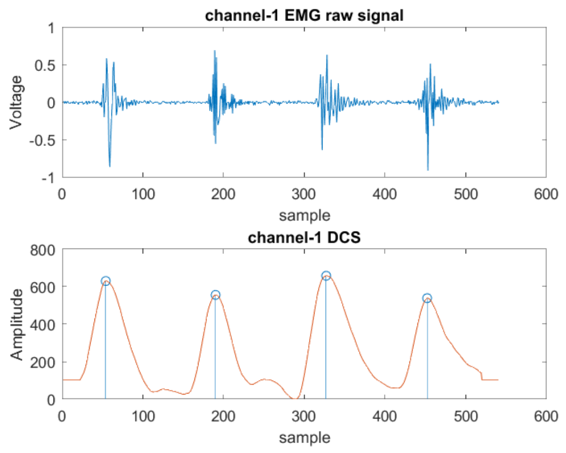


Figure 2.9: DCS and point of change for one channel.

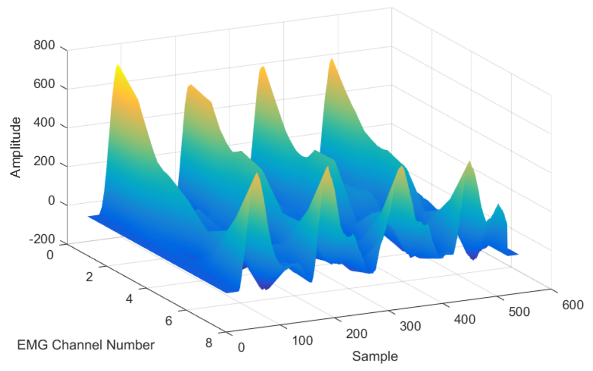


Figure 2.10: DCS of all eight EMG channels.

Figure 2.11: DCS of EMG signals.

We further adopt wavelet transform (WT) [53] to improve the movement detection accuracy. The prerequisite of using WT in DCS is that the WT decompositions of EMG signals are multidimensional Gaussian. Fig. 4.2 presents an example of the histograms of randomly selected 600-sample and its WT decomposition at five scales. It shows that our case meets the prerequisite. WT is applied to both the before and after segments. The choice of motherlet is crucial when adopting WT in signal processing. In [29], they conclude that the best wavelet for human movement EMG signal processing is the second-order Coiflet associated with the first five decomposition scales obtained by Shannon entropy criterion. The results of our experiment reinforce their conclusion. Fig. 4.1 illustrate multiscale decomposition of EMG signal in Fig. 2.1 using a second-order Coiflet orthogonal wavelet. The time interval between samples in is 5 milliseconds.

The DCS corresponding to signals in Fig. 4.1 is depicted in Fig. 5.1 and Fig. 5.2. We observe that DCS in some channels has larger maximum than others and larger maximum can make the movement detection more accurate. From Fig. 5.2, channel 1 and channel 8 which are next to each other have greatest maximum. This is because the muscle used to perform these actions majorly sits close to each other. The detection decision is included in Fig. 5.1 as blue circles. We easily observe that the turning point of the DCS indicates the existence of an action as expected. In addition, we could find two bumps which indicate the releasing movement in the DCS of the first two keystrokes.

Algorithms

Our next task is to extract the timestamps of starting points for movements from the DCS described above. We develop two algorithms to accomplish the task. Movement detection algorithm is developed to calculate the DCS and detect the pushing movements in the EMG signals. In the movement detection algorithm, we set up threshold T to get rid of the releasing movements. The value of T is set to 350 empirically. Then we redirect the output timestamps to change point fusion algorithm. Empirically, we use two channels and the threshold x in the change point fusion algorithm is set to 20 which is 100 milliseconds.

Algorithm 1 movement detection algorithm

```
 1: At each sample, the $DCS$ is calculated according to (3) using the two segments $S_b^{t_m} : x_{i;i=t_m-W,\ldots,t_m-1}$
    and $S_a^{t_m} : x_{i;i=t_m+1,\ldots,t_m+W}$
 2: **if** The DCS has a turning point at that sample which indicates that it may be a finger pushing movement
    or a finger releasing movement **then**
 3:     **if** There is a releasing movement before **then**
 4:         This is a pushing movement, record the timestamp
 5:         Move to the next sample
 6:     **else**
 7:         **if** The difference between this movement and the former pushing movement exceeds a threshold $T$
            **then**
 8:             This is a releasing movement
 9:             Move to the next sample
10:         **else**
11:             This is a pushing movement, record the timestamp
12:             Move to the next sample
13:         **end if**
14:     **end if**
15: **else**
16:     Move to the next sample
17: **end if**
18: Output the timestamps recorded
```

Algorithm 2 change point fusion algorithm

```
 1: Get the recorded timestamps from the output of movement detection algorithm for selected channels
    and sort them into list $L1$ ascendingly.
 2: Generate an empty list $L2$
 3: Start from the first element $l_m$ in $L1$ and do the following.
 4: **if** Any timestamp from other selected channels are close to $l_m$ within threshold $x$ **then**
 5:     Add $l_m$ into list $L2$
 6:     Delete timestamps close to $l_m$ within threshold $x$ in list $L1$
 7:     Delete $l_m$ in list $L1$
 8:     Go to the next element in list $L1$
 9: **else**
10:     Go to the next element in list $L1$
11: **end if**
12: Output the list $L2$
```

2.4.3 Finger And Number Classification

Finger and number classification subsystem are similar to each other, so we introduce them together in this part. Finger classification subsystem is about classifying which finger is a user using from a part of EMG signal. While the job of number classification subsystem is to determine which number has been tapped from a segment of EMG signal. With the input event detection subsystem, we now have the starting point for each finger action. Here we set up a window for the EMG signal at each starting point. Empirically, classification achieves decent performance when the size of the sliding windows is 45 samples which is 225 milliseconds for finger classification subsystem and 60 samples which is 300 milliseconds for number classification subsystem. The insight here is that the action of tapping is larger than the action of stroking key. Besides, we add offset to the starting point so that the sliding window could include the signal for the whole action.

Feature extraction and classification

We extract Hudgins feature set [45] from each motion. The Hudgin's time-domain features are comprised of five different features for a given classification window. Here we divide the classification window into five equally-divided segments as in Fig. 2.12 and each of the segments will have five features. So there will be a total of 30 features per channel (including the undivided classification window). These features include mean absolute value (MAV), difference MAV, zero crossing, slope sign changes and waveform length. Then we take advantage of the labeled samples collected in the initialization phase to do a supervised learning using SVM classifier in our implementation. After training, the SVM classifier could give us which finger or number a new signal segment is related to.

2.4.4 Hand Movement Tracking

The last subsystem is coordinate transformation subsystem which calculates projection of distance and direction of the hand movement between every two successive keystrokes onto keyboard plane. The distance and direction derivation sections are similar to the technique used in [8] and we follow their symbol and sign in our description of this subsystem. Our scheme and theirs differs on the coordinate alignment part. [8] assume the adversary has placed other accelerators on the target plane (which is keyboard plane in our case). However, in our case, we only assume the keyboard is placing on a flat plane and all users' forearms have similar initial position towards the keyboard.

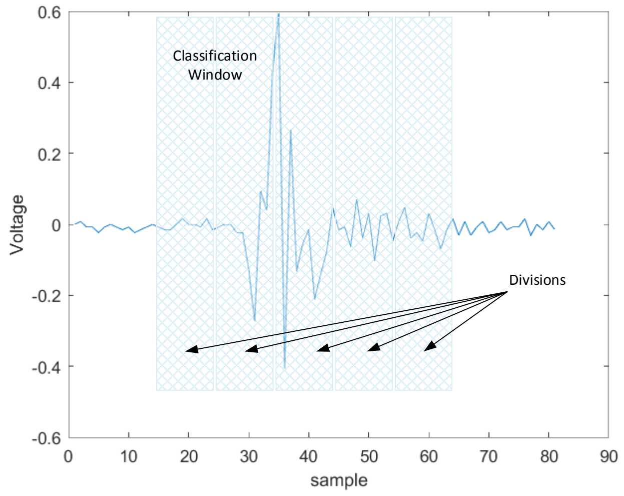


Figure 2.12: Classification window and its divisions.

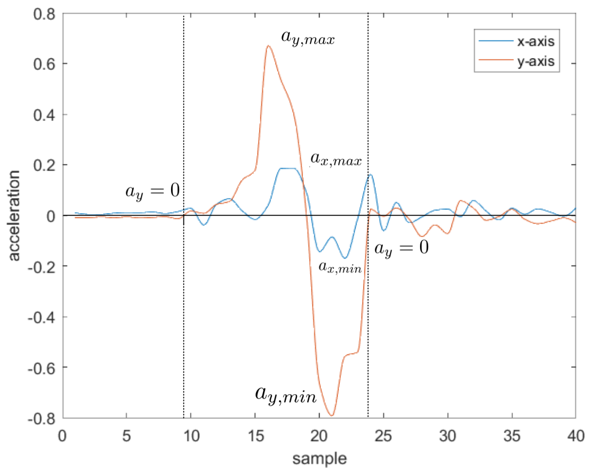


Figure 2.13: Acceleration after projection.

Projection matrix

In order to calculate the displacement of hand moving on keyboard, we need to perform coordinate system transformation. So our goal is to obtain a projection matrix. According to our assumption, it only needs to be calculated once. The first coordinate system we build is the keyboard coordinate and the second coordinate system is the device coordinate. The job required is to transform the displacement in the device coordinate to the keyboard coordinate. This way, we can observe the finger movement projected to keyboard plane directly. For the sake of calculating the displacement between two consecutive keystrokes, we assume the origin of the two coordinate systems overlap. To calculate the P matrix, we need to have the correspondences between three different points in the two coordinate systems. According to our assumptions, the gravity is parallel with the z axis of keyboard coordinate. Thus, we assume the coordinate for gravity is $(0, 0, 1)$ for convenience, which will not affect the construction of P. The initial reading of accelerometers in Myo is caused by the gravity, so let us assume the initial readings are (x_1, y_1, z_1). And (x_1, y_1, z_1) is according to the device coordinate. This vector in the keyboard coordinate will be $(0, 0, 1)$ according to our assumption. We obtain the other two points by asking a user to type in 'f' 'r' and 'f' 'g' respectively. This way, we get the vector $(0, 1, 0)$ and $(1, 0, 0)$ on keyboard coordinate and at the same time, we record the readings of Myo. The readings of Myo can be used to construct the displacement of hand in every two consecutive keystrokes. The method applied here to get the displacement is integration. It is well known that the integral of acceleration is velocity and the integral of velocity is displacement. Let us assume the recorded vectors are (x_2, y_2, z_2) and (x_3, y_3, z_3). So we get the linear algebra formula $\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix} * M$.

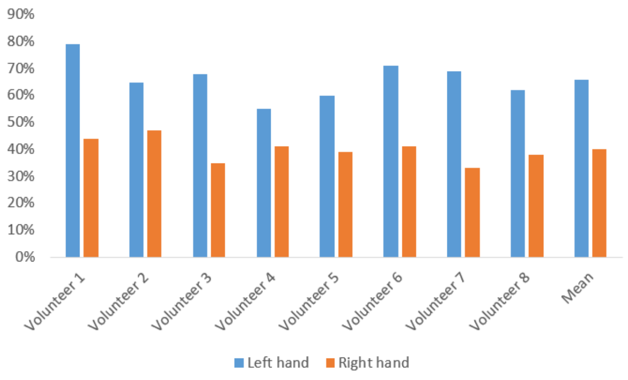


Figure 2.14: Number classification accuracy

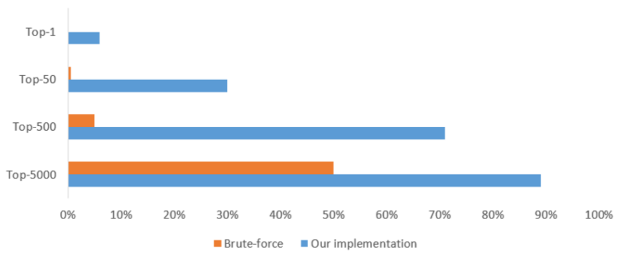


Figure 2.15: Top-k success rate

Figure 2.16: Evaluation on inferring PIN sequence.

By solving these linear equations, we can obtain the linear transformation matrix M. And the projection matrix P can be constructed by extract the first two columns of M.

Distance estimation and direction derivation

Because we obtain the starting points of the movements from movement detection, we can extract the accelerometer sensor within the time interval between two consecutive keystrokes out accurately. Besides, the movement of forearm is before the stroking of keys, so we add an offset here to capture the whole interval of the movements. Fig. 7 shows an example of acceleration data after projection between two keystrokes 'v' and 't'. To get rid of the noise brought by hand vibration, we can easily observe that acceleration captured during the two consecutive motions has unique patterns on x and y axes (i.e., either up-and-down or down-and-up shapes due to different moving directions). Thus, we follow the technology in [8] to get the starting point and ending point by first zero-crossing point occurring before and after the unique acceleration pattern. So the acceleration is always like a pattern of $[0, a_{k,max}(a_{k,min}), 0, a_{k,min}(a_{k,max}), 0]$ (k could be x or y).

Therefore, our strategy can be separated into following parts: 1) extract the 3-axis acceleration between the releasing and pressing points of two consecutive keystrokes; 2) project the 3-axis accelerometer data to the keyboard plane; 3) examine the 2 dimensional data to find $[a_{x,max}, a_{x,min}, a_{y,max}, a_{y,min}]$; 4) find the starting point of the the movement by searching the first time that acceleration crosses the axis (i.e., zero-crossing point) before $a_{k,max}$ or $a_{k,min}$, whichever comes first; 5) similarly, find the ending point by searching the zero-crossing point after $a_{k,max}$ or $a_{k,min}$, whichever comes later; The two accelerations after projection within the range of starting and ending points correspond to forearm movement and are employed to calculate the distance and direction of the forearm movement. And we set the hand position at starting point as the origin of both coordinate systems.

The distance calculation is trivial. We consider the movements in both x and y axes bounded by their starting and ending points. As the distance is two times integration of accelerations, we apply trapezoidal rule to approximate the distance on each axis.

We employ the acceleration data for x and y axis and the results of distance calculation to do direction derivation. From the comparison of the positions of $[a_{x,max}, a_{x,min}, a_{y,max}, a_{y,min}]$, we can derive the direction range. The whole 360° can be split into eight direction ranges (start from x-axis). 0° to 45° is range 1, 45° to 90° is range 2 and so on. So if the position of $a_{y,max}$ is before $a_{y,min}$, it means the direction angle sits in 0° to 180°. If the position of $a_{x,max}$ is before $a_{x,min}$, it means the direction angle sits in 270° to 90°. With these two comparisons, we can locate the direction angle into a 90° range. Then we compare the distance obtained from distance calculation. If the absolute value of distance along x axis is larger than the absolute value of distance along y axis, it means the direction angle is either in 315° to 45° or 135° to 225°. Combining this comparison with former comparisons, we can locate the direction angle into a 45° range, which can be used to differentiate between consecutive keystrokes pairs like 'f' 'v' and 'f' 'b'.

2.5 Evaluation on Information Extraction System

We conduct experiments on 8 volunteers, and all the participants are between 20 and 40 years old, including 3 women and 5 men. All the volunteers have the ability to type in words following the standard type method [54] fluently. All participants are instructed to type or tap as they usually do. The participants are also instructed to avoid huge body movements and keep the wrist always above the desk when typing words. They are instructed to hold the iPad with both hands and tab with thumbs when unlocking screen.

2.5.1 Evaluation Matrices

We develop the following metrics to evaluate our system.

Classification accuracy: To evaluate the performance of classifier, we define classification accuracy as the possibility of correct classification. The ground truth is recorded by us during the experiments.

Top-k success rate: Given an experimental run of a password or PIN activity, our algorithm could return multiple top candidates of password or PIN sequence. We define that a Top-k success hit if the password or PIN resides in the returned k candidates list.

2.5.2 Implementation and Evaluation

In the experiment to infer PIN, we ask the volunteers to tap in ‘0’ to ‘9’ each for 20 times for two rounds. The data gathered from the first round is used to train the SVM classifier while the data collected in the second round is used to do the testing. Then, the volunteers are instructed to type in different length-4 PIN sequences. We adopt the input event detection subsystem to extract the timestamps of starting points and it fully detected all the movements. We start by evaluate the performance of number classification subsystem. There is one SVM classifier for each volunteer trained by the their own labeled samples. The classification accuracy for each volunteer is shown in Fig. 8.1 We have also explored whether it is possible to generate a general classifier for all volunteers. However, the general classifier achieves classification accuracy close to random guess. The reason is that the EMG signals collected from volunteers relate to the structure of the volunteers’ muscle and every volunteer has different muscle structure.

We can observe that a big difference of the number detection accuracy between left hand and right hand exists. This is because people tend to type in ‘1’ ‘4’ ‘7’ ‘8’ ‘0’ with left hand and the others with right hand. And the numbers touched by right hand are close to each other. The way we use to generate candidate is basically replacing classification outcome one by one. For example, if the ground truth is ‘5709’. But the classifier give us ’5749’. Then the top-4 candidate list will be ‘0749’ ‘5049’ ‘5709’ and ‘5740’. So we have recovered the right typing in the top-3 candidate list. Fig. 8.2 presents the top-k success rate of the PIN sequence reconstruction compared to simple brute-force.

In the experiment to recover passwords, we ask the volunteers to type in ‘a’ ‘s’ ‘d’ ‘f’ ‘j’ ‘k’ ‘l’ ‘;’ each for 20 times for classification, and another 20 times for testing. Then, we ask volunteers to type in multiple passwords. The construction of the passwords include lower case letters, upper case letters, numbers and symbols. We first evaluate the performance of finger classification subsystem. Input event detection subsystem is employed here and it has one hundred percent accuracy. There is one SVM classifier for each volunteer trained by their own labeled samples. The classification accuracy for each volunteer is shown in Fig. 9.1.

Basically, the way we use to generate candidate is to replace one classification outcome one by one. For example, if the ground truth is ‘see’, which is {ring finger, middle finger, middle finger}. But the classifier gives us {ring finger, index finger, middle finger}. Then the top-3 candidate list will be {pinky finger, index finger, middle finger}, {ring finger, middle finger, middle finger} and {ring finger, index finger, index finger}. So we have recovered the right typing in the top-3 candidate list. The other factor we use to generate the candidate list is from the coordinate transformation subsystem. Coordinate transformation subsystem can make it easy to differentiate situations like ‘r’ and ‘v’. Fig. 9.2 presents the success rate of the password reconstruction.

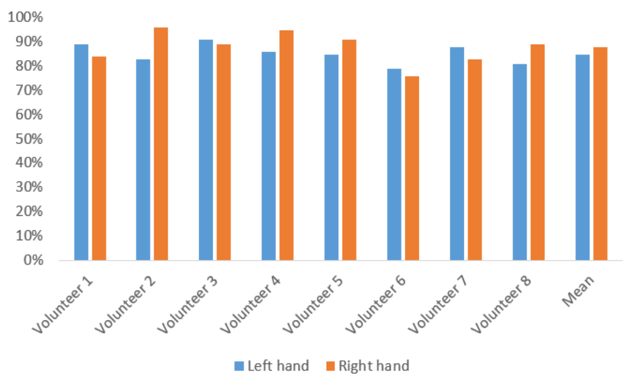


Figure 2.17: Finger classification accuracy

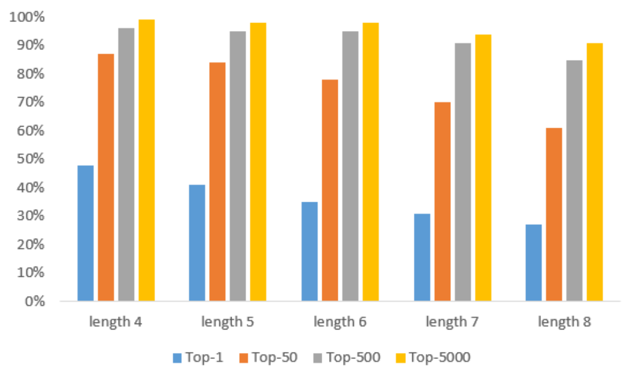


Figure 2.18: Top-k success rate

Figure 2.19: Evaluation on inferring passwords.

2.6 Further exploration: Password Extraction Without Prior Information

In the supervised implementation, we assume our application includes an initialization phase, during which a user is instructed to do a series of tapping actions. We make a further effort to make our attack even more stealthy and practical. We get rid of the training phase in our unsupervised implementation. The assumption reduces to that the application instead only need to have the ability to record EMG sensor data of the user typing in an article.

2.6.1 Extracting user muscle models without labeled data

The difference between supervised and unsupervised implementation lies on the finger classification subsystem. Other than that, the procedure is the same. In finger classification subsystem for unsupervised implementation, it uses model obtained from unsupervised learning. We record the whole process of a user typing an article and then we apply k-means clustering method with principal component analysis (PCA) to all the unlabeled samples collected from the article. With the letter frequency analysis of English text, the subsystem can know which cluster corresponds to which finger. For example, if one of the letters ‘a’ ‘q’ ‘z’ appears in the article, it means one keystroke with little finger. And according to the letter frequency ranking of the sum of ‘a’ ‘q’ ‘z’ in English text, we can know the correspondence between finger and cluster. Because the cluster with the greatest amount of samples relates to the finger used most frequently in English text typing. The frequency analysis of English text is in Fig. 10.1 [55]. In comparison, Fig. 10.2 is the frequency analysis of the article we used in our experiment. After training, the classifier could give us which finger a new signal segment is related to.

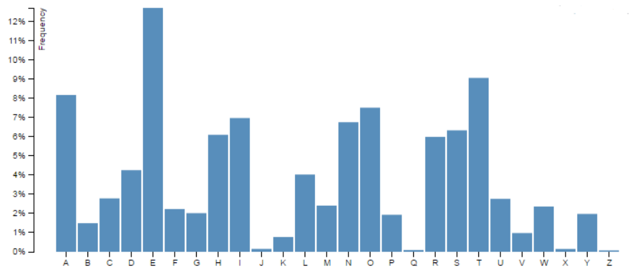


Figure 2.20: In English literature.

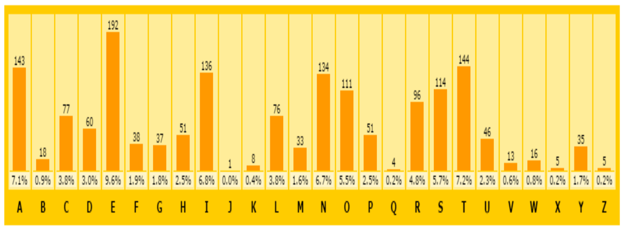


Figure 2.21: In our article.

Figure 2.22: Comparison of relative frequencies in English literature and in our article.

2.6.2 Experiments and results

As usual, we evaluate the performance of finger classification subsystem first. We ask the volunteers to type in an article with two thousand letters. Then, we ask volunteers to type in same passwords as the first implementation. The EMG data gathered is put into the input event detection subsystem. We implement a small experiment here to test the performance of our input event detection subsystem. We extract the signal segment of twenty letters from each volunteer and combine the signal segments together. Then we put the composite signal into our input event detection subsystem. It turns out if we set the maximum false positive rate to be five percent, our subsystem could detect eighty four percent of the events. With the timestamps, we can extract the keystroke samples out. We adopt k-means clustering to the unlabeled samples to find the centroids. The finger classification accuracy for each volunteer is shown in Fig. 11.1. What worth mentioning here is that thumb movement is not required when recovering password. So whenever the sample segment to be classified is close to the centroid for thumb, we label it as index finger. We apply the same way as the first implementation to generate candidate list and Fig. 11.2 presents the top-k success rate of the password reconstruction.

2.7 Related Work

2.7.1 Digital Key Steal

It has been a long history of adversaries trying to steal the key entries of users on key-based security systems. A popular tool broadly employed by adversaries is keylogger which can log all the keystrokes on the computer. The only drawback of this tool is that it leaves footage on the victim's computer. Some other traditional attacks in [38, 39] rely on shoulder surfing and hidden cameras. In these kinds of attacks, the malicious code will gain access to the direct

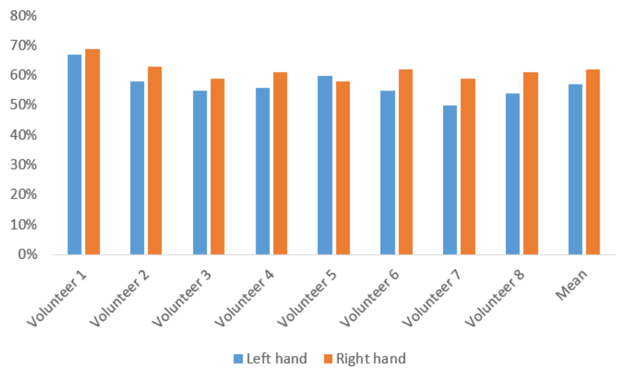


Figure 2.23: Finger detection accuracy

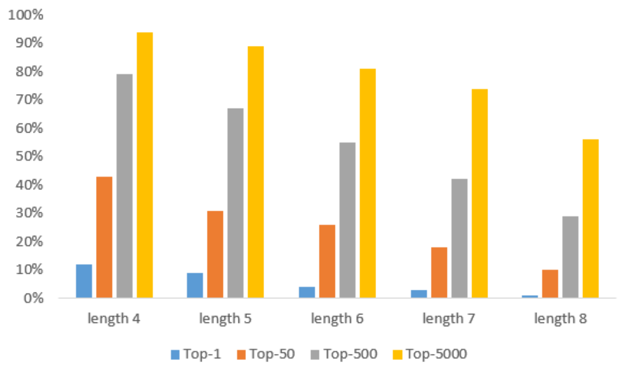


Figure 2.24: Top-k success rate

Figure 2.25: Evaluation on unsupervised typing model.

visual image of the key entry process. However, the capability to gain access to camera is a strong assumption. In order to achieve stealthiness, we can see another line of work which focuses on developing novel side-channels to infer the key entries. For example, the sound produced by different keystroke can be a valid side-channel to infer keys as described in [35]. Following this line of research, a bunch of other valid side-channels are discovered such as electromagnetic emanations [56], acoustic emanations [37, 57, 58], optical emanations [59], and even the vibration of wooden desk [36]. The main drawback of these side-channels is that special equipments need to be deployed beforehand. Researchers noticing this drawback try to install malicious application on smartphone to exclude the strong assumption. However, the experiment results in [36, 37] indicates that smartphones need to be placed close enough to the keyboard by the victim, which is not the case in most scenarios. In recent years, wearable devices are becoming part of modern life. Researchers who notice the trend start to exploit sensors on wearable devices to do keystroke or PIN sequence inference. Two recent papers [8, 7] leverage sensors on smartwatch to infer PIN sequence. Explicitly, both of them take advantage of the accelerometers to measure the distance between two different input on ATM machine to recover PIN sequence. [8] takes one more step to get rid of training phase which is required in [7]. [7] also provides a method which employs accelerometers and audio recorder on smartwatch to infer keystrokes on keyboard. However, it can only recover user's typed words and will not work with non-contextual inputs. [60] employs camera to capture user's hand and the back side of the touch screen to recover smartphone lock PIN. The method has a low inference accuracy and it assumes the capability for malicious code to access sensitive sensor.

2.7.2 Human Movement Detection

Detection of human movement by simple threshold methods and simple energy comparison between neighbor signal windows has been presented in [44, 45, 46] and in [47]. However, those methods are not suitable for cases where signals appear dynamic and noisy. Also, to obtain the timestamp of movement visually as in [48] is neither accurate nor efficient in our case. In addition, unlike classical detection problems where every event is well identified in its time and frequency contents, EMG signals are nonstationary and is extremely complex on time-frequency domain. Therefore, EMG signals cannot be analysed with classical methods, and no unique database can be set up for any person [28]. Fortunately, the generalized likelihood ratio test in [49, 50] can be utilized to build DCS which can be used to detect human movement [28, 29, 30, 31, 32]. Nonetheless, their method cannot be directly applied to our case because our EMG signals has eight channels and requires distinguishing between two similar patterns (i.e., pushing and releasing of keys).

2.8 Summary

In this work, we study on a new kind of side-channel information leakage on gesture control device in AR. By exploiting the EMG and acceleration signals available on the wearable device, we are able to recover passwords from keyboard and user login PIN on touch screen device. To succeed in the attacks, we address unique challenges in using the EMG signal. More specifically, we invent new movement detection algorithms based on DCS to reliably detect movement events of fingers. In our exploit with unsupervised learning, we avoid the assumption of labeled sensor data which makes our attack stealthy. In the same time, it is able to recover complex passwords constructed with lower case letters, upper case letters, numbers and symbols with mean success rate of 56% in the first 5000 trials. Furthermore, our exploit with supervised learning is able to achieve mean success rate 91% in the same settings. Through experiment data recorded with volunteers, we show that our exploits can be applied to users from different age and gender group. Lastly, we provide a discussion on the possible mitigation to the attack.

