3-23-2020

On-device Security and Privacy Mechanisms for Resource-limited Devices: A Bottom-up Approach

Leonardo Babun
Florida International University, lbabu002@fiu.edu

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

ON-DEVICE SECURITY AND PRIVACY MECHANISMS FOR RESOURCE-LIMITED DEVICES: A BOTTOM-UP APPROACH

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL AND COMPUTER ENGINEERING

by

Leonardo Babun

2020

To: Dean John Volakis
College of Engineering and Computing

This dissertation, written by Leonardo Babun, and entitled On-device Security and Privacy Mechanisms for Resource-limited Devices: A Bottom-Up Approach, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Kemal Akkaya

Alexander Perez-Pons

Leonardo Bobadilla

A. Selcuk Uluagac, Major Professor

Date of Defense: March 23, 2020

The dissertation of Leonardo Babun is approved.

Dean John Volakis
College of Engineering and Computing

Andres G. Gil
Vice-President for Research and Economic Development
and Dean of University of Graduate School

Florida International University, 2020

DEDICATION

To my family.

ACKNOWLEDGMENTS

I would like to express my gratitude to the members of my dissertation committee for their insightful comments, encouragement, and generous support. In addition, I would like to express my deepest gratitude to my major professor, Prof. A. Selcuk Uluagac, for his priceless guidance, mentorship, motivation, patience, and immense support while completing this research and during all my doctoral graduate studies. His insights and words of encouragement have often inspired me and encouraged me to overcome all difficulties. I am deeply indebted to him for his tireless support. Also, I would like to thank my colleagues from the Cyber-Physical Systems Security Lab (CSL) for their encouragement, accompaniment, and collaboration through all these years. Finally, I would also like to acknowledge the support provided by the U.S. Department of Energy, U.S. National Science Foundation, the University Graduate School, and the Department of Electrical and Computer Engineering at Florida International University. This dissertation is mostly based upon the work supported by the U.S. Department of Energy under Award Number DE-OE0000779 and the U.S. National Science Foundation under Award Number NSF-1663051.

ABSTRACT OF THE DISSERTATION

ON-DEVICE SECURITY AND PRIVACY MECHANISMS FOR RESOURCE-LIMITED DEVICES: A BOTTOM-UP APPROACH

by

Leonardo Babun

Florida International University, 2020

Miami, Florida

Professor A. Selcuk Uluagac, Major Professor

This doctoral dissertation introduces novel mechanisms to provide on-device security and privacy for resource-limited smart devices and their applications. These mechanisms aim to cover five fundamental contributions in the emerging Cyber-Physical Systems (CPS), Internet of Things (IoT), and Industrial IoT (IIoT) fields. First, we present a host-based fingerprinting solution for device identification that is complementary to other security services like device authentication and access control. Then, we design a kernel- and user-level detection framework that aims to discover compromised resource-limited devices based on behavioral analysis. Further we apply dynamic analysis of smart devices applications to uncover security and privacy risks in real-time. Then, we describe a solution to enable digital forensics analysis on data extracted from interconnected resource-limited devices that form a smart environment. Finally, we offer to researchers from industry and academia a collection of benchmark solutions for the evaluation of the discussed security mechanisms on different smart domains. For each contribution, this dissertation comprises specific novel tools and techniques that can be applied either independently or combined to enable a broader security services for the CPS, IoT, and IIoT domains.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Emerging smart devices and their accompanying apps have changed the way we live and perform day-to-day activities. In fact, they represent an essential component of new engineering domains like the Internet of Things (IoT), the Cyber-Physical Systems (CPS), and the Industrial IoT (IIoT). In these domains, the Tnternet-enabled devices are interconnected and interact with each other, the physical world, and with the users to enable different novel services and applications. Some of these applications are directly related to the control and support of critical infrastructure like the smart grid [N. 14, KBKK12, Kou12] and the implementation of commodity environments like the smart home or the smart office [LL15, KSH16]. For instance, in a smart home scenario, motion sensors may activate the smart thermostat when the presence of the user is detected, or the smart lock may keep the door locked at night time.

Nonetheless, several recent research works have proven CPS, IoT, and IIoT devices (Figure 1.1) and applications to be vulnerable to cyber attacks [FJP16, Kus13, NM15]. Indeed, different mechanisms can be used to gain access to systems or leak sensitive information through these devices [BAU19, ABC+18, CMT+19]. For instance, from the previous smart home example, an attacker can have access to information being leaked from the motion sensor to know when the owners are not inside the house [CBS+18, BCMU19, SBAU19, BSAU18]. Similarly, in a smart grid, compromised devices can poison critical measurements from the network [BAU19]. In general, device vulnerabilities can be found at all levels of a device's/system's architecture. At the hardware level, manufacturers can use unauthorized components or hardware configurations that can impact the general performance of the devices or leave back-doors open for future attacks [BAU19]; at the kernel level, attackers can

(a) CPS Devices (b) IoT Devices (c) IIoT Devices

Figure 1.1: Sample CPS, IoT, and IIoT resource-limited devices that are vulnerable to cyber attacks.

tamper devices to change the configuration of the systems with malicious purposes or inject malicious code to change the behavior of the devices' operating systems (OS) and/or drivers; and finally, at the user level, apps can be used to leak sensitive data to malicious servers. Current cybersecurity solutions are not comprehensive and do not consider attacker models that cover all these threats and only focus on specific types of attacks. Also, most of the current solutions are intended to only specific device domains (e.g., CPS or IoT only and not both) and they fail to cover multi-domain scenarios. In this context, more comprehensive cybersecurity solutions that protect and/or detect malicious devices and applications from multiple smart domains are necessary.

Fingerprinting Resource-limited Devices

At the core of the Cyber-Physical Systems (CPSs) (e.g., smart grid, healthcare CPS, oil and water treatment plants, etc.), smart devices such as Remote Terminal Units (RTUs), Programmable Logic Controllers (PLCs), and Intelligent Electronic Devices (IEDs) are utilized to collect data from the infrastructure, provide two-way communications, and monitor the health of the operations in real time. However, these devices also present an opportunity for attackers to have access to sensitive information and the critical Cyber-Physical System (CPS) infrastructure. For instance, insiders can impersonate real CPS devices via spoofing to gain access to the systems, steal infor-

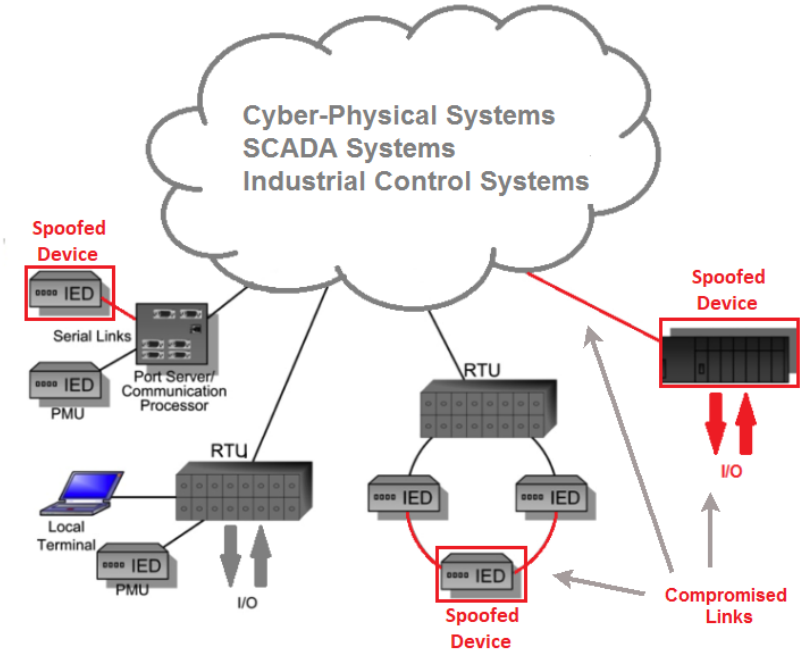


Figure 1.2: Example of CPS network with spoofed devices.

mation, make other devices in the network to behave erratically, or spread malware (Figure 1.2) [D. 12].

Protecting against such attacks stemming from spoofed devices can be very challenging, considering the device diversity in the CPS infrastructure. An attacker may use spoofed devices with software and hardware architectures very similar to real devices. Additional, these fake devices may be capable enough to perform the attacks while mimicking real CPS operations. In these scenarios, device fingerprinting techniques can be used to identify original devices and discriminate them from the impersonators. However, current fingerprinting solutions either require of extensive analysis of network packets or study the behavior of very dynamic network metrics [ZDLZ14, DBCM16, KBC05, LPBZ12, SPJ15, XZSH16, FSL+16]. As a result, in most cases these solutions introduce significant overhead to devices and systems, putting the execution of critical time sensitive CPS tasks at risk.

Detection of Compromised Resource-limited Devices

Critical infrastructure networks such as utility, production, and distribution systems are pillars of any nation and economy. They depend on intelligent and advanced Cyber-Physical Systems (CPS) to guarantee the efficient and reliable delivery of the data generated within these networks. These vital delivery systems have recently been going through a massive effort to modernize their CPS infrastructure. For instance, in the specific case of the power grid, a substantial effort has already been made to modernize the traditional decade-old grid to the next generation of technology (i.e., *smart grid*). The core concept of the smart grid relies on the integration of the underlying electrical distribution with two-way communications capabilities between the smart CPS devices in the grid. The uses of CPS devices in the grid allows new functionalities and state-of-the-art computing systems for the smart grid infrastructure over the traditional power grid [Y. 12]. Nonetheless, new security concerns stem from the use of CPS devices by the modern power grid.

Some of these concerns are related to:

- *Increased number of devices:* thousands of new two-way communications-enabled devices embedded into the smart grid represent new entry points that can be exploited by potential attackers.
- *High interconnectivity:* the smart grid interconnects a variety of different heterogeneous networks (i.e., home area networks, WAN, neighborhood area networks, etc.) and devices, increasing the potential risk of malware spread.
- *Higher attack payload:* the amount of sensitive information available in the smart grid makes it very attractive for cyber attackers.
- *Vulnerable and well-known computing technologies:* the smart grid adopts well-known commercial communication technologies as well as their numerous security threats (e.g., viruses, worms, trojans, etc.).

Indeed, with all its dependency upon device operations and communications, the CPS is highly vulnerable to any security risk stemming from devices. Especially, the use of compromised devices can wreak havoc on the smart grid's critical functionalities [D. 12, NM15] and can cause catastrophic consequences to the integrity of the smart grid data and operations. Recent examples like the Stuxnet and Sandworm worm attacks [Eur12, HSS17, Reu16] have proven that compromised devices represent a serious threat for the smart grid. Specifically, in the case of Stuxnet, the worm first targeted computers controlling Programmable Logic Controllers (PLCs)), to then change the configuration of the PLCs and cause the uranium centrifuges to behave erratically [Kus13]. The same way, in the case of Sandworm, the attack first targeted computing systems using the BlackEnergy Trojan [Kas16] to gain control over Remote Terminal Units (RTUs) and substation breakers to cause power blackouts [J. 16]. Due to these real attacks, understanding the behavior of the smart devices, particularly the compromised ones, has become more critical than ever. In fact, several government agencies focus their efforts to protect the critical infrastructure using behavioral-based approaches [Nat18].

Sensitive Information Leakage in IoT Applications

The Internet-of-Things (IoT) has quickly evolved to a new era where third-party developers are able to build applications. Several companies have already introduced useful programming platforms such as Apple HomeKit [Appa], OpenHab [Opec], and Smartthings [Sam] that provide developers APIs for controlling and interacting devices.

These IoT programming platforms allow developers to access various information to write custom automation and rules to control the connected devices and their applications. The methods and attributes inside the developed apps/programs are available to call in an application and gain access to the devices that are used to

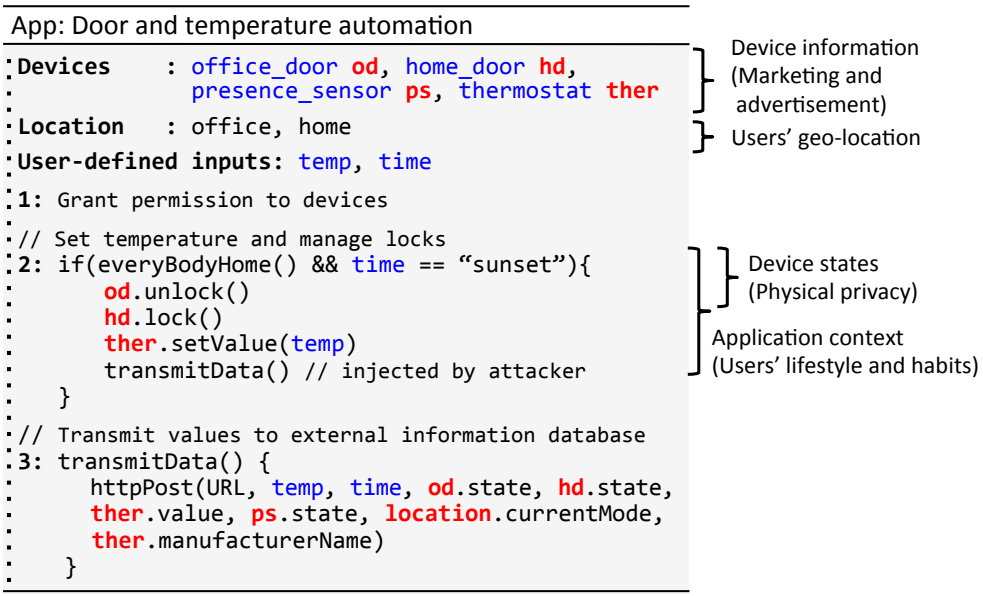


Figure 1.3: An example application deeming privacy risks. In addition to performing the expected task, this application leaks sensitive information to the web.

manage the applications' behavior. However, the programming platforms provide only coarse-grained controls for regulating access to information and provide no insight into what information is sensitive and how they are being used by the applications after the initial permissions are set by the users. This presents a number of unique privacy risks if a sensitive data leaks.

To illustrate how IoT applications may misuse sensitive values inside an app/program block, consider the application illustrated in Figure 1.3. The code sample demonstrates an IoT application that has been granted access to lock, unlock doors, and set the temperature when all family members are present at home after sunset. A POST request is embedded in the application logic to store the values in a remote system. This threat is vital when a malicious application profits from misusing sensitive values. Such values become high-profile privacy risks for users (see comments on the right of Figure 1.3). The aggregated data from a number of sensors and users can be used to threaten physical safety through monitoring door lock preferences and

the presence of individuals, to send sensor information to advertisers, and to acquire the profile of an individual's lifestyle and habits.

Privacy Concerns in IoT Apps

Users install IoT apps to manage and control smart devices such as the smart thermostat, door lock, and camera. These apps necessarily have access to sensitive information to implement their functionalities, communicate to external servers, and send notifications to users [CFP+19, Smag, Opeb, Appb]. The sensitive information is either obtained through APIs provided by an IoT platform (e.g., whether the door is unlocked (`door.unlocked`) or the lights are off in the kitchen (`kitchenLights.off`)), or they are simple texts defined by a developer or user at install time, for instance "the door is open", and "the kids returned to home".

Previous research has demonstrated that IoT apps may leak sensitive information to unauthorized parties [CBS+18, FPR+16]. Additionally, many IoT apps transmit data to remote servers without users permission for data visualization or profile user behaviors, such as their energy usage. However, users have no knowledge and control over what type of sensitive data apps access or who see these data.

With the rapidly growing IoT devices and apps, users are oblivious to the privacy risks that IoT apps pose. Existing IoT techniques focus on either analyzing the app's context to extract permissions of IoT apps [JCW+17, TZL+17] or use static analysis to find sensitive data flows [CBS+18]. These approaches (albeit useful) have limitations in over-approximating data-leaks, leading to false positives. Dynamic methods isolate sensitive data within sandboxes, which requires intensive developer effort [FPR+16]. Additionally, none of the solutions consider sensitive data leaks of untainted *strings* defined by a user or developer and contains, which we have found that out of 540 analyzed IoT apps, 64% of them leak through these strings. Lastly, the systems do not consider users' privacy preferences. There exist tools in other domains such as

mobile phones, [PCD+18, ARF+14, EGH+14, GTGZ14, PXY+13, QRZ+14] yet these cannot be applied to analyze privacy risks of IoT app source code as IoT apps possess a few unique challenges in terms of programming languages and structures.

Digital Forensics Analysis of IoT Data

The Internet of Things (IoT) has quickly evolved as a network of Internet-enabled physical devices. The IoT devices communicate with each other and interact with the users' day-to-day activities through sensors. These capabilities enable the concept of the *smart setting* (i.e., smart environments). Such an environment improves the quality of the life of the people while handling a new set of data previously untapped [NSG+14]. In general, the smart devices sense the users' activities to change the general state of the surroundings based on (1) what the users do, (2) the smart environment setup policies, and (3) the state of the devices. The interaction between devices and users in this settings generates data with tremendous forensic value [ST17]. For instance, in a smart office setup containing motion sensors and smoke detectors, the state of the motion sensors may reveal the presence of individuals at unauthorized hours. Also, the data extracted from the smoke detectors may provide insights about the exact location of sudden spikes in temperature values or the presence of smoke right before a fire incident, which would be very valuable for insurance claims. In fact, insurance companies have started giving incentives to customers if they install measures like smart lighting, smart energy management systems, or smart fire and water monitoring devices [Con, MIT].

Nonetheless, *current IoT programming platforms do not provide any means for forensic analysis.* Indeed, the limitation of available computing resources in the majority of the smart devices [BAU17, ABC+18] and the distinctive cloud-based architecture of IoT make it very challenging to store data inside the devices for forensic purposes. Additionally, the most popular IoT platforms (e.g., Samsung SmartThings,

openHab, Apple HomeKit) do not provide the mechanisms to access and indefinitely store IoT data in the cloud [CDS16]. Previous works have used logging techniques to acquire data from smart apps and devices [SKK+18, AYSM17] to implement, for instance, IoT data provenance analysis [WHBG18]. However, these works either (1) do not specifically focus on utilizing the acquired IoT data to implement forensic solutions, (2) assume trusted devices which may be unrealistic nowadays [Smae, Mal17], (3) or do not directly consider forensically-relevant activities and behavior from users and smart apps in their threat models.

1.1 Research Purposes

This doctoral dissertation introduces novel mechanisms to provide on-device security and privacy for resource-limited smart devices and their applications in the emerging CPS, IoT, and IIoT domains. These mechanisms aim to cover five fundamental contributions: (1) a host-based solution for device fingerprinting and identification that is complementary to other security services like device authentication and access control; (2) a kernel- and user-level compromised device detection process; (3) dynamic analysis of smart devices' applications; (4) a solution to enable digital forensics analysis of data extracted from resource-limited devices; and (5) on-device security benchmarks for the evaluation of the described security mechanisms on different smart domains (Figure 1.4). For each contribution, this dissertation comprises specific novel tools and techniques that can be applied either independently or combined to enable a broader drvices for the CPS, IoT, and IIoT.

1.2 Research Problem

The research problem has five main components:

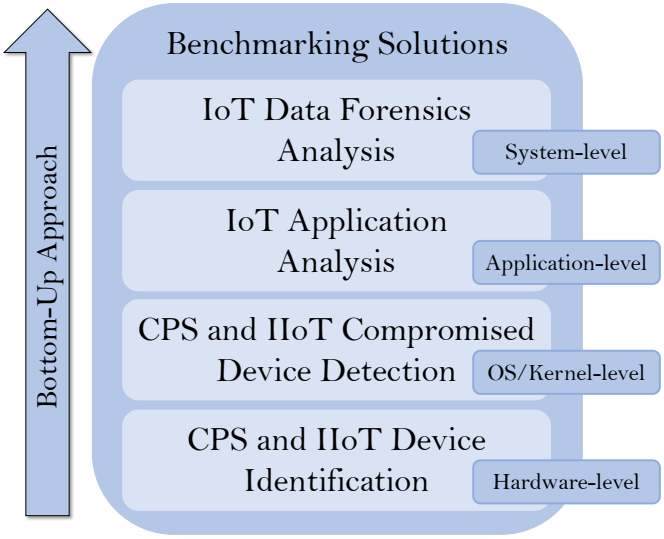


Figure 1.4: This dissertation introduces a bottom-up approach to provide security and privacy mechanisms to protect resource-limited devices and their controlling applications.

1. *Device Identification:* IoT and other resource-limited devices are prone to be spoofed by attackers to steal their identity. Also, fake devices can be used to gain unauthorized access to systems. In these scenarios, the precise identification and classification of resource-limited devices used across different engineering domains (e.g., CPS, IoT, and IIoT) is fundamental to support other security services like authentication and access control.

2. *Compromised Device Detection:* Resource-limited devices from critical infrastructures like the smart grid can be compromised causing measurement poisoning and information leakage. The study of the different tools and techniques that can be utilized for the detection of vulnerabilities and threats affecting smart resource-limited devices at different architectural levels is an open research problem. We evaluate our security approach with a use case that detects compromised CPS devices in the smart grid.

3. *Dynamic-Runtime Analysis of Applications:* Thousands of new apps are being utilized to control and manage resource-limited devices from different domains (e.g., critical infrastructure, commodity). These applications may contain malicious code to steal sensitive information from users and systems and compromise their privacy and integrity, respectively. These applications' source code can be analyzed statically, but the static tools can not protect against malicious activities happening at run-time. To solve this research problem, researchers focus on the design and application of algorithms, tools, and techniques to enable effective analysis of resource-limited apps without affecting the general performance of the devices.

4. *Digital Forensics Analysis:* Smart devices and sensors may be used in a cooperative way to create smart environments. In these settings, ample data is generated as a result of the interactions between devices and the users' daily activities. This data may contain valuable forensic information about events and actions occurring inside the smart environment. Nonetheless, current smart app programming platforms do not provide any digital forensics capability to identify, trace, store, and analyze the data produced in these settings. External solutions are necessary to enable digital forensics analysis on the resource-limited devices' data, so malicious activities from tampered devices and apps can be detected.

5. *Security Benchmark Solutions:* Security and privacy solutions require specific benchmarks to perform effective evaluation. The design of benchmarking solutions that can be utilized across different domains is an open research problem.

In general, effective mechanisms for on-device security and privacy must guarantee the safeguard of the devices at different levels: (1) hardware, (2) operating system, (3) application, and (4) smart environment. Such protection must be provided without

compromising the general performance of the resource-limited device and with minimal overhead to the systems. Additionally, the security solutions must be scalable to guarantee effectiveness against current and future cyber threats affecting the devices. Finally, since attacker models applied to different domains may differ substantially, specific security benchmark solutions may be necessary to evaluate the effectiveness of the security and privacy-preserving mechanisms here discussed.

1.3 Significance of the Study

Nowadays, resource-limited smart devices are integrated into multiple engineering domains (i.e., CPS, IoT, and IIoT). These devices are capable of extracting, handling, and processing data from critical infrastructures that, if compromised, can reveal sensitive information from users and systems. Thus, securing the devices against cyber attacks may represent the first step towards the protection of the critical systems and users. In this thesis, we aim for the safeguard of resource-limited smart devices by proposing effective security and privacy-preserving mechanisms that focus on five main cybersecurity areas: (1) the proper identification and classification of devices through their software and hardware characteristics; (2) the detection of compromised devices by studying their behavior; (3) the study of potential threats affecting the devices via malicious applications; (4) the implementation of solutions to enable digital forensics analysis on resource-limited device data; and finally, (5) the design of useful smart devices' benchmarking solutions for the evaluation of current and future cyber-solutions.

1.4 Organization of the Dissertation

The rest of this dissertation is organized as follows. In Chapter 2, we present background information to support the this dissertation. Then, in Chapter 3, we discuss the related work. In Chapter 4, a novel identification framework for resource-limited CPS and IIoT devices is detailed. Then, Chapter 5 introduces another framework to identify compromised CPS and IIoT devices. Later, in Chapter 6, we investigate security and privacy concerns in applications used to control resource-limited smart devices. Here, we introduce a tool to analyze apps and uncover security and privacy risks to the users and the smart systems in real time. Further, Chapter 7 investigates the feasibility of performing forensics analysis on data extracted from smart environments to infer security risks. Finally, we conclude the dissertation and propose future research paths in Chapter 8.

CHAPTER 2

PRELIMINARIES

In this chapter, we present an introductory background information about the fundamental building blocks of this dissertation. More detailed information can be found in the related chapters.

2.1 Overview of Cyber-Physical Systems

Cyber-Physical Systems (CPSs) confirm the integration between virtual and physical processes. In this context, the physical domain refers to capabilities that act over physical objects. On the other hand, the virtual domain constitutes the set of software and embedded systems intended to guarantee two-way communications, monitor the realization of the physical processes, and provide control [GKGK16]. In general, one can characterize CPSs networks by using the following features:

- *Type of task performed*: depending on the specific application and their location inside the CPS architecture, the type of task performed by CPS devices may range from just a simple service generated by a local host to an essential component of a more complex and centralized process. In any case, *individual CPS processes are assumed to be simple, deterministic, and very specific actions* that support the entire system in a distributed topology [FSL+16].
- *Resource availability*: the total resource availability in CPS processes *depends on the type of device performing every particular task*. In general, we can group CPS devices into *resource-rich* and *resource-limited* devices [L. 17]. Resource-limited devices have simple hardware (e.g., single-core CPU and limited memory) and software architecture that allows for the execution of simple, specific tasks. On the other hand, resource-rich devices have more complex Operating System

(OS) architecture and run with multi-core CPUs and plenty of memory. These capabilities allow them to execute more complex processes inside the CPSs.

- *Timing properties*: as we mentioned before, one of the main goals of the cyber domain in CPS is the monitoring and control of physical processes, which is achieved through rigorous timing control mechanisms. In general, *temporal behavior of CPS is expected to be very precise, and should not change too much over time [FSL+16].*

2.2 CPS Device-class Identification

Traditionally, device-class classification has been performed by considering the branch, model, specific device metrics characteristics, and the activities the devices should perform in the network [U.S10, U.S18, RBAU19]. The host-based fingerprinting mechanism introduced in this dissertation provides a more secure approach to identify types of devices in the network that also considers (1) the device behavior at the OS or kernel level and (2) its performance at the hardware level. The main advantage of an approach that includes behavior and hardware performance into its analysis is that it allows for a more secure identification approach that does not depend on device characteristics or metrics that can be spoofed by savvy (or even naive) attackers.

For this dissertation, we evaluate the following features to define a specific *device class*:

- *Device metrics*: may include the device's branch and model. Additionally, the intended application of the device is considered as part of the device characteristics (e.g., routers and firewall may be divided into two different groups based on their different intended use).

- *Device behavior*: characterizes the device response to specific challenges at the OS and kernel levels. We study device's behavior based on the collection of system and function calls triggered while reacting to specific challenges.
- *Device performance*: characterizes the device response to specific challenges at the hardware level. We study the device's performance by evaluating the device's memory and CPU utilization as well as the execution time while reacting to specific challenges or stimulants.

2.3 System-level Smart Grid Substation Architecture

The National Institute of Standards and Technologies (NIST) defines the smart grid as a set of seven different interconnected domains [NIS14]. Specifically, two of these domains are responsible for the generation and transmission of electricity, while the other four provide business, operations, and customer support. Finally, at the center of the smart grid architecture, the distribution domain (i.e., smart grid substations) acts as a communication and control hub for the entire infrastructure, which makes it especially attractive to cyber attackers [Y. 12].

In Figure 2.1, we present a simplified version of the smart grid distribution domain architecture. Here, three main operation layers can be highlighted [JRSK17, ISTC13, YCW+14]:

- *Process Level*: permits the data acquisition and control at the lowest level of the smart grid substation architecture. The devices at the process level (i.e., merging units) extract state information from sensors, transducers, and actuators and deliver command controls from the upper layers.
- *Bay Level*: permits the two-way communication between the process level and the upper operation layers of the smart grid substations. Here, Industrial Eth-

ernet switches interconnect different control and protection EIDs to allow: (1) protection and control of the data exchanged between bay level and upper and lower layers and (2) protection of the data exchanged between devices located inside the bay level.

- *Station Level*: provides user interfaces and enable applications for engineering and control of the lower layers. Here we can highlight operations from the communication system, the time synchronization system, the substation data collection and control, and servers and workstations.

