Securing Internet-of-Things via Fine-grained Network Detection and Prevention

Tianlong Yu

CMU-CS-20-120

August, 2020

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Vyas Sekar (Co-Chair)
Srinivasan Seshan (Co-Chair)
Yuvraj Agarwal
Virginia Smith
Susanta Nanda (Symantec Research Labs)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation award number CNS1440056; by the SEI Directors Office award number FA8702-15-D-0002; by the Semiconductor Research Group award number 2018JU2779; and by a fellowship from Symantec. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Internet-of-Things, Network Security, Distributed Machine Learning

Abstract

The Internet-of-Things (IoT) has quickly moved from the realm of hype to reality with estimates of over 25 billion devices deployed by 2020. While IoT has huge potential for societal impact, it comes with several key security challenges—IoT devices can become the entry points into critical infrastructures and can be exploited to leak sensitive information. Traditional host-centric security solutions in today's IT ecosystems (e.g., antivirus, software patches) are fundamentally at odds with the realities of IoT (e.g., poor vendor security practices and constrained hardware). We argue that the network will have to play a critical role in securing IoT deployments. However, the scale, diversity, cyber-physical coupling, and cross-device use cases inherent to IoT require us to rethink network security along three key dimensions. First, current enforcement architecture cannot enforce context-based and agile security postures needed to protect IoT devices. Second, current detection mechanisms cannot learn the network-side behaviors for a single IoT device. Third, there is no mechanism to learn the complex environment-device or cross-device interactions for IoT devices.

To tackle these problems, we build a fine-grained network detection and prevention system for IoT devices. The workflow of the system is as follows. In the first step, the system can learn single-device behaviors as well as cross-device interactions from historical records. Then, the system can convert the single-device behavioral models and interactions models into regulating security policies, and enforce such security policies in a context-based and agile manner to protect the IoT devices.

However, there are several key challenges. To learn single-device behaviors, the main challenges are the lack of single-device behavioral models and how to address the data pollution issue in a realistic setting. For learning complex interactions, it is hard to define a model to capture the environment-device interactions and cross-device interactions. Besides, learning such an interaction model for IoT devices faces the challenge of insufficient data and privacy issues. For the enforcement part, it is hard to design an expressive context-based and agile policy abstraction that can capture security postures needed for IoT devices. Also, it is hard to design a scalable and responsive controller to orchestrate the enforcement architecture.

Next, we briefly describe our solutions to address these challenges. To model the network behaviors of an IoT device, we design a robust behavioral model inference mechanism called RADIO to build benign behavioral models from potentially polluted network traces. To learn the complex IoT interactions, we build a distributed learning mechanism called LoFTI to learn the IoT interaction model across multiple smart homes. To provide context-based and agile enforcement, we build a new enforcement architecture called PSI (Precise Security Instrumentation). Leveraging recent advances in SDN (Software-Defined Networking) and NFV (Network Function Virtualization), PSI protects each IoT device with dedicated software middleboxes enforcing context-based and agile policies.

Acknowledgments

I would like to express my sincere gratitude to my advisors Vyas and Srini for their support of my Ph. D study and research, for their knowledge, patience and insightful guidance.

I would like to thank Yuvraj and Ginger for their immense suggestions and help in building this thesis.

I would like to thank Susanta for his long-term support and help since I interned in Symantec Research Labs.

I would like to thank my family for their understanding and support over my phd years.

I am truly grateful to all those who helped me during all these years in CMU. Thank you all for your kind support and help.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet-of-Things (IoT) has quickly moved from hype to reality; Gartner, Inc. estimates that the number of deployed IoT devices will grow from 5 Billion in 2015 to 25 Billion in 2020[3]. Like other disruptive technologies, such as smartphones and cloud computing, IoT holds the potential for societal scale impact by transforming many industries as well as our daily lives.

While IoT has huge potential, it also comes with its share of challenges. Most vendors only deal with parts of the IoT ecosystem and, typically, their priorities have been providing novel functionality, getting their products to market soon, and making them easy to use. Unfortunately, security and privacy risks have not received as much attention. Since IoT devices will typically be embedded deep inside networks, they are attractive targets and may become the "weakest point" for breaking into enterprises or smart homes[5], or for leaking sensitive information about users and their behaviors as well as causing physical hazards[4]. These are not hypothetical concerns as several actual attacks have already been reported. For example, IoT devices were used as bots to launch DDoS or spam[2], smart meters were hacked to lower utility bills[1], and handheld scanners were compromised to enter logistics firms[6].

1.1 Security issues for IoT devices

To understand the challenges that come with IoT devices, we first look at the security issues faced by IoT devices. From the single-device perspective, the IoT devices are threatened by anomalous behaviors including code injection and authentication bypass. Then from the device-interactions perspective, the IoT devices are threatened by malicious device interactions that can cause physical hazards and privacy leakage.

1.1.1 Single-device vulnerabilities

As shown by previous work on firmware analysis [49, 54, 55, 109] and by reports of security incidents [10, 11, 23], existing IoT devices are threatened by single-device vulnerabilities:

- IoT devices often have flaws when processing commands/arguments, which allows the attacker to inject code via the flawed commands or arguments [10, 11, 49, 54, 55].

- Many IoT devices contain flawed authentication routines (backdoors) [109], which allow the attacker to bypass authentication.
- Since the IoT devices are deployed inside the enterprise networks, they often have unrestricted access to other IoT devices or computers, which allows the attacker to use the compromised IoT devices as stepping stone for more advanced attacks [23].

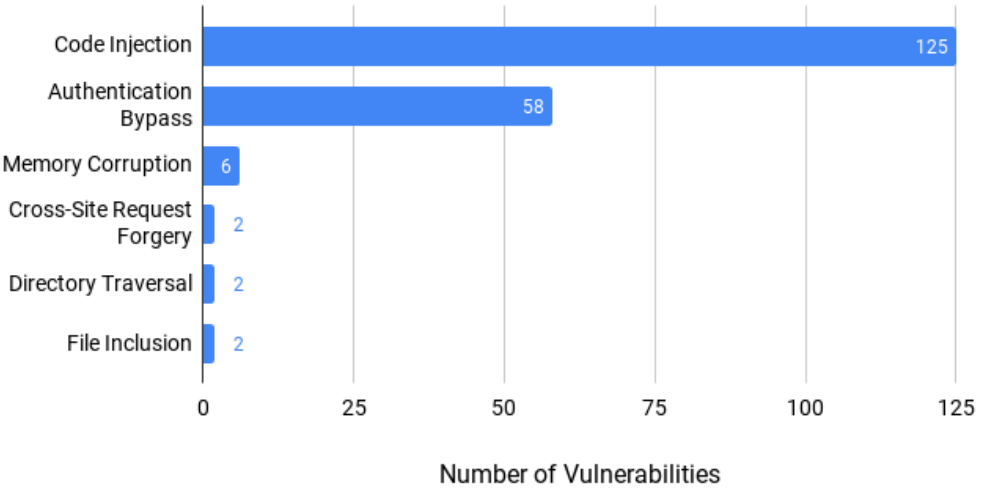

Figure 1.1: Different vulnerabilities for commonly used enterprise IoT devices from 1999 to 2018 from NVD [29].

To identify the possible existence of the above vulnerabilities in IoT deployments, we select a few devices to represent each IoT device class and analyzed their vulnerabilities. Figure 1.1 shows the number of different types of vulnerabilities reported between 1999 to 2018 in the National Vulnerability Database [29] for Cisco IOS router/switch, FreeNAS, D-Link camera, HiSilicion DVR, Philips Hue smart light, OpenWRT fire alarm, LoRA sensor gateway, Alexa Echo Dot2 and Google Mini smart speakers. Code injection and authentication bypass are the primary sources of vulnerabilities. We also consider the unrestricted outbound access, which is commonly seen in incidents after the IoT devices are compromised [23].

1.1.2 Privacy leakage and physical hazard that caused by environment-device and cross-device interactions

Another important part of the IoT security issue is that: attackers can leverage environment-device and cross-device interactions to cause privacy leakage and physical hazard. Table 1.1 shows a list of recently reported IoT breaches. A number of breaches are related to privacy leakage. For Nest cameras, the attacker can steal the private video/image. For Samsung smart TVs, the attacker can remotely turn on the audio recording (via backdoor access) for eavesdropping. Other breaches can cause physical hazards in smart homes. For example, the attacker can open smart doors or smart windows to break into the house. Besides, the attacker can control heaters to overheat the bedroom, or control the stove to cause fire, or remotely control smart toilets to cause water overflow, especially dangerous for families with babies, people with disabilities and

Device	IoT Breach	Attack Type
Nest Camera	Attacker steal private video/image and demand subscribe[14]	Privacy leakage
Multiple Smart Doors	Attacker open smart door to break into the house[16]	Physical hazards
Multiple Smart Heaters	Attacker can control the heater to overheat the bedroom [15]	Physical hazards
Samsung Smart TV	Attacker can remotely turn on audio recording for eavesdropping[22]	Privacy leakage
Multiple Smart Stoves	Attacker can control the stove to cause fire[20]	Physical hazards
Multiple Smart Window	Attacker can open the smart window to break into the house[19]	Physical hazards
Philips Hue Smart Light	Attacker can use drone to turn off smart light to cause a blackout[17]	Physical hazards
Multiple Smart Toilets	Attack can remotely control smart toilet to cause water overflow[21]	Physical hazards

Table 1.1: Recently reported IoT breaches. Attackers are using IoT devices to cause privacy leakage or physical hazards.

the elderly. An interesting case shows that the attacker can even use a drone to turn off smart light to cause a blackout of a house. From these breaches, we can see that a wide range of IoT devices is being used to cause privacy leakage or physical hazard. This is because the IoT devices can interact with the physical environment around them.

For such attacks that cause privacy leakage or physical hazards, the behaviors of a single IoT device appear to be normal, but the environment-device or cross-device interaction patterns will deviate from normal environment-device or cross-device interaction patterns. For example, the behavior of opening the smart window may appear to be normal. However, when the temperature is low, such behavior of opening the smart window is likely to be a malicious attack to cause physical hazards.

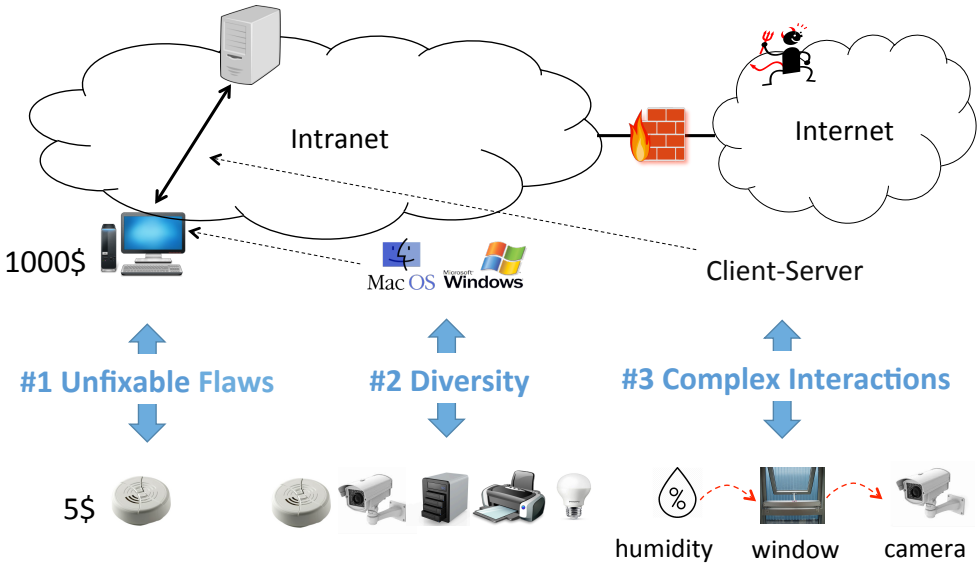

Figure 1.2: What is unique about IoT devices and challenges for IoT security.

1.2 Limitations of Current Approaches

Today's IT security ecosystem, which relies on a combination of static perimeter network defenses (e.g., firewalls and intrusion detection/prevention systems), the ubiquitous use of end-host

based defenses (e.g, antivirus), and software patches from vendors, is fundamentally ill-equipped to handle IoT security issues (Figure 1.2). Specifically, the scale, heterogeneity, use cases, and device and vendor constraints of IoT devices means that traditional approaches fall short along three key dimensions:

- **Learning single-device behaviors**: To protect an IoT device, the security administrators or users need to learn and distinguish its benign behaviors (e.g., a camera's zoom-in behavior) and malicious behaviors (e.g., a camera being code injected). The current approaches rely on security administrators or users to manually learn these behaviors and configure ACLs (access control lists) to restrict the behaviors of a single IoT device. However, the scale and diversity of IoT devices make it hard to manually learn single-device behaviors in a correct and efficient manner, e.g., an ACL rule to prevent camera code injection may block legitimate accesses as well. Also, zero-day attacks are common in IoT attacks, making it even harder to manually learn and distinguish benign/malicious IoT behaviors when the historical records are polluted.
- **Learning complex interactions**: The complexity of the environment-device interactions and cross-device interactions makes it hard to use permission mechanisms (e.g., role-based access control) to protect IoT devices from privacy leakage or physical hazard. A model that can capture such complex interactions is needed. Moreover, even if such a model is available, there are several issues for learning the model. First, for a particular smart home or an IoT device instance, there may not be sufficient historical data to learn the interaction model, which will result in high FPs and FNs. For example, if a user just deployed the camera and there will be no historical data to build the interaction model. To obtain more data, a natural way is to collect the usage data in a centralized place (e.g., cloud back-end) and learn a global model from all the data. However, in this case, the raw data on IoT devices will flow out of the smart home, which will raise privacy concerns from the users.
- **Enforcement architecture**: Even if there are mechanisms to learn the single-device behaviors and complex interactions and convert into IoT security policies, the current security enforcement architecture based on host-centric defense and perimeter-centric defense is too coarse-grained to enforce IoT security policies correctly and efficiently. IoT devices typically do not run full-fledged operating systems, require low-power consumption, and are resource-constrained. Moreover, the longevity of these devices means that vulnerable devices (e.g., default passwords, unpatched bugs) remain deployed long after vendors cease to produce or support them. Also, since IoT devices operate deep inside the network, traditional perimeter defenses are ineffective. Thus, traditional host-centric or device-centric mechanisms (e.g., antivirus, patches) are impractical to expect in an IoT world. Finally, given that the environment and device behaviors can change rapidly, we need to rapidly reassess and update the security posture. Unfortunately, today's security enforcement schemes stem from a static mindset and cannot handle such dynamics.

1.3 Fine-grained Detection and Prevention System for IoT devices

To address the fundamental challenges of current approaches, we propose to build a fine-grained detection and prevention system for IoT devices. Our system can learn single-device behaviors as well as complex IoT interactions, generating regulating security policies, and enforce such security postures in a context-based and agile manner.

More specifically, our solution has three main pieces as listed below.

1.3.1 RADIO: Robust Behavioral Anomaly Detection for IoT devices

RADIO is a robust learning mechanism to learn single-device behavioral models for IoT devices. Designing a mechanism to learn single-device behaviors is challenging on two fronts. First, we need a behavior model tailored towards IoT devices to abstract the key characteristics of the device behaviors (e.g., commands or arguments used) from network traffic. Second, in practical settings, the network traces for learning normal behavior models are unlabeled and potentially polluted (e.g., with zero-day attacks). We address these challenges in designing RADIO. We design a novel learning mechanism that can build benign behavior models (finite-state-machines) for IoT devices, from unlabeled and potentially polluted network traces. To handle pollution, RADIO integrates an outlier detection mechanism with the FSM inference algorithm to generate multiple candidate models, and leverages IoT-specific heuristics to filter out polluted models and selects the best benign model.

1.3.2 LoFTI: Learning Context-based Security Policies via Federated Multi-Task Learning

LoFTI is a distributed learning mechanism to learn complex IoT interactions in a privacy-preserving manner. Designing such a learning framework is challenging on two fronts. First, the accuracy is constrained by insufficient data in some smart homes and the diversity of IoT access patterns across different smart homes. Second, since we rely on usage patterns of IoT devices, users will have privacy concerns. We address these challenges in designing LoFTI. Based on prior user studies, we identify six general types of features (e.g., environment variables such as temperature or humidity) to capture contextual access patterns. We build a simple machine learning model with a temporal structure to achieve a good trade-off between accuracy and communication/computation cost. We design a custom data augmentation mechanism to address the issue of unbalanced data in learning.

1.3.3 PSI: Precise Security Instrumentation

PSI is a new enforcement architecture that can enforce the context-based and agile security policies needed to secure the IoT devices. Designing such an enforcement architecture is difficult on two fronts. The first issue is how to redesign the data plane and control plane of network enforcement to support the context-based and agile policies needed to secure IoT devices. To address

this, PSI protects each IoT device with dedicated software middleboxes enforcing context-based, agile policies, leveraging recent advances in SDN and NFV, The second issue is how to address the scalability and responsiveness issue caused by supporting more fine-grained policies (i.e., context-based and agile policies). To address this issue, PSI provides a scalable and responsive controller for orchestration using proactive forwarding, horizontal scaling, and prefetching techniques.

Next, we will discuss how to evaluate our designs and summarise the results.

1.4 Summary of Results

In evaluating the proposed solutions, we focus on answering the following questions:

- *How accurate is RADIO against single-device IoT attacks (e.g., code injection), and how robust is RADIO against polluted network traffic?* In Section 4, we evaluate RADIO and show that RADIO achieves low FPs and FNs (FPR $< 1\%$ and FNR $< 0.01\%$) comparing with prior anomaly detections [35, 39, 53, 88, 95] (FPR $> 5\%$ and FNR $> 57\%$). In Section 4, we also show that RADIO is robust against polluted traces (FPR $< 1\%$ and FNR $< 0.01\%$ when the percentage of polluted traffic ranges from 1% to 15%), and RADIO's performance is robust across a range of configuration settings (FPR $< 1\%$ and FNR $< 0.01\%$) and does not require fine-grained tuning.
- *How accurate is LoFTI against IoT interaction attacks (e.g., open the smart window to break-in), and how good is LoFTI's trade-off between accurancy and communication/computation cost?*

  In Section 5, we evaluate LoFTI and show that: LoFTI achieves high detection accuracy and low FPs and FNs when compared with single-home learning and all-home learning, reducing FNR by 24.2% and FPR by 49.5%; LoFTI achieves high coverage over the contextual policies needed when compared with the current approach to manually generating contextual policies; LoFTI's key features are effective for detecting malicious IoT access causing physical hazards or privacy leakage. In Section 5, we also show that: LoFTI's simple SVM-based model with temporal structure can achieve a good trade-off between accuracy and communication/computation cost.
- *How effective is PSI's enforcement, and how scalable is PSI for supporting fine-grained enforcement?* In Section 6, we demonstrate the security benefits by showing that: when working against stealthy attacks, PSI identifies and mitigates 35% more attacks than a distributed Firewall/IPS solution. In Section 6, we also demonstrate the scalability, responsiveness and resilience of PSI by showing that: proactive context-based forwarding reduces the end-to-end latency of enforcement by at least $10X$ over the baseline performance. PSI's DAG prefetching mechanism reduces security downtime during FSM transitions from seconds scale to zero. With the optimizations, a single PSI controller can support a network with 100K devices, and can support complex policies with up to 10 states with a size-10 DAG for each state. Our scale-out scheme cuts the enforcement response time down from seconds to $10ms$ even in the presence of an adversary.

1.5 Organization

The rest of the thesis is organized as follows.

Chapter 2 discusses two motivating examples to demonstrate the uniqueness of IoT security and highlight the limitations of current security approaches.

Chapter 3 presents an overview of the fine-grained network detection and prevention system. It first describes the high-level vision of the system. Then it lists out the challenges to realize the vision. After that, it shows our insight to address the challenges. Next, it describes the practical techniques designed based on our insight.

Chapter 4, 5 and 6 describes the three key contributions of this thesis: 1) Robust Behavioral Anomaly Detection for IoT devices (RADIO) - a robust learning mechanism to learn single-device behavioral models for IoT devices; 2) Learning Context-based Security Policies via Federated Multi-Task Learning (LoFTI) - a distributed learning mechanism to learn complex IoT interactions in a privacy-preserving manner; 3) Precise Security Instrumentation (PSI) - a new enforcement architecture that can enforce the context-based and agile security policies needed to secure the IoT devices.

Chapter 7 shows how to integrate RADIO, LoFTI and PSI as an end-to-end system, and demonstrates two use cases for the integration.

Chapter 8 discusses the related works of RADIO, LoFTI and PSI.

Chapter 9 summaries the contribution of this thesis, discuss the limitations and future works.

Chapter 2

Motivation

To highlight the security challenges for IoT devices, we use several motivating examples to show why existing mechanisms are ineffective for securing IoT devices.

2.1 Motivating example 1: HackingTeam incidence

We describe a realistic attack scenario to highlight the above security issues and why existing security mechanisms, including host-based detection [25, 28], signature-based detection [94, 100], ACLs and network-based anomaly detection [42, 45, 68, 72, 76, 95, 113], are impractical to protect enterprise IoTs. Figure 2.1 depicts a real-world security incident (an IoT-based intrusion against the HackingTeam - a famous hacker organization[23]). In step 1, the attacker found a zero-day flaw and compromised the router using code injection. In step 2, after compromising the router, the attacker obtained access to the NAS via a backdoor (authentication bypass). In step 3, he used the NAS to remotely login to the Blackberry Server and exfiltrated valuable data.

2.1.1 Broken network enforcement

Current enforcement architecture, including host-based enforcement [25, 28] and hardware middleboxes are impractical to enforce the context-based and agile security postures needed to secure the IoT devices. Host-based enforcement [25, 28] is impractical because IoT devices have limited resources to support any host-based detection. Therefore, in step 1, the attacker chose to attack the router rather than the webserver. Hardware middleboxes use static IP-based rules, and cannot enforce context-based and agile policies needed. For example, a hardware middlebox cannot enforce a policy to prevent NAS's remote access to the server in step 3, under the context that the NAS has been accessed by the router.

Lack of behavioral model for IoT devices

Current detection mechanisms are unable to accurately infer and restrict the behaviors for IoT devices. Signature-based network detection [94, 100] is ineffective because zero-day attacks are very common for IoT devices. For example, in step 1, attacker easily found a zero-day code

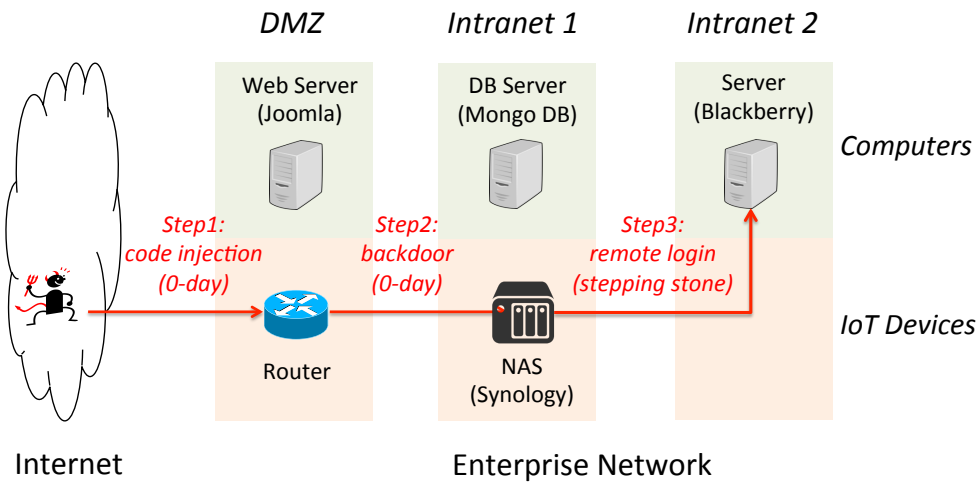

Figure 2.1: Attackers intrude the enterprise via IoT devices.

injection vulnerability within two weeks and evaded the signature-based detection between the Internet and the DMZ zone. Network-layer access control (ACLs) fails because of the scale and diversity of the enterprise IoT devices. In step 2, the administrators cannot simply block the access from the DMZ to the NAS because the NAS was used to backup some servers in DMZ. Current network anomaly detection is easy to evade because they either focus on low-level traffic statistics [68] or only model certain types of attack behaviors with specific patterns (e.g., malware C&C or port scanning) [42, 45, 72, 76, 95, 113]. In step 3, the attacker performs remote login at a low frequency to avoid causing flow-statistics anomalies. Also, the remote login behavior is a legitimate computer behavior (although it is abnormal for a NAS), so the anomaly detection specialized for computer activities[42, 45, 72, 76, 95, 113] will not raise any alert.

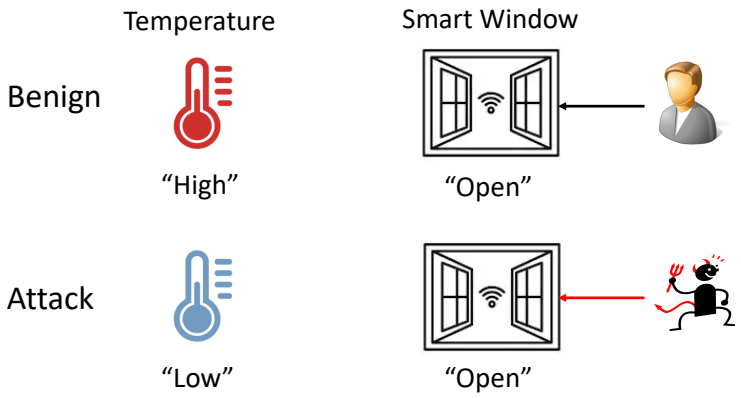

Figure 2.2: Benign/malicious smart window scenarios.

2.2 Motivating example 2: Physical break-in against smart homes

To prevent privacy leakage and physical hazards, current approaches[43, 48, 70, 78, 102] rely on access control policies with contextual information to decide if an action of an IoT device should be allowed or blocked. In Figure 2.2, we present a motivating example to show how contextual policy can prevent physical hazards or privacy leakage. A user will access and open the smart window when the temperature is high. The attacker, however, will not follow such benign *context* -"temperature is high" and will open the window to break into the house even when the temperature is low. Therefore, if there is a contextual policy that only allows access to open the window when the temperature is high, this policy is able to prevent the attack.

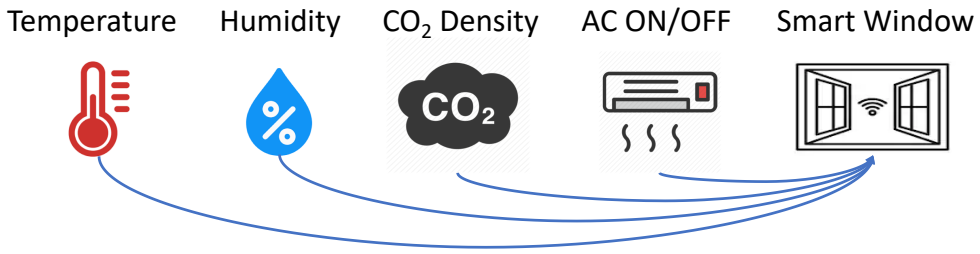

Figure 2.3: Complex interactions across smart windows, AC units and the environment.