Chapter 3

Defending against Compromised Network Components

3.1 Privacy-preserving Radio Context Attestation in Cognitive Radio Network

With the large scale deployment of smart devices, the world has witnessed an increasing utilization of wireless communications in the last decade [61]. According to Cisco, global mobile data traffic will reach about 25 Exabytes per month by the end of 2019 [62]. Wireless communities throughout the world have recognized the shortage of spectrum for commercial broadband uses and acknowledged the urgent need for an effort to make more efficient use of the available spectrum.

One of the key technologies to improve spectral efficiency is spectrum sharing in a cognitive radio network (CRN) [63], where opportunistic access to the radio spectrum that was originally allocated to the primary users (PUs) exclusively is now allowed to be accessed by secondary users (SUs) when the spectrum is not used by the PUs. In [64], the U.S. Federal Communications Commission (FCC) has described a dynamic spectrum management framework for a Citizen Broadband Radio Service (CBRS) governed by a spectrum access system (SAS). Based on the spectrum utilization plans from PUs and radio environment maps from sensing partners such as Google, SAS manages the use of available spectrum opportunities for SUs by granting transmission permits to CRs based on their access level and location.

While spectrum sharing holds great promises, correct operations of SAS often assume honest participation. CRs need to faithfully report the sensing results and strictly follow the transmission permits issued by the SAS. Due to the dynamic reconfigurabiity, selfish users or malicious attackers can easily reconfigure their radios to gain unfair advantage or to cause harm to the network.

One way to ensure the operational correctness is via remote attestation. Remote attestation is a process of making a claim about properties of a *target* by supplying evidence to an *appraiser* or *verifier* over a network [65]. The primary objective of remote attestation is to provide verifiable evidence about the state of software executing on a system. This evidence is intended to ensure that targets will not engage in some class of misbehavior. The process of verifying the verifiable evidence on appraiser is called appraisal. Remote attestation may be used to address a number of trust problems, including guaranteed invocation of software, delivery of premium content to trusted clients, assuaging mutual suspicion between clients, and more. In the context of CRN, remote attestation provides cryptographically verifiable evidence on the state of a CR device to prove the compliance of the CR. In [66], remote attestation is used to measure the cognitive radio context as a compliance check, however, the problem of spectrum availability privacy receive little attention. In order for the verifier to assess whether the received configuration is correct or not, he will need to possess the full knowledge of legit configurations. In CRN, this configuration consists of software configuration, radio configuration and location. To make the decision on the compliance of CR, the verifier not only need to know the CR radio context but also the full spectrum information, which is often sensitive. For example, leakage of location trajectory and transmission parameters is a serious concern for PUs that are sensitive military devices [67]. With these highly sensitive information, a malicious actor can infer where a military base is located and where an army is heading towards [68]. And leakage of software configurations can also allow an adversary to make use of known vulnerabilities in a CR device [69].

Existing approaches towards protection of spectrum information privacy is to treat the SAS as a sensitive database, and secure computation techniques are used to construct privacy-preserving queries [70]. However, direct application of these techniques can lead to significant scalability issue both in the number of radio devices and the number of possible configurations. First, the number of possible configurations for each device is not a small number since a radio context configuration consists of not only the software configuration but also the location and radio transmission parameters, which leads to many possible combinations of legit radio context configuration. Second, cryptographic privacy-preserving methods often involves significant computation overhead even if the problem can be formulated as a multi-party computation problem. Furthermore, billions of radio devices are expected to be connected to the mobile network, this shear scale would require an efficient method to handle the configuration verification in CRN attestation.

In this work, we present PRIvacy-preserving Radio cOntext atteSTation in cognitivE Radio networks (PriRoster). We achieve the goal of preserving privacy of a local appraiser (LA) on an edge base station (BS) by introducing trusted hardware, i.e. Intel SGX [71]. While building a secure system on top of Intel SGX is mostly a development effort, the integration of Intel SGX to preserve privacy in CRN radio context attestation is challenged by scalability requirement and by side channels on Intel SGX.

The first challenge is scalability when integrating Intel SGX for mutual verification between CRs and BSs. For a CR device to establish trust on an edge BS before uploading the attestation report, the CR device needs to perform remote attestation on the SGX enclave inside the edge BS. However, CR devices are resource-constrained and frequently performing remote attestation on SGX enclave consumes energy and adds unacceptable computation burden on the Intel Attestation Service (IAS). Furthermore, creating independent SGX enclaves for a large amount of CR devices introduces a large computation load on the edge BSs. In PriRoster, CR devices delegate the power-consuming attestation on SGX enclaves to the more powerful SAS server and only one enclave is needed on each edge BS for conducting local appraisals.

The second challenge is the privacy leakage from memory access side channel on Intel SGX. Memory access side channel is a known vulnerability on Intel SGX [72, 73, 74]. A privileged software can observe the memory access pattern of an enclave to extract sensitive information. In our case, an edge BS can infer the radio context of CR devices from their memory access patterns which are observable by the edge BS. In PriRoster, we design oblivious appraisal functions for preventing memory access pattern leakage.

To summarize, our contributions are:

- We propose PriRoster, a privacy-preserving radio context attestation technique that allows a untrust verifier to carry out remote attestation of a CR device's context without knowing the device's context information itself. This technique can effectively conceal the operational parameters of the PUs' as well as the CR devices' from untrusted network components such as an intermediary edge BS.
- We consider a systematic network-wide large-scale remote attestation which allows a large number of remote devices be attested simultaneously and efficiently. We propose a novel trust transfer mechanism to address the scalability problem raised in this scenario. Individual devices can rely on the attestation result done by an trusted entity rather than each carrying out a separate attestation process.
- To address the memory side channel limitation of Intel SGX, We design an oblivious appraisal function that effectively prevents leakage of sensitive PU information through memory access at the edge BS.
- We build a prototype system of PriRoster using USRP, Raspberry Pi, Intel NUC, and Amazon AWS. The prototype system shows the feasibility of the PriRoster framework.

3.2 Background

3.2.1 Spectrum Sharing in CRN

To tackle the problem of spectrum scarcity, spectrum sharing is proposed to allow new entrants to utilize the radio spectrum allocated to incumbents when the spectrum is not in use.

The spectrum sharing solutions can be divided into two categories: decentralized and centralized. Decentralized solutions are not reliable because of sensing challenges such as hidden node problem. The centralized dynamic spectrum management framework is increasingly attracting more attention. FCC has proposed a centralized dynamic spectrum management framework for CBRS governed by SAS. It is a three-tiered spectrum authorization framework accommodating a variety of commercial uses on a shared basis with incumbent federal and non-federal users of the 3.5 GHz band. The three tiers are: Incumbent Access(IA), Priority Access (PA), and General Authorized Access (GAA) [75]. IA has the highest priority while GAA has the lowest. The CR devices in this paper refer to the devices at PA or GAA level.

The SAS is capable of dynamic frequency assignment and interference management[76]. The core of the SAS is a database system which receives feedings from incumbent users regarding spectrum usage information, such as usage duration and operational parameters. Operational parameters include primary user identity, location, transmission power, antenna parameters, and interference tolerance. With the spectrum usage information provided, the SAS determines the available frequency within an area at a time slot and assign them to nearby CRs and determines the maximum transmission power [76, 75]. Meanwhile, SAS is responsible for detecting and removing CRs that do not obey its assignment.

3.2.2 Radio Context Attestation in CRN

The security of the SAS system involves the protection of the SAS databases and functions at the servers and the confidentiality and integrity protection of the operational CR devices in the field. In [66], we proposed a remote attestation framework for CRNs that aims to ensure the operational integrity of the CR devices by remote radio context attestation. As shown in Fig. 3.1, there are three major entities in the architecture- SAS, Regulatory Authority (RA) and Local Appraiser (LA). RA is a regulatory entity like FCC and LA denotes a local appraiser typically hosted on an edge base station. RA informs SAS to start attestation tasks by sending it an attestation token. Upon receiving the token, SAS delegates its appraisal tasks to LA and LA performs local appraisal of attestation reports from radio devices. In that architecture, both RA and LAs are trusted entities in the network. However, since edge base stations do not have same security level as SAS and is more likely to be compromised, the sensitive information is not safe kept on LA. Thus, in this paper, we consider the protection of sensitive information released to LAs and we integrate trusted hardware to mitigate information leakage from LAs.

3.2.3 Intel SGX

Intel SGX is Intel's latest instruction extensions that allows processes to shield part of their address space from privileged software such as operating system and hypervisor. Processes on SGX-capable platform can construct trusted execution environments called enclaves. In-

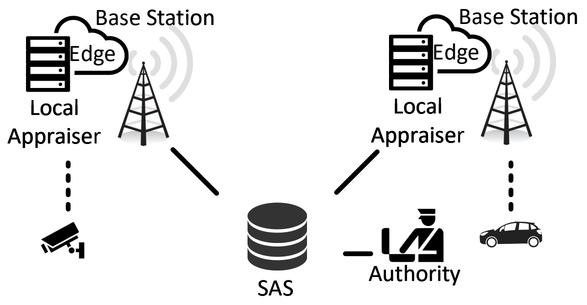


Figure 3.1: Radio Context Attestation in CRN.

tegrity and confidentiality guarantees are provided to security-sensitive computation conducted inside the enclaves. Intel SGX also provides remote attestation and provision, which allows a remote party like a SAS server to verify an application enclave's identity and securely provision keys, credentials, and other sensitive data to the enclave on an untrust host, such as an edge BS.

Despite the new security capabilities brought by Intel SGX, there are some known security limitations in modern Intel processors. Although Intel's autonomous memory encryption engine (MEE) encrypts data in DRAM, if an attacker sniffs the address bus physically, he or she can observe a cache line-granularity side channel, which has been confirmed at both page [73] and cache line level [72]. We integrate oblivious function to mitigate this leakage.

3.3 PriRoster System Model and Assumptions

System Goals

PriRoster is designed to take a network-wide attestation of CR devices. The aggregated attestation report, if successfully verified, is a cryptographical proof of the compliance of all the CR nodes to the spatial-temporal sensitive radio policy. During this process, the radio context of individual CRs should not be accessible by BSs, and neither should the BSs learn the full details of the PU's operational parameters.

Threat Model

For CR devices, we assume attackers can gain control of a CR device by conducting software attacks. They can thus modify radio related parameters like transmission power, modulation method and more. Attackers can also fabricate network packets coming out of the controlled device. We do not consider hardware attacks. For edge BSs, we assume there could be an malicious actor like a malicious insider or a remote attacker controlling its computing platform. The malicious actor can intercept or fabricate information in and out the edge BS via its network interface. We assume an adversary can use privileged software to observe fine-grained memory trace.

Assumptions

We assume CRs are equipped with trusted hardware components like widely available ARM TrustZone [77]. We assume CRs' software stack contains normal world and secure world. And the integrity of secure world software is guaranteed by secure boot. We assume certificates of SAS and RA are preloaded to the secure world of CR nodes and certificates of both RA and CR nodes are available to SAS. We assume remote attestation report generation is sitting inside trusted hardware and software attack cannot reveal or modify the process. We assume CR devices are powerful enough to perform asymmetric cryptograhic primitives. For edge BSs, we assume they are equipped with Intel SGX [71]. We assume edge BSs can control the privileged software like hypervisor and operating system but cannot modify hardware.

3.4 PriRoster Framework

PriRoster is a network-wide radio context attestation framework that allows secure and scalable verification of operational integrity for a large number of CR devices in a spectrum sharing network. In order to keep the framework scalable, radio context appraisal of CR nodes is delegated to edge BSs while only aggregated attestation results are sent back to SAS. However, radio context (location, spectrum usage, power level, operating time and software configuration) of CR node contains sensitive information. Thus, local appraisal should not leak actual radio context on CR nodes to edge BS. Besides, SAS compliance rules used in local appraisal needs protection since this information can be used to infer sensitive information of primary users like location of military radios. Therefore, in our design, we target at preventing both CR's radio context and compliance rules in local appraisal from being leaked to edge BS. To achieve this goal, are three major challenges:

- Conducting local appraisal at untrusted edge nodes may leak sensitive information including radio context and compliance rules. To provide privacy-preserving radio context attestation, we implement LA's functionalities in an enclave on the edge BS. This process is detailed in Sec. 3.4.1.
- To scale up, multiple devices with same service request are assigned to share one enclave at a BS. However, remote attestation of the LA enclave needs to be conducted by each CR device to establish the trust on the LA enclave by the CR devices. This leads to non-negligible energy consumption at each CR and a tremendous amount of attestation burden on IAS server. We propose a trust transfer design which delegate the task of

remote attestation of LA enclave from CRs to SAS thus minimize the number of remote attestations that need to be done in Sec. 3.4.2.

- Intel SGX provides confidentiality and integrity for enclave programs, however, there are known security limitations of Intel SGX itself. For example, although privileged software cannot access enclave memory, it can be used to observe memory access pattern [73]. Therefore, an attacker controlling privileged software can potentially disclose sensitive information such as software configuration of CR. To mitigate this kind of side channel attack, we realize oblivious software configuration appraisal by designing oblivious function in Sec. 3.4.3.

3.4.1 Privacy-Preserving Single Device Attestation

In this section, we present privacy-preserving remote attestation of radio context on a single device. We take advantage of trusted hardware (i.e. Intel SGX enclave) for defending against compromised edge BS. From a high level view, SAS distributes radio context attestation request and LA enclaves conduct the radio context attestation on behalf of SAS. Before delegating radio context attestation task to LA enclave, SAS needs to assess the trustworthiness of LA enclave's execution environment by performing remote attestation on it. Similarly, CR node needs to first verify the trustworthiness of LA enclave before accepting the attestation request from it. Then CR sends its radio context report to correctly verified LA enclave with confidence that both the integrity and confidentiality are guaranteed. In the end, LA enclave sends local appraisal results to SAS and single device attestation is completed.

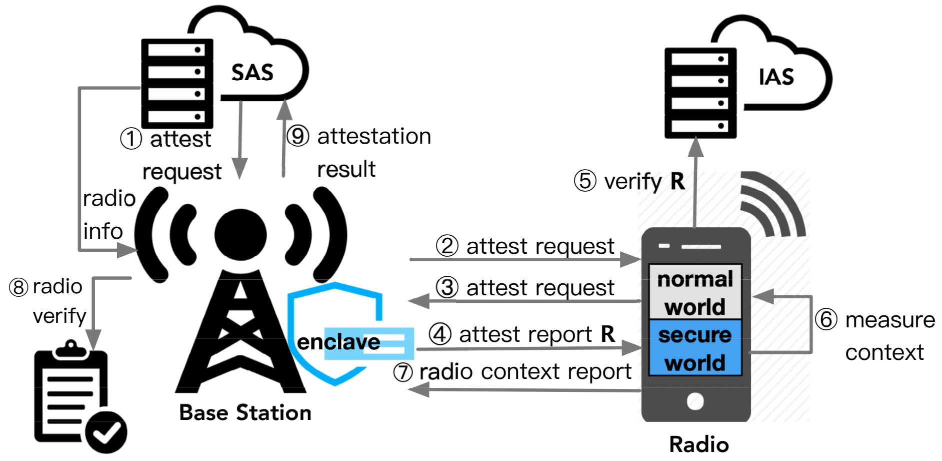


Figure 3.2: Privacy-Preserving Device Attestation

As shown in Fig. 3.2, authority initiates a radio context attestation. SAS pushes a remote attestation request to BSs in step ① . Each BS forwards the request to the CRs within its range in step ② . Upon receiving the request, a CR in turn requests to attest the execution environment of the LA enclave running on the BS in step ③ . LA enclave replies with its enclave attestation report R to the CR node in step ④ . With the help of IAS, the CR assesses enclave's trustworthiness in step ⑤ . Only if a positive verification response from IAS is received, will the CR start radio context measurement in step ⑥ . Then the attestation report is sent to LA in step ⑦ . With the information regarding compliance rules (radio assignment information and correct software configuration) received from SAS, LA enclave conducts radio context verification for the CR in step ⑧ . In the end, LA enclave sends back attestation result to SAS in step ⑨ .

The detail of radio context attestation protocol is outlined in Fig. 3.3, describing a successful protocol run. Note that we assume SAS and CR nodes know the public key of RA, and RA and CRs also know the public key of SAS, as described in our assumptions in Sec. 3.3. In addition, SAS has to set up a LA enclave on each untrusted edge node involved with the help of IAS before delegating radio context attestation task to it. After successfully setting up the LA enclave, an unique attestation key used to produce signature will be burned into each newly established LA enclave. SAS conducts authentication on LA enclave by verifying LA enclave's signature against an endorsement certificate created by manufacturer Intel. A secure channel between SAS and LA enclave will be established after LA enclave is successfully set up. Acronyms and parameters definition are shown in Table. 3.1.

Table 3.1: Acronyms & Parameter Definition

RA	Regulatory authority
SAS	Spectrum access system
LA	Local appraiser
IAS	Intel Attestation Service
k_{ij}	Shared secret key between CR d_i and local enclave appraiser E_j
k_{AE_j}	Shared key between global appraiser and base station enclave E_j
$\hat{S}_i$	Measured software configuration of d_i
$\hat{f}_i$	Measured frequency band used by d_i
$\hat{p}_i$	Measured power level of d_i
$\hat{L}_i$	Location measurement of d_i
$Conf$	Correct software configuration at SAS
τ	Attestation token from RA
N_A	Nonce generated by RA for attestation
d_i	Identification of CR device i
MAC_{ij}	MAC generated by d_i using key k_{ij}

Steps ① and ② show the propogation of radio context attestation request from SAS to CR devices. After mutual authentication with RA, SAS obtains a valid token τ from RA. SAS

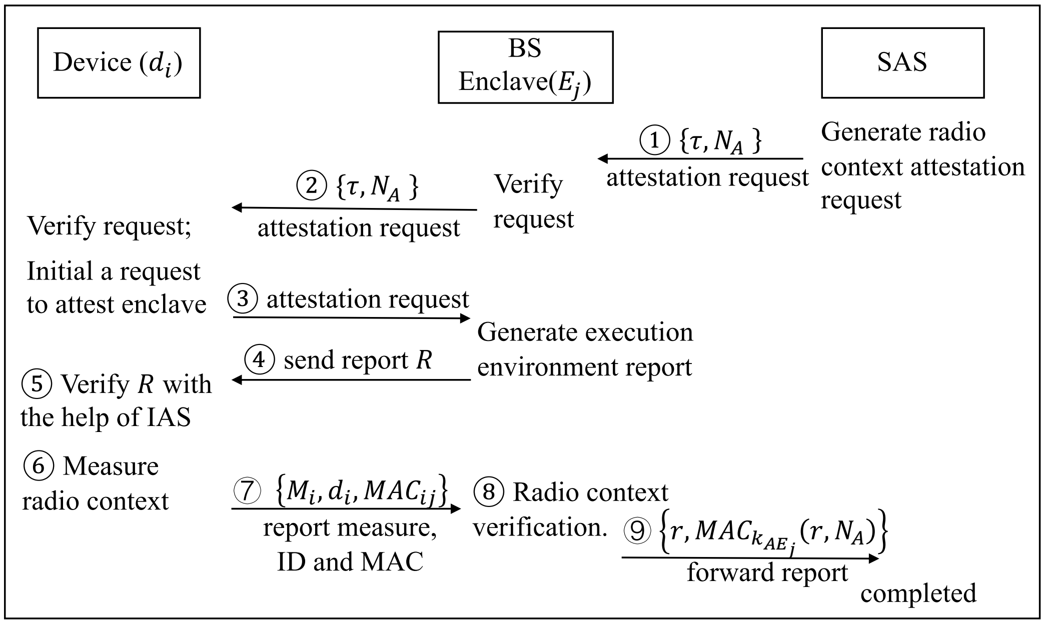

Device (d_i)
BS Enclave(E_j)
SAS
① $\{\tau, N_A\}$ attestation request
Generate radio context attestation request
② $\{\tau, N_A\}$ attestation request
Verify request
Verify request;
Initial a request to attest enclave
③ attestation request
Generate execution environment report
④ send report R
⑤ Verify R with the help of IAS
⑥ Measure radio context
⑦ $\{M_i, d_i, MAC_{ij}\}$ report measure, ID and MAC
⑧ Radio context verification.
⑨ $\{r, MAC_{k_{AE_j}}(r, N_A)\}$ forward report
completed

Figure 3.3: The radio context attestation protocol

sends the attestation request consisting of τ and a nonce N_A to local enclaves. Nonce N_A is used to resist the replay attack and to associate an attestation request with the corresponding attestation report. It can prevent an adversary from reusing old attestation requests, thus stopping potential DoS attacks where an adversary spams attestation requests on the network.

Steps ③ to ⑤ describe attestation of the LA enclave. Upon receiving a radio context attestation request, the CR node verifies the token generated by RA and check the included nonce N_A to ensure the freshness of this request. If the request is verified correctly, CR node initializes a request to attest the execution environment of LA enclave. This verification is done with the help of IAS, and detail of SGX enclave attestation can be found in [78].

Steps ⑥ to ⑨ are the radio context measurement and report process. Radio context M_i is measured by the attestation routine inside ARM TrustZone of CR device. A CR device i then generates the response $\{M_i, d_i, MAC_{ij}\}$, where $MAC_{ij} = MAC(M_i, d_i, N_A)$, using the shared secret key k_{ij} between CR device i and LA enclave j. MAC value is used to ensure both source and content integrity of the report. M_i, the radio context, contains four parts, $\{\hat{S}_i, \hat{f}_i, \hat{p}_i, \hat{L}_i\}$, which will be explained in step ⑧ , verification of radio context, as follows.

The software configuration $\hat{S}_i$ generated by hashing the memory pages is verified by checking against a set of known benign device software configurations received from SAS. If $\hat{S}_i$ is not on the list, then it is likely that the CR platform software stack is compromised. However, there is no known list of compliant radio configurations due to dynamic spectrum availability.

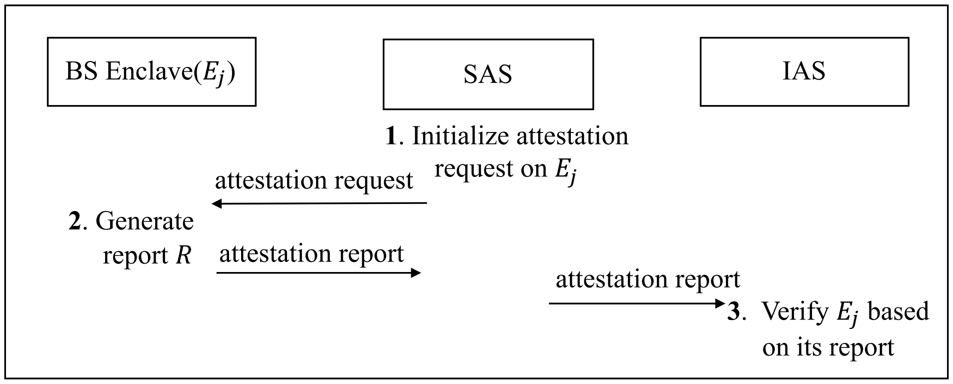


Figure 3.4: Trust Establishment of SAS on SGX enclave

To verify the radio configuration, LA enclave first verifies if the used channel $\hat{f}_i$ reported by CR is the same as what is assigned by SAS. Then the power level $\hat{p}_i$ is compared with the maximum power allowed by SAS. In conclusion, CRs are audited by LA enclave to ensure that they do not exceed the maximum transmission power at given location on assigned channel by SAS. In the end, LA sends the attestation result r of a CR with corresponding $MAC(r, N_A)$ to SAS in Step ⑨ .

3.4.2 Privacy-Preserving Multiple Devices Attestation

Running single device attestation described in Sec. 3.4.1 can satisfy the security requirement but it is not scalable. If one has to set up an LA enclave for each CR device, a large number of enclaves will have to be established which is a big burden for the host. In our PriRoster design, only one LA enclave is established at the edge BS node, and this one LA enclave will serve multiple CRs associated to this BS.

Another scalability concern is that, by the naive design, each CR device needs to carry out a remote attestation on the LA enclave it associates before it sends radio context report to the enclave. This would be duplicated efforts if multiple CRs are connected to a same LA enclave. Considering that a remote attestation is a much more expensive process comparing to a cryptographic authentication, in our PriRoster design, we release the resource-constrained CR deviced from the burden of carrying out the remote attestation of the LA enclave. Instead, we delegate the attestation of the LA enclave to the more resourceful SAS and transfer the trust established on the LA enclave by SAS to each individual CRs through an authentication protocol.

The task delegation is a two-step process: Trust Establishment and Trust Transfer.

Trust Establishment: Fig. 3.4 shows the trust establishment on LA enclave by SAS through

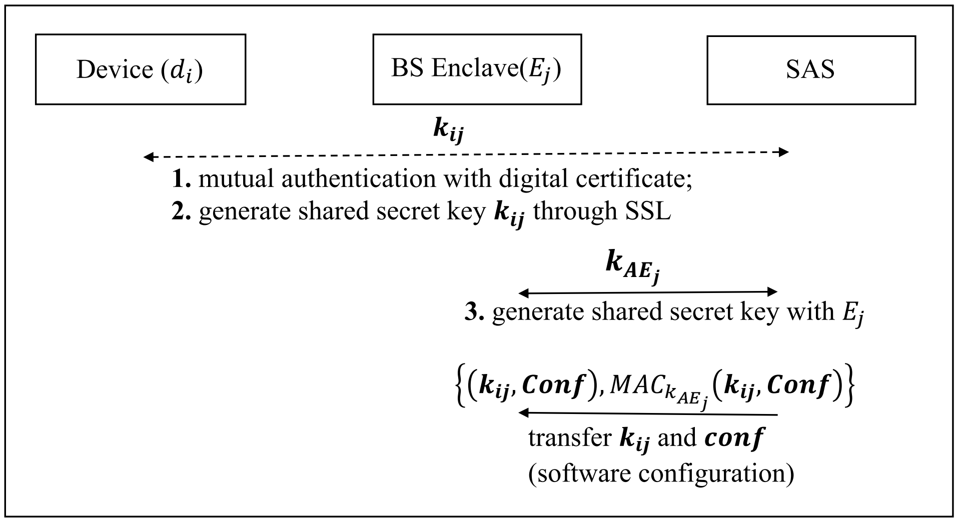

Device (d_i)
BS Enclave(E_j)
SAS
k_{ij}
1. mutual authentication with digital certificate;
2. generate shared secret key k_{ij} through SSL
k_{AE_j}
3. generate shared secret key with E_j
$\{(k_{ij}, Conf), MAC_{k_{AE_j}}(k_{ij}, Conf)\}$
transfer k_{ij} and $conf$
(software configuration)

Figure 3.5: Trust transfer procedure by transferring symmetric key.

conducting remote attestation on enclaves. SAS first initializes a remote attestation request on enclave E_j to assess the execution environment trustworthiness of local enclave. Local enclave E_j generates a report and sends it back to SAS. Once the attestation result is verified correctly by SAS with the help of IAS, SAS's trust on LA enclave will be established.

Trust Transfer: Following trust establishment, SAS can transfer its trust on a LA enclave to individual CRs assocaiated with that LA, through authentication protocol. We propose two implementations of trust transfer: i) Symmetric Key Transfer, ii) Public Key Certificate Distribution. Essentially, the task of attesting the trustworthiness of enclave is delegated to SAS.

i) Symmetric Key Transfer: Trust transfer through transferring symmetric key is outlined in Fig. 3.5. Both SAS and CR devices have their own public keys so mutual authentication can be done between SAS and any CR i, and a shared secret key k_{ij} can be generated securely during this process, where j denotes the BS that the CR is associated with. SAS then securely transmits this shared secret key k_{ij} to LA enclave j. For a CR device, the keys are stored in its trusted hardware and cryptographic computations are performed in its secure world. Instead of carrying out a remote attestation on LA j, CR i now relies on authentication of LA j based on the shared secret k_{ij} in order to gain trust on LA enclave j.

ii) Public Key Certificate Distribution Alternatively, SAS can issue a certificate with an expiration time to an LA enclave once a successful attestation is done. LA enclave sends both the attestation request and its signed certificate to the CRs to start radio context attestation at each individual CR device. CRs establish trust on LA enclave by verifying the received certificate. In the end, CRs send back the radio context report to trusted LA.

However, this method requires constant certificate verification on the CR side. And this is not suitable for defending against hardware attack. Thus, we choose the symmetric key transfer scheme.