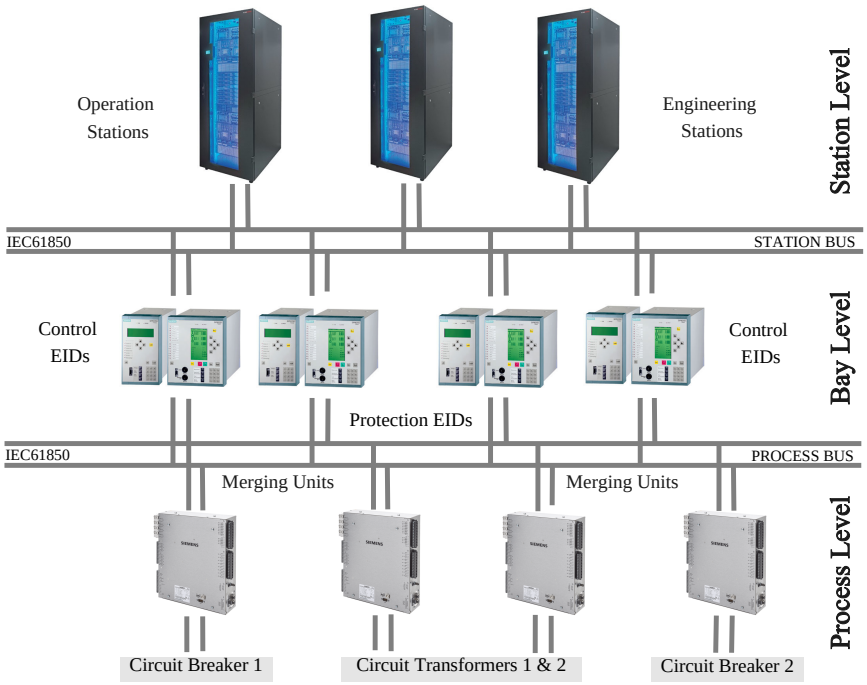


Figure 2.1: System-level interaction of smart grid substation devices. The two-way communications under protocol suite IEC61850 can be established both horizontally (between devices from the same level) and vertically (between devices from different levels).

The IEC61850 protocol suite enables the real-time communications between devices from different substations levels (vertical communications) and devices within the same level (horizontal communications) using Manufacturing Message Specification (MMS), Generic Object Oriented Substation Events (GOOSE), and Sampled

Measured Values (SMV) messages [C. 13, S. 10]. Specifically, this standard includes many underlying protocol stacks to support and monitor a variety of time-critical services. Indeed, IEC61850 supports real-time operations, abstracts services, and interoperability between devices used in energy automation [IEC03c, IEC03d].

2.3.1 Behavioral Analysis of Smart Grid Devices

For this dissertation, we focus on the behavioral characteristics of the smart grid substation devices while they communicate and perform either intra- or extra-level operations (i.e., horizontal and/or vertical communications) in the smart grid substation. We define *behavioral characteristics of devices at the system level* as the effect of the device's substations activities on the device's kernel. Let assume that there is a device O performing control operation at the Bay level. These operations can be represented as the set O_P where:

$$O_P = \{O_{P_0}, O_{P_1}, O_{P_2}, ...O_{P_N}\}, \tag{2.1}$$

then, we define the *system-level behavioral characteristic* of O as the function set BC due to the reflection of O_P at the device's kernel level [PX17, ZM17], that is:

$$BC = f_{kernel}(\{O_P\}), \tag{2.2}$$

In all the cases, we characterize the devices' kernel activity while the devices perform their regular smart grid substation operations. *Indeed, utilizing BC for the compromised device classification allows for a proper generalization of security solutions so they can also be successfully applied in other CPS domains outside the smart grid.*

2.3.2 Genuine Smart Grid Devices

We consider a smart grid device as *genuine* when no hardware nor software alteration or tampering has been performed on the device before, during, or after the manufacturing process. To further characterize and identify genuine devices, we define the parameter Index of Likeness (ILI). The ILI computes the similarity between individual operations O_i performed by a single device while executing a specific task T in different time intervals. Similar modeling approaches have been utilized in the literature to characterize CPS [LFPF18, CPS18a]. The universe of operations performed by a device to complete a task T at time instant $t = 0$ can be defined as:

$$T(t = 0) = \{\cup_{i=0}^{\infty} O_i : \exists O_i \in T\}, \tag{2.3}$$

and the value of ILI for different t can then be expressed as:

$$\rho_{ILI_t ILI_{t+i}} = \frac{\sum O_t O_{t+i} - n\overline{O_t O_{t+i}}}{n s_{O_t} s_{O_{t+i}}}, \tag{2.4}$$

where O_t represents the set of operations O_i performed by the device to complete the task T at the time instant t and O_{t+i} represents the set of operations performed to complete the same task T at the time instant $t+i$. In the same equation, n represents the cardinality of O and s_{O_t} and $s_{O_{t+i}}$ represent the standard deviation of O.

Based on our model, a genuine or *ground truth* smart grid device is expected to have a high value of ILI on average. This assumption has been supported in the literature by other research works that characterize Cyber-Physical Systems (CPS) devices (including smart grid devices) as highly deterministic systems [ZDLZ14]. In general, for processes running over time, ILI is expected to take values between 0 and 1: 0 is the result of entirely uncorrelated O_is and 1 is the result of remarkably high correlated O_is. For a more realistic analysis, our dissertation work considers some inherent level of randomness within the device operations. This assumption prevents

two Os from being completely identical even if one same device performs similar tasks repeatedly over time.

Ground-truth Smart Grid Devices

In the context of this dissertation, ground truth devices constitute particular cases of devices that are known as genuine. We assume full availability to ground-truth devices from every device class present in the smart grid. For instance, the compromised device detection framework introduced in Chapter 5 uses these ground-truth devices during its learning process. In the following, we define the practical values of ILI that allow for the characterization of ground-truth devices.

2.3.3 Compromised Smart Grid Devices

The smart grid (and other CPS) devices can be compromised either directly and indirectly. The direct method occurs in cases where the devices are compromised during any of the steps of the supply chain process [Int18, Asb09] or via insiders, by *directly* changing the configuration of the devices or their executing apps. Here, the attackers directly target the CPS devices without any other intermediate device. On the other hand, indirect methods are most commonly used and usually require initial access to the computing systems controlling the CPS devices in the network. Once the attacker gains access to those computers, they can change the configuration and behavior of the edge devices [Sym18, DBU20, DEB+19].

In this dissertation, we envision that the detection framework introduced in Chapter 5 can be utilized to detect compromised devices in both the supply chain and in the field. For that reason, our dissertation considers that genuine devices can be compromised during any stage of the manufacturing and application process. Specifically for our analysis, *we consider a compromised smart grid device as a genuine device*

with some malicious function installed on it. The malicious function can be due to compromised hardware or software component [K. 13, SS16b]. Also, the malicious function is expected to change the basic operations of the genuine device. In general, this function can be injected before, during, or after the device's manufacturing process. In Listing 2.1, we show realistic samples of a compromised device due to code injection. In this specific example, the malicious functions aim to (1) cause degradation on the device's resources and (2) save critical data on a file to be sent later to attackers.

Listing 2.1: Example of malicious code injected to compromised smart grid devices

```
1  void stress_mem()
2  {
3      srand(time(NULL));
4      long size = rand()%2147483647;
5      malloc(size);
6  }
7
8  void save_and_send_later (GooseSubscriber subscriber)
9  {
10     FILE *f = fopen("/root/baduser/data.dat", "a");
11         fprintf(f, "%" PRIu64 "\n", GooseSubscriber_getCriticalValue(subscriber));
12         fclose(f);
13 }
```

To further describe the compromised devices, we recall Equation 2.4. Here, the set of operations O is compromised with a malicious subset O_m executed to perform the malicious activity [KAJM16]. That is, for compromised devices, the malicious activity impacts the value of ILI by inserting malicious operations O_{m_i} to O. Such operations change the device's kernel behavior (Equation 2.2) so additional function or system calls are generated (see Listings 2.2 and 2.3). In general, the set O_m is expected to follow certain statistical distribution as detailed later in our adversary model. Finally, for compromised devices, Equation 2.4 takes the form:

$$\rho_{ILI_t ILI_{t+i}} = \frac{\sum O_t O_{m_t} O_{t+i} O_{m_{t+i}} - n\overline{O_t O_{m_t} O_{t+i} O_{m_{t+i}}}}{n s_{O_t} s_{O_{m_t}} s_{O_{t+i}} s_{O_{m_{t+i}}}}, \tag{2.5}$$

where the term O_{m_t} represents the malicious operations executed at time t and $O_{m_{t+i}}$ represents the malicious operations executed at time $t+i$.

Listing 2.2: System calls extracted from a genuine device

```
1  pthread_detach
2  malloc
3  malloc
4  free
5  free
6  signal
7  malloc
8  malloc
9  free
10 free
11 .
12 .
13 .
14 .
15 .
16 .
```

Listing 2.3: System calls extracted from a compromised device

```
1  pthread_detach
2  malloc
3  malloc
4  malloc
5  malloc
6  open
7  free
8  free
9  signal
10 malloc
11 malloc
12 malloc
13 malloc
14 open
15 free
16 free
```

2.3.4 Behavioral Analysis of Smart Grid Devices

Behavioral analysis of smart grid devices may utilizes changes in kernel's behavioral patterns to identify compromised devices. There are three main architectural challenges that this analysis needs to overcome:

1. Challenge 1: *The device class needs to be considered.* Different types of devices are expected to have different behavior; however, similar devices can also behave differently based on their specific tasks. Such ambiguity can lead to mistakenly identify genuine devices as compromised. For that reason, our detection framework in Chapter 5 incorporates (1) device resources (e.g., CPU and memory), (2) type of device, and (3) device task context into the analysis.
2. Challenge 2: *Device classes are very diverse.* Device class classification would represent an implementation challenge due to the high device diversity present in the smart grid [NIS14]. Additionally, after the initial classification, the list of devices would need to be checked periodically due to possible changes in network topology or new devices added to the network.
3. Challenge 3: *Smart grid devices operations are not fully deterministic.* OS operations possess some degree of randomness that reflects on the device operation

list O. During the detection process, the security framework introduced in Chapter 5 needs to discriminate between additional operations present in the call lists due to legitimate random processes and real malicious activities.

2.3.5 Classes of Smart Grid Devices

For this dissertation, we group the smart grid devices into different classes. Then, we expect that devices from different classes have different behavior. To correctly group the devices, we consider three main features that address the challenges above: device's computing resource availability, device's type, and device's task context.

Resource availability– we define two different types of devices based on the availability of their computing resources: *resource-rich* and *resource-limited* devices.

- *Resource-limited devices*: these devices have simple hardware and software architecture. They run with low-performance CPUs and have minimal memory capability. In general, the randomness of the resource-limited devices' kernel behavior highly depends on their software architecture [ZDLZ14]. Also, these devices are built to execute specific tasks inside the smart grid network. Some devices in this group are PLCs and RTUs.
- *Resource-rich devices*: these smart grid devices are close in configuration to full-capacity computers. They have a full Operating System (OS), faster multi-core processors, and significantly higher memory than the resource-limited devices. This type of devices executes specialized tasks inside the smart grid network. Some devices in this group are IEDs and PMUs.

Moreover, we group the devices depending on their specific application, brand, and model. For instance, PMUs from the same model and manufacturer can be grouped together while RTUs and PLCs are not considered of the same type. We

consider this classification because the devices from different classes have found to behave differently, even if they perform similar tasks.

Finally, the class-classification process of smart grid devices considers the device's task context. For our purposes, the task context involves the type of activity that the devices are performing and their specific logical location inside the smart grid network. That is, we consider that devices of the same type can behave differently if they are handling different types of data from different parts of the network.

In general, we consider that the devices perform similar and repetitive tasks over time [ZDLZ14]. Then, our detection framework (Chapter 5) takes advantage of this mode of operation to detect compromised devices based on changes in their expected behaviour.

2.3.6 Open-source Design Approach

The smart grid testbed used in this dissertation utilizes open-source *libiec*61850 libraries [M. 16] to exchange smart grid time-critical messages using the GOOSE format [C. 13] among different devices. The use of open-source software provides some additional design advantages: (1) our solution is more flexible, (2) the framework (Chapter 5) is more open to customizations which translate on being highly configurable, and finally, (3) our solution can be easily adapted to other open standards which increases interoperability. Therefore, to keep the detection framework from Chapter 5 open-source, we implement our solutions on Linux-based systems. This approach is considered realistic since a very high percentage of smart grid devices still utilize some variant of Unix-based OS [RB15]. We believe that, due to the open-sourced and configurable nature of our testbed, it constitutes an effective benchmark to test the performance of this and other security tools designed to protect the smart grid, that follows the behavioral analysis.

2.3.7 Extracting Operations from Smart Grid Devices

We utilize system and function call tracing techniques to extract the set of individual operations O from the devices. These operations are analyzed while the devices perform specific smart grid tasks T. We combine function and system call analysis, so the device's activity is detailed from both kernel and application-level, which increases the robustness of the detection framework introduced in Chapter 5. For attackers trying to exploit the calls to stealth their activities, the inconsistencies between system and function calls triggered by the same process can also indicate the presence of malicious activities. We take advantage of the open-sourced Unix-based nature of our testbed to effectively utilize library interposition and ptrace as system and function call tracing techniques, respectively.

Tracing system calls with library interposition. We use dynamic library interposition (LI) since this is a general-purpose system call tracing method that can be applied to most C-compiled programs [LBAU17]. LI takes advantage of the use of a shared object defined inside the runtime library. This object is in charge of fetching the system calls at the kernel level. At runtime, LI hooks this shared object to intercept the calls and take control of the applications' behavior.

Tracing function calls with ptrace. At the user level, we use Process Trace (i.e., ptrace), a popular Unix-based tool to trace function calls. Ptrace uses an external process that acts as a parent for the C compiled program that wants to be traced. Once the external process attaches to its child, the parent application has full control of every time the traced application makes a function call.

Finally, for cases where the smart grid devices do not use Unix-based OS (e.g., Real-Time Operating System (RTOS)), similar approaches are utilized to trace the system and function calls. Similar hooking techniques are possible to use because these other systems behave in similar ways as Linux since they are also POSIX-

compliant OS. In general, the tracing technique utilized for hooking into the system and function calls is a configurable feature that depends on every specific application [LBAU17].

2.4 Resource-limited App Taint Sources and Sinks

2.4.1 Sources of Sensitive Information in IoT Apps

IoT apps have access to data that can be highly private. We classify taint sources into five groups [CBS+18]. Taint sources include *sensor states* (e.g., door locked/unlocked), *device information* (e.g., manufacturer name and brand details), *user inputs* (e.g., a temperature value to set the heating point of a thermostat), and *location* (e.g., geo-location such as kitchen and living room, and geographical location such as zip number).

2.4.2 Functions to Leak Information in IoT Apps

IoT apps define specific APIs to send information out from the apps (i.e., taint sinks) [Smah, Opec]. These APIs define methods to send data out of the apps using two different mechanisms (i.e., Internet calls and messaging communications) [Smaf, Smaj, CBS+18].

Internet. IoT apps may act as web services or request HTTP calls to external services defined by the developer or, in very rare occasions, by the user. First, web services expose URLs to predefined endpoints that allow requests from IoT apps. For instance, web services request whether an appliance is on to model energy usage of an environment. Second, IoT apps use HTTP calls along with system information

for app functionality. For instance, an IoT app may send device type and model to obtain its detailed specifications.

Messaging. Messaging can be categorized into two types as well: SMS and Push notifications. These are used to inform an app user or other recipients about changes in an environment, for instance, notifying users when battery of a door lock is below a threshold.

Sink calls require two types of information to be executed, the recipient and body. Recipients define whom to send or request information, and the body contains the necessary information to make the call. In messages calls, the recipients are the phone numbers and are often defined by the user at install time [CBS+18], while the body can be specified either by the developer or the user. In Internet calls, the recipients are, in most cases, external web server addresses are defined by the developer (some previous works have reported Internet call's URLs also defined by the user [CBS+18]). On the other hand, the body contains information obtained from the app depending on its permissions (e.g., device location and device states).

2.5 IoT Application Context

Context represents the functionality of an IoT app that considers real-world needs of users and environments. We use two sources of information to infer IoT app context, app description block, and app permissions. First, IoT programming platforms often provide guidelines to regulate the way that the functionality of the apps presented to the users in IoT apps [Smag, Opeb, Appb]. This information is usually needed to inform the user about the functionality of the apps that helps the user to understand the app purpose and used at install at time. For instance, a smoke-alarm app description includes a string that says "This app sounds alarm when smoke is detected.", and definition for devices, "Which alarm?". Second, IoT apps require user permission

to implement the app logic. The user provides this information at install-time and, in some platforms, it can be updated later at the user's convenience. User-defined information includes devices, inputs for device functionality (e.g., min and max temperature in a smart thermostat control app) to trigger event handlers after reaching a specific temperature threshold, device location information to control specific set of devices (e.g., devices in a kitchen or living room), and contact information (e.g., phone number) to receive notifications from the apps.

2.6 Generic Smart Environment Architecture

The high-level architecture of a smart environment setup is presented in Figure 2.2. In this dissertation, we consider *Smart Environment* as the interconnection of Internet-enabled devices and sensors that are capable of communicating with each other and interacting with the users (the smart environment is expected to change its overall state based on the user's activity). These devices and sensors create a local network that is remotely managed via both *controller apps* (installed in the user's smartphone) and *smart apps* installed either in the cloud backend (i.e., cloud-based architecture) or a central device called a Hub (i.e., hub-based architecture) [ZMR17].

Through the controller app, the user can change the device settings and receive notifications from the system. Then, the system executes the smart environment's logic by considering inputs from the different devices' states, the sensors' readings, and the user-defined settings via the smart apps. Several IoT programming platforms (e.g., Samsung SmartThings, Apple's Homekit, or OpenHAB) share similar settings and architectural characteristics to deploy their systems.

While deploying the smart environment, users may download the smart apps from some of the available official repositories online [Smag, Opea], depending on the IoT platform used. Then, the apps are uploaded and installed into the cloud servers using

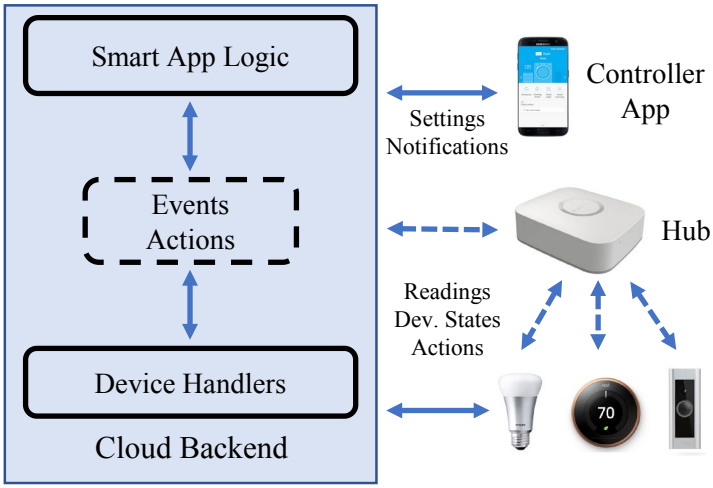


Figure 2.2: Sample smart environment setup and architecture. The use of a central controller (i.e., hub) is optional (dashed lines).

the user's smart platform credentials. However, for some of the open-source platforms like SmartThings, users can even develop their own apps or use source code that other developers make freely available online through third-party app repositories [Smac].

2.7 Smart App Structure

In general, IoT programming platforms [Smab,Opeb,Appc] define the means to access and handle the smart apps' data and to transmit it out of the applications. Through these processes, smart apps utilize sensor readings, user-defined inputs, and apps' events to execute the application logic either in the hub (i.e., hub-based setup) or in the cloud (i.e., cloud-based setup). Therefore, through the analysis of the smart applications' structure, one can identify and label smart app resources that could potentially contain relevant information for forensic purposes. In the following, we detail these forensically-relevant resources.

Events. These are used to respond to changes in the physical environment. At install time and depending on the app context, a smart application subscribes to a set of device events. Then, event handler methods are called every time such events occur.

Actions. After an event occurs, the smart app calls an action through the event handler to control the subscribed device. Actions specifically define how a smart environment setup responds to changes in device states.

User-defined Inputs. Smart applications often require inputs from users either to manage the application logic or to define device settings [AZZ+17]. These inputs, which are considered granted permissions, are defined at install time and set specific thresholds to trigger actions. Also, user inputs may define contact information used to send notifications to specific recipients.

Device Information. Device information is used to match smart app events and actions with specific devices. *Logging this information allows for tracking authorized or unauthorized changes (e.g., device replacement or tampering) in the smart environment.*

Time and Location. Successful forensic analysis of a smart environment requires not only the timing information of the events/actions but also their physical locations. In that way, such events can be directly attached to specific geo-location information of smart devices. Additionally, unique identifiers can be assigned to the different data logs.

Smart app programming platforms define APIs [Smaf, Oped] to transmit data outside the applications. We can categorize these APIs into two main groups:

Internet. IoT smart apps logic usually run as cloud-based services, which imposes a noticeable difference if compared to other domains (e.g., smartphone apps) where local solutions are preferred. Therefore, smart apps are designed to directly act as web services or make calls to external services defined by the developers. These calls

can make requests to smart apps via endpoints to obtain information such as device states, events, or even propose specific actions to control the smart solution.

Notifications. Smart applications can also define custom messages to send notifications to users. These notifications can be of the three different types: (1) push notification in the mobile app, (2) email, or (3) short message services (SMS). Such categorization allows the developer to create a dedicated notification system to alert the user when specific events occur.

CHAPTER 3

LITERATURE REVIEW

In this chapter, we present the related work to which we referred for each individual research work in this dissertation.

3.1 Identification of Resource-limited Devices

Device fingerprinting is an appealing research area that follows two main paths: device-class and device-host fingerprinting.

3.1.1 Device-class Fingerprinting

Device fingerprinting is an appealing research area that follows two main paths: device-class and device-host fingerprinting. In this work, we focus on the former. In [DBCM16] the feasibility of large-scale host fingerprinting via motion sensors is analyzed with 90% accuracy. Other works use microscopic deviations in clock skews to identify specific devices [KBC05,LPBZ12,SHS12], however, these approaches are vulnerable to simple countermeasures and require the analysis of several network packets for accurate results. Authors in [DBC14] use embedded acoustic devices (microphone and speakers) on smartphones to fingerprint individual devices. Even when they report accuracy values in the range of 98%, these results are only possible in close-range distances (0.1 meters). Similar directions were followed in [ZDLZ14, PBL15] where frequency responses of devices' speakers are used to identify individual devices. Other works report the use of the devices' behavior as a response to specific network packets (i.e., stimulant) [NHO12, BCKP08]. In spite of their positive results, these types of fingerprinting techniques also come with some limitations. For instance, the proposed approaches only apply for specific types of network protocols (e.g., transport layer

protocols like UDP, TCP, etc.) or they are vulnerable to the network dynamics such as WiFi channel characteristics, traffic delay, etc.

3.1.2 Device-host Fingerprinting

As for the identification of different classes of devices, in [KCB10], the authors propose a passive *blackbox* technique for determining the type of access point (AP) connected to a network based on its behavior. In [FSL+16], the authors use time as a baseline for device type fingerprinting. In this case, the proposed fingerprinting methods are mainly based on (1) the response time to network-based interactions (cross-layer fingerprinting) and (2) the response time to physical operations (physical fingerprinting). Although their results are promising, the first approach highly depends on configurable network attributes like the level of priority of TCP messages and ACK implementation. Further, the second proposed method also depends on the SCADA system configuration. In different works, passive device class fingerprinting are proposed by using the timing distributions between network packets as the fingerprinting features [URC+13, RUB15]. In similar approaches applied to domains other than CPS, researchers propose the analysis of network dynamics to infer IoT device classes [TDS+19, MMH+17, NMM+18, DJ17].

3.1.3 Behavioral-based Device Fingerprinting

Several security approaches make use of system and function call analysis to regulate and monitor the behavior of specific applications [Gar03, Blu13, LBAU17]. For instance, researchers have proposed the use of system and function call analysis for the design of intrusion detection systems (IDS) [JS99, FKF+03], the identification of operating system functions [HB99], sandboxing [KZ13], and the implementation of soft-

ware portable packages. Also, some works have demonstrated that similar approaches are suitable for the classification of behavioral anomalies [FKF+03, MKLP12]. Although these last works report high overhead introduced to the systems, other similar implementations are more lightweight [SBDB01].

3.2 Detecting Compromised Resource-limited Devices in CPS

3.2.1 Security Challenges of Cyber-physical Systems

There are several works studying security challenges in the the cyber-physical systems (CPS) [KHLF10, TAA15, SSG+16, WL13]. In general, cyber attacks against CPS are categorized into four different groups: denial of services (DoS) attacks, malicious data injection attacks, traffic analysis attacks, and high-level application attacks [WL13]. In [DS12], [X. 12], [AVJA15], and [BGK16], the authors provide several examples of DoS attacks impacting different parts of the smart grid architecture. Most of these attacks are executed from compromised hosts, servers, and devices inside the smart grid.

Examples of malicious data injection attacks can be found in [DS12,SS16a,KPJ16]. One compelling case is studied in [ZM16]. In this work, the authors analyze four different types of attacks in the state estimation process and examine the least-effort data injection attack to find the optimal attack vector.

In the case of traffic analysis attacks, authors in [SC11] describe how an attacker can monitor and intercept the frequency and timing of transmitted messages over the CPS network to deduce information and user's behavior. In [N. 14], high-level application attacks are described as the way an attacker can disrupt the essential functions of a power system (i.e., state estimation, power flow measurement, etc.).

3.2.2 Detection of Compromised Resource-limited Devices

The topic of compromised devices detection has not been extensively studied in the literature. In most cases, researchers focus on proposing anomaly detection mechanisms [Kos16] for different types of attacks in the smart grid [SGLL13, HPK+14, FHDK16, OEV+16], without particularizing on the attack sources (e.g., compromised devices). In a few cases, however, the particular behavior of the smart grid device is considered. In [YMA+16], the authors study the minimal number of compromised sensor that can be used to manipulate a given number of smart grid states effectively. Further, they consider the optimal Phasor Measurement Unit (PMU) placement to defend against this type of data integrity attacks.

Some other works focus on other CPS and industrial environments. In [MR16], the authors propose a vector-valued model-based cumulative sum procedure to identify compromised sensors in CPS. Even though this work achieves good results in simulation environments, its threat model only considers false data injection attacks. Also, no results are shown on the overhead introduced to the CPS devices, essential to consider suitable security applications for real-time critical infrastructures like the smart grid. In a different approach, the authors in [VBC14] apply attestation approaches to detect comprised devices in the CPS. In this work, however, they utilize stimulant-response mechanisms to detect compromised devices based on their specific reaction to controlled inputs, which can also be impractical for the smart grid and results can depend on several undesired networks' and physical channels' dynamics. Other works propose similar attestation approaches [CPS18b, LGGG07, VBC14] to detect attacks in CPS. However, these works focus on building models of the entire CPS network instead of focusing on individual devices, which impacts the overhead and the general performance of the proposed solutions. Finally, most of these works

apply to Wireless Sensor Networks (WSN) and are not directly applicable to the smart grid domain.

Intelligent, secure packaging, outbound beaconing, and better tracking systems are some of the countermeasures that are being proposed to prevent the introduction of compromised devices in the smart grid supply chain [D. 12, M. 13, Y. 16, U. 14]. However, skilled attackers could have remote access to legitimate devices (e.g., RTUs, PMUs, IEDs, etc.) outside the supply chain and create opportunities for tampering smart grid devices in the field.

3.2.3 Call Tracing Techniques for Security Applications

Function and system call tracing techniques constitute a powerful method for regulating and monitoring applications behaviour [Blu13], so they have been largely used in security applications [Gar03]. System and function call tracing techniques can be found in applications like intrusion detection and confinement [JS99], binary detection of OS functions [HB99], sandboxing [KZ13], and software portable packages [GE11]. Specifically, in [E. 01], the authors use system call tracing to implement intrusion detection systems (IDS). Also, in [FKF+03] and [MKLP12], the authors proposed anomaly detection mechanism based on information obtained from system calls behavior analysis. In these cases, the implementation of the security tools resulted too heavy in terms of system overhead. One similar application with improved system overhead can be found in [SBDB01]. In this case, the proposed solution is required to run continuously and serves the purpose of complementing antivirus software.