2.2.1 Complex interactions are hard to capture

There are two approaches to generate contextual policies. The first approach is to *preinstall* contextual policies specified by developers or users [70, 102]. The second approach is to use *runtime prompts* with contextual information to let the user make the allow/block decision [65, 70, 78]. Both approaches heavily rely on the knowledge and effort of the developers/users to anticipate malicious behaviors. Thus is easy to misconfigure and have poor coverage over all contextual policies needed, causing high FPs/FNs. For example, in Figure 2.3, whether the smart window will be opened depends on multiple factors including temperature, humidity, CO_2 density and whether AC is on/off, and such context can be different between smart homes. Suppose a handcrafted policy is preinstalled to block the access to the smart window when the temperature is above preconfigured thresholds, such thresholds can be easily misconfigured and result in high FPs/FNs. Then, suppose there are runtime prompts with temperature, humidity, CO_2 density and whether AC is on/off, such context can be different every time the smart window is accessed. Considering the number of IoT devices in a smart home, there could be hundreds of prompts per day and cause the user to be unable or unwilling to handle all the prompts.

2.3 Summary

In this chapter, we walked through two motivating examples. The first example shows the broken network enforcement and the lack of single-device behavior models for IoT devices. The second

example shows that the complex interactions of IoT device are hard to capture. We will show how to address the IoT security issues in the next chapter.

Chapter 3

System Overview

In this chapter, we first introduce the high-level vision for our fine-grained detection and prevention system for IoT devices. Then we discuss the challenges to realize this vision. After that, we share our insight to address these challenges. Next, we show the practical technical design based on our insights.

3.1 High-level vision

Figure 3.1 shows a high-level vision of the fine-grained detection and prevention system for IoT devices. There are three key components - RADIO, LoFTI and PSI. The RADIO module learns the single-device behavioral models from single-device historical records. The LoFTI module learns the environment-device and cross-device interaction models from interaction historical records. Then the PSI module converts the single-device behavioral models and the interaction models into concrete IoT security policies and enforce such policies to protect IoT devices. In PSI, a logically centralized control plane monitors the contexts of different devices and the operating environment and generates a global view for the policy enforcement, and customized micro middleboxes (micro network-security functions) are acting as security gateways for each IoT device. To enable immediate deployment, we assume the enterprise has a well-provisioned on-premise cluster with a pool of commodity server machines. In a home scenario, we envision an upgraded version of an IoT router (e.g., Google OnHub) with computing capabilities. Each IoT device's first-hop edge router or wireless access point (AP) is configured to tunnel packets to/from the device to the cluster or an IoT router.

Next, we describe the workflow of the fine-grained detection and prevention system. Initially, PSI instantiates and configures individual micro middleboxes and the necessary forwarding mechanisms to route packets of the IoT devices to these micro middleboxes. Then, from the events from the devices and micro middleboxes, the RADIO module will infer a single-device behavioral model for each IoT device, and the LoFTI module will learn the environment-device interaction and cross-device interaction patterns for each smart home/enterprise. Then single-device behavioral model inferred by RADIO and the environment-device interaction and cross-device interaction pattern learned by LoFTI are converted into context-based and agile security postures.

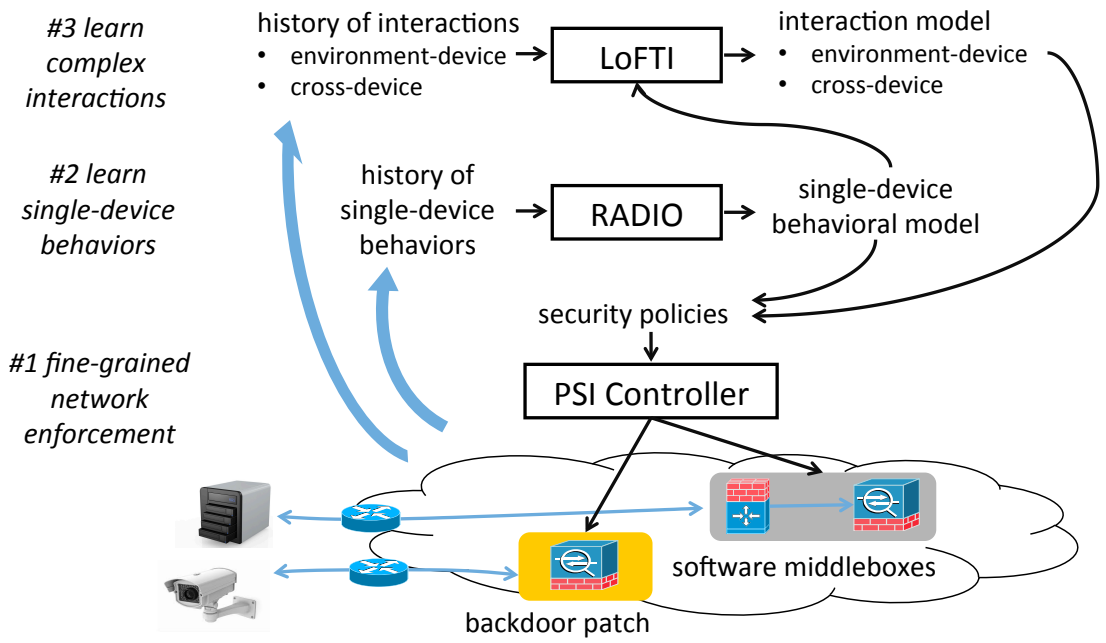

Figure 3.1: High-level vision of the thesis.

Finally, PSI enforce such postures to protect the IoT devices from code injection, authentication bypass, privacy leakage or physical hazard.

Next, we will walk through our vision of the three key components - RADIO, LoFTI and PSI.

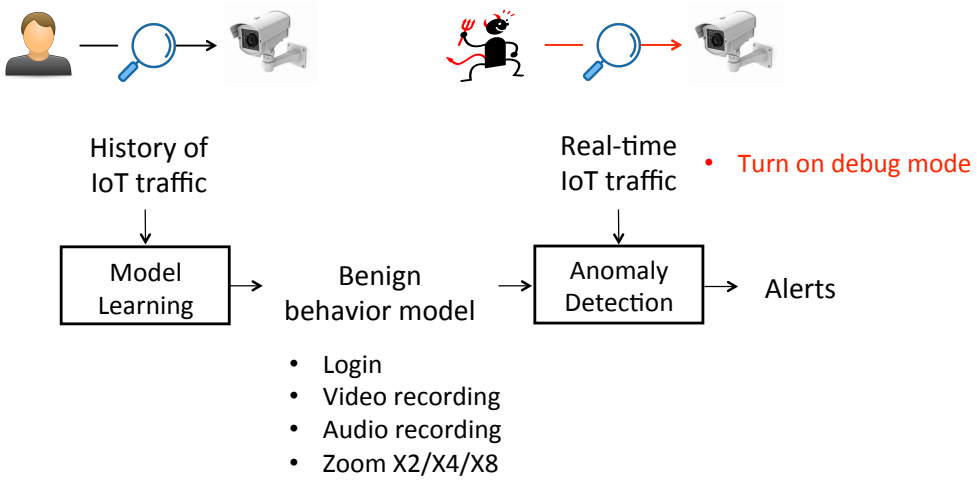

Figure 3.2: RADIO's vision.

3.1.1 RADIO module

RADIO's goal is to build a behavioral anomaly detection mechanism by learning single-device behaviors for IoT devices. As shown in Figure 3.2, RADIO will observe the history of IoT traffic. Then, RADIO will provide a model learning module to learn the benign behavior model for IoT devices based on history. For example, RADIO will use the model to capture the login, video recording, audio recording and zoom behaviors of the camera. At runtime, RADIO will observe the real-time IoT traffic and compare it with the benign behavior model and raise an alert on any anomaly. For example, the malicious behavior to turn on the debug mode for the camera will deviate from the benign behavior model and RADIO will raise an alert on this malicious behavior.

3.1.2 LoFTI module

In LoFTI, we aim at learning contextual access control policies from the historical records in smart homes. Suppose the history of IoT accesses with the corresponding context and action is given (e.g., temperature, humidity, the state of the AC and the actions performed on the smart window in Figure 3.3). The learning mechanism will learn a function F, so that the F can decide to allow or block an access in the future based on the corresponding context and action. For example, in Figure 3.3, the history shows that the user will open the window when the temperature and humidity are high (i.e., feels hot), and the attack will open the window even when the temperature and humidity are relatively low. Then our goal is to learn function F to decide to block/allow the access to open the window based on the temperature and humidity in the future.

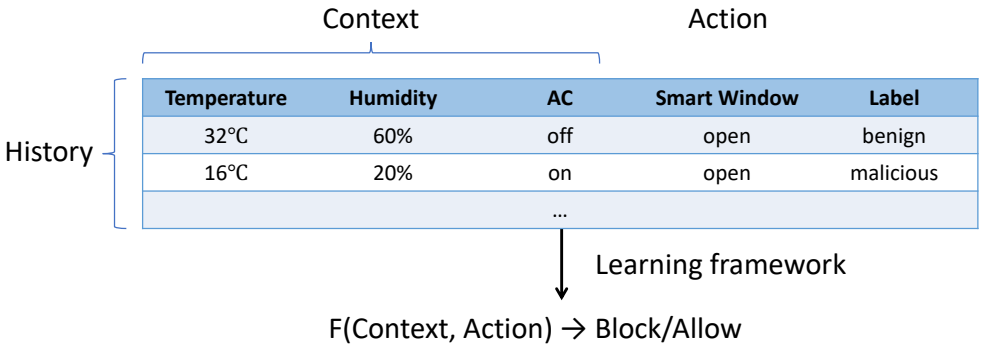

Figure 3.3: LoFTI's vision.

3.1.3 PSI module

The goal of PSI is to enable context-based and agile enforcement to protect IoT devices. Figure 3.4 shows a high-level vision of PSI. PSI envisions customized micro-middleboxes that act as security gateways for each IoT device. A logically centralized PSI controller monitors the contexts of different IoT devices and the operating environment and generates a global view for policy enforcement. Based on this view, it instantiates and configures individual middleboxes

and the necessary forwarding mechanisms to route packets to these middleboxes. This vision is quite general and can naturally support a range of IoT management models; e.g., directly connected devices vs. IoT hubs vs. smartphone-controlled. To enable immediate deployment, we assume the enterprise has a well-provisioned on-premise cluster with a pool of commodity server machines. In a home scenario, we envision an upgraded version of an IoT router with computing capabilities. Each IoT device's first-hop edge router or wireless access point (AP) is configured to tunnel packets to/from the device to the cluster or an IoT router.

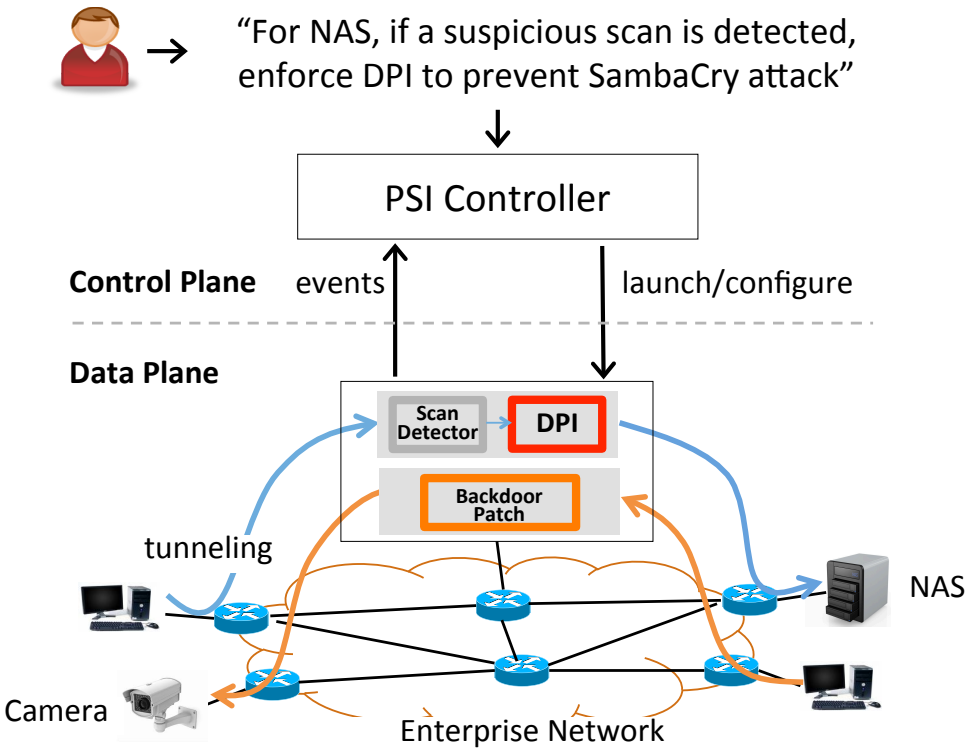

Figure 3.4: PSI's vision.

3.2 Challenges

There are several challenges to realize the above high-level vision and each component.

3.2.1 Learning single-device behaviors

For RADIO, it is hard to learn single-device behaviors via existing security solutions. First, manually learning and crafting attack signatures, such as in existing IDS or IPS systems (e.g., Bro IDS [94], Snort [100] and Suricata [35]), is not effective since zero-day exploits are commonly utilized for attacks against IoT devices. Second, statistical flow-based detections and learning [42, 45, 68, 72, 76, 95, 113] often work well with attacks that exhibit statistical patterns (e.g., port scanning, C&C), but attacks against IoT devices in enterprise networks are often

stealthy. This is evident in the HackingTeam infiltration where the attackers mention that they specifically targeted one vulnerable IoT device with low attack frequency to avoid raising any alerts from statistical anomaly detection methods. Third, the scale and diversity of enterprise IoT deployments render manually learn and configure access control to be ineffective.

3.2.2 Learning complex interactions

Our goal in LoFTI is to build a machine learning framework to automate and simplify the process of generating contextual access control policies for IoT devices. The machine learning framework will learn the contextual policies from the historical records of contextual IoT access, and use the learned contextual policies to decide to allow or block a particular IoT access in the future.

However, there are two goals that are hard to simultaneously achieve in designing such a machine learning framework: *accuracy* and *privacy*. To see why, consider two strawman solutions. On one extreme, if each home relies on its own historical records (*single-home learning*), some smart homes may have insufficient data to learn an accurate contextual access control model. On the other extreme, to avoid this insufficient data issue, we could collect all the historical records of all smart homes in a centralized place (e.g., AWS/Azure/Google cloud) and learn one model (*all-home learning*). However, such all-home learning may not be accurate since different smart homes can have diverse contextual access patterns. Also, this all-home learning mechanism requires the historical data of a smart home to be transferred out of the smart home (e.g., to the cloud), and will raise users' privacy concerns.

3.2.3 Enforcement architecture

There are several challenges to realize the vision of PSI based on current security enforcement architecture with hardware appliances and static routing. First, in terms of deployment complexity, we need a dedicated hardware appliance attached to every device. Even ignoring the complexity of rewiring the network, this is an uphill task given the number of wireless, mobile, and virtualized devices. The second issue is cost; adding as many physical hardware appliances as there are devices is a non-starter for medium-to-large scale enterprise networks with tens of thousands of devices. Third, such an omnipotent device that can dynamically reconfigure its traffic processing does not exist today. While future solutions (e.g., [104]) may offer such a consolidated appliance, we have to embrace the practical concern that security functions are fragmented across different vendors with diverse capabilities. Finally, even if we had the appliances, the policy abstractions offered by current frameworks are not expressive enough to capture agile security postures.

3.3 Our insights

To address the above challenges, we rely on several insights about IoT devices.

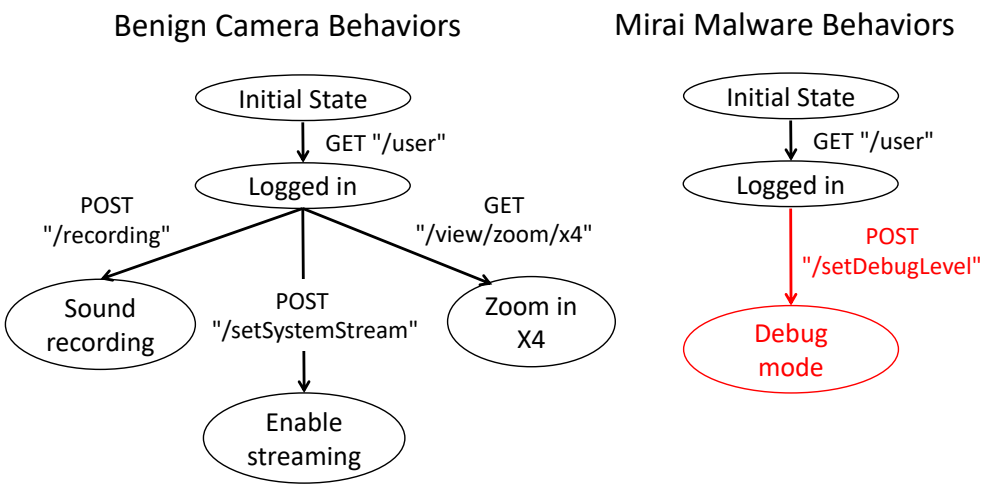

Figure 3.5: The limited benign behaviors of a camera and the attack behaviors (Mirai Malware[40]) against it.

3.3.1 A single IoT device has a limited set of behaviors

Our key insight of RADIO is that each IoT device is typically used for a limited number of specialized use cases. Therefore, each device has a limited set of behaviors. To perform attacks, attackers are likely to deviate from the specialized use cases, so attacks will be logically different from the limited device behaviors. Note that both the benign behaviors and malicious behaviors can be supported by the network protocols employed by the IoT device. Therefore, modeling the network protocol of the IoT device is not sufficient to distinguish benign/malicious behaviors. For example, Figure 3.5 shows the difference between the benign behaviors of a camera and the Mirai malware[40] against it. The benign behaviors include logging in, recording sound, enable streaming and zoom in. The attack behavior, however, includes sending a HTTP POST message with URI "/setDebugLevel" to force the camera to debug mode (so the attacker can change the camera's firmware). Thus, detecting *behavioral anomalies* can help us identify IoT-related malicious activities.

3.3.2 Federated Multi-Task learning can address the insufficient data issue and privacy issue for IoT devices

Our insight for LoFTI is that we can apply Federated Multi-Task learning[110] to address the issues of *insufficient data*, *privacy* and *diversity*. The high-level process of Federated Multi-Task learning is shown in Figure 3.6. In step 0, the cloud backend will initiate the model F and task correlation Ω. In step 1, the cloud backend will send the model parameter w_i to each smart home i. In step 2, the edge node in smart home i will calculate the local model parameter update Δw_i based on local data X_i and model parameter w_i. In step 3, the edge node in smart home i will send the model parameter update Δw_i back to the cloud backend. In step 4, the cloud backend will update the task correlation Ω and the model F based on updates from all smart homes $\{\Delta w_i\}$. Then step 1, 2, 3 and 4 will be executed for multiple iterations until the model F

converges.

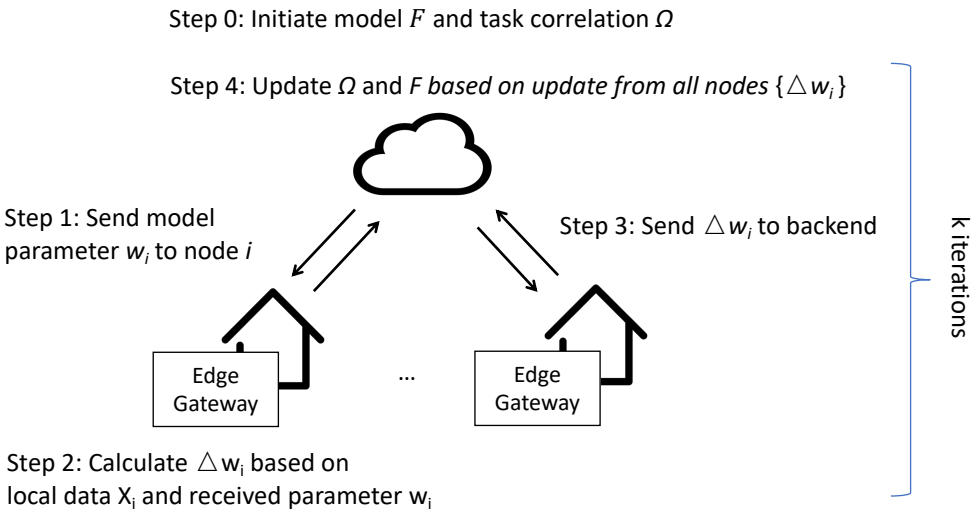

Figure 3.6: Federated Multi-Task learning workflow.

Federated Multi-Task learning can address *insufficient data* issue because the model F is learned from the data $\{X_i\}$ from each smart home in a distributed manner. The *diversity* issue is addressed as each smart home is considered as an individual learning task, and the task correlation Ω can capture the diverse correlations (e.g., distinguishing common part and unique part) and allow each task to be customized. There will be fewer *privacy concerns* for the user as the raw data X_i for smart home i is kept locally within each smart home, only model parameters $\{\Delta w_i\}$ is transmitted to the cloud backend. Note that highly skilled attackers may still be able to infer private information from the model parameters (with higher cost). However, it is possible to combine Federated Multi-Task learning with masking or differential privacy techniques to further protect the private information in the model parameters.

3.3.3 SDN and NFV can enable the context-based and agile enforcement for IoT devices

In PSI, our insight is that SDN and NFV can enable the context-based and agile enforcement for IoT devices. More specifically, PSI can decouple the deployment of security appliances from topological constraints by *tunneling* a device's traffic to a server cluster. This cluster can provide an appropriate appliance for any traffic, on demand. To address the cost issue, PSI leverages NFV to build "tiny" virtualized appliances to share commodity hardware and reduce cost. Note that one commodity server can support up to hundreds of such appliances[86]. To address the lack of dynamic omnipotent appliances, PSI uses SDN capabilities to compose existing appliances (e.g., Snort or Bro) to dynamically steer the traffic within the cluster.

3.4 Making Fine-grained Detection and Prevention System Practical

3.4.1 Practical Challenges for RADIO

To learn the network behaviors of a single IoT device, the main challenges are the lack of network behavior models for IoT devices and how to address the data pollution issue in realistic setting:

- *Abstraction for modeling IoT behaviors*: It is obvious that the limited set of high-level IoT behaviors are concealed in complex network traffic patterns. It is hard to decide how to abstract the key characteristics of the complex network traffic patterns to represent the high-level IoT behaviors. Also, the abstraction needs to be expressive enough to capture IoT behaviors contained in a wide range of network protocols.
- *Learning IoT behavior model from unlabeled behavior samples*: It is difficult to obtain clean and labeled samples in practical setting [111]. It is difficult if not impossible to obtain a comprehensive set of attack behavior samples since zero-day attacks are common for enterprise IoT devices. Benign behavior samples are also difficult to enumerate since the management cost to generate all possible benign samples in a dedicated clean environment is high considering the scale and diversity of IoT devices. Passive monitoring over real deployment is the most practical solution but it is prone to pollution. Therefore, RADIO has to provide a robust learning mechanism that can learn benign behavior models from unlabeled, potentially polluted behavior samples.

RADIO tackles these issues and detects behavior anomalies based on benign behavior models learned from unlabeled, potentially polluted network traffic. To provide a precise abstraction for modeling the network behaviors of IoT devices, we identified three key network characteristics of IoT behaviors, and carefully define the states and transitions of the FSM model to abstract these key characteristics in Section 4. Then to learn the FSM model from IoT traffic, the main issue is to learn IoT behavior model from unlabeled network traces. There are two subproblems. The first subproblem is that it is hard to infer the FSM states with unlabeled behavior samples. Existing FSM inference algorithms either rely on both positive and negative samples [57, 93] or rely on extra structural information (e.g., while loop structure of program) [103] to infer the FSM states. However, considering IoT traffic, it is hard to obtain labeled positive samples (attack samples) or extra structural information (no program structure). To address this issue, RADIO provides a modified FSM inference algorithm to leverage the internal correlation inside the IoT traffic to infer the FSM states. The second subproblem is that the samples can be polluted, and malicious behaviors will be included in the benign behavior models learned. To handle pollution, RADIO novelly integrates an outlier detection mechanism with the FSM inference algorithm to generate multiple candidate models, and leverages IoT heuristics to filter out polluted models and selects the best benign model.

3.4.2 Practical Challenges for LoFTI

There are several challenges in making LoFTI practical:

- **Defining key features to extract contextual IoT access patterns**: Given the wide range of information in smart homes, it is hard to define a set of key features that are expressive enough to extract various contextual IoT access patterns. For example, a naive solution may use MAC/IP address of the IoT devices as a key feature, which are not effective and similarity in IP address can even be noise to the learning framework.
- **Building model with high accuracy and low computation/communication cost**: More complex ML models (CNN, RNN, LSTM) often have higher accuracy as they can encode more information than less complex models (Naive Bayes, SVM). However, complex machine learning models also result high computation/communication cost, especially in a distributed learning framework like LoFTI. It is hard to achieve a good trade-off between accuracy and cost.
- **Addressing unbalanced contextual IoT access records**: In reality, the historical records for privacy leakage/physical hazards are much fewer comparing with benign IoT access records. For example, there can be more than 10000 records in 100 smart homes about user opening the smart window in the historical records, with less than 100 records about attacker opening the smart window. As a result of such unbalanced datasets, the decision boundary will be pushed towards the attack space and result in high FNs.

To address these issues, we explore the design space of LoFTI in terms of feature extraction, model building and dataset construction and address the challenges above. We generalize 6 types of key features to capture the contextual IoT access patterns and demonstrate their effectiveness in distinguishing benign/malicious IoT access. We build a simple SVM-based model with extra temporal structures to achieve high detection accuracy and low communication/computation cost. We provide a specialized data augmentation mechanism to balance the contextual access historical record dataset.

3.4.3 Practical Challenges for PSI

For building the enforcement architecture, it is hard to design an expressive policy abstraction to allow the admins to express context-based and agile policies. Also, it is hard to design a scalable and responsive controller for the enforcement architecture. More specifically, there are two subchallenges:

- *Expressive policy abstractions* Traditional policy abstractions (e.g., in firewall, IDS, ACL) that rely on a simple if-match-then-action paradigm are not sufficient for the agile and context-aware defenses we envision. To this end, we develop an expressive policy abstraction based on an intuitive combination of finite state machines to capture the security state and directed acyclic graphs to capture context-based security actions. We design a Policy Engine that interprets the policy abstraction and computes the real-time security intent updates for each IoT device based on the current context.
- *Scalable and responsive orchestration* Naive orchestration schemes to realize PSI would cause several scalability and responsiveness problems. For instance, naively sending the first packet of every context change to a central controller introduces new control plane attacks [107]. Naive scheme to launch new middlebox instances at every context change

wastes time and resources (CPU/memory). Finally, if the orchestration cannot scale out, the attacker can easily overload the system of PSI To address these issues, we develop a scalable orchestration platform using a combination of proactive orchestration, pre-fetched installation of middleboxes, and horizontal scaling.

To address these issues, PSI makes the following contributions. To address the policy abstraction issue, we design a PSI policy abstraction that can express agile and contextual traffic processing. This allows us to express rich multi-stage security-relevant processing mapped to a security-relevant state for each device; e.g., a host in a normal state is subject to simple IDS-followed-by-firewall but in "suspicious" state may be subject to additional on-demand exfiltration detection modules. We also provide mechanisms to incorporate legacy policies that need to be applied to a group of devices. To address the scalability and responsiveness issue, PSI applies the SDN/NFV mechanisms. However, naively applying SDN/NFV mechanisms in security context is problematic and can introduce new avenues for DoS attacks. We develop a scalable orchestration platform by synthesizing three key ideas: (a) proactive forwarding schemes based on logical tags that do not need to involve the controller; (b) effective techniques for horizontally scaling the controller infrastructure; and (c) prefetching future enforcement states to improve responsiveness.