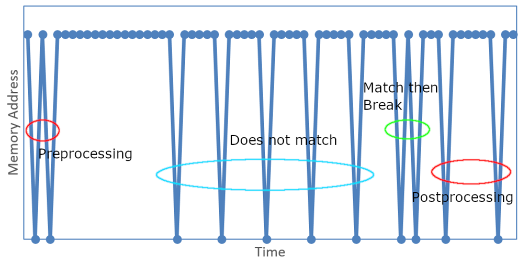
Note that authentication and attestation establish different levels of trust. Crypto authentication protocols only verify the keying material. As long as the party being authenticated demonstrates the knowledge of the secret keying material, the trust is established. However, enclave attestation verifies not only the keys, but also the code and data integrity inside the enclave. Authentication can only ensure that the party holds the right key, while attestation can also ensure the operational integrity of the party. Therefore, the trust transfer is not at the same trust level. The transfer would remain at the same level if the following assumption holds: no successful attack to the enclave between the SAS attestation and the CR authentication. We made this assumption as it is very likely to be true and the delegation of attestation tasks allows significant computation savings in the overall system.

3.4.3 Defense Against Side Channel Attack

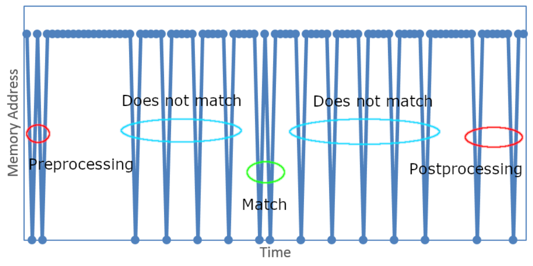
One of the primary tasks in software configuration appraisal is the verification of the cryptographic hash of the system memory that captures the software configuration. If the hash checksum does not match any of the known good configurations, then the device is considered compromised. However, if a matched is found before reaching the end of lists of legitimate configurations, the function returns without doing further comparisons. However, such early termination of comparison leaks side channel information allowing the attacker to extract the software configurations of the target under attestation. We perform a experiment to demonstrate the side channel information leakage of this design in Fig. 3.6(a).

While an enforced full traversal design would solve the early termination of hash comparison, the attacker can also exploit memory access pattern on the preparation of the result network packet. More specifically, he can observe if the attestation pass or fail based on if the real device id memory is loaded or the stub id memory is loaded. We show evaluation of this information leakage in Fig. 3.6(b). In comparison, with integration of the following oblivious function, we design an oblivious appraisal process whose memory trajectory is shown in Fig. 3.6(c). The detail design is discussed in Sec. 3.7.2.

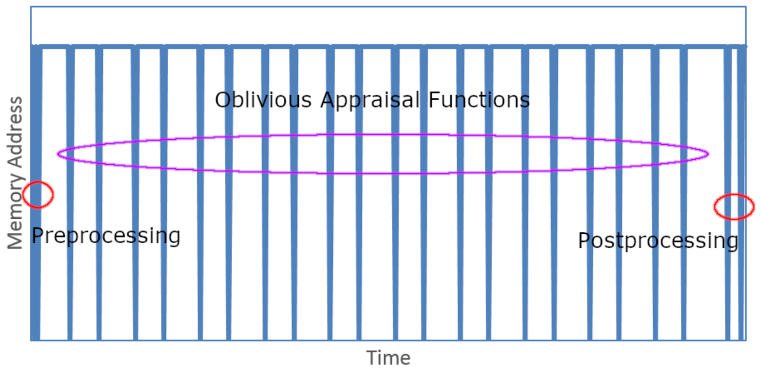
To mitigate this information leakage, we implemented an oblivious software configuration appraisal by designing oblivious function with X86 cmovz instruction. X86 cmovz instruction moves source operand to destination operand if condition code is true. When both source and destination operands are put in registers, this data transfer turns out to be oblivious and leaks no information about the branch selection. Our design is similar to [74, 79, 80]. An OCompare() function is used to hide the trace of software configuration comparison by using cmovz instruction. This function takes in input including hash of two software configurations and return the device id only if the two hashes match. Note that, the hashes here are trimmed to fit in register. The authors consider trimmed hash is robust enough for



(a) Memory Access Pattern of Naive Appraisal Process.



(b) Memory Access Pattern of Appraisal Process with Full Traversal Design.



(c) Memory Access Pattern of Oblivious Appraisal Process.

Figure 3.6: Memory Access Pattern of Native Appraisal Process (a), Full Traversal Design (b), and Oblivious Appraisal (c).

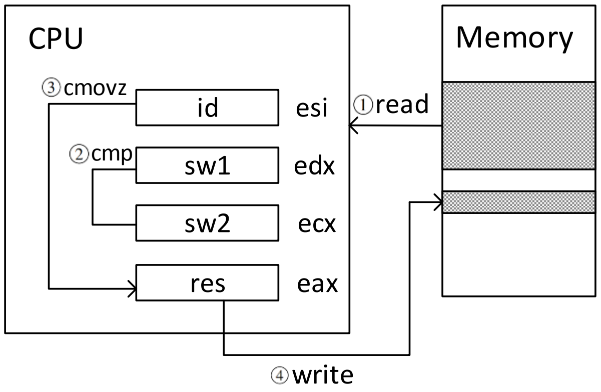


Figure 3.7: OCompare() function diagram.

current circumstance. If the two configurations mismatch, this function does not change the return buffer for result. The function has four main steps, 1) both values are loaded into register, 2) the cmp instruction compares received hash of software states and update Zero Flag (ZF) in EFLAGS register to reflect the comparison, 3) the cmovz instruction copies id into the destination register according to ZF, 4) the test instruction resets EFLAGS register by comparing known values. Fig. 3.7 shows the process. OCompare() presents the same memory access pattern since the operation is done all within registers. Therefore, an attacker can not distinguish from memory traces which software configuration is selected.

3.5 Security Analysis of PriRoster Local Appraisal

In this section, we analyze the security of PriRoster local appraisal process in terms of radio context, compliance rules and memory oblivious function.

Confidentiality of the Radio Contexts and Spectrum One of the primary security goals of PrivacyScopeis to ensure the confidentiality of configurations of the prover (CR) from verifier (BS). There are two aspects of confidentiality in the attestation process, the confidentiality of individual provers (CRs) and the set of legal configurations derived from the sensitive spectrum information. The individual prover's configurations are protected via either remote attestation or trust verification in the transfer process. More precisely, with remote attestation, the CR can verify not only the identity but also the configuration of the system that processes his submitted information. As a result, the information is protected by the TEE in BS. Through the trust transfer process, individual CRs leverage verification of authentication token to alleviate the process of the remote attestation to the trust on authority in that he has performed the attestation and have verify the environment appropriately. For the spectrum availability, since all the information are processed within the TEE and is only used to perform attestation, its protections will be based on the security guarantee of the TEE.

Defense against Side Channel We define a program's interaction with memory as a trace execution τ which records the access type (read or write) and address of some contents. We express our proof using a simulation-based technique: for each run of a software configuration comparison procedure that yields a trace τ, we show that there exists a simulator program, whose software configuration under comparison is different from the original comparison procedure, that simulates the interaction of the original comparison procedure with memory by producing a trace τ' indistinguishable from τ. More precisely, we define indistinguishability similar to semantic security in cryptography using a game between a system that runs the comparison procedure (or the simulator) and a computationally bounded adversary that interacts with the system to observe the trace and attempts to guess whether it interacts with the original procedure or the simulator. The comparison procedure is secure when such adversaries guess correctly with probability at most $\frac{1}{2}$ plus a negligible advantage.

To ensure security of comparison procedure, we first need to evaluate the OCompare() function in Fig. 3.7. Since the code operates on the processor registers only and never accesses memory, it operates within the (trusted) boundary of the sealed processor chip. As such, evaluations that involve registers only are not recorded in the trace τ, hence, we consider any register-to-register data manipulation secure. As such, we evaluate full traversal design with OCompare() function. Since we use a full traversal design, different software configuration input will all go through all the OCompare() functions. Simulation of the program with a different software configuration as input cannot be differentiated from original trace τ by the adversary.

3.6 Implementation of PriRoster Prototype

For CR device prototype hardware setting, we select Raspberry Pi 3 as application processor and USRP N210 as baseband processor. USRP N210 has been one of the standard radio platform for CR research. For CR device software setting, we apply TrustZone to build a trusted environement for the attestation software. To be specific, we use OPTEE secure kernel [81] in the secure world and build a OPTEE Static Trusted App called ATTEST with approximately 1000 software line of code (SLOC) to serve as attestation software. We use Ubuntu 15.04 with 4.6.3 ARM 64 bit Linaro Linux kernel in normal world. The radio core device driver libUHD is the software for controlling USRP N210. It sits in the normal world and is loaded in an address known to ATTEST at runtime. The radio parameters used by LibUHD are saved as global variables in a specific memory location known to ATTEST. Upon receiving a valid remote attestation request, ATTEST will perform SHA256 checksum of the linear memory map of libUHD and code page of Operating System kernel and embed the hash result with retrieved radio parameters inside the attestation report. We refactor openSSL 1.0.1f library for cryptographic operations and secure communication.

For edge BS, we choose Intel NUC which supports Intel SGX natively. The NUC is powered by Intel i7-6770HQ Skylake CPU with 6MB cache at 2.6 GHz and 8GB DRAM. We use ubuntu 16.04 and the local appraisal enclave is built with Intel SGX SDK v2.4. For SAS, we choose AWS EC2 instance with 64 bit Ubuntu Server 18.04 LTS. According to lshw, it is using Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz and 983MiB system memory.

We implement remote attestation between CR node and LA enclave on Intel NUC, Raspberry Pi and remote attestation between SAS and LA enclave on Intel NUC, AWS cloud. We register our self-signed certificate with Intel SGX remote attestation service and retrieve SPID from Intel by contacting Intel customer support. We store the private key for the self-signed certificate inside secure world of Raspberry Pi and on AWS cloud.

3.7 Evaluation on PriRoster System

Our evaluation of the proposed system focus on two main aspect - scalability of in large radio network context attestation, and the ability to protect confidential configuration information against side channel leakage of the TEE during the verification process.

3.7.1 Prototype Comparison

In order to effectively compare three designs, we individually benchmark the primitives used in the protocols. To be specific, we benchmark instantiating remote attestation on CR node, instantiating remote attestation on AWS cloud, trust establishment and trust transfer process of PriRoster.

Primitives Benchmarks

We measure the time for a single CR device to perform a successful remote attestation on LA enclave from connection establishment with IAS server to disconnection. It turns out the average time needed is 366.45ms for this remote attestation. We also use a AVHzY USB Power Meter Tester to supply power for Raspberry pi and collect measurement of consumed power. The collected power consumption for performing a successful remote attestation on LA enclave for a single CR device is 0.28J on average. On the other hand, we measure the time for SAS to perform a successful remote attestation on LA enclave. The average time for this remote attestation is 32.7ms. We implement trust establishment and trust transfer process on Raspberry Pi and AWS cloud instance using Linux socket. We evaluate the process and the outcome shows that this process takes 2.57ms on average. And the energy consumed on CR device for trust transfer is on average 0.003J. Table. 3.2 summarizes the benchmark results for primitives.

Table 3.2: Primitive Benchmark

HW	Function	Time(ms)	Energy(J)
Pi	Remote attestation	366.45	0.28
Pi	Trust Transfer	2.57	0.003
AWS	Remote attestation	32.7	-

Table 3.3: Design Benchmark Comparison

Design	Pi Energy	IAS Time	SAS Time	Single BS Time
Single device design	37.33 kWh	6.56 years	181 days	10.80 mintutes
Single enclave design	37.33 kWh	5.57 years	2.9 hours	9.16 minutes
PriRoster	0.4 kWh	5.81 hours	$p * 14.4$ days	3.92 seconds

p is the percentage of CRs that join a new BS per unit time

Design Benchmark Comparison

We focus on computation overhead and energy consumed brought by difference between the three designs of prototypes. Thus, we skip overlapped processes like radio context attestation report generation on CR nodes in these designs. For simplicity of demonstration, we assume that in real life setting, there are 1,500 CR devices connected to one edge BS and there exists 320,000 edge BS in the U.S. [66]. IAS server is assumed to serve clients one by one. We assume IAS time is composed of AWS time and Pi time, since IAS participates SGX enclave attestation in both cases.

We first present the design of every CR device conducting its own remote attestation on LA enclave to establish trust. In this single device design, there are 1,500 independent enclaves existing on each edge BS, and enclaves are created or destroyed with CR's joining and leaving BS. Therefore, CRs need to attest LA enclaves per radio context attestation request. For simplicity of comparison, we assume all CR nodes are static for now. LA enclave attestation consumes 37.33 kWh for all CR devices under all BSs. SAS need to perform 960,000,000 times of remote attestation which takes 363 days for a single cloud instance. Task at a single BS including enclave attestation by CRs, SAS takes around 11 minutes. And the overall processing time for IAS is 6.56 years of single machine time.

In the single enclave design, only one LA enclave is created on a BS for 1,500 CRs. Thus, SAS only needs to perform one time of remote attestation on this enclave respectively. But all CRs still need to attest enclaves. So altogether the attestation time for single BS is around 9 minutes. Similar to single device design, CRs need to attest LA enclave per radio context attestation request. SAS needs to perform 320,000 times of remote attestation, which takes 2.91 hours for a single cloud instance. The overall processing time for IAS is 5.57 years of single machine time.

In PriRoster, each CR device does not need to remote attest LA enclave but it needs to perform trust establishment and trust transfer process the first time it joins in a network. SAS only needs one attestation on this enclave respectively. Enclave attestation (by SAS) together with trust transfer at a single BS takes around 3.92s. The trust establishment and trust transfer process of all CRs at SAS takes 14.28 days and cost 0.4 kWh for a single cloud instance. Note that, the trust establishment and trust transfer process only takes place at CRs's joining time, so the runtime burden for SAS will be much lighter. The overall processing time for IAS server is 5.81 hours of single machine time. Note that we can easily establish multiple cloud instances and use multiprocessing for bootstrapping the attestation time. Suppose we have 16 threads on one server, this process only takes 20min. Table. II summarizes the benchmark results for design comparisons.

3.7.2 Oblivious Appraisal Process

We show the effectiveness of oblivious appraisal function in this section. We use dynamic instrumentation tool, Intel Pin Tool 3.0 [82], for tracing memory access pattern.

We choose full traversal design to protect against side channels brought by early termination design. In addition, to hide memory access trace, we apply oblivious compare function OCompare(). For every comparison, we use OCompare() to replace previous comparison function. At the end of the comparison procedure, device id is saved in result buffer if a match is found or else a stub value will be saved in result buffer. Fig. 3.6(c) shows the oblivious appraisal process and for all matches, the memory traces stay the same. As in Fig. 3.6(c), we can see that an attacker cannot infer which software configuration is matched since all comparisons' memory trace appear to be the same.

3.8 Related Work

Although PriRoster is the first work to provide privacy-preserving radio context attestation, there has been closely related works on remote attestation, CRN security and side channels in trusted execution environment.

Remote attestation of software on a prover for a single appraiser is well studied. The prover is the device under attested and it sends a status report of its current execution state to an appraiser. Since malicious software on the prover could potentially forge the report, various methods have been proposed to promise the trustworthiness of the report. For example, [83, 84, 85, 86, 87, 88] put secure hardware in use and [89, 90, 91, 92, 93] take advantage of trusted software. Recent interest arises on malicious actors with hardware attack capabilities also. [94, 95] take a first step to use remote attestation for protecting against hardware attacks. Besides attestation of one prover to one appraiser, [96, 97] propose swarm attestation for integrity of a group of devices. In this work, we consider remote attestation under a centralized edge computing architecture using secure hardware.

For CRN security, [98, 99, 100, 101] propose authentication of CR device with signal at the physical layer and [102, 103] propose detecting and preventing malicious CR at device level. Although authentication can verify the identity of a CR device and device level security protects a CR device from being compromised, they cannot ensure authority that every connected CR device is benign and complies to transmission permissions at runtime in our case. To ensure authority the operational integrity of the CR devices and provide insights for authority to verify their compliances, [66] comes up with remote attestation of radio context. Despite [66] provides operational integrity of CRN, the potential privacy leakage inside edge BS of the network is not considered.

Side channel information leakage on trusted system remains an active area of research [73, 104, 74, 79, 80, 105, 106, 107, 108]. [73] proposed page-fault side-channel attacks on SGX, where an attacker controlling priviledged software could extract secrets from enclave execution by tracking memory access patterns at the granularity of memory pages. [109] demonstrates another attack approach by using branch shadowing to infer the control flow of the execution inside an enclave. Branch shadowing requires frequently interrupting the victim enclave and this observation enables effective detection methods [104, 107]. [74, 79, 80] research on information leakage of search index through memory access pattern. [105] proposes a generic path ORAM [106] enclave for hiding memory traces. In PriRoser, we put memory access pattern side channel under consideration and design OCompare() function for preventing information disclosure of this type.

3.9 Summary

In this paper, we propose PriRoster, a privacy-preserving radio context attestation framework for CRN. PriRoster integrates trusted hardware, Intel SGX, to prevent information leakage at edge BS. Our system has two key innovations. To solve the scalability challenge in remote attestation of a large network, we design a novel trust transfer protocol to allow an effective trade-off between security guarantee and scalability. To address the side-channel information leakage at the TEE, we design an input oblivious algorithm to enable radio context verification without leaking memory access information. A prototype of PriRoster is implemented to demonstrate the feasibility of the system in terms of computation, energy overhead, as well as the memory access pattern.

Chapter 4

Hardening Trusted Hardware Ecosystem

4.1 Detecting Information Leakage in Intel SGX Enclaves

With more and more data being collected and analyzed, there is an increasing concern on privacy implication of the sensitivity of information on individuals. While we are enjoying such rapid advancement in data science, many consider this a step backwards on the fundamental civil right to privacy. In an effort to tackle this fundamental tradeoff between data utility and data privacy, much work has been done to enable secure computation on confidential data, where only the results are revealed but not the original data. Secure computation techniques are generally divided into two categories, cryptographic techniques [110] and system mechanisms [12, 11, 111]. Trusted Execution Environment (TEE) is one of the more popular system methods designed to ensure secure computation on private data given its ability to host arbitrary computation with limited overhead.

However, even though techniques leveraging TEE aim at providing privacy assurance to users, the security protection of system actually depends on both the TEE and the TEE-protected applications themselves. For example, while the Intel SGX architecture can guarantee the integrity and confidentiality of execution, it does not address leakage of private data due to program code vulnerabilities or intentionally injected backdoors within SGX-protected program. The code executing in a SGX enclave can inadvertently or maliciously leak private data outside its trust boundary. Majority of recent research [73, 112, 109] focus on addressing potential data leakage on TEE, while few works [113] focus on private data leakage by TEE-protected applications themselves.

Since TEE-protected applications may contain malicious logic embedded by attacker or data leakage bugs brought by programmers, it is important for users of these secure applications to audit and validate them. However, ensuring trustworthiness of TEE-protected applications manually requires security expertise and is not scalable for upcoming large amount of TEE-protected applications. Therefore, an automatic verification tool for users to detect leakage in

TEE-protected applications is desired. [113] formally specifies semantics of TEE-protected application and applies classical information flow analysis [15] to automatically detect leakage on it. The classical information flow analysis applies the noninterference property that essentially ensures no mutation of sensitive data can lead to observable changes in program state from the perspective of an outside observer. However, the noninterference property is not suitable for widely adopted ML algorithms in the era of IoT and cloud computing. ML algorithms use private data to train models which are observable for cloud service provider. Thus, ML programs always violate classical noninterference property. Therefore, a new property is desired for defining information leakage in ML programs.

Inspired by the noninterference property, we formally define nonreversibility property of enclave application in this paper. Violations of nonreversibility property implies malicious actor can infer sensitive input by observing the output generated by program code running in an enclave. Thus, nonreversibility is applicable to analyzing data leakage issues in widely adopted ML code modules. Detecting violation of nonreversibility is challenging, there are fundamental challenges that are different from noninterference. First, it is not trivial to determine if the recovery from output to input is deterministic or not. Second, even if there is a data flow from sensitive input to output, it is not entirely clear what the exact relationship is between input and output, which is crucial in determining recoverability. To tackle these challenges, we design and develop PrivacyScope, a static code analyzer that detects violations of the nonreversibility property by enclave program code. PrivacyScope employs symbolic execution to track propagation of private data and records path conditions when program branches throughout the exploration based on a symbolic program input. At the conclusion of program analysis, PrivacyScope generates a report detailing any leakage of private data. PrivacyScope works seamlessly with the secure development environment. As a demonstration, we extend the Intel SGX software development ecosystem with PrivacyScope to automatically detect violation of nonreversibility property on enclave modules.

The main contributions of this work are as following:

- We formally define *nonreversibility* property to characterize the notion of secret data leakage in ML programs. Inspired by noninterference property, nonreversibility accomplishes this by establishing a deterministic relationship between program input and output in TEE-protected application.
- We construct PRIML language to formally describe our proposed innovative approach, PrivacyScope, which automatically detects violations of the nonreversibility property in a TEE-protected application.
- We present a proof-of-feasibility implementation of PrivacyScope leveraging the Clang Static Analyzer and demonstrate the viability and efficacy of our approach by evaluating its performance by analyzing select ML applications executing in Intel SGX enclaves.

4.2 Background

4.2.1 Intel SGX

Intel Software Guard Extensions (SGX) is an extension of the Intel's processor architecture designed to safeguard code and data against unauthorized modification and disclosure. SGX guarantees the integrity and confidentiality of user code and data by providing a processor-hardened, processor-protected, trusted execution environment called an enclave. A user application executing within an enclave is subject to heightened security measures enforced by the processor. Remote attestation, key and credential provisioning are other critical features of the Intel SGX architecture. Remote attestation allows a remote party to verify the authenticity of application code module executing inside an enclave.

Despite its strength, SGX suffers from several hardware security limitations including SGX page faults, cache timing, address bus monitoring and processor monitoring [73]. There has been research working on addressing these limitations like defending against cache timing attacks in [114]. We consider these security limitations orthogonal to our intent of this paper and believe these limitations must be addressed independently from PrivacyScope.

4.2.2 Symbolic Execution

Symbolic execution is a popular program analysis technique that dates back to the 1970s to test whether certain properties can be violated by a piece of software [115, 116]. The key idea is to allow a program to take on symbolic inputs. Then the program is abstractly interpreted by an symbolic execution engine. During interpretation, path condition and symbolic memory store are recorded for each explored control flow path.

By symbolically interpreting TEE-protected applications, PrivacyScope logs symbolic expression of targeted input arguments and uses logged information to track any explicit leakages. Additionally, by tracking target input arguments in path conditions and by combining that information with the returned result from the application, PrivacyScope can detect any implicit leakages. An alternative way to find explicit leakage is to use data flow analysis (DFA) frameworks [117, 118]. Symbolic execution is orders more complex in terms of complexity comparing to DFA. However, most data flow frameworks are path insensitive and are hard to be used for finding implicit leakages.

4.2.3 Clang Static Analyzer

The Clang Static Analyzer is an open source analysis tool for finding bugs in C/C++, and Objective-C programs during compilation phase. The analyzer is a symbolic execution engine that abstractly interprets program code and traces out possible execution paths. After generating the possible execution paths, the analyzer performs conceptually a reachability analysis. During the analysis, the analyzer enters predefined bug checkers. A bug is found by hitting a state where some violation of checking invariants are satisfied. PrivacyScope is a special checker we create, which is implemented on top of Clang Static Analyzer for identifying explicit and implicit leakage of enclave program.

Clang Static Analyzer leverages a region-based memory model to perform path-sensitive symbolic program analysis [119]. The Analyzer can process most forms of C expressions, containing arbitrary levels of pointer dereferencings, pointer arithmetics, composite arrays and struct data types, arbitrary type casts, dynamic memory allocations, etc. These features are critical to PrivacyScope, given that pointer operation is commonplace in C/C++ code and composite structs are widely used in all C/C++ software including data analytic modules.

4.3 Threat Model And Assumptions

The goal of PrivacyScope is to discover deterministic leakage of user private data of an ML application. The threat model follows that of TEE-based secure computation. User private data are encrypted for storage outside the environment, and when they are used for training the mode, the data is decrypted only inside the container for consumption. With recent advances in machine learning, along with the packaging, it is becoming increasingly accessible to the general public, even to those without a machine learning background. As privacy concerns continue to grow, it is likely that the majority of the computation on user data will be conducted in a privacy-preserving manner. And the security of the applications running in the container, which are granted unlimited access to user private data, will be crucial in user privacy protection. However, with new customizations of individual training methods for various application domains, it can be very challenging for an individual user of the ML system, potentially without any expertise on programming language, information flow and machine learning to recognize subtle ways the ML applications can deterministically leak user data. As a result, we assume that there can be unintentional or intentional logic in the TEE-protected application that will leak contents deterministically. PrivacyScope is a detection framework that can take user-defined privacy leakage rules written as PRIML language extension detailed in Section 4.5.1, and automatically analyze TEE-protected programs to see if there is any deterministic information flowing from the input (such as user private data) to the output (such as ML models).

Although PrivacyScope aims at preventing the TEE-protected application from leaking user's private data in a deterministic manner. It is not designed to detect potential information leakage due to various side channels or covert channels. We provide a brief discussion in Section 4.8.1 on potential extension to protect against side/covert channel information leakage.

4.4 Nonreversibility Property

In traditional definition of secret leakage, for a user to keep her data confidential, she would define a policy stating that change of her confidential data cannot affect any publicly observable data. This policy allows programs to perform manipulation and modification on secret data, as long as any observable outputs of these programs do not reveal information about the secret data. Since this policy states that no visible public data is interfered with confidential data, this sort of policy is called a noninterference policy [120]. Intuitively, noninterference for programs guarantees that "*a variation of confidential (high) inputs does not cause a variation of public (low) outputs*" [121].

However, noninterference policy is not suitable for our scenario. In our scenario, machine learning algorithm sits in enclave. We view secret inputs received by enclave as high and the training output to the cloud server as low. In traditional definition, any change of high data would not interfere with low data. But output trained model changes according to received input secret data. In this case, noninterference policy is always violated in machine learning algorithm. Thus, we need a finer grade policy. In contrast to noninterference, we define nonreversibility to guarantee that *a variation of a single confidential (high) input could cause a variation of public (low) outputs, but keeping this single confidential (high) input and (low) inputs unchange will not always lead to the same public (low) outputs.* We provide the formalization of nonreversibility in the following.

We extend notations from [121] and rigorously formalize noninterference and nonreversibility using the machinery of programming-language semantics. We assume that computation starts in an input state $s = (s_H, s_L)$. s_H and s_L contain the initial values of variables of *high* and *low*, respectively. s_h represents any single variable inside s_H and s_l represents any single variable inside s_L. The program either terminates in an output state $s' = (s'_H, s'_L)$ with output values for the high and low variables, or diverges. Thus, the semantics $[\![P]\!]$ of a program P is a function $[\![P]\!] : S \to S_\perp$ (where $S_\perp = S \cup \perp$ and $\perp \notin S$) which maps an input state $s \in S$ either to an output state $[\![P]\!]s \in S$, or to $\perp$ if the program fails to terminate. We define *equivalence relations* $=_L$ and $=_h$. $=_L$ means two states are the same if they are equal on all the low variable (i.e. $s =_L s'$ if and only if $\forall s_l \in s_L$ and $\forall s'_l \in s'_L, s_l = s'_l$). $=_h$ means there exists one variable inside the high variables are the same for two states (i.e. $s =_h s'$ if and only if $s_h = s'_h$). We characterize the observation power of an attacker by a relation $\approx_L$ on behaviors such that two behaviors are related by $\approx_L$ if and only if they are indistinguishable to the attacker. Relation $\approx_L$ implies that the attacker can observe the low variables. For a given semantic model, noninferference is formalized as follows. P is secure if and only if $\forall s_1, s_2 \in S, s_1 =_L s_2 \implies [\![P]\!]s_1 \approx_L [\![P]\!]s_2$. This reads "if two input states share the same low values, then the behaviors of the program executed on these states are indistinguishable by the attacker." Whereas, nonreversibility is formalized as below. P is secure if and only if

$$\forall s_1, s_2 \in S$$
$$s_1 =_L s_2$$
$$s_1 =_h s_2$$
$$\exists h' \neq h, s_{1_{h'}} \neq s_{2_{h'}} \implies [\![P]\!]s_1 \not\approx_L [\![P]\!]s_2$$

which reads "if two input states share the same low values and the same value of one single high variable, then there exists one other high variable that when it is different for the two input states, attacker will observe different behaviors of the program executed on these states." If this other high variable does not exist, then one single high variable will be leaked and P will not be secure. Because, an attacker would be able to look at P and reverse the related computations of that single high variable. According to the formal definition of nonreversibility, the program $l := h_1 + 4$ is insecure and the value of h_1 can be inferred by attacker by observing l. However, the program $l := h_1 + 4 + h_2$ is secure because if h_2 is changed, l observed by attacker will also be changed. And attacker cannot infer value of h_1 without knowledge of h_2. Note that, the probability distribution of inferring h_1 from l will thus be determined by h_2 in this case.