3.3 Resource-limited Device Application Analysis

3.3.1 Static Analysis of IoT Applications

Mirroring the expansion of IoT itself, there has been an increasing amount of recent research exploring IoT security. These works centered on the security of emerging IoT programming platforms and IoT devices. For example, Fernandes et al. [FJP16] identified design flaws in permission controls of SmartThings home apps and revealed the severe consequences of over-privileged devices. In another paper, Xu et al. [XWP14] surveyed the security problems on IoT hardware design. Other efforts have explored vulnerability analysis within specific IoT devices. For example, Oluwafemi et al. [OKGP13] investigated the security risks in light systems controlled by compromised automation systems and Ho et al. [HLM+16] studied the security of smart locks. These works have found that apps can be easily exploited to gain unauthorized access to control devices and leak sensitive information of users and devices.

Many of previous efforts on taint analysis focus on the mobile-phone platform [C+07, ARF+14, GKP+15, EGH+14, ZJS+11, GLL+13]. These techniques are designed to model domain-specific challenges like on-demand algorithms for context and object sensitivity. Several efforts on IoT analysis have focused on the security and correctness of IoT programs using a range of analyses. To restrict the usage of sensitive data, FlowFence [FPR+16, RFP16] enforces sensitive data flow control. Yet, FlowFence requires additional developer effort and computational overhead at run-time. ContexIot [JCW+17] is a permission-based system that provides contextual integrity for IoT programs at run time. It is designed to infer the app context automatically and to enforce permissions based on that context.

Tool Name	Domain	Source Code Analysis	Dynamic Analysis	Semantics Analysis	Privacy Analysis	NLP/ML Analysis	User Awareness	Overhead Evaluation	Freely Available
TaintDroid [EGH+14]	Android	●	●	○	●	○	●	●	●
FlowDroid [ARF+14]	Android	●	○	○	●	○	○	○	●
FlowCog [PCD+18]	Android	●	●	●	○	●	●	○	●
FlowFence [FPR+16]	IoT	●	●	○	○	○	○	●	●
ContextIoT [JCW+17]	IoT	●	●	○	○	○	●	●	○
SAINT [CBS+18]	IoT	●	○	○	○	○	●	○	●
ProvThings [WHBG18]	IoT	●	●	○	○	○	○	●	○
iRuler [WDY+19]	IoT	●	○	●	○	●	○	○	○
IOTWATCH	IoT	●	●	●	●	●	●	●	●

Table 3.1: Comparison between IOTWATCH and other similar analysis tools for Android and IoT apps.

3.3.2 Dynamic Analysis of IoT Applications

Information Flow Analysis (IFA) tools have been proposed for the mobile phone [ZJS+11, GLL+13, C+07, ARF+14, GKP+15, EGH+14] and IoT apps [FJP16, CBS+18, RFP16]. Table 3.1 compares IoTWatcH with other IFA tools. Though these solutions are effective in identifying sensitive data flows, they only consider privacy concerns from data flows that contain tainted variables. FlowCog [PCD+18] recently proposed a solution to address this issue. Here, the authors analyze context from Android app views and establish data flow dependencies based on app context. However, due to specific architectural differences, Android-based solutions cannot be directly applied to IoT apps. There exist a few systems for IoT app analysis. FlowFence [FPR+16] makes the consumers of IoT data declare their intended activities to enforce data flow policies. More recently, SainT, a static analysis system, is proposed to detect sensitive data flows of IoT apps [CBS+18]. SainT does not consider sensitive data leaks at runtime and data leaks-through strings, while FlowFence often over-approximates the data leaks which would lead to false positives when a precise dependency between taint variables and the leaks cannot be established. Indeed, our analysis showed that 64% of the analyzed apps leak sensitive data through simple strings that did not include information from taint variables, yet the strings contain sensitive information; thus the analysis of strings are required. Lastly, ProvThings [WHBG18] uses static and dynamic analysis to collect provenance to identify the root cause of attacks in IoT apps. ProvThings is limited to analyze dependencies between events and data states and does not offer any built-in privacy analysis.

3.4 Digital Forensics on Resource-limited Device Data

Smart settings such as smart home/office systems have become popular in recent years. However, there exists no platform for forensic analysis of data from a smart environment that considers the interaction between smart devices, users, and security policies in place. Also, existing solutions assume trusted devices [WHBG18] which despite considered in other works [FJP16, FPR+16, JCW+17], can be unrealistic in several scenarios. In this section, we discuss existing forensic data analysis platforms for smart environments and their shortcomings.

3.4.1 Forensic Data Collection from the Smart Environment

Previous approaches collect data from the smart environment and devices. Some of these works only focus on vendor-specific devices [CPLK12], present general methods to only collect data without any future analysis [OJES13, WD16], or have proposed models to collect data for forensic purposes using IoT [PNR15, HV17, BGV18]. Kebande et al. proposed a generic approach, *DFIF-IoT*, to analyze digital forensic data in IoT settings [KR16]. Here, the authors presented a generalized method to capture forensic data from IoT devices including cloud, network, and device-level forensic data. However, this work only discusses theoretical approaches without any real-life implementation and evaluation, which decreases the practicality of this work. Zawoad et al. proposed FAIoT, a forensic-aware eco-system to collect forensic data from IoT platforms [ZH15]. FAIoT collects data systematically and organizes it for further analysis. One shortcoming of this work is that there is no implementation of data analysis as part of the framework. Chung et al. proposed a forensic framework to collect and analyze forensic data in an IoT eco-system [CPL17]. However, this solution is limited to Amazon Alexa with only one device-specific solution.

3.4.2 Smart Data Logging

Previous works have used logging to have access to information as a result of smart app executions. Some IoT programming platforms provide the ways for logging smart app data [Win, Iri, Log, Smai]. Personalized open-source solutions are also popular [Sim, Smae]. In this context, solutions like ProvThings [WHBG18] provide a comprehensive analysis of smart apps and smart devices logs. Specifically, the authors propose a platform-centric approach that looks at activities from smart apps for data provenance purposes. However, the analysis is limited to only consider the temporal relationship between devices and apps events without providing further and deep analysis on the data. Lastly, this work assumes trusted devices which despite considered in other works [FJP16, FPR+16, JCW+17], can be unrealistic in several scenarios.

CHAPTER 4

HOST-BASED RESOURCE-LIMITED DEVICE CLASS IDENTIFICATION

4.1 Introduction

In this chapter, we address the challenge of applying expensive fingerprinting techniques to resource-limited devices by introducing Stop-and-Frisk (S&F), a novel CPS signature-based fingerprinting framework intended to perform device class identification and complement other traditional security mechanisms in the CPS infrastructure. Specifically, the fingerprinting approach combines system and function call tracing techniques, signal processing, and performance analysis on the devices to implement a signature-based challenge-response fingerprinting solution. By using this approach, S&F studies the behavior of the devices within a CPS infrastructure as well as their software/hardware architecture and configuration to identify real CPS devices and discriminate them from impersonators.

To test the efficacy of S&F, we implement a realistic CPS testbed that contains different classes of devices with different resources (i.e., memory and CPU), architectures, computing capabilities, and configurations. The experimental results demonstrate that, by combining behavioral and performance analysis at both kernel and hardware levels, S&F achieves excellent rate for the CPS device class identification. Finally, the performance analysis shows minimal overhead on the CPS devices' computing resources.

The contributions of this chapter are as follows:

1. Stop-and-Frisk*:* We designed and implemented a novel CPS signature-based fingerprinting approach that performs CPS device class identification using system and function call information, signal processing, and performance analysis

Fingerprinting Tool	Requires Extensive Analysis	Depends on Network Dynamics	Analyzes Physical Metrics	Analyzes OS Metrics	Yield High Overhead
[KBC05, URC+13, KCB10, FSL+16, RUB15]	●	●	○	○	X
[SPJ15, LPBZ12, ZDLZ14]	●	○	●	○	X
S&F	○	○	●	●	○

● - Yes, ○ - No, and X - Not reported

Table 4.1: Comparison between Stop-and-Frisk and other fingerprinting techniques.

of the devices. The fingerprinting solution may complement other traditional security mechanisms and provide dependability to CPSs.

2. *Realistic CPS testbed:* We designed a realistic and representative CPS security testbed that includes different classes of CPS devices with varying resources and configurations.

3. *High performance:* The results from the performance evaluation prove that the fingerprinting framework achieves very high accuracy while introducing minimal overhead in the utilization of the computing resources available in the CPS devices.

4.1.1 Differences from Existing Works.

Table 4.1 compares S&F against other fingerprinting techniques based on specific design and implementation criteria. We selected these features as we believe some of them may directly impact the performance of the time-critical CPS infrastructure. For instance, Table 4.1 shows that, as opposed to S&F, other fingerprinting technique require of extensive data analysis or depend on specific network metrics to achieve their results. Surprisingly, these solutions do not offer specific overhead performance analysis, even though fingerprinting solutions that focus on the behavior of the network dynamics may impact the performance of the CPS infrastructure. Additionally,

their effectiveness either depend on the network's configuration, the analysis of extensive amount of data, or the observation of fingerprinting features over long periods. S&F is different from other discussed solutions since it is host-based and device-centered. Also, it does not require traffic monitoring, the study of the interaction between CPS devices and other network equipment, nor need to overcome inevitable errors or overhead (e.g., latency) that can be introduced from changes in network dynamics. The introduced framework implements a signature-based device type fingerprinting mechanism that studies the behavior and performance of the CPS devices at both hardware and kernel levels. S&F utilizes a challenge-response approach where the devices perform standard CPS functionalities and operations. Finally, our technique achieves excellent identification results while introducing very little overhead to the CPS devices at downtime.

4.2 Threat Model and Use Case

Stop-and-Frisk considers an attacker that inserts unauthorized devices into a CPS infrastructure. The unauthorized devices intend to spoof real CPS devices to gain access to restricted areas of the network and perform malicious operations. These malicious operations may include: (1) stealing sensitive information, (2) poisoning physical measurements, and (3) creating the conditions for new attacks in the future. We consider these realistic scenarios where the attackers do not have access to original CPS devices with the same hardware and software configurations used in the field. Instead, they utilize spoofing devices that mimic real CPS network operations. Finally, in this chapter, we do not consider counterfeit or compromised (either by hardware or software) real CPS devices as part of the threat model as they are outside of the scope of this fingerprinting work. In the following, we present a simple use case that

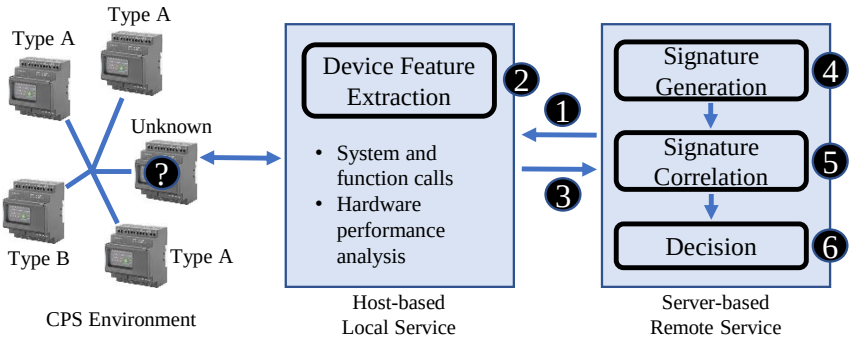


Figure 4.1: Architecture of STOP-AND-FRISK to identify different CPS device classes.

describe the problem scope of S&F. Then, we overview the device class fingerprinting framework and present details of its modules and processes.

4.2.1 Problem Scope

We present the problem scope through a use case. Let's assume that there is a CPS network infrastructure where devices of different types interact to execute a task T. The specific class of some of the devices in the setup is known (i.e., Type A and Type B); however, an *Unknown* device is also present. There are two direct benefits derived from implementing S&F: (1) the network operators may be able to verify that the devices in the network are of the expected class (based on the specific tasks they are executing) and (2) they may be able to identify unknown devices and determine if they are, in fact, authorized to be present or not.

Since most of the CPS devices perform time-critical operations in the network, our CPS device class fingerprinting framework is intended to be applied at the device's patch- or maintenance-time (i.e., downtime). That way, S&F's operations would focus on individual devices and would put minimal overhead on the systems. Such operations require of the interaction of two different services: a server-based remote service (running from a remote server that monitors the CPS environment) and a host-based local service (running on the CPS devices).

4.3 Overview of S&F

Figure 4.1 depicts the general overview of the device class fingerprinting framework. First, a scheduler running in the remote server that executes S&F makes a signature request to the CPS device (i.e., localhost) at downtime (❶). Such a request activates the host-based local service that implements the *Device Feature Extraction* module (❷). This module executes a secret challenge and extracts software- and hardware-related data generated during the device's reaction to the challenge. Specifically, it hooks into the device's activity and extracts lists of system and function calls. Additionally, the Device Feature Extraction monitors the performance of the device regarding CPU utilization, memory utilization, and application execution time while extracting the calls. Once finished, the module derives specific features from the collected data. Specifically, these features are related to the set of function and system calls triggered, the amount of every different call type, and their arguments, respectively. Additionally, it computes the CPU utilization, the amount of memory allocated to execute the challenge's response, and the total execution time. Once all the required features are acquired, the local service sends it to the remote server using a secure channel (e.g., VPN-based data exchange) for further analysis (❸).

On the server side, the collected features are then utilized to generate the signature of the CPS device inside the *Signature Generation* module (❹). Further, the generated signature is correlated (❺) against other signatures previously extracted from known-type CPS devices (i.e., ground-truth). Finally, a threshold-based decision algorithm defines the unknown device class (❻).

4.4 Device Feature Acquisition

The first step in the fingerprinting process is to obtain the list of features used to create the device-class unique signature. In general, the implementation of the introduced host-based fingerprinting technique is simple and CPS operation-friendly since (1) it does not require traffic monitoring over long periods and (2) the devices are monitored at downtime and under standard CPS operating conditions.

4.4.1 Challenge-Response Approach

S&F uses a challenge-response mechanism to study the behavior and performance of the unknown CPS devices. Such study generates device type-specific signatures that are used later for identification purposes. The Device Feature Extraction module running in the host is the one in charge of securely storing and executing the challenge. There are some advantages associated with storing and executing the challenge locally in the host devices. First, it eliminates the need of creating extra secure channels to receive files from the remote server, especially at downtime when connection capabilities may be limited. Second, S&F may automatically flag as unauthorized those devices with the wrong challenge or where the Device Feature Extraction module is unavailable at test time (which adds additional security layer in case attackers try to mimic the performance of S&F). Third, even if the attackers can still implement S&F in the spoofing devices, the final decision depends on the device behavior rather than on metrics that can be easily spoofed. Assuming that the attackers can change the device's behavior and also modify the hardware performance results in S&F's signature, they still need to guess the right values to guarantee that the fake signature would correlate with the one stored in S&F's server for the specific device type analyzed.

4.4.2 Parametric Call List (PCL)

S&F utilizes system and function call hooking techniques [LBAU17] to obtain all the system and function calls that a specific CPS device-class triggers as a response to a stimulant (i.e., challenge). From the call lists, the Device Feature Extraction module obtains the value of distinctive metrics such as (1) the set of specific triggered calls, (2) the total number of calls by type (e.g., *malloc*, *free*, *open*, etc.), and (3) the value of specific call's arguments (e.g., the amount of memory allocated by *malloc*). We refer to this list of parameters extracted from the system and function calls as *Parametric Call List* (PCL), which is defined as:

$$PCL_i = \{x_i \in X_i : \exists X_i \wedge X_i \neq \emptyset\}, \tag{4.1}$$

where PCL_i represents the PCL extracted from device i, x_i represents the parameters of the calls extracted from device i, and X_i represents the call lists extracted from device i. Finally, the hooking technique utilized to extract the system and function calls is irrelevant to our design and it is specific to every device's architecture and OS [LBAU17]. Due to reduced complexity, the PCL generation is completed on the host devices. Once finalized, the PCL is sent to the S&F remote server using secure communication channels.

4.4.3 Device Performance Index (DPI)

The second feature used in our framework to identify CPS device classes is the *Device Performance Index* (DPI). Since call lists can be faked by attackers to mimic authentic CPS devices, S&F further combines hardware performance analysis as part of its identification mechanisms. As mentioned in Chapter 2 (Section 2.1), CPS devices are not expected to change their general functionality, so the average system performance is also expected to remain steady over time [FSL+16]. Hence, as every class of CPS

device has specific functionalities, the device's performance obtained under regular operating conditions can be used for identification purposes.

S&F integrates three different metrics into the DPI analysis: memory utilization (α), CPU utilization (β), and execution time (γ). Since every metric is evaluated over a certain time interval t of interest, we can consider every metric as a vector quantity as:

$$\begin{aligned} \vec{\alpha} &= \{|\alpha|, \phi_\alpha\}, \\ \vec{\beta} &= \{|\beta|, \phi_\beta\}, \\ \vec{\gamma} &= \{|\gamma|, \phi_\gamma\}. \end{aligned} \tag{4.2}$$

where $|\alpha|$, $|\beta|$, and $|\gamma|$ represent the magnitude of memory utilization, CPU utilization, and execution time, respectively and ϕ_α, ϕ_β, and ϕ_γ represent the *direction of change* (e.g., positive for increased utilization and negative for decreased utilization) of every metric in t, respect to any previous time interval t_0. From here, we can define DPI as the volume of the parallelepiped (Figure 4.2) whose adjacent sides are defined by the averaged metrics $\vec{\alpha}$, $\vec{\beta}$, and $\vec{\gamma}$. The parallelepiped volume can be found via the *scalar triple product* of these metrics as:

$$\begin{aligned} DPI &= |\overline{\vec{\gamma}} \cdot (\overline{\vec{\alpha}} \times \overline{\vec{\beta}})|, \\ &= \epsilon_{ijk} \overline{\gamma^i \alpha^j \beta^k} : i, j, k \in 1, 2, 3...n, \\ &= det \begin{bmatrix} \gamma_1 & \gamma_2 & \gamma_3 & \dots & \gamma_n \\ \alpha_1 & \alpha_2 & \alpha_3 & \dots & \alpha_n \\ \beta_1 & \beta_2 & \beta_3 & \dots & \beta_n \end{bmatrix}. \end{aligned} \tag{4.3}$$

S&F computes the DPI for every CPS device class and includes this value, along with the PCL, as part of the unique device signature. Due to a higher complexity compared to the PCL generation, S&F generates the DPI in the Signature Generation module, and before creating the specific device signature.

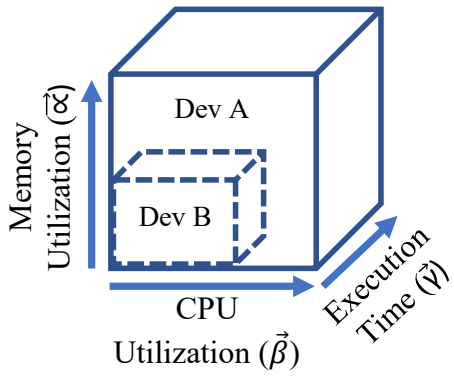


Figure 4.2: Three-dimensional representation of the DPI of two different CPS device classes. The DPI of device class A is greater than the DPI of device class B in around 2x, 1.4x, and 2.5x of memory, CPU utilization, and execution time, respectively.

4.5 Device Signature Generation

Once S&F completes the device feature acquisition (i.e., PCL and DPI), it generates the unique CPS device-class signature based on the extracted features. Such features are selected, so they guarantee a comprehensive analysis on the device behavior and performance at both OS and hardware levels, while keeping identifiable device characteristics (e.g., network metrics, device ID) out of the analysis to preserve privacy. The signature structure must solely guarantee the correct identification of the device classes so the spoofing or unauthorized devices can be detected. The final signature for every CPS device type has the following format:

$$[\mu(PCL_{lists}), \mu(DPI_{lists})]. \tag{4.4}$$

where $\mu(PCL_{lists})$ represent the average of parameters extracted from the system and function calls and $\mu(DPI_{lists})$ represents the average of CPU utilization, memory utilization, and execution time values included in the DPI.

4.6 Ground Truth Devices - Learning Phase

The effectiveness of S&F in identifying different classes of CPS devices relies on the use of reliable ground-truth device-based signatures. S&F correlates the ground-truth signatures against the behavior and performance of the unknown devices for identification purposes. In general, ground truth-capable devices must adhere to two basic rules: (1) they must indeed characterize the behavior of the device classes in their network region, and (2) they must perform stationary deterministic operations inside the CPS infrastructure over time. The first rule guarantees that the device's metrics and the *type of activity* (i.e., the device's specific application inside the CPS network) performed by the ground-truth device are both considered to define its class (Chapter 2, Section 2.1). On the other hand, the second rule guarantees reliability. S&F requires that ground-truth devices behave in a deterministic way to guarantee that, if the same challenge is applied, every device class always generates the same signature over time. Steady behavior constitutes a realistic requirement since previous research works have highlighted the deterministic behavior of CPSs [FSL+16]. Finally, the mechanism of obtaining signatures from ground-truth devices is known as *learning phase*. Once S&F completes this process, it stores the ground truth device-based signatures into a signature database (SDB).

Ground-truth devices characterize a CPS network region. As discussed in Chapter 2 (Section 2.2), device class classification have been traditionally focusing on the branch, model, and the specific application of the devices. We also utilize these metrics to perform a first-round classification of the potential ground-truth devices classes and to determine how many different classes may be present in the network. We also utilize this information to better organize signatures in the SDB. Thus, we first *group* the different classes of devices in the network based on these metrics. Further, we ap-

ply behavioral and performance analysis to extract the signatures from every device class.

To evaluate the reliability of the ground truth devices, we calculate the autocorrelation of different PCLs and DPIs obtained while the devices execute the same process (i.e., challenge) but at different time intervals t. We use Equations 4.5 and 4.6 to calculate PCL and DPI autocorrelation, respectively, as follows:

$$\rho_{PCL_{t_i},PCL_{t_{i+1}}} = \frac{\sum PCL_{t_i} PCL_{t_{i+1}} - n\overline{PCL_{t_i} PCL_{t_{i+1}}}}{n s_{PCL_{t_i}} s_{PCL_{t_{i+1}}}}, \tag{4.5}$$

$$\rho_{DPI_{t_i},DPI_{t_{i+1}}} = \frac{\sum DPI_{t_i} DPI_{t_{i+1}} - n\overline{DPI_{t_i} DPI_{t_{i+1}}}}{n s_{DPI_{t_i}} s_{DPI_{t_{i+1}}}}, \tag{4.6}$$

where PCL_{t_i}, $PCL_{t_{i+1}}$, DPI_{t_i}, and $DPI_{t_{i+1}}$ represent PCL and DPI metrics extracted from the same CPS process, but executed at different time interval, n represents the size of the arrays PCL and DPI, and s represents the standard deviation.

Algorithm 1: Generate Signature (Learning Phase)

1: $iterations \leftarrow N$
2: $PCL_{lists} \leftarrow null$
3: $DPI_{lists} \leftarrow null$
4: **for** $i = 0$ to $iterations - 1$ **do**
5: $\quad PCL_{lists}[i] \leftarrow getParamList()$
6: $\quad DPI_{lists}[i] \leftarrow getDPIndex()$
7: **end for**
8: **for** $i \in 0...size(PCL_{lists}) - 1$ **do**
9: $\quad gTVec \leftarrow \rho_{y_i,y_{i+t}}(PCL_{lists}[i], PCL_{lists}[i+1])$
10: **end for**
11: $grdTh \leftarrow \mu(gTVec)$
12: **if** $grdTh > \xi$ **then**
13: $\quad SDB \leftarrow [\mu(PCL_{lists}), \mu(DPI_{lists})]$
14: **end if**

Algorithm to obtain CPS ground truth device-based signatures during the learning process.

Algorithm 1 details the process of obtaining the ground-truth signatures during the learning phase. Initially, in Line 1, the number of iterations (for averaging purposes) is initialized and the local variables PCL_{lists} and DPI_{lists} are declared. These

variables contain the list of parameters from every iteration, i. Then, in Lines 5 and 6, system and function call tracing techniques are applied to obtain the PCL at different time intervals t. Also, the algorithm guarantees the calculation of the DPI for every iteration. In Line 9, the autocorrelation vector between the different time intervals of PCL is calculated. Later, in Line 11, the average of all autocorrelation values is computed. Finally, if the autocorrelation value is greater than ξ from Line 12, the algorithm accepts the evaluated CPS device as ground-truth, and stores its signature into the SDB (Line 13). In practice, the value of the threshold ξ is agnostic and can be determined based on the specific characteristics of the operations in the ground-truth device.

Algorithm 2: Identify Device Class (Prediction Phase)

1: $CPSsignList \leftarrow SDB$
2: $iterations \leftarrow N$
3: $PCL_{lists}, DPI_{lists}, CPSdeviceID \leftarrow null$
4: $signatue \leftarrow null$
5: **for** $i = 0$ to $iterations - 1$ **do**
6: $\quad PCL_{lists}[i] \leftarrow getParamList()$
7: $\quad DPI_{lists}[i] \leftarrow getDPIndex()$
8: **end for**
9: $signature \leftarrow [\mu(PCL_{lists}), \mu(DPI_{lists})]$
10: $corrXYmax \leftarrow 0$
11: **for** $i = 0$ to $size(CPSsignList) - 1$ **do**
12: $\quad corrXY \leftarrow \rho_{x,y}(CPSsignList(i), signature)$
13: $\quad$ **if** $corrXY > \delta$ & $corrXY > corrXYmax$ **then**
14: $\quad\quad CPSdeviceID \leftarrow i$
15: $\quad\quad corrXYmax \leftarrow corrXY$
16: $\quad$ **end if**
17: **end for**

Stop-and-Frisk algorithm for CPS device class identification.

4.7 Signature Correlation and Decision - Prediction Phase

On the server side, S&F correlates the signature obtained from the unknown CPS device against the ground-truth signatures stored in the SDB. This process is known as *prediction phase* and is detailed in Algorithm 2. The process to obtain the signature of the unknown CPS device follows similar steps as in Algorithm 1. However, this time the system is not required to calculate autocorrelation since S&F assumes that the signature of the unknown CPS device is valid after n different iterations (Lines 2, 6 and 7 in Algorithm 2). Once the unknown signature is finally generated in the server (Line 9), S&F calculates the correlation between *signature* and all the unique CPS ground-truth signatures from the SDB (Line 12) using Equation 4.7 and Equation 4.8:

$$\rho_{PCL_X,PCL_Y} = \frac{\sum PCL_{X_i}PCL_{Y_i} - n\overline{PCL_X PCL_Y}}{ns_{PCL_X}s_{PCL_Y}}, \tag{4.7}$$

$$\rho_{DPI_X,DPI_Y} = \frac{\sum DPI_{X_i}DPI_{Y_i} - n\overline{DPI_X DPI_Y}}{ns_{DPI_X}s_{DPI_Y}}. \tag{4.8}$$

where n represents the size of PCL_X (i.e., ground truth PCL), PCL_Y (i.e., unknown device PCL), DPI_X (i.e., ground truth DPI), and DPI_Y (i.e., unknown device DPI), $\overline{PCL_X}$, $\overline{PCL_Y}$, $\overline{DPI_X}$, and $\overline{DPI_Y}$ represent the mean value, and s_{PCL_Y}, s_{PCL_Y}, s_{DPI_X}, and s_{DPI_Y} represent the standard deviation, respectively.