3.5 Summary

In this chapter, we first show the high-level vision of the fine-grained detection and prevention system. Then, we discuss the challenges to realize this vision. After that, we present our insights about how to address the challenges and our design to make the system and its key components practical. Next, in Chapter 4, 5 and 5, we will discuss the detailed technical pieces to realize our vision.

Chapter 4

Learning Single-device Behaviors

In Chapter 3, we have seen the vision of RADIO to learn the single-device behavioral model for IoT devices, and shown that the key questions to make RADIO practical are:

- How to create an expressive abstraction for modeling IoT behaviors?
- How to learn such IoT behavior model from unlabeled behavior samples?

In this chapter, we will answer these two questions with detailed design of RADIO's network behavioral model and robust learning mechanism. After that, we will present the result of evaluating RADIO for its accuracy and robustness.

4.1 Network Behavioral Model for IoTs

Given the *historical network traces*, we need an abstraction for modeling IoT-specific behaviors. The abstraction needs to satisfy two requirements. First, the abstraction needs to be able to capture the key network characteristics of IoT device behaviors. Second, the abstraction needs to be expressive enough to capture IoT behaviors contained in a wide range of network protocols.

4.1.1 Capturing key network characteristics for IoT devices

We start with an example to show what are the key network characteristics for IoT devices and how to capture them. Figure 4.1 illustrates a camera's authentication behavior. The HTTP authentication behavior includes 4 ADUs (Application Data Units). In ADU1, the user is trying to access the camera with the request method `GET` and URI `/user`. Then in ADU2, the camera notifies the user to continue authentication by sending back status code `401`. Then user authenticates using ADU3. The camera notifies the user that the authentication is successful with status code `200` in ADU4. From this example, there are three key aspects of the behavior: 1) the commands and arguments included in each ADU (e.g., the request method, URI and status code); 2) the direction of each ADU, i.e., whether the ADU is sent `to` or `from` the IoT device; 3) the sequence of the ADUs, e.g., the sequence of ADU1 to ADU4 indicates the authentication routine of the camera.

Now, we can think of some strawman solutions for modeling the IoT behaviors. One natural starting point is to use the coarse-grained flow statistics such as packet counts [68]. However,

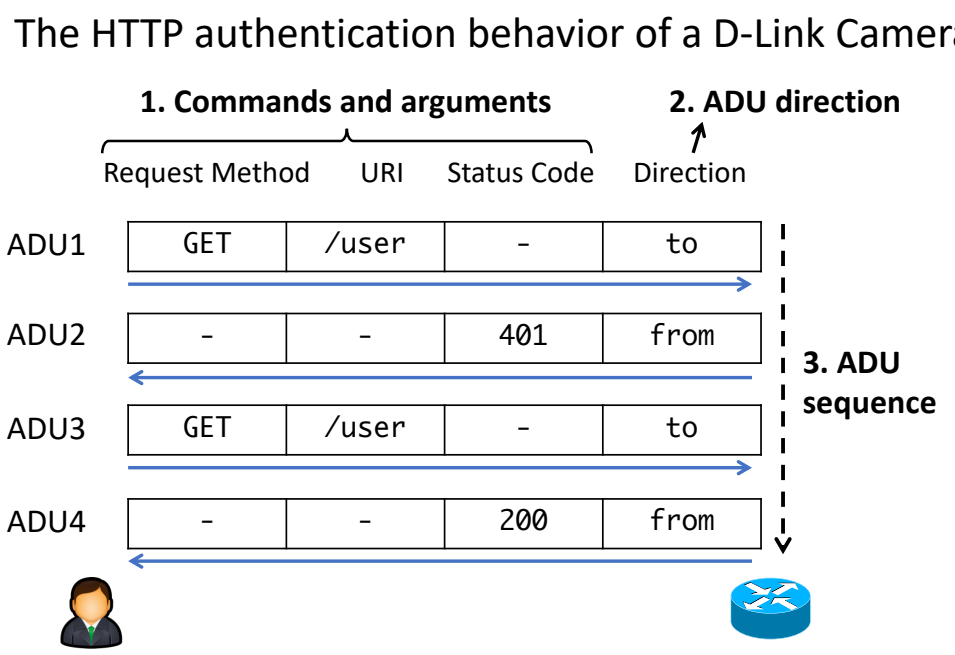

Figure 4.1: A camera's authentication behavior.

the flow statistics do not capture IoT's commands and arguments at application-level (e.g., URI). Alternatively, we can try to build a bag-of-words model that counts the appearance of different ADU field values. However, the bag-of-words model cannot capture the directionality or sequential semantics (e.g., status code `401` appears before status code `200`, otherwise authentication is bypassed).

To overcome the limitations of the strawman solutions, we suggest the following network behavior model. Instead of flow-level statistics, we choose to abstract application-level ADU fields [1] that contain the device-function-specific commands and arguments for enterprise IoTs [2]. To capture the directional semantics, we explicitly include the direction of the ADU (from device or to device) to model the interaction between the user and the device (e.g., HTTP GET from admin to router). Finally, to capture the sequential semantics of a session, we define a FSM model (Finite State Machine) to abstract the sequential pattern from the ADU series.

To capture both ADU fields and ADU direction, we define an *abstracted ADU* m as the combination of a set of ADU fields $\{f_1, ..., f_v\}$ and an ADU direction d_m. The ADU direction d_m captures whether the ADU is sent from the device or to the device. For example, the *abstracted ADU* for ADU 1 in Figure 4.1 is $\{GET, /user, -, to\}$.

To represent the sequence of ADUs in a compact way, we abstract the sequence of ADUs using a FSM model. We define the FSM model as a Mealy Machine [3]. A Mealy Machine FSM is a six tuple $< S, s_0, \Sigma_I, \Sigma_O, \delta, \lambda >$, where S is a finite non-empty set of states, $s_0 \in S$ is the initial state, Σ_I is a finite set of input symbols,Σ_O is a finite set of output symbols, $\delta : S \times \Sigma_I \rightarrow S$ is the

[1] An ADU field is defined as a sequence of lines of characters with special syntax defined by the protocol format.
[2] Existing protocol reverse engineering mechanisms[56] can identify the ADU fields.
[3] Another alternative is Moore Machine. We did not choose Moore Machine as it has "rejecting" states that relies on attack samples (as counter examples) to generate.

transition relation, $\lambda : S \times \Sigma_I \rightarrow \Sigma_O$ is the output relation. To define the transition relation δ and output relation λ, we leverage the *causality* between a request ADU and a response ADU, i.e., at a given state of the FSM (e.g. S_0), a specific request ADU (e.g. GET) will result in a specific response ADU (e.g. 401). Therefore, we define the transition relation as $\delta : S \times \{m_{request}\} \rightarrow S$ and the output relation $\lambda : S \times \{m_{request}\} \rightarrow \{m_{response}\}$, where m is the *abstracted ADU*. For example, for the camera's HTTP authentication behaviors in Figure 4.1, the transition relation at S_0 is $\{S_0\} \times \{GET\} \rightarrow \{S_1\}$ and the output relation at S_0 is $\{S_0\} \times \{GET\} \rightarrow \{401\}$, which means at given state S_0, input GET will yield output 401.

Abstracted CMP Messages

ID	Subset {m_a}	Subset {m_b}
	Direction+Command	Subcommand
1	from + Will	Echo
2	from + Do	Suppress Go Ahead
3	to + Suboption	Terminal Type
4	to + Do	Negotiate About
5	to + Will	Environment Option
6	from + Suboption	Remote FlowControl
7		Linemode
8		New Environment Option
9		Status
10		End

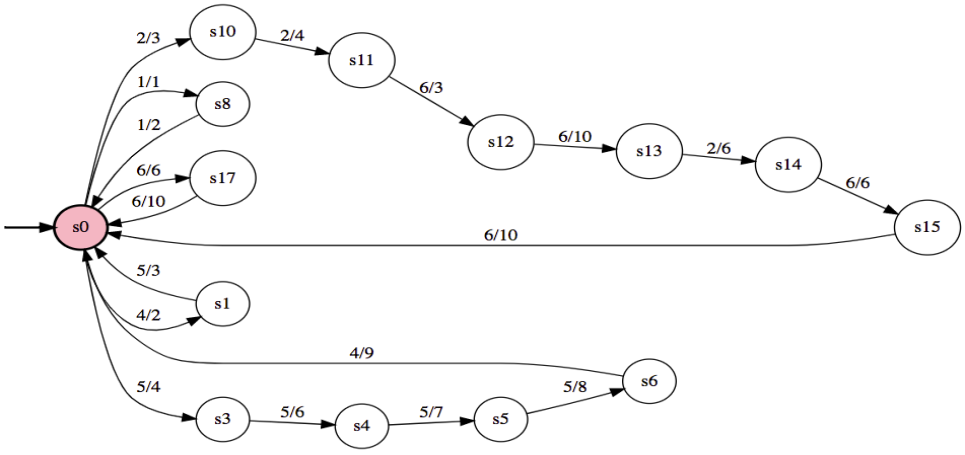

Figure 4.2: Cisco router's behaviors captured by RADIO's FSM model.

4.1.2 Expressiveness to capture IoT behaviors contained in a wide range of network protocols

However, the above definition cannot handle general protocols that may not have request and response ADU pairs (e.g., Telnet). This is because the above definition relies on the *causality* between request and response (i.e., a certain request result in a certain response). To abstract protocols without request and response ADU pairs, our idea is to also leverage the *causality* between the ADU fields inside each ADU, i.e., the occurrence of some ADU fields result in the occurrence of some other ADU fields in an ADU. Figure 4.2 demonstrates the end-to-end

example for Cisco Cluster Management Protocol (CMP), a typical protocol without request and response ADU pairs. In Figure 4.2, the occurrence of some ADU fields (such as `Suboption` command) will result in the occurrence of some other ADU fields (such as `Terminal Type` and `Remote FlowControl`). Based on this idea, an abstracted ADU m can be split into two subsets of ADU fields $m_a = \{f_a\}$ and $m_b = \{f_b\}$, e.g. $\{f_a\}$ is the command and $\{f_b\}$ are the combination of its arguments. At a given state of the FSM, a subset of ADU fields $\{f_a\}$ will result in a specific subset of ADU fields $\{f_b\}$. Therefore, for such protocols, we define the transition relation as $\delta : S \times \{m_a\} \rightarrow S$ and the output relation $\lambda : S \times \{m_a\} \rightarrow \{m_b\}$. After the extension, RADIO's abstraction for modeling IoT behaviors is expressive enough to capture IoT behaviors contained in a wide range of network protocols, including `Telnet`, `HTTP`, `FTP`, `SNMP`, `SMB`, `DNS` and vendor-specific protocols such as `CMP`.

4.1.3 Handling diverse IoT protocols

Now, we discuss how to extend the above behavior model to handle diverse IoT protocols. The key issue here is how to construct the L7 ADUs from the network packets of an IoT protocol. In this part, we start with unencrypted traffic, and will extend to handle encrypted traffic in the next part.

In general, the task of constructing the L7 ADUs from the packets relies on protocol specifications and protocol parsers. For common network protocols (e.g., http, telnet and snmp), we can rely on protocol parsers in IDSes and IPSes. For example, Suricata [35] offers a http parser called the `http.lib` to construct ADUs (e.g., $GET\ /user$) from http packets.

```
ReadRegister - content:"|08 00 04|"; offset:7; depth:3;
R3 - content:"|03 00|"; offset:5; depth:2;
ResponseRegister - content:"|08 00 04|"; offset:7; depth:3;
ReadCoils - content:"|00 00 04|"; offset:7; depth:3;
CoilID - content:"|01 00|"; offset:5; depth:2;
ResponseCoils - content:"|00 00 04|"; offset:7; depth:3;
ReadDiscreteInputs - content:"|04 00 04|"; offset:7; depth:3;
InputID - content:"|02 00|"; offset:5; depth:2;
ResponseDiscreteInputs - content:"|04 00 04|"; offset:7; depth:3;
```

Figure 4.3: Protocol specification of the modbus protocol.

Then, for IoT-specific protocols, we can leverage the protocol specifications. For example, Figure 4.3 shows part of the protocol specifications of the modbus protocol (IoT protocol widely used industry control systems). In the protocol specification, an ADU can be identified by three elements: 1) certain signature in the packet payload (denoted as content); 2) the starting position of the signature (denoted as offset); 3) the length of the signature (denote as the depth). For example, in Figure 4.3, the `ReadRegister` ADU can be identified by the signature pattern of $00\ 00\ 04$, the offset 7 and the depth 3.

Such protocol specifications can be obtained from the design documentation of the IoT protocol or from previous works that can learn protocol specifications from network traces[56].

4.1.4 Handling encrypted traffic

Next, we discuss about the possibility to extend the above behavior model to handle encrypted IoT traffic. The key obstacle here is that the above behavior model is based on L7 ADUs, but for encrypted IoT traffic, we only have visibility to information below L4.

Fortunately, previous works [41, 50] demonstrated the possibility to side channel L7 behaviors from encrypted traffic (e.g., web application behaviors over https). Therefore, in this paper, we demonstrate it is possible to apply the key below-L4 features from previous side channeling works (e.g., packet direction, frame timestamp and frame length) and developed a clustering mechanism to identify L7 ADUs based on the below-L4 information of the encrypted IoT traffic.

The input for our mechanism is a sequence of packets with L4 information [4]. The output of our mechanism is a segmented and typed sequence of ADUs. There are two major steps in our approach:

The first step is to segment the sequence of packets into subsequences of packets, where each subsequence of packets is an ADU. To achieve this, our insight is that the time stamp of the packets in the same ADU (or same pair of ADUs) are much closer to each other than the time stamp of the packets in different ADUs. For example, Figure 4.4 demonstrates a sequence of packets of the smart light and their frame size. It is clear that packets of the same pair of ADUs are close to each other in terms of their time stamps, and the packets of different pairs of ADUs are distanced by a few seconds to each other. Here each pair of ADUs is an `ON` or `OFF` operation of the smart light. Figure 4.4 shows that the user turned on or off the smart light every few seconds in the experiment setting. Based on this insight, we can perform a clustering (e.g., X-means) to segment sequence of packets into subsequences of packets based on their timestamps. For protocols without request and response ADU pairs, each sequence of packets is an ADU. For protocols with request and response ADU pairs, we can use the packet direction to further segment the subsequence into two subsequences for request and response ADUs.

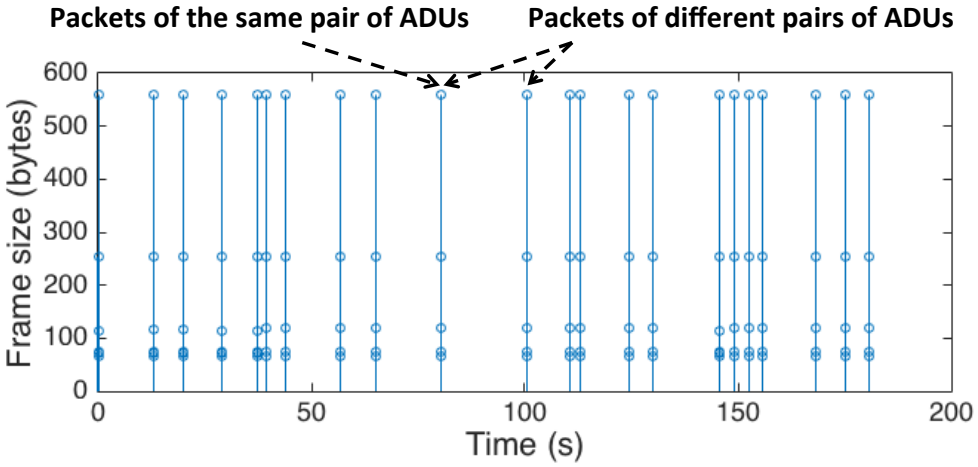

Figure 4.4: A sequence of packets of the smart light and their frame size.

Once we have segmented the sequence of packets into subsequences, the second step is to

[4] We assume single threaded communication which is common for IoT devices.

identify the type of ADU for each subsequence. To achieve this, our idea is to perform another clustering on all subsequences to cluster them into different types of ADUs based on key features (e.g., frame length) of the packets shown by previous side-channeling works [41, 50]. The key issue here is how to calculate the similarity of different subsequences as the number of packets can be different in different subsequences. In this paper, we use a standard signal processing technique called Dynamic Time Warping to calculate the similarity between different subsequences.

After the above two steps, we are able to convert a sequence of packets with L4 information into a segmented and typed sequence of ADUs for RADIO to build the L7 behavior models. Note that we use clustering so no label is required.

4.2 Learning IoT Behavioral Model

Given the unlabeled and potentially polluted *historical traces* from a set of aggregated IoT devices, RADIO needs to learn a FSM model to represent benign behaviors. We already know that a historical trace can be considered as a sequence of messages (ADUs). To learn a FSM model from a set of such sequences (set of historical traces), there are two issues.

The first issue is how to infer the FSM states. It is well-known that classic FSM inference algorithms requires a positive set of sequences (sequences that would appear in the FSM) and a negative set of sequences (sequences that would never appear in the FSM) to determine the states of the FSM and build a non-trivial FSM (proved by Gold et al. [67]). For example, if there is only a set of positive sequence $\{a, b\}$ and $\{a, c\}$, then the classic FSM inference algorithms will build a FSM with one state and three self-transitions a, b and c. Thus any sequence composed by a, b or c is included in the FSM, even if $\{b, c\}$ is a malicious sequence. Previous FSM-based system-call anomaly detection approaches [103], however, avoid this issue by assuming that the states can be determined by program structure (e.g., while loops). However, in the context of IoT historical traces, there is only one unlabeled set of sequences, and there is no program structure information that can determine the states to build a non-trivial FSM.

The second issue is pollution. Even if we can build a non-trivial FSM from only one set of sequences, this does not address the pollution issue. If the one set of sequences for the algorithm is polluted, then the FSM model will contain malicious behaviors, thus the FNs will be high.

4.2.1 RADIO's idea

Now we present our ideas to address these two issues. We start by considering a classic FSM inference algorithm called RPNI[5] (Regular Positive and Negative Inference) [93]. However, the FSM inference algorithms (including RPNI) are *supervised* algorithms, requiring both labeled benign samples and labeled attack samples (Gold et al. [67]), which are hard to obtain in practical enterprise settings. To address this issue, our first step is to modify the *supervised* RPNI

[5]The state-of-the art FSM inference algorithms include RPNI, Biermann & Feldman's algorithm[44], and DeLeTe2[57]. Biermann & Feldman's algorithm and DeLeTe2 generate non-deterministic FSM, while RPNI generates deterministic FSM. Nondeterministic means it can transition to, and be in, multiple states at once, which is contradicting the deterministic behaviors of enterprise IoT devices.

algorithm into a *semi-supervised* algorithm that can work with benign samples alone. For this, we leverage the *causality* in the IoT traffic (e.g., a login request resulting in a login status response) to determine the FSM states.

However, modifying the RPNI algorithm alone does not address the pollution issue. If the benign samples for the *semi-supervised* algorithm is polluted, then the FSM model will contain malicious behaviors, thus the FNs will be high. To address this issue, our idea is to design an *unsupervised* algorithm by novelly integrating the *semi-supervised* RPNI algorithm with an outlier detection algorithm called RANSAC[66][6]. The *unsupervised* algorithm will generate multiple candidate FSM models using the *semi-supervised* RPNI algorithm, and leverage RANSAC algorithm to filter out models with malicious behaviors (outliers), and select the best benign model from the candidate models.

4.2.2 RPNI Modification

Basic RPNI The basic *supervised* RPNI algorithm is shown in Figure 4.5. In Step 1, benign or "positive" samples are used to build a tree where each sample is a path from root to leaf. In Step 2, basic RPNI merges nodes to compact the tree into FSM model, and ensure that the labeled attack or "negative" samples is not included in the FSM model. For example, in the bottom left figure of Figure 4.5, in Step 2, if S_0 and S_2 are merged, then the counterexample $GET, 200$ (indicating an authentication bypass) will be included. When there are no attack samples, all the nodes will be merged, and result in a single-state FSM with all malicious behaviors included, and the FNs will be high.

RADIO's modification for RPNI We modify the RPNI algorithm to a *semi-supervised* algorithm without requiring attack samples. We achieve this by replacing the traditional merge with two causality heuristics: (1) at any state, same request ADU yields same response ADU, and (2) at any state, same ADU fields subset m_a yields same message fields subset m_b. Our observation is that the protocols used by the enterprise IoTs either have strong correlations between their request and response ADUs (e.g. the request method and the status code of HTTP) or have strong correlations between the commands and arguments inside an ADU (e.g. the command and subcommand of Telnet). For any protocol we have considered in this paper, at least one heuristic can be applied. Therefore, we believe that the two heuristics are general for all network protocols. More specifically, for any protocol, consider a network session composed by a sequence of ADUs. If the protocol is composed of request and response pairs (e.g., HTTP, SMB and SNMP), the occurrence of a response highly depends on the previous request. If the protocol is not composed of request and response pairs (e.g. Telnet, DNS and MQTT), then the occurrence of some ADU fields highly depends on some other ADU fields. For example, for Telnet, the ADU field `subcommand` highly depends the ADU field `command`.

To extend RPNI, we first need to adapt basic RPNI to Mealy Machine, so we add a Step 0 to pair abstract message and build a Mealy Machine tree in Figure 4.5. Then we modify Step 2 to use the causality heuristics, e.g., in the bottom right figure in Figure 4.5, in Step 2, our causality

[6]In particular, two popular state-of-the-art outlier detection algorithms are RANSAC[66] and Hough Transformation[61]. The latter is designed for image structures (e.g., pixels) and is not applicable for IoT models.

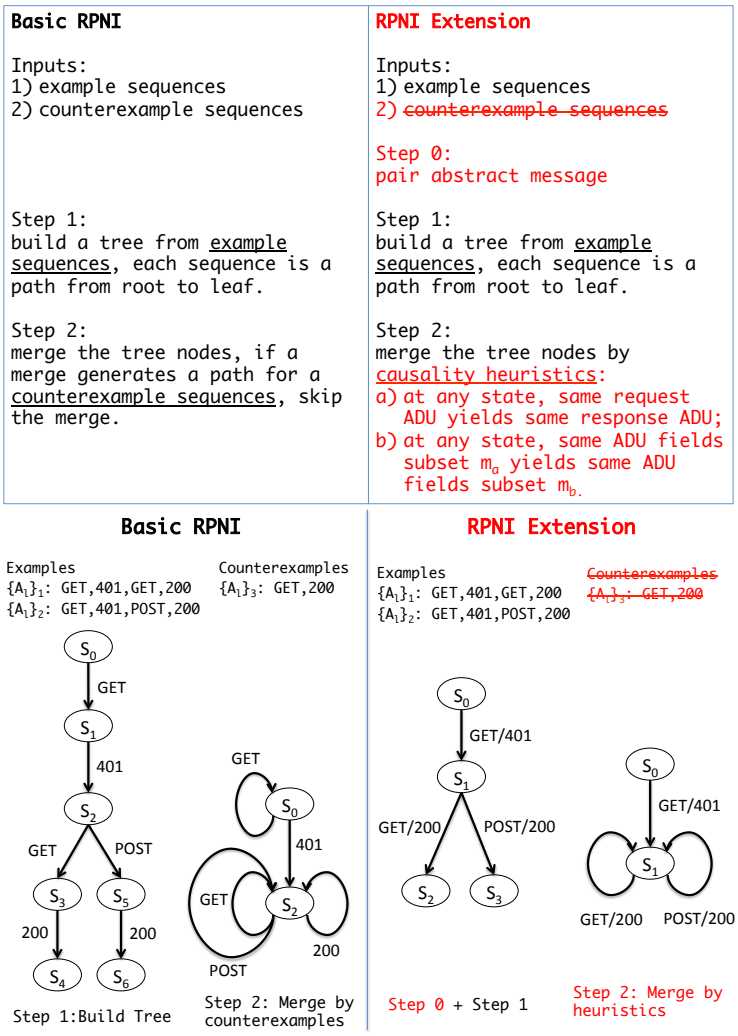

Figure 4.5: Basic RPNI algorithm and our extension.

heuristics will prevent S_0 and S_1 from being merged, otherwise a GET input in S_0 will result in two outputs - 200 and 401, which breaks the deterministic nature of Mealy machine.

4.2.3 RANSAC Modification

Basic RANSAC

The basic idea in RANSAC (Random Sampling and Consensus) is to randomly sample data from a polluted training set, build a model using the sampled data, and then evaluate how well this model fits the inliers (benign behaviors) in the dataset. RANSAC will repeat this process for a number of iterations to generate a set of candidate models, then pick the model that best fits the inliers (benign behaviors).

More specifically, as shown in Figure 4.6, for a set of samples, the algorithm first selects a random subset of size r (e.g., 10% of the data set size) and builds a model from the subset (Line

1 Input: D - a set of data points
2 for $iterations \leftarrow 1$ to k
3 $D_{iter} \leftarrow SampleRand(D, r)$
4 ▷ build a base model from r samples
5 $Model_{iter} \leftarrow Learn(D_{iter})$
6 $D_{add} \leftarrow \{\}$
7 $D_{test} \leftarrow D - D_{iter}$
8 for each $d \in D_{test}$
9 ▷ check if the rest of the data fits the model.
10 if $DataFit(d, Model_{iter}) < dt$
11 $D_{add} \leftarrow D_{add} \bigcup d$
12 ▷ check if a good model is found
13 if $D_{add} > n$
14 ▷ Update the model with all "inliers"
15 $Model_{iter} \leftarrow Learn(D_{iter} \bigcup D_{add})$
16 ▷ evaluate the $loss$ on "inliers"
17 $loss = ModelLoss(Model_{iter}, D_{add})$
18 return the $Model_{iter}$ with the smallest $loss$ in k iterations

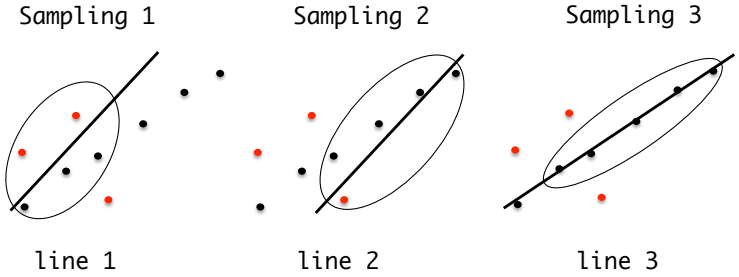

Figure 4.6: Basic RANSAC and three iterations generating different models (lines 1 - 3 in the images below).

numbers 3 - 5). For example, in Figure 4.6, it builds a line from a subset of points. The algorithm tests all other items against the model based on a $DataFit$ function and if the mismatch is less than a threshold dt, then the item is said to fit the model well, and it adds the item to the subset (Lines 8 - 11). For example, in Figure 4.6, the $DataFit$ function can be defined as the Euclidean distance from a point to the line. Eventually, if the subset size is greater than a size n, the model is considered to be reasonably good. It then evaluates the model with a $ModelLoss$ function and computes the loss (Lines 13 - 17). For example, in Figure 4.6, the $ModelLoss$ function can be defined as the sum of Euclidean distance from all the points to the line. It repeats this process for k iterations and selects the model with the minimum loss as the best model. In this process, models with malicious behaviors (outliers) will yield relatively greater model loss and will be filtered out.