4.5 PrivacyScope Static Analyzer

4.5.1 A General Language: PRIML

For precise declaration of how PrivacyScope works under the hood, we extend notations in [122] and introduce a language called PRIML: PRivacyscope InterMediate Language. PRIML is used for precise declaration purpose, while no compiler or symbolic execution engine is implemented for PRIML. We create PRIML because of the complex semantic model of C/C++ language. PRIML captures the primal semantic model of C/C++. We describe how PrivacyScope analyzes programs written in PRIML to reveal the core ideas. In addition to the PRIML examples in this section, we also show how PrivacyScope is implemented on top of Clang Static Analyzer and is applied to C/C++ enclave modules in section 4.6.

The Backus normal form grammar for PRIML is presented below. A PRIML program is composed of a sequence of statements. Statements consist of assignments and conditional branches. By design, all PRIML expressions are free from any side effects - they do not change the program state. We use "$\diamond_b$" and "$\diamond_u$" to represent binary (e.g. addition, subtraction and etc.) and unary operators (e.g. logical negations, XOR and etc), respectively. The statement get_secret(*secret*) retrieves high variable from *secret* while the statement declassify(*exp*) uncovers a value to the outside world (this is potentially observable for a malicious actor).

For simplicity, we only consider expressions (constants, variables, etc.) which evaluate to 32-bit integer values. We also omit type-checking semantics of PRIML and assume all program under evaluation are well typed, e.g. binary operands are integers or variables.

$$
\begin{array}{lll}
\textit{stmt } s & ::= & \text{skip} \mid \textit{var} := \textit{exp} \mid s_1; s_2 \\
 & & \mid \text{if } \textit{exp} \text{ then } s_1 \text{ else } s_2 \\
\textit{exp } e & ::= & \textit{exp} \diamond_b \textit{exp} \mid \ \diamond_u \textit{exp} \mid \textit{var} \\
 & & \mid \text{get\_secret}(\textit{secret}) \mid v \mid \text{declassify}(\textit{exp}) \\
\diamond_b & ::= & \text{typical binary operators} \\
\diamond_u & ::= & \text{typical unary operators} \\
\textit{value } v & ::= & \text{32-bit unsigned integer}
\end{array}
$$

The operational semantics of a language specify unambiguously how the program should be interpreted in that language. We first define the base operational semantics before we specify program analysis. Each statement rule is of the form:

$$\frac{\text{computation}}{<\text{current state}>, \text{stmt} \leadsto <\text{end state}>, \text{stmt'}}$$

Rules are read from bottom up and left to right. Given a statement, PRIML interpreter pattern-matches at statement to find an applicable rule. For instance, the statement $x := e$ is interpreted according to *ASSIGN* rule. Then the interpreter evaluates the computation given on the top of the rule, and if successful, transitions to the end state. If no rule matches, then the machine halts abnormally. Δ maps a variable to its value for a given execution context, e.g. $\Delta[x]$ denotes the current value of variable x. We denote updating a context variable x with value v as $x \leftarrow v$. Thus, updating the value of variable x to the value 10 in context Δ is denoted as $\Delta[x \leftarrow 10]$. We denote evaluation of an expression e to a value v in the context of Δ by $\Delta \vdash e \Downarrow v$. PRIML interpreter evaluates expression e by matching e to an expression evaluation rule and performing the corresponding computation.

The complete operational semantics for PRIML are shown below. In addition, the context and statement $\Delta, \textit{skip}$ indicates a termination.

$$\frac{v \text{ is input from } \textit{secret}}{\Delta \vdash \text{get\_secret}(\textit{secret}) \Downarrow v} \text{ INPUT} \quad \frac{}{\Delta \vdash \textit{var} \Downarrow \Delta[\textit{var}]} \text{ VAR}$$

$$\frac{\Delta \vdash e \Downarrow v,\ v' = \diamond_u v}{\Delta \vdash \diamond_u e \Downarrow v'} \text{ UNOP} \quad \frac{}{\Delta \vdash v \Downarrow v} \text{ CONST}$$

$$\frac{\Delta \vdash e_1 \Downarrow v_1,\ \Delta \vdash e_2 \Downarrow v_2,\ v' = v_1 \diamond_b v_2}{\Delta \vdash e_1 \diamond_b e_2 \Downarrow v'} \text{ BINOP}$$

$$\frac{\Delta \vdash e \Downarrow v, \Delta' = \Delta[var \leftarrow v]}{\Delta, var := e \rightsquigarrow \Delta', skip} \text{ ASSIGN}$$

$$\frac{\Delta \vdash e \Downarrow 1}{\Delta, \text{if } e \text{ then } s_1 \text{ else } s_2 \rightsquigarrow \Delta, s_1} \text{ TCOND}$$

$$\frac{\Delta \vdash e \Downarrow 0}{\Delta, \text{if } e \text{ then } s_1 \text{ else } s_2 \rightsquigarrow \Delta, s_2} \text{ FCOND}$$

$$\frac{\Delta' = \Delta, s_1}{\Delta, s_1; s_2 \rightsquigarrow \Delta', s_2} \text{ COMP} \qquad \frac{}{\Delta, skip; s \rightsquigarrow \Delta, s} \text{ SKIP}$$

$$\frac{\Delta \vdash e \Downarrow v, \text{declassify } v}{\Delta, declassify(e) \rightsquigarrow \Delta, skip} \text{ DECLASS}$$

4.5.2 PrivacyScope Program Analysis

In this section, we describe how PrivacyScope analyzes programs written in PRIML language. We present how PrivacyScope works for PRIML to shed light upon how PrivacyScope works for C/C++. The objective of the analysis is to detect any violation of nonreversibility property in PRIML programs. PrivacyScope achieves this by combining taint tracking and forward symbolic execution. Taint tracking is used to track the flow of *high* information from its sources to its sinks. Forward symbolic execution is used to reason about the behavior of the program under analysis given initial inputs. PrivacyScope represents the path condition of program execution as a logical formula, thus reducing the reasoning of a program's behavior to domain of logic. PrivacyScope represents variables symbolically and thus can examine program execution spanning multiple input space of the program at one time.

We express PrivacyScope in terms of the operational semantics of PRIML. To keep track of the taint status of each program value, we redefine values in PRIML to be tuples of the form $< v, \tau >$, where v is a value in the initial language, and τ is the taint status of v. τ is modeled by a security semi-lattice with join operation only shown in Fig. 4.1. In this semi-lattice, sensitive data is labeled by t1, t2 and more. If a variable is labeled by $\perp$, it means it is not sensitive. While if a variable is labeled by $\top$, it means it is tainted by two or more taint sources, so revealing it would not break nonreversibility. We also introduce a new mapping τ_Δ which maps variables to taint status.

To enable forward symbolic execution in PRIML, we introduce the following changes to

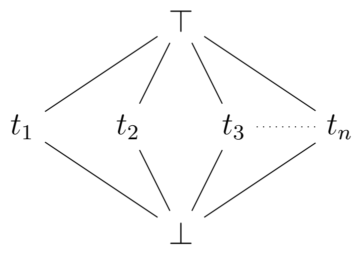


Figure 4.1: Security semi-lattice for taint status

Component	Policy Check
$P_{get\_secret}(secret)$	t_n
$P_{const}()$	$\bot$
$P_{unop}(t), P_{assign}(t)$	t
$P_{binop}(t_1, t_2), P_{cond}(t_1, t_2)$	see Fig. 4.2
$P_{declassify\_check}(v, t, \pi, \tau_{\Delta}[\pi])$	see Alg. 3

Table 4.1: PrivacyScope's policy for nonreversibility violation.

PRIML.

value v	::=	32-bit unsigned integer | *exp*
π	::=	Contains current constraints on symbolic variables due to path branches

These changes make partially evaluated symbolic expressions valid for a value in PRIML. Thus, when get_secret(*secret*) is evaluated symbolically, it can return a symbol instead of a concrete value.

t_1	t_2	$P_{binop}(t_1, t_2), P_{cond}(t_1, t_2)$
$\top$	$\top$	$\top$
$\top$	t_2	$\top$
t_1	$\top$	$\top$
t_1	t_2	$\top\ if\ t_1 \neq t_2\ else\ t_1$
t_1	$\bot$	t_1
$\bot$	t_2	t_2
$\bot$	$\bot$	$\bot$

Figure 4.2: Truth table for $P_{binop}(t_1, t_2)$ and $P_{cond}(t_1, t_2)$

[1] $\neg$ operator negates the most recent added path constraint in π

Algorithm 3

$P_{declassify\_check}$

```
1: if $t \neq \perp$ or $\top$ then
2:     abort and report explicit leakage;
3: end if
4: if $\tau_\Delta[\pi] \neq \perp$ or $\top$ then
5:     if $\pi$ is in hashmap $hm$ then
6:         if $v \neq hm[\pi]$ then
7:             abort and report implicit leakage;
8:         else
9:             remove $\pi$ in $hm$
10:            continue program analysis
11:        end if
12:    else
13:        $hm[\neg^1\pi] := v$
14:        continue program analysis
15:    end if
16: else
17:    continue program analysis
18: end if
```

Statement	Δ	τ_Δ	abort
$h_1 := 2*$ get_secret($secret$)	$\{h_1 \to 2*s_1\}$	$\{h_1 \to t_1\}$	$false$
$h_2 := 3*$ get_secret($secret$)	$\{h_1 \to 2*s_1,\, h_2 \to 3*s_2\}$	$\{h_1 \to t_1,\, h_2 \to t_2\}$	$false$
$x := h_1 + h_2$	$\{h_1 \to 2*s_1,\, h_2 \to 3*s_2,\, x \to 2*s_1 + 3*s_2\}$	$\{h_1 \to t_1,\, h_2 \to t_2,\, x \to \top\}$	$false$
$declassify(x)$	$\{h_1 \to 2*s_1,\, h_2 \to 3*s_2,\, x \to 2*s_1 + 3*s_2\}$	$\{h_1 \to t_1,\, h_2 \to t_2,\, x \to \top\}$	$false$
$declassify(h_1)$	$\{h_1 \to 2*s_1,\, h_2 \to 3*s_2,\, x \to 2*s_1 + 3*s_2\}$	$\{h_1 \to t_1,\, h_2 \to t_2,\, x \to \top\}$	$true$

Table 4.2: Simulation of PrivacyScope detecting explicit leakage

After introducing the aforementioned changes, Table 4.1 presents PrivacyScope policy for detecting nonreversibility violation. It introduces taint status into the system by marking up all values returned by get_secret($secret$) with different tainted status. Taint is then propagated through the program according to propagation rules. For constants, they are labeled as insensitive. For assignment and unary operations on a variable, they keep the same taint label for the variable. Fig. 4.2 shows taint label propagation rule for binary operation and conditional branches. The policy checks if $declassify(e)$ leaks secret whenever a value is revealed. Alg. 3 depicts $declassify(e)$ process. It first checks if the variable is labeled as sensitive. If yes, it reports an explicit leakage. If no, it then checks if the path constraint is

Statement	Δ	π	τ_Δ	hm	abort
$h := 2*$ get_secret($secret$)	$\{h \to 2*s\}$	$true$	$\{\pi \to \perp$ $h \to t_1\}$	$\{\varnothing\}$	$false$
if $h - 5 == 14$ then $declassify(0)$ else $declassify(1)$	$\{h \to 2*s\}$	$[(2*s) - 5 == 14]$	$\{\pi \to t_1$ $h \to t_1\}$	$\{\neg[(2*s) - 5 == 14] \to 0\}$	$false$
if $h - 5 == 14$ then $declassify(0)$ else $declassify(1)$	$\{h \to 2*s\}$	$\neg[(2*s) - 5 == 14]$	$\{\pi \to t_1$ $h \to t_1\}$	$\{\neg[(2*s) - 5 == 14] \to 0\}$	$true$

Table 4.3: Simulation of PrivacyScope detecting implicit leakage

sensitive. It uses a hashmap hm to assist with checking whether the revealed variable differs in distinct branches. Finally, at the end of the last path's interpretation, $declassify(e)$ checks if there is any item in hashmap hm. If so, it concludes that there is an implicit violation of nonreversibilitity. Note that, this last step is omitted in Alg. 3 to simplify the explanation. We summarize the PrivacyScope operational semantics as following.

$$\frac{v \text{ is a fresh symbol}}{\tau_\Delta, \Delta \vdash \text{get\_secret}(secret) \Downarrow < v, P_{secret}(secret) >} \text{ PS-INPUT}$$

$$\frac{}{\tau_\Delta, \Delta \vdash var \Downarrow < \Delta[var], \tau_\Delta[var] >} \text{ PS-VAR}$$

$$\frac{\tau_\Delta, \Delta \vdash e \Downarrow < v, t >, \ < v', t' > = \diamond_u < v, t >}{\tau_\Delta, \Delta \vdash \diamond_u e \Downarrow < v', P_{unop}(t) >} \text{ PS-UNOP}$$

$$\frac{}{\tau_\Delta, \Delta \vdash v \Downarrow < v, P_{const}() >} \text{ PS-CONST}$$

$$\frac{\tau_\Delta, \Delta \vdash e_1 \Downarrow < v_1, t_1 >, \ \tau_\Delta, \Delta \vdash e_2 \Downarrow < v_2, t_2 >}{\tau_\Delta, \Delta \vdash e_1 \diamond_b e_2 \Downarrow < v_1 \diamond_b v_2, P_{binop}(t_1, t_2) >} \text{ PS-BINOP}$$

$$\frac{\begin{array}{c}\tau_\Delta, \Delta \vdash e \Downarrow < v, t >, \Delta' = \Delta[var \leftarrow v], \\ \tau'_\Delta = \tau_\Delta[var \leftarrow P_{assign}(t)]\end{array}}{\tau_\Delta, \Delta, var := e \rightsquigarrow \tau'_\Delta, \Delta', skip} \text{ PS-ASSIGN}$$

$$\frac{\begin{array}{c}\tau_\Delta, \Delta \vdash e \Downarrow < e', t' >, \pi' = \pi \wedge (e' = 1), \\ \tau'_\Delta = \tau_\Delta[\pi \leftarrow P_{cond}(t', \tau_\Delta[t])]\end{array}}{\pi, \tau_\Delta, \Delta, \text{if } e \text{ then } s_1 \text{ else } s_2 \rightsquigarrow \pi', \tau'_\Delta, \Delta, s_1} \text{ PS-TCOND}$$

$$\frac{\begin{array}{c}\tau_\Delta, \Delta \vdash e \Downarrow < e', t' >, \pi' = \pi \wedge (e' = 0), \\ \tau'_\Delta = \tau_\Delta[\pi \leftarrow P_{cond}(t', \tau_\Delta[t])]\end{array}}{\pi, \tau_\Delta, \Delta, \text{if } e \text{ then } s_1 \text{ else } s_2 \rightsquigarrow \pi', \tau'_\Delta, \Delta, s_2} \text{ PS-FCOND}$$

$$\frac{\tau'_\Delta, \Delta' = \tau_\Delta, \Delta, s_1}{\tau_\Delta, \Delta, s_1; s_2 \rightsquigarrow \tau'_\Delta, \Delta', s_2} \text{ PS-COMP}$$

$$\frac{}{\tau_\Delta, \Delta, skip; s \rightsquigarrow \tau_\Delta, \Delta, s} \text{ PS-SKIP}$$

$$\frac{\tau_\Delta, \Delta \vdash e \Downarrow < v, t >, P_{declassify\_check}(v, t, \pi, \tau_\Delta[\pi])}{\tau_\Delta, \Delta, declassify(e) \rightsquigarrow \tau_\Delta, \Delta, skip}$$

PS-DECLASS

Next, we present two simplified code segments written in PRIML to further explain the program analysis function of PrivacyScope.

Example 1. Consider the following program:

$$h_1 := 2 * get\_secret(secret)$$

$$h_2 := 3 * get\_secret(secret)$$

$$x := h_1 + h_2$$

$$declassify(x)$$

$$declassify(h_1)$$

We can easily see that declassifying x does not violate nonreversibility property. Knowledge of the value of x does not immediately reveal the value of the first *secret*. However, declassifying h_1 allows an attacker to infer the value of the first *secret* by dividing the observed value with 2. Table. 4.2 presents a simulation of how PrivacyScope detects a leakage. Row 5 shows a leakage since the taint status of h_1 is t_1, while row 4 does not because the taint status of x is $\top$.

Example 2. Consider the following code snippet:

$$h := 2 * get\_secret(secret)$$

$$\text{if } h - 5 == 14 \text{ then } declassify(0) \text{ else } declassify(1)$$

By observing the declassified output, an attacker can easily infer if h is equal to 19 or not and, ultimately recover the *secret*. Table. 4.3 illustrates how PrivacyScope detects implicit leakage. Row 3, for instance, reports a leakage since the taint status for π is t_1, and the value retrieved from the hashmap hm is 0 which is different from what $declassify$ is outputting (1). Row 2 of Table. 4.3, on the other hand, does not report a leakage even though the taint status for π is t_1. Because nothing is stored in the hashmap hm before the interpretation.

4.5.3 Incorporating PrivacyScope in an Intel SGX enclave

Intel SGX enclave modules are typically written in C/C++. Therefore, the following part incorporates aforementioned core ideas written in PRIML and presents how PrivacyScope is integrated into Intel SGX ecosystem. Each Intel SGX enclave declares one or more entry points into the enclave. Referred to as ECALLS by the Intel SGX SDK, these interfaces allow untrusted outside applications access trusted code running inside the enclave. An enclave may also have OCALLS, which allow trusted enclave code to call out to the untrusted application. To configure an application to run in an Intel SGX enclave, an enclave interface definition file is created. An EDL file resembles a traditional C header file and contains prototype declarations for all ECALL and OCALL interfaces. The enclave EDL file defines how data is marshalled between enclaves trusted code and the outside untrusted applications.

To pass secret data to enclave code for further processing, enclave uses ECALL type interface. Similar to C function prototypes, the ECALL interface parameters are annotated with attributes like [*in*] and/or [*out*]. A parameter with [*in*] attribute is used for marshalling data from outside untrusted application into the enclave. So in our example, secret data is passed into enclave via [*in*] parameter(s). Interface parameters with [*out*] attribute are used to marshal data from inside the enclave to the outside untrusted application. The Intel SGX SDK abstracts out the details of the data marshalling by generating the necessary proxy code. In short, [*in*] parameters correspond to get_secret(*secret*) in the previous section.

Prior to performing code analysis, PrivacyScope processes an XML configuration file, provided by user, containing function names that the user is interested in evaluating. PrivacyScope also extracts information included in the SGX EDL configuration file. Following a quick initialization step, PrivacyScope analyzes program code using the approach outlined in the previous section and generates a report summarizing the outcome of the code analysis including any violations of nonreversibility property. For *explicit* information leakage cases, the report describes how program output can be used to infer its (secret) input thus assisting developers in securing their code. For implicit information leakage, the report provides path conditions and returns results which can result in leakage of secret data.

4.6 Evaluations on PrivacyScope Prototype

4.6.1 Implementation

For our proof-of-concept prototype, we use the Intel SGX SDK version 2.0 to construct a trusted application running in a SGX enclave powered by an Intel NUC, running under Ubuntu 14.04 TLS. Built on top of Clang v7.0.0, we build a prototype of PrivacyScope by adding over 1 KLOC. We have evaluated the prototype of PrivacyScope by porting open source machine learning programs written in C/C++ to Intel SGX enclaves and analyzing the enclave modules for data leakage.

4.6.2 Illustration: a Leakage Example in C

To illustrate PrivacyScope operation on real Intel SGX enclave module written in C, we provide an illustrative example here. For simplicity, our example neglects decryption of secret data. However, PrivacyScope does consider decryption of encrypted secret data, it records decryption function names from Intel SGX IPP library in a predefined list. And when PrivacyScope meets predefined decryption functions, it assigns the symbolic value of

Execution State	Line #	*stmt*	*env*	σ	π
A	2	int temporary = secrets[0] + 100;	$\emptyset$	$\emptyset$	True
B	3	output[0] = temporary + 1;	$secrets \rightarrow reg_0$ $secrets[0] \rightarrow reg_1$ $temporary \rightarrow reg_1 + 100$	$reg_0 \rightarrow SymRegion$ $reg_1 \rightarrow reg_0[0]$	True
C	4	if (secrets[1] == 0)	$secrets \rightarrow reg_0$ $secrets[0] \rightarrow reg_1$ $temporary \rightarrow reg_1 + 100$ $output \rightarrow reg_2$ $output[0] \rightarrow reg_1 + 101$	$reg_0 \rightarrow SymRegion$ $reg_1 \rightarrow reg_0[0]$ $reg_2 \rightarrow SymRegion$	True
D	5	return 0;	$secrets \rightarrow reg_0$ $secrets[0] \rightarrow reg_1$ $temporary \rightarrow reg_1 + 100$ $output \rightarrow reg_2$ $output[0] \rightarrow reg_1 + 101$ $secrets[1] \rightarrow reg_3$	$reg_0 \rightarrow SymRegion$ $reg_1 \rightarrow reg_0[0]$ $reg_2 \rightarrow SymRegion$ $reg_3 \rightarrow reg_0[1]$	$reg_0[1] == 0$
E	7	return 1;	$secrets \rightarrow reg_0$ $secrets[0] \rightarrow reg_1$ $temporary \rightarrow reg_1 + 100$ $output \rightarrow reg_2$ $output[0] \rightarrow reg_1 + 101$ $secrets[1] \rightarrow reg_3$	$reg_0 \rightarrow SymRegion$ $reg_1 \rightarrow reg_0[0]$ $reg_2 \rightarrow SymRegion$ $reg_3 \rightarrow reg_0[1]$	$reg_0[1] \neq 0$

Table 4.4: Exploration of illustrative example

secret data to decrypted secret data. For illustration, we define ECALL function in EDL file for processing secret data as *int enclave_process_secret*([*in*] *secrets*, [*out*] *output*). Source code of *enclave_process_secret* function is shown in Listing 4.1. In this simplified example, we can easily see that *secrets*[0] is explicitly leaked and *secrets*[1] is implicitly leaked.

```c
int enclave_process_data(char *secrets, char *output){
    int temporary = secrets[0] + 100;
    output[0] = temporary + 1;
    if(secrets[1] == 0)
        return 0;
    else
        return 1;
}
```

Listing 4.1: Code snippet of illustrative example in C

Next we show how PrivacyScope explores the illustrative example and identifies explicit and implicit privacy leakage using the symbolic execution engine of Clang Static Analyzer. First, we need to define the state that the symbolic execution engine must maintain. We define the state as a 4-tuple $(stmt, env, \sigma, \pi)$ similar to [119] where:

- *stmt* represents the next statement in source code to be evaluated. In our illustrative example, a *stmt* can be an assignment, a conditional branch, or a return statement.
- *env* is the environment which maps from *lvalue* (an *lvalue* is an expression with an object type according to C standard [123]) expressions to memory regions (abstract

representation of memory objects) reg_i. A memory region can be the subregion of another region. And when a variable of array type is defined, it has subobjects called elements. For array elements, we have *ElementRegion* like $reg_i[0]$ with its super region reg_i to represent the array region. As for when a pointer is pointing to some unknown memory block, we have *SymRegion* for representing the memory block pointed to by the symbolic pointer. *SymRegion* represents a region that serves as an alias for either a real region, a NULL pointer, etc. It essentially is used to map the concept of symbolic values into the domain of regions.

- σ is a store which maps from memory regions to concrete values, symbolic values α_i or memory regions reg_i.
- π denotes path constraints on symbolic values. At the beginning of a symbolic execution, π is set to True. As the symbolic execution engine explores the program statements, π grows when branches are met and assumptions on taking any branch are recorded in π as a formula. This formula indicates how execution can reach any *stmt* in the program code under analysis.

Depending on *stmt*, the symbolic engine of Clang Static Analyzer changes states as following:

- The evaluation of an assignment $x = e$ updates the environment *env* and store σ. e can be any legal expression involving unary or binary operators over symbolic or concrete values. If e contains unknown *lvalue* expressions in the context of current execution state, new reg_is are initialized for the unknown expressions in updated *env* and mapping between newly initialized reg_i and context of current execution state is created in updated σ. Assuming e_s is the symbolic expression of evaluating e, e_s is associated with x in updated *env*.
- The evaluation of a conditional branch $if\ e\ then\ stmts_{true}\ else\ stmts_{false}$ affects the path constraints π. When a conditional branch is met, the symbolic engine will fork and create two new execution states, one with path condition $\pi_{true} = \pi \wedge e_s$ and the other with $\pi_{false} = \pi \wedge \neg e_s$, where e_s is the symbolic expression by evaluating e. Symbolic engine will follow the two newly created execution states one by one.

Table 4.4 presents the symbolic exploration of our illustrative example. Initially (execution state A), the path condition is set to true and *env* and σ are null. During the evaluation of line 2 right-hand side (RHS), the first evaluated expression is *secrets*, so a new region reg_0 is generated and associated with *secrets* in *env* and reg_0 is mapped to *SymRegion* in σ. Then it follows the evaluation of $secrets[0]$ which leads to a new expression region map of $secrets[0]$ to reg_1 in *env*. σ is also updated with a new map of reg_1 to $reg_0[0]$. After the evaluation of RHS, the result, $reg_1 + 100$, is associated with *temporary*. Next (execution state B), the evaluation of RHS of line 3 returns $reg_1 + 101$ within the execution context and

Box 1: Report generated by PrivacyScope for the illustrative example.

```
illustrative_example.c:5:9: warning: The memory region ‘reg_$3 <char element {SymRegion {reg_$0 <char* secrets>}, 1 S64b, char}>’ and its concrete value ‘{0, 0}’ breaks privacy and implicitly leaks sensitive data!
illustrative_example.c:8:6: warning: The memory region ‘SymRegion{reg_$1 <char* output>}’ and its symbolic value ‘(reg_$2 <char* secrets>}, 0 S64b, char}>) + 101’ breaks privacy and explicitly leaks sensitive data!
```

Open Source ML Code	Size (LoCs)	Execution Time (sec.)
LinearRegression	161	2.549s
Kmeans	179	4.654s
Recommender	117	1.758s

Table 4.5: Performance evaluation

the result is associated with the evaluation of line 3's left-hand side (LHS). The evaluation of line 3's LHS brings a new region reg_2 and it is associated with *output* in *env*. Meanwhile, reg_2 is mapped to *SymRegion* in σ. Besides, the result of line 3's RHS, $reg_1 + 101$, is associated with $output[0]$ in *env*. When a conditional branch is met (execution state C), the engine will fork into two execution states (D and E) with opposite path constraints on the comparison statement. During execution of state C, a new region reg_3 is assigned to $secrets[1]$ in *env* and reg_3 is an *ElementRegion* of reg_0. By evaluating the comparison statement, two opposite path conditions, $reg_0[1] == 0$ and $reg_0[1] \neg = 0$, are added into π of execution states D and E. The evaluation of return statement calls the procedure of policy check similar to previous $P_{declassify\_check}$.

When PrivacyScope starts exploring the target function *enclave_process_ data*, it first goes into EDL file and fetches parameters as specified by user predefined rules. If no rules are predefined, the default action is to mark $[out]$ attribute parameters as potential leaking point, and $[in]$ attribute parameters as secrets. Then PrivacyScope starts exploration of source code as described in previous paragraphs. During the exploration, PrivacyScope introduces taint status to secret variables and propagates the tainting as mentioned in program analysis for PriML. When *enclave_process_data* function returns or ends, PrivacyScope performs a policy check similar to $P_{declassify\_check}$. For explicit privacy leakage check, PrivacyScope checks $[out]$ parameters to see their taint status. In the illustrative example case, $output[0]$ is tainted by t_1, so $output[0]$ explicitly leaks the value of $secrets[0]$. For implicit privacy leakage check, PrivacyScope utilizes hashmap *hm* and find that the returned values are different for different π which branch on subobject of *secrets*. The warning report generated for the illustrative example is shown in Box 1.

4.6.3 Performance Evaluation

Our primary objective is to detect any violation of nonreversibility property in a trusted code executing inside a SGX enclave. However, current version of SGX SDK does not support

	Approach			Target Leakage							Target System
	Type System	Dynamic Analysis	Static Analysis	Explicit flow		Implicit flow					
				NonInt*	NonRev*	NonInt*	NonRev*	Termination	Timing	Probability	
Taintdroid[124]		✓		✓							Android
HDL Checker[125]	✓			✓		✓					Hardware Design
Panorama[126]		✓		✓							Windows OS
Androidleaks[127]			✓	✓							Android
Covert Flow Checker[128]	✓							✓			An Imperative Language
Timing Leaks Transformer[129]	✓								✓		An Imperative Language
Multithread Possibilistically NonInt[130]	✓									✓	A Concurrent Language
Jflow[15]	✓			✓		✓					Java
Moat[113]	✓			✓		✓					Intel SGX Enclave
This work			✓		✓		✓				Intel SGX Enclave

* NonInt is short for noninterference, NonRev is short for nonreversibility.