After computing correlation, the decision process starts. The logical condition in Line 12 evaluates that (1) the correlation between the unknown device and signature i from the database is greater than a certain threshold δ and (2) this value of correlation is a maximum obtained from all the iterations in Algorithm 2. If such a condition holds, the unknown CPS device is deemed to be the same CPS device class as CPS device i from the database (Line 14). On the other hand, if the condition in Line 13 is never satisfied, the unknown device is classified as *Unknown*, and flagged by S&F. One can observe that, from Algorithms 1 and 2, the value of the correlation threshold

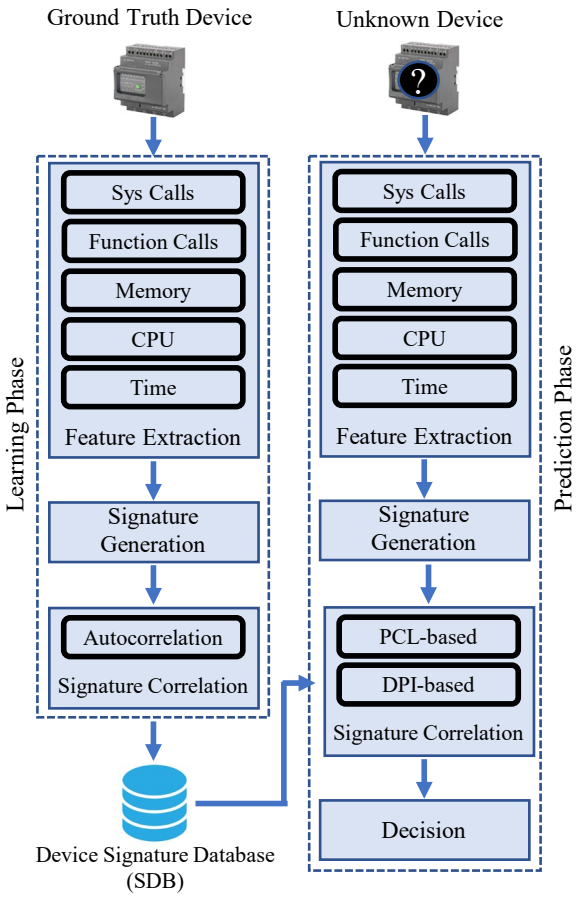


Figure 4.3: We introduce a device-class identification framework using call tracing techniques, signal processing, and device performance analysis.

δ is a configurable parameter and depends on the threshold ξ used to generate the ground-truth device signature.

Finally, the processes described in Algorithms 1 and 2 are summarized in Figure 4.3. One can notice that, for both learning and prediction phases, S&F reuses the first two modules of its architecture since they contain similar operation steps. In both phases, S&F needs to extract features and create signatures from ground truth or unknown devices, respectively.

4.8 Performance Evaluation

In this section, we present experimental results that demonstrate the effectiveness of S&F to fingerprint and identify different classes of CPS devices. The main purpose of these experimental evaluations is to answers the following research questions:

- **RQ1: Learning Phase.** How the fingerprinting framework performs during the learning phase? (Section 4.9).
- **RQ2: Prediction Phase.** What is the accuracy of S&F in fingerprinting CPS devices while implements (1) PCL correlation only, (2) DPI correlation only, and (3) combines both PCL and DPI analysis? (Section 4.10).
- **RQ3: Overhead.** What is the overhead introduced by S&F to the CPS devices? (Section 4.11).

4.8.1 Testbed Implementation

We implemented our CPS device fingerprinting testbed considering the characteristics of the CPS described in Chapter 2 (Section 2.1). These characteristics were included in our testbed as follows:

1. *Diversity in hardware and software resources*: We included 11 different CPS device classes with a variety of computing resource characteristics and different hardware/software configurations. This diversity makes our testbed representative of a large population of real CPS devices; from small devices with limited resources to resource-rich devices [L. 17].
2. *Discriminate then Regroup*: Our implementation must prove the effectiveness of S&F in first, discriminating all different types of CPS devices (to avoid false positives outcomes) and second, grouping devices of the same type together (to

Class #	**Device ID**	**Model Name**	**Hardware Specifications**	**Operating System**	**Release**
1	BB_1	Black	AM355x Cortex-A8 @ 1GHz 512MB DD3 RAM	Linux Beaglebone 4.1.5	Debian 8.3 Jessie
2	BB_2	Black	AM355x Cortex-A8 @ 1GHz 512MB DD3 RAM	Linux Beaglebone 3.8.13	Debian 7.9 Wheezy
2	BB_3	Black	AM355x Cortex-A8 @ 1GHz 512MB DD3 RAM	Linux Beaglebone 3.8.13	Debian 7.9 Wheezy
3	GZ	2-1.0	AMD GX210HA @ 1GHz 1GB DD3 RAM	Linux Ubuntu-mate 4.4.0	Ubuntu 16.04 xenial
4	LM_1	A-10	A10 Cortex-A8 @ 1GHz 512MB DD3 RAM	Linux A10Lime 3.4.90	Debian 3.4.90
4	LM_2	A-10	A10 Cortex-A8 @ 1GHz 512MB DD3 RAM	Linux A10Lime 3.4.90	Debian 3.4.90
5	ODR	XU3	HMP Samsung Exynos 5422 Cortex A15 Quad core @ 2GHz A7 Quad core @ 900MHz 2GB DD3 RAM	Linux Odroid 3.10.96	Ubuntu-Mate 16.04
6	OPi_1	PC	H3 Quad core Cortex-A7 @ 1GHz 1GB DD3 RAM	Linux Orange Pi Kali 3.4.39	Kali 2.0
6	OPi_2	PC	H3 Quad core Cortex-A7 @ 1GHz 1GB DD3 RAM	Linux Orange Pi Kali 3.4.39	Kali 2.0
7	RPi_1	2B	Cortex-A7 @ 900MHz 1GB DD3 RAM	Linux Raspberry Pi 4.1.7	Raspbian 8.0 jessie
7	RPi_2	2B	Cortex-A7 @ 900MHz 1GB DD3 RAM	Linux Raspberry Pi 4.1.7	Raspbian 8.0 jessie
8	RPi_3	3B	Cortex A53 Quad core @ 1.2GHz 1GB DD3 RAM	Linux Raspbery Pi 4.4.11	Raspbian 8.0 jessie
8	RPi_4	3B	Cortex A53 Quad core @ 1.2GHz 1GB DD3 RAM	Linux Raspbery Pi 4.4.11	Raspbian 8.0 jessie
9	RPi_5	1B	ARM1176 @ 700MHz 512MB DD3 RAM	Linux Raspberry Pi 4.1.13	Raspbian 7.0 wheezy
10	LTP_1	Dell E6520	Intel Core i7-2760 QM @ 2.4GHz 6GB DD3 RAM	Linux 3.19.0	Ubuntu 14.04 trusty
11	LTP_2	Toshiba P55W	Intel Core i5-5200 @ 2.7GHz 6GB DD3 RAM	Linux 4.4.0	Ubuntu 16.04 xenial

Table 4.2: Different device classes used in our CPS testbed.

avoid false negatives outcomes). To evaluate both metrics, we also included in our testbed multiple devices of some specific classes (Table 4.2). With this, we can determine how S&F performs on classifying different types of devices into different classes, and also, in grouping different devices of the same type into a common class.

3. *CPS-specific tasks and processes*: During the learning phase, the devices included in our testbed performed real CPS networking operations following the IEC61850 communication standard [C. 13, IEC03d]. IEC61850 is a protocol suite that defines the communication standards for electrical substation automation systems. For that, we used an open-source version of the IEC1850 standard (i.e., *libiec*61850) available online [M. 16].

4. *Multiple OSes*: Our experimental CPS devices ran 11 different Linux versions. Using different versions of Linux constitutes a realistic approach since most of the current CPS devices in the field still use some variant of Unix-based OS [RB15]. Additionally, the use of open source approaches in S&F enables the implementation of flexible solutions. In fact, the use of different non-Unix-based OSs in our testbed would not impact the evaluation process of S&F. Previous research works have detailed specific system and unction call hooking techniques that can be applied to all major operation systems. In the end, obtaining the PCL and DPI from devices with different operating systems is independent to S&F's architecture and more related to specific implementation challenges. Finally, despite their noted differences, we purposely kept some similarities among the different devices classes in the testbed. For instance, as shown in Table 4.2, most of the devices are Debian-based systems using ARM CPU architecture. Such an implementation approach would additionally challenge S&F into iden-

tifying device classes whose differences are based on small features instead of taking advantages of very noticeable architectural differences.

The specific devices included in the testbed are detailed in Table 4.2. The rationale behind including many different devices classes in our testbed stem from the fact that CPS infrastructures contain a high diversity of devices. Also, the hardware and software characteristics of the devices may considerably vary in real scenarios. Specifically, in the CPS infrastructure, one can find devices with limited and rich computing resources, various software configurations, and different architectures. In this context, CPS devices' hardware and software characteristics are specific to their functionalities and applications. As a consequence, small changes in the CPS devices' configuration should be highly noticeable in their general behavior. In this chapter, we exploit these characteristics of CPS networks and devices to implement a fingerprinting technique that identifies different CPS device classes based on their behavior and performance. In all the experiments, we computed the results after averaging 30 different runs for all the covered scenarios. The scenarios comprised the application of Algorithm 1 and Algorithm 2 on the devices included in our testbed (Table 4.2) in order to: (1) generate a trustworthy signature database, (2) evaluate device-class similarities and differences, (3) identify different CPS device classes, and finally (4) evaluate the overhead that S&F introduces to the CPS devices' computing resources. During the learning phase, we randomly selected one device from every different class in our testbed. Then, we studied the behavior of the selected devices while they performed real CPS operations. For this purpose, we calculated the autocorrelation of similar CPS processes running in the devices at different time intervals (we set the threshold $\xi = 0.7$). For the cases where a deterministic behavior was identified, the devices were accepted as ground-truth. Then, S&F applied the challenge-response

approach discussed in Section 4.3 to extract the signatures. These signatures were then stored in the SDB for identifications purposes during the prediction phase.

For the prediction phase, we set the threshold $\delta = 0.7$, which marks the range from moderate to high correlation that is widely accepted in the literature [Ros01]. Depending on the specifics of the evaluated CPS environment, this value of δ may change during real S&F implementations. For instance, in practical applications of S&F, the analysis over a group of well-known devices (i.e., control group) may give the best decision threshold value for the specific network region. Finally, since we are working with UNIX-based OSes, we utilized library interposition [GE11] and *ptrace* function [R.] to extract the lists of system and function calls, respectively and generate the PCL. Also, we utilized the *top* and *time* commands to extract relevant information for the DPI. Finally, as an additional evaluation challenge, we used the same challenge to trigger responses from all the tested devices. That way, S&F detects and flags the differences between devices classes based not on the CPS tasks they process, but only on the relative differences in specific behavior and hardware performance.

4.8.2 Performance Metrics

To measure the performance of S&F, we further compute the standard classification metrics of accuracy, recall, precision, and specificity. We define these metrics in Equations 4.9, 4.10, 4.11 and 4.12, respectively.

$$A_{CC} = \frac{(T_P + T_N)}{(T_P + T_N + F_P + F_N)}, \tag{4.9}$$

$$R_{EC} = \frac{T_P}{(T_P + F_N)}, \tag{4.10}$$

$$P_{REC} = \frac{T_P}{(T_P + F_P)}, \tag{4.11}$$

$$S_{PEC} = \frac{T_N}{(T_N + F_P)}. \tag{4.12}$$

where T_P stands for true positive or the case where a CPS device is correctly classified as of some specific class; T_N stands for true negative or the case where a CPS device is correctly classified as of not from some specific class; F_P stands for false positive or the case where a CPS device is identified using the wrong signature; and finally F_N stands for false negative or the case where a CPS device whose signature has been previously stored in the database cannot be correctly identified.

4.9 Performance of S&F during the Learning Phase

As described in Section 4.3, the first step towards applying S&F is to find a reliable set of unique signatures that characterize the different CPS device classes. The signature generation process uniquely uses autocorrelation between different realizations of PCL and DPI. Moderate to high values of autocorrelation (typically over 0.7 [Ros01]) indicate that the specific CPS device (which is assumed to be a trusted CPS device with no prior tampering or unauthorized components) can be used as ground-truth to create a reliable signature for its class.

Figure 4.4 depicts the evaluation results after applying Algorithm 1 (Section 4.3) over randomly selected devices from different classes included in the testbed. One can observe that, in all the cases, the autocorrelation values are over the threshold xi, which indicates the deterministic behavior of the devices over time. Again, we obtained these results after 30 different PCL and DPI runs in every device at different time intervals. These results constituted a strong indicator that ground-truth signatures can be obtained for all the devices in the testbed. Finally, once the ground-truth CPS devices were accepted, the signatures were generated and stored in the SDB.

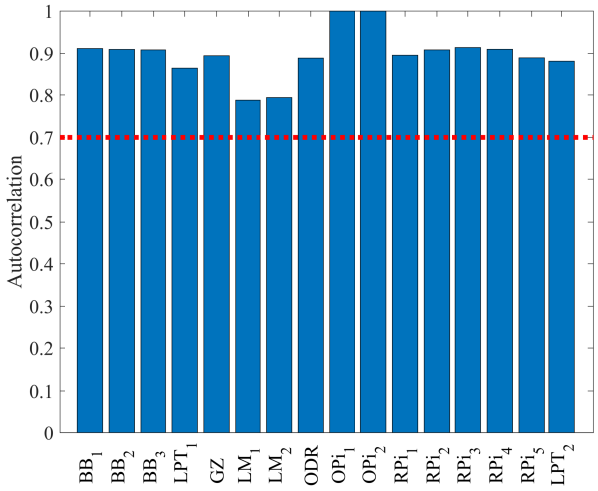
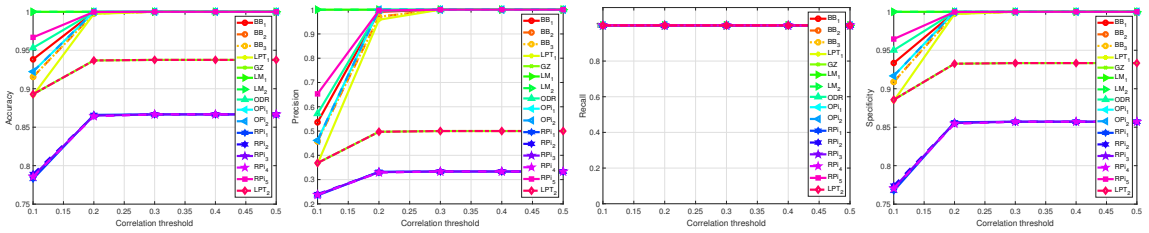


Figure 4.4: Experimental results from the evaluation of the learning phase: autocorrelation results after applying Algorithm 1 on all the devices selected as ground-truth.

4.10 Performance of S&F during the Prediction Phase

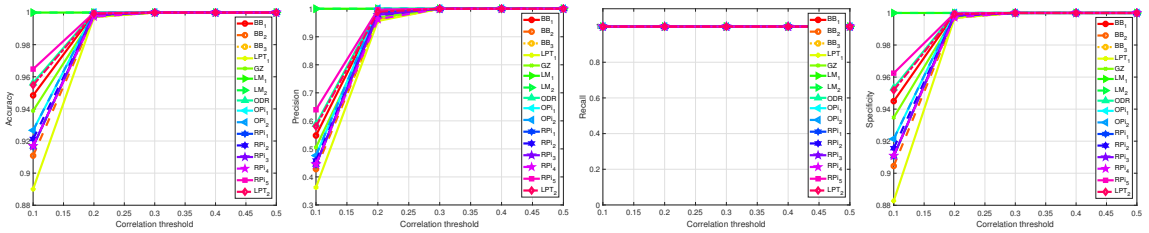
The primary goal of S&F is to predict the right class of every different host based on similarities in OS's behavior, hardware architecture, and configuration. Additionally, S&F must be able to group devices from the same class effectively. Before executing the prediction phase, S&F challenged the unknown devices, extracted its features, and created their unique signatures. S&F applies system and function call hooking techniques (i.e., library interposition and *ptrace*) to generate the PCL of the unknown devices. At the same time, S&F extracted the performance features used to calculate the DPI of every single device. Once these processes were completed, S&F sent this information to the remote server for processing. The prediction phase was then activated by applying Algorithm 2 (Section 4.3).

To thoroughly test the efficacy of S&F and evaluate the real contribution of every fingerprinting feature, we first analyzed the performance of the framework by using PCL-based correlation only. Then, we evaluated how the results improved after incorporating the DPI analysis.



(a) Accuracy (b) Precision (c) Recall (d) Specificity

Figure 4.5: Evaluation of the experimental results after considering PCL correlation only: (a) accuracy, (b) precision, (c) recall, and (4) specificity. One can observe that, in some cases, lower accuracy results were obtained due to false positives among some device classes. These results were improved after combining PCL-based correlation with DPI analysis (Figure 4.6).



(a) Accuracy (b) Precision (c) Recall (d) Specificity

Figure 4.6: Evaluation of the experimental results after considering correlation and device performance index for decision: (a) accuracy, (b) precision, (c) recall, and (4) specificity. One can notice how the overall metrics improved if compared results shown in Figure 4.5.

PCL correlation analysis only. By applying PCL correlation only, S&F achieved accuracy values over 87% for devices RPi_4, RPi_2 and BB_1, and over 94% for GZ, BB_3, LPT_2 and BB_2, respectively (Figure 4.5). On the other hand, precision, recall, and specificity metrics were also affected due to false positive events, as shown in Figure 4.7. In this case, one can observe that S&F incorrectly decided the devices LM_1 and GZ as from the same class. These results impacted the general performance of S&F as observed in Figure 4.5. For instance, one can observe that, for some cases of incorrect class identification, accuracy values never went over 86%.

Figure 4.7 depicts the correlation map (NxN PCL-based correlation matrix) between all the device classes in the testbed before considering DPI analysis. A darker

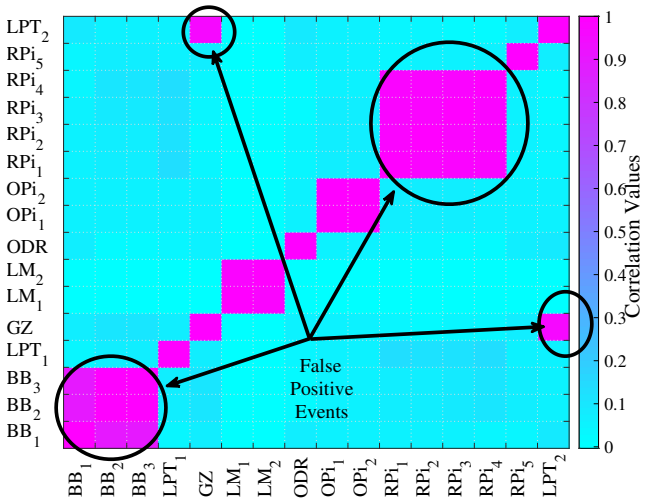


Figure 4.7: Correlation matrix for device-class identification using PCL approach only. False positives as a result of applying only PCL-based correlation are circled (e.g., between GZ and LPT_2 devices).

color indicates high correlation while lighter colors indicate lower correlation values between PCLs from different devices. As per Table 4.2, one should expect a total of 11 different CPS device classes based on the different computing resources and software/hardware configurations. However, from PCL-based correlation analysis, S&F was only able to identify nine out of a total of 11 different classes of devices. From Figure 4.7, one can observe that, for instance, the fingerprinting framework mistakenly confused GZ and LPT_2 as of the same class.

DPI correlation analysis only. We calculated the DPI values for every CPS device in the testbed using Equation 4.3. In Figure 4.8, the results of the DPI calculation are shown. As observed, several DPI values from different devices were very similar, which negatively impacts the feasibility of using this feature for identification purposes. However, for some specific devices, DPI values were significantly different between different classes. To better understand this analysis, we further represented the DPI values versus the average DPI of all the devices classes in the testbed (Figure 4.9). From this figure, it is unclear that DPI analysis may significantly contribute to discriminating devices from different classes.

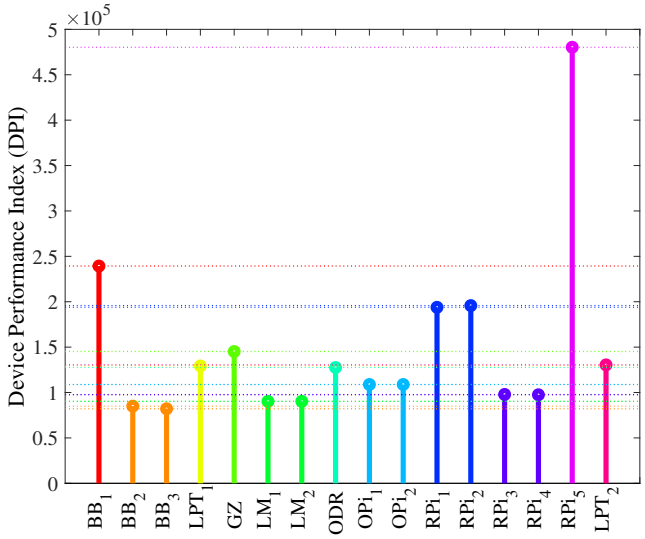


Figure 4.8: Average value of the DPI for all the devices included in the CPS testbed. Experimental results shown that using the DPI-only approach to identify devices may lead to some false positives due to overlapping.

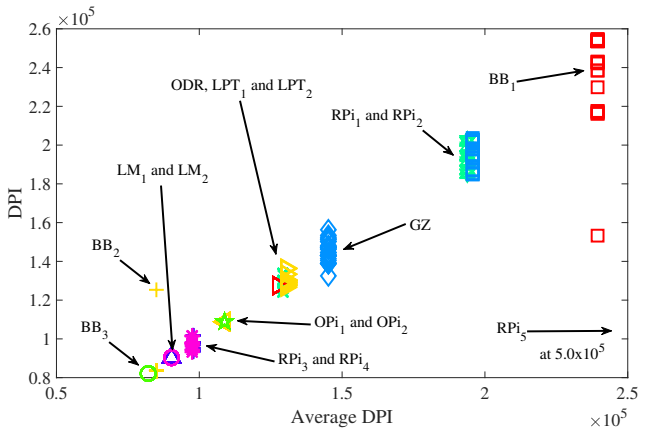


Figure 4.9: The spatial distribution of the device's DPI shows false positive results due to overlapping between devices from different classes.

PCL and DPI analysis combined. We combined both PCL- and DPI-based correlation analysis and obtained a new decision map in Figure 4.10. This time, S&F was able to identify 11 different device classes by removing false positives from previous results. Also, performance metrics significantly improved after incorporating the DPI correlation. In Figure 4.6, we represent the performance metrics used to evaluate S&F. One can observe that all the different metrics significantly improved

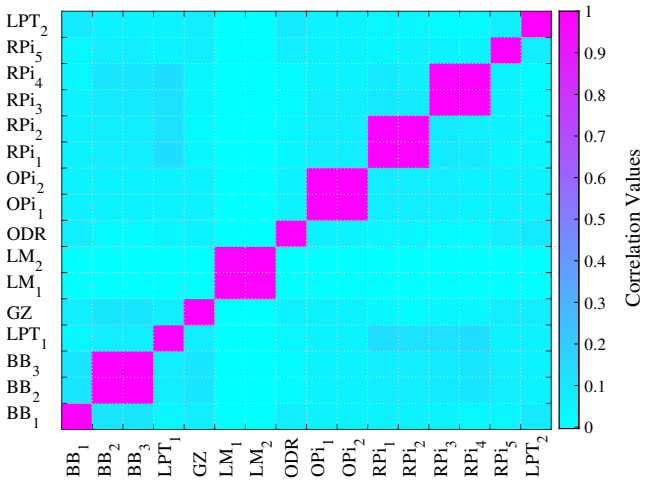


Figure 4.10: After combining PCL-based correlation techniques and DPI analysis, S&F was able to identify the 11 different class of devices included in the CPS testbed from the decision map.

if compared with the values presented in Figure 4.5. For the four different metrics, S&F obtained performance values close to 100%.

4.11 Overhead Introduced by S&F

Table 4.3 summarizes the overhead introduced by S&F to the CPS devices. Despite the benefits of our technique, the characteristics of the CPS devices do not allow for too much overhead (Chapter 2). We calculated the system overhead by analyzing the performance of the devices (1) under normal operating conditions and (2) while applying our fingerprinting technique. In Table 4.3, *ET* refers to Execution Time, *CPU* refers to CPU utilization, *MEM* refers to memory utilization, and *Total MEM* refers to the percent of memory that our technique utilizes out of the total memory available in every device. One can observe that the maximum overhead introduced by S&F regarding increments in execution time, memory, the percentage of total memory, and CPU utilization is 0.04%, 19.8%, 0.0218%, and 1.5%, respectively. Different from other existing solutions, the fingerprinting framework introduced in this chapter did

Devices	Without Stop-and-Frisk			With Stop-and-Frisk			Overhead			
	RT (s)	Mem (KB)	CPU (%)	RT (s)	Mem (KB)	CPU (%)	ET (s)	Mem (KB)	Total Mem (KB)	CPU (%)
	value	value	value	value	value	value	%	%	%	%
BB_1	60.050	1304	3.0	60.055	1343	3.0	0.008	3	0.0039	0
BB_2	60.025	652	2.0	60.049	696	2.03	0.04	6.7	0.0085	1.5
BB_3	60.015	640	2.0	60.029	684	2.0	0.023	6.9	0.0085	0
GZ	60.020	2358	1.0	60.02	2419	1.0	0	2.6	0.0061	0
LM_1	60.007	640	2.0	60.026	752	2.0	0.03	17.5	0.0218	0
LM_2	60.001	640	2.0	60.02	752	2.0	0.03	17.5	0.0218	0
ODR	60.042	656	3.0	60.042	708	3.0	0	7.3	0.0026	0
OPi_1	60.070	504	2.96	60.057	604	3.0	$\sim$0	19.8	0.01	1.35
OPi_2	60.070	504	3.0	60.06	604	3.0	$\sim$0	19.8	0.01	0
RPi_1	60.040	1568	2.0	60.04	1614	2.0	0	2.9	0.0046	0
RPi_2	60.040	1585	2.0	60.04	1629	2.0	0	2.7	0.0044	0
RPi_3	60.030	1575	1.0	60.038	1630	1.0	0.013	3.5	0.0055	0
RPi_4	60.030	1566	1.0	60.041	1624	1.0	0.018	3.7	0.0058	0
RPi_5	60.040	1537	5.0	60.047	1599	5.0	0.01	4	0.012	0
LPT_1	60.001	2042	1.0	60.001	2154	1.0	0	5.5	0.0018	0
LPT_2	60.013	2125	1.0	60.015	2173	1.0	0.003	2.2	0.0008	0

Table 4.3: Average of system overhead introduced by S&F on the devices included in the testbed.

not require to monitor the CPS devices over long periods. S&F was able to identify the different device classes with high accuracy from the device's behavior and performance data extracted after only 30 sec of challenge response. Finally, please note that, since S&F is execute during device's downtime, the current overhead introduced, besides minimal, only affect the devices while they are not performing time-critical CPS operations.

4.12 Summary and Benefits

The CPS device class identification framework introduced here allowed for improved dependability of CPS systems through the monitoring and detection of spoofing or unauthorized devices trying to gain access to or being utilized in critical CPS infrastructures. For the threat model considered in this chapter, device class classification is suitable to detect spoofing devices in CPS rather than host-based fingerprinting that would require deeper analysis while providing unnecessary result metrics (S&F provides details of the device characteristics rather than the device specific identification).

Our technique demonstrated an excellent rate in the CPS device class identification after analyzing a representative set of different device classes included in a realistic testbed. The device diversity in the testbed, regarding computer resource availability and software and hardware configurations, is considered genuinely representative of the real CPS device diversity. The fingerprinting framework was able to discriminate different classes of devices with high accuracy while grouping devices from the same class together. S&F's evaluation resulted in several false positive results after implementing the PCL, and the DPI approaches independently. However, accuracy metrics improved significantly after combining the results of both correlations in a single analysis.

Additionally, based on experimental results, our technique does not yield significant overhead on the use of the devices' computing resources. In fact, after reviewing Table 4.3 (Section 4.8), we may conclude that the overhead introduced by S&F is minimum if compared with the benefits that this tool can bring to the CPS security domain. Also, since S&F analyzes the behavior and performance of the devices at downtime, our solution can be considered complementary to other security mechanisms proposed in CPS (Chapter 3). Finally, although the research work presented in this chapter focuses on CPS device class fingerprinting, our approach can also be applied to many other domains outside the CPS realm, as long as some necessary conditions can be met (Section 4.3).