RADIO extension for RANSAC

However, RANSAC is not directly applicable for the FSM model generated by the *semi-supervised* RPNI algorithm. There are two modules in RANSAC that are hard to define for the FSM model of IoT behaviors. First, we need to define a $DataFit$ function to find whether an ADU series fits

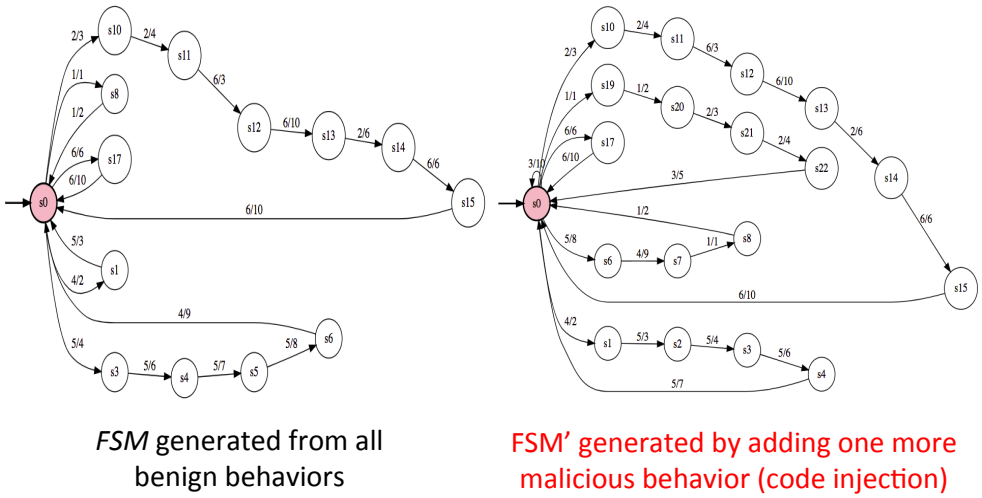

Figure 4.7: Comparison of benign/malicious model.

an FSM model. More importantly, we need a $ModelLoss$ function to evaluate how well a model can represent all benign behaviors.

First, we show how we define the $DataFit$ function by taking advantage of the structural differences of the FSM models. The input of a $DataFit$ function is a ADU series A and a model ψFSM, and the output is whether this ADU series A fits the model ψFSM. First, we convert each ADU series A to a sequence of transitions $\delta_1, ..., \delta_A$. A strawman solution to define the $DataFit$ function is to match $\delta_1, ..., \delta_A$ with ψFSM starting from the initial state s_0, and count the number of unmatched transitions. However, the problem with this definition is that the transitions generated by some attack behaviors A' may deviate only slightly from the benign model ψFSM, resulting in a very small mismatch. To address this issue, our idea is to amplify the mismatch between ψFSM and A' by computing the structural difference between the original model ψFSM and a model $\psi FSM'$ including attack behaviors A'. Here ψFSM is learned from the original subset of benign ADU series set, say $\{A_l\}_{origin}$, and $\psi FSM'$ is learned from $\{A_l\}_{origin} \cup A'$. The structural difference is defined as the Levenshtein distance between two FSM models. If one transition δ' is different between A' and ψFSM, this transition will impact multiple transitions for $\psi FSM'$ in RPNI's merge process, magnifying the difference between ψFSM and $\psi FSM'$. For example, Figure 4.7 shows the ψFSM generated from all benign behaviors, and the $\psi FSM'$ generated by adding one more malicious behavior (code injection) to the subset. We can observe that the small difference (2 transitions) in Figure 4.2 is amplified to a huge structural difference between ψFSM and $\psi FSM'$ in Figure 4.7 (8 transitions). Based on this idea, we define the data fit function $DataFit$ as follows:

$$DataFit(\psi FSM, A) = lev(\psi FSM, \psi FSM'), \tag{4.1}$$

where $\psi FSM' = RPNIExtension(\psi FSM, A')$ and lev is the Levenshtein distance between two FSMs, defined as:

```
Input: {A_l} - a set of ADU series
for iterations ← 1 to k
    {A_l}_iter ← SampleRand({A_l}, r)
    ▷ build a base model from r samples
    ψFSM_iter ← RPNIExtension({A_l}_iter)
    {A_l}_add ← {}
    {A_l}_test ← {A_l} − {A_l}_iter
    for each A ∈ {A_l}_test
        ▷ check if the rest of the ADU fits the FSM.
        if DataFit(A, ψFSM_iter) < dt
            {A_l}_add ← {A_l}_add ⋃ A
    ▷ check if a good model is found
    if {A_l}_add > n
        ▷ Update the model with all "inliers"
        ψFSM_iter ← RPNIExtension({A_l}_iter ⋃ {A_l}_add)
        ▷ evaluate the loss on "inliers"
        loss = ModelLoss(ψFSM_iter, {A_l}_add)
return the ψFSM_iter with the smallest loss in k iterations
```

Figure 4.8: RADIO's learning mechanism.

$$lev(\psi FSM_1, \psi FSM_2) = (|\{\lambda_1\}| + |\{\lambda_2\}| - 2 * |\{\lambda_1\} \cap \{\lambda_2\}|) \\ +(|\{\delta_1\}| + |\{\delta_2\}| - 2 * |\{\delta_1\} \cap \{\delta_2\}|), \tag{4.2}$$

where $\{\lambda_1\}$ and $\{\lambda_2\}$ are the set of states for ψFSM_1 and ψFSM_2, and $\{\delta_1\}$ and $\{\delta_2\}$ are the set of transitions for ψFSM_1 and ψFSM_2.

Next, we show how to define the $ModelLoss$ function. The $ModelLoss$ function is used to evaluate how likely an FSM model is a good benign behavior model. We posit that an attacker will need to use extra states and/or state transitions beyond what the benign model has for the limited set of IoT behaviors to launch an attack. Based on this idea, we define the $ModelLoss$ function as follows:

$$ModelLoss(\psi FSM, \{A_l\}) = \omega_\lambda * |\{\lambda\}| + \omega_\delta * |\{\delta\}|, \tag{4.3}$$

where $\omega_\lambda * |\{\lambda\}|$ represents the loss caused by using extra transitions and $\omega_\delta * |\{\delta\}|$ represents the loss caused by using extra states. Together they capture how much the attacker's behavior deviates from the limited enterprise IoT behaviors.

4.2.4 Putting it together

Now we put the RPNI extension and RANSAC extension together. The pseudo code is given in Figure 4.8. Comparing with basic RANSAC, the input is changed to a set of ADU series. We replaced the model learning with our RPNI extension and applied RADIO's $DataFit$ and $ModelLoss$ function.

4.3 Results

In this section, we evaluate RADIO and show that

- RADIO achieves low FPs and FNs (FPR $< 1\%$ and FNR $< 0.01\%$) comparing with prior anomaly detections [35, 39, 53, 88, 95] (FPR $> 5\%$ and FNR $> 57\%$).
- RADIO is robust against pollution (FPR $< 1\%$ and FNR $< 0.01\%$ when the percentage of polluted traffic ranges from 1% to 15%),
- RADIO's behavior model can be aggregated across device instances for behavior coverage and scalability,
- RADIO's performance is robust across a range of configuration settings (FPR $< 1\%$ and FNR $< 0.01\%$) and does not require fine-grained tuning.

4.3.1 Setup

We evaluate RADIO by considering the common enterprise IoT devices, i.e., routers, switches, NAS, cameras, DVRs, smart lights, smart plugs, fire alarms, sensor gateways and smart speakers (e.g., Amazon Alexa). We setup a sandbox IoT environment in an organization with the above 9 classes of IoT devices deployed.

One challenge for evaluating RADIO is that there are few benign or attack traces available for IoTs in enterprise setting. To address this challenge, we conducted an IRB-approved study to build a network traffic dataset for 9 types of IoT devices (Figure 14). The dataset contains 371.6 MB of pcaps with >700K packets, including >10,000 benign traces and >1000 attack traces. Each trace contains all the packets of a benign or malicious session. To collect benign traces, we invite 10 human users to operate on the IoT devices and record the network traffic as benign traces. We provide the official user interfaces of the IoT devices (webpage, terminal or mobile apps) to the users. The users are free to use any operation provided by the user interfaces. For example, D-Link camera is controlled by a webpage with audio on/off button, streaming on/off button and zoom-in X2, X4, X8 buttons. The users are free to click these buttons. To collect attack traces, we build an IoT attack bundle, including several recent IoT exploits, such as the Mirai malware[40] (against Camera), DNS rebinding[38] (against smart speakers, smart lights, routers and switches), Alexa eaves-dropping attack[36] (against smart speakers), SambaCry[33] (against NAS) and CMP/SNMP exploits[9] (CIA Vault 7 Exploits against routers and switches). Each attack trace includes a unique sequence of real-world exploits (e.g., different code injections) and after-exploit operations (e.g., exfiltrate video/voice or modify credentials) that are commonly seen in enterprises. In total, we used 24 exploits (2-3 exploits per device type) and 92 after-exploit operations (8-10 per device type) to build the attack traces. Then to build an unlabeled trace set for evaluation, we randomly mix the attack traces and benign user traces by percentage (e.g., mix 100 attack traces and 900 benign traces for a trace set with 10% of polluted traces.).

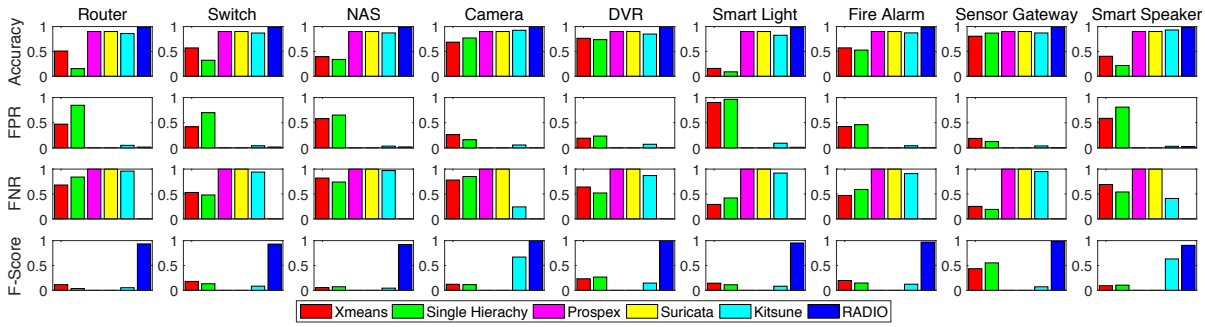

Figure 4.9: We measure the FP and FN of five different approaches over routers, switches, NASes, cameras, DVRs, smart lights, fire alarms, sensor gateways and smart speakers. 10% of the traces for each device classes are attack samples.

4.3.2 Accuracy

Figure 4.9 compares RADIO with several candidate solutions on False Positive Rate (FPR), False Negative Rate (FNR), accuracy, and F-score. The Accuracy is defined as $Accuracy = \frac{TP+TN}{Pos+Neg}$. The FPR describes the occurrence of False Positives, defined as $FPR = \frac{FP}{FP+TN}$. The FNR describes the occurrence of False Negatives, defined as $FNR = \frac{FN}{FN+TP}$. The F-Score reflects the detection performance combing FP and FN, defined as $FScore = \frac{2*Precision*Recall}{Precision+Recall}$, where $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$. These measurements are taken over the unlabeled traces sets for 9 classes of enterprise IoT devices. Each unlabeled traces set contains 1000 traces and 10% of them are attack traces.

We compare RADIO with other anomaly detection approaches: two clustering-based anomaly detection (Ngram + Xmeans by Antonakakis [39] and Editing Distance + Single Hierarchy by Perdisci [95]), two FSM-based approaches (Prospex[53] and the commercial Suricata IDS[35]), and a flow-statistics-based approach (Kitsune[88]). To apply the approach by Antonakakis [39], we extract Ngrams from the ADU series of each session, and use Xmeans to cluster the sessions. To apply the approach by Perdisci [95], we calculate the Editing Distance among the ADU series of each session, and use the Editing Distance vector of each session to cluster the sessions. To apply the approach by Prospex[53], we implemented the FSM inference mechanism in Prospex[53] to infer the FSM model. We evaluate the performance of other approaches by selecting the best result yield by different parameter combinations[7] while using a fixed set of parameters for RADIO (sampling size r = 4% of all samples, model assertion threshold n = 20% of all samples and iteration k = 10000 times).

The first row shows that RADIO achieves the highest accuracy. The second and third row shows that RADIO achieves both low False Positive Rate and low False Negative Rate (average FPR $< 1\%$ and average FNR $< 0.01\%$). The clustering approaches often have a high False Positive Rate (average FPR $> 44\%$ and average FNR $> 57\%$) because benign traces are often

[7]For Xmeans, we adjust the number of seeds from 2 to 20 and maximum number of clusters from 2 to 20. For Single Hierarchy Clustering, we adjust the number of clusters from 2 to 20 and try three different distance functions Euclidean Distance, Manhattan Distance and Chebyshev Distance. For FSM-based approaches, there are no parameters to adjust.

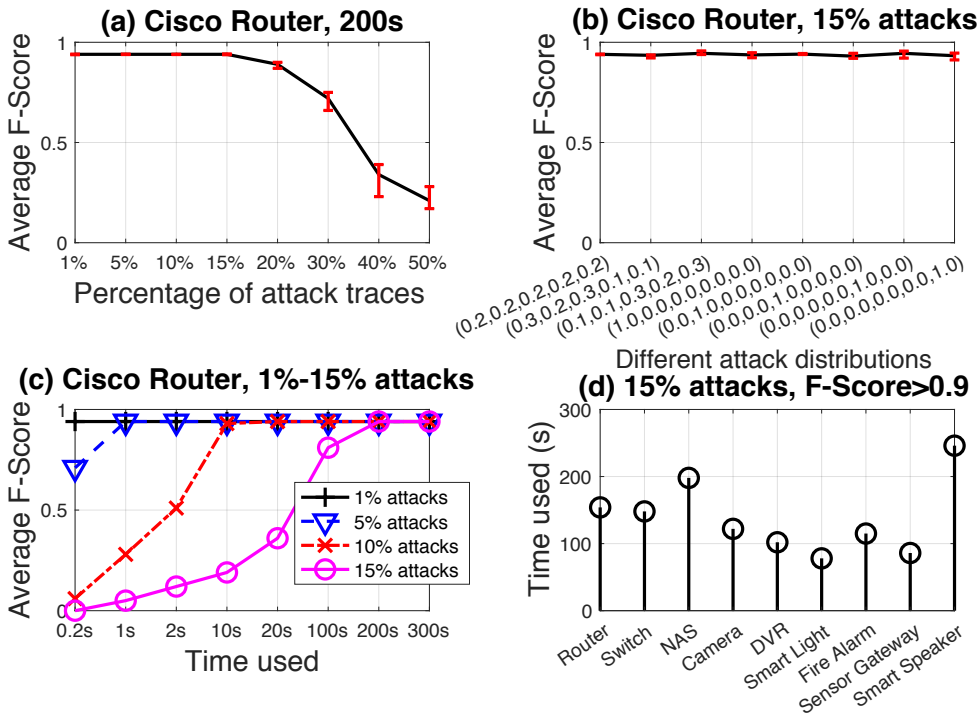

Figure 4.10: RADIO is able to generate a precise FSM representation (F-Score>0.9) in an acceptable amount of time ($< 248s$) for polluted sample set with attacks ranging from 1% to 15% (left figure), and for different devices (right figure) when the sample sets are heavily polluted (15%).

grouped into different clusters and only one cluster is considered as normal by their mechanism. The FSM-based approaches often have a high False Negative Rate (average FNR $> 99\%$), because their mechanism cannot filter out the polluted data in the historical traces, so the behavior models they generate contain malicious behaviors. The flow-statistics-based approach Kitsune[88] (FPR $> 5\%$ and FNR $> 79\%$) is only effective for statistical anomalies (Mirai C&C for camera and DNS rebinding for smart speakers), but not effective for more stealthy attacks including code injection and authentication bypass. The last row shows that RADIO's detection performance combining FP and FN outperforms the comparison approaches and achieves a F-Score close to 1. In summary, even with a fixed (default) parameter, RADIO outperforms the best results of other approaches under optimal parameters by 4X.

4.3.3 Robustness against pollution

Next, we show that RADIO is robust to unlabeled, potentially polluted sample sets from diverse enterprise IoTs. There are four factors that may impact the detection accuracy of RADIO: the fraction of attack samples; the diversity of attack types in the attack samples; the time (number of iterations) used by RADIO to build the model; and the diverse classes of IoT devices. We evaluate the robustness of RADIO against the impact of these four factors (by default, we set the fraction of attack samples to 15%; the attack types to be evenly distributed; the time used to

$200s$; and the device class to be Cisco routers).

First, we change the fraction of attack samples in the sample sets and measure the accuracy of RADIO, as shown in Figure 4.10(a). We generate 8 polluted trace sets (each with 1000 traces) for Cisco router with the fraction of attack traces ranging from 1% to 50%[8]. We measure the average F-Score (with min-max error) in 100 runs. We can see that, when the fraction of attack traces is less than 15%, RADIO is accurate (F-Score>0.9). Note that if the fraction of attack traces is higher than 15%, then the attack is "noisy" and should be easy for the administrators to identify (e.g., by flow statistics).

Next, we evaluate the impact of the distribution of different types of attacks in the sample sets, as shown in Figure 4.10(b). We use 5 different types of attacks (including four code injections and the DNS rebinding attack) to build the attack traces set (of 150 attack traces) for Cisco router. We change the distribution of the 5 types of attacks and measure RADIO's average F-Scores (with min-max error bar) in 100 runs. For example, in Figure 4.10(b), the distribution "(0.2, 0.2, 0.2, 0.2, 0.2)" in the X axis means that each type of attack occupies 20% of the attack traces. We can see that the variation of the F-Score is small under different distributions. This is because RADIO magnifies the structural changes to the FSM behavior model caused by attacks, thus is robust against the distribution changes of different types of attacks.

Then we change the time (number of iterations) used by RADIO to build the model to see if RADIO can generate a precise FSM model (F-Score close to 1) in a reasonable amount of time. In Figure 4.10(c), we change the iteration number k in the RANSAC component to control the time used to generate the FSM model and measure the F-Score for traces sets with fraction of attack traces ranging from 1% to 15% (for Cisco routers). We can see that, when the fraction of attack traces changes from 1% to 15%, RADIO can always generate a precise FSM model within 200 seconds (F-Score is 0.935).

We also show that RADIO is able to generate a precise FSM model for different classes of IoT devices. Figure 4.10(d) shows the time used by RADIO to generate a precise FSM model (F-Score>0.9) for 9 different classes of enterprise IoTs. We can see that, even when 15% of the traces are polluted, RADIO can generate a precise FSM model for different IoT devices within 248 seconds (the smart speaker case). Since building the model is one-time effort in the offline stage, the time cost of $248s$ is acceptable.

4.3.4 Performance analysis

In this part, we analyze the performance of RADIO in the offline stage and online stage. In offline stage, the main performance index is the training time (i.e., the time used to process the historical traces and learn the benign model for an IoT device). We ran the offline learning process for 9 classes of IoT devices on a MacBook Pro machine, with 2.2 GHz Intel Core i7 and 16GB of RAM. As shown in Figure 4.10 (d), the training time for every class of IoT devices is less than 300s, which is acceptable because the learning stage only needs to be performed once.

In the online stage, the main performance indexes are the network throughput and latency when RADIO's detection is in place. We measured the average network throughput and latency when RADIO's detection is deployed (i.e., customized Bro scripts that implements RADIO's

[8]By theory, the RANSAC module will not work when the fraction surpasses 50%[66].

Device	Correctly Clustered	Not Correctly Clustered
smart light	PUT {"on" : $true$} PUT {"on" : $false$} 200 OK	
camera	GET $\backslash user$ POST $\backslash recording$ POST $\backslash setSystemStream$ GET $\backslash view\backslash zoom\backslash X2$ POST $\backslash setDebugLevel$ 401 200 OK	GET $\backslash view\backslash zoom\backslash X4$ GET $\backslash view\backslash zoom\backslash X8$

Table 4.1: The ADUs that are correctly clustered and not correctly clustered.

FSM model). The average network throughput is 6.21Gbps and latency is 4.3 ms. The above throughput is sufficient for IoT devices as the throughput needed by an IoT device is low. Calculated from our dataset, the average throughput needed by an IoT device is less than 0.1 Mbps. Also, the latency has trivial impact on the performance of IoT devices. Such latency is not surprising since IDSes such as Bro are tapped to a switch and are not in the critical path of the network packet (use mirrored packet rather than the original packet).

4.3.5 Handling encrypted traffic

In this part, we demonstrate the possibility for RADIO to handle encrypted traffic using the side channeling and clustering technique. To evaluate, we ask human user to perform a known sequence of actions (e.g., turn on the smart light) on smart light and camera via https and record the sequence of packets for these actions. Then we apply the side channeling and clustering technique to segment the sequence of packets and generate a sequence of typed ADUs. We convert the known sequence of actions into the ground truth of sequence of ADUs and compare the sequence of typed ADUs with the ground truth from known actions.

We show the correctly and incorrectly distinguished ADUs in Table 4.1. The side channeling and clustering technique successfully clustered most of the ADUs for smart light and camera. The only exception is that GET $\backslash view\backslash zoom\backslash X2$, GET $\backslash view\backslash zoom\backslash X4$ and GET $\backslash view\backslash zoom\backslash X8$ are clustered into one cluster rather then three clusters, as the key features (e.g., frame length) are quite close. However, this cluster is still different from the cluster of the malicious ADU (i.e., POST $\backslash setDebugLevel$), thus will not impact the detection accuracy of RADIO. This experiment demonstrates that it is possible to extend RADIO to handle encrypted IoT traffic, based on our side channeling and clustering technique.

4.3.6 Aggregated behavior model for coverage and scalability

In this part, we show that, by building aggregated behavior model from only a few device instances, RADIO can provide good behavior coverage and scalability. We focus on aggregated behavior model across IoT device instances of the same device type (e.g., all Cisco 3640 routers), or of the same class & vendor (e.g., all Cisco routers) in an enterprise. We leave the aggregated model for IoT device instances of different vendors (e.g. all Cisco and Juniper routers) or deployed in different enterprises for future work.

In the first experiment, we evaluate how many device instances are needed to generate a

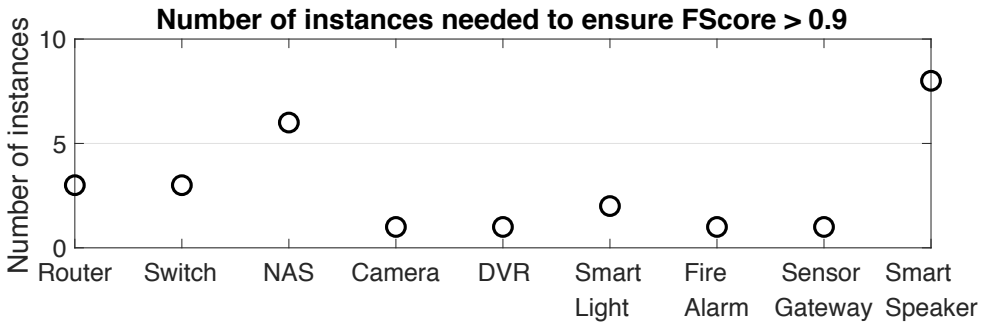

Figure 4.11: Number of instances needed for aggregated models.

good model (F-Score $>$ 0.9) to cover the benign behaviors for one type of devices (as shown in Figure 4.11). We can see that all the device type need at most 8 instances to provide a good coverage (F-Score$>$0.9). And while Camera, DVR, Smart light, Fire Alarm and Sensor Gateway only need one instance to ensure F-Score is greater than 0.9, routers, switches, NASes and smart speakers need more instances. This is because the behavior options for the routers, switches, NASes and smart speakers are more diverse than the cameras, DVR, smart lights, fire alarms and sensor gateways. For example, the NASes provides 36 operations while the smart lights only provide two operations (on and off).

Next, we explore the aggregated behavior model across device instances of the same class & vendor (e.g., Cisco 3640 routers and Cisco 7200 routers). In Figure 4.12, we generate the behavior model from a few router instances (5 instances) of one type (Cisco 3640) in the offline stage, then apply the model to a few other instances (5 instances) of a different type (e.g. Cisco 7200) for detection. Then we measure the F-Score to see if the model is suitable for sharing. Figure 4.12 shows the heatmap of the F-Scores We can see that the F-Score varies little across the instances of the same class & vendor (Cisco routers), which means it is possible to share the behavior model across device instances of the same class & vendor.

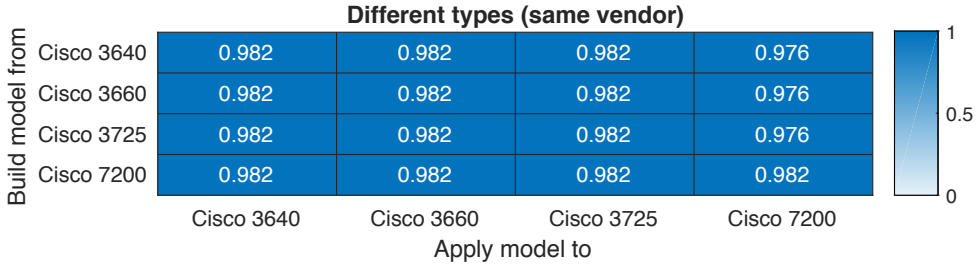

Figure 4.12: F-Scores for aggregated behavior model.

4.3.7 Sensitivity to parameters

Next, we show that RADIO does not require parameter tuning to maximize its detection rate by measuring the average F-Score of RADIO in 100 runs under different parameters. There are three parameters for RADIO: the threshold of the $DataFit$ function that judge if a sample fits a

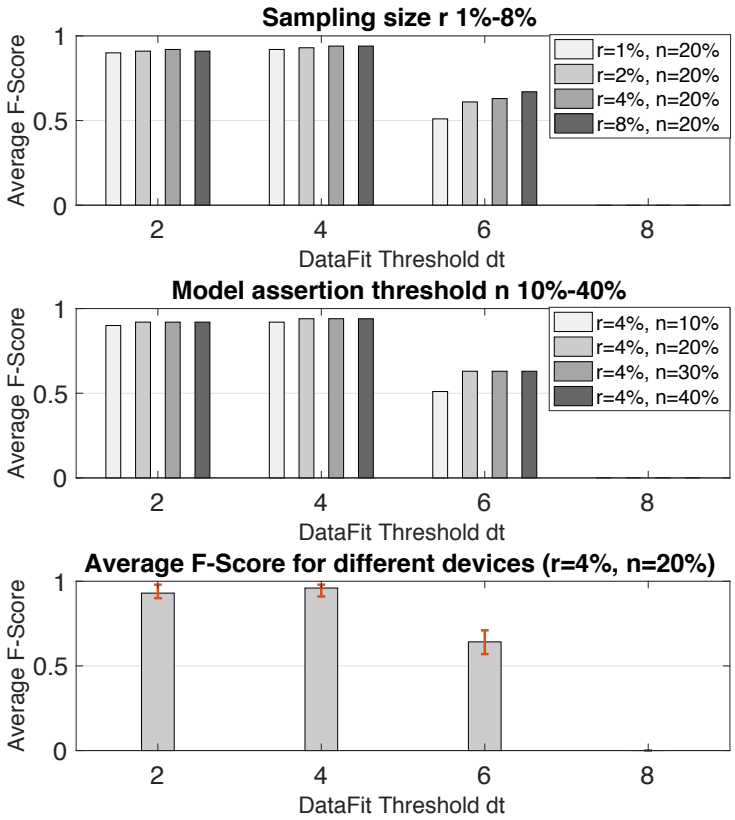

Figure 4.13: The average F-Score for different combination of parameters. The top and middle figure are measured on Cisco router and the bottom figure is measured on different devices.