Table 4.6: Systematic approaches for detecting secret leakage

easy migration of legacy application even if it is written in C/C++ and there is no existing open source implementation of ML algorithms implemented inside enclave using the Intel SGX SDK. Thus, we had to port open source ML algorithms so that they can be executed inside a SGX enclave. We selected three popular open source ML projects from the public Github repository and ported them using Intel SGX SDK. They included LinearRegresssion, Kmeans and Recommender [131, 132, 133]. To evaluate the efficacy of PrivacyScope, we inserted malicious code inside the ported ML code. The results were examined by the authors and the efficacy of PrivacyScope solution was verified by the authors manually. During this process, we also detected multiple preexisting secret leakage in the open source Recommender implementation. We provide a detailed explanation of these findings in the next section. Table 4.5 summarizes performance data, including the PrivacyScope code analysis time for each of the three open source ML projects. The analysis time was measured using the Linux OS built-in time utility. The total execution time was computed by summing up the usr and sys times.

4.6.4 Case Studies

The goal of PrivacyScope is to assist users and developers in identifying potential information leakage vulnerabilities in program code intended to run in a TEE-protected environment. PrivacyScope accomplishes this objective by identifying any violations of the nonreeversibility property. In this section, we present two case studies to demonstrate PrivacyScope's capabilities by analyzing the behavior of two open source ML programs. These two examples by no means cover all the possible scenarios. In the first case, PrivacyScope uncovers implementation defects caused by inadvertent coding errors during software development. In the second case, we illustrate how PrivacyScope can detect malicious code inserted into the codebase by a malicious actor.

Finding information leakage in Recommender

Our first case study is the analysis of a C library for product recommendations/suggestions using collaborative filtering (CF) [134]. Recommender analyzes and learns from collective feedback of a large number of users. It then uses user preference to predict and recommend the most appropiate products for a particular user. We found 6 violations of the nonreversibility property in this open source ML project.

In a normal workflow, Recommender first selects a learning method for its learning process. After that, Recommender grabs users' id and their grading of preference for products to train the model. After training procedure, the resulting model can be used to predict how much a user may like a product previously ungraded for this user. In our scenario, we are concerned of service provider knowing the exact grading for products of a single user. As a matter of fact, we find several learning methods implemented in Recommender would lead to leakage of user's grading (e.g. learn_mf_bias, learn_basic_mf, learn_mf_neighbor and learn_social). In the following paragraphs, we go through the leakage in learn_mf_bias and learn_basic_mf and how we find them with the help of PrivacyScope in detail.

We start with an easy to understand leakage in learn_mf_bias and followed by a more complicated leakage in learn_basic_mf. In the learn_mf_bias learning method, when the hyper parameter of the learning algorithm is set to particular value, sensitive data can be reversed from the output. According to current setting in machine learning as a service[135], it is commonplace for service provider to control the hyper parameters of a learning algorithm. Listing 4.2 shows the function called when Recommender selects learn_mf_bias as its learning method. The sensitive data is transferred into this function through learning_param. The outcome of this function is data struct lfactors which contains the learnt model. During the process inside function learn_mf_bias, it calls another function calculate_average_ratings with sensitive data tset and output data struct lfactors as in Listing 4.3. With help of PrivacyScope, we find out that the ratings_average field of lfactors could potentially leak sensitive data.

PrivacyScope reports an explicit leakage because it finds out tset->ratings_sum labeled as sensitive input crosses security boundary and is transferred out through lfactors->ratings_average. The authors manually verify the finding and figure out tset->ratings_sum contains sum of the first user's rating if training_set_size is set to 1. What's more, if the service provider set the hyper parameter to receive one rating at a time and use it to train a model, the service provider can infer all the individual ratings for every user through simple computation on the lfactors->ratings_average.

```
struct learned_factors* learn_mf_bias (learning_algorithm_params_t*
    learning_param)
{
    /* learning_param contains secret input in its member tset*/
  struct training_set* tset = learning_param->tset;
    /* lfactors contains resulting learnt model*/
  struct learned_factors* lfactors = init_learned_factors (&params);
  ...
  calculate_average_ratings (tset , lfactors);
  ...
  return lfactors;
}
```

Listing 4.2: function learn_mf_bias

```
calculate_average_ratings (struct training_set* tset , learned_factors_t*
    lfactors)
{
  /* training_set_size is a hyper parameter selected by service provider */
  double average_rating = (double) tset->ratings_sum / ( (double)
   training_set_size);
  average_rating = (average_rating - 1) / 4.0;
  /* ratings_average leak sensitive data */
  lfactors->ratings_average = log ( (double) average_rating / (1 -
   average_rating) );
}
```

Listing 4.3: function calculate_average_ratings

In learn_basic_mf, it does not have ratings_average issue, however, during our porting of Recommender to Intel SGX enclave, we find out that it contains improper usage of random generator. And this leads to same initial matrix values for a new model training. We have reported this bug to the Github code owner and he has confirmed the bug. We further investigate will any leakage happens if the bug is intentional planted by service provider. By assuming this, PrivacyScope finds out logics hidden inside this function which leads to sensitive data leakage.

We explain the improper usage of random generator here first. Normal use of srand() function should output a random number like srand(time(0)), however, Recommender use srand(0) to set the seed of the random number generator to 0 every time. This causes the random generator to create the same output everytime. For example, in Listing 4.4, we uses srand(0) to set the seed of random number generator to 0. And we print out the random number it generates with rand(). Every time this program will print the same number sequence.

```
    srand(0);
    for(int i = 0; i<5; i++)
        printf(" %d ", rand());
    return 0;
```

Listing 4.4: improper srand usage

Recommender uses generate_random_matrix to create random matrix for initial training state, however, generate_random_matrix sets seed to 0 always and uses box_muller to get random number. The number sequence box_muller generates is always the same because of the same seed is used. We show generate_random_matrix and box_muller in Listing 4.5and Listing 4.6 respectively.

```c
double** generate_random_matrix(int nrow,int ncol,int seed)
{
  ...
  srand(seed);
  ...
        /* normal random variate generator */
         matrix[i][j] = box_muller(0,0.1);
  ...
  return matrix;
}
```

Listing 4.5: function generate_random_matrix

```c
double box_muller(double m, double s)
{
  ...
  x1 = 2.0 * rand()/RAND_MAX - 1.0;
  x2 = 2.0 * rand()/RAND_MAX - 1.0;
  ...
}
```

Listing 4.6: random generator

Now we know the generate_random_matrix in Recommender always create the same initial state for training, we can continue to see how PrivacyScope find sensitive data leakage. We first set all the initial states set by generate_random_matrix to a manmade concrete value. For example, in Listing 4.7, lfactors->item_factor_vectors and lfactors->user_factor_vectors are generated by function generate_random_matrix, so we set concrete values for them. After this, we run the program analysis on the learning method.

```c
struct learned_factors* init_learned_factors (struct model_parameters * params
    )
{
    ...
  lfactors->item_factor_vectors = generate_random_matrix (params->items_number
    , params->dimensionality , params->seed);
  lfactors->user_factor_vectors = generate_random_matrix (params->users_number
    , params->dimensionality , params->seed);
    ...
  return lfactors;
}
```

Listing 4.7: initialization function of learning model

PrivacyScope reports item_factors and user_factors fields of data struct lfactors leak sensitive data. After manual verification of function learn_basic_mf in Listing 4.8, we confirm the leakage. We find r_iu stores sensitive data and it is passed to e_iu because r_iu_estimated is a concrete value. For the loops, the symbolic execution engine unrolls them to a limit and the limit is set to 4. The sensitive e_iu then is passed to function compute_factors as predicted_error in Listing 4.10. Sensitive predicted_error is computed with concrete values and service provider selected hyper parameters to set the value of item_factors and user_factors. However, item_factors and user_factors are labeled as output, thus this breaks the rule set by PrivacyScope. In the case where service provider controls the hyper parameters and knows the number sequence of random number generator, he can choose to take user ratings one at a time and inferring the rating by knowing item_factors and user_factors.

```
struct learned_factors* learn_basic_mf(learning_algorithm_params_t*
    learning_params)
{
  struct learned_factors* lfactors = init_learned_factors(&learning_params->
    params);
    ...
  /* every field of data struct params are hyper parameters selected by
    service provider*/
  for (k = 0; k < learning_params->params.iteration_number; k++)
  {
    for (r = 0; r < learning_params->params.training_set_size; r++)
    {
        /* r_iu stores the sensitive input */
       r_iu = learning_params->tset->ratings->entries[r].value;
            ...
        /* r_iu_estimated is concrete value as shown in Listing 9*/
       r_iu_estimated = estimate_item_rating(item_factors, user_factors,
    learning_params->params.dimensionality);
        /* e_iu is labeled sensitive */
       e_iu = r_iu - r_iu_estimated;
       compute_factors(item_factors, user_factors, learning_params->params.
    lambda, learning_params->params.step, e_iu, learning_params->params.
    dimensionality);
    ...
  return lfactors;
}
```

Listing 4.8: function learn_basic_mf

```
double estimate_item_rating(double* user_vector, double* item_vector, size_t
    dim)
{
  double sum = 0;
  /* item_factors, user_factors are set as concrete values because they are
    generated by generate_random_matrix */
  for (size_t i = 0; i < dim; i++)
    sum += user_vector[i] * item_vector[i];
  return sum;
}
```

Listing 4.9: function estimate_item_rating

```
void compute_factors( double* item_factors, double* user_factors, double
    lambda, double step, double predicted_error, size_t dimensionality)
{
  /* step, lambda, dimensionality are hyper parameters selected by service provider */
  for ( size_t i = 0; i < dimensionality; i++)
  {
      /* item_factors and user_factors leak sensitive data*/
    item_factors[i] = item_factors[i] + step * (predicted_error * user_factors[i] - lambda * item_factors[i]);
    user_factors[i] = user_factors[i] + step * (predicted_error * item_factors[i] - lambda * user_factors[i]);
  }
}
```

Listing 4.10: function compute_factors

Verifying effectiveness of PrivacyScope in Kmeans

Our second case study is mimicking a malicious enclave writer and embedding sensitive data leakage logic inside enclave programs. We add explicit and implicit leakage logic to open source machine learning program Kmeans [133].

In the normal workflow of Kmeans, it first sets up hyper parameters like maximum iteration number, number of objects, number of centroid points, method to calculate distance. Then, it populates objects into data struct config. After that, Kmeans performs kmeans algorithm on the data struct config and output the learnt centroid points. as in Listing 4.11.

```
void enclave_kmeans(char *objects, char* result){
  ...
  /* populate objects */
  for (i = 0; i < config->num_objs - 1; i++)
  {
    config->objs[i] = &(objects[i]);
  }
  ...
  /* algorithm converges when the assignments no longer change or max_iteration is reached */
  while (1)
  {
    ...
    /* Assignment step: Assign each observation to the cluster whose mean has the least squared Euclidean distance */
    update_r(config);
    /* Update step: Calculate the new means to be the centroids of the observations in the new clusters */
    update_means(config);
    ...
  }
  ...
}
```

```
21 }
```

Listing 4.11: Kmeans code snippet

In this case, the sensitive data we want to protect is the objects. We verify effectiveness of PrivacyScope by inserting assignment statements of objects to result explicitly before the enclave ends at line 19 of Listing 4.11. We also embed implicit leakage at line 19 of Listing 4.11 to verify the effectiveness of PrivacyScope. PrivacyScope reports leakage in both cases. What worth mentioning here is that PrivacyScope is built on top of Clang static analyzer's symbolic engine, so PrivacyScope inherits its unsoundness. The symbolic engine unrolls the loop with a default limit of 4. Although this value is configurable, an enclave writer can bypass the detection easily and this is a well-known challenge in finding bugs with symbolic execution.

4.7 Related Work

4.7.1 Information Flow Analysis Methods

Use of information flow analysis to detect information leakage within programs has been the subject of much research in the past decades. Language-specific methods typically augment type systems so that they can statically check the flow of private data within programs that manipulate the data. [15], for example, integrates information flow analysis into the Java language type system, while [129, 15, 130] propose innovative analysis methods for imperative and concurrent language. [125] detects security vulnerabilities in hardware design written in HDL. In these solutions, they focus on addressing the noninterference property in programs.

In addition to aforementioned information flow analysis, a plethora of methods have focused on detecting, measuring and understanding the nature of privacy leakage on different platforms. Static and dynamic tainting analysis are two major categories of methods for detecting privacy leakage. Dynamic methods typically monitors the system at runtime and examine the system as it executes the code. Static methods on the other hand use compile time analysis to predict the impact of code execution on private data. Static methods can have a higher false positive rate as compared to dynamic methods [127]. [124] leverages dynamic tainting to detect intentional leakage of private data in the Android operating system environment while [126] examines system-wide information flow for malware detection in the Windows platform. Finally, [113] uses model checking to verify the information flow properties of code running in a trusted enclave. Table 4.6 summarizes key recent research work in the area of information analysis methods, highlighting the target environment for each approach.

4.7.2 Secure Systems on Trusted Hardware

There have been many recent advances in the area of trusted hardware platforms including the development of commercial off-the-shelf secure processors. ARM TrustZone [136] offers a processor capable of executing in a secure world as well as a normal world with isolated address spaces. TPM+TXT [137] provides attestation on the execution state of a platform, but all privileged software must execute in the trusted computing base. SGX [138, 71, 139, 140], an extension to the Intel Architecture, offers confidentiality and integrity guarantees via a trusted processor without requiring any trust on the part of infrastructure software.

Multiple secure systems have been recently built on top of these trusted hardware platforms [12, 88, 111, 11, 135]. VC3 [12] and Opaque [111] offer SGX-protected data processing platform, assuming that, the code executing inside each enclave is trusted. Thus, their confidentiality guarantee is based on the assumption that enclave code does not leak secrets. In these cases, PrivacyScope can be used to confirm this assumption. Ryoan [11] and Chiron [135] utilize sandboxing to prevent untrusted enclave module from leaking secret data from side channels. To be adopted, these methods require a certain level of trust to be established between users, service providers and their solution. PrivacyScope can strengthen users' trust in these scenarios. All of these methods require a modicum of trust between the user and the trusted computing platform. PrivacyScope is designed to address this concern in the Intel SGX architecture.

4.7.3 Privacy Leakage in Machine Learning

Using ML to train models on big data poses many privacy challenges. ML models can uncover and expose surprising and unexpected personally identifiable information such as relationships and associations violating privacy. [141, 142, 143] take a first step to conceptualize privacy in the new era and enforce data use rules through designing new policy specification languages and corresponding enforcing systems. [142] creates a language for specification of origin-based privacy rules and implements a prototype of a type system to enforce such policies. In [143], the authors present LEGALEASE - a language to specify privacy specifications and restrict how private data must be handled. [141] presents Thoth which provides data use policies enforcement through a kernel-level compliance layer. PrivacyScope can integrate such policies and enforce more rules other than nonreversibility in the future.

4.8 Discussion and future work

4.8.1 Covert and Side Channels

In this section, we briefly highlight potential covert channels that can be exploited to leak private data in the Intel SGX computing environment. Each covert channel compromises PrivacyScope's security goals. We will address mitigation techniques to safeguard against these attacks in future works.

- *Timing channel*: Malicious actor can infer secret through recording the time spent for program executions. PrivacyScope can be extended to simulate the execution time for program paths and detect if execution time depends on secret in the future.
- *Probabilistic channel*: Malicious actor can infer secret through observing probability distribution of declassified data.
- *Power channel*: Malicious actor can measure the power consumption cost for program executions and infer secret.

PrivacyScope does not address side channels brought by hardware limitations on Intel SGX processor itself. We consider these limitations orthogonal to PrivacyScope and we list them as following.

- *SGX page faults*: Privileged software, e.g. OS or hypervisor, can maliciously control page tables of an enclave to observe a memory access pattern of enclave program execution.
- *Cache timing*: Side channel exists for two processes running on the same core. These two processes can use cache timing to infer knowledge of the other.
- *Address bus monitoring*: While all processed data is encrypted prior to exiting the SGX processor package, a malicious user using sniffer or a modified RAM chip can monitor the address bus and achieve a memory access pattern side channel.

4.8.2 Prior Knowledge on User Data

When the adversary has prior knowledge of user private data (e.g. knows the distribution of variable values), PrivacyScope requires that knowledge to be incorporated in the model specification to ensure the soundness of the privacy leakage analysis. For instance, given the function $F(A, B) = A + B$, where A is a privacy-sensitive scalar variable and B has a value of zero 99% of the time, and 1 otherwise. Then the attacker can conclude with a high degree of confidence (i.e. 99%) that the output value is the same as the value of the privacy-sensitive input variable A. To mitigate this problem, the users of PrivacyScope are required to incorporate that knowledge by extending PRIML language and the analysis engine so that it can handle with new semantics.

4.8.3 Limitations and Future Work

PrivacyScope is built on top of symbolic execution engine and symbolic execution is known to have limitation on scalability. Although enclave code does not have large size, they will become larger in the future. Also, our design needs enclave writer and cloud service provider to send user the source code of enclave for validation. This may hurt the intellectual property of cloud service provider. In the future, we plan to analyze on binary enclave code directly instead of C/C++ source code. This would protect the intellectual property of cloud service provider.

4.9 Summary

In this work, we formally define new nonreversible property on top of classical noninterference property, which is more suitable for machine learning programs. Using our newly proposed language, PRIML, we describe our innovative program analysis approach using formal semantics. PRIML's formal semantics can be extended by users who wish to introduce their own specialized notion of nonreversibility. We design and implement PrivacyScope, a prototype static analysis tool that implements the rules as defined in PRIML to detect nonreversibility violation in programs executing inside Intel SGX enclave. We show the efficacy of our prototype by applying PrivacyScope to find sensitive data leakage and maliciously embedded code in open source machine learning programs.

Acknowledgment

This work was supported in part by the Office of Naval Research under grant N00014-19-1-2621, US National Science Foundation under grant CNS-1837519, and Virginia Commonwealth Cyber Initiative.

Chapter 5

Protecting the Physical World

5.1 Continuous Attestation for Unmanned Vehicles

With the breakthroughs in wireless and battery technology, networked CPS are increasingly integrated into our society nowadays. From the perspective of safety and security, CPS is fundamentally different from the traditional IT system due to its ability to affect the physical environment. Among these CPS, unmanned vehicles are particularly interesting due to their ability to move in the physical space, this ability breaks the assumption on physical perimeter safety made by the system.

While traditionally only available in military applications [144], unmanned vehicles are increasingly common in our daily lives, and is expected to be fully ubiquitous available in our household and around the blocks [145]. Several companies have started commercializing drones to for package and pizza delivery [146], or even to transport passengers [147]. When these systems fail, human lives can be at stake rather than financial lost.

Recognizing the importance of unmanned vehicles, the security of these CPS has attracted significant amount of interest. The crucial question in this topic is that whether a unmanned vehicle is trustworthy. To answer this question, we have to satisfy the following two requirements. First, we need to be able to verify the integrity of its current system running state. Second, we have to know the continuous status that this unmanned vehicle has experienced during a mission. Take food delivery as an example, we need to first know that the unmanned vehicle does not contain malicious logic to attack people and second, we need to know that it is not compromised during its way and potentially has changed the food.

Remote attestation is a powerful security service for verifying the integrity of certain runtime property of a remote system. It is composed of two parties, the verifier and the prover. The verifier can verify the integrity of the prover. Through verifying the runtime integrity of the prover [18, 19, 20], existing remote attestation can satisfy the first requirement. However, existing remote attestation solutions can not satisfy the second requirement. Because one of the fundamental drawbacks in existing remote attestation schemes is that remote attestation allows the verifier to assess the prover system for an instance of time. While the common assumption behind attestation is that if the system has an initial trusted state, and with sufficient isolation and protection algorithm, the system will stay secure. However, this gives rise to a well known problem in remote attestation with the time of check versus time of use attack (TOCTOU), where programs are compromised between the attestations. With recent attacks such as water holing [148] attacking and ROP, attacker can gain and maintain control of the target system not only without changing the static code pages that the attestation routine verifies but also maintain access without leaving any footprint. Thus, existing remote attestation cannot attest on recent past that a unmanned vehicle experienced.

In order to tackle this problem, we desire a continuous attestation system. One straightforward way to build continuous attestation is for the prover to record its control flow sequences continuously, then it presents verifier the report when prompted [18]. As a result, the control flow sequence between attestation would serve as the evidence for behavior between attestation, therefore achieving the the continuous attestation. However, we identify three problems of this straightforward plan. First, the performance overhead is prohibiting. In our preliminary experiment with the ArduPilot [149] control system, the performance overhead for such control flow recording is as high as 200 times. This heavy burden overwhelms the ArduPilot scheduler, breaks the assumption of real-time constraints and leads to a unresponsive system. Second, the constant recording of control flow sequence generate complex trace information. In order for a verifier to verify the integrity of this complex trace, it must maintain a huge database and conduct heavy-weighted computation. As the continuous attestation time span grows, this problem grows exponentially. Third, the control flow sequence cannot defend against data-only attacks like non control data attacks. [20] propose to use def-use data flow on manually annotated critical values to solve this problem. However, this further increases the performance burden on runtime.

To solve the aforementioned challenges, we propose ConAttest. ConAttest is the first continuous attestation scheme for unmanned vehicles. In ConAttest, we bring in the ideas in the memory isolation research area [150]. In the memory isolation research area, they focus on the compartmentalization of the software. By separate the memory into segments for different modules of the software, they can quarantine the vulnerability in one module to prevent it from affecting other modules. And the transition between memory view of different modules is called view switch. This way, the power of the data-only attack is shrunken into a quarantined area instead of the whole system. In ConAttest, we also take advantage of compartmentalization to prevent against data-only attack. Other than that, we extend compartmentalization and propose a new runtime property, view switch sequence, for the purpose of continuous attestation. ConAttest uses view switch sequence as an indicator of the continuous behavior integrity of an autopilot system. We call this new runtime property View-Flow Integrity (VFI). VFI requires significantly less amount of performance overhead to achieve, and it can detect malicious control flow attacks or data-only attacks across modules by comparing VFI to its reference model. Note that, we support control flow recording for each module in ConAttest. If a user is concerned about particular highly vulnerable module, she could choose to integrate control flow sequence attestation for that particular module. By offer this option, ConAttest brings a tradeoff between fine-grained control flow monitoring and coarse-grained view switch monitoring. This enables a trade-off between mission time-scale and behavior monitoring granularity, which contributes to the continuity goal.

To enable VFI, ConAttest inspects the source code of an autopilot system and identify vulnerable modules and critical modules. ConAttest takes a description of compartments and uses LLVM to automatically separate the modules within an autopilot system. After the separation, ConAttest instruments the modules with a jump to a ViewSwitch trampoline whenever the module is about to transmit the control to or read the global data from another module. The ViewSwitch trampoline traps into kernel space and changes the MMU for access control of page tables. At the same time, it traps into secure world and records the view switch activity inside the secure world. This way, ConAttest enables VFI attestation by sending the VFI report stored in the secure world to a remote verifier.

Furthermore, we include a secure timer design in our ConAttest to satisfy the continuous requirement. As described in [151], attestation can be conducted probabilistically. This creates a trade-off between infection detection rate, trace record generation, and energy consumption. We create a secure timer to enable the self-initiated feature. This allows our system to create the same trade-off for reducing energy consumption and relieve the verifier to verify on the over-complicated report. This further approaches the continuity goal by providing another tradeoff between mission time-scale and infection detection granularity.

In summary, ConAttest provides a continuous attestation capability to autopilot systems. This requires tackling several challenges that form our individual technical contributions:

- We provide customizable compartmentalization and memory isolation for autopilot software. This feature adds another defense layer against runtime attacks.
- We propose VFI property to accomplish continuous attestation. VFI allows us to continuously monitor system running state to detect across view runtime attacks.
- We propose and implemented a secure timer design to support self-initiated runtime property recording. This allows us to tailor the security-energy trade-offs for a different level of infection detection rate.
- We port ARM Trustzone to a commercial drone platform, Navio2. And we implement ConAttest prototype on Navio2 platform and conduct evaluations.

5.2 Background

5.2.1 Compartmentalization

Memory isolation is a fundamental and powerful security property. It ensures that no application can have read, write or execute permission on memory locations outside of its own assigned area. In modern systems, the Memory Management Unit (MMU) is provided for high-end platforms and Memory Protection Unit (MPU) is provided for embedded devices to support memory isolation. However, existing autopilot systems do not take advantage of the memory isolation feature. One reason is that most autopilot systems aim at working on an embedded system. In order to ease the porting effort among embedded systems, autopilot systems tend to not use the advanced memory isolation provided by a high-end desktop system. The other reason is that current autopilots are developed without a focus on security. Lack of memory isolation makes protecting the autopilot system even harder. Because now all code can access all data and peripherals without any active mitigation. And a runtime exploit is able to access every single component on an autopilot system.

A line of research [150] tries to tackle this issue. Their main idea is to partition the software into several segments and put each segment into an allocated location. During the executing of one segment, it has no or limited access permission on other segments. [150] starts with analyzing the source code of an embedded application and identifying code and data that should be put into the same segment. This way, the transition between segments can be reduced and this mitigates runtime overhead. View switch instrumentation is added to support secure view switch among segments. Fig. 5.1 illustrates the memory view during the execution of each segment.

In ConAttest, we integrate the idea of memory isolation and view switch instrumentation. Memory isolation is proven to be effective against data-only attacks and a continuous record of view switch activity is used to prove the running state of a system for a continuous time span. The runtime overhead of switching between segments depends on the number of segments and the frequency of segment switch. This brings us the capability to tailor the security-performance trade-off and also the opportunity to optimize the system.

5.2.2 Runtime Attestation

Software remote attestation can be divided into two categories, static remote attestation and runtime attestation. Conventional static attestation approaches targets at proving that the software under attest loaded is unmodified during its initialization [96, 97, 94, 95, 84, 152, 153, 91, 154, 155, 156, 90, 85, 83, 157, 158, 159]. However, static attestation cannot detect runtime attacks including code-resuse attacks [18] and data-only attacks [20]. Recent progress in remote attestation aims at tackling this issue by attesting on runtime properties.

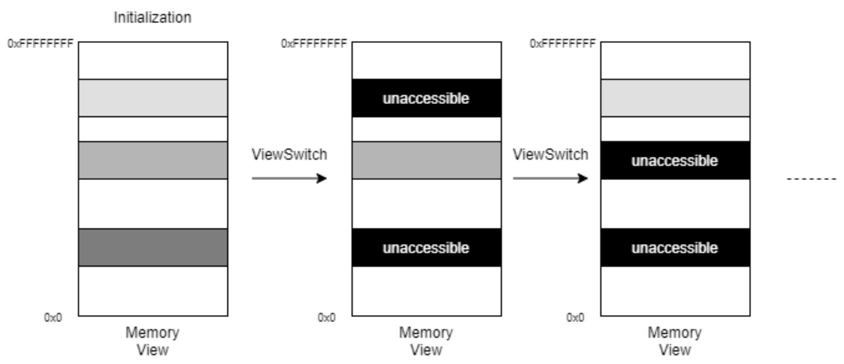


Figure 5.1: Illustration of memory view switch process.

[18] is the first work working on the runtime property attestation track. They propose to use the control flow sequence of the prover program as a runtime property to be attested. The main idea is to record the control-flow paths during an attestation snapshot. As we know, the program is composed of the control-flow graph (CFG). Each CFG node is a basic block which contains a set of assembly instructions. For each basic block, it ends with a branching instruction which transits the control flow to another location instead of going to the next instruction. [18] instruments the binary to record each control flow transition and includes ARM Trustzone to safely store the control flow path record for future reporting. This control flow attestation allows a remote verifier to detect runtime attacks based on code-reuse techniques. However, the main drawback of this method is the scalability and incapability of detecting non-control data attacks.

Following this research direction, [20] is the most recent work. It tries to tackle the aforementioned two drawbacks by proposing operation-scoped control flow integrity and critical variable integrity. [20] separate the functionality of a program into operations. They narrow down the full program control flow monitoring to operational grade. This way, the scalability issue can be mitigated. They also allow a developer to annotate critical data values in an operation. This critical value may be a value that is critical for physical operation, for example, to what degree a robot arm moves. By adding instrumentation to enforce define-use consistency for critical value, they can detect data-only attacks. However, this method requires manual annotation from the developer side and requires rewriting of software.