4.13 Conclusion

CPS use smart devices to collect data and monitor critical operations. However, these devices can be spoofed by attackers to get access to systems, steal information, or disrupt critical operations. In this chapter, we presented Stop-and-Frisk (S&F), a novel CPS signature-based fingerprinting framework used to improve CPS dependability by performing CPS device class identification and complement traditional CPS security mechanisms. Specifically, our approach combined system and function call tracing techniques, signal processing, and device performance analysis to implement a challenge-response-based device class fingerprinting solution. Moreover, we evaluated the efficacy of S&F on a realistic testbed that included different classes of CPS devices with different hardware, software resources, and configurations. Experimental results demonstrated that S&F achieves excellent rate in the identification of CPS devices classes. Also, our performance analysis revealed that the use of the fingerprinting framework would not yield a significant overhead on the CPS devices' computing resources.

CHAPTER 5

DETECTION OF COMPROMISED RESOURCE-LIMITED DEVICES

5.1 Introduction

In this chapter, we review a configurable system-level framework to detect compromised devices performing unauthorized operations inside the CPS infrastructures. Specifically, we introduce our solution through a use case that features the detection of compromised devices in the smart grid [Bab18, Bab19, KBAU18]. The detection framework utilizes system and function call tracing techniques, signal processing, and statistical analysis to detect compromised devices based on their unexpected behavior. In order to test our framework, we designed a realistic representative smart grid substation testbed in which generic CPS devices performed essential operations conforming to the International Electrotechnical Commission 61850 (IEC61850) [IEC03d, IEC03c, IEC03a, IEC03b] protocol suite. The IEC61850 is a protocol suite that defines the communication standards for electrical substation automation systems [C. 13]. The implemented testbed includes both resource-limited (e.g., RTUs, PLCs, and resource-rich (e.g., Phasor Measurement Units (PMUs), Intelligent Electronic Devices (IEDs)) CPS devices. In the testbed, the devices use open-source *libiec*61850 libraries [M. 16] to exchange smart grid time-critical messages using the GOOSE format [C. 13].

In addition, the adversary model complies with the security requirements specified by the standardization organizations [The10] for the smart grid. In total, we consider six different types of compromised devices defined by different combinations of device computing resources and attack payloads.

Finally, we evaluate the performance of our framework by detecting and analyzing behavioral differences between compromised and ground truth devices using three

different detection methods. Experimental results demonstrate that the detection framework achieves an excellent detection rate. Performance metrics reveal accuracy values between 95% and 99% for the different types of devices and detection methods analyzed. Additionally, detailed performance analysis shows minimum overhead on the use of the smart grid devices' computing resources (i.e., CPU and memory). On average, memory and CPU utilization does not increase more than 0.03% and 1.9%, respectively.

Contributions: The contributions of this chapter are as follows:

1. We designed a configurable system-level framework that combines system and function call tracing techniques with signal processing and statistical analysis to detect compromised smart grid devices. To the best of our knowledge, this is the first work that utilizes these techniques in detecting compromised devices in the smart grid.

2. To test the efficacy of our framework, we designed a realistic smart grid substation testbed that included both resource-limited and resource-rich devices. These devices followed a GOOSE publisher-subscriber communication model using open-source *libiec*61850 libraries. The implemented testbed represents a valuable configurable benchmark for this, and other research works on CPS security via behavioral analysis.

3. In the adversary model, we considered six different types of compromised devices with different computational resources and attack payloads.

4. We evaluated the performance of our framework by detecting behavioral differences between the compromised device and ground truth devices. We obtained accuracy results over 95% and precision results over 93% for all the different attacks scenarios and types of devices analyzed. These metrics demonstrated

that the detection framework is highly effective to recognize compromised smart grid devices using behavioral analysis.

5. Finally, our analysis shows that the framework does not represent a significant overhead in terms of computing resources.

5.1.1 Differences from Existing Works

Our framework is different from other solutions discussed in Chapter 3 which, in most cases, focus on specific threats to the smart grid instead of considering multiple types of threats acting on different type of devices (e.g., resource-rich and resource-limited). As discussed, there are also cases where different approaches are used for the detection of compromised devices and/or monitoring application behavior. Only in a few of these cases, the solution is intended to be applied in the smart grid domain. In addition, to succeed, these solutions need to monitor constantly-changing environments like network traffic and computational systems or need to challenge the devices with specific inputs to study their response, which constitutes a limitation in terms of system overhead, resource utilization, and real-time analysis. Differently, our framework has a simpler model and is lightweight in terms of system overhead while providing excellent detection rate of the compromised smart grid devices while they are performing typical real-time CPS operations. Also, we introduce a configurable framework for both the supply chain and the smart grid operation field which is envisioned friendly and adaptive enough to be easily applied either within supply chain testing scenarios and while the devices are performing real-time operations inside the smart grid infrastructure. Finally, our detection framework may also complement the existing security mechanisms in the smart grid domain with its open-source and configurable nature.

5.2 Adversary Model

Our adversary model considers, conforming to the NIST guidelines, three possible threats in the smart grid that are directly related to the use of compromised devices [N. 14]:

1. *Threat 1 (Information leakage)*: the compromised device opens additional communication channels to leak valuable smart grid information to the adversary (another untrusted insider or outsider) in real-time.
2. *Threat 2 (Measurement poisoning)*: the compromised device generates fake data that can be used to poison the real status of the smart grid.
3. *Threat 3 (Store-and-send-later)*: the compromised device stores information in hidden files that are recovered later by an attacker.

Based on these three well-defined threats and considering both resource-limited and resource-rich smart grid devices, we further define six different types of compromised devices as part of the adversary model:

1. *Compromised Device 1 (CD_1)*: the resource-limited device creates additional instances of the IEC61850 GOOSE publisher object and starts leaking information through unauthorized communication channels.
2. *Compromised Device 2 (CD_2)*: the resource-limited device allocates small and unauthorized amounts of memory to create fake data and poison real measurements.
3. *Compromised Device 3 (CD_3)*: the resource-limited device creates unauthorized hidden files to store critical information which is retrieved later by the attacker.
4. *Compromised Device 4 (CD_4)*: the resource-rich device creates additional instances of the IEC61850 GOOSE subscriber object and starts leaking information through unauthorized communication channels.

5. *Compromised Device 5 (CD_5)*: the resource-rich device allocates small and unauthorized amounts of memory to create fake data and poison real measurements.

6. *Compromised Device 6 (CD_6)*: the resource-rich device creates unauthorized hidden files to store critical information which is retrieved later by the attacker.

A summary of the adversary model, its impact on device resources, and the targeted security services of such attacks in the smart grid infrastructure are given in Table 5.1.

Adversary Model			
Name	*CPS Device resource availability*	*Computing resources impacted*	*Security services compromised*
CD_1	Limited	Memory, CPU, communications	Confidentiality
CD_2	Limited	Memory, CPU	Integrity
CD_3	Limited	Memory, CPU	Authenticity, confidentiality
CD_4	Rich	Memory, CPU, communications	Confidentiality
CD_5	Rich	Memory, CPU	Integrity
CD_6	Rich	Memory, CPU	Authenticity, confidentiality

Table 5.1: Threats to the smart grid devices assumed in this dissertation.

We also assume that the compromised devices perpetrate their attacks following a Poisson distribution. Poisson allows for randomly and efficiently spacing the attacks and constitutes a valid model to emulate the randomness of such events [Ros01].

The behavior of the compromised device is modeled as follows. Consider t=$[0, T]$, the communication interval between the two smart grid devices. The probability of having an attack from a compromised device $CD_i \in \{CD_1, CD_2, CD_3, CD_4, CD_5, CD_6\}$ can be expressed as:

$$P_{cd} = \frac{\lambda^k e^{-\lambda}}{k!}, \quad k \in \mathbb{R}, \tag{5.1}$$

where λ is the average number of attacks in the interval of time t and k is the total number of attacks in the same interval.

5.3 Overview of the Detection Framework

In this section, we describe the framework to detect compromised devices in the smart grid. Also, we present the details of our detection approaches and decision algorithm. Figure 5.1 depicts the general architecture of the framework. As discussed before, the main goal of the framework is to decide if an unknown smart grid device is genuine or compromised [L. 17]. In this chapter, the term *unknown* refers to the level of uncertainty regarding the smart grid device being compromised or not. Initially, as part of the *learning process*, a ground-truth device from a specific device class is evaluated to generate its corresponding device-class signature or Ground-Truth Profile (GTP). This signature contains behavioral profiling information from the device and is utilized to decide whether an unknown device from the same class is genuine or compromised. Once the signature is obtained, it is stored in the *Ground-Truth Profiles Database*. In our implementation, we define a separate service to execute the learning process. Such a service separation permits the generation of new signatures every time that new devices join the network. Also, an independent learning process guarantees the replacement of old signatures every time that known devices assume new roles in the smart grid network.

The second part of the framework (also known as *detection process*) starts by extracting a similar profiling signature from the unknown device. Here, we assume that we have enough information to classify the device into some specific device class. Then, three different detection methods are applied to compare and correlate the unknown signature to the corresponding GTP from a similar device class stored in the database. Comparison and correlation results are then used to remove uncertainty and decide if the unknown device has been compromised or not.

We envision the detection framework as a secured, centralized, and supervised agent virtually located inside the smart grid network. There are several advantages

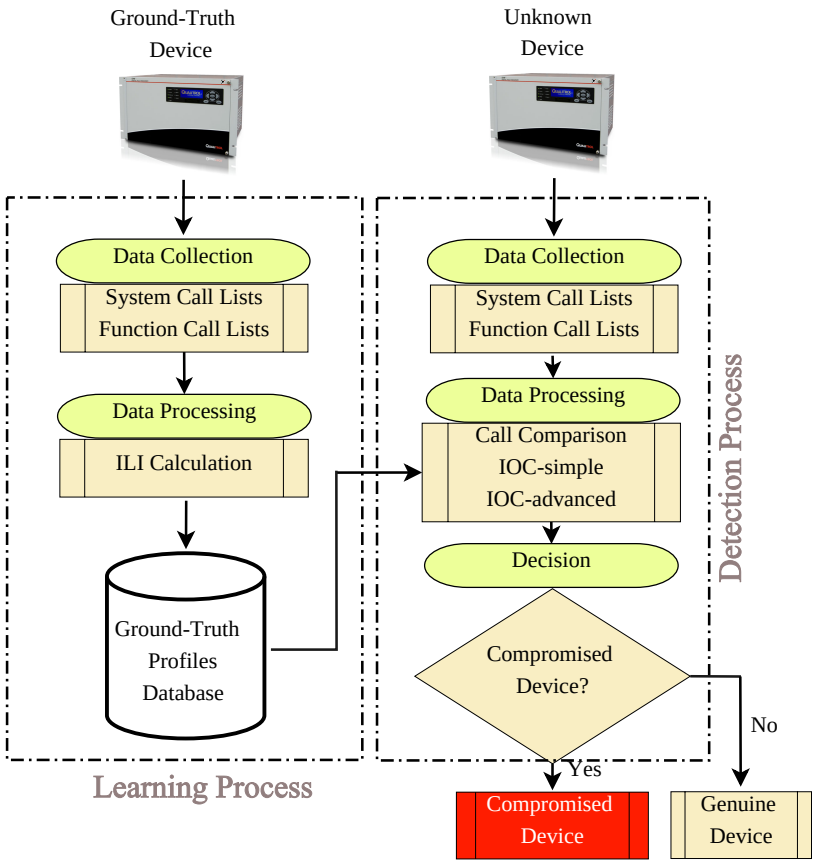


Figure 5.1: Configurable framework introduced to monitor and detect compromised smart grid devices. The learning process creates signatures based on ground-truth devices that are utilized later to decide on potentially-compromised devices.

from this implementation model; first, our framework would be compliant with the security challenges of the smart grid [HCG17b, HCG17a, JRSK17]; second, a centralized solution represents a better option to monitor remotely-located devices from different networks; and third, a supervised agent allows for monitoring group of devices without degrading or interrupting critical tasks inside the smart grid. Figure 5.2 depicts a simplified implementation example of our framework. Here, IED devices exchange information between different substation level networks while a detection agent is monitoring them. Inside the devices, a lightweight scheduler (sch) runs parallel processes at the kernel level to hook into the devices' tasks and extract behavioral information. The collected information is sent to the server along with specific device class

information using secure TCP-IEC61850 channels via either proxy or VPN-tunnel protected (depending on the smart grid device capability). Then, on the server-side, every scheduled action is processed using either priority or first-in-first-out (FIFO)-based queues. The priority is assigned depending on the device class and may also regulate the frequency of the scheduler's execution. For every detection process, the server executes queries to the GTP database using the device class ID and receiving the corresponding behavioral signature. Finally, the server correlates the scheduler data with the stored signature and decides on the devices as being compromised or not.

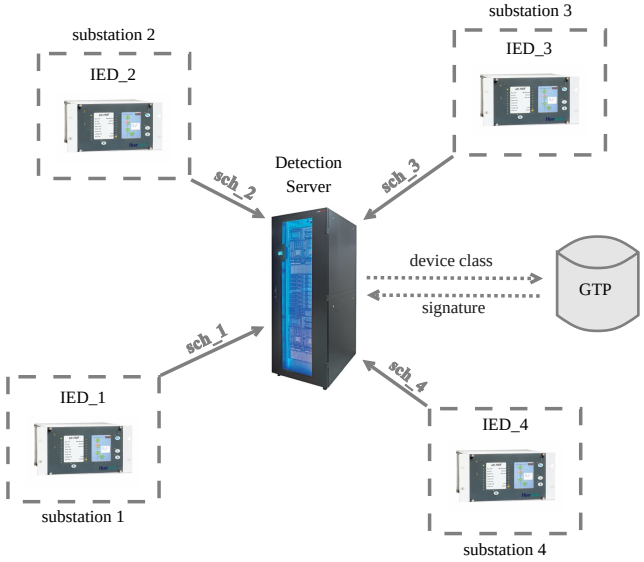


Figure 5.2: Example implementation of the framework to detect compromised CPS devices.

5.3.1 Probability of Detecting a Compromised Device

In Section 5.2, we presented the probability of having a specific attack during a time interval t, considering the device is compromised. In this section, we formally describe the probability of detecting such attacks by using the described approach. To generalize, we consider that the statistical relationship between the two discrete

random variables X and Y that represent the ground-truth signatures and the timed operation of the unknown smart grid devices follow a bi-variate distribution B. From here, we assume that the probability of having a particular specific sequence of calls in the GTP is $P(X)$. The same way, we assume a specific sequence of calls extracted from the unknown device with probability $P(Y)$.

When an attack occurs, and it is detected, the expected value $E(X)$ and $E(Y)$ of the random variables representing both the GTP and the unknown device call list are $P(X)$ and $P(Y)$, respectively. From here, we can determine the variance V of the attack indicator (an attack indicator is represented by the value of the random variable that would indicate the presence of an attack) ϕ_x and ϕ_y from both the GTP and the unknown call lists as:

$$Var(\phi_X) = E(\phi_X^2) - E(\phi_X)^2 = P(X)(1 - P(X)), \tag{5.2}$$

$$Var(\phi_Y) = E(\phi_Y^2) - E(\phi_Y)^2 = P(Y)(1 - P(Y)). \tag{5.3}$$

We directly establish the statistical correlation between the random variable X and Y as the co-variance of these attack indicators:

$$\begin{aligned} Cov(\phi_X, \phi_Y) &= E(\phi_X \phi_Y) - E(\phi_X)E(\phi_Y), \\ &= E(\phi_{X \cap Y}) - E(\phi_X)E(\phi_Y), \\ &= P(X \cap Y) - P(X)P(Y). \end{aligned} \tag{5.4}$$

Then, we can define the correlation between ground-truth device signatures and the unknown smart grid devices based on the probability of detecting the attacks.

$$\rho(\phi_X, \phi_Y) = \frac{P(X \cap Y) - P(X)P(Y)}{\sqrt{P(X)(1-P(X))P(Y)(1-P(Y))}}, \\ = \frac{(P(X|Y) - 1)P(Y)}{\sqrt{P(X)(1-P(X))P(Y)(1-P(Y))}}, \tag{5.5}$$

where $P(X|Y)$ represent the conditional probability of detecting an attack on a smart grid device after assuming a ground-truth signature from the same device class has been found. In general, we describe the successfulness of the introduced detection approach to be the jointly bi-variate variable (X_i, Y_j) with probability of occurrence $P(X_i > X_j | Y_i > Y_j)$ for any pair of calls i, j.

5.3.2 Learning Process

The primary goal of the learning process is to populate the Ground-Truth Profiles Database that contains all the GTPs from device classes in a specific smart grid network region. The execution of the learning process solves the first two architectural challenges of our framework described in Chapter 2 (Section 2.3.4). The learning process classifies the ground-truth devices into device classes and keeps the GTP database up-to-date. For every different class of devices, the learning process performs two specific tasks: (1) GTP data collection and (2) GTP data processing.

GTP Data Collection– this stage applies library interposition and ptrace tools to extract the lists of system and function calls, respectively. The calls are extracted while the ground-truth devices execute regular smart grid substation tasks T. As a result, for every iteration of T, the learning process generates new lists of system and function calls from the ground-truth device. In the end, the data collection process generates a set of system and function call lists. Every list contains detailed information about the specific operations that the devices executed at both the kernel and the user level in every different run of T.

GTP Data Processing– the data processing stage calculates the ILIs for every different ground-truth device class. The concept of ILI introduced in Chapter 2 (Section 2.3.2) evaluates how much deterministic the performance of a ground-truth device is over time. The more deterministic, the higher the ILI value and the more suitable the ground-truth device is to obtain its GTP. In total, the framework calculates two different values of ILIs, one from the set of system call lists and one from the set of function call lists, respectively. To successfully calculate the ILIs, the framework assigns a different weight δ_i to every different type of system or function call in the order that they appear. The assignment of δ_i weights constitutes another configurable feature of our framework. This can be done randomly (the weights are considered normally distributed for simple processes where the different system or function calls have the same level of impact on the completion of the task T) or by following a specific assignment criterion (adaptive assignment). The adaptive assignment depends on the importance of the specific calls and the type of application that is being evaluated. As a result of the assignment step, the framework generates a random variable R that takes values between δ_{min} and δ_{max}. This variable describes the behavior of O for every different system or function call list. Finally, the framework calculates the ILIs using the Equation 2.4. In the end, the ILI values are compared against a configurable threshold σ. Initially, the framework selects an initial value for the threshold based on the device class, and then it continues adjusting this value until the average performance reaches the desired target value for that specific class. If both ILI values are above σ, the GTP is accepted and stored in the database.

Equation 5.6 represents the general format of the GTP used in our framework. The final profile contains information about the device class (DeC), the entire set of system and function call lists (SCL), and the threshold σ. At the end of the learning

process, the Ground-Truth Profiles Database contains all the possible signatures that characterize the different device classes within a specific smart grid network region.

$$GTP = \{DeC, SCL, \sigma\}, \tag{5.6}$$

5.3.3 Detection Process

The main goal of the detection process is to use the profile information stored in the Ground-Truth Profiles Database to determine if the unknown devices are being compromised or not. This process performs three main tasks:

1. *Data collection*: this step follows almost the same sequence of operations detailed in the learning process. However, this time, the framework obtains the call lists from a single execution of T on the unknown devices from the smart grid networks. No T task is fixed for computing purposes nor is repeated over time.

2. *Data processing*: constitutes the core of the detection process. In this step, the framework combines three different detection mechanisms to detect compromised devices. The application of every detection approach is decided on-demand, which has a positive impact on reducing the total overhead introduced by our framework.

3. *Decision*: finally, the decision algorithm processes the results from data collection and processing to decide if the unknown device is genuine or compromised.

In the following, we provide details about the three detection mechanisms.

Detection mechanisms

Our framework implements three different detection mechanisms. To utilize computational resources efficiently, the detection mechanisms are applied orderly on-demand. That means, our framework utilizes each detection approach in an ordered fashion, and it always uses the best effort to make a final decision by applying the minimum number of detection steps.

System and Function call list comparison– The simplest detection approach directly compares the SCL from the GTP to the system and function call lists extracted from the unknown device. The comparison schema considers the type and amount of system and function calls in both GTP and the newly extracted lists. This mechanism is implemented, as shown in Equation 5.7. Specifically, the comparison approach generates a *call vector* that contains the total number of different calls extracted from the unknown device of class c and normalized against the term $GTP(c; SCL)$. Equation 5.7 details this process:

$$call\_vector = \{\frac{unkc_0}{GTP_{SCL_0}}, \frac{unkc_1}{GTP_{SCL_1}}, ..., \frac{unkc_n}{GTP_{SCL_n}}\}, \tag{5.7}$$

where the term $unkc_0$ refers to the amount of system or function calls of type 0 extracted from the unknown device and the term GTP_{SCL_0} refers to the amount of system or function calls of type 0 extracted from the GTP of the same device class. As inferred from Equation 5.7, call vector's items of value 0 represent types of calls that are present in the GTP SCL but not in the lists of calls acquired from the unknown device. On the contrary, call vector's items of value ∞, represent calls extracted from the unknown device, but that cannot be found in the corresponding GTP. In general, the execution of this first detection approach is very light in terms of computing resources.

Index of Correlation Simple– A second detection mechanism calculates the statistical correlation between call lists from the unknown device and the GTP. In this case, in addition to the type and amount of calls, the framework considers the order in which these calls are being triggered. The result from calculating such statistical correlation is known as *Index of Correlation simple* (IOC-simple). IOC-simple is similar to the ILI value obtained during the learning process. The main difference between both is that IOC-simple first determines the statistical correlation between call lists from the unknown device and the corresponding GTP class. Here, an assignment criterion is also used to convert calls into specific δ_i values.

$$IOC - simple_{O_{GTP},O_{unk}} = \frac{\sum o_{GTP} o_{unk} - n\overline{o_{GTP} o_{unk}}}{n s_{o_{GTP}} s_{o_{unk}}}, \tag{5.8}$$

where o_{GTP} represents the set of individual calls in the GTP and o_{unk} represents the set of individual calls extracted from the unknown device.

Index of Correlation Advanced– As mentioned in Chapter 2 (Section 2.3.2), one should not expect smart grid devices to perform operations in a completely deterministic pattern. This limitation exposes the third architectural challenge of our framework (see Section 2.3.4) since legitimate random operations can be mistaken as compromised behavior. To overcome this constraint, we further apply a more advanced IOC calculation (IOC-advanced). In IOC-advanced, our framework combines the values from O_{i_unk} to O_{i+h_unk} in O_{unk}. This operation results in a new random vector O'_{unk} smaller in size and with a lower random component. The index h represents the number of individual calls from the original list that are combined to create the new set O'_{unk}. This index value h is proportional to the amount of randomness that one intends to remove from the original O_{unk} and constitutes another configurable parameter in our framework.

Algorithm 3: Steps for the detection and decision processes.

1: $compromised \leftarrow 0$
2: $UNK(DeC_{unk}, SCL_{unk}) \leftarrow$ unknown device profile
3: $GTP(DeC_{gtp}, SCL, \sigma) \leftarrow$ GTPs from Database
Detection:
4: **if** Exists DeC_{gtp} & $DeC_{unk} == DeC_{gtp}$ **then**
5: Calculate IOC
6: **end if**
Decision:
7: **if** $IOC < \beta$ **then**
8: $compromised \leftarrow 1$
9: **end if**

5.3.4 Decision Process

The final step of our framework is the decision process. In this step, our framework compares results from the three detection mechanisms against a threshold β to decide if the unknown smart grid device is compromised or not. The value of β depends on the device class, and it is always a function of the threshold σ determined during the training process and stored in the GTP. The relationship between σ and β values depends on the targeted accuracy performance for every device class. In general, for devices with a higher deterministic behavioral pattern, a higher value of β is recommended. This design approach reduces the chances of false negatives during the decision process. On the other hand, for devices with lower deterministic behavior, a lower value of β may be sufficient to reduce false negatives. Finally, note that this decision threshold is also configurable. The initial value of β for every device class can be adjusted to an optimal in real-time and while the framework monitors the devices in the field. In the next section, we analyze practical values of β for different types of device classes.

Finally, Algorithm 3 details the detection-decision process of the framework. In lines 1, 2, and 3 the variables *compromised*, UNK, and GTP are initialized with 0, the profile of the unknown device, and all the signatures from the database, respec-

tively. Then, if a signature of the unknown device's class exists (Line 4), the values IOC (simple and advanced) are calculated in Line 5. Finally, if the value of IOC is lower than the threshold β, the device is decided as *compromised*.

5.4 Performance Analysis and Discussion

In this section, we analyze the performance of the detection framework. In all the cases, we obtain the results after averaging 30 different runs of all the covered scenarios. The scenarios include six different types of attackers as a result of the combination of three different threats and two different types of devices based on their computational resources, as described in Section 5.2. Also, we assume that the devices are correctly grouped based on their type. Moreover, we measure the accuracy of our framework with accuracy, precision, recall, and specificity metrics. Finally, we evaluate the performance of the framework in terms of its overhead (e.g., CPU utilization, memory usage, and execution time).

5.4.1 Evaluation with a Realistic Smart Grid Testbed

Our framework considers a realistic scenario from a smart grid substation. The testbed's configuration includes a publisher-subscriber two-way communications configuration which sends and receives IEC61850-compliant GOOSE messages [C. 13]. For this purpose, we utilize an open-source version of IEC61850 [M. 16] protocol running on Linux-based systems. Our resource-limited devices (i.e., GOOSE publishers) run on a Raspberry Pi 2B, using Advanced RISC Machine (ARM) 32 bits architecture with limited memory and CPU. On the other hand, the resource-rich devices (i.e., GOOSE subscribers) run on a Linux Ubuntu 14.04 system with a more powerful CPU and higher memory configuration. Finally, we utilize two different hooking techniques:

ptrace (that performs function call tracing) and library interposition (that performs system call tracing). In our configuration, the publishers open the communication session and wait for the subscribers to connect. Once the devices create and open the communication sockets, the publishers start sending GOOSE messages to the subscribers every one second for a total time interval t of 60 seconds. After the t seconds, the devices close their communication channels. For every compromised device, the malicious threat is active n times during the communication sessions as described in the adversary model (see Section 5.2). Finally, as detailed in Table 5.1, compromised devices CD_1, CD_2, and CD_3 correspond to resource-limited devices of any class that have been compromised with Threats 1, 2, and 3 respectively (see Section 5.2) and compromised devices CD_4, CD_5, and CD_6 correspond to resource-rich devices of any class that have been compromised with Threats 1, 2, and 3, respectively. Despite that the initial application of our testbed was intended to evaluate the performance of the detection framework in realistic scenarios, we believe that, due to its open-source and configurable nature, it can also be used as a benchmark to effectively evaluate the performance of other security tools applied to the smart grid.

5.4.2 Detection Performance

In the following, we detail the performance of our framework after applying the three detection mechanisms introduced in Section 5.3.

System and Function call lists comparison

Tables 5.2 and 5.3 summarize some of the system and function calls captured from the resource-limited and the resource-rich devices, respectively. Columns *Genuine* and CD_i (i: 1 to 6) in both tables list the average rate of the system and function calls normalized against the GTP for genuine and compromised devices, respectively.

Call Tracing Technique	Type of Call	Genuine	CD_1	CD_2	CD_3
ptrace	*brk*	∼1	∼1	6.7	∼1
	clone	∼1	12.5	∼1	∼1
	close	∼1	∼1	∼1	3.2
	*fstat*64	∼1	∼1	∼1	8.8
	lseek	∼1	∼1	∼1	∼1
	*mmap*2	∼1	2.4	4.4	2.4
	mprotect	∼1	2.8	1.1	1
	munmap	∼1	∼1	2	13
	open	∼1	∼1	∼1	5
	rt_sigprocmask	∼1	8.7	0.3	0.3
	rt_sigaction	∼1	∼1	3	3
Interposition	*close*	∼1	∼1	∼1	∼1
	free	∼1	3.2	∼1	∼1
	malloc	∼1	3.3	∼1	∼1
	memcpy	∼1	∼1	∼1	∼1
	memset	∼1	∼1	∼1	∼1
	mmap	∼1	12.5	∼1	∼1
	mprotect	∼1	12.5	∼1	∼1
	pthread_create	∼1	12.5	∼1	∼1
	sendto	∼1	4.3	∼1	∼1
	signal	∼1	24	∼1	∼1
	socket	∼1	∼1	∼1	∼1
	usleep	∼1	3.5	∼1	∼1

Table 5.2: Normalized rate of the system and function calls captured after using our framework to detect compromised resource-limited devices (e.g., RTUs, PLCs): calls due to malicious activities are grayed.