FSM model; the sampling size r; and the assertion size n that describes the number of samples to assert a FSM is good. Here we set the number of iterations k to a fixed value (10000 times) as RANSAC does not have a constraint on the number of iterations. Figure 4.13 shows the results for different combination of parameters. The X axis is the threshold of the $DataFit$ function and the Y axis is the average F-Score (in 100 runs). In the top figure, we change the sampling size r from 1% to 8% (of all samples) and observed that F-Score does not change much across different sampling size, even though r of 4% yields slightly higher F-Score. In the middle figure, the F-Score stays the same when the assertion size n is greater than 10%. Then the bottom figure shows the F-Score when r is 4% and n is 10%, and together with top and middle figures, it shows that the detection performance is good when threshold of the $DataFit$ function is less than or equal to 4. This is because when the threshold of the $DataFit$ function is too large, attacks will also be included in the FSM representation and the FN will be high. The above measurements suggest that RADIO is not sensitive to the configuration of parameters.

4.3.8 Diagnostic Utility

Finally, we show that the FSM behavior model of RADIO is useful for enterprise security administrators to diagnose the attacks against enterprise IoT devices (e.g. which command/argument

Abstracted CMP Messages

ID	Subset $\{m_a\}$ Direction+Command	Subset $\{m_b\}$ Subcommand
1	from + Will	Echo
2	from + Do	Suppress Go Ahead
3	to + Suboption	Terminal Type
4	to + Do	Negotiate About
5	to + Will	Environment Option
6	from + Suboption	Remote FlowControl
7		Linemode
8		New Environment Option
9		Status
10		End

Remote code injection (CVE 2017-3881)

1/1
1/2
2/3
2/4
3/5
3/10

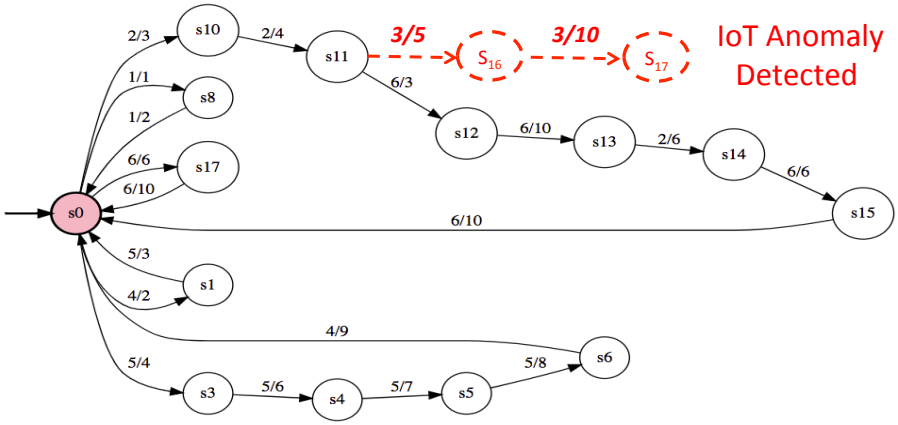

Figure 4.14: How RADIO's benign behavior model can help to identify a code injection attack (CVE 2017-3881).

is flawed and used by attacker). As shown in Figure 4.14, RADIO can automatically identify the mismatches between the benign behavior model and the ADU series of the attack, and will raise an alert containing the mismatched states and transitions. Using the alerts we obtained from the detection of RADIO, we were able to obtain the information about several flaws the IoT attack exploits, which are not available from NVD[29]. This shows that the FSM behavior model of RADIO can help the admin quickly identify the flaws of the IoT devices and better respond to the security issues of the enterprise IoT devices.

Chapter 5

Learning Complex IoT Interactions

In Chapter 3, we presented LoFTI's vision to learn context-based policies for IoT devices. We have also seen that the key issues to make LoFTI practical are:

- How to define key features to extract contextual IoT access patterns?
- How to build model with high accuracy and low computation/communication cost?
- How to address unbalanced contextual IoT access records?

In this chapter, we will answer these questions with detailed design of LoFTI's learning mechanism. After that, we will present the result of evaluating LoFTI for its accuracy and communication/computation cost.

5.1 Defining key features for contextual IoT access

LoFTI framework extracts a set of key features from the historical record dataset to capture the contextual IoT access patterns. Given a wide range of information we can collect from IoT devices, it is challenging to identify the salient features.

To address this challenge, we leverage the concept of *contextual integrity* (*CI*)[92], which claims that appropriate information flows (i.e. IoT accesses, in our case) are the ones that conform with the *contextual norms*. Here *contextual norms* refer to five independent parameters: data subject, sender, recipient, information type, and transmission principle. Now the issue is how to define the *contextual norms* that should regulate IoT accesses.

Previous efforts[43, 48, 60, 65, 78, 102, 116] have manually crafted different sets of contextual norms and have shown that they are closely related with IoT accesses; however, their definitions have not been very consistent. In this paper, we summarize and generalize these *contextual norms* into six types:

User: The *user* context norm is defined as the role of the person who is accessing the IoT device. Roles can be *parent*, *child*, *household employee* (e.g. a babysitter), or *visitor*. The *user* context norm corresponds to the *sender* in the *CI*, capturing who initiated an IoT access.

Device: The *device* context norm is a combination of device type and device instance identifier. Device type is a combination of device class (e.g., camera), device vendor (e.g., D-Link)

and device SKU (Stock Keeping Unit, e.g., DCS-932L). The *device* context norm corresponds to the *recipient* in *CI*, capturing which device is being accessed.

Time of the day: The *time of the day* context norm is a combination of hour of the day, minute of the hour and second of the minute. This context norm corresponds to the *transmission principle* in *CI*, capturing the time condition under which the IoT access should be allowed.

Environment variables: The *environment variables* context norm is a list of variables describing the physical environment of a smart home, such as temperature, humidity, CO_2 density, etc. This context norm corresponds to the *transmission principle* in *CI*, capturing the environmental conditions under which the IoT access should be allowed.

Device states: The *device states* context norm is a list of variables describing the states of the IoT devices. Examples of such states include *smart window opened*, *camera turned on*, and *smart oven turned off*. This context norm also corresponds to the *transmission principle* in *CI*, capturing the conditions of device states under which the IoT access should be allowed.

Action: The *action* context norm is an operation to be performed by the IoT device by the access, such as *open* the smart window and *turn on* the camera. This context norm corresponds to the *data subject* in the *CI*, capturing the subject of the IoT access.

Let's denote the *user* as u, the *device* as d, the *time* as t, the *environment variables* as $\{e\}$, the *devices' states* as $\{s\}$ and the *action* as a. Then the context norms for LoFTI can be expressed as $(u, d, t, \{e\}, \{s\}, a)$.

Next, we demonstrate the expressiveness of LoFTI's contextual norms by showing a list of contextual IoT access patterns LoFTI can capture, as shown in Table 5.1. In the first scenario, a visitor watches a video recorded on the camera (privacy leakage). In this case, the three key features – the user's role as a *visitor*, the device *camera*, and the action of *playing the recorded video* – are captured. In the second scenario, a family member takes shower at night, and the key feature *time* is captured. Similarly, in the third scenario – opening a window when it is hot – the key features temperature and humidity are captured. The fourth scenario covers a cooking activity and the key features – the states of microwave oven and fridge – are captured.

Pattern	u	d	t	$\{e\}$	$\{s\}$	a
Visitor watch camera video record	visitor	camera	...	...	...	play_video_record
Take shower at night	family	shower	21:35:24	...	...	turn_on_shower
Open window when feeling hot	family	window	...	temperature=$35°C$	...	open_window
Use stove when cooking	family	stove	...	...	microwave_on	turn_on_stove

Table 5.1: Exemplar contextual IoT access patterns that can be captured by LoFTI's contextual norms.

5.2 Building a model with high accuracy and low computation/communication cost

Considering the distributed learning nature of federated multi-task learning, building a machine learning model for contextual IoT access raises challenging trade-offs between *model accuracy* and *computation/communication cost*.

To understand these trade-offs, we start by exploring the design space for building a machine learning model. At a high level, the machine learning model can be considered as a function F

that takes the contextual norms $(u, d, t, \{e\}, \{s\}, a)$ as input x and outputs a binary decision, i.e., whether to block or allow the action a given the context norms.

There are several options to build such a function F, ranging from a simple SVM model to more complex CNN (Convolutional Neural Network) and LSTM (Long Short-Term Memory) models. We discuss how these models perform in terms of *model accuracy* and *computation/communication cost* in detail below.

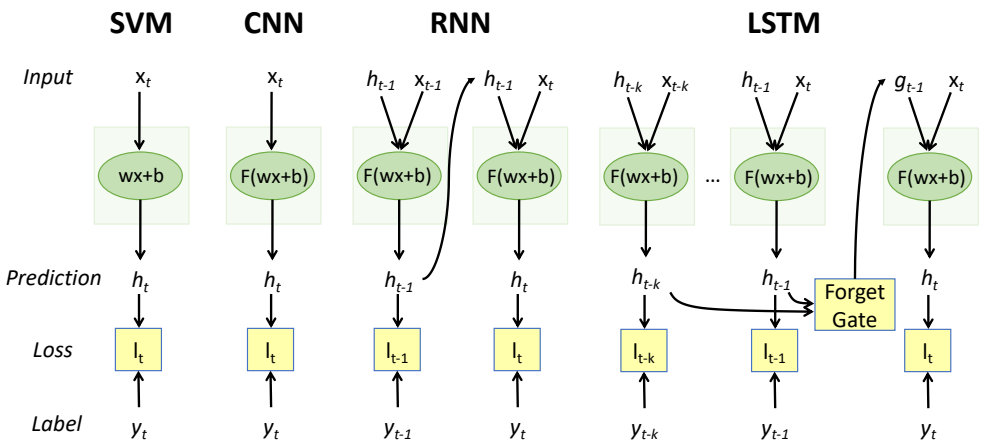

Figure 5.1: Comparing the structures of different machine learning models.

To understand the trade-off, it is useful to look into the structures of different machine learning models, as shown in Figure 5.1. Given input x_t at time t, a (linear) SVM model makes a prediction h_t with a simple function F as $wx + b$. The model then minimizes the loss l_t between the predictions $\{h_t\}$ and labels $\{y_t\}$ by adjusting the parameters w and b in the learning stage to train an accurate function F. CNN models also follow this process, except that they adopt a more complex function $F = f(...f(wx + b))$, where F is a composition of multiple (possibly) non-linear functions (e.g., a sigmoid function). The issue with SVM and CNN is that they have no "memory", i.e. the prediction h_t completely depends on input x_t at the current time t, and cannot encode patterns in history. In contrast, RNN models use previous prediction h_{t-1} as an input (along with other input x_t) to compute the current prediction h_t, thus capturing the sequential patterns in history. However, this model "forgets" very fast as the impact of prior predictions decrease exponentially [73] with each iteration. LSTM addresses this issue by adding a *forget gate* to encode a wide range of predictions h_{t-k} to h_{t-1}. The forget gate can learn to determine how much impact a previous prediction h_{t-i} should have on the current prediction h_t.

Accuracy: Complex models such as LSTM are relatively more accurate as they are able to encode more information than simpler models such as SVM; and more generally, as the models get more complex, their accuracy gets better. However, it is not clear how much information needs to be encoded for contextual IoT access patterns to achieve a satisfying level of accuracy.

Computation/communication cost: Since federated multi-task learning works in a distributed manner, different machine learning models will have different impact on the computation and communication cost. The computation cost includes the CPU and memory consumption on each edge computing node and on the cloud back-end server. The communication cost includes

the bandwidth consumption and latency introduced when transmitting the model parameters between the edge computing nodes and the cloud back-end server.

To achieve high accuracy and low communication/computation cost, we can build a simple model with an additional structure that captures the key contextual IoT access patterns. A simple model helps reduce communication/computation cost and a carefully designed additional structure to capture the key contextual IoT access patterns, cuts down the accuracy gap.

More specifically, from Figure 5.1, we observed that the main difference between a simple model (SVM) and a complex model (RNN and LSTM) is how the temporal information (long-term memory and short-term memory) is captured. The key question here is how much temporal information is needed for capturing contextual IoT access patterns. To answer this question, we make two domain-specific observations:

Short-term correlation is needed to capture contextual IoT access patterns: We observe that all the device states and some discrete environment variables at a given time t are highly correlated with the corresponding device states and discrete environment variables at a time $t-1$. For example, if a camera state transits from `off` to `on`, this short-term correlation from time $t-1$ to time t is needed. The intuition behind this observation is that all the device states and some discrete environment variables are internally following certain state machines, and in a state machine, the next state is highly dependent on its previous state.

Long-term correlation can be aggregated: For environment variables and device states, capturing the specific long-term sequence in history is not necessary. The aggregated environment variables and device states can be as effective. For example, considering the contextual access pattern that "a user feels hot in the last hour and opens the smart window". Here, the action of open the smart window is only related with the average temperature and humidity in the last hour, not specific temperature or humidity sequence.

Based on the two observations above, our idea is to add an extra temporal structure on top of simple SVM model, as shown in Figure 5.2. The extra temporal structure captures the short-term correlation and long-term aggregation. The detailed pseudo code of the temporal structure is given in Figure 5.3. To capture the short-term correlation, we apply *word2vector* processing on device states and discrete environment variables from time $t-h$ to time t, and generate vectors of *2-grams* to capture the short-term correlation between two adjacent times (Line 10 in Figure 5.3). To capture the long-term correlation, we aggregate the continuous environment variables from time $t-h$ to time t by its average value (Line 13 in Figure 5.3), and we aggregate the discrete device states and environment variables from time $t-h$ to time t by the appearance number of *2-grams* (Line 10 in Figure 5.3). In this way, LoFTI's temporal structure captures the short-term correlation and aggregated long-term correlation.

Next, we compare LoFTI and LSTM in terms of computation cost and communication cost. In distributed machine learning, the computation cost is measured by the number of CPU FLOPS, and the communication cost is measured by the number of model parameters transmitted in each iteration.

We start with LSTM's computation cost. In LSTM, there are 8 matrix-matrix multiplications per layer per time-step. Suppose N is the number of features, M is the hidden unit size, L is the minibatch size. Each matrix-matrix multiplication is $A \times B$, where A is a matrix of size $M \times N$ and B is a matrix of size $N \times L$. The matrix-matrix multiplication $A \times B$ uses $ML(2N-1)$ CPU FLOPS. Suppose r is the number of layers and h is the number of timesteps. Then, the total

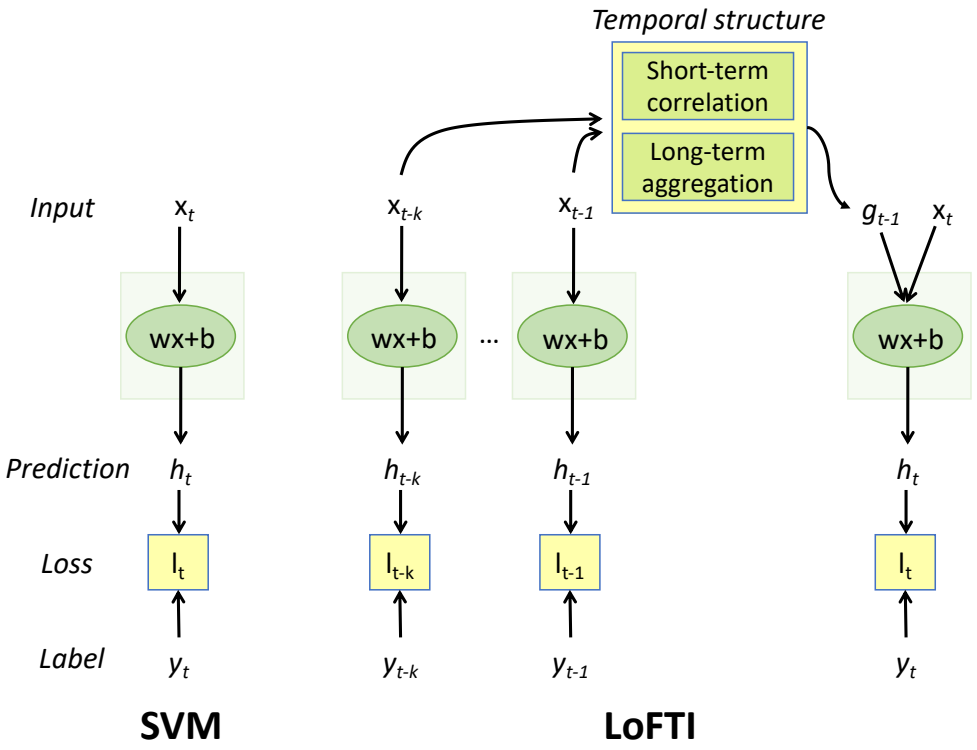

Figure 5.2: LoFTI model.

FLOPS for LSTM is $r * h * 8 * ML(2N - 1)$.

Next we calculate the FLOPS for LoFTI. We know that the total FLOPS for a linear SVM is proportional to the number of parameters. Suppose N is the number of features and h is the number of timesteps for long-term aggregation for LoFTI (we set it equal to the LSTM's timesteps). In LoFTI's model, it takes $N * h$ FLOPS to calculate the temporal structure and $2 * N$ FLOPS to calculate the model parameters. Therefore, the total FLOPS for LoFTI is $2 * N + N * h$. It is obvious that LoFTI's computation cost is much less than LSTM.

Next, we compare the communication cost of LoFTI and LSTM. Since FMTL only transmit parameters, the communication cost is propotional to the number of model parameters. The number of model parameters of LSTM is given by: $4 * (N * M + M^2)$, where N is the number of features and M is the number of hidden unit. The number of model parameters of LSTM is given by: $2 * N$, where N is the number of features. Therefore, it is obvious that the communication cost of LoFTI is lower than LSTM.

In summary, LoFTI reduces the computation cost by applying a simple model with a concise temporal structure.

5.3 Addressing unbalanced contextual IoT access records

A straightforward approach to construct the dataset is to collect the benign and malicious contextual IoT access records as it is. However, while benign users are frequently accessing the IoT

```
Input: {x_{t-h}, ..., x_{t-1}, x_t} is the sequence of features
    t is current time, h is history time window
    m is the number of key features
{env} is the environment variable set
{state} is the environment variable set
Output: g_t is the temporal aggregation
for i ← 1 to m
  if x_{t,i} ∈ {env} or x_{t,i} ∈ {state}
    if x_{t,i} is discrete
      g_{t,i} = word2vector({x_{t-h,i}, ..., x_{t-1,i}})
    end
    if x_{t,i} is continuous
      g_{t,i} = mean({x_{t-h,i}, ..., x_{t-1,i}})
    end
  end
end
```

Figure 5.3: LoFTI's temporal structure.

devices, attackers only need a few accesses to cause privacy leakage or physical hazards. This means that the number of benign IoT accesses will be much larger than the number of malicious IoT accesses, i.e., the dataset is unbalanced. Therefore, the decision boundary will be pushed towards the attack samples and causing high FNs. A strawman solution to balance the dataset would be to copy the attack samples and construct a repetitive dataset. However, the repetitive attack samples cannot capture the variation of attacks and fails to provide coverage. A general approach in machine learning to address unbalanced data is called *data augmentation*[120]. The idea is to introduce variance in the samples to balance out data and improve coverage. However, common data augmentation mechanisms, including cropping, rotation and flipping are closely tied to image representation (as it has originated from computer vision community), and are not applicable for processing the contextual access abstraction for IoT devices.

Therefore, we designed a customized *data augmentation* approach for processing the contextual access abstraction for IoT devices. Specifically, we provide two mechanisms:

Context random sampling: The context random sampling mechanism is inspired by the cropping of an image. But instead of selecting a region of the image (considering locality), our mechanism selects a subset of the environment variables and device states with a fixed ratio r (commonly set to 10% to 20% in sampling papers[66]). The idea is to perform random sampling on the environment variables and device states to narrow down the correlation between environment variables/device states and IoT access with certain action.

Adding contextual noise: The second mechanism is adding noise to the context. This is similar to adding noise to an image, but instead of adding noise to pixels, LoFTI adds noise to the environment variables of the attack sample. The idea is to increase the coverage of environment variables in attack samples. However, deciding how much noise to add can be a challenge. If too much noise is added, the attack sample may deviate to the benign space and cause FNs. If too little noise is added, the coverage will be low. To address this issue, we provide a mechanism to automatically decide the amount of noise being added. The first step is to estimate a rough decision boundary L. Then, for each attack sample, we calculate the distance metric d to the

Cases	Device	Attack	Type
1	Camera	Private video leakage[14]	privacy leakage
2	Smart Door	Break-in when user is asleep[16]	physical hazards
3	Heater	Bedroom overheating[15]	physical hazards
4	Smart TV	Eavesdropping[22]	privacy leakage
5	Stove	Fire hazards[20]	physical hazards
6	Smart Window	Break-in[19]	physical hazards
7	Smart Light	Blackout[17]	physical hazards
8	Smart Toilets	Water overflow[21]	physical hazards

Table 5.2: Test cases for malicious IoT access based on reported IoT breaches.

rough boundary. LoFTI ensures that the noise added to the attack sample does not exceed ϵd, where ϵ is a constant within $(0, 1)$ range. In this way, LoFTI provides a heuristic-based method to ensure that the noise is bounded and can increase the coverage without increasing the FNs.

5.4 Evaluation

In this section, we evaluate LoFTI and show that:

- LoFTI achieves high detection accuracy and low FPs and FNs when compared with single-home learning and all-home learning, reducing FNR by 24.2% and FPR by 49.5%.
- LoFTI achieves high coverage over the contextual policies needed when compared with current approach to manually generating contextual policies.
- LoFTI's key features are effective for detecting malicious IoT access causing physical hazards or privacy leakage.
- LoFTI's simple SVM-based model with temporal structure can achieve a good trade off between accuracy and communication/computation cost.
- LoFTI's data augmentation effectively addresses the unbalanced data issue.
- LoFTI can detect new attacks not in the dataset, if there are historical records about the IoT device being attacked.

5.4.1 Dataset

To evaluate LoFTI, we use a large public IoT historical records dataset - CASAS [7] with real IoT deployment in more than 400 smart homes in cities including Kyoto, Paris, Milan, etc. The type of IoT devices in the dataset includes smart TV/DVD, smart oven/fridge/microwave, smart door/sofa/bed/cupboard, motion/power/temperature sensors, etc. The CASAS dataset includes historical records about how normal human users are interacting with these IoT devices in typical smart home setting. To test how well LoFTI can detect physical hazards and privacy leakage, we constructed a list of physical hazards and privacy leakage cases based on IoT breaches in news reports [14, 15, 16, 17, 18, 19, 20, 21, 22], as shown in Figure 5.4. We add these cases randomly to each of the 400 smart homes.

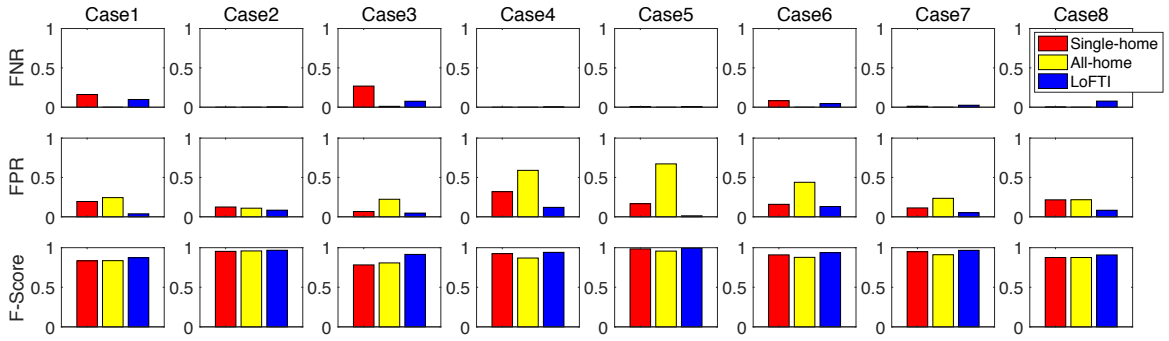

Figure 5.4: Comparing single-home learning, all-home learning and LoFTI.

5.4.2 Accuracy

In this part, we measure the *accuracy* of LoFTI, and compare LoFTI with the state-of-the-art single-home learning approach and all-home learning approach.

The *accuracy* is evaluated by the FNR (False Negative Rate), FPR (False Positive Rate) and F-Score. The FPR describes the occurrence of False Positives, defined as $FPR = \frac{FP}{FP+TN}$. The FNR describes the occurrence of False Negatives, defined as $FNR = \frac{FN}{FN+TP}$. The F-Score reflects the detection performance combing FP and FN, defined as $FScore = \frac{2*Precision*Recall}{Precision+Recall}$, where $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$.

Approach	FNR	FPR	F-Score
Single-home	0.0832	0.1697	0.8867
All-home	0.0318	0.3409	0.8534
LoFTI	0.0241	0.0857	0.9378

Table 5.3: Overall.

Table 5.3 shows the overall FNR, FPR and F-Score comparing single-home learning, all-home learning and LoFTI. The overall result shows that LoFTI achieves low FNR/FPR and high F-Score, reducing FNR by 24.2% and FPR by 49.5% when compared with the best results from single-home learning and all-home learning approaches.

Next, we compare single-home learning, all-home learning and LoFTI case by case. Figure 5.4 shows the FNR, FPR and F-Score for each case in Table 5.2 for single-home learning, all-home learning and LoFTI. The first row depicts the FNR, the second row depicts the FPR for different cases. The last row depicts the F-Score combining the impact of FP and FN. From the result, we can see that single-home learning result in high FNR in case 1, 3 and 6. All-home learning result in high FPR in case 1, 3, 4, 5, 6, 7 and 8. From Figure 5.4, we can see that single-home learning and all-home learning either result in high FNR or high FPR, and LoFTI achieves low FPs and FNs across all the test cases.

Case	1	2	3	4	5	6	7	8
LoFTI>All>Single	✓	✓	✓					
LoFTI>Single>All				✓	✓	✓	✓	✓

Table 5.4: Detailed analysis for the test cases.

Table 5.4 shows the detailed analysis for the test cases. We rank single-home learning, all-home learning and LoFTI by the F-Scores they achieve in each case. For case 1,2 and 3, LoFTI achieves higher accuracy (F-Score) than all-home learning than single-home learning. This is because the *insufficient data* issue has a major impact in these cases. To confirm this, we counted the number of IoT access records relevant for distinguishing malicious/benign access in each home (e.g., if malicious access is turn on Camera, we count all the access to Camera). We found that, in case 1,2 and 3, the average relevant IoT access records in each smart home is less than 200, and some smart home have less than 50 relevant IoT access records. This explains the *insufficient data* issue in case 1,2 and 3. For case 5 to 8, LoFTI achieves higher accuracy (F-Score) than single-home learning than all-home learning. This is because the *diversity* issue has a major impact in these cases. For example, in case 6, different users in different smart homes will open their window under different temperature/humidity; in case 5, different users in different smart homes have different interaction patterns with their smart fridge/microwave oven/cupboards before they turn on the oven. In summary, LoFTI achieves low FPs and FNs comparing with single-home learning and all-home learning.

Device	Policy
Camera	If user is at home, block the access to turn on the camera
Smart Door	When user is asleep, block the access to open the smart door
Smart Heater	When temperature $> 16°C$, block access to turn on heater
Smart TV	No policy found
Smart Stove	No policy found
Smart Window	No policy found
Smart Light	From 6pm to 8am, smart light can be turned on
Smart Toilets	No policy found

Table 5.5: Contextual policies from user study[70]

5.4.3 Comparison with manually generated policies

In this part, we compare LoFTI with the current approach to manually generate contextual policies. We compare with an user survey[70] that asks 425 users to allow/block IoT accesses (e.g., turn on the camera) based on context norms (e.g., time of the day, whether user is at home or not).