Existing attestation schemes share common characteristics. They aim at attesting either the load time integrity or runtime integrity of a remote system at a snapshot. They do not consider the integrity for a time scale of a mission for a CPS. With the advance in unmanned vehicles, future applications rely on a CPS to finish a mission, for example, unmanned Uber vehicle pickup. The verifier not only needs to attest the current state of a system but also a continuous state of a system. ConAttest bare this in mind and targets at bringing this capability.

5.2.3 ARM TrustZone and Attestation

ConAttest depends on ARM TrustZone as the Trusted Computing Base (TCB). TrustZone is an optional security feature for ARM architectures. It is a hardware security extension and it includes support from ARM processor, bus fabric, and system peripherals on a System on Chip (SoC). TrustZone is available on both Cortex-A processors (for mobile phones and high-end embedded systems) and Cortex-M processors (for low-end IoT systems). The high-level idea of TrustZone is to partition the whole SoC into two worlds: secure world and normal world. In a typical use case of TrustZone, the normal world will run normal applications and OS as usual like Linux kernel, while the secure world runs security-related applications or a tiny trusted OS. The two worlds run in parallel. The secure world contains a small code base and has a higher privilege level than the normal world. The normal world is restricted by hardware barriers from accessing resources allocated to the secure world like tagged caches, banked registers, and secure peripherals. Whenever a normal want to access the functionalities provided by the secure world, it needs to use special instruction to trap into the secure world and then secure world responses to the requests.

ConAttest uses OPTEE as a trusted OS in the secure world. A typical use of OPTEE is to design and implement code in the normal world called client application (CA) and code in the secure world called trusted application (TA). CA is responsible for invoking the TA in the secure world, while TA is responsible for the execution of any security-critical tasks. In ConAttest, the VFI measurement engine is implemented as a TA in OPTEE. Thus, the record of VFI is protected from the potentially malicious normal world. Another security feature of TrustZone is that it provides provisions of per-device private keys and trusted certificates. This feature allows us to perform straightforward authentication and secure communication between a TrustZone equipped prover device and a remote verifier.

5.2.4 Ardupilot

ConAttest use an autopilot software called Ardupilot [149] for evaluation. Ardupilot enables the creation and use of unmanned vehicle systems. Coupled with ground control software, unmanned vehicles running Ardupilot can have advanced functionality including real-time communication with operators. Ardupilot supports Copter, Plane, Rover, Submarine, and Antenna Tracker. Ardupilot software is composed of three parts, shared library, vehicle-specific flight code, and hardware abstraction layer code. During compilation, external libraries like MAVLink is also linked into the final binary. MAVLink library is used for parsing MAVLink messages sent and received from the ground station. In ConAttest, we consider external libraries untrustworthy, especially the MAVLink library. Since the MAVLink library can receive messages through radio remotely. Meanwhile, the trustworthiness of other libraries and code is customizable by the user.

Two essential differences between CPS and other applications are that: First, they have strict real-time constraints; Second, they are composed of control loop tasks. In a typical CPS, control tasks at various time-scale are implemented to interact with the physical world. And for each control task, it usually has a strict time constraint on when it should be run again. Ardupilot also has this CPS property. Ardupilot uses a setup-loop structure to implement control tasks. During the setup phase, each control task is assigned both a frequency and a maximum execution time. During the loop phase, the frequency is used in timer callback to decide if that control task runs when it is called. The maximum execution time is used by the scheduler of Ardupilot. If the remaining time of a loop is less than the maximum execution time, the processing task won't be executed and will be delayed to the next loop. However, because of this design choice, Ardupilot becomes unresponsive if it cannot finish all tasks in a loop in most cases. In our preliminary experiment on adopting the control flow recording of Ardupilot, the system stops responding. This is because the runtime overhead overwhelmed the system and it cannot finish it control loop tasks in time. Such a situation further motivates the necessity of VFI.

At runtime, Ardupilot is a single process. Depending on the board it is built for, Ardupilot may support single or multithreading (Linux and PX4). However, in Linux, all threads of a process share the same memory view. So compromising any control task inside any thread can lead to accessing all code and data of all control tasks. Such circumstance motivates the importance of memory isolation in Ardupilot software runtime. In ConAttest, we support user-defined compartments to enable a different degree of memory isolation. Note that, ConAttest is built on the Linux system and can take advantage of MMU, while the methodology of ConAttest can also be applied to a system with MPU and trusted hardware, for example, new ARM Cortex-M chips with MPU and TrustZone.

5.3 Problem Setting for ConAttest

Runtime attacks exploit vulnerabilities in software during its execution. A vulnerability is an exploitable bug hidden in the software during its development. The prerequisite of runtime attacks is an exploitable bug in programs. Thus, one way to defend against runtime attacks is to create a bug-free program. However, the software is developed by human beings, and human makes mistakes. Static and dynamic tools are developed to help humans find their mistakes. static analysis tools like Model checking and Symbolic execution are developed to find bugs, also dynamic tools like fuzzing help. However, static methods have a state explosion challenge and usually have to make a trade-off between soundness and completeness. While dynamic methods have the challenge of exploring all the possible inputs to cover all execution states. A large program could have close to infinite input space, and dynamic methods have trouble in traversing all input space. How to develop the bug-free program is still an open research problem.

Bugs seem to be inevitable in the program. Another direction is to investigate on runtime attacks mitigation methods. Memory isolation is one of the most common and effective techniques among computer system security services and can mitigate runtime attacks by separate the memory space of the program. For example, process memory isolation isolates each process's memory space from other processes. If an attacker has compromised one victim process, she cannot use this victim process to launch an attack on other processes according to this memory access control. Efficient in-process memory isolation is a recent topic, [160] uses Intel MPK to facilitate such capability. Runtime attacks mitigations in the same memory space are also developed to make a bug harder to exploit for an attacker. Basic mitigations include Data Execution Prevention (DEP) to stop code injection, Stack Canaries to stop stack smashing, and Address Space Layout Randomization (ASLR) to probabilistically make attacks harder. However, they can be bypassed through information leaks or code reuse attacks. Advanced mitigation such as Control-Flow Integrity (CFI) restrict the control flow to only jump to allowed target addresses for indirect control-flow transfers. But it has its own pitfall. First, CFI is stateless, thus an attacker can choose any of the targets in the allowed ones for each dispatch. Second, data-only attacks allow an attacker to affect the program within the allowed control flow. [18] use the control flow sequence to provide states in control flow and use it for attestation. Their method subjects to non-control data attacks.

In our perspective, instead of detecting runtime attacks in single memory space, we focus attacks that cross memory isolation boundaries to achieve their goal. These attacks include control-flow hijacking, code reuse, and data-only attacks. Fig. 5.2 shows a generic boundary-crossing runtime attack example on a program. In this graph, there are five basic blocks (i.e. the circles). B_x is a function that writes on target data. B_2 contains an arbitrary write vulnerability and B_3 contains a control-flow vulnerability. The attacker's goal is to write on target data (shown in the black box). The attacker can achieve it by exploiting the control-flow vulnerability in B_3 to call B_x and indirectly writes on target data, or he can use arbitrary write vulnerability in B_2 to directly write on target data. Note that, both these two ways need to cross the memory isolation boundary. This memory boundary crossing activity alone does not indicate malicious activity. However, this view switch activity in a continuous record of view switches indicates that an attack is launched. This is similar to statelessness in CFI comparing to statefulness in control flow sequence attestation [18].

5.4 Threat model and Assumptions

We assume the platform contains trusted hardware like ARM TrustZone. We assume the code running inside secure world is trustworthy and we assume the attackers cannot launch hardware attacks to break the protection of TrustZone. We assume the platform is equipped with MMU or MPU to support hardware-based memory isolation. We assume the platform contains a secure timer in TrustZone. We assume the autopilot software is running in a single address space. We assume the availability of autopilot source code. ConAttest performs analysis and instrumentation on source code. We assume that the autopilot software itself is

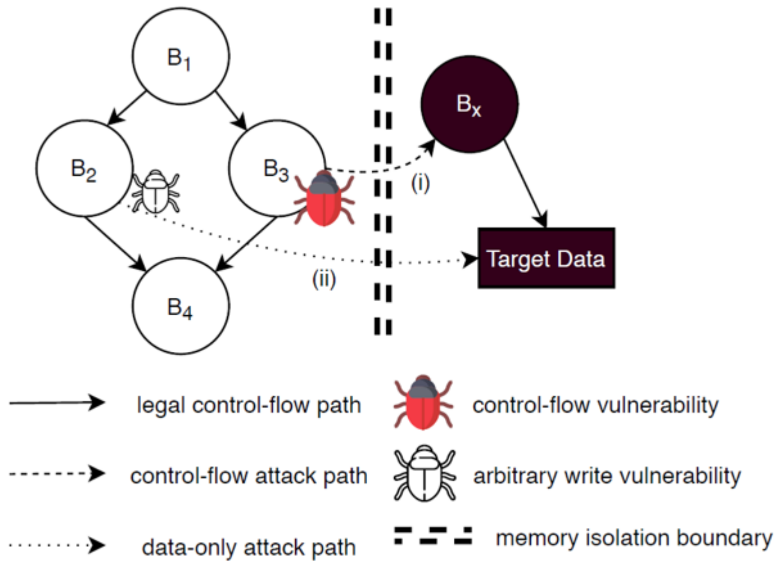


Figure 5.2: Abstract view of boundary crossing runtime attacks.

trustworthy which means it may contain exploitable bugs but not malicious. We assume that attackers could find a memory corruption vulnerability in any of the compartments of the autopilot software except security-critical compartments and they are capable of exploiting memory corruption. Once the attackers have successfully launched their exploit, they can further reuse existing code in the compartment to achieve their goal. We assume the attacker can also launch a data-only attack. Since the autopilot only has one address space, without ConAttest, the attackers can maliciously utilize the code, data and peripheral devices mapped to the memory space. We assume the attacker's goal is to temper with security-sensitive data in a security-critical compartment. We assume the attackers cannot tamper with the instrumented ViewSwitch trampoline. We assume the attacker cannot dynamically inject code. These can be achieved by taking advantage of MMU or MPU.

ConAttest applies defenses to (1) isolate memory corruption vulnerability in a compartment and prevent it from affecting the entire system: (2) record the view switch activity continuously to allow a remote verifier to detect malicious system state for a recent past. (3) launch self-initiated view switch monitoring with the help of a secure timer to achieve practical continuous VFI attestation.

5.5 ConAttest Design

ConAttest has one main goal which is to enable continuous attestation. To achieve this goal, ConAttest brings in two essential mechanisms. The first one is a new runtime property VFI. VFI introduces dramatically lower runtime overhead comparing to control-flow runtime property. The runtime overhead of VFI also depends on the granularity of compartments. Users can further reduce the runtime overhead by tailoring the partition of software. The second one is secure timer enabled self-initiated VFI monitoring. This security feature brings in a trade-off between attack detection rate versus energy consumption and verification difficulty. Another goal of ConAttest is to defend against code reuse attacks or data-oriented attacks across compartment boundaries. To accomplish this goal, ConAttest separate the compartments' data and code segments in the memory map and enforce that each compartment can only access its corresponding code and data during execution. If a compartment wants to cross the boundary, it is trapped into ViewSwitch trampoline and ViewSwitch trampoline switches memory view securely and tells ARM TrustZone to store this activity.

5.5.1 ConAttest System Model

Fig. 5.3 shows our system model. The verifier is pre-distributed with a database of valid VF paths. The prover is equipped with ARM TrustZone and a secure timer. We consider the case when a drone flies out of its base and reach proximity to the verifier. The verifier wants to attest the drone for if it has been compromised during its flight. The flight could easily last from minutes to hours.

In ConAttest, after the drone takes off, VF recording is initiated by a secure timer according to the user-defined parameter during the mission (①). The prover starts monitoring the VF path of autopilot once initiated (②). After a while, the secure timer times up and fire an interrupt to ask prover to stop recording and generate a hash of current recorded VF path and store it in secure storage (③ and ④) . The $VF\text{-}Path_i$ is accumulated into an authenticator *Auth* by hashing the view switch activities (④). During the flight, the secure timer continuously triggers the prover for the aforementioned activity. Once the flight is finished and the drone reaches its destination, the verifier starts to attest the continuous VFI of the autopilot software on a drone. She sends a challenge c that includes the autopilot software ID and a nonce to ensure freshness to prover (❶). Then the prover replies all the reports generated by stored VFI hash starting from the takeoff (❷ and ❸). Prover generates the attestation report $r = Sig_K(Auth, c)$ by signing the previous authenticator *Auth* and received challenge c with a key K known only to the prover (❸). In the end, the verifier queries its local valid VF database and verifies all r_i she received from the prover. The verifier only trusts the drone if all the queries succeeded (❹). The valid VF database is generated offline by measuring each possible VF path. This needs to be done once per autopilot software. We delegate this task to the vendor of the autopilot software. We assume the verifier receives the database from a trusted source.

Note that we are aware that exploration of all possible execution paths of a generic program is an open problem. However, in this paper, we are focusing on autopilot software. An essential property for autopilot software is that it is composed of control loops. And their behavior tends to have a periodic pattern. In this case, the amount of valid VF path for an

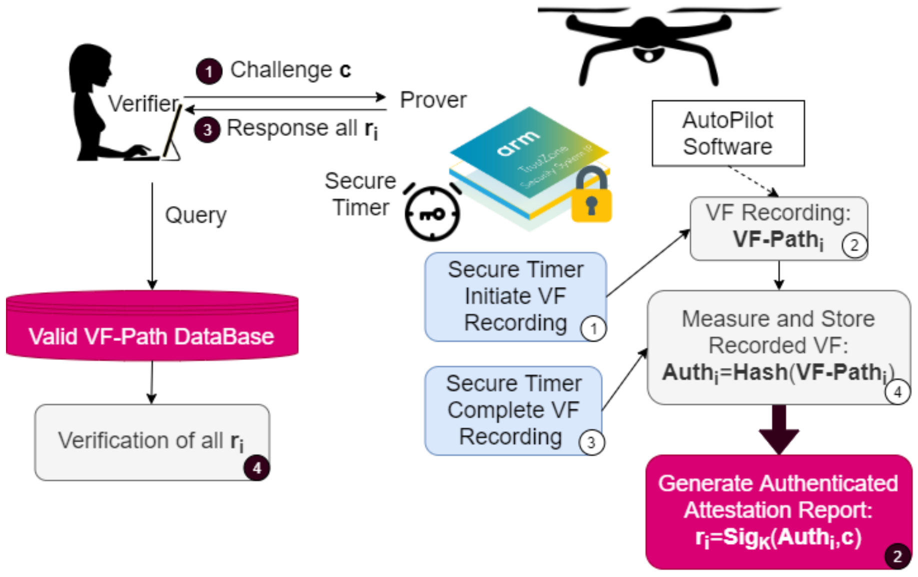

Verifier
1 Challenge c
Prover
3 Response all r_i
Query
Valid VF-Path DataBase
Verification of all r_i 4
Secure Timer
AutoPilot Software
VF Recording: VF-Path$_i$ 2
Secure Timer Initiate VF Recording 1
Measure and Store Recorded VF: Auth$_i$=Hash(VF-Path$_i$) 4
Secure Timer Complete VF Recording 3
Generate Authenticated Attestation Report: r_i=Sig$_K$(Auth$_i$,c) 2

Figure 5.3: Overview of ConAttest

autopilot software is limited.

5.5.2 Compartmentalization

In this paper, a compartment is defined as an isolated code and data region. Each instruction and data belongs to exactly one compartment. Fig. 5.4 shows three compartments, where each compartment has read/write permission on its data region and read/execute permission on its code region. When a compartment tries to call a function or access data in another compartment, a view switch activity is performed. For example, as shown in Fig. 5.4(b), MsgRec() is a function in compartment B for receiving the message. Here, compartment B is the library for processing messages. When MsgRec() receives a command from the ground station to fly higher, it first enters compartment A, which is a library for status estimation and flight control. Then EKF() function takes over the control and starts estimating altitude. After that, EKF() pass control to RateCTL() to calculate motor rates. In the end, compartment C is entered and motor speeds up. This whole process includes the view switch from compartment B to compartment A and compartment A to compartment C. ConAttest securely record such view switch flow and use it as VFI runtime property for attestation purpose. ConAttest uses MMU to set permissions on each region while MPU could also satisfy ConAttest's requirement in theory.

The starting point of our workflow is to partition the autopilot software. We partition the autopilot software by its functionalities. Autopilot software is normally composed of four categories of libraries. They include hardware abstraction layer libraries for easier porting purposes, sensor libraries for extracting sensor information, core libraries for piloting and

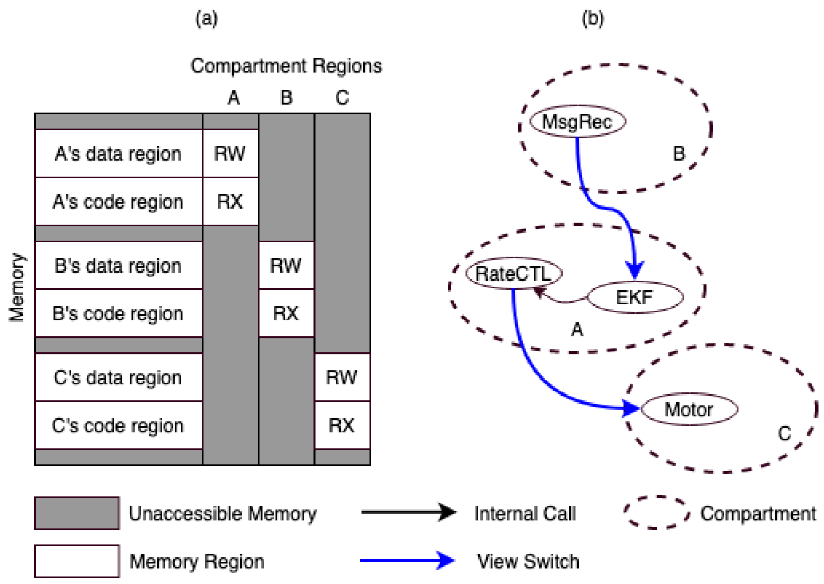


Figure 5.4: Illustration of ConAttest's concept of compartments. (a) ConAttest isolates memory with access permissions shown in columns. (b) ConAttest records view switch activities.

navigating, and communication layer libraries for sending and receiving messages to the ground station. Note that, communication layer library is usually an external project (e.g. MAVLink). ConAttest requires communication libraries to be put in a single compartment, while other libraries are optional and depend on the user's specification. We argue that external libraries may not be as trustworthy. ConAttest offers a compartment policy configuration feature to users. The compartment policy defines how libraries should be grouped into compartments. If a user has a security-sensitive code or data region, she can use the compartment policy configuration feature to group them into a single compartment and thus provides a higher security level for this region. The policy affects the performance and isolation of compartments which also means the security of the autopilot software. For example, if two code regions that frequently call each other are placed in different code compartments then view switch will occur frequently. From a security point of view, if a security-sensitive global variable is placed in the same compartment with an untrustworthy external library, then an activity of malicious attack on this global variable cannot be recording in VFI attestation. ConAttest enables the user to explore the performance-security trade-offs for meeting their requirements.

5.5.3 Program Instrumentation and View Switch

ConAttest relies on MMU to isolate the address space of individual compartments. During the linking phase, the linker maps compartments into the various memory region. The

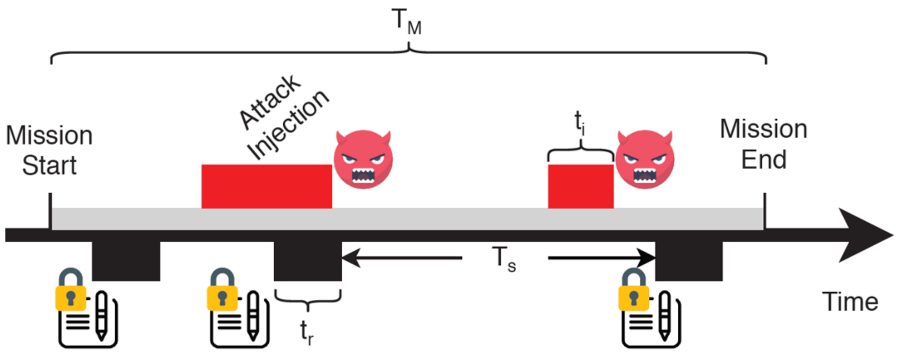


Figure 5.5: Randomized VFI measurement.

MMU is set up so that each compartment cannot access the memory region occupied by other compartments. ConAttest inserts trampoline code into autopilot software during the compilation pass to support view switches. Invocation to the trampoline code is instantiated by instrumenting the autopilot software. Every function call between compartments and every return instruction is instrumented to invoke a view switch routine. Also, every data access across the compartment is instrumented. Upon invoked, the trampoline code in user space traps into a kernel module. The kernel module configures the MMU properly and further trap into secure world. Then the measurement engine in the secure world is invoked and the view switch activity is recorded.

To instrument all function call and return between compartments, static program analysis is used to identify all the functions a compartment has. Then by finding out the functions who call functions in another compartment, the instrumentation on function calls can be achieved. For instrumenting return instruction, the call graph is utilized. If a function is not called by any function in another compartment, then the return instruction in this function does not need instrumentation. The return instruction needs instrumentation when the opposite happens. We use static analysis conservatively and leave issues like indirect function calls to dynamic analysis. The dynamic analysis gives only true view switch points with the trade-off that it needs to be determined during execution. We manually add instrumentation according to the results from dynamic analysis.

5.5.4 Secure Timer Initiated VF Recording

We propose a further trade-off of security by enabling probabilistic measurement with secure timer initiated VFI recording. We set a secure timer in TrustZone to randomly initiate the VFI recording during a mission, so an attack can be detected if, at that temporal moment, VFI recording is turned on. While when VFI recording is turned off, the attack activity is missed.

We also seek to answer the fundamental question in randomized continuous attestation, which is how to trade-off between security and energy. More specifically, the security is measured by P_d - the probability of detecting an intrusion and the energy can be measured by the additional energy, E_a, consumed by the device in recording VF. One of method to optimize for best utility is to formulate the utility as function combination two factors, where $\mathcal{U} = \mathcal{F}(P_d, E_a)$. More specifically, let T_M be the time for conducting a mission by an attested unmanned vehicle. Let T_s be the average time interval between two distinct secure-timer initiated randomized measurements. We assume during a secure timer initiated measurement, the average recording time is t_r. We assume within a T_M period, there are in average I non-overlapping injections and for each injection, each takes an average time of t_i to conduct. The overview of randomized continuous attestation is shown in Fig. 5.5, the injection would only be detected if it overlaps with randomized attestation in the time. The probability of injection detection P_d can be calculated as following: $P_d = \frac{T_s * \sum_I t_i}{T_M * t_r}$. The energy cost can be written as a function of mission time span, recording frequency, and average time for each recording where $E_a = \mathcal{F}_{E_a}(T_M, T_s, t_r)$. Then to formulate this problem is to examine the probability of detecting an adversary per energy cost, thus the utility can be expressed as

$$\mathcal{U} = \frac{P_d}{E_a}$$

where

$$P_d = \mathcal{F}_{P_d}(T_M, T_s, t_i, I, t_r) = \frac{T_s * \sum_I t_i}{T_M * t_r}$$
$$E_a = \mathcal{F}_{E_a}(T_M, T_s, t_r)$$

5.6 Implementation of ConAttest

This section presents our prototype implementation of ConAttest on a Navio2 drone.

5.6.1 ConAttest Prototype

To demonstrate the efficacy of ConAttest, we prototyped it on a popular drone platform, Navio2. Navio2 platform includes two essential parts, a Raspberry Pi 3 (RPI3) board for running autopilot software (i.e. ArduPilot) and a Navio2 daughterboard for providing various sensing capabilities. The reason why we choose this platform is that RPI3 is equipped with ARM's TrustZone-A. ARM TrustZone-A can provide security services to meet the requirements of the trusted prover. Also, commercially available drones based on ARM TrustZone-M is not there yet. ConAttest can also be applied to future autopilot platform equipped with ARM TrustZone-M and MPU. On the Navio2 platform, a customized Real-time preemptible kernel based on Linux kernel v 4.14.95 is running on RPI3. Device drivers to communicate with Navio2 daughterboard is also included in the customized Linux kernel. The Linux distribution used is Raspbian GNU/Linux 9 (stretch) 64 bit. For assembling and soldering up the drone, we list the essential parts as following: 4S 6000mAh 14.8V 55C LiPo

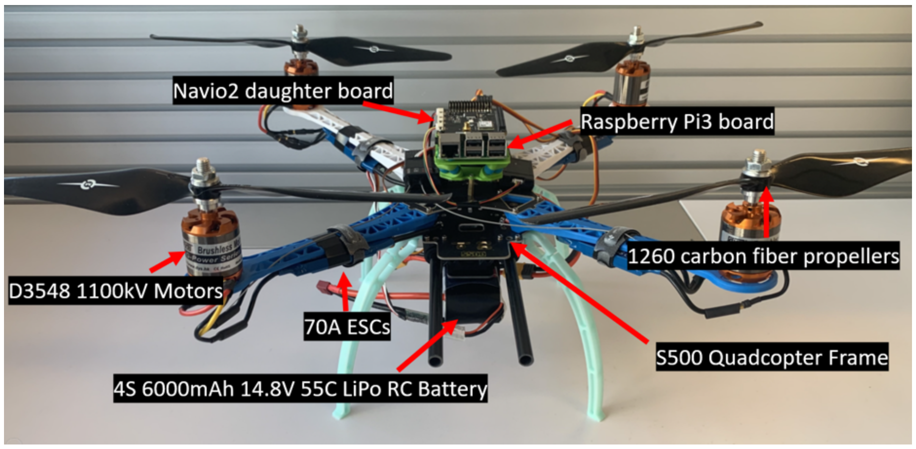


Figure 5.6: Our prototype of ConAttest implemented on Navio2

RC Battery, S500 Quadcopter Frame, 70A ESC, D3548 1100kV Motor and 1260 carbon fiber propellers. The reason we pick these parts to show is that designing a drone needs to consider lifting power versus weight. And the parts we show is crucial to this balance. And picking wrong parts may lead to a drone that can't take off. We show our prototype in Fig. 5.6. In addition, program analysis and program instrumentation are implemented as new passes in LLVM 4.0 [161]. Call graph analysis is implemented in Python leveraging the NetworkX graph library [162]. Trampoline is provided in the form of a C runtime library.

5.6.2 Porting OPTEE to Navio2 Platform

Although the RPI3 hardware offers the capability of ARM TrustZone, the original Navio2 platform does not utilize it. Thus, the first challenge for implementing ConAttest on Navio2 platform is to build up secure world. We choose to port a popular open-source trusted OS, OPTEE, to Navio2 platform. OPTEE is a Trusted Execution Environment (TEE) designed as a companion to a non-secure Linux kernel running on ARM Cortex-A cores using the TrustZone technology. OPTEE is a companion with ARM Trusted Firmware (ATF) which provides a reference implementation of secure world software for Armv8-A and Armv8-M.

In order to port OPTEE to RPI3 with Linux kernel, we first need to know the normal boot process of RPI3. There are some important files in the FAT32 filesystem on the SD Card. They are second-stage bootloader bootcode.bin, third stage bootloader start.elf, and a config.txt file containing configuration parameters for both the VideoCore and loading of the Linux Kernel (load addresses, device tree, uart/console baud rates and etc.). Note that, RPI3 has a VideoCore and an ARM processor. Starting from power on, the VideoCore is responsible for booting the system. It loads the first stage bootloader from a ROM embedded within the SoC. The first stage bootloader is designed to load the second stage bootloader (i.e. bootcode.bin). The second stage bootloader is executed on the VideoCore and loads the third stage bootloader (i.e. start.elf). The third stage bootloader is where all the action happens. It starts by reading config.txt. Once the config.txt file has been loaded and parsed, the third stage bootloader will load kernel image into the shared memory allocated to the ARM processor, and release the ARM processor from reset. The Linux kernel then starts booting.

To port OPTEE, we add a boot stage before the third stage bootloader loads kernel. First, we compile ATF, OPTEE, and U-boot. U-boot is an open-source, primary boot loader used in embedded devices to packaging the instructions to boot the device's operating system kernel. We combine ATF, OPTEE, and U-boot into a single image. ATF is the starting point of this image. And ATF starts its trusted boot process when it gets the control. After ATF finishes its own initial boot stages and set up of OPTEE, it passes control to U-boot for it to set up and load Linux kernel. Then we tweak config.txt to make it load this image instead of Linux kernel image. The OPTEE runtime footprint in memory is set up by trusted boot in this stage and the U-boot will load kernel image to memory and transfer control to the Linux kernel booting process. This solves the first challenge of porting OPTEE to the Navio2 platform.