Values greater than ∼1 (marked in gray) in columns CD_1 to CD_3 and CD_4 to CD_6 represents extra system or function call activity due to the presence of malicious operations. That is, extra call activity reveals the presence of malicious activity in the devices. One can notice that, by using ptrace, our framework identified all cases of compromised devices. On the other hand, in the case of library interposition, only CD_1 and CD_4 were properly detected. Also, the reader can notice that in the case of genuine devices, the normalized rate values of system and function calls are very close to 1 in all the cases.

Call Tracing Technique	Type of Call	Genuine	CD_4	CD_5	CD_6
ptrace	brk	$\sim$1	$\sim$1	8.3	$\sim$1
	$clone$	$\sim$1	23	$\sim$1	$\sim$1
	$close$	$\sim$1	6.5	6.8	6.75
	$fstat$	$\sim$1	12	12.5	12.25
	$mmap$	$\sim$1	4.1	6.64	2.6
	$mprotect$	$\sim$1	3.4	$\sim$1	$\sim$1
	$munmap$	$\sim$1	23	26	24
	$open$	$\sim$1	6.5	6.75	6.8
	$rt\_sigaction$	$\sim$1	8.3	$\sim$1	$\sim$1
Interposition	$free$	$\sim$1	15.6	$\sim$1	$\sim$1
	$malloc$	$\sim$1	15.6	$\sim$1	$\sim$1
	$memcpy$	$\sim$1	17.8	$\sim$1	$\sim$1
	$memset$	$\sim$1	24	$\sim$1	$\sim$1
	$mmap$	$\sim$1	24	$\sim$1	$\sim$1
	$mprotect$	$\sim$1	24	$\sim$1	$\sim$1
	$pthread\_create$	$\sim$1	24	$\sim$1	$\sim$1
	$pthread_detach$	$\sim$1	24	$\sim$1	$\sim$1
	$recvfrom$	$\sim$1	15.7	$\sim$1	$\sim$1
	$signal$	$\sim$1	24	$\sim$1	$\sim$1
	$socket$	$\sim$1	24	$\sim$1	$\sim$1
	$usleep$	$\sim$1	15.7	$\sim$1	$\sim$1

Table 5.3: Normalized rate of system and function calls captured after using our framework to detect compromised resource-rich devices (e.g., PMUs, IEDs): calls due to malicious activities are grayed out.

IOC-simple

The first detection approach could not identify Threats 2 and 3 when the framework utilized library interposition. To overcome this limitation, we applied our second detection mechanism, IOC-simple. As explained in Section 5.3, to utilize the framework efficiently, the framework applies the different detection approaches in an ordered fashion as needed.

Figure 5.3(a) shows the results after applying IOC-simple to system and function call lists from GTP and compromised devices. In this figure, *R-R* refers to resource-rich devices, and *R-L* refers to resource-limited devices.

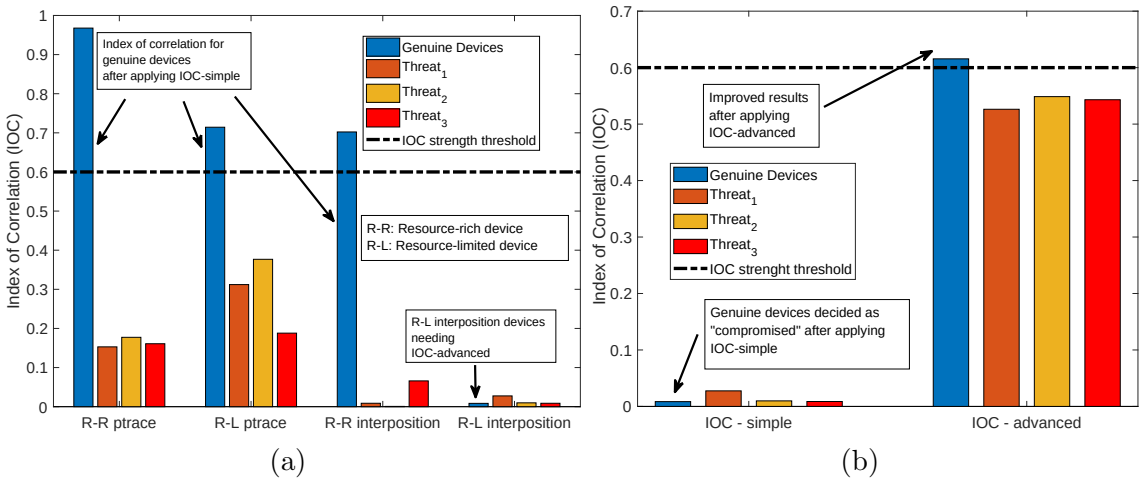


Figure 5.3: Index of Correlation between GTP and unknown devices: (a) Resource-rich and resource-limited devices after applying our IOC-simple and (b) IOC-advanced results comparison between genuine and compromised resource-limited devices (using system call lists from library interposition only).

The reader can observe that, by using ptrace, we obtain low IOC values (in the range of 0.15 to 0.35) between function call lists from GTP and compromised devices. By setting the correlation strength threshold to 0.6 (moderate to high correlation [Ros01]), our framework detects all the cases of the compromised devices. For the case of library interposition, the framework performs very well for resource-rich compromised devices. However, for resource-limited compromised devices IOC-simple under-performs when the framework applies library interposition. In this particular case, IOC-simple from genuine devices falls under the threshold, triggering false positive results. We relate these results to higher random activity in the resource-limited compromised devices' kernel [CGJ15].

IOC-advanced

To overcome the previous limitation, we can apply the IOC-advanced technique. By using this approach, our framework can obtain new call lists with more deterministic behavior from the resource-limited devices and enhance the statistical correlation

between these type of devices and their corresponding GTP. In Figure 5.3(b), the reader can observe how IOC values from resource-limited genuine devices overcome the threshold mark while the compromised devices are still under the borderline. There exists a trade-off between the amount of randomness that can be removed from system call lists without impacting the decision process. If the value of h is too significant, critical behavioral information can also be potentially removed from the call lists, limiting the performance of the decision algorithm in cases where tasks T are too simple.

5.4.3 Performance Metrics

To further measure the efficacy of our detection methods, we calculate the standard performance metrics of accuracy, recall, precision, and specificity. These metrics are defined in Equations 5.9, 5.10, 5.11, and 5.12:

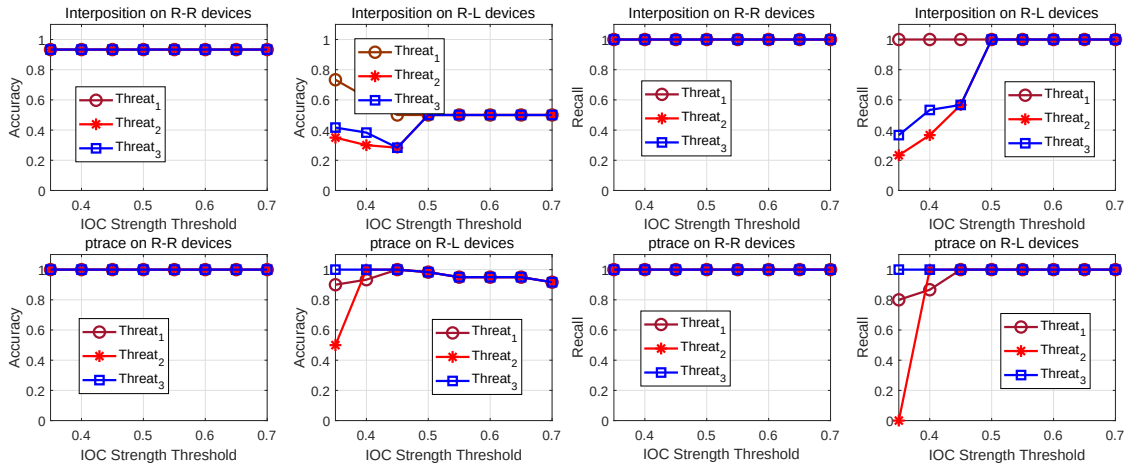
$$A_{CC} = \frac{(T_P + T_N)}{(T_P + T_N + F_P + F_N)}, \tag{5.9}$$

$$R_{EC} = \frac{T_P}{(T_P + F_N)}, \tag{5.10}$$

$$P_{REC} = \frac{T_P}{(T_P + F_P)}, \tag{5.11}$$

$$S_{pec} = \frac{T_N}{(T_N + F_P)}. \tag{5.12}$$

where T_P stands for true positive or the case where a compromised device is decided as compromised; T_N stands for true negative or the case where a genuine device is decided as genuine; F_P stands for false positive or the case where a genuine device is decided as compromised; and finally F_N stands for false negative or the case where a compromised device is decided as genuine. First, we evaluate the performance of our framework with IOC-simple. Then, the improved results are shown after applying IOC-advanced.



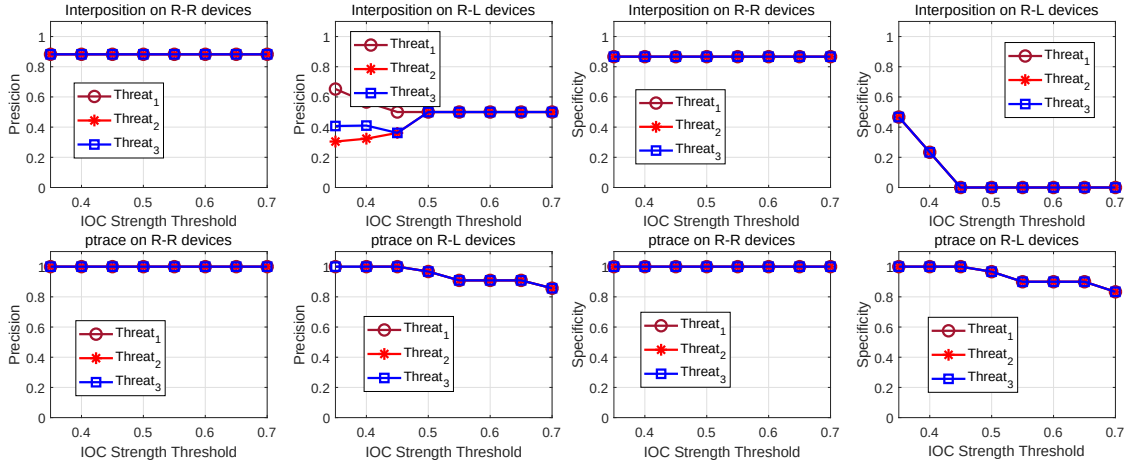
(a) Accuracy (b) Recall

Figure 5.4: Figures compare the performance of the IOC-simple algorithm on six different types of compromised devices after using library interposition and ptrace: (a) Accuracy, (b) Recall.

In Figure 5.4(a), we evaluate the overall accuracy of our detection techniques over the six different types of compromised devices. Since accuracy comprises T_P and T_N results, this metric describes how well the framework can positively decide between genuine and compromised devices without errors. In general, for ptrace, our framework achieves an excellent accuracy performance (between 0.95 to 1) for all types of devices. However, this analysis also reveals the performance limitations of the framework for detecting resource-limited compromised devices in the case of library interposition (top-right case in Figure 5.4(a)). Here, the framework achieves a low accuracy value of 0.5.

In Figure 5.4(b), we evaluate the overall recall performance of the framework. In this case, recall metrics show how well our framework detects the six different types of compromised devices. Based on these results, the reader can observe that the framework achieves the maximum recall (maximum value of T_Ps) for the selected threshold $\beta = 0.6$. In the case of resource-rich devices, recall performance was high for all the threshold values. On the other hand, for resource-limited devices, we can

notice low recall values for Threats 2 and 3 when the threshold values are under 0.6 for the case of library interposition.



(a) Precision (b) Specificity

Figure 5.5: Figures compare the performance of the IOC-simple algorithm on six different types of compromised devices after using library interposition and ptrace: (a) Precision, (b) Specificity.

Figure 5.5(a) depicts the precision evaluation. Precision values represent the statistical relationship between the number of successfully detected compromised devices against the number of times that the framework fails to correctly decide a device as genuine. By looking at the precision results, one can observe that our framework under-performs in the case of library interposition for resource-limited devices.

Finally, we utilize specificity metrics to evaluate the true negative rate, that is, how effectively our framework discriminates genuine devices. In Figure 5.5(b) (top right), one can observe that, for the case of resource-limited devices with library interposition, the framework achieves very low specificity. These results limit the application of IOC-simple to decide on this particular type of devices. Specificity value of 0 at β threshold between 0.45 and 0.7 demonstrates that a device was not correctly decided as genuine in this case. However, in all the remaining three cases, the framework performs very well.

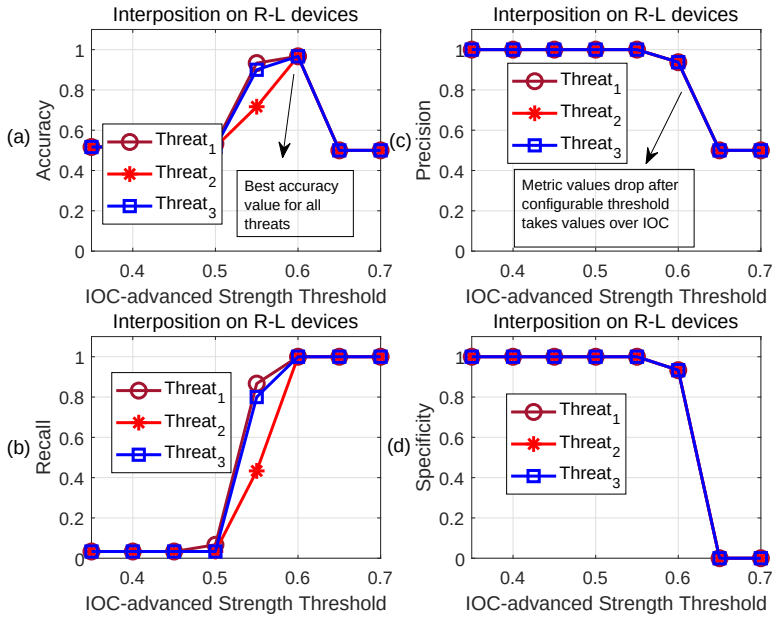


Figure 5.6: Performance metrics after applying IOC-advanced for the detection of resource-limited devices when library interposition is utilized: (a) Accuracy, (b) Recall, (c) Precision, and (d) Specificity.

By analyzing the results in Figures 5.4 and 5.5, one can compare the performance of the detection framework on resource-limited and resource-rich devices for the two hooking techniques applied. Most evaluation metrics diminish their performance when the framework applies the IOC-simple algorithm to detect resource-limited devices using library interposition. These results reflect on the fact that for this type of devices, a more robust detection mechanism is necessary. To improve these results, we utilize the framework with the IOC-advanced algorithm. Figure 5.6 depicts the improvements in all the performance metrics after applying IOC-advanced for compromised resource-limited devices with library interposition. In this figure, one can observe that the correlation threshold of 0.6 provided the best results overall for this particular testbed. Also, the framework obtained significant improvements in accuracy and precision if compared with the case of IOC-simple (accuracy improved from 0.5 to 0.96, and precision improved from 0.5 to 0.93). Finally, recall metrics retained its high performance at the selected threshold value (recall = 1).

5.4.4 System Overhead

We expect our framework to perform with high accuracy and scalability without introducing too much overhead. Table 5.4 summarizes the average of system overhead on resource-limited and resource-rich devices. The metrics RT, ST, UT, Mem, and CPU correspond to the values of real-time, system-time, user-time, memory, and CPU, respectively. In this table, *NF* (No Framework) represents the case where devices were evaluated without utilizing the framework, and *WF* (With Framework) represents the cases where we evaluated the performance while utilizing the framework. Additionally, *LI* represents the cases where we applied library interposition. Finally, *R-R* refers to resource-rich devices, and *R-L* refers to resource-limited devices. Results in Table 5.4 demonstrate that the utilization of the detection framework does not introduce significant overhead on the devices. Particularly, in the case of resource-limited devices, the framework utilizes 0.03% more of memory (out of the total memory available in the devices) and 1.9% more of the CPU. For resource-rich devices, the framework utilizes 0.001% more of memory (out of the total memory available on the device) and an almost negligible amount of CPU. In summary, for both resource-limited and resource-rich devices, library interposition introduces the most overhead to the system. However, this overhead is considerably low if compared with similar applications proposed in the literature [SBDB01, FKF+03].

Metrics	*NF*		*WF*			
	value	value	*ptrace (%)*		*LI (%)*	
	R-R	*R-L*	*R-R*	*R-L*	*R-R*	*R-L*
RT (s)	60.00	60.11	0.05	3.8	0.01	0.1
ST (s)	0.49	3.60	8.1	3.6	10.2	5.5
UT (s)	0.31	0.49	16.1	0.31	6.4	2.0
Mem (KB)	1967.5	1827.5	1.1e-3	4.3e-5	3.0e-2	1.0e-3
CPU (%)	1	6.02	0	1.9	0	1

Table 5.4: Average system overhead on resource-rich and resource-limited devices after using the framework.

To further study the impact of our framework, we analyzed this overhead considering a real resource-limited smart grid device. In Table 5.5, we summarize the main specifications of Remote Terminal Unit RT2020.

Item	*Specification Values*
Processor	Dual Core ARM A9 667 MHz
Dynamic Memory (RAM)	128 MB
Program Memory (Flash)	4 MB
Nonvolatile Memory	4 Mb
Real Time Clock Resolution	1 ms
Execution Cycle Time	$\leq$ 100 ms

Table 5.5: Specification values for Remote Terminal Unit RT2020 [Hon14].

Looking at Table 5.5, we can conclude that for the worst case of resource utilization (library interposition on a resource-limited device), the increment in execution time because of the use of our framework would only represent up to 2.3 cycle times. Additionally, our framework would only take 0.1% of the total memory of a real resource-limited smart grid device.

5.4.5 Benefits and Features

There are several benefits associated with the design of our framework:

1. *Excellent detection rate*: the framework demonstrated an excellent rate for the detection of compromised smart grid devices by combining three different detection methods: system and function call comparison, IOC-simple, and IOC-advanced.

2. *Minimum overhead*: the detection framework does not represent significant overhead on the use of computing resources.

3. *Specific vs. generic solution*: the framework is designed to address the specific problem of compromised smart grid device detection. The adversary and system

model introduced in this chapter follow the security requirements and architecture characteristics of the smart grid. However, the approaches introduced here for the detection of compromised smart grid devices are perfectly suitable for other CPS security domains outside the smart grid domain.

4. *Comprehensive adversary model*: the adversary model used in this chapter considers both resource-limited and resource-rich compromised devices. Also, it combines three different threats affecting the smart grid.

5. *Compromised device diversity*: Our framework is suitable for a great range of different compromised devices. The design of our system-level framework makes it also suitable for detecting hardware counterfeiting [GFT13, Ch.15, A. 15] as observed from the system level. System and function call comparison and statistical techniques are powerful tools capable of detecting changes in hardware and system configuration. This makes our framework an appealing solution to monitor and detect a wide range of different types of compromised devices.

5.5 Conclusion

The smart grid vision depends on the secure and reliable two-way communications between smart devices (e.g., IEDs, PLCs, PMUs). Nonetheless, compromised smart grid devices constitute a serious threat to a healthy and secure distribution of data in the grid. In this chapter, we reviewed a system-level configurable framework capable of monitoring and detecting compromised smart grid devices. Our framework combines system and function call tracing techniques (i.e., ptrace, library interposition), signal processing, and statistical analysis (basic and advanced) to detect compromised device behavior. To the best of our knowledge, this is the first work that utilizes

these techniques in detecting compromised devices in the smart grid. Moreover, we evaluated the performance of our framework on six different types of compromised devices, conforming to realistic smart grid scenarios. Such devices exchanged smart grid GOOSE messages utilizing an open-source version of the IEC61850 protocol suite. Specifically, we analyzed the efficacy of our framework under six different adversarial settings affecting devices with different resource availability. Experimental results demonstrated that our framework successfully detects different types of compromised device behavior in a variety of different environments with high accuracy. Also, our performance analysis reveals that the use of the detection framework yield minimal overhead on the smart grid devices' computing resources.

CHAPTER 6

SECURITY AND PRIVACY ANALYSIS OF RESOURCE-LIMITED DEVICE APPLICATIONS

6.1 Introduction

In this chapter, we introduce IoTWatcH, a novel dynamic analysis tool that uncovers the privacy risks of IoT apps at runtime. We designed and built IoTWatcH based on an IoT privacy survey of human subjects that use different IoT devices. The survey aimed at understanding the privacy concerns and expectations of users when they use IoT devices and apps. IoTWatcH provides users with a simple interface that allows them to specify their privacy preferences (e.g., location, device states, etc.) at install-time. It then adds extra logic to the app's source code to collect app information at runtime. The collected information is used to classify the data sent out of the IoT app into privacy preferences of users through Natural Language Processing (NLP) techniques. Also, IoTWatcH analyzes the recipients of the data and detects leaks to unauthorized parties. Finally, IoTWatcH notifies the users about the sensitive data-leaks and privacy concerns in IoT apps, allowing them to make informed decisions about their privacy. Privacy concerns include any app behavior that would put the sensitive information at risk. For instance, an Internet communication that sends the sensitive information to a remote server in plain text.

To evaluate IoTWatcH, we trained an NLP model with IoT strings extracted from 380 SmartThings market apps. The model is used to classify IoT app strings (e.g., "the door is locked", and "kitchen lights are turned off") to user privacy preferences. Then, we analyzed 160 different IoT apps to evaluate its accuracy at runtime. IoTWatcH successfully classified 146 IoT strings to privacy preferences with an average accuracy of 94.25% and precision of 95%. Among its findings, IoTWatcH also

flags 35 IoT apps that leak sensitive data to unauthorized recipients. Finally, IoTWatch yields minimal overhead to the IoT apps execution, introducing on average 105 ms additional latency.

Summary of Contributions. The contributions of this chapter are as follows:

- We conducted an IoT privacy survey with 123 IoT users through a set of structured questions. Although the survey is not the major focus of this manuscript, it is instrumental for the IoT community, researchers, and users to understand users' privacy concerns, their privacy preferences, and expectations when they use IoT devices and apps.
- We designed and built IoTWatch, a dynamic privacy analysis tool for IoT apps. IoTWatch adds extra capabilities to an IoT app's source code to provide users with a privacy interface and collect app data. The interface allows users to easily specify their privacy preferences such as device states and location. Also, it permits the monitoring of data leaks and privacy behaviors in IoT apps at runtime. Lastly, IoTWatch informs the users when an app's leak matches with the privacy preference of a user. We made the IoTWatch freely available to the community at `https://iotwatch.appspot.com/`.
- We analyzed 540 current IoT apps during the implementation and evaluation of IoTWatch. First, we trained IoTWatch with 380 apps. Then we evaluated its accuracy on the remaining 160 IoT apps (120 market and 40 malicious IoT apps). IoTWatch classifies 146 IoT strings that are sent out of the apps into privacy labels with 94.25% accuracy. Additionally, it successfully flags 62 sensitive leakages (29 via messaging and 33 via Internet communication) to unauthorized parties in 35 IoT apps. IoTWatch detects sensitive data leaks without significant overhead, introducing on average 105 ms latency to an app's execution.

6.1.1 Differences from Existing Works

IoTWatcH is a novel dynamic tool that uncovers privacy risks of IoT apps. Considering the privacy expectations of IoT users, IoTWatcH instruments IoT apps to collect and analyze the data sent to external parties in real time. IoTWatcH performs rich NLP-based classification of IoT strings to four user-friendly privacy labels, which are also customizable. Also, IoTWatcH analyzes the recipients of the sensitive information to uncover data leaks and privacy behaviors. Finally, IoTWatcH implements notification mechanisms to inform its findings to the user.

6.2 IoT Privacy Survey

We conducted an IoT privacy survey to understand the privacy concerns of IoT users when they use various IoT devices and apps. Although the survey is not the major focus of this manuscript, it provides rich insights into the users experiences and expectations on control over IoT app permissions, IoT privacy nudges, and their interplay. The entire survey was authorized by the institutional ethics review board (IRB) and occurred between April 2019 and May 2019.

Privacy Survey Goals. We aim to answer the following questions: (1) what are the privacy concerns of IoT users?, (2) is there a need for privacy analysis tools designed for IoT?, and (3) what are the user expectations, in terms of usability requirements, for privacy analysis tools?

We created 26 different questions organized into three categories. These categories align with three specific privacy survey goals: (1) the characterization of the participants, (2) privacy concerns of IoT users, and (3) the need for IoT privacy analysis tools and their usability requirements. We provide the details of the questions in Appendix 6.2.3) and present the profiles of participants and our key findings below.

Survey Overview and Recruitment. We made the survey available to participants for four weeks. The users could access the survey and submit their responses via an online questionnaire hosted on Google Forms [Gooc]. The questionnaire included single choice questions (e.g., yes, no), multiple-choice questions, and free-form questions (detailed in Appendix 6.2.3). We made all the questions required except for the ones requesting an additional explanation from users in the form of free-text input. Finally, we recruited the participants using recruitment emails sent to lists of students, faculty, and staff in our institutions. The emails included a brief explanation about the survey and link to the online form.

Participant Characteristics. We recruited 123 participants of which 69 participants (56.1%) were in the range of 18-25 years old and 37 (30.1%) were in the range of 26-35 years. The remaining 17 (13.8%) participants were 36 years or older. The majority of the participants (110 (89%)) had at least completed some bachelor-level courses and 37 (30%) were enrolled in graduate-level courses. A total of 112 (91.05%) users shared that they currently use or are planning to use IoT devices in their homes. Finally, 19 (15.4%) participants knew how to develop their own IoT apps while 82 (66.7%) participants had previous experience installing apps from an IoT market or via using the source code of IoT apps available online.

Ethics and Analysis. The human subjects review board of our institutions approved the privacy survey. The participants had to be over 18 years old to participate. The survey did not collect any personal information from participants, other than an institutional email address that was requested for compensation purposes. We did not allow participants to submit multiple responses, but they had the chance to change their answers anytime before the survey closing date. We processed and accepted all the responses obtained from the participants. Further, we directly quantified the responses from single- and multiple-choice questions. Finally, we used two

independent researchers to analyze the free-from responses and did not consider any answers flagged as potential outliers.

Compensation. After the survey's closing date, every participant was compensated either with extra-credit in their coursework or a gift card with a monetary value. The student participants could opt for receiving extra credit or monetary compensation. Faculty and staff all received gift cards.

6.2.1 Survey Results

Privacy Concerns of IoT Users. The participants were concerned about their private information being inadvertently leaked to unauthorized parties. Specifically, 65 (52.8%) participants felt uncomfortable about their personal data (e.g., their password to login into edge devices), their behavior and habits (e.g., when they go to sleep), location (e.g., whether they are home or away), device's settings (e.g., heating value of a thermostat) and time configuration (e.g., when kids leave home), and device states (e.g., whether the door is locked or not) being handled by IoT systems. Also, at least 89 (72.4%) participants expressed to be aware of IoT apps collecting their sensitive information and sending it to remote servers for data analytics such as profiling their energy usage and for advertisement purposes [ABC+18]. Finally, 103 (83.7%) participants expressed privacy concerns on the use of IoT systems, and 88 (71.5%) mentioned having heard about privacy issues in IoT systems from the news or other media.

The Need for Real-time Privacy Tools. In total, 112 (91.1%) participants raised broad concerns about lack of an existing tool that informs the user regarding potential privacy risks of IoT systems in real-time. Also, 119 (96.74%) participants found the idea of using a tool to uncover privacy risks in IoT highly plausible. Our participants were willing to use automatic tools that modify (i.e., instrument) original IoT apps

Expectations of survey's participants	% Agreement
Real-time privacy analysis	91.6%
Configurable privacy preferences	87.5%
Control over unauthorized data disclosure	86.6%
On-demand privacy controls	81.6%
Timely privacy notifications	85.3%
Inter-rater Reliability	**86.5%**

Table 6.1: Participant responses when asked about their expectations from a privacy analysis tool, which guided the design of IoTWatch. The percentage of agreement among the survey's participants showed a strong inter-rater reliability.

to enable privacy analysis in real-time. Out of the 123 participants, 119 (96.74%) expressed their support to this option if the tool is verified by the IoT platform.