To compare with this approach, we collect all the policies (user made allow/block decisions) in the user study[70] that are related with the IoT devices listed in our dataset, as shown in Table 5.5. We then convert the user generated policies into context-based rules and carefully maps the context from the user study to the corresponding context in our dataset. For example, we map the context that "user is asleep" in the user study to the context that "smart bed is occupied" in our dataset. We then apply the converted context-based rules on our testing dataset and measure the FNR and FPR. We provide no comparison for smart TV, smart window, smart stove and smart toilet, as we cannot find user generated policies for them in the user study.

For the rest 4 types of IoT devices - camera, smart door, smart heater and smart light, Figure 5.5 shows the FNR and FPR comparing user policy generated from survey and LoFTI. In

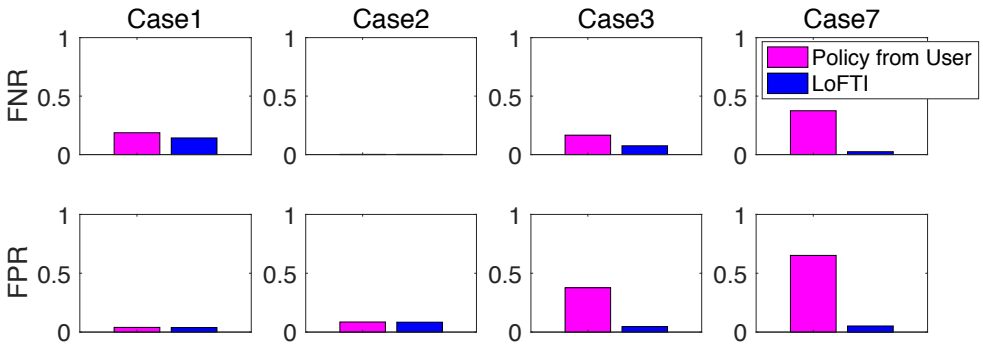

Figure 5.5: Comparing user policy from survey and LoFTI.

case 1 and case 2, the FNR and FPR for user policy generated from survey and LoFTI are similar. This is because the scenario is relatively simple. In case 3, the FNR and FPR for user policy are higher for two reasons. First, user policy generated from survey would set a fixed value for temperature, which is not precise and not customized for different smart homes. Second, user policy generated from survey fails to cover other environment variables that also impact the access of smart heater, e.g., humidity. In case 3, the FNR and FPR for user policy are much higher because it captures the wrong context. The user policy from survey depend on time as context to guide the access of smart light. However, in CASAS dataset, we observed that it is the movement of human sensed by motion sensors that guides the access of smart light. This is a typical case of misconfiguration by manually generated contextual policies.

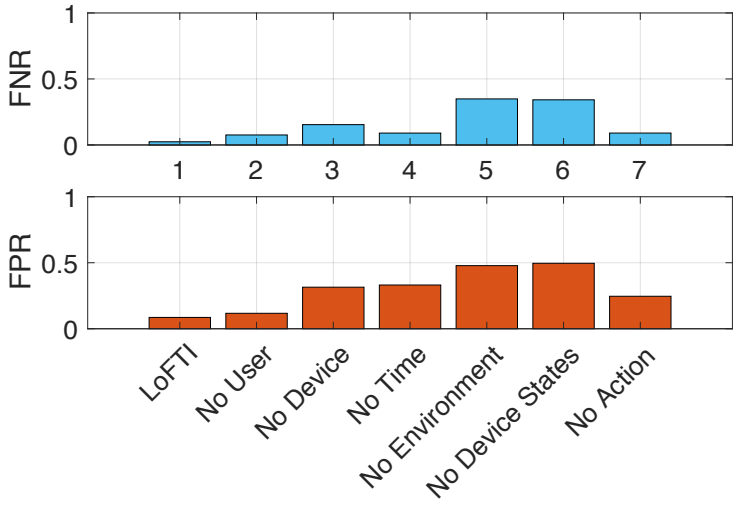

Figure 5.6: Effectiveness of LoFTI's key features.

5.4.4 Effectiveness of LoFTI's key features

In this part, we evaluate how effective the key features generalized by LoFTI in terms of distinguishing benign and malicious contextual IoT access. To do so, we set the value fields of each

key features to default value and measure how much the FNR and FPR will increase, as shown in Figure 5.6. The result shows that if any of the 6 types of key features defined by LoFTI is set to default, the FNR and FPR will increase. Therefore all 6 types of key features are effective in distinguishing benign and malicious contextual IoT access. We can also see that environment variables and device states cause the highest increase in FNR and FPR. This is not surprising as environment variables and device states are critical features to capture the context. Here the "user" feature is not causing a high increase in FNR and FPR. This is because of the CASAS dataset, where there are not many homes recorded with multiple users.

5.4.5 Effectiveness of LoFTI's simple SVM-based model with temporal structure

In this part, we will show that LoFTI's simple SVM-based model with temporal structure can achieve a good trade off between accuracy and communication/computation cost.

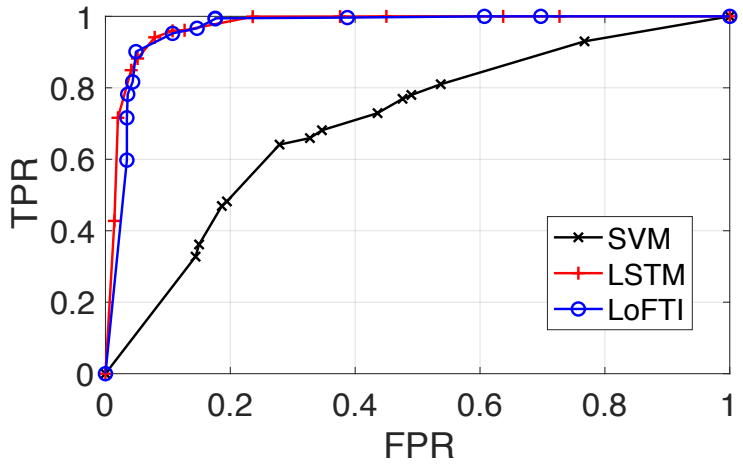

Figure 5.7: ROC curve for SVM, LSTM and LoFTI.

Accuracy: We compare LoFTI with the most simple machine learning model - SVM and most complex machine learning model - LSTM, from common machine learning models considered (including SVM, CNN, RNN, LSTM). As shown in Figure 5.7, we evaluate their detection capability by plotting the ROC (Receiver Operating Characteristic) curve. The x axis is the TPR (True Positive Rate) defined as $TPR = \frac{TP}{TP+FN}$, and the y axis is the FPR defined as $FPR = \frac{FP}{FP+TN}$. If the curve is more close to the $(0, 1)$ point, the corresponding approach has a stronger detection capability. To generate the curves, we adjust the regularization parameter c for SVM; we adjust the regularization parameter c and the window size of the temporal structure for LoFTI; we adjust the number of layers r, timesteps h and hidden units M for LSTM. From Figure 5.7, we can see that LoFTI can achieve a detection capability close to LSTM, stronger than SVM. This shows that LoFTI using simple model with temporal structure can achieve a good accuracy.

Computation cost: In this part, we compare LoFTI and LSTM in terms of computation cost and communication cost. We measure the RMSE (Root Mean Square Error) in each iteration for

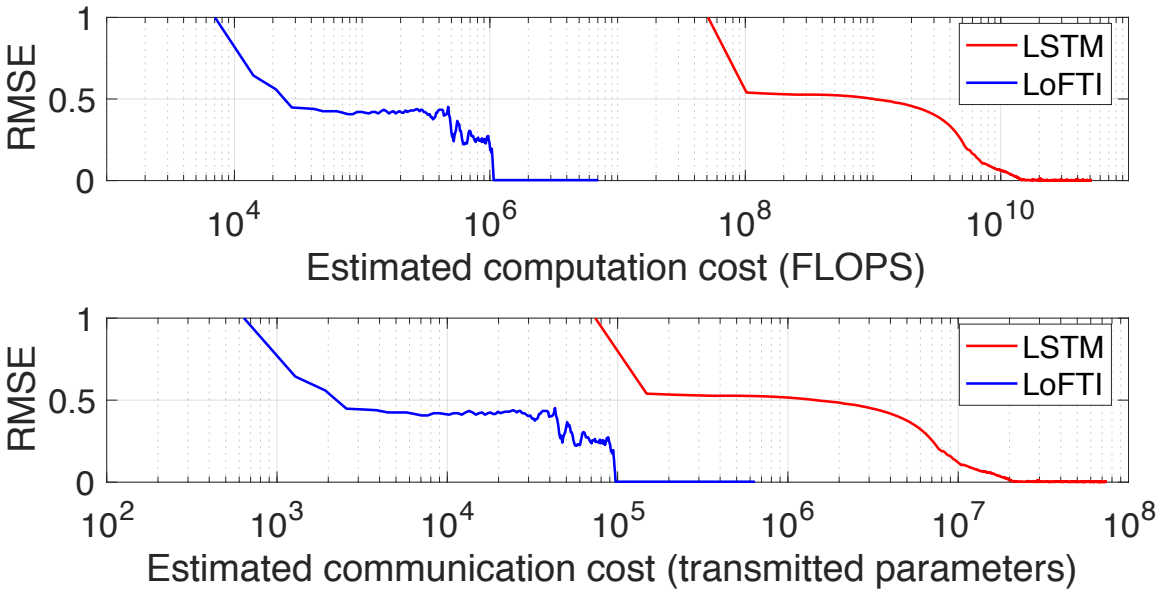

Figure 5.8: Estimated communication/computation cost.

LoFTI and LSTM. Then we estimate the computation cost and communication cost for LoFTI and LSTM in each iteration. Therefore, we can estimate how much computation cost and communication cost are needed to reduce RMSE (Root Mean Square Error) close to 0 (training complete).

We estimate the computation cost (CPU FLOPS) for LSTM as $r * h * 8 * M * L * (2 * N - 1)$, where r is the number of layers, h is the number of timesteps, N is the number of features, M is the hidden unit size, L is the minibatch size. We estimate the computation cost (CPU FLOPS) for LoFTI as $2 * N + N * h$, where N is the number of features and h is the time window span (we set it equal to the LSTM's timesteps).

We estimate the communication cost (transmitted parameters) for LSTM as $4 * (N * M + M^2)$, where N is the number of features and M is the number of hidden unit. We estimate the communication cost (transmitted parameters) for LoFTI as $2 * N$, where N is the number of features.

Figure 5.8 shows the result. We can observe that the computation cost of LSTM is 4 magnitudes higher than LoFTI; while the communitation cost of LSTM is 2 magnitudes higher than LoFTI. This result is as expected considering the complexity of neural network layers and gate structures used by models such as LSTM.

In summary, LoFTI can achieve a good trade-off between accuracy and computation/communication cost.

5.4.6 Effectiveness of LoFTI's data augmentation

In this part, we evaluate the effectiveness of LoFTI's data augmentation to address the *unbalanced data* issue. As shown in Figure 5.9, we compare the FNR and FPR with different dataset construction mechanisms including unchanged, repetitive and LoFTI's data augmentation. From

Figure 5.9 we can see that the unchanged dataset result in high FNR. This is because the large number of benign samples are pushing the decision boundary towards the attack samples (to avoid FPs), and result in high FNR. The repetitive mechanisms cannot fully address this issue because it still has a poor coverage over attack samples, so the FNR is still high. LoFTI's data augmentation mechanism successfully reduces the FNR while keep FPR almost at the same level.

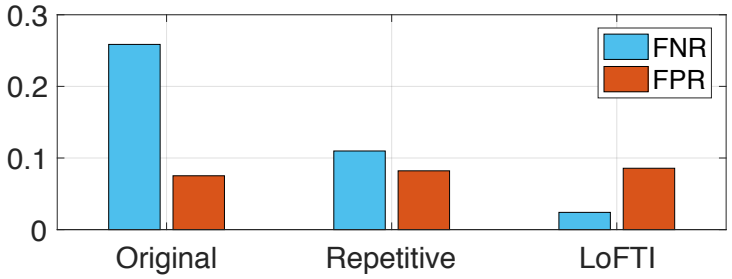

Figure 5.9: Different dataset construction mechanisms.

5.4.7 LoFTI's capability of handling new attacks

Since LoFTI is a supervised learning approach. One interesting question to ask is whether LoFTI is able to detect new attacks that have never appeared in the training dataset before. To test this, we use 3 types of new attacks and test them against the model learned from dataset without samples of these attacks, as shown in Table 5.6.

New case	Device	Attack	Type
A	Camera	Eavesdropping	privacy leakage
B	Smart Door	Break-in when nobody at home	physical hazards
C	Smart TV	Copy watching records	privacy leakage

Table 5.6: New IoT attacks for testing.

Figure 5.10 shows the FNR and FPR for the 3 types of unknown IoT attacks and their corresponding known attacks. For case A and case C, LoFTI still can achieve low FNR and FNR. For case B, the FNR and FNR on unknown IoT attacks are significantly higher. This is because for case A and case C, the context during the new attack is different from the context of the benign access. For case B, the context during the new attack is close to the context of the benign access.

In summary, LoFTI can detect unknown IoT attacks if the context during the new attack is different from the context of the benign access. We also note that LoFTI cannot detect unknown IoT attacks against an IoT device that has no historical record of benign and malicious access in all smart homes. In such case, it is understandable as LoFTI is constrained by the dataset it can learn from.

5.4.8 Masking sensitive parameters

The federated learning guarantees that the attacker is not able to see the raw data. However, a strong attacker who can observe the parameters of the model (e.g., via compromising the cloud backend or MITM attack) may be able to infer certain private information about the smart homes.

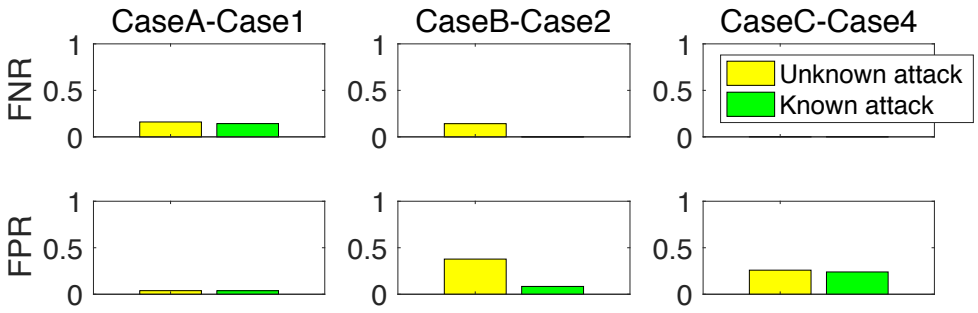

Figure 5.10: FNR/FPR on unknown/known IoT attacks.

In this part, we perform a test to see if we can mask some sensitive environment variables in the learning process to prevent the observation from attackers, without significant decrease the detection accuracy. In this test, we mask a number of sensitive information from previous sensitive information tracking literature[43, 48, 115], including device location, device manufacturer, firmware version, device IP address, open ports, zip code and contact list. Then we measure the FNR, FPR and F-Score before and after the mask operation.

From Table 5.7, we can see that the FNR and FPR before and after masking are the same. It shows that some of the sensitive information, including device location, device manufacturer, firmware version, device IP address, open ports, zip code and contact list, can be masked without significantly decreasing the detection accuracy. In the next step, we plan to explore other strategies such as *differential privacy* to see if we can mask a wider range of sensitivity information without significantly decreasing detection accuracy.

Approach	FNR	FPR	F-Score
Before mask	0.0241	0.0857	0.9378
After mask	0.0241	0.0857	0.9378

Table 5.7: FNR/FPR before/after masking.

Chapter 6

Enforcement

In Chapter 3, we presented PSI's vision to provide context-based and agile enforcement for IoT devices. We have also seen that the key issues to make PSI practical are:

- How to provide expressive policy abstraction?
- How to provide scalable and responsive orchestration?

In this chapter, we will answer these questions with detailed design of PSI's enforcement architecture. After that, we will present the result of evaluating PSI for its enforcement benefits.

6.1 PSI Policy Abstraction

In this section, we identify key expressiveness requirements that PSI policies should satisfy and then describe our solution.

6.1.1 Requirements

We begin by highlighting three key requirements that the PSI policy abstraction should meet to help the administrator to express intents that govern how the devices' traffic should be processed/forwarded.:

- *Context-based forwarding & processing:* The administrator should be allowed to define a set of context for each device, and express forwarding & processing based on the context. For example, if a host is sending oversized DNS packets (detected by header checker), the administrator should be allowed to define a context to note that a host is suspicious for data exfiltration (DNS-based), and specifies that the oversized DNS packets should be forwarded to a DPI for payload check to prevent potential data exfiltration.
- *Agile intent evolution:* The administrator's intent over a device's traffic may evolve over time as the context changes. For instance, if a IoT device initiated a HTTP connection to a suspicious website, the context is changed from device is normal to device is suspicious, and the inbound traffic to this device should be put under deeper scrutiny (, DPI) to stop malicious downloads.

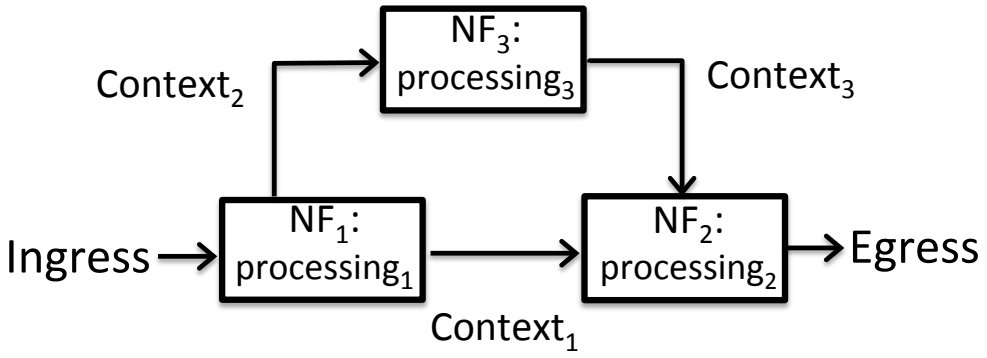

Figure 6.1: Express context-based intent with ψDAG.

A natural question then is whether prior policy abstractions can satisfy the requirements. We evaluated a number of existing approaches including distributed Firewall/IPS configurations (Cisco's PIX[31],Bro[94]) and more recent SDN policy languages (Kinetic [81], Merlin [114], Group-based Policies (GBP) [30], PGA [96]), and found that they cannot meet the two requirements. For distributed Firewall/IPS configurations, the context and agility that can be expressed is limited to one type of tools, , a firewall cannot tell a oversized packet to be forwarded to a IPS for DPI. We qualitatively evaluated distributed Firewall/IPS configurations in the coverage test, and the result shows that the context-based and agile policies prevented 35% more potential attacks than distributed Firewall/IPS configurations. Similarly, SDN policy languages are largely limited to switch contexts (packet header attributes), and their agility is limited to static forwarding capabilities.

Our goal in designing the PSI policy abstraction is not to claim a novel theoretical or programming language contribution (, [52, 98]). Rather, it is a practical abstraction for capturing context-dependent and agile processing.

6.1.2 High-Level Ideas

Context-based processing

To enable administrator to express traffic forwarding and processing based on context, PSI uses a Directed Acyclic Graph called ψDAG, as shown in Figure 6.1. Formally, this is a two tuple $\psi DAG = \langle NFInstances, NextHops \rangle$. Each vertex in ψDAG represents a processing function (, IPS) denoted as $NFInstance$ that can: (1) tag the traffic based on the processing outcome (, tag packet with "oversized" as context); (2) apply custom process based on the tags (, check the payload for packet with "oversized" context). The edge relation $NextHops$ specifies the intended sequence of traversal through processing tools with respect to context; , packet with "oversized" context is forwarded to a IPS (to check payload).

Supporting intent evolution with ψFSM

While the ψDAG abstraction captures context-dependent processing, it does not capture the evolution of security intent. For instance, the operator may have an inkling of future "states"

of the host (contexts at different time) and may want to proactively express the intent for these subsequent likely states as well. Similarly, the intent may evolve as new information arrives; , new vulnerabilities or new alerts from external sources. To capture such intent evolution, we introduce the ψFSM abstraction. Formally, the ψFSM is a Moore machine $\psi FSM := \langle S, s_{start}, \mathcal{E}, \mathcal{F}, T, O \rangle$ that maps the traffic's state to a specific ψDAG via the output function O ($S \rightarrow \mathcal{F}$). In the simplest case, we have a single global state for a specific class with a single ψDAG. More generally, we can define state *transitions* based on different events $E_j \in \mathcal{E}$ and the ψDAG will depend on the current state.

Note that from Section 3, our policy is isolated at a per device/traffic granularity. Therefore, each device/traffic is assigned with a ψFSM and a ψDAG; the ψFSM captures current state and future state evolution; and the ψDAG captures the current intent with respect to the current state.

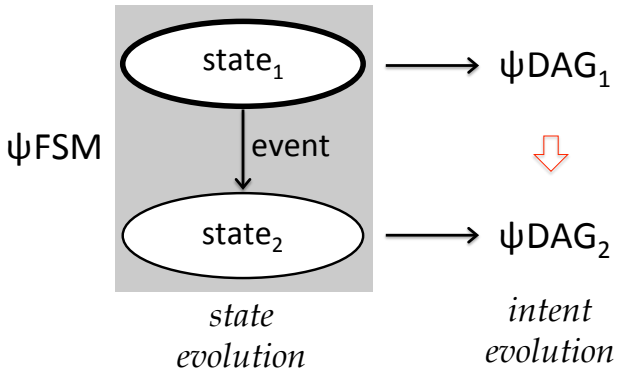

Figure 6.2: PSI agile intent evolution.

6.1.3 Illustrative Example

Let's consider an end-to-end example to put the ideas together and see how PSI policy abstraction enables us to express effective security postures. Figure 6.3 illustrates a dynamic policy that can protect a NAS device. There are two key structures for the dynamic policy in this example. The first structure is the FSM structure. Each state in the FSM captures the security state of the NAS. The second structure is DAG (Directed Acyclic Graph) structure. Each DAG specifies the security enforcement for a given security state of the NAS. The NAS is vulnerable to the code injection attack called SambaCry. Initially the NAS is at a normal state. The policy will enforce a scan detector for the NAS at the normal state. When the NAS is being scanned, it will transfer to scanned state. Then, a deep packet inspection will be enforced to prevent the code injunction against the NAS. Note that this policy abstraction is context-based because the enforcement is based on the security state of the device. And this policy abstraction is agile because it can evolve from one enforcement to another over the time.

Once we have the abstract policy, the PSI Policy Engine will interpret the abstract policy and compute the real-time security *intent* (ψDAG) for each device's traffic based on current context.

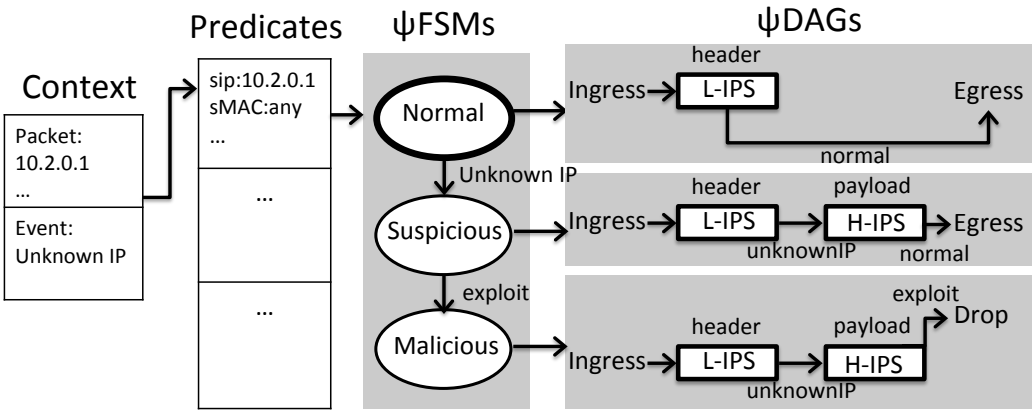

Figure 6.3: A dynamic scrutiny policy expressed via PSI.

6.2 PSI Controller

In this section, we describe the PSI controller's orchestration mechanisms to translate the high-level intents into a concrete realization. To highlight the key scalability and responsiveness challenges, we begin with a simple *reactive* design. Then, we discuss our ideas to address these challenges: proactive tag-based forwarding, elastic controller scaling, and ψDAG prefetching.

6.2.1 Conceptual View and Challenges

The input to the PSI orchestration module is the policy intent (ψDAG) for each device. The goal of the controller is to translate these into a concrete realization; , launch $\psi mbox$ instances and set up forwarding rules. Recall that PSI intents capture dynamic packet processing at two levels. First, the forwarding path may depend on flow-specific context information from upstream nodes in the ψDAG.

A seemingly natural solution to implement these intents is to adopt a *reactive* mechanism, which configures forwarding rules and deploy $\psi mboxes$ on-demand whenever the context changes.

Per-packet processing

Suppose a packet belonging to class i arrives at a network interface (either at a $\psi mbox$ or a switch). By default, this node will not have any forwarding rule. Thus it buffers the packet and sends an `PKTIN` event to the controller with the packet's header and any relevant processing context. On receiving the `PKTIN` event, the controller retrieves the current state ${s_i}^{current}$ corresponding to this packet's class and uses it to get $\psi DAG^{current}$. Based on the current context and the node that generated the packet, the controller decides the next hop for this packet and sets up forwarding rules to ensure that the packet will traverse the intended path.

Event processing

The previous discussion handles the case for a given state s. Other types of events (, IDS alerts) may trigger a state transition in the ψFSM_i, e.g., transition from normal state to suspicious state in Figure 6.3. This in turn may require a new ψDAG to be instantiated. In this case, the controller retrieves the ψFSM_i corresponding to the event,[1] looks up the ${s_i}^{current}$ and identifies the ${s_i}^{next}$. The controller then identifies the new ψDAG^{next} corresponding to the new state s, deactivates the current ψDAG, and launches new $\psi mbox$ instances to implement ψDAG^{next}, and updates internal data structures to indicate a state change.

Challenges

While this above workflow is conceptually correct, we identify two key challenges:

- *Scalability with adversarial workloads:* First, handling every packet presents a fundamental scaling challenge - a single controller has to process a control message for every packet in the network. Second, even if we do not interpose the controller on every packet at every hop, the controller needs to deal with a large number of events in any reasonable-sized network that will induce ψDAG updates.[2] Thus, an adversary can easily saturate the CPU, memory, or the control channel bandwidth with this naive approach.
- *Security downtime:* Transitioning from the $\psi DAG^{current}$ to the ψDAG^{next} will need new VMs to be launched and other forwarding rules to be setup. Even with fast VM bootup techniques, there will be a non-trivial latency. Thus, adversaries can exploit these delays in setting up ψDAG^{next} to achieve their goals for the types of multi-stage attacks described earlier.

6.2.2 Key Ideas in PSI

Next, we describe how address each of these challenges.

Proactive context-based forwarding

The reactive controller does not scale as it interposes on every packet at every logical forwarding between two $\psi mboxes$.

To avoid this, our goal is to keep packets in the data plane as much as possible [63]. We achieve this with a proactive forwarding approach extending prior work [63]. Each $\psi mbox$ *tags* outgoing packets specifying the relevant context needed for forwarding along the ψDAG. For example, all packets are initially under "normal" tag and the tag changes to "suspicious". The forwarding logic of the network switches will incorporate these tags as part of their packet processing actions. For example, if the packet is tagged as "normal", it is forwarded to port 1, and if the packet is tagged as "suspicious" it is forwarded to port 2. Note that because the

[1] We assume that events are annotated with the class value (i).