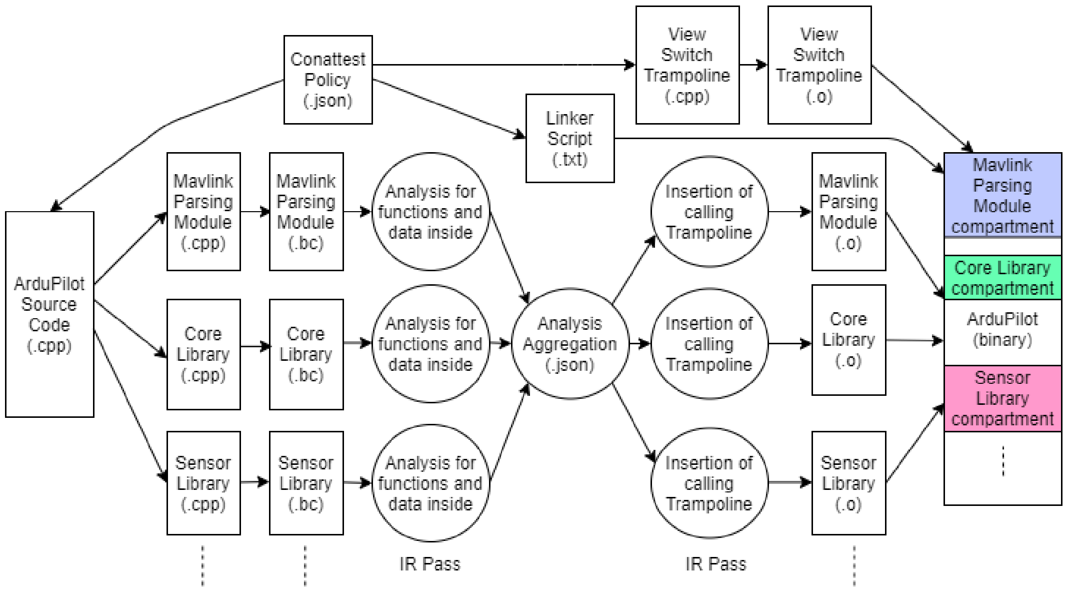

Conattest Policy (.json)
View Switch Trampoline (.cpp)
View Switch Trampoline (.o)
Linker Script (.txt)
ArduPilot Source Code (.cpp)
Mavlink Parsing Module (.cpp)
Mavlink Parsing Module (.bc)
Analysis for functions and data inside
Core Library (.cpp)
Core Library (.bc)
Analysis for functions and data inside
Sensor Library (.cpp)
Sensor Library (.bc)
Analysis for functions and data inside
Analysis Aggregation (.json)
Insertion of calling Trampoline
Insertion of calling Trampoline
Insertion of calling Trampoline
Mavlink Parsing Module (.o)
Core Library (.o)
Sensor Library (.o)
Mavlink Parsing Module compartment
Core Library compartment
ArduPilot (binary)
Sensor Library compartment
IR Pass
IR Pass

Figure 5.7: Overview of ConAttest development Workflow.

5.6.3 Enabling Secure Timer on Navio2 Platform

OPTEE does not support secure timer on RPI3 and RPI3 is not using Generic Interrupt Controller (GIC). All the GICv1 to v4 implementation in OPTEE does not work on RPI3. In this situation, we have to set up the interrupt control and secure timer in ATF and OPTEE.

So the second implementation challenge is to set up a secure timer during ATF trusted boot and implement the secure timer interrupt routing and handling in ATF and OPTEE.

The SoC model RPI3 uses is BCM2837, and the architecture of it is the same as BCM2836 [163]. From the Linux source of RPI3 using the bcm2836-l1-intc device tree, we know that the BCM2836 has a per-CPU interrupt controller for the timer. It is clear the next step is to find a secure timer source and route its interrupt line to secure world. So we need to first identify a secure timer source and second, set up interrupt routing.

For the BCM2837 SoC, RPI3 has a system timer (part of GPU core) and a local timer (part of ARM core), these two timers are implemented by sp804 module [164, 165]. Besides, an ARM clock is derived from the GPU core clock. RPI3 uses ARM Cortex-A53 which has four cores. And Cortex-A53 core timers can use external crystal timer or local peripheral (APB) clock source [166]. In our setting, Cortex-A53 core timers use an APB clock source. APB clock is half the speed of the ARM clock. Thus essentially, the GPU code can potentially affect the core timer. Note that, here we trust the GPU code since it is distributed by RPI3 vendor. A more secure way would be using a secure external crystal timer. For each Cortex-A53 core, there are 4 timers, an EL1 Non-secure physical timer, an EL1 Secure physical timer, an EL2 physical timer, and a virtual timer [10]. We are using EL1 Secure physical timer in our timer interrupt for self-initiated VF recording. Here EL means Exception level, and both the normal world and secure world have their own ELs. The higher the number, the more privileged the mode is.

Now that we have a secure timer source, we can now set up interrupt routing. RPI3 is not an open hardware platform, so it lacks of hardware documentation. We base our development on an unofficial document for the previous version of RPI [167]. The goal for us in this part is to set core0 EL1 Secure physical timer interrupt to the route as FIQ to core0 before control is transferred to the normal world. To achieve this, we conduct the following steps: ❶ Set core0 FIQ source to EL1 Secure physical timer of core0. ❷ Route EL1 Secure physical timer of core0 to FIQ interrupt type. ❸ Set counting time for EL1 Secure physical timer of core0. ❹ Enable EL1 Secure physical timer of core0. Then we set up the secure timer interrupt handlers. For each interrupts coming, depends on which world a core is working in, it has a different outcome. For FIQ in the normal world, it traps into secure EL3 and forwards to secure world. For fiq in secure world, the secure world handles it directly. In our case, we write a handler in both secure EL3 and secure EL1.

5.6.4 Compartmentalization Development Flow

As shown in Fig. 5.7, the first step of development flow is to create a user-defined ConAttest Policy file. This file specifies which libraries or source code should be put into the same compartment and where the memory region is for each compartment. This file is used for partitioning the ArduPilot software, generate linker script and supply compartment meta information to ViewSwitch trampoline. In the next step, each compartment is compiled into

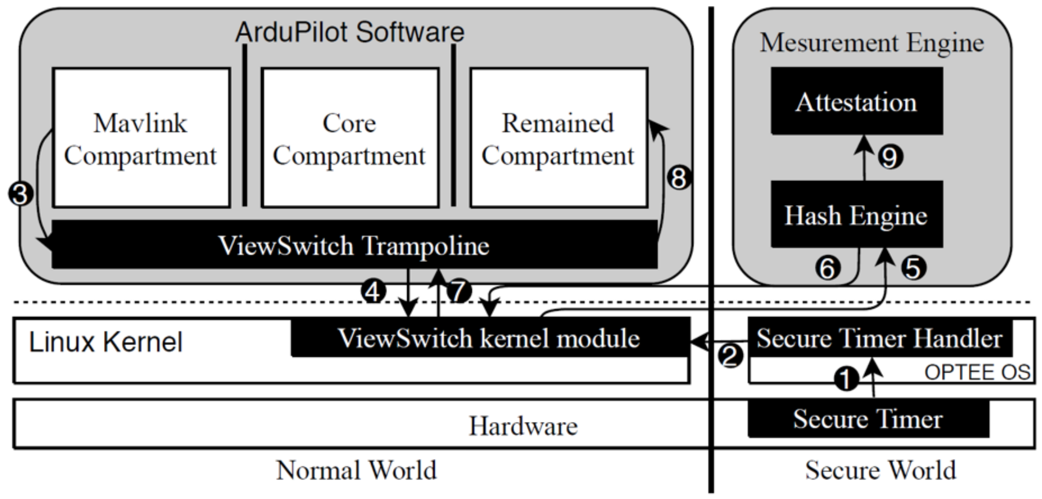


Figure 5.8: Overview of ConAttest runtime.

LLVM bitcode file. Our first IR pass performs program analysis on each compartment's bitcode file. It extracts each function and data names in each compartment. The Analysis aggregation step is an aggregated JSON file with function and data name information for all compartments. Our second IR pass takes in the JSON file created in the previous step and inserts calling to trampoline into each compartment's bitcode file before direct function calls to another compartment. We notice that indirect function calls lead to segment faults in our case. So we manually identify and fix the segment faults through dynamic analysis later. For the return instructions, we use NetworkX to analyze the call graph generated by LLVM and insert the trampoline as described in Section 5.5.3. After the instrumentation, compartments are compiled into object files. In the meantime, ViewSwitch Trampoline is also compiled to object file. ViewSwitch Trampoline traps into kernel space when a view switch is needed. A linker script is used to link all the compartments' object files and the view switch object file to a single binary. The linker script tells the linker to put each compartment into its assigned memory region.

5.6.5 ConAttest Runtime

Fig. 5.8 shows the runtime for ConAttest. Note that we omit the communication software and procedure for the verifier to connect to the attestation module in the measurement engine in this figure for simplicity. During the flight, a secure timer initiates a runtime VFI trace request from time to time. It does so by generates a secure timer interrupt (❶) and the handler for this interrupt in OPTEE OS would pick it up, clear the interrupt flag, set the finish recording time, and turns on the VFI recording switch in ViewSwitch kernel module (❷). Then the ViewSwitch kernel module starts to monitor view switch activity in ArduPilot software. Within the ArduPilot, for each compartment crossing activity, the software instrumentation calls into ViewSwitch Trampoline (❸). The ViewSwitch Trampoline then traps into the ViewSwitch kernel module (❹). If the recording switch is turned on, then ViewSwitch kernel module traps into secure world and pass view switch activity data to Hash engine (❺) through OPTEE OS besides configuring MMU. After the Hash engine hash the view switch activity data with existing data, it passes the control back to OPTEE OS which in turn returns control back to the ViewSwitch kernel module (❻). In the end, the ViewSwitch kernel module passes control to the ViewSwitch trampoline (❼) which then gives control back to the target function in the target compartment (❽). Right before the drone arrives at its destination, the verifier securely connects to the Attestation module in the measurement engine and requests for VFI attestation report. The attestation module extracts the reports generated by the Hash engine along the way (❾) and sends it back to the verifier through a secure channel.

Besides, we also support control flow monitoring in our measurement engine. We allow a user to optionally specify modules they want to monitor control flow on. We use CSI framework [168] to instrument selected modules and allows them to trap into measurement engine and record the control flow similar to [18]. We highly suggest to only use this feature for security-critical tiny modules, because it generates large runtime overhead on the ArduPilot software and requires large database and computation power for the verifier.

5.7 Evaluation on ConAttest Prototype

In this section, we evaluate ConAttest prototype on the Navio2 platform to answer the following questions:

- What is the performance impact of control flow instrumentation on real time constraints of real-time ArduPilot software in Sec. 5.7.1?
- What is the performance impact of ConAttest on real time constraints of real-time ArduPilot software in Sec. 5.7.2?

5.7.1 Performance Impact of control flow instrumentation

As a concrete demonstration of the negative impact of control flow instrumentation to ArduPilot software runtime, we use [168] to implement control flow instrumentation in the software. More specifically, we randomly selected a number of real-time tasks in ArduPilot and measured the impact of control instrumentation on their real-time constraints. Our control-flow instrumentation traps into secure world and uses a trusted application to do the CF recording.

From our preliminary result, the control flow instrumentation increases the runtime overhead of ArduPilot software to an average of 168 times. We observed that the control flow instrumentation severely break the real-time constraints of the system. And the system becomes unresponsive with control flow instrumentation. This motivates the necessity of ConAttest.

5.7.2 Performance Impact of ConAttest

Fig. 5.9 shows the performance of 12 real-time tasks in comparison with their corresponding real-time constraints, with and without the instrumentation added by ConAttest. We utilize the high-resolution performance counter on the RPI3 hardware to generate the measurements. As described in Sec. 5.2.4, the main thread of ArduPilot software sets up the scheduling, and all tasks are assigned a deadline real-time constraint. If the deadline real-time constraint is not met, then the system could potentially become unresponsive. This is because not all tasks can be processed. Thus, the performance overhead of ConAttest must be small enough for real-time tasks to meet the deadlines. Notice that, in this figure, we use two compartments, one is the MAVLink library and the other one is the remained ones. We put SITL on RPI3 for this evaluation. And the data access instrumentation and return instruction instrumentation are not implemented.

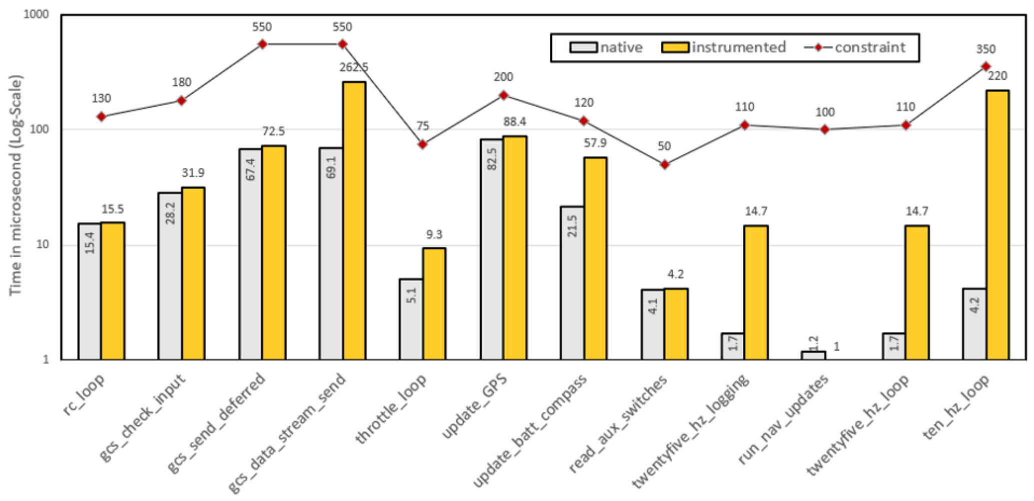


Figure 5.9: Performance impact of ConAttest on real-time tasks with deadline constraints. The overhead introduced by ConAttest is marked on top of every bar that represents the execution time with ConAttest. The results are the average of more than 100 runs.

5.8 RELATED WORK

Remote Attestation Tab. 5.1 summaries recent advancements in remote attestation area. For single device attestation, conventional attestation [96, 97, 94, 95, 84, 152, 153, 91, 154, 155, 156, 90, 85, 83, 157, 158, 159] use integrity of program memory during loading time to build up attestation report for verification. [96, 97, 94, 95] propose static property attestation based network-wide attestation, while [155, 156, 85, 83, 157] propose extending hardware support to improve the performance of static property attestation.

However, conventional defense mechanism cannot detect runtime attacks such as ROP or data-only attack. More recent efforts focus on runtime control flow property to build an attestation report. And this control flow mechanism can defend against control data attack and partial non-control data attack. [18] is the first work to introduce this control flow attestation. They enable a prover to attest the exact control-flow path of an executed program to a remote verifier. In these two years, researchers further include data flow to defend against data-only attacks with attestation [19, 20]. Besides the attestation contents change, the researcher also includes hardware change to either improve the performance of the current scheme or to avoid necessity of halting [169, 170]. [171] also consider TOCTOU attack through hardware changes.

Nonetheless, all of the aforementioned runtime attestation schemes put their attention on runtime attack in the same memory space, without taking advantage of modern memory isolation techniques. Also, they introduce heavy runtime overhead to the instrumented software or platform. And the computation burden of verification on the verifier is nontrivial. In addition, they are based on verifier initiated on-demand attestation which requires the prover to respond to attestation requests anytime. While in our case, unmanned vehicles during a mission might not have this luxury. In this paper, we consider runtime attacks across memory isolation. We hide the security-critical variables in its own compartment. Besides, we use a secure timer to allow the prover to securely self-initiate view flow recording continuously. The ConAttest prototype shows the practical side of our methodology.

Memory Isolation and Compartmentalization A line of research focuses on enabling light-weight memory isolation. Memory isolation requires two fundamental supports, the memory isolation support and the switching support. Techniques for isolation and switching can be based on operating systems [172], hypervisors [173], language runtimes [174], Intel MPK [160] and more. In ConAttest, we use software instrumentation to switch and MMU to isolate the memory. However, our method requires frequent switch into kernel space and has high overhead. Existing research on light-weight memory isolation could potentially be integrated and improve the performance of ConAttest.

There has been a body of research focusing on decomposing programs into multiple isolated compartments and reduce the attack surface in a system. [175] proposes Privtrans which partitions an application into privileged and unprivileged processes using static analysis. Glamdring [176] allows a developer to annotate on sensitive data and applies data

Table 5.1: Remote Attestation Research in Recent Years

Project	Year	Target	Attested Property	Runtime Attack Detection		Hardware Property	Required Change	Continuum
				Control data	Non-control data			
Seda [96]	2015	Software/Network	static memory	N	N	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
SANA [97]	2016	Software/Network	static memory	N	N	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
DARPA [94]	2016	Software/Network	static memory	N	N	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
Us-aid [95]	2018	Software/Network	static memory	N	N	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
PUF [84]	2014	SoC	Circuit	N	N	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
BoardPUF [152]	2015	PCB	Circuit	N	N	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
VM [153]	2004	Software	static memory	N	N	Software-only	N	On-Demand: Verifier Initiated
Swatt [91]	2004	Software	static memory	N	N	Software-only	N	On-Demand: Verifier Initiated
Pioneer [154]	2005	Software	static memory	N	N	Software-only	N	On-Demand: Verifier Initiated
ReDAS [155]	2009	Software	static memory	N	N	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
TrustVisor [156]	2010	Software	static memory	N	N	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
VIPER [90]	2011	Software	static memory	N	N	Software-only	N	On-Demand: Verifier Initiated
Timing [85]	2012	Software	static memory	N	N	Hardware-assisted	N	On-Demand: Verifier Initiated
SMART [83]	2012	Software	static memory	N	N	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
TrustLite [157]	2014	Software	static memory	N	N	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
TyTAN [158]	2015	Software	static memory	N	N	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
Drive [159]	2017	Software	static memory	N	N	Software-only	N	On-Demand: Verifier Initiated
C-flat [18]	2016	Software	Control flow	Y	Partial	Hardware-assisted	Software Instrumentation	On-Demand: Verifier Initiated
Lo-fat [169]	2017	Software	Control flow	Y	Partial	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
Atrium [171]	2017	Software	Control flow	Y	Partial	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
Litehax [170]	2018	Software	Control flow/ Data flow	Y	Y	Hardware-assisted	Hardware	On-Demand: Verifier Initiated
DIAT [19]	2019	Software/Network	Control flow	Y	Partial	Hardware-assisted	Software Instrumentation	On-Demand: Verifier Initiated
OEI [20]	2020	Software	Control flow/ Data flow	Y	Y	Hardware-assisted	Software Instrumentation	On-Demand: Verifier Initiated
This paper	2020	Software	View flow	Y	Y	Hardware-assisted	Software Instrumentation	Continuously always on

and control-flow analysis to partition an application into security-sensitive and non-sensitive compartments. The security-sensitive compartment is put in an Intel SGX enclave for execution. [177] proposes to partition Android applications into compartments and put security-sensitive compartments in ARM TrustZone environment for execution. ACES [150] partitions a single bare-metal system with MPU into multiple compartments. In addition, ACES enables intra-process compartmentalization and creates compartment creation automation. Minion [178] implements partitioning and efficient real-time memory view switching on a commercial drone, 3DR IRIS+. 3DR IRIS+ also uses ArduPilot as its autopilot software. In this paper, ConAttest uses static analysis to create the compartments of ArduPilot software and allows the user to define compartment policy.

5.9 Discussion

In this section, we first investigate the control model of autopilots. The control model shows that view switch behavior has periodic characteristics. Second, we explore how to apply ConAttest to Paparazzi autopilot based unmanned vehicles.

5.9.1 Control model of autopilots

Real-time aspects of autopilots are mostly defined by requirements for low latency sensor acquisition and control responses and specialized low-level interfaces, such as I2C, SPI, CAN, and PWM outputs. To satisfy these requirements, autopilots separate control tasks based on the time scale as shown in Fig. 5.10. Control tasks can generally be modeled as a set of nested control loops. Each control loop has a reference setpoint and the current vehicle

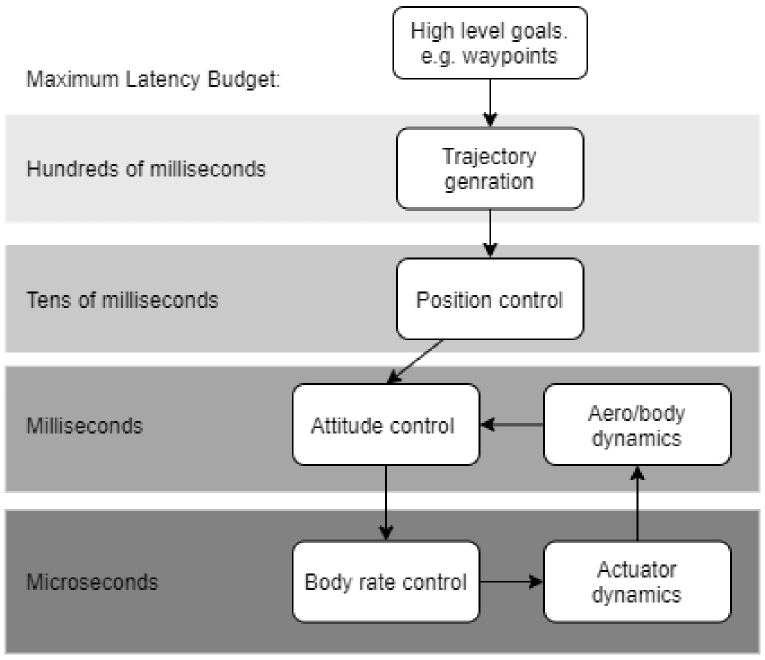


Figure 5.10: The general dynamical model and nested control architecture for autopilots. Different levels have different update frequency and latency requirements.

state as input. It generates the reference for the next inner loop. Even more, advanced control structures often can be described as a set of nested control loops. The outer loops generally have less strict timing requirements compared to the innermost control loops, thus giving the system designer more flexibility on which platform to implement the outer layers. In ConAttest, the compartment policy is configurable and any control loop can be isolated as a control loop. Because of the nested loop structure, enter and exit of the compartment is happening periodically. Thus, the view switch flow follows the periodic characteristics of control loops. And this insight can be further used for computation complexity reduction of VFI for the verifier. We leave this optimization to future works.

5.9.2 ConAttest on Paparazzi

In this section, we discuss how to apply ConAttest to another autopilot system. Paparazzi is an open-source drone hardware and software project encompassing autopilot systems and ground station software for multi-copters/multi-rotors, fixed-wing, helicopters, and hybrid aircraft. The Paparazzi autopilot typically runs an RTOS or bare-metal on a small microcontroller (MCU), such as Lisa/M or Pixhawk, and has limited memory and computational power. The autopilot code is written in C. Onboard code is split between periodic tasks and event tasks. Periodic tasks are scheduled time-sensitive tasks executed at specific periodic intervals. Examples include navigation actions, control loops, and periodic telemetry messages. Event tasks are carried out in response to something. For example, receiving new GPS data or a datalink message. Thus, ConAttest here needs to put periodic tasks and event tasks in two separate compartments.

Paparazzi autopilot is normally a single process on embedded systems or bare metal. So there is no mechanisms to isolate the memory view of different control tasks. So to apply ConAttest, we must choose a method to isolate the memory space and put compartments into each isolated memory space. MPU could be used for this purpose. Also, Paparazzi does not support a secure world, so the integration of ARM TrustZone is needed for the secure recording of VF.

5.10 Summary

Existing remote attestation focuses on a snapshot of integrity and lacks continuum. It is challenging to provide continuum due to the fine-grained granularity characteristics of current schemes. Besides, the runtime overhead of existing schemes introduce too much burden for a real-time unmanned vehicle, which leads to unresponsiveness. Thus, to balance between performance and security requirements of a unmanned vehicle is necessary for achieving continuum. In order to solve aforementioned problems, we present a new runtime property, VFI. Comparing to control flow sequence integrity, VFI relax the granularity while in the meantime, offers certain degree of attack detection capability. We have presented ConAttest, a new security architecture that introduces security-performance trade-off to continuous remote attestation based on VFI. ConAttest is able to detect view crossing runtime attacks instead of to detect runtime attack in a single address space. ConAttest also allows user to configure attack detection rate versus energy consumption. At the end, we show that ConAttest maintains the responsiveness of real-time unmanned vehicle at its original level by implementing it on a commercial drone platform and evaluating the performance overhead.

Chapter 6

Conclusion

With the coming of IoT, people enjoy an easier and more convenient life. Yet this new paradigm of everything going online also brings the serious concern of security and privacy issues. With more attack surfaces exposed by newly coming IoT frameworks, it is of utmost importance to perform security review on the IoT architecture as a whole. IoT is not only about the devices, but also about the networking and cloud computing supports.

6.1 Research Summary

In this dissertation, we explore security in IoT architecture spanning a broad spectrum including sensing capabilities, networking, cloud computing, and endpoint devices. Solutions for unique security and privacy challenges are proposed to further our understanding of IoT as a whole. We present our perspectives of security and privacy to shape the future of IoT. Specifically, we have the following findings.

- In [9], we study the potential privacy leakage in a newly coming commercial gesture control device, Myo. We discuss the privacy concern on the unique property, always-on sensing capabilities, of IoT devices. With more and more sensors equipped on IoT devices, it requires serious privacy inspection for them. Otherwise, the new sensors might leak crucial information from the user. In this work, we show that by applying digital signal processing techniques, EMG signal collected from Myo can be used to infer a password typed on a keyboard or a PIN pressed on a mobile phone. We build a prototype on Myo and demonstrate that the inference is practical and it is important to consider the privacy implication when introducing a new sensor to an IoT device.
- In PriRoster [13], we turn our attention to the networking layer. Comparing to the central data center, edge BS does not have the same level of security level. However, large scale radio context attestation requires edge BS to be trustworthy. In this work,

we propose to utilize Intel SGX to build up trust in Edge BS. Meanwhile, we design a trust transfer protocol to significantly reduce the Intel SGX remote attestation time. Also, we consider memory access side-channel attacks on Intel SGX ecosystem. We defend against the side-channel leakage by implementing oblivious comparing functions. We build up a prototype of PriRoster and show both the microbenchmark evaluation and oblivious memory access patterns. By this work, we aim to provide a trustworthy networking architecture to prepare for the IoT era.

- In PrivacyScope, we focus on the cloud layer, which is also the brain of IoT architecture. Cloud computing stores all the sensitive data collected from IoT users, however, data breakage happens constantly on cloud providers. TEE is a new solution to solve this issue and Intel SGX is one of the most promising TEE technologies. Nonetheless, although Intel SGX can guarantee the integrity of running an enclave program, the enclave program itself might have information leakage bug or code injected by a malicious enclave writer. In this work, we propose PrivacyScope to automatically discover information leakage logic in the enclave program, especially ML programs. We implement the PrivacyScope prototype and integrate it into Intel SGX ecosystem. PrivacyScope allows a user to analyze on enclave program before she trusts a remote cloud provider and submits her sensitive information to him.

- In Conattest, we focus on the continuous system state integrity of IoT devices. Unmanned vehicle is an essential part of the incoming IoT world. It is important for people to know if an unmanned vehicle is trustworthy or not. For example, when a food delivery service is conducted by a drone, the drone needs to prove to the user that there is no suspicious behavior during the flight. In this work, we propose VFI to allow proving a continuous benign system state of an unmanned vehicle to a verifier. We combine the advance in both memory isolation line of research and runtime remote attestation line of research to achieve practical continuous VFI attestation. We implement Conattest prototype on a commercial drone platform and demonstrate its efficacy.

6.2 Future Work

Many topics explored in this dissertation can be further extended as follows.

- [9] identifies the privacy leakage of EMG signal. With more IoT devices coming to the market, more sensing capabilities are also on their way. For example, a smartwatch introduces a bunch of new sensors, including an ambient light sensor used for monitoring heart rate. It would be interesting to extract the activity pattern of a user using this heart rate monitor. Also, people's emotional status is correlated with heart rate. Thus, potentially this sensor can be used to detect the emotion a user keeps. With the workflow provided in [9], this sensor can be extended to other privacy leakage scenarios.

- [13] provides a new paradigm for IoT architecture, which is trustworthy edge computing. We believe by applying TEE to edge nodes, a lot of applications can be thought of. Besides, [13] proposes a general method for setting up large-scale trustworthy edge nodes. This method can be further specialized according to application scenarios. Also, we investigate the memory side channel of Intel SGX enclaves. This is not the only side channel. Other architectural drawbacks in TEEs might also be considered.

- PrivacyScope focuses on the information leakage inside enclaves. PrivacyScope uses static analysis on enclave source code to achieve its goal. It would be interesting if the same goal could be achieved without source code but only the binaries. Another interesting topic would be to extend the privacy policy specified in PrivacyScope. For example, one can add more information flow policies to enforce the flow of information inside the enclave.