User's Expectations. To understand the characteristics of an easy-to-use privacy tool, we asked participants a set of questions to evaluate their usability expectations. Table 6.1 summarizes the participants' responses. Here, we detail the percentage of agreement among users for every privacy feature and the inter-rate reliability score. Finally, we evaluated the participants' approval of four different privacy labels utilized to classify information accessed by IoT apps. The term "Device-info" was considered appropriate to define information from devices (e.g., device states or device type) by 109 (88.6%) participants, while the label "User-behavior" received 103 (83.7%) positive responses to define information related to the user (e.g., what the user does, how the user configure his/her IoT system). Furthermore, the label "Location" was approved by 110 (89.4%) participants to define information related to the location of devices and users. Lastly, the label "Date-time" was approved in 100 (81.3%) responses to define timing-related information.

6.2.2 Summary of Findings

Our findings shed new light on the need for cooperative privacy management practices between users and IoT markets, which mitigate privacy risks based on user privacy preferences. Table 6.1 presents the needs of the users that align with their control over their privacy preferences. We obtained an inter-rate reliability of 86.5%, which can be considered strong. Here, we summarize the privacy features that guides design and development of IoTWatcH considering participants responses.

Real-time Privacy Analysis. The participants reflected their opinion about being aware of apps privacy behavior in what information leaves an IoT system and where it is transmitted. Additionally, they reported that they expect to have minimal configuration when new devices are dynamically plugged into their IoT systems and new IoT apps are installed.

Configurable Privacy Preferences. The participants mentioned that fears about lack of privacy preference controls limit their willingness to use IoT devices. For instance, they prefer to have categories that define a high-level category that shows the information-type leaving the IoT system, such as "the door is locked" associated with a specific privacy label and "the mode is changed to sleep" with another label.

Unauthorized Data Disclosure. The participants like having better control over the disclosure of any private information. For instance, they prefer to be notified when IoT systems share their data with other parties. Additionally, participants mentioned strategies that notify unencrypted Internet requests or hard-coded messaging recipients, which mitigates the consequences of privacy violations.

On-demand Privacy Controls and Privacy Notifications. The majority of the participants acknowledged the effectiveness of configurable privacy preferences over the IoT apps. They mentioned these configurations help them have a better experience with a few numbers of notifications and minimal runtime delay.

6.2.3 Example IoT Privacy Survey Questions

We present a list of representative IoT privacy survey questions from all the categories. The entire user study can be found at `https://anonymous.com`.

Participant Characterization

1. Do you use, have used, or are you planning to use any IoT device?

   ( ) Yes

   ( ) No

   ( ) Maybe

2. What is your technical experience with IoT apps?

   ( ) I can build, implement, code my own IoT app

   ( ) Installed/can install/configure an IoT app using the source code available online

   ( ) Installed/can install/configure an IoT app's marketplace (Google Play, App Store, etc.)

   ( ) I just know how to press the buttons

   ( ) I have no idea how to deal with IoT apps

Security and Privacy Concerns in Smart Apps

1. What information would you consider sensitive if used in IoT apps/devices? Please check all that apply.

   ( ) My personal data (e.g., email address, phone number, residential address, etc.) ( ) Whatever I do, my behavior (e.g., when I arrive home, when I leave home, I go to sleep, etc.)

   ( ) My (or my devices') location (e.g., my location while using the apps, etc.)

( ) My device settings/the way I configure the devices (e.g., Time of the day the lights turn On/Off, my thermostat temperature settings, etc.)

( ) My (or my devices') timing (e.g., the time passed since I left home, the time passed since I went to sleep, etc.)

( ) Information from my devices (e.g., device type, manufacturer, device IDs, etc.)

( ) Data from my devices (e.g., door state open or close, light on or off, etc.)

( ) Other

If you selected "Other", please explain:

2. Have you heard or personally have privacy concerns on the use of the IoT devices and systems?

   ( ) Big concerns ( ) Some concerns

   ( ) I do not, but I know someone that does

   ( ) Never thought about it, until now

   ( ) I do not care

Privacy Analysis Tools and Features

1. Do you think there is a need for a tool to check for security privacy risks from the smart apps?

   ( ) Yes

   ( ) No

   ( ) Maybe

2. Would you be willing to use available automatic tools that analyze and modify smart apps to enable security and privacy analysis in real-time?

   ( ) Yes

   ( ) No

   ( ) Maybe

6.3 Problem Statement and Threat Model

Problem Statement. We use an example source code abstracted from a `smart-lock-control` app (Figure 6.1) to illustrate the privacy concerns and behaviors in IoT apps. The expected behavior of the app is to lock the door and notify to user-defined contacts that the door is locked when the user leaves the house. At install-time, the user grants permissions to the smart lock, presence sensor, and enters a phone number for messaging notifications (❶). The app subscribes to two event handlers `f1` and `f2` to implement the app functionality. The event handlers are invoked based on the presence sensor's state (user-present and user-not-present) (❷). When the user leaves home, "not-present" event handler (i.e., `f2`) locks the door, sends a message notification, and transmits out the door lock state to a remote server (i.e., http://support.com) (❸). However, the actual behavior of the app adds a piece of code that invokes a function (i.e., `leakinfo()`) sending a *string* that contains "Nobody is Home" to a hard-coded phone number (❹). This string is highly private and informs an adversary that the house is empty. This information can be abused, for instance, to facilitate a burglar to break into the house. This example shows that a user does not have control over what an IoT app does with the sensitive data, who sees it, and what they do with it. Unfortunately, IoT development platforms do not provide users with sufficient information to make informed decisions about their privacy preferences in IoT environments.

```
1:  // smart-lock-control app: "controls the smart lock"
2:  // Permissions Block (1)
3:        Device: smart_lock sl, presence pr
4:        User-defined inputs: phone
5:  // Events Subscription (2)
6:        subscribe(pr, "present", f1)
7:        subscribe(pr, "not present", f2)
8:  //Events Handler (3)
9:        f1(){…}
10:       f2(){  sl.lock()
11:              notifyUser(sl.state, phone)
12:              leakInfo()}
13: // Sends notification to the user
14:       notifyUser(state, number){
15:          sendSMS("Your lock is: " + state, phone)
16:          POST("http://support.com", sl.getLocation())
17: // Leaks sensitive data to attacker (4)
18:       leakInfo(){
19:              sendSMS("Nobody is Home", 123-456-7890)}
20:
```

Figure 6.1: An example IoT app leaking sensitive data to a hard-coded phone number and performing insecure HTTP calls.

Limitation of Existing Privacy Tools. There exist systems to identify sensitive data-flows in IoT apps. For instance, SAINT is a static system that uses taint analysis to identify sensitive data-flows in IoT apps [CBS+18]. FlowFence, a dynamic system, uses quarantined modules to enforce data-flow policies on the use of sensitive data [FPR+16]. However, these approaches are limited in precision and the number of privacy policies enforced. For instance, SAINT and FlowFence have no way of knowing if an app leaks sensitive data through developer- or user-defined strings. For instance, the string "Nobody is home" leaked through `leakinfo()`). Meanwhile, our analysis of 540 IoT market apps showed that 64% of apps potentially leak sensitive data through strings that do not include any tainted data, yet the string is sensitive. Lastly, static systems like SAINT, iRuler [WDY+19], and privacy tools for trigger-action platforms [IFT] fail to detect sensitive data leaks from methods defined dynamically in IoT apps [Met]. Additionally, there are no approaches that allow users to examine their sharing and privacy preferences over individual IoT apps. For instance, a user may desire to share her energy usage data with a third party in a

specific IoT app yet she wants to restrict sharing all other sensitive information with third-parties. This requires a personalized privacy setting for each app that gives control to the users over what to share.

In contrast to previous approaches, IoTWatcH analyzes data-flows at runtime; thus, the flow's content is analyzed to determine whether a data-flow constitutes a privacy concern or not. Such a runtime analysis capability overcomes the limitations of static analysis tools that fail to consider taint variables generated dynamically. Additionally, IoTWatcH provides additional user interfaces that allow users to configure their privacy settings for each app and informs the users about its findings. We provide a detailed comparison of IoTWatcH with other privacy tools for IoT and Android apps in Chapter 3.3.2.

Threat Model and Assumptions. We consider sensitive data leaks in IoT apps through malicious apps or unintentional developer mistakes. We consider sensitive information leaks in IoT apps via messaging and Internet connections, apps that transmit data to the recipients that are not authorized by users, or apps that transmit sensitive data without proper data protection mechanisms implemented. We do not consider safety and security violations in IoT apps [CMT18, CTM19]. Additionally, we do not track data-flows via push notifications or *sink functions* that are authorized by OAuth (e.g., a user authorizes a third-party service through OAuth protocol to share the device states for data visualization).

6.4 Approach Overview

IoTWatcH, a runtime privacy analysis system, collects the information exchanged with external parties in IoT apps to uncover privacy risks. The collected information is used to classify sensitive data into easy-to-understand privacy labels. The findings are checked against users' privacy preferences and users are notified if there is a

```
1:  // iotwatch-enabled smart-lock-control app
2:  Description: "controls the smart lock"
3:  // Permissions Block
4:        Device: smart_lock sl, presence pr
5:        User-defined inputs: phone  C1
6:  // Event Subscription
7:        subscribe(pr, "present", f1)
8:        subscribe(pr, "not present", f2)
9:  //Event Handler
10:       f1(){…}
11:       f2(){ sl.lock()
12:             notifyUser(sl.state, phone)
13:             leakInfo()}
14: // Send notification to the user
15:       notifyUser(state, number){
16:             sendSMS("Your lock is: " + state, phone)  C2
17:             sendPush("App is using data of type:"
18:                   classification)}  C3
19: // Leak sensitive data to attacker
20:       leakInfo(){
21:             sendSMS("Nobody is Home", 123-456-7890)  C4
22:             sendPush("Privacy Risk of type:" context
23:                   "with data of type:" classification)}  C5
```

Figure 6.2: An example of an IoT app instrumented by IoTWatch to support the notification interface.

mismatch. IoTWatch helps users define their privacy preferences and have better control over their sensitive data and maintain better privacy practices.

6.4.1 Understanding Leakage in IoT Apps

We use the `smart-lock-control` app depicted in Figure 6.1 to illustrate the logical steps of IoTWatch (Figure 6.2). The instrumentor first adds extra logic to the app's source code to implement a user interface (UI). The UI allows a user to select a set of privacy labels and the type of potential leakage mechanisms to be tracked by IoTWatch. We provide users with four privacy labels: location, user-behavior, device-info, and date-time, and two potential leakage mechanisms: messaging and Internet, to define their privacy profile. Additionally, the code instrumentor adds extra logic to send app data to IoTWatch server at runtime. The data includes user-defined input variables (i.e., phone) (C1) and the content and recipients of the functions used to send data out of the app. The sample code in the `smart-lock-control` app includes two messaging functions (C2 and C4). IoTWatch instruments both the messaging

functions to collect their content (i.e., "Your lock is: " + state, and "Nobody is Home") and their recipients (i.e., phone and 123-456-7890). Lastly, the instrumentor inserts extra code to enable the notification of IoTWatcH's results to users (C3 and C5).

At install-time, a user configures two pieces of privacy settings for the app based on her privacy preferences: (1) the privacy labels that are of interest of the user and (2) the types of possible leakage methods to be tracked by IoTWatcH. These privacy settings define the user's privacy profile and are used to notify the user when her sensitive data is transmitted out of an app. The user then installs the instrumented app, which transmits its data to the IoTWatcH server once a specified data-flow is flagged. This information enables IoTWatcH to identify the type of sensitive information that the IoT app used and to analyze the information obtained from the app to uncover privacy data leaks and potential privacy behaviors. IoTWatcH implements a novel algorithm to verify whether the sensitive data is sent to recipients defined by the user via matching the user-defined inputs to the data recipients. For instance, for the first potential leakage (C2), the data is sent to a user-defined phone number (C1). For the second possible leakage (C4), the data is sent to a hard-coded phone number, which might indicate a privacy violation to the user. Also, by analyzing the content of these potential leakages with NLP-based techniques, IoTWatcH verifies that the app uses sensitive information regarding the devices (i.e., "Your lock is:") and location (i.e., "Nobody is Home"). Lastly, IoTWatcH informs the user of its findings and generates privacy awareness. For the first possible leakage, IoTWatcH sends a push notification with the privacy labels of the information included in the message (C3). For the second potential leakage, IoTWatcH informs the privacy content of the message and also alerts the user regarding the potential privacy violations of the user's privacy (C5).

6.4.2 Terminology Used

We define some of the terms used to explain IoTWatch's architecture.

Sink-calls. IoT programming platforms define specific APIs to send information out of the apps (i.e., sink-calls) [Smah, Opec] as external data-flows. In this manuscript, we focus on sink-calls of type messaging and Internet. Sink-call methods require two types of information: the *recipient* and the *content*. Specifically, the recipients define where the information contained in the sink-call is being sent to, and the content defines the message or data sent in the form of IoT strings.

Privacy Labels. We define four different privacy labels (i.e., date-time, device-info, location, and user-behavior) to classify IoT strings (i.e., sink-call content) in IoT apps that users can specify. These privacy labels were selected based on the findings and takeaways extracted from a privacy survey as noted in Section 6.2.

Privacy Profile. We consider the collection of privacy labels and notification preferences defined by the IoTWatch's user as a privacy profile. At install-time, the user selects the preferred type of privacy information and communication methods (e.g., messaging, Internet) to be tracked by IoTWatch.

Privacy Leakage. We consider any data that is sent to an external recipient that is not authorized (i.e., defined by the user at install-time) or acknowledged (i.e., informed to the user via the app's description block) by the user of the IoT app as a leakage.

Privacy Behavior. We consider any sink-call that sends data to an authorized recipient, but that potentially puts the sensitive information at risk as a privacy behavior. For instance, Internet communications sending the information to legitimate servers in plain text.

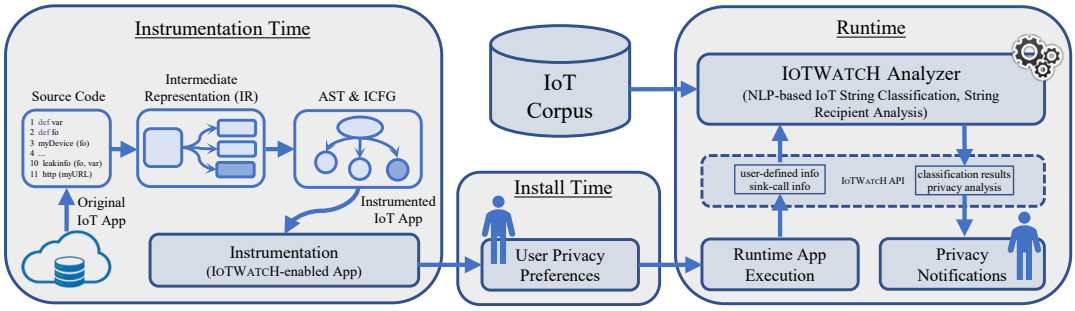


Figure 6.3: Overview of IoTWatch architecture. Three main stages are highlighted: first, IoT apps are modified at instrumentation time to enable IoTWatch; second, the user selects their privacy preference at install time; finally, at runtime, IoTWatch analyzes the IoT app data to uncover privacy risks and behaviors.

6.5 IoTWatcH

Figure 6.3 illustrates IoTWatch's architecture which includes processes performed in three main phases: instrumentation time, install time, and runtime. At instrumentation time, the code instrumentor adds extra logic to the app's source code (1) to implement a privacy interface (Section 6.5.1) where users specify their privacy preferences, and (2) to collect app information required for IoTWatch analysis. At install time, the user defines the type of privacy information they desire to be tracked and notified about by IoTWatch. Finally, at runtime, the app sends its information to IoTWatch server, which classifies the collected data into privacy labels through NLP techniques (Section 6.5.2). In addition, IoTWatch performs analysis on the data recipients (Section 6.5.2) for potential privacy leakages. In the case of sink-calls of type messaging, it matches their recipients to user-defined inputs to check if the apps send sensitive information to unauthorized parties. In the case of Internet communications, it checks whether an app sends information to the remote servers using unencrypted HTTP calls. IoTWatch's classification process is supported by a model implemented from a market IoT corpus. Lastly, IoTWatch informs the user about its findings (Section 6.5.3).

6.5.1 Code Instrumentor

IoTWatcH's code instrumentor analyzes the source code of an original IoT app to build an intermediate representation (IR). The IR allows one to extract the sensor-computation-actuator paradigm of IoT apps [CBS+18, CMT18]. The use of an IR enables the design of generic solutions that can be implemented for different IoT programming platforms (e.g., Smasung SmartThings and OpenHAB) [CBS+18]. Then, the instrumentor extracts the Abstract Syntax Tree (AST) of the app and implements custom *node visitors* to build the Inter-procedural Control Graph (ICFG). The ICFG is used to flag user-defined inputs in the permission block of the app, and the recipients and content of the sink-call functions (i.e., messaging and Internet). Then, the instrumentor adds extra code to collect and transmits this data to the IoTWatcH's server, and to implement push notifications that informs the user about IoTWatcH's findings in real-time. The code instrumentor groups collected data into two different categories: (1) app information and (2) sink-call information. We detail each of them as follows:

App Information. IoTWatcH's code instrumentor visits the permission block (Figure 6.2) in IoT apps and extracts information defined by the user. Specifically, it collects user-defined inputs required to implement the sink-calls (e.g., phone numbers to receive notifications from the app). As we detail in Section 6.5.2, this data is matched with the information directly extracted from sink-calls (i.e., recipients) to detect, for instance, data sent to unauthorized recipients (i.e., not defined by the user), which may lead to potential privacy issues for the user.

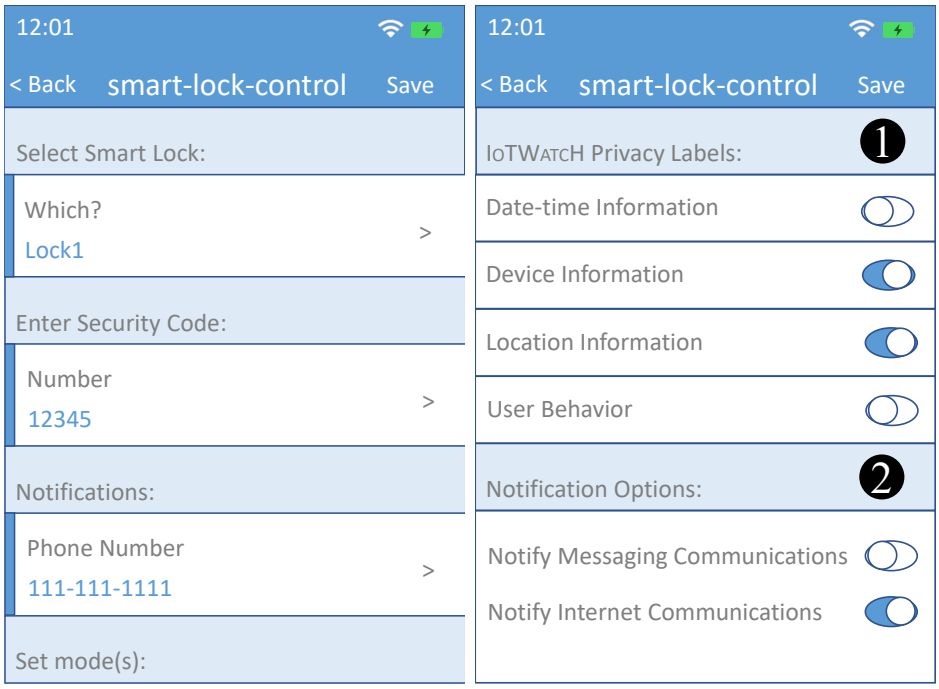
Sink-Call Information. It includes the content and recipients of messaging and Internet communications. For instance, in the app's source code depicted in Figure 6.2, IoTWatcH extracts the content "Your lock is: + *state*" from the first messaging function in Line 16, and the content "Nobody is Home" from the second messaging

function in Line 21. As per the recipients, it extracts the recipient's value contained in the variable *phone* from the first messaging method (Line 16) and the hard-coded phone number "123-456-7890" from the second messaging function (Line 21). IoTWatch uses the content of the sink-calls to make the user aware of the type of sensitive information that IoT apps handle. We use an NLP-based model to analyze the content of external data-flows and classify them into four privacy labels that are easy to understand by the user. Also, it uses the recipient information in IoTWatch analyzer to uncover sensitive data leaks. Our tool matches the recipient information extracted from messaging and Internet communications with both the user-defined and user-acknowledged information at install-time. The user-defined information is entered by the user (Line 5) and the user-acknowledged information is informed by the developer via the app's description block (Line 2) and approved by the user. In cases where the sink-call (i.e., messaging or Internet) is executed using unauthorized recipients, the flow is flagged as *leak*, and the user is informed. With this analysis, IoTWatch creates awareness of sensitive information being disclosed to unauthorized or malicious recipients. Besides, for Internet communications, we verify that the recipient supports data encryption. With this, our tool guarantees that the sensitive information is protected from potential eavesdroppers.

Selective App Instrumentation

In addition to collecting app information, IoTWatch performs a selective code instrumentation to support on-demand privacy analysis/notifications and to facilitate the analysis of encrypted IoT strings.

Privacy User Interface. The instrumentor adds additional code to implement a UI and create a privacy profile of the user. Figure 6.4(a) shows the original user interface of an IoT app presented to the user during install-time, and Figure 6.4(b)



(a) Original IoT app (b) Instrumented IoT app

Figure 6.4: (a) Install-time interface of an IoT app and (b) Instrumented IoT app interface: IoTWatcH interface enables users (1) to select privacy labels and (2) to identify unauthorized recipients when a sensitive information is leaked.

illustrates the selective code instrumentation options of the IoTWatcH-enabled app. IoTWatcH's instrumentor does not impact the UI experience of the IoT app at runtime, but offers new privacy features at install-time not available in the original app. The instrumented app offers the user the possibility to create a privacy profile and receive notifications regarding specific privacy labels that are of interest to the user (❶). Also, it allows for selecting which privacy concerns (e.g., option to notify leaks from messaging or Internet communications) must be analyzed and informed by IoTWatcH (❷). Such a design supports the expectations of the IoT app users with (1) configurable privacy preferences, (2) on-demand privacy controls, and (3) timely privacy notifications (Table 6.1) as were summarized in Section 6.2. Finally, since we target open-source IoT platforms, the use of selective instrumentation does

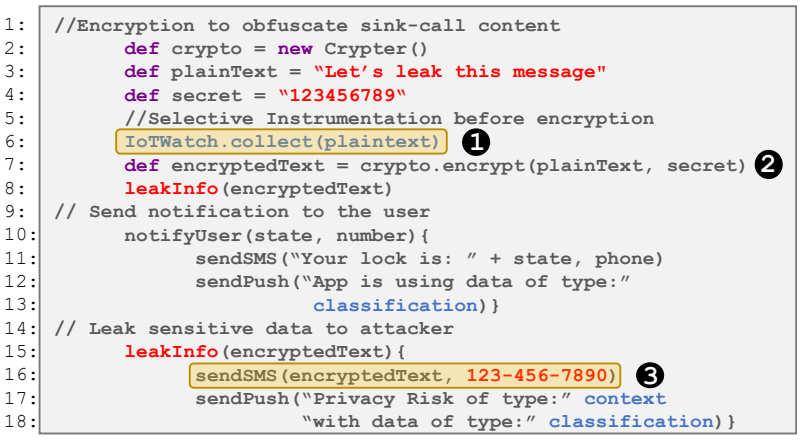


Figure 6.5: Sample IoT app that encrypts the sensitive data to be leaked in an attempt to bypass the NLP analysis of IoTWatcH. The selective instrumentation capabilities of IoTWatcH permits the analysis of the data before it is encrypted.

not change or impact the original user interface of the app that may be considered as intellectual property. For closed-source platforms, we envision developers using the features offered in IoTWatcH to evaluate and improve the protection of sensitive information and the privacy of users.

We detail Algorithm 4, which describes the source code analysis and app instrumentation process in IoTWatcH. Our privacy tool requires open access to the IoT apps' source code (Line 1). This requirement does not constitute a limitation since several IoT platforms (e.g., SmartThings, OpenHAB) make the source code of the apps available online through official and third-party repositories [Smag, Opea, Smac]. The first step towards analyzing the apps is to generate the IR (Line 2) and extract the ICFG (Line 3) of the app. From there, the app's data-flow analysis starts by flagging relevant app and data-flow information (e.g., user-defined data, data-flow recipients). Then, the automatic instrumentation process starts by inserting extra code that defines the different IoTWatcH-related *global* variables inside the `installed()` and `updated()` methods (Lines 7, 8, 13, and 14). In that way, IoTWatcH always keeps the current value of the flagged data through all the different node executions.

Also, the instrumentor implements a UI to define the privacy preferences of the user (Line 9). Finally, one can notice from Line 22 and Line 26 that two different methods are inserted to monitor messaging and Internet communications, respectively.

Algorithm 4: *App source code analysis and code instrumentation in* IoTWatcH

```
1: appSC ← app source code
   {Source Code Analysis}
2: IR ← generateIR(appSC)
3: ICFG ← generateICFG(appSC)
   {Instrumentation}
4: for All ICFG nodes invoking sinks do
5:    if node has not been previously visited then
6:       if node invokes install method then
7:          global ← user-defined recipients
8:          global ← user-defined URLs
9:          Implements Privacy Settings UI
10:         Flag ICFG node as visited
11:      end if
12:      if node invokes install method then
13:         global ← user-defined recipients
14:         global ← user-defined URLs
15:         Flag ICFG node as visited
16:      end if
17:      if node invokes update method then
18:         Update global
19:         Flag ICFG node as visited
20:      end if
21:      if node invokes messaging sink then
22:         Insert WatchMsg method
23:         Flag ICFG node as visited
24:      end if
25:      if node invokes Internet sink then
26:         Insert WatchInt method
27:         Flag ICFG node as visited
28:      end if
29:   end if
30: end for
```

IoTWatcH vs. Data Encryption. The use of encryption to hide the content of a sink-call may limit the effectiveness of NLP techniques. For instance, NLP models created from plain-text data would fail to classify encrypted strings. However, selective instrumentation may facilitate the analysis of IoT apps that codify or encrypt

the data that is sent out to hide their intent. In general, IoT programming platform only permits limited (i.e., white-listed) number of libraries to implement the app's execution [Smaa]. In popular IoT platforms like Samsung SmartThings, this group of white-listed APIs does not include a single class that implements encryption. Also, as a general rule, IoT platforms reject obfuscated apps when developers submit them for approval during the vetting process. However, there still several reasons to allow encryption-handling in IoTWatch. First, malicious apps may skip the vetting process of IoT programming platforms and while still be in the app market. Second, encryption handling allows for a privacy tool that is not only exclusive to specific IoT platforms, but that can be generalized as broader solution. As explained before, the use of a limited (i.e., white-listed) number of libraries to code the IoT apps facilitates the detection of encryption functions by IoTWatch. Figure 6.5 shows a sample IoT app that implements encryption to hide its intentions of leaking a sensitive string via messaging. The method `crypto.encrypt` is known to be approved to implement encryption. IoTWatch's selective instrumentation extracts the call graph of the app and detects the presence of a function implementing encryption (❷) on a tained variable `plainText`. The result of the encryption is stored in a new taint variable `encryptedText`, which is later leaked via messaging (❸). To solve this challenge, the instrumentor tracks the flow's path of the encrypted variable and inserts extra code to collect its content before it is encrypted (❶).

During our analysis, out of 540 current market and malicious IoT apps, we did not find a single case of an app encrypting the content while using messaging. However, we found several cases of encryption in Internet communications. IoTWatch's instrumentor bypasses encryption on the content of the Internet calls by analyzing the entire data-flow path within the apps (i.e., from the data source to the sink functions), and by collecting the information related to the sink-call content before it is

encrypted or codified. In the specific case of encrypted Internet communications, the instrumentor flags the call graph nodes containing the `https` call methods and adds additional code to collect its content before the encryption call is executed.