[2] Even if the likelihood of updates for a single device is low, at any given time there are likely to be several devices that need updates.

controller has a logically global view it can proactively install these per-tag forwarding rules for each class $class$ without waiting for a PKTIN event.

Now, there might be two potential deployment concerns. First, we need sufficient space in the packet header to add these tags. This is not an issue as new standards for virtual network forwarding and network service chaining headers explicitly include additional header space for metadata [82, 118]. With software switching (, OpenvSwitch) and new switch pipelines [46], it is possible to add flexible header matching rules based on these tag bits. Furthermore, these header tags are only needed inside the $\psi cluster$, where we are not constrained by legacy networking. Second, the $\psi mboxes$ have to explicitly expose these tags. Prior work shows that the modifications required to commodity middleboxes to add the tagging logic is less than 50 lines of code [63].

Scale-out controller

While proactive forwarding addresses the scaling problem in dealing with PKTIN arrivals, it does not address the issue of an adversary sending crafted data to generate a large event/alert volume to overload the controller. Note, however, that it may be hard to distinguish events triggered by adversary actions vs. legitimate users. Rather than introduce ad hoc anomaly detection algorithms to try and differentiate legitimate vs. adversary-induced events, we exploit the fact that the PSI design allows us to logically partition the event handling across different traffic classes. That is, we can simply *horizontally scale* the PSI controller and add more instances as needed depending on the offered load [59, 84]. This is especially easy because the different traffic classes are independent and do not introduce any synchronization bottlenecks at the controller.

We design a scale-out mechanism similar to elastic scaling solutions in cloud deployments [13, 59]. While the elastic scaling solutions only migrates switches states, PSI's scale-out mechanism migrates device attributes, security states and policy specifications within the predicates, ψFSM and ψDAG structures. A simple runtime monitor inspects the response time for each controller instance. If the response time is starting to increase more than a preset threshold, it invokes an elastic scaling routine that adds an extra control instance and splits the traffic classes currently handled by the overloaded instance across the added instances, and migrate the corresponding structures.

ψDAG prefetching

The combination of proactive context-based forwarding, partitioning, and elastic scaling effectively addresses the scalability bottleneck. However, the problem of security downtime during ψFSM transitions still remains.

To address this, we use the following idea. Since the controller has the entire ψFSM described by the policy intent, it can look ahead and predict the next k possible states for each class $class$. Then, it proactively installs the $\psi mboxes$ corresponding to the ψDAGs for these next k possible states, in order to mask the latency involved when these might be needed in the future.

One concern might be that this needlessly increases the resources used by the $\psi cluster$ as ψDAG instances may never be exercised. While this is theoretically valid, in practice, we can address it as follows. First, the controller installs the future ψDAGs for the same class to be

multiplexed on the same hardware as the current ψDAG for that class. Since there is only one active ψDAG for a given traffic class at a given time, this incurs no additional hardware resources. Second, we implement some simple optimizations to do an *incremental* launch; , if there are common $\psi mbox$ instances across the future ψDAGs then we can reuse these instead of cloning them. This last optimization also indirectly helps to reduce the downtime by reducing the number of new VMs we need to launch.

6.3 Results

We now evaluate PSI's ability to deal with attacks launched from deep inside the smart home/enterprise network (stealthy attacks). To do so, we conduct an attack graph analysis against two example stealthy attacks: an example insider threat attack and an example APT. We compare distributed Firewall/IPS's and PSI's ability to detect and mitigate both attacks over three different network topologies. Through this analysis, we show that PSI is capable of identifying and mitigating 35% more potential stealthy attacks than a distributed Firewall/IPS approach, as demonstrated in Table 4.1. These results are due to topological blind spots that PSI is designed to address, and which are common on live enterprise networks.

Attack graph analysis[105] evaluates defensive systems via graph coverage. Each attack is expanded to a graph showing the potential routes the attacker can use to achieve their goals, and defenses are evaluated by comparing the number of paths each defense cuts off from the attack. We chose attack graph analysis because it integrates the structural impact of the network's design on the attack's effectiveness, and enables us to identify the cause of defensive failure. Our attack graphs consist of a tree, G, where each node is a device on the network, and each edge is an attack step. In G, an attack is a path from the root device (the source of the attack) to the leaf (target), e.g., an malware exploit from a laptop to a server through a local switch. An attack is *prevented* if one step of the attack is detected and prevented by the defense system, e.g., a IPS connected to the switch detects and blocks the malware traffic. This yields a coverage metric of the form: $coverage = \frac{num.\ of\ prevented\ attacks}{num.\ of\ all\ possible\ attacks}$. This coverage metric evaluates how many potential attacks are prevented in a given network topology.

Topology	distributed Firewall/IPS Coverage	PSI Coverage
mini-stanford [80]	52%	92%
apt-mcafee [32]	59%	91%
pix-cisco [34]	56%	89%
all	56%	91%

Table 6.1: Coverage over stealthy attacks.

In our evaluation, we instantiate the three enterprise networks in our testbed and install distributed Firewall/IPS and PSI respectively. For distributed Firewall/IPS, we assume there are no resource constraints and deploy a distributed instance at every switch in the topology. To setup insider threats in each enterprise network, each time a device is set as an insider and it exfiltrate data from all other devices using ftp-based exfiltration or DNS-based exfiltration[12]. For APT, we setup a device in external network as an attacker, and assume it can always break-in with a zero-day attack. We, then, use two pcap traces (Angler EK and Magnitude EK[24]) to emulate

the following port-scanning and exploits phases of the APT attack and see if the exploit traffic can reach another device (if yes, the APT attack on the device is valid).

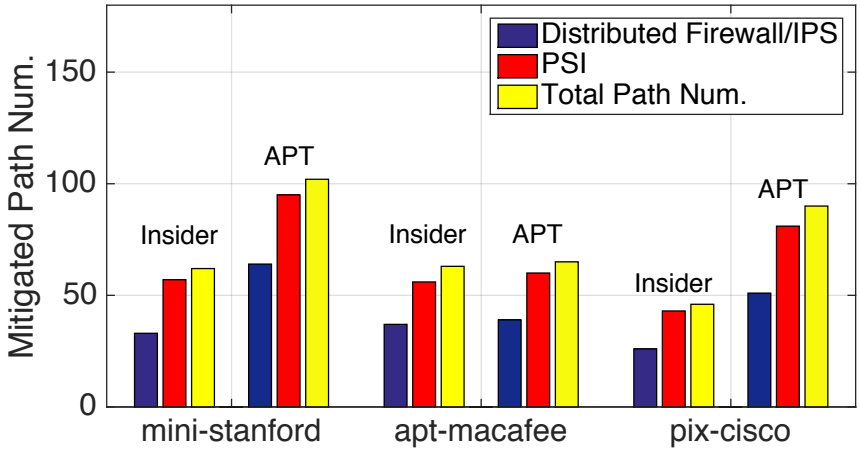

Figure 6.4: Detailed results about coverage for insider threats and APT in each enterprise network.

The detailed evaluation results are presented in Figure 6.4 and Table 6.1. In summary, out of 428 potential attack paths, distributed Firewall/IPS mitigates 240 paths (56% coverage), while PSI mitigates 392 potential paths (91% coverage). We analyzed the 152 paths that PSI mitigated but distributed Firewall/IPS did not—all of them are caused by one or more fundamental topology constraints that distributed Firewall/IPS cannot address. Specifically, 103 paths involve NAT/DHCP; 75 paths involve devices connected to multiple switches (, For high bandwidth or failure tolerance [27]); and 54 paths involve dumb, unmanaged switches. PSI cannot provide perfect coverage because first step, a zero-day attack, is undetectable. In summary, PSI improves the coverage for stealthy attacks by 35%.

Chapter 7

Integrating RADIO, LoFTI and PSI

In this part, we show how to integrate RADIO, LoFTI and PSI as the end-to-end system, by walking through two usecases.

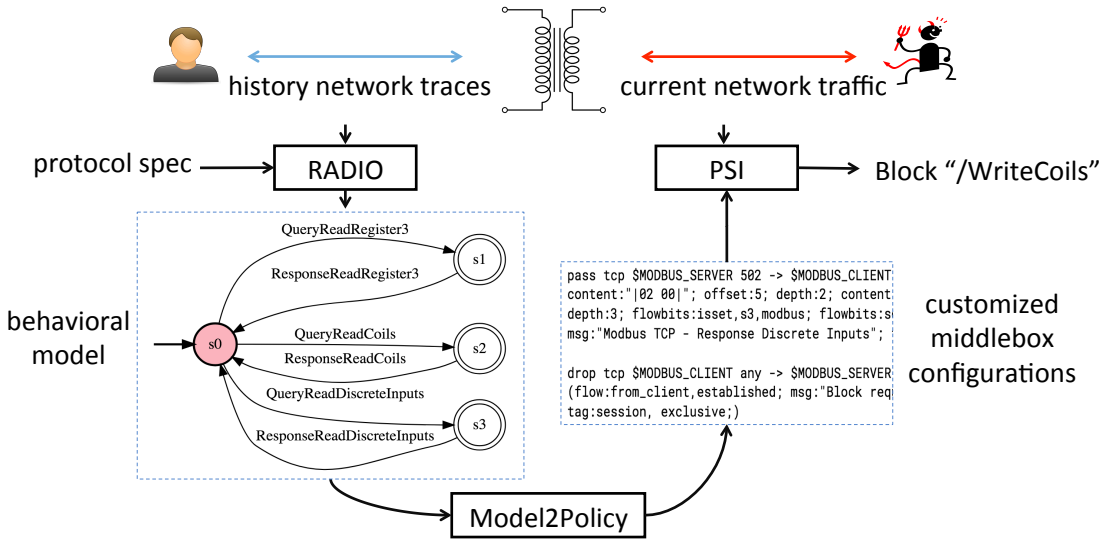

Figure 7.1: The workflow of integrating RADIO and PSI.

7.1 Integrating RADIO and PSI: IoT autopatch

The first usecase is automatically patching the IoT devices. As we know, one key issue for IoT devices is that host-based patching is hard. This is because of several reasons. First, the IoT devices are running firmware that is harder to patch than computer software. The longevity of IoT devices means that the IoT device may be used long after its vendor cease operations. Also, the bad IT practice in IoT devices is causing the patches to be error-prone. The patching process is easy to misconfigure.

To address this issue, our idea is to combine RADIO and PSI. RADIO can learn the benign behavioral model for IoT devices. PSI can enforce the security policies that only allows the

benign behaviors of IoT devices. Combining RADIO and PSI, the system can patch the IoT devices on the network by allowing all the benign behaviors and blocking anything else.

Figure 7.1 presents the high-level workflow of integrating RADIO and PSI. In the first step, given the protocol specifications, RADIO learns the single-device behavioral model from the historical network traces. For example, the behavioral model in Figure 7.1 is a FSM containing the `QueryReadRegister`,`QueryReadCoils` and the `QueryReadDiscreteInputs` behaviors. Then, the *Model2Policy* module convert the behavioral model into customized middlebox configurations in PSI. Finally, PSI will deploy the customized middlebox on the current network traffic and block any traffic that mismatches the benign behavioral model. For example, the code injection attack to use `WriteCoils` will be dropped. In this way, the IoT device is patched and protected.

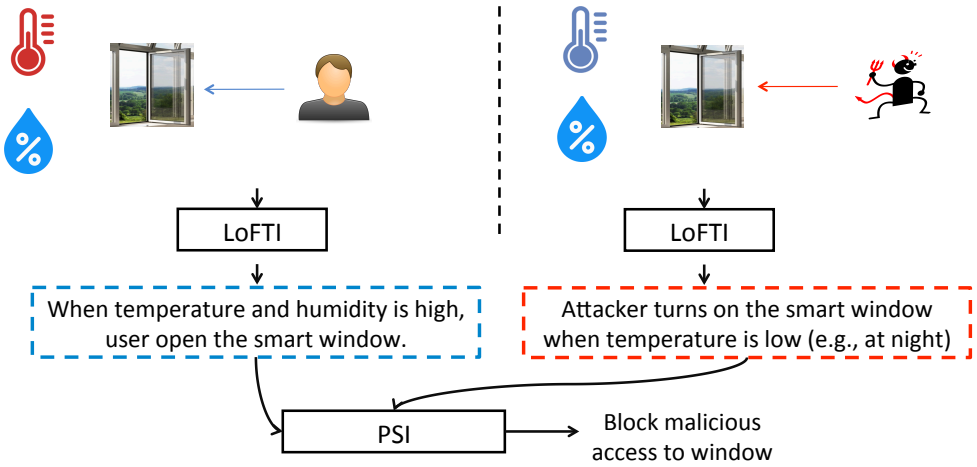

Figure 7.2: The workflow of integrating RADIO and PSI.

7.2 Integrating LoFTI and PSI: Context-based and dynamic policy enforcement for smart homes

The second usecase is the context-based and dynamic policy enforcement for smart homes. We know that context-based and dynamic policy enforcement are needed to protect the smart homes. For example, opening the window should be allowed or blocked based on the temperature and humidity (context). Also, the relation between the temperature and humidity and whether opening the window should be allowed or blocked may envolve over time (dynamic).

To achieve such context-based and dynamic policy enforcement, our idea is to use LoFTI to learn and update the context-based policy periodically. Then, we can use PSI to enforce such context-based policy.

Figure 7.2 presents the high-level workflow of integrating RADIO and LoFTI. In the first step, LoFTI will learn the context-based policy from multiple smart homes from the historical records of context and IoT access. The context-based policy is in the form of a machine learning prediction function F, with context and access as input and whether to allow or block the access

as output. For example, the function F will take in the temperature and humidity and decide that when the temperature and humidity is high (as a numeric relation), user can open the smart window. Then, in the runtime, LoFTI will monitor the context and access. Next, PSI will use the context and access in runtime and the context-based policy F to decide if an IoT access should be allow or blocked. For example, if an attacker is trying to open the window at night when temperature is low for breaking-in, the malicious access will be blocked.

However, there are two issues to realize this workflow:

- Supporting machine learning Policies: PSI only supports explicit human-input policies (i.e., FSM with human specified states and transitions), need to extend to support machine learning generated policies (i.e., $F(context, action) -> block/allow$) with model parameters.
- Granularity differences: PSI's policy is at a per-device granularity, while LoFTI is at a per-device-per-action granularity. We need to extend PSI to support a finer-grained policy.

To address these issues, we adopt two design points, as shown in Figure 7.3. First, we add the classifier as part of PSI's policy abstraction to support machine learning generated policies. The classifier acts as the function F, takes contexts and actions as input, and output whether an IoT access should be allowed or blocked. Then, we add a tree-like structure to support enforcement at per-action granularity. For example, if the classifier predicts that the action to *open* the smart window should be *blocked*, then the tree structure will output *DAGOpenBlock* as the enforcement to block the open-window action.

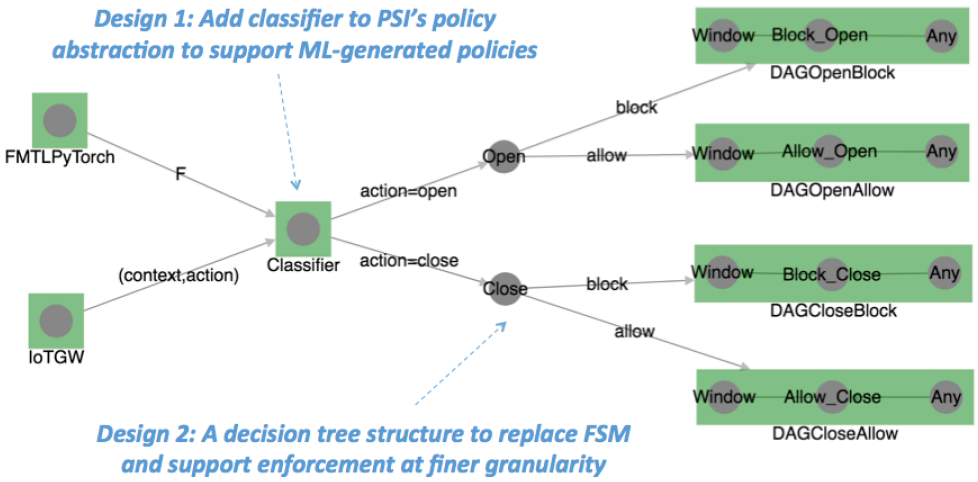

Figure 7.3: The design of integrating RADIO and PSI.

Next, we will explain how to modify PSI's policy abstraction to support LoFTI policies. The key is to replace the ψFSM structure of PSI with the classifier and the tree structure, to support ML-based policy at per-action granularity. The ψDAG structure remains the same to support context-based enforcement.

Chapter 8

Related Works

8.1 Related works for RADIO

Firmware based analysis: Firmware analysis approaches[49, 54, 55, 109] expose the vulnerabilities inside the device's firmware. However, firmware may not always be available to enterprises for analysis. Other attestation approaches[85] detect unauthorized modification of IoT firmware, but they are limited to detecting firmware modification attacks.

Anomaly detection approaches: The closest work is a rich body of previous anomaly detection approaches[42, 51, 53, 68, 76, 79, 88, 90, 95, 113, 119] that use statistical or behavioral anomalies to alert on attacks. As shown in Table 8.1, Previous network anomaly detection approaches include (1) classification[42, 88, 113], (2) clustering & outlier detection[68, 76, 95], and (3) FSM-based detection [51, 53, 79, 119].

Flow-statistics-based approaches including Kitsune[88] and Botminer[68] are useful for detecting data exfiltration (by flow data bytes), botnets (by connection statistics) or scans, but cannot detect more stealthy IoT attacks including code injection or authentication bypass. Classification-based approaches [42, 90, 113] need labeled attack and benign samples to train the classifier and cannot apply to unlabeled samples. Clustering-based approaches can apply to unlabeled samples, but none of the previous work provide a precise model to capture IoT behaviors. Nazca[76] focuses on graphic representation to detect malware distribution, and Ho [72] focuses on email representation to detect spearphishing. Of all the clustering approaches we mentioned, we found two of them that can be adapted to detect IoT attacks. We can use Ngram method, as described by Antonakakis [39], to calculate similarity between message series followed by the Xmeans method to cluster them. Likewise, we can use editing distance method, as described in Perdisci [95], to calculate similarity between message series and then use single hierarchical clustering to cluster them. Such approaches, however, yield high FP and FN rates because it is hard to exhaustively cover all the benign message series. FSM-based approaches including Prospex[53] and Suricata[35], can be adapted to detect IoT-specific anomalies. However, they cannot differentiate attack samples from benign ones in an unlabeled sample set in the FSM inference process and may generate states and transitions representing attack behaviors. As a result, they result in high FNs.

Other IoT security work: Several other related works [64, 87] also studied the IoT security

Approach	Class	Method	Input	Target	Precise Model	Pollution
Bartos [42]	Classification	SVM	Proxy logs	Malware	✗	✗
Webwitness[90]	Classification	Statistic classifier	Proxy logs	Malware downloads	✗	✗
Soska [113]	Classification	Ensemble of decision trees	Webpage	Infected website	✗	✗
Antonakakis [39]	Clustering	Ngram + Xmeans	DNS messages	Dga malware	✗	✓
Nazca[76]	Clustering	Graph clustering	packets	Malware distribution	✗	✓
Ho [72]	Outlier detection	Anomaly scoring	Emails	Spearphishing attacks	✗	✓
Botminer[68]	Clustering	X-means	Flow statistics	Botnet	✗	✓
Perdisci [95]	Clustering & Outlier	Hierarchical Clustering	HTTP packets	HTTP-based Malware	✗	✓
Cho [51]	FSM-based	L Star	Packets	Botnet	✓	✗
Mace[53]	FSM-based	L Star	Packets	Protocol anomalies	✓	✗
Prospex[53]	FSM-based	Prefix tree	Packets	Protocol anomalies	✓	✗
Suricata[35]	FSM-based	Handcrafted protocol FSM	Packets	Protocol anomalies	✓	✗
RADIO	FSM + Outlier	RPNI + RANSAC	ADU series	IoT anomalies	✓	✓

Table 8.1: Overview of the state-of-the-art approaches focusing on network anomaly detection.

problem. FlowFence[64] provides data protection for IoT applications. IoT Sentinel[87] tries to identify the device types in a smart home setting. Both of these complement our work by focusing on other aspects of IoT security.

8.2 Related works for LoFTI

Federated Multi-Task Learning: MOCHA[110] is a novel learning mechanism that combines federated learning and multi-task learning. We build LoFTI based on MOCHA[110].

Manually generating contextual IoT policies: A rich body of previous works are focusing on manually generating contextual policies for IoT devices [65, 70, 78]. He et. al.[70] provide a user study to identify the critical contextual information that should guide IoT access in smart homes. FlowFence[65] and ContexIoT[78] build user-prompt frameworks to prompt detailed contextual information to let users make better IoT access control decisions. FlowFence focuses on data flow and ContexIoT focuses on control flow. The issue with manually generating contextual policies is that it is easy to misconfigure and hard to cover all the contextual policies needed.

Sensitive information tracking: Another group of related works [43, 48, 58, 115] propose to prevent privacy leakage by tracking the sensitive information in mobile applications for IoT devices. However, this approach requires the source code of the mobile applications which is often not available. Also, since the tracking is based on mobile apps, it can only detect malicious/flawed mobile applications, and cannot handle malicious IoT access directly from the network (via weak authentication or backdoor).

Other IoT access control works: SmartAuth[116] provide an NLP-based approach to check the consistency between the IoT application description and IoT application source code. SmartAuth[116] cannot detect malicious access when there is no consistency issue. ESOs[102] is an integration platform that enables cross-platform enforcement for contextual IoT policies. This related work can be combined with LoFTI to address a broader scope of problems.

8.3 Related works for PSI

Firewalls and IDSes Middleboxes such as firewalls and IDS/IPS [94, 99] are the "workhorses" of network security mechanisms today. Unfortunately, these have well-documented concerns with respect to (a) performance (, 30% of administrator disable useful security features such as DPI and anti spoofing [26]); (b) misconfigurations (, [91]); (c) lack of expressiveness to tackle novel threats; and (d) undesirable tradeoffs between stronger security postures vs. user backlash [26]. By design, PSI addresses these pain points. PSI's vision shares conceptual similarity with classical work on distributed firewalls [77]. Other work addresses orthogonal problems related to scalability and alert correlation (, [101]).

SDN, NFV, and Security Prior work has aimed to take advantage of SDN and NFV to make network security enforcement more flexible. Ethante [47] uses a centralized controller controls switches at critical points to authorized traffic. FlowGuard[75] resolves interfering ACL policies from firewalls in SDN network. However, Ethane or FlowGuard does not support security policies beyond ACLs and do not allow more advanced security policies captured by ψDAGs and ψFSMs. Flowtags [63] and Simple [97] are two other related works close to PSI. Flowtags provides interfaces between middleboxes and the SDN controller to enable the enforcement of "fixed" ψDAG. Simple provides simplified traffic steering over a set of statically deployed middleboxes. FlowTags or Simple supports neither dynamically changing ψDAGs (which are accommodated in PSI using ψFSMs) nor guarantees policy isolation.

FRESCO [106] implements detection and mitigation modules in the SDN controller. However, the controller becomes a critical bottleneck for scalability and requires reimplementing functionality that is commonly available in security middleboxes. OFX [112] and Kinetic [81] focuses on networks composed of switches. PBS [74] shares some our motivation in addressing security challenges induced by BYOD. In contrast to PSI, PBS: (a) only focuses on BYOD mobile apps; (b) involves a reactive controller; and (c) needs end-point instrumentation on Android. Other work focuses on exploiting SDN/NFV to provide elastic scaling of security functions [62]; such elastic scaling is orthogonal to the focus of PSI. That said, the PSI $\psi cluster$ can leverage these elastic scaling capabilities if needed.

Controller scaling, either via horizontal scaling (, [59, 83]) or proactive orchestration (e.g., [62, 63]) are active areas of work in SDN/NFV. PSI synthesizes and extends these efforts through a combination of proactive tag-based forwarding and elastic scaling.

Policy languages There has been renewed interest in programming abstractions for networks [81, 89]. Kinetic provides a domain specific language and an SDN controller to dynamically change OpenFlow switch actions [81]. However, Kinetic is constrained and cannot express richer policies that involve stateful middleboxes [69]. PGA [96] provides support for composing forwarding policies across aggregates and detecting conflicts. PSI provides a richer abstraction that subsumes these prior efforts.

Industry efforts Google's BeyondCorp initiative [37] focuses on authentication and user trust, and can conceivably be defeated by evading the authentication system (, a malicious insider). PSI's focus on traffic behavior is intended to be robust to insider threat and other attacks by avoiding a single point of trust. VMWare's NSX and microsegmentation tackles datacenter security by pushing firewalling functionality into hypervisors to tackle "east-west" traffic, which is not protected by perimeter firewalls [118]. However, the security mechanisms and abstractions

are restricted to simple firewalling rules. In contrast, PSI targets a much richer set of policies that can involve multiple security middleboxes and does not rely on every device to run atop a hypervisor. Finally, the vision of SDN/NFV is gaining a lot of traction in industry; , Cisco's Evolved services platform [8] describes a high-level architecture similar to PSI. Based on public documentation, however, it does not specifically tackle the kinds of security, policy, and scalability challenges we describe here. These trends further corroborate our arguments that the PSI architecture is viable, and likely the inevitable culmination in response to today's security woes.

Chapter 9

Contributions, Limitations and Future Works

In this chapter, we first summarise our contributions to designing the fine-grained network detection and prevention system. Then, we discuss the limitations of the system. We conclude the thesis with potential future works to further secure the IoT devices.

9.1 Summary of Contributions

9.1.1 Current security ecosystem for IoT devices

Today's security ecosystem, which relies on a combination of static perimeter network defenses (e.g., firewalls and intrusion detection/prevention systems), the ubiquitous use of end-host based defenses (e.g, antivirus), and software patches from vendors (e.g., Patch Tuesday), is fundamentally ill-equipped to handle IoT deployments. Specifically, the scale, heterogeneity, use cases, and device and vendor constraints of IoT means that traditional approaches fall short along three key dimensions.

The first dimension is learning signatures and anomalous behaviors for a single IoT device. The *diversity* of IoT devices and vendors inevitably means that traditional approaches of discovering attack signatures (e.g., honeypots) will be insufficient and/or non-scalable. Also, traditional flow-statistic-based anomaly detection cannot be directly applied to capture the application-level behaviors of IoT devices (e.g., recording behavior of a camera).

The second dimension is learning the interactions of IoT devices. IoT devices can interact with other devices via explicit channels (e.g. a single app may use multiple IoT devices) or implicitly by affecting the *physical* world around them (e.g., an IoT light bulb may trigger an IoT light sensor). Thus, IoT devices can affect both applications that use them explicitly as well as applications with implicit or indirect *cross-device dependencies*. The result is that the security for an IoT deployment is likely to be complex and dynamic since they depend on both physical (e.g. environmental parameters) and computational (e.g., the state of other related devices) contexts.

The third dimension is the enforcement mechanisms. Since IoT devices operate deep inside the network, traditional perimeter defenses are ineffective. At the same time, IoT devices

typically do not run full-fledged operating systems, require low-power consumption and are resource-constrained. Moreover, the longevity of these devices means that vulnerable devices (e.g., default passwords, unpatched bugs) remain deployed long after vendors cease to produce or support them. Thus, traditional host- or device-centric mechanisms (e.g., antivirus, patches) are impractical to expect in an IoT world. Finally, given that the environment and device behaviors can change rapidly, we need to rapidly reassess and update the system's security posture. Unfortunately, today's security enforcement schemes stem from a static mindset and cannot handle such dynamics.