- Conattest proposes to use VFI in runtime attestation. The VFI record generation is not optimized in Conattest. Techniques like multi-set Hash, hashmap count can be used to further tradeoff between security and performance. Also, Conattest focuses on ARM Cortex-A processors, while ARM Cortex-M processors are coming to the market. ARM Cortex-M processors are specially designed for IoT and offer MPU and ARM TrustZone-M. To extend the methodology of Conattest to newly IoT devices is another interesting research path.

Bibliography

[1] K. Ashton *et al.*, "That 'internet of things' thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[2] `https://www.gartner.com/newsroom/id/2905717`.

[3] `https://www.digitaltrends.com/home/why-hackers-hack-security-cameras/`.

[4] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, *et al.*, "Comprehensive experimental analyses of automotive attack surfaces.," in *USENIX Security Symposium*, pp. 77–92, San Francisco, 2011.

[5] B. Gardiner, "In spectrum auction, winners are at&t, verizon and openness," *Wired*, 2008.

[6] G. Wang and T. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *Infocom, 2010 proceedings ieee*, pp. 1–9, IEEE, 2010.

[7] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1273–1285, ACM, 2015.

[8] C. Wang, X. Guo, Y. Wang, Y. Chen, and B. Liu, "Friend or foe?: Your wearable devices reveal your personal pin," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pp. 189–200, ACM, 2016.

[9] R. Zhang, N. Zhang, C. Du, W. Lou, Y. T. Hou, and Y. Kawamoto, "From electromyogram to password: Exploring the privacy impact of wearables in augmented reality," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 9, no. 1, p. 13, 2017.

[10] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, *et al.*, "Secure multiparty computation goes live," in *International Conference on Financial Cryptography and Data Security*, pp. 325–343, Springer, 2009.

[11] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A distributed sandbox for untrusted computation on secret data.," in *OSDI*, pp. 533–549, 2016.

[12] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "Vc3: Trustworthy data analytics in the cloud using sgx," in *Security and Privacy (SP), 2015 IEEE Symposium on*, pp. 38–54, IEEE, 2015.

[13] R. Zhang, N. Wang, N. Zhang, Z. Yan, W. Lou, and Y. T. Hou, "Priroster: Privacy-preserving radio context attestation in cognitive radio networks," in *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 1–10, IEEE, 2019.

[14] "Insider threats." `https://www.uscybersecurity.net/insider-threats-2018-statistics/`.

[15] A. C. Myers, "Jflow: Practical mostly-static information flow control," in *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 228–241, ACM, 1999.

[16] D. Volpano, C. Irvine, and G. Smith, "A sound type system for secure flow analysis," *Journal of computer security*, vol. 4, no. 2-3, pp. 167–187, 1996.

[17] R. Zhang, N. Zhang, A. Moini, W. Lou, and Y. T. Hou, "Privacyscope: Automatic analysis of private dataleakage in tee-protected applications," in *To appear in the IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2020.

[18] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-flat: control-flow attestation for embedded systems software," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 743–754, ACM, 2016.

[19] T. Abera, R. Bahmani, F. Brasser, A. Ibrahim, A.-R. Sadeghi, and M. Schunter, "Diat: Data integrity attestation for resilient collaboration of autonomous systems.," in *NDSS*, 2019.

[20] Z. Sun, B. Feng, L. Lu, and S. Jha, "Oei: operation execution integrity for embedded devices," *arXiv preprint arXiv:1802.03462*, 2018.

[21] "Microsoft's hololens is super limited – and hella magical." `https://www.cnet.com/products/microsoft-hololens-hands-on/`, April 2016. [Online; posted 1-April-2016].

[22] "Goldman sachs has four charts showing the huge potential in virtual and augmented reality." `https://www.pinterest.com/pin/389561436498667612/`, Jan 2016. [Online; posted Jan-13-2016].

[23] A. Maiti, O. Armbruster, M. Jadliwala, and J. He, "Smartwatch-based keystroke inference attacks and context-aware protection mechanisms," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pp. 795–806, ACM, 2016.

[24] "Myo." `https://www.myo.com/`, 2016.

[25] E. N. Marieb and K. Hoehn, *Human anatomy & physiology.* Pearson Education, 2007.

[26] J. V. Basmajian and C. De Luca, "Muscles alive," *Muscles alive: their functions revealed by electromyography*, vol. 278, p. 126, 1985.

[27] C. J. De Luca, A. Adam, R. Wotiz, L. D. Gilmore, and S. H. Nawab, "Decomposition of surface emg signals," *Journal of neurophysiology*, vol. 96, no. 3, pp. 1646–1657, 2006.

[28] M. Khalil and J. Duchêne, "Uterine emg analysis: a dynamic approach for change detection and classification," *IEEE Transactions on Biomedical Engineering*, vol. 47, no. 6, pp. 748–756, 2000.

[29] Y. Al-Assaf, "Surface myoelectric signal analysis: dynamic approaches for change detection and classification," *IEEE transactions on biomedical engineering*, vol. 53, no. 11, pp. 2248–2256, 2006.

[30] W. El Falou, M. Khalil, and J. Duchene, "Ar-based method for change detection using dynamic cumulative sum," in *7th IEEE Internat. Conf. on Electronics, Circuits and Systems, ICECS*, vol. 1, pp. 157–160, 2000.

[31] O. Mustapha, D. Lefebvre, M. Khalil, G. Hoblos, and H. Chafouk, "Filters bank derived from the wavelet transform for real time change detection in signal," in *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pp. 1–6, IEEE, 2008.

[32] O. Mustapha, D. Lefebvre, G. Hoblos, H. Chafouk, and M. Khalil, *Fault Detection Algorithm Based on Filters Bank Derived from Wavelet Packets.* INTECH Open Access Publisher, 2008.

[33] B. Hudgins, P. Parker, and R. N. Scott, "A new strategy for multifunction myoelectric control," *IEEE Transactions on Biomedical Engineering*, vol. 40, no. 1, pp. 82–94, 1993.

[34] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to platt's smo algorithm for svm classifier design," *Neural Computation*, vol. 13, no. 3, pp. 637–649, 2001.

[35] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *IEEE Symposium on Security and Privacy*, vol. 2004, pp. 3–11, 2004.

[36] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp) iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 551–562, ACM, 2011.

[37] T. Zhu, Q. Ma, S. Zhang, and Y. Liu, "Context-free attacks using keyboard acoustic emanations," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 453–464, ACM, 2014.

[38] D. Balzarotti, M. Cova, and G. Vigna, "Clearshot: Eavesdropping on keyboard input from video," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pp. 170–183, IEEE, 2008.

[39] F. Maggi, S. Gasparini, and G. Boracchi, "A fast eavesdropping attack against touchscreens," in *Information Assurance and Security (IAS), 2011 7th International Conference on*, pp. 320–325, IEEE, 2011.

[40] H. Wang, T. T.-T. Lai, and R. Roy Choudhury, "Mole: Motion leaks through smartwatch sensors," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 155–166, ACM, 2015.

[41] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion.," *HotSec*, vol. 11, pp. 9–9, 2011.

[42] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: your finger taps have fingerprints," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 323–336, ACM, 2012.

[43] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pp. 113–124, ACM, 2012.

[44] F. H. Chan, Y.-S. Yang, F. Lam, Y.-T. Zhang, and P. A. Parker, "Fuzzy emg classification for prosthesis control," *IEEE transactions on rehabilitation engineering*, vol. 8, no. 3, pp. 305–311, 2000.

[45] K. Englehart, B. Hudgins, P. A. Parker, and M. Stevenson, "Classification of the myoelectric signal using time-frequency based representations," *Medical engineering & physics*, vol. 21, no. 6, pp. 431–438, 1999.

[46] G. Tsenov, A. Zeghbib, F. Palis, N. Shoylev, and V. Mladenov, "Neural networks for online classification of hand and finger movements using surface emg signals," in *2006 8th Seminar on Neural Network Applications in Electrical Engineering*, pp. 167–171, IEEE, 2006.

[47] K. A. Farry, I. D. Walker, and R. G. Baraniuk, "Myoelectric teleoperation of a complex robotic hand," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 775–788, 1996.

[48] C. Jorgensen, D. D. Lee, and S. Agabont, "Sub auditory speech recognition based on emg signals," in *Neural Networks, 2003. Proceedings of the International Joint Conference on*, vol. 4, pp. 3128–3133, IEEE, 2003.

[49] C. L. Fancourt and J. C. Principe, "On the use of neural networks in the generalized likelihood ratio test for detecting abrupt changes in signals.," in *IJCNN (2)*, pp. 243–252, 2000.

[50] M. Barkat, *Signal detection and estimation.* Artech house, 2005.

[51] M. Khalil and J. Duchêne, "Dynamic cumulative sum approach for change detection," *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, vol. 47, no. 4, p. 1205, 1999.

[52] O. A. Grigg, V. Farewell, and D. Spiegelhalter, "Use of risk-adjusted cusum and rsprtcharts for monitoring in medical contexts," *Statistical methods in medical research*, vol. 12, no. 2, pp. 147–170, 2003.

[53] C. K. Chui, *An introduction to wavelets*, vol. 1. Academic press, 2014.

[54] "Learn how to touch type.." `http://www.ratatype.com/learn`, 2016.

[55] R. Lewand, *Cryptological mathematics.* MAA, 2000.

[56] M. Vuagnoux and S. Pasini, "Compromising electromagnetic emanations of wired and wireless keyboards.," in *USENIX security symposium*, pp. 1–16, 2009.

[57] Y. Berger, A. Wool, and A. Yeredor, "Dictionary attacks using keyboard acoustic emanations," in *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 245–254, ACM, 2006.

[58] L. Zhuang, F. Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, p. 3, 2009.

[59] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm, "ispy: automatic reconstruction of typed input from compromising reflections," in *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 527–536, ACM, 2011.

[60] D. Shukla, R. Kumar, A. Serwadda, and V. V. Phoha, "Beware, your hands reveal your secrets!," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 904–917, ACM, 2014.

[61] "From radiotelegraphy to worldwide wireless: How itu processes and regulations have helped shape the modern world of radiocommunications." `https://www.itu.int/itunews/manager/display.asp?lang=en&year=2006&issue=03&ipage=radiotelegraphy&ext=html`.

[62] "Optimising leds for wireless communication." `https://compoundsemiconductor.net/article/99050/Optimising_LEDs_for_wireless_communication/feature`.

[63] A. Ghasemi and E. S. Sousa, "Spectrum sensing in cognitive radio networks: requirements, challenges and design trade-offs," *IEEE commun. Mag.*, vol. 46, no. 4, 2008.

[64] M. M. Sohul, M. Yao, T. Yang, and J. H. Reed, "Spectrum access system for the citizen broadband radio service," *IEEE Commun. Mag.*, vol. 53, no. 7, pp. 18–25, 2015.

[65] G. Coker, J. Guttman, P. Loscocco, and et al., "Principles of remote attestation," *Int. J. of Inf. Security*, vol. 10, no. 2, pp. 63–81, 2011.

[66] N. Zhang, W. Sun, W. Lou, and et al., "Roster: Radio context attestation in cognitive radio network," in *2018 IEEE CNS*, pp. 1–9, 2018.

[67] X. He, R. Jin, and H. Dai, "Camouflaging mobile primary users in database-driven cognitive radio networks," *IEEE Wireless commun. Letters*, 2018.

[68] "He strava heat map and the end of secrets." `https://www.wired.com/story/strava-heat-map-military-bases-fitness-trackers-privacy/`.

[69] S. Jajodia, "Adversarial and uncertain reasoning for adaptive cyber defense: Building the scientific foundation," 2015.

[70] B. Bahrak, S. Bhattarai, A. Ullah, J.-M. J. Park, J. Reed, and D. Gurney, "Protecting the primary users' operational privacy in spectrum sharing," in *2014 IEEE International Symposium on Dynamic Spectrum Access Networks (DYSPAN)*, pp. 236–247, IEEE, 2014.

[71] V. Costan and S. Devadas, "Intel sgx explained.," *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[72] F. Brasser, U. Müller, A. Dmitrienko, and et al., "Software grand exposure: Sgx cache attacks are practical," *arXiv preprint arXiv:1702.07521*, p. 33, 2017.

[73] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *Security and Privacy (SP), 2015 IEEE Symposium on*, pp. 640–656, IEEE, 2015.

[74] W. Sun, R. Zhang, W. Lou, and Y. T. Hou, "Rearguard: Secure keyword search using trusted hardware," *IEEE INFORM*, 2018.

[75] M. Palola, M. Höyhtyä, P. Aho, M. Mustonen, T. Kippola, M. Heikkilä, S. Yrjola, V. Hartikainen, L. Tudose, A. Kivinen, R. Ekman, J. Hallio, J. Paavola, M. Mäkeläinen, and T. Hänninen, "Field trial of the 3.5 ghz citizens broadband radio service governed by a spectrum access system (sas)," 03 2017.

[76] M. M. Sohul, M. Yao, T. Yang, and J. H. Reed, "Spectrum access system for the citizen broadband radio service," *IEEE Communications Magazine*, vol. 53, pp. 18–25, July 2015.

[77] A. ARM, "Security technology building a secure system using trustzone technology (white paper)," *ARM Limited*, 2009.

[78] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing, and M. Vij, "Integrating remote attestation with transport layer security," *arXiv preprint arXiv:1801.05863*, 2018.

[79] O. Ohrimenko, F. Schuster, C. Fournet, and et al., "Oblivious multi-party machine learning on trusted processors.," in *USENIX Security Symp.*, pp. 619–636, 2016.

[80] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital side-channels through obfuscated execution.," in *USENIX Security Symp.*, pp. 431–446, 2015.

[81] "Optee." `https://github.com/OP-TEE/optee_os`.

[82] V. J. Reddi, A. Settle, D. A. Connors, and et al., "Pin: a binary instrumentation tool for computer architecture research and education," in *2004 workshop on Computer architecture education: held in conjunction with the 31st Int. Symp. on Computer Architecture*, p. 22, ACM, 2004.

[83] K. Eldefrawy, G. Tsudik, A. Francillon, and et al., "Smart: Secure and minimal architecture for (establishing dynamic) root of trust.," in *NDSS*, vol. 12, pp. 1–15, 2012.

[84] J. Kong, F. Koushanfar, P. K. Pendyala, and et al., "Pufatt: Embedded platform attestation based on novel processor-based pufs," in *51st Annu. Design Automation Conference*, pp. 1–6, ACM, 2014.

[85] X. Kovah, C. Kallenberg, C. Weathers, A. Herzog, M. Albin, and J. Butterworth, "New results for timing-based attestation," in *2012 IEEE Symposium on Security and Privacy*, pp. 239–253, IEEE, 2012.

[86] H. Park, D. Seo, H. Lee, and et al., "Smatt: Smart meter attestation using multiple target selection and copy-proof memory," in *Computer Science and its Applications*, pp. 875–887, Springer, 2012.

[87] S. Schulz, A.-R. Sadeghi, and C. Wachsmann, "Short paper: Lightweight remote attestation using physical functions," in *4fourth ACM Conf. on Wireless network security*, pp. 109–114, 2011.

[88] N. Zhang, K. Sun, W. Lou, and et al., "Case: Cache-assisted secure execution on arm processors," in *2016 IEEE S&P*, pp. 72–90, 2016.

[89] R. Kennell and L. H. Jamieson, "Establishing the genuinity of remote computer systems.," in *USENIX Security Symp.*, pp. 295–308, 2003.

[90] Y. Li, J. M. McCune, and A. Perrig, "Viper: verifying the integrity of peripherals' firmware," in *18th ACM CCS*, pp. 3–16, 2011.

[91] A. Seshadri, A. Perrig, L. Van Doorn, and et al., "Swatt: Software-based attestation for embedded devices," in *null*, p. 272, IEEE, 2004.

[92] A. Seshadri, M. Luk, and A. Perrig, "Sake: Software attestation for key establishment in sensor networks," in *Int. Conference on Distributed Computing in Sensor Systems*, pp. 372–385, Springer, 2008.

[93] A. Vasudevan, J. McCune, J. Newsome, and et al., "Carma: A hardware tamper-resistant isolated execution environment on commodity x86 platforms," in *7th ACM Symp. on Information, Computer and commun. Security*, pp. 48–49, 2012.

[94] A. Ibrahim, A.-R. Sadeghi, G. Tsudik, and et al., "Darpa: Device attestation resilient to physical attacks," in *9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pp. 171–182, 2016.

[95] A. Ibrahim, "Aid : Autonomous attestation of iot devices," 2018.

[96] N. Asokan, F. Brasser, A. Ibrahim, and et al., "Seda: Scalable embedded device attestation," in *22nd ACM SIGSAC CCS*, pp. 964–975, 2015.

[97] M. Ambrosin, M. Conti, A. Ibrahim, and et al., "Sana: secure and scalable aggregate network attestation," in *2016 ACM SIGSAC CCS*, pp. 731–742, 2016.

[98] X. Jin, J. Sun, R. Zhang, and et al., "Specguard: Spectrum misuse detection in dynamic spectrum access systems," *IEEE Trans. on Mobile Computing*, 2018.

[99] X. Jin, J. Sun, R. Zhang, and Y. Zhang, "Safedsa: Safeguard dynamic spectrum access against fake secondary users," in *22nd ACM SIGSAC CCS*, pp. 304–315, ACM, 2015.

[100] V. Kumar, J.-M. Park, and K. Bian, "Blind transmitter authentication for spectrum security and enforcement," in *2014 ACM SIGSAC CCS*, pp. 787–798, ACM, 2014.

[101] Y. Liu, P. Ning, and H. Dai, "Authenticating primary users' signals in cognitive radio networks via integrated cryptographic and wireless link signatures," in *2010 IEEE S&P*, pp. 286–301, 2010.

[102] Y. Dou, K. C. Zeng, Y. Yang, and et al., "Madecr: Correlation-based malware detection for cognitive radio," in *2015 IEEE INFOCOM*, pp. 639–647, 2015.

[103] C. Li, A. Raghunathan, and N. K. Jha, "An architecture for secure software defined radio," in *Conference on Design, Automation and Test in Europe*, pp. 448–453, 2009.

[104] M.-W. Shih, S. Lee, T. Kim, and et al., "T-sgx: Eradicating controlled-channel attacks against enclave programs," in *2017 NDSS*, 2017.

[105] S. Sasy, S. Gorbunov, and C. W. Fletcher, "Zerotrace: Oblivious memory primitives from intel sgx," in *NDSS*, 2017.

[106] E. Stefanov, M. Van Dijk, E. Shi, and et al., "Path oram: an extremely simple oblivious ram protocol," in *2013 ACM SIGSAC CCS*, pp. 299–310, ACM, 2013.

[107] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting privileged side-channel attacks in shielded execution with déjá vu," in *2017 ACM on Asia CCS*, pp. 7–18, 2017.

[108] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "Truspy: Cache side-channel information leakage from the secure world on arm devices.," *IACR Cryptology ePrint Archive*, vol. 2016, p. 980, 2016.

[109] S. Lee, M.-W. Shih, P. Gera, and et al., "Inferring fine-grained control flow inside sgx enclaves with branch shadowing," in *26th USENIX Security Symp.*, pp. 16–18, 2017.

[110] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*, vol. 20. Stanford University Stanford, 2009.

[111] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform.," in *NSDI*, pp. 283–298, 2017.

[112] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2421–2434, ACM, 2017.

[113] R. Sinha, S. Rajamani, S. Seshia, and K. Vaswani, "Moat: Verifying confidentiality of enclave programs," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1169–1184, ACM, 2015.

[114] F. Liu, H. Wu, and R. B. Lee, "Can randomized mapping secure instruction caches from side-channel attacks?," in *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy*, p. 4, ACM, 2015.

[115] J. C. King, "Symbolic execution and program testing," *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[116] R. S. Boyer, B. Elspas, and K. N. Levitt, "Select—a formal system for testing and debugging programs by symbolic execution," *ACM SigPlan Notices*, vol. 10, no. 6, pp. 234–245, 1975.

[117] J. B. Kam and J. D. Ullman, "Monotone data flow analysis frameworks," *Acta Informatica*, vol. 7, no. 3, pp. 305–317, 1977.

[118] F. E. Allen and J. Cocke, "A program data flow analysis procedure," *Communications of the ACM*, vol. 19, no. 3, p. 137, 1976.

[119] Z. Xu, T. Kremenek, and J. Zhang, "A memory model for static analysis of c programs," in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pp. 535–548, Springer, 2010.

[120] J. A. Goguen and J. Meseguer, "Security policies and security models," in *Security and Privacy, 1982 IEEE Symposium on*, pp. 11–11, IEEE, 1982.

[121] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," *IEEE Journal on selected areas in communications*, vol. 21, no. 1, pp. 5–19, 2003.

[122] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)," in *Security and privacy (SP), 2010 IEEE symposium on*, pp. 317–331, IEEE, 2010.

[123] "Understanding lvalues and rvalues in c and c++." `https://eli.thegreenplace.net/2011/12/15/understanding-lvalues-and-rvalues-in-c-and-c`.

[124] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.

[125] A. Ferraiuolo, R. Xu, D. Zhang, A. C. Myers, and G. E. Suh, "Verification of a practical hardware security architecture through static information flow analysis," in *ACM SIGARCH Computer Architecture News*, vol. 45, pp. 555–568, ACM, 2017.

[126] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 116–127, ACM, 2007.

[127] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale," in *International Conference on Trust and Trustworthy Computing*, pp. 291–307, Springer, 2012.

[128] D. Volpano and G. Smith, "Eliminating covert flows with minimum typings," in *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pp. 156–168, IEEE, 1997.

[129] J. Agat, "Transforming out timing leaks," in *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 40–53, ACM, 2000.

[130] D. Volpano and G. Smith, "Probabilistic noninterference in a concurrent language 1," *Journal of Computer Security*, vol. 7, no. 2-3, pp. 231–253, 1999.

[131] "Recommender." `https://github.com/GHamrouni/Recommender`.

[132] "Linearregression." `https://github.com/aluxian/CPP-ML-LinearRegression`.

[133] "Simple c routines for generic k-means implementations." `https://github.com/pramsey/kmeans`.

[134] "Recommender." `https://github.com/GHamrouni/Recommender/tree/master/src`.

[135] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," *arXiv preprint arXiv:1803.05961*, 2018.

[136] T. Alves and D. Felton, "Trustzone: Integrated hardware and software security-enabling trusted computing in embedded systems (july 2004)."

[137] D. Grawrock, *Dynamics of a Trusted Platform: A building block approach.* Intel Press, 2009.

[138] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in *2nd Int. workshop on hardware and architectural support for security and privacy*, vol. 13, ACM New York, NY, USA, 2013.

[139] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions.," *HASP@ ISCA*, vol. 11, 2013.

[140] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution.," *HASP@ ISCA*, vol. 10, 2013.

[141] E. Elnikety, A. Mehta, A. Vahldiek-Oberwagner, D. Garg, and P. Druschel, "Thoth: Comprehensive policy compliance in data retrieval systems.," in *USENIX Security Symposium*, pp. 637–654, 2016.

[142] H. Nissenbaum, S. Benthall, A. Datta, M. C. Tschantz, and P. Mardziel, "Origin privacy: Protecting privacy in the big-data era," tech. rep., NEW YORK UNIVERSITY New York United States, 2018.

[143] S. Sen, S. Guha, A. Datta, S. K. Rajamani, J. Tsai, and J. M. Wing, "Bootstrapping privacy compliance in big data systems," in *Security and Privacy (SP), 2014 IEEE Symposium on*, pp. 327–342, IEEE, 2014.

[144] D. L. Haulman, "Us unmanned aerial vehicles in combat, 1991-2003," 2003.

[145] `https://www.businessinsider.com/drone-technology-uses-applications`.

[146] `https://www.businessinsider.com/drone-delivery-services`.

[147] https://www.uber.com/us/en/atg/technology/.

[148] https://securelist.com/holy-water-ongoing-targeted-water-holing-attack-in-asia/96311/.

[149] "Ardupilot." https://ardupilot.org/.

[150] A. A. Clements, N. S. Almakhdhub, S. Bagchi, and M. Payer, "{ACES}: Automatic compartments for embedded systems," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 65–82, 2018.

[151] X. Carpent, G. Tsudik, and N. Rattanavipanon, "Erasmus: Efficient remote attestation via self-measurement for unattended settings," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1191–1194, IEEE, 2018.

[152] L. Wei, C. Song, Y. Liu, J. Zhang, F. Yuan, and Q. Xu, "Boardpuf: Physical unclonable functions for printed circuit board authentication," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 152–158, IEEE Press, 2015.

[153] V. Haldar, D. Chandra, and M. Franz, "Semantic remote attestation: a virtual machine directed approach to trusted computing," in *USENIX Virtual Machine Research and Technology Symposium*, vol. 2004, 2004.

[154] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. Van Doorn, and P. Khosla, "Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems," in *ACM SIGOPS Operating Systems Review*, vol. 39, pp. 1–16, ACM, 2005.

[155] C. Kil, E. C. Sezer, A. M. Azab, P. Ning, and X. Zhang, "Remote attestation to dynamic system properties: Towards providing complete system integrity evidence," in *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pp. 115–124, IEEE, 2009.

[156] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "Trustvisor: Efficient tcb reduction and attestation," in *2010 IEEE Symposium on Security and Privacy*, pp. 143–158, IEEE, 2010.

[157] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: A security architecture for tiny embedded devices," in *Proceedings of the Ninth European Conference on Computer Systems*, p. 10, ACM, 2014.

[158] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "Tytan: tiny trust anchor for tiny devices," in *Proceedings of the 52nd Annual Design Automation Conference*, p. 34, ACM, 2015.

[159] A. Rein, "Drive: Dynamic runtime integrity verification and evaluation," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 728–742, ACM, 2017.

[160] A. Vahldiek-Oberwagner, E. Elnikety, N. O. Duarte, M. Sammler, P. Druschel, and D. Garg, "{ERIM}: Secure, efficient in-process isolation with protection keys ({MPK})," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pp. 1221–1238, 2019.

[161] "Llvm." http://llvm.org/.

[162] "networkx." https://networkx.github.io/.

[163] https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837/README.md.

[164] https://www.studica.com/blog/raspberry-pi-timer-embedded-environments.

[165] https://www.raspberrypi.org/forums/viewtopic.php?t=9882.

[166] https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2836/QA7_rev3.4.pdf.

[167] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0500d/BABIGHII.html.

[168] T. B. Schardl, T. Denniston, D. Doucet, B. C. Kuszmaul, I.-T. A. Lee, and C. E. Leiserson, "The csi framework for compiler-inserted program instrumentation," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 2, pp. 1–25, 2017.

[169] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A.-R. Sadeghi, "Lo-fat: Low-overhead control flow attestation in hardware," in *Proceedings of the 54th Annual Design Automation Conference 2017*, p. 24, ACM, 2017.

[170] G. Dessouky, T. Abera, A. Ibrahim, and A.-R. Sadeghi, "Litehax: Lightweight hardware-assisted attestation of program execution," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2018.

[171] S. Zeitouni, G. Dessouky, O. Arias, D. Sullivan, A. Ibrahim, Y. Jin, and A.-R. Sadeghi, "Atrium: Runtime attestation resilient under memory attacks," in *Proceedings of the 36th International Conference on Computer-Aided Design*, pp. 384–391, IEEE Press, 2017.

[172] J. Litton, A. Vahldiek-Oberwagner, E. Elnikety, D. Garg, B. Bhattacharjee, and P. Druschel, "Light-weight contexts: An {OS} abstraction for safety and performance," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 49–64, 2016.

[173] Y. Liu, T. Zhou, K. Chen, H. Chen, and Y. Xia, "Thwarting memory disclosure with efficient hypervisor-enforced intra-domain isolation," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1607–1619, 2015.

[174] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham, "Efficient software-based fault isolation," in *Proceedings of the fourteenth ACM symposium on Operating systems principles*, pp. 203–216, 1993.

[175] D. Brumley and D. Song, "Privtrans: Automatically partitioning programs for privilege separation," in *USENIX Security Symposium*, vol. 57, 2004.

[176] J. Lind, C. Priebe, D. Muthukumaran, D. O'Keeffe, P.-L. Aublin, F. Kelbert, T. Reiher, D. Goltzsche, D. Eyers, R. Kapitza, *et al.*, "Glamdring: Automatic application partitioning for intel {SGX}," in *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, pp. 285–298, 2017.

[177] K. Rubinov, L. Rosculete, T. Mitra, and A. Roychoudhury, "Automated partitioning of android applications for trusted execution environments," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 923–934, IEEE, 2016.

[178] C. H. Kim, T. Kim, H. Choi, Z. Gu, B. Lee, X. Zhang, and D. Xu, "Securing real-time microcontroller systems through customized memory view switching.," in *NDSS*, 2018.