Rather than chase the hopeless goal of IoT devices that are secure-by-construction, we need pragmatic "bolt-on" solutions for securing IoT that acknowledge the realities of the marketplace (e.g., existing devices, poor software practices, patch unavailability) and the practical challenges associated with IoT (e.g., resource or software constraints). Unlike traditional IT ecosystems where host-based detection and prevention are prevalent, we believe that the device and ecosystem limitations of IoT will need the network to (re)emerge as the key vantage point for enforcing security policies.

9.1.2 RADIO Contributions

In this thesis, we present RADIO (Robust behavioral Anomaly Detection for enterprise IoT devices), which detects behavior anomalies based on benign behavior models learned from unlabeled, potentially polluted network traffic. To provide precise abstraction for modeling the network behaviors of IoT devices, we identified three key network characteristics of IoT behaviors, and carefully define the states and transitions of the FSM model to abstract these key characteristics. Then to learn the FSM model from IoT traffic, the main challenge is to learn IoT behavior model from unlabeled network traces. There are two issues. The first issue is that it is hard to infer the FSM states with unlabeled behavior samples. Existing FSM inference algorithms either rely on both positive and negative samples [57, 93] or rely on extra structural information (e.g., while loop structure of program) [103] to infer the FSM states. However, considering IoT traffic, it is hard to obtain labeled positive samples (attack samples) or extra structural information (no program structure). To address this issue, RADIO provides a modified FSM inference algorithm to leverage the internal correlation inside the IoT traffic to infer the FSM states. The second issue is that the samples can be polluted, and malicious behaviors will be included in the benign behavior models learned. To handle pollution, RADIO novelly integrates an outlier detection mechanism with the FSM inference algorithm to generate multiple candidate models, and leverages IoT heuristics to filter out polluted models and selects the best benign model.

9.1.3 LoFTI Contributions

In LoFTI, we explore the design space in terms of feature extraction, model building and dataset construction and address the challenges above. We generalize 6 types of key features to capture the contextual IoT access patterns and demonstrate their effectiveness in distinguishing benign/malicious IoT access. We build a simple SVM-based model with extra temporal structures to achieve high detection accuracy and low communication/computation cost. We provide a specialized data augmentation mechanism to balance the contextual access historical record dataset.

We show that LoFTI framework can achieve high accuracy (FNR is 0.024 and FPR is 0.086) while addressing user's privacy concerns by guaranteeing that raw IoT data will be stored locally within each smart home.

9.1.4 PSI Contributions

We make several contributions in designing PSI. We design a PSI policy abstraction that can express agile and contextual traffic processing. This allows us to express rich multi-stage security-relevant processing mapped to a security-relevant state for each device; , a host in normal state is subject to simple IDS-followed-by-firewall but in "suspicious" state may be subject to additional on-demand exfiltration detection modules. We also provide mechanisms to incorporate legacy policies that need to be applied to a group of devices.

We provide a scalable and responsive control plane. Naively applying SDN/NFV mechanisms in security context is problematic and can introduce new avenues for DoS attacks [108]. We develop a scalable orchestration platform by synthesizing three key ideas: (a) proactive forwarding schemes based on logical tags that do not need to involve the controller; (b) effective techniques for horizontally scaling the controller infrastructure; and (c) prefetching future enforcement states to improve responsiveness.

9.2 Limitations

In this part, we discuss the limitations of the fine-grained detection and prevention system for IoT devices.

Need for protocol specifications in RADIO: One concern, however, is there may be proprietary and/or non-standard protocols in use in some environments. In this case, we need manually crafted protocol specifications or need additional reverse engineering to understand these protocols. Fortunately, related works[56] have demonstrated that it is possible to automatically reverse engineering non-standard network protocols to solve this issue.

Need labels for learning IoT interactions in LoFTI: We designed a supervised learning mechanism to combine with federated multitask learning mechanism, which means labeled data are needed in LoFTI. However, labeling the attacks of physical hazards or privacy leakage requires non-trivial efforts. It is an open question if an unsupervised learning mechanism can be designed to combine with federated multitask learning mechanism. This is because it is unclear whether the optimization form of federated multitask learning is solvable or not in unsupervised setting. We hope that further works in machine learning area can help to address this issue.

Covert channels in PSI: As with any system, PSI might be vulnerable to covert channels used by attackers; , malware via encrypted cloud storage. We conjecture that dynamic, customized, and isolated enforcement that PSI offers, can be an enabler for detecting such advanced threats; , a specialized MITM only applied for suspected users or using advanced behavioral detection techniques that would otherwise result in high false positives.

9.3 Discussions and Future Works

In this part, we will present discussions on the three pieces of the system and present future works.

9.3.1 RADIO

Attack evasion: Any behavioral anomaly detection is subject to evasion attacks that follow legitimate behaviors and RADIO is no different. However, many of the core attack behaviors that grant the attackers higher privilege or sensitive information intrinsically violates the legitimate patterns and RADIO would be effective. It can be an interesting future work to measure how good RADIO is against attack evasion and how to improve RADIO's robustness on that frontier.
Non-enterprise IoT: We hypothesize that the insight on behavior being restricted is more broadly true for IoT deployments beyond enterprises. Some of the enterprise IoT devices (e.g. NAS, camera or smart speaker) also widely exist in smart home environment or industry IoT environment. An interesting direction for future work is to deploy RADIO in smart home environment and industry IoT environment and evaluate its effectiveness.

9.3.2 LoFTI

Beyond smart home: In this thesis, we validate LoFTI using smart home datasets. However, as a distributed learning framework to learn contextual security policies, we envision that LoFTI can be applied in other areas beyond smart home settings, such as industry control systems.
Diverse deployment: In this thesis, we assume that the deployment of IoT devices in the smart homes are similar or there is a mapping mechanism to mapping different IoT devices and their states (e.g., bedroom camera in home A can be mapped to bedroom camera in home B). In CASAS dataset, the deployment of IoT devices in the smart homes are similar as many of the smart homes are government supported homes for elder people. We will explore the mapping mechanism for diverse deployment in the future.
Model privacy: In this thesis, we provide the guarantee that the attacker is not able to see the raw data in the smart home. A strong attacker who can see the parameters of the model (e.g., via compromising the cloud backend or MITM attack) may be able to infer certain private information about the smart homes. However, this put a high bar on the attackers. We envision LoFTI may combine that other techniques such as *differential privacy* to solve this issue.
Grouping smart homes: In LoFTI, we assign a learning task to each smart home. An interesting direction to explore is to see if we can group the smart homes to share learning task to further improve the accuracy and reduce communication/computation cost.

9.3.3 PSI

Attacks against PSI: Like other SDN-based systems, PSI's control plane is a valid target for attackers. For instance, a commonly perceived attack against centralized control is the control channel or switch rule space exhaustion [107]. Note, however, that by using proactive tag-based

orchestration PSI is resilient to such attacks. The use of virtual appliances also creates the potential for resource exhaustion attacks [117]. Note, however, that this problem is not unique to PSI. In fact, NFV offers new opportunities to *elastically scale* deployments depending on the offered load, unlike traditional hardware, to mitigate such attacks [62, 71]. One interesting future work would be to explore mechanisms to detect and mitigate such attacks against the control plane.

9.4 Final Remark

While IoT devices are benefiting many aspects of our everyday lifes, they are emerging threats to enterprises and smart homes from the cyber security perspective. We argue that traditional security approaches are fundamentally at odds with the IoT ecosystem (e.g., perimeter defense, antivirus, patching) and inadequate to provide the single-device protection, IoT interactions protection, and context-based and agile enforcement needed for securing IoT devices. This thesis explores the design space of learning single-device behaviors and cross-device interactions, as well as providing context-based and agile enforcement to protect the IoT devices. We build RADIO for robust learning for single-device behaviors. We build LoFTI for distributed learning for complex IoT interactions. We also build PSI for context-based and agile enforcement. In evaluation, we show that RADIO is accurate and robust to pollution, LoFTI can achieve high accuracy and low computation/communication cost, and PSI overcomes topological constrains and provide better enforcement coverage. By integrating RADIO, LoFTI and PSI, we show that our fine-grained network detection and prevention system has the potential of enabling practical IoT security use cases, including automated patching for IoT devices and context-based and dynamic policy enforcement for smart homes. We believe our system can be combined with the trending edge-cloud infrastructure to enhance the security of IoT devices.

Bibliography

[1] Smart meters can be hacked to cut power bills. http://www.bbc.com/news/technology-29643276, . 1

[2] Fridge sends spam emails as attack hits smart gadgets. http://www.bbc.com/news/technology-25780908, . 1

[3] Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015. http://www.gartner.com/newsroom/id/2905717. 1

[4] Will giving the internet eyes and ears mean the end of privacy? http://www.theguardian.com/technology/2013/may/16/internet-of-things-privacy-google, . 1

[5] The Internet of Things Is Wildly Insecure - And Often Unpatchable. http://www.wired.com/2014/01/theres-no-good-way-to-patch-the-internet-of-things-and-thats-a-huge-problem/, . 1

[6] Hackers attack shipping and logistics firms using malware laden handheld scanners. http://www.securityweek.com/hackers-attack-shipping-and-logistics-firms-using-malware-laden-handheld-scanners. 1

[7] Casas datasets. http://casas.wsu.edu/datasets/. 5.4.1

[8] Cisco Evolved Services Platform. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/service-provider-strategy/brochure-c02-731348.html. 8.3

[9] Cve-2017-3881 how to mitigate cia vault 7 exploits on your cisco switches. http://info.stack8.com/blog/how-to-mitigate-cia-vault-7-exploits-cve-2017-3881-on-your-cisco-switches. 4.3.1

[10] Cve-2017-3881 detail. https://nvd.nist.gov/vuln/detail/CVE-2017-3881, . 1.1.1

[11] Cve-2017-6744 detail. https://nvd.nist.gov/vuln/detail/CVE-2017-6744, . 1.1.1

[12] 13 signs that bad guys are using dns exfiltration to steal your data. https://theworldsoldestintern.wordpress.com/2012/11/30/dns-exfiltration-udp-53-indicators-of-exfiltration-udp53ioe/.

6.3

[13] Amazon web service: Elastic load balancing. http://aws.amazon.com/elasticloadbalancing/. 6.2.2

[14] Watch a hacker access nest cameras and demand people subscribe to pewdiepie. https://www.vice.com/en_us/article/xwb8j7/watch-a-hacker-access-nest-cameras-and-demand-people-subscribe-to-pewdiepie, . 1.1.2, 5.4, 5.4.1

[15] Target attack shows danger of remotely accessible hvac systems. https://www.computerworld.com/article/2487452/target-attack-shows-danger-of-remotely-accessible-hvac-systems.html, . 1.1.2, 5.4, 5.4.1

[16] Have a smart lock? yeah, it can probably be hacked. https://www.cnet.com/news/have-a-smart-lock-yeah-it-can-probably-be-hacked/, . 1.1.2, 5.4, 5.4.1

[17] Researchers hack philips hue lights via a drone; iot worm could cause city blackout. https://www.computerworld.com/article/3139860/researchers-hack-philips-hue-lights-via-a-drone-iot-worm-could-cause-city-blackout.html, . 1.1.2, 5.4, 5.4.1

[18] Alexa has been eavesdropping on you this whole time. https://www.washingtonpost.com/technology/2019/05/06/alexa-has-been-eavesdropping-you-this-whole-time/?noredirect=on, . 5.4.1

[19] Sustainability hacks: Automatic window control. https://hackaday.com/2011/09/29/sustainability-hacks-automatic-window-control/, . 1.1.2, 5.4, 5.4.1

[20] Half baked iot stove could be used as a remote controlled arson device. https://hackaday.com/2017/04/20/half-baked-iot-stove-could-be-used-as-a-remote-controlled-arson-device/, . 1.1.2, 5.4, 5.4.1

[21] The ultimate nightmare: Researchers learn how to hack connected smart-home toilets. https://bgr.com/2014/06/12/smart-home-toilets-hacked/, . 1.1.2, 5.4, 5.4.1

[22] How cia mi5 hacked your smart tv to spy on you. https://www.zdnet.com/article/how-cia-mi5-hacked-your-smart-tv-to-spy-on-you/, . 1.1.2, 5.4, 5.4.1

[23] Hacking team hack. https://gist.github.com/Sjord/ac8dfff3a3ac3180c065f370f24b30a8, . 1.1.1, 1.1.1, 2.1

[24] malware-traffic-analysis.net. http://malware-traffic-analysis.net/. 6.3

[25] Mcafee. https://www.mcafee.com/us/index.html, . 2.1, 2.1.1

[26] McAfee Report Reveals Organizations Choose Network Performance Over Advanced Security Features. http://www.mcafee.com/us/about/news/2014/q4/20141028-01.aspx, . 8.3

[27] Server nic teaming to multiple switches. http://itknowledgeexchange.techtarget.com/network-engineering-journey/server-nic-teaming-to-multiple-switches/. 6.3

[28] Norton by symantec. https://us.norton.com/. 2.1, 2.1.1

[29] Nvd. https://nvd.nist.gov/. (document), 1.1, 1.1.1, 4.3.8

[30] Opendaylight,group based policy. https://wiki.opendaylight.org/view/Group_Based_Policy_(GBP). 6.1.1

[31] Cisco pix 500 series security appliances. http://www.cisco.com/c/en/us/products/security/pix-500-series-security-appliances/index.html. 6.1.1

[32] Diary of a rat. http://www.provision.ro/threat-management/data-security/content-aware-and-dlp/diary-of-a-rat-remote-access-tool. 6.3

[33] Sambacry, the seven year old samba vulnerability, is the next big threat. https://www.guardicore.com/2017/05/samba/. 4.3.1

[34] springbok. https://github.com/conix-security/springbok. 6.3

[35] Suricata ids/ips. https://suricata-ids.org/. 1.4, 3.2.1, 4.1.3, 4.3, 4.3.2, 8.1, 8

[36] Eavesdropping with amazon alexa. https://www.checkmarx.com/2018/04/25/eavesdropping-with-amazon-alexa/, 2018. 4.3.1

[37] Google beyondcorp. https://beyondcorp.com/, 2018. 8.3

[38] Iot security flaw leaves 496 million devices vulnerable at businesses: Report. https://crn.com/news/internet-of-things/300106806/iot-security-flaw-leaves-496-million-devices-vulnerable-at-businesses-report.html, 2018. 4.3.1

[39] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *USENIX security symposium*, volume 12, 2012. 1.4, 4.3, 4.3.2, 8.1, 8

[40] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *USENIX Security Symposium*, 2017. (document), 3.5, 3.3.1, 4.3.1

[41] Noah Apthorpe, Dillon Reisman, and Nick Feamster. A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic. *arXiv preprint arXiv:1705.06805*, 2017. 4.1.4, 4.1.4

[42] Karel Bartos, Michal Sofka, and Vojtech Franc. Optimized invariant representation of

network traffic for detecting unseen malware variants. In *USENIX Security Symposium*, pages 807–822, 2016. 2.1, 2.1.1, 3.2.1, 8.1, 8

[43] Iulia Bastys, Musard Balliu, and Andrei Sabelfeld. If this then what?: Controlling flows in iot apps. In *CCS*, 2018. 2.2, 5.1, 5.4.8, 8.2

[44] Alan W Biermann and Jerome A Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE transactions on Computers*, 100(6):592–597, 1972. 5

[45] Riccardo Bortolameotti, Thijs van Ede, Marco Caselli, Maarten H Everts, Pieter Hartel, Rick Hofstede, Willem Jonker, and Andreas Peter. Decanter: Detection of anomalous outbound http traffic by passive application fingerprinting. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 373–386. ACM, 2017. 2.1, 2.1.1, 3.2.1

[46] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014. 6.2.2

[47] Martín Casado et al. Ethane: Taking control of the enterprise. In *Proc. SIGCOMM*, 2007. 8.3

[48] Z Berkay Celik, Leonardo Babun, Amit K Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A Selcuk Uluagac. Sensitive information tracking in commodity iot. *Usenix Security*, 2018. 2.2, 5.1, 5.4.8, 8.2

[49] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards automated dynamic analysis for linux-based embedded firmware. In *NDSS*, 2016. 1.1.1, 8.1

[50] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *2010 IEEE Symposium on Security and Privacy*, pages 191–206. IEEE, 2010. 4.1.4, 4.1.4

[51] Chia Yuan Cho, Eui Chul Richard Shin, Dawn Song, et al. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 426–439. ACM, 2010. 8.1, 8

[52] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An open-source tool for symbolic model checking. In *Computer Aided Verification*, pages 359–364. Springer, 2002. 6.1.1

[53] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. Prospex: Protocol specification extraction. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 110–125. IEEE, 2009. 1.4, 4.3, 4.3.2, 8.1, 8

[54] Andrei Costin, Apostolis Zarras, and Aurélien Francillon. Automated dynamic firmware analysis at scale: a case study on embedded web interfaces. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 437–448. ACM, 2016. 1.1.1, 8.1

[55] Ang Cui, Michael Costello, and Salvatore J Stolfo. When firmware modifications attack: A case study of embedded exploitation. In *NDSS*, 2013. 1.1.1, 8.1

[56] Weidong Cui, Jayanthkumar Kannan, and Helen J Wang. Discoverer: Automatic protocol reverse engineering from network traces. In *USENIX Security Symposium*, pages 1–14, 2007. 2, 4.1.3, 9.2

[57] François Denis, Aurélien Lemay, and Alain Terlutte. Learning regular languages using rfsas. *Theoretical computer science*, 313(2):267–294, 2004. 3.4.1, 5, 9.1.2

[58] Wenbo Ding and Hongxin Hu. On the safety of iot device physical interaction control. In *CCS*, 2018. 8.2

[59] Advait Dixit, Fang Hao, Sarit Mukherjee, TV Lakshman, and Ramana Kompella. Towards an elastic distributed sdn controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 7–12. ACM, 2013. 6.2.2, 8.3

[60] Colin Dixon, Ratul Mahajan, Sharad Agarwal, AJ Brush, Bongshin Lee, Stefan Saroiu, and Paramvir Bahl. An operating system for the home. In *NSDI*, 2012. 5.1

[61] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972. 6

[62] Seyed K Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Flexible and elastic ddos defense using bohatei. In *USENIX Security Symposium*, 2015. 8.3, 9.3.3

[63] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags. In *Proc. NSDI*, 2014. 6.2.2, 8.3

[64] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. Flowfence: Practical data protection for emerging iot application frameworks. In *USENIX Security Symposium*, pages 531–548, 2016. 8.1

[65] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. Flowfence: Practical data protection for emerging iot application frameworks. In *Usenix Security*, 2016. 2.2.1, 5.1, 8.2

[66] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 4.2.1, 6, 8, 5.3

[67] E Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967. 4.2, 4.2.1

[68] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *USENIX security symposium*, volume 5, pages 139–154, 2008. 2.1, 2.1.1, 3.2.1, 4.1.1, 8.1, 8

[69] Mark Handley, Vern Paxson, and Christian Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *USENIX Security Symposium*, pages 115–131, 2001. 8.3

[70] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. Rethinking access control and authentication for the home internet of things (iot). In *Usenix Security*, 2018. (document), 2.2, 2.2.1, 5.5, 5.4.3, 8.2

[71] Victor Heorhiadi, Michael K. Reiter, and Vyas Sekar. New opportunities for load balancing in network-wide intrusion detection systems. In *Proc. CoNEXT*, 2012. 9.3.3

[72] Grant Ho, Aashish Sharma, Mobin Javed, Vern Paxson, and David Wagner. Detecting credential spearphishing in enterprise settings. 2017. 2.1, 2.1.1, 3.2.1, 8.1, 8

[73] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 5.2

[74] Sungmin Hong, Robert Baykov, Lei Xu, Srinath Nadimpalli, and Guofei Gu. Towards sdn-defined programmable byod (bring your own device) security. In *NDSS*, 2016. 8.3

[75] Hongxin Hu, Wonkyu Han, Gail-Joon Ahn, and Ziming Zhao. Flowguard: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 97–102. ACM, 2014. 8.3

[76] Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, Giovanni Vigna, Sung-Ju Lee, and Marco Mellia. Nazca: Detecting malware distribution in large-scale networks. In *NDSS*, volume 14, pages 23–26, 2014. 2.1, 2.1.1, 3.2.1, 8.1, 8

[77] Sotiris Ioannidis, Angelos D Keromytis, Steve M Bellovin, and Jonathan M Smith. Implementing a distributed firewall. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 190–199. ACM, 2000. 8.3

[78] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, Atul Prakash, and Shanghai JiaoTong Unviersity. Contexlot: Towards providing contextual integrity to appified iot platforms. In *NDSS*, 2017. 2.2, 2.2.1, 5.1, 8.2

[79] Jayanthkumar Kannan, Jaeyeon Jung, Vern Paxson, and Can Emre Koksal. Semi-automated discovery of application session structure. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 119–132. ACM, 2006. 8.1

[80] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *Proc. NSDI*, 2012. 6.3

[81] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russ Clark. Kinetic: Verifiable dynamic network control. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 59–72, Oakland, CA, May 2015. USENIX Association. ISBN 978-1-931971-218. URL `http://blogs.usenix.org/conference/nsdi15/technical-sessions/presentation/kim`. 6.1.1, 8.3

[82] Teemu Koponen, Keith Amidon, Peter Balland, Martin Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, Rajiv Ramanathan, Scott Shenker, Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wend-

landt, Alexander Yip, and Ronghua Zhang. Network virtualization in multi-tenant datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 203–216, Seattle, WA, April 2014. USENIX Association. ISBN 978-1-931971-09-6. URL `https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/koponen`. 6.2.2

[83] Teemu Koponen et al. Onix: A Distributed Control Platform for Large-scale Production Network. In *Proc. OSDI*, 2010. 8.3

[84] Anand Krishnamurthy, Shoban P Chandrabose, and Aaron Gember-Jacobson. Pratyaastha: an efficient elastic distributed sdn control plane. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 133–138. ACM, 2014. 6.2.2

[85] Yanlin Li, Jonathan M McCune, and Adrian Perrig. Viper: verifying the integrity of peripherals' firmware. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 3–16. ACM, 2011. 8.1

[86] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. Clickos and the art of network function virtualization. In *Proc. USENIX NSDI*, pages 459–473, 2014. 3.3.3

[87] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, Tommaso Frassetto, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. Iot sentinel demo: Automated device-type identification for security enforcement in iot. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 2511–2514. IEEE, 2017. 8.1

[88] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. 2018. 1.4, 4.3, 4.3.2, 8.1

[89] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software-defined networks. In *Proc. NSDI*, 2013. 8.3

[90] Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. Webwitness: Investigating, categorizing, and mitigating malware download paths. In *USENIX Security Symposium*, pages 1025–1040, 2015. 8.1, 8

[91] Timothy Nelson, Christopher Barratt, Daniel J Dougherty, Kathi Fisler, and Shriram Krishnamurthi. The margrave tool for firewall analysis. In *LISA*, 2010. 8.3

[92] Helen Nissenbaum. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford University Press, 2009. 5.1

[93] José Oncina and Pedro Garcia. Inferring regular languages in polynomial update time. 1992. 3.4.1, 4.2.1, 9.1.2

[94] Vern Paxson. Bro: A system for detecting network intruders in real-time. In *Computer Networks*, pages 2435–2463, 1999. 2.1, 2.1.1, 3.2.1, 6.1.1, 8.3

[95] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *NSDI*, volume 10, page 14, 2010. 1.4, 2.1, 2.1.1, 3.2.1, 4.3, 4.3.2, 8.1, 8

[96] Chaithan Prakash, Jeongkeun Lee, Yoshio Turner, Joon-Myung Kang, Aditya Akella, Sujata Banerjee, Charles Clark, Yadi Ma, Puneet Sharma, and Ying Zhang. Pga: Using

graphs to express and automatically reconcile network policies. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 29–42. ACM, 2015. 6.1.1, 8.3

[97] Zafar Qazi, Cheng Tu, Luis Chiang, Rui Miao, and Minlan Yu. SIMPLE-fying middlebox policy enforcement using sdn. In *Proc. SIGCOMM*, 2013. 8.3

[98] Markus N. Rabe, Peter Lammich, and Andrei Popescu. A shallow embedding of hyper-ctl*. *Archive of Formal Proofs*, April 2014. ISSN 2150-914x. `http://afp.sf.net/entries/HyperCTL.shtml`, Formal proof development. 6.1.1

[99] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999. 8.3

[100] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999. 2.1, 2.1.1, 3.2.1

[101] Sebastian Roschke, Feng Cheng, and Christoph Meinel. A flexible and efficient alert correlation platform for distributed ids. In *Network and System Security (NSS), 2010 4th international conference on*, pages 24–31. IEEE, 2010. 8.3

[102] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Situational access control in the internet of things. In *CCS*, 2018. 2.2, 2.2.1, 5.1, 8.2

[103] R Sekar, Mugdha Bendre, Dinakar Dhurjati, and Pradeep Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE, 2000. 3.4.1, 4.2, 9.1.2

[104] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K. Reiter, and Guangyu Shi. Design and Implementation of a Consolidated Middlebox Architecture. In *Proc. NSDI*, 2012. 3.2.3

[105] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284. IEEE, 2002. 6.3

[106] Seungwon Shin, Phillip Porras, Vinod Yegneswaran, Martin Fong, Guofei Gu, and Mabry Tyson. FRESCO: Modular composable security services for software-defined networks. In *Proc. NDSS*, 2013. 8.3

[107] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 413–424. ACM, 2013. 3.4.3, 9.3.3

[108] Seungwon Shin, Yongjoo Song, Taekyung Lee, Sangho Lee, Jaewoong Chung, Phillip Porras, Vinod Yegneswaran, Jiseong Noh, and Brent Byunghoon Kang. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 78–89. ACM, 2014. 9.1.4

[109] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Firmalice-automatic detection of authentication bypass vulnerabilities in binary

firmware. In *NDSS*, 2015. 1.1.1, 8.1

[110] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434, 2017. 3.3.2, 8.2

[111] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 305–316. IEEE, 2010. 3.4.1

[112] John Sonchack, Jonathan M Smith, Adam J Aviv, and Eric Keller. Enabling practical software-defined networking security applications with ofx. In *NDSS*, 2016. 8.3

[113] Kyle Soska and Nicolas Christin. Automatically detecting vulnerable websites before they turn malicious. In *USENIX Security Symposium*, pages 625–640, 2014. 2.1, 2.1.1, 3.2.1, 8.1, 8

[114] Robert Soulé, Shrutarshi Basu, Robert Kleinberg, Emin Gün Sirer, and Nate Foster. Managing the network with merlin. In *Proc. HotNets*, 2013. 6.1.1

[115] Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of ifttt recipes. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1501–1510. International World Wide Web Conferences Steering Committee, 2017. 5.4.8, 8.2

[116] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, XianZheng Guo, and Patrick Tague. Smartauth: User-centered authorization for the internet of things. In *USENIX Security Symposium*, 2017. 5.1, 8.2

[117] Matthias Vallentin, Robin Sommer, Jason Lee, Craig Leres, Vern Paxson, and Brian Tierney. The nids cluster: Scalable, stateful network intrusion detection on commodity hardware. In *Recent Advances in Intrusion Detection*, pages 107–126. Springer, 2007. 9.3.3

[118] Vmware. Next generation security with vmware nsx and palo alto networks vm-series. In *White Paper*, pages 1–25, 2013. 6.2.2, 8.3

[119] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. Homonit: Monitoring smart home apps from encrypted traffic. In *CCS*, 2018. 8.1

[120] Amy Zhao, Guha Balakrishnan, Fredo Durand, John V Guttag, and Adrian V Dalca. Data augmentation using learned transformations for one-shot medical image segmentation. In *CVPR*, 2019. 5.3