6.5.2 IoTWatcH Analyzer

IoTWatcH uses NLP techniques to analyze and classify an app's sink-call contents to four privacy labels. Additionally, it flags messaging and Internet communications that disclose sensitive information to unauthorized recipients. Finally, it uncovers privacy concerns from apps that do not protect the sensitive information from passive observers. Figure 6.3 illustrates the logical steps of the IoTWatcH's analyzer at runtime, that includes: (1) classification of sink-call content through NLP techniques, (2) privacy analysis of recipients of sensitive information, and (3) building a user privacy notification interface to inform the user about IoTWatcH's findings.

Algorithm 5 details the steps followed in IoTWatcH's analyzer. First, in Line 1, the analyzer extracts all the information received from the IoT app through IoTWatcH's REST API. Also, variables are initialized in Lines from 2 to 9. After IoTWatcH completes the initialization step, the tool applies multi-class classification on the app's data-flow contents in Line 12. Then, it also performs data-flow recipient analysis by correlating recipients from sink-calls with user-defined data to detect for privacy behaviors (Lines 14 and 17). Finally, IoTWatcH's results are sent back (Line 21) to the user as a Push Notification.

Classification of Sink-Call Content

The content of messaging and Internet communications in IoT apps may include sensitive information from taint variables or might be just string messages with privacy implications to the user. As noted earlier, the sink-call content classification

Algorithm 5: *Sink-call recipient analysis and privacy classification of IoT strings at runtime*

```
1: inputs ← IoTWatch's API content
2: userinfo ← input.userinfo
3: labels ← input.selectedLabels
4: content ← input.content
5: recipient ← input.recipients
6: url ← input.url
7: flowResult ← NULL
8: labelResults ← NULL
9: semanticsLabels ← NULL
10: for input ∈ inputs do
11:     if (content ≠ NULL) then
12:         labelResult ← NLP(content)
13:     end if
14:     if url is insecure then
15:         flowResult ← "privacy concern"
16:     end if
17:     if recipient ∉ user − info then
18:         flowResult ← "privacy concern"
19:     end if
20: end for
21: sendResults(flowResult, labelsList)
```

of IoTWatch takes the content of messaging and Internet communications as input and assigns it to privacy labels returned as the output. For instance, if the sink-call contains the string message "the door is unlocked", the classifier performs semantic analysis on the string and classifies it as "Device-info". The conversion of IoT strings into privacy labels helps the user to understand how IoT apps use sensitive information so they can make informed privacy decisions. Below, we present how to construct a training set for classification from a corpus of IoT apps.

Constructing Privacy Labels. IoTWatch classifies IoT app sink-call contents into a set of privacy labels. The use of privacy labels provide three main advantages. First, they allow users to understand what type of data the apps leak. For instance, a string that contains "The mode has been changed to Home" can be presented to users as leaking location. Second, the privacy labels permit users to have control over

their privacy preferences. For instance, a user may desire to allow an app to transmit energy usage of a thermostat to a remote server, yet she restricts other types of privacy-related information. Third, the privacy labels enable an on-demand notification system that guarantees an improved user experience with less disruptive and more intuitive notifications, and minimal runtime delay added to the app's execution time.

We define four privacy labels through the analysis of the semantics of strings extracted from messaging and Internet communications in IoT apps (although adding more privacy labels later is a straightforward task). The privacy labels are based on user feedback that we acquired via a privacy survey (Section 6.2). Also, to adequately protect the user's privacy, some strings may require the use of more than one privacy label to guarantee completeness on the classification. For instance, the message string "The door will remain open for another 5 minutes" supports multi-labeling of types Device-info and Date-time. Similarly, "Garage door is not opening since the car was not present at Home, less than 15 sec ago", would be labeled as Device-info (the door is not opening so it is still closed), Location (the car was not present at Home), and Date-time (15 sec ago). Some examples of IoT sink-call contents assigned to the various privacy labels are presented in Table 6.2. In general, the use of multiple privacy labels to a single text considers more complex semantics structures of string and reveals more privacy-sensitive information contained in the leaked message. As we explain later in this section, we successfully use NLP to achieve this goal. We present the four privacy categories used to classify IoT strings:

- *Date-time:* defines an app text that contain time or date information. For instance, a messaging call that sends the string "door is unlocked at 5:00 pm" contains time information that specifies the time that the door would be unlocked.

App Sink-Call Content	Assigned Privacy Labels
Thermostat is turned on.	device-info
The door will remain open for another 5 minutes.	device-info, date-time
Door is not opening since car was not present at Home, less than 15 sec ago.	device-info, location, date-time
Sleep time set for you as requested.	user-behavior

Table 6.2: Examples of leaked strings extracted from IoT apps and their assigned privacy labels. Observe that IoTWatch is capable of assigning multiple privacy labels to specific strings with more complex semantics.

We found that many IoT apps contain date-time information for reporting the state of a device at any given time.

- *Device-info:* defines the text that contain the states of devices and also device information (i.e., device type, model, manufacturer). For instance, an Internet call that contains "energy usage" transmits out the power state of a thermostat. We assign all strings that contain information from devices with a device-info label to inform the users about potential privacy violations.
- *Location:* defines the text that reveal physical and geo-location location of users and devices. For instance, the string "kids have arrived home" contains physical location information, and "the patio door is unlocked" contains the geo-location of a house.
- *User-behavior:* defines the text that provides information about user preferences. We found that many IoT apps leak strings in messaging calls about app configuration and user activities. For instance, an app includes a string, "the user mode changed to vacation from home".

NLP Model Construction. Our search for an adequate corpus that characterizes IoT app's data-flows faced particular challenges. First, we could not find any existing IoT corpus. Second, most of the datasets available online contain raw unlabeled

data that would require a considerable amount of pre-processing time and resources. Third, the privacy labels considered by IoTWatch could not be inferred from n-gram shingles extracted from a single corpus only. Thus, we combined different knowledge-based datasets to create a larger corpus. We combined the natural language datasets from Google Books N-grams [Goob] and Wikidata [Wik], which contain strings related to geographic (location), economic (devices, user's goods), climate (location), and encyclopedic (general knowledge) datasets. We structured, cleaned, and manually labeled the crawled data. First, we divided the corpora into single shingles (i.e., n-grams of n=1). Then, for cleaning purposes, we filtered out punctuation and stop words. We tested the first NLP model with 61 IoT strings extracted from 45 market IoT apps [Smag, Smac]. An average value of 72% accuracy proved that the first considered model could not accurately represent information extracted from IoT environments. Based on these results, we decided to train the NLP model using specific IoT corpora only.

We implemented an NLP model that uses a specific IoT corpus for training purposes. Then, we used the model to classify unknown sink-call contents to privacy labels. To train the model, we used a supervised learning approach that required labeled data as input. We found that the supervised approach yields better results than other approaches such as keyword-search. This is because often, the IoT string does not include enough information to assign the labels through simple keyword-search-based analysis. For instance, keyword-search would fail to identify the user-behavior in a messaging communication that leaks a string "the kids left home". Moreover, we used *doc2vec* [Ayy18] to represent every n-gram IoT corpus text into a multi-dimensional vector. We use doc2vec over other known approaches like bag-of-words (BoW) [LM14] as it considers the entire structure of the text to perform syntax analysis (i.e., multi-word expression analysis) [MCCD13]. We then performed topic

classification of the IoT strings using an automated machine learning approach [Auta]. Note that the analysis of NLP results from different classification algorithms is out of the scope of this manuscript. In fact, our automatic machine learning approach selects the best classification algorithm depending on the structure of text; thus, the best accuracy value is always guaranteed in IoTWatch. In Section 6.6, we provide details of the NLP implementation, including the IoT corpus used to train the model.

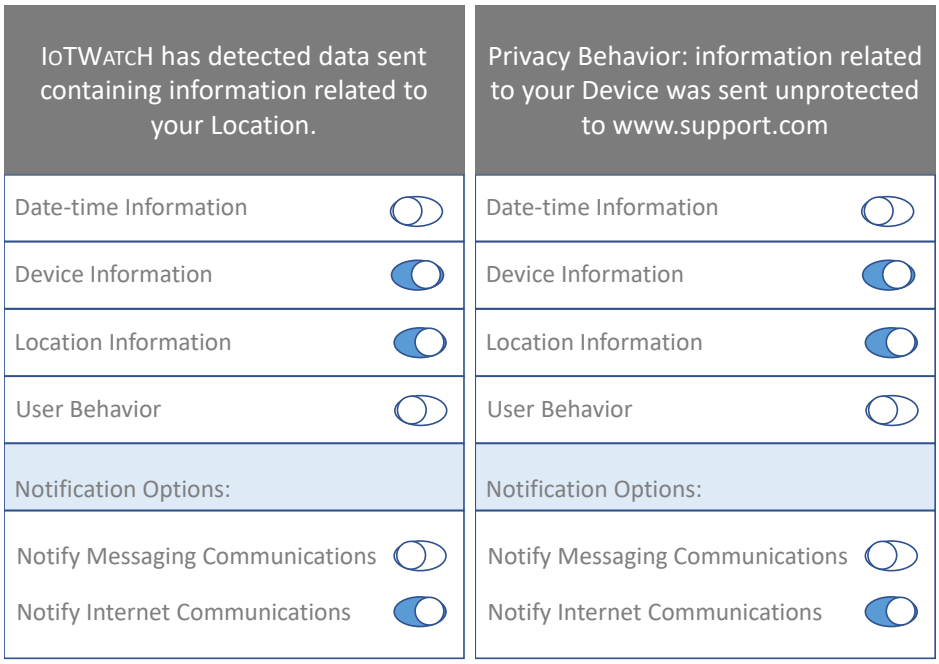
IoT App Corpus. Our study of sink-call contents (i.e., IoT strings) from 540 real IoT apps showed that they pose a few unique characteristics compared to data-flows from other domains. First, the size of the texts extracted is usually three to four shingles on average, yet it contains highly private data (e.g., "the door is unlocked"). Second, their linguistic structure is minimal regarding semantics (e.g., "mode changed to away") compared to other short documents extracted from popular general-knowledge corpora [N. ,Goob,Wik]. Third, their meanings usually are closely attached to the app's context (e.g., "if the user is not-present, turn off the light"). Based on these facts, we designed an NLP model that can be effective for classifying semantic-deficient, but information-rich texts. To understand whether leaked strings includes sensitive information and assign them a privacy label, we first implemented a classification model using publicly available data corpora [N. ,Goob,Wik]; however, due to the specific characteristics of IoT texts, we obtained very low classification accuracy. To improve these initial results, we constructed an *IoT-specific* corpus for classification purposes that successfully considers and solves the challenges above. To do so, we first collected the content of messaging and Internet calls from current IoT market apps. We pre-processed the resulting dataset by filtering out punctuation and stop words. Further, we manually labeled the IoT strings to the four privacy labels. Here, we applied multi-labeling to contents that contained information related to more than one privacy label. We detail constructing the IoT corpus in Section 6.6.

Sink-Call Recipient Analysis

IoTWatcH performs sink-call recipient analysis via messaging by matching recipients of the data with information defined by the user at install-time. For instance, in Figure 6.1, the user defines an authorized recipient in the variable *phone*. IoTWatcH extracts this information and correlates it with the app data extracted at runtime. In this specific example, two outcomes are possible. In Line 15, a messaging function is executed using *phone* as the recipient. Since IoTWatcH recognizes that the user previously authorized this recipient, the flow does not represent a leakage. However, a second messaging function is executed in Line 19. In this case, IoTWatcH sees discrepancies between the recipient used (i.e., 123-456-7890) and the one that was defined by the user, and flags it as a leakage. Also, the analyzer checks the capabilities of the recipients of Internet communications to support encryption and adequately protects the sensitive information from, for instance, eavesdroppers. Back to our example, in Figure 6.1, an Internet communication is executed in Line 16. IoTWatcH extracts the recipient used (i.e., URL) and observes that the remote server is being accessed via Hyper Text Transfer Protocol (HTTP), so the information is sent out of the app in plain text. This constitutes a privacy behavior that could expose the content of the Internet message to passive observers (i.e., eavesdroppers). In this case, IoTWatcH flags this flow as a potential privacy concern for the user.

6.5.3 Response to App Data Leaks

Our privacy survey (Section 6.2) shows that users of IoT apps desire on-demand privacy analysis and notifications. Based on this feedback, IoTWatcH implements two different privacy notification options (Figure 6.6). First, it allows a user to select specific privacy labels (one, multiple, or all) to create a privacy profile and receive notifications. For instance, if the user is only concerned about the use of data related



(a) User configuration (b) Notification to users

Figure 6.6: IoTWatcH's findings are informed to the users through push notifications. The findings include (1) the privacy labels assigned to the sink-call content, and (2) the potential privacy concerns associated with the IoT communications.

to location information, she may select the location label so that IoTWatcH informs if a sink-call content contains information related to her and the devices' location. In this case, the user would not receive notifications regarding the use of other types of sensitive information within the IoT app. Second, IoTWatcH allows the user to decide if she desires to be notified regarding messaging, Internet calls, or both, whenever they potentially leak data to unauthorized parties. The approach of implementing a selective user-specific notification system provides flexible privacy options to the user, lowers the latency overhead of IoTWatcH by reducing the number of processed and classified IoT strings, and enhances the user experience by reducing the number of notifications at runtime. Finally, even though its flexible and very configurable nature, IoTWatcH enables all the privacy labels and notifications options by default. On

the one hand, technically-enthusiastic users that fully understand the privacy risks of IoT apps may create their own privacy profiles by disabling install-time options in IoTWatcH, for a better user experience. On the other hand, users that are not aware of the privacy implications of sensitive information being leaked to third parties via IoT apps, or that do not completely understand the privacy labels in IoTWatcH, may rely on the default options.

6.5.4 IoTWatcH API

The analyzer collects app data to uncover data leaks and privacy behaviors in IoT apps. We implemented a REST API to enable effective data exchange and communication between the instrumented IoT app and IoTWatcH's analyzer running in the cloud (Figure 6.3). From the app to the server, the API constructs a JSON object with the user-defined recipients and the sink-call information. From the server to the app, another JSON is transmitted including the analysis results and the notifications to the user. Listing 6.1 illustrates an example of a JSON object used to send data from an IoT app to IoTWatcH analyzer. Once the app data is received, the analyzer extracts the data required to enable recipient analysis and NLP-based classification of the IoT strings. The API also handles IoTWatcH's privacy notifications to the user. Once the privacy analysis is completed, IoTWatcH sends back to the user another JSON object containing its findings (Listing 6.2). We implemented the API using the *asynchttpv1* class of Samsung SmartThings [Smah] which allows asynchronous and encrypted data exchange between the instrumentor and the analyzer.

Listing 6.1: An example of a JSON object sent from an IoT app to IoTWatcH for further analysis.

```
/* An example of a JSON object sent to Daint analytics tool */

 data "{
  'exfiltration':{
    'texttype':'PLAIN_TEXT',
    'calltype':'Messaging',
    'phone':'111-111-1111',
    'content':'The door was opened for 10 min'
    'userrecipients':'123-456-7890',
  }
}" "https://iotwatchanalyticstool.com/classifytext/"
```

Listing 6.2: An example IoTWatcH response as a JSON object received by an IoT app.

```
/* An example of a JSON object as response from our analytics tool */

 data "{
  'exfiltration':{
    'texttype':'PLAIN_TEXT',
    'classification':['device-info', 'date-time']',
    'risklevel': 'privacy concern'
  }
}"
```

6.6 IoTWatcH's Implementation Details

We implemented IoTWatcH for IoT applications developed for Samsung SmartThings, which is the IoT platform that has the highest share of devices and applications in the current IoT market [SSIPD, Smab]. Samsung SmartThings apps are developed in Groovy, a dynamic programming language that supports static compilation. Static compilation permits for all methods and classes in the apps to be annotated at compile time, which makes this information fully available to the instrumentation portion of IoTWatcH.

Code Instrumentation. IoTWatcH traverses on the Abstract Syntax Tree (AST) of the IoT app's IR through the `ASTTransformation` class and builds an app's Intra-procedural Control Flow Graph (ICFG) [CBS+18]. IoTWatcH involves around 1700 lines of code written in Groovy to analyze the app source code, construct the IR, generate the ICFG, and perform the code instrumentation. We implemented IoTWatcH's instrumentor as a web application using Groovy programming language. We made the instrumentor available online at: `https://IoTWatcH.appspot.com/`.

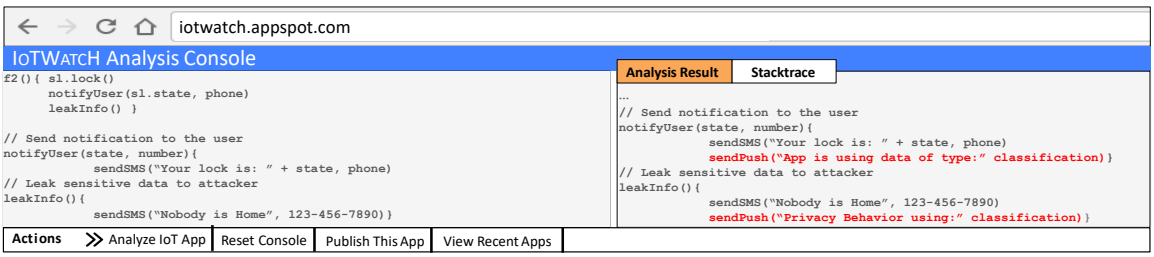


Figure 6.7: The left console is the analysis area where the user inputs the original IoT app. The right console returns the output of the instrumentation process. We made IoTWatcH's instrumentor freely available to the community at https://IoTWatcH.appspot.com/.

Figure 6.7 depicts details of the online version of IoTWatcH's instrumentor. At the left console, the user inputs the IoT app source code that needs to be modified to enable IoTWatcH, and at the right console, the tool automatically returns the IoTWatcH-instrumented app. Below, we detail the implementation steps of IoTWatcH.

Collection of IoT Corpus. To avoid overlapping between the data used for training and evaluation of IoTWatcH, out of 540 apps, we selected 380 current market apps crawled from SmartThings repositories [Smag, Smac] to build the IoT corpus. The app population included apps from 6 different categories: *Convenience*, *Smart Home Automation*, *Entertainment*, *Personal Care*, *Security & Safety*, and *Smart Transportation*. From the selected apps for training, we extracted a total of 2014 different IoT strings. We then labeled these strings according to the four privacy labels. Specifically, 46.8% of the strings contained information related to the IoT devices, while 20.8% contained relevant information related to Date-time. The remaining 19.2% and 13.2% of the IoT strings shared information related to location and user-behavior, respectively. Additionally, we allowed up to 75% of inter-labeling assignment to the strings, meaning, up to three different privacy labels can be assigned to a single string. Figure 6.8 depicts statistical details of the privacy label distribution used to build the IoT corpus. In total, we applied multi-labeling to 72% of the privacy strings in the IoT corpus. Finally, we used 75% of the total corpus to train the classifier. Then,

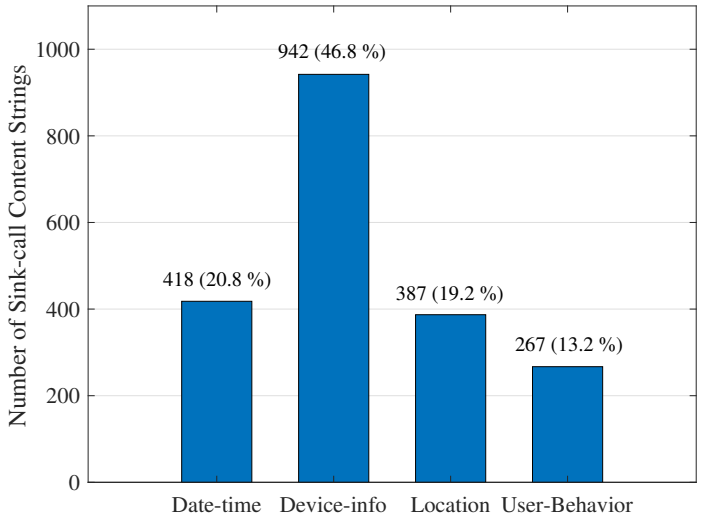
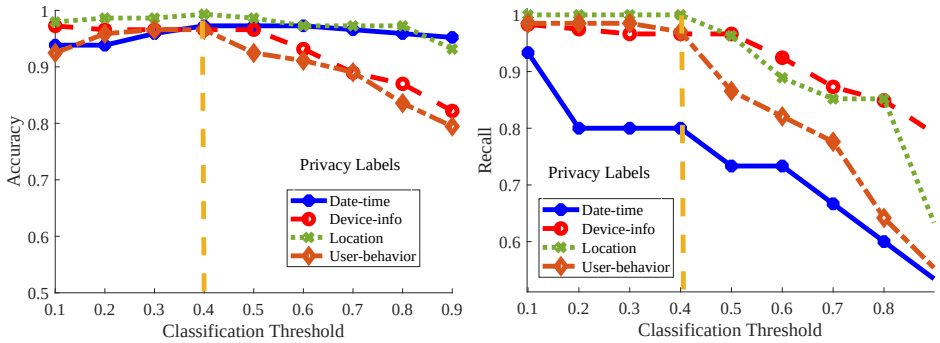


Figure 6.8: Distribution of privacy labels among all the IoT strings included in the corpus.

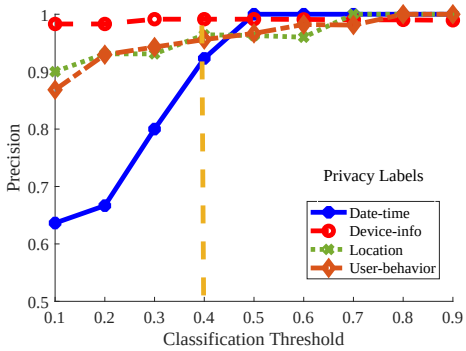
we verified the obtained model with the remaining 25% of the data. Initial testing results on the NLP model showed an average precision of 94.3% and recall of 89.6%.

Classification of IoT Privacy Strings. We use Automatic Machine Learning (Auto-ML) tools offered by Google App Engine [Auta] to perform privacy classification of IoT strings. Among its benefits, modern auto-ML approaches perform neural architecture search, enable hyper-parameter optimization, and utilize advanced model architectures to classify contents to privacy labels with high accuracy. More particularly, we implemented a custom multi-class multi-label model using the Natural Language API offered by Google (Google-NL) [Autb]. Google-NL offers a suite of ML algorithms that automatically optimize the algorithm parameters based on the specific algorithm utilized and the characteristics of the dataset to guarantee the highest accuracy. Our initial analysis of 2014 IoT strings showed remarkable semantic similarities among them; thus, the use of labeled data reduces the training time considerably. Finally, we noticed that only two (0.5%) IoT strings in the corpus were in an idiom different from English (Spanish in both cases). Thus, we implemented our NLP solution to classify strings defined in English.

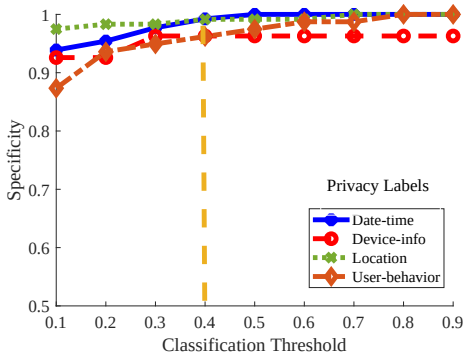


(a) Accuracy

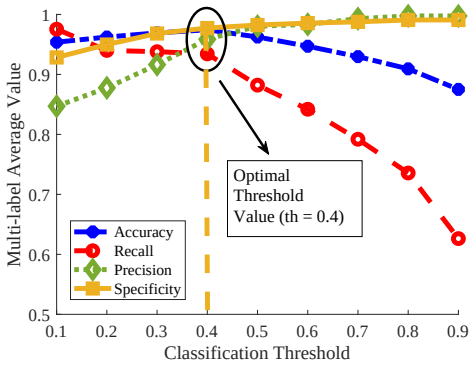
(b) Recall



(c) Precision



(d) Specificity



(e) Average Value

Figure 6.9: Evaluation results in the classification of IoT strings extracted from messaging and Internet communications to user-friendly privacy labels: (a) accuracy, (b) recall, (c) precision, (d) specificity, and (e) the average value of all considered performance metrics.

6.7 Performance Evaluation

We evaluated IoTWatch based on the three research questions (RQ) below:

RQ1 What is the effectiveness of IoTWatcH in correctly classifying the IoT strings into privacy labels? (Section 6.7.3).

RQ2 What is the effectiveness of IoTWatcH in detecting sensitive data leaks and potential privacy behaviors in IoT apps? (Section 6.7.4).

RQ3 What is the runtime performance overhead of IoTWatcH in terms of latency and storage? (Section 6.7.5).

We used a total of 540 current IoT apps to implement and evaluate IoTWatcH. Out of the total, 380 SmartThings apps were used to build the IoT corpus and train the NLP model, and the other 160 apps were used to evaluate its performance. For comprehensiveness, we included in the evaluation phase 120 SmartThings market apps [Smag, Smac], and 40 malicious apps crawled from the IoTBench repository [IoT17]. IoTBench is an IoT-specific test corpus used to evaluate systems designed for IoT app security and privacy. It includes flawed apps that perform various malicious activities, including sensitive data leaks via both messaging and Internet functions. After instrumentation of the evaluation apps was completed, we executed the instrumented apps in the SmartThings IDE [Smad], which is a propriety simulation environment to execute SmartThings apps. IoTWatcH's instrumentor adds 25% more lines of code (LoC) to the apps on average, which translates into adding 65 LoC to an IoT app that has an average size of 265 LoC. While being executed in SmartThings IDE, the instrumented apps send their privacy data to IoTWatcH's analyzer which runs on a Python web server hosted on Google App Engine [Gooa].

We evaluated IoTWatcH's accuracy in classifying strings extracted from IoT messaging and Internet communications into users' privacy preferences (i.e., privacy labels) (Section 6.7.3). Also, we check its effectiveness in detecting sensitive data leaks and flagging potential privacy behaviors in IoT apps (Section 6.7.4). As noted earlier in Section 6.4.2, a leak is defined as a piece of information that leaves an IoT app

without user consent. In IoT apps, the consent of a user is either asked through an app description block or through acknowledging the flow's recipients at install-time. However, we found that, out of 160 apps, only 121 (75.6%) define some type of app's description. Additionally, in most cases users are not informed nor given the chance to assert the app's intent or the recipients of sensitive information. For instance, a messaging function using a hard-coded recipient not entered by a user, or an Internet communications that sends sensitive data to a server that is not informed to the user via an app description block is clearly a leak. Finally, we define privacy behaviors that put the sensitive information at risk. For instance, an Internet communication that sends data to a remote server without using encryption.

6.7.1 Evaluation Metrics

The performance metrics used during IoTWatcH's evaluation:

1. *True Positive (TP)* represents the number of times a privacy label is correctly applied to an IoT string for certain threshold th.
2. *True negative (TN)* represents the number of times a privacy label is correctly discriminated for certain threshold th.
3. *False Positive (FP)* is the number of times a privacy label is incorrectly assigned to certain IoT string for certain threshold th.
4. *False Negative (FN)* is the number of times a privacy label is incorrectly discriminated for certain threshold th.
5. *Accuracy* is the overall ability of IoTWatcH to correctly apply privacy labels to the IoT string for every different th.
6. *Recall* is the ability of the classifier to correctly assign the privacy labels to a specific IoT string after considering both the correctly classified and the incorrectly ignored privacy labels for every value of th.

Label Category	Messages without Leaks or Privacy Concerns	Internet without Leaks or Privacy Concerns	Messages with Leaks or Privacy Concerns	Internet with Leaks or Privacy Concerns	Total	% Total
Device-info	47	12	43	17	119	52.3
Date-time	7	1	7	0	15	6.6
User-behavior	21	2	34	10	67	29.3
Location	8	0	14	5	27	11.8
Total	83	15	98	32	228	100

Table 6.3: Distribution of privacy labels used during IoTWatch's evaluation.

7. *Precision* is the ability of our classifier to correctly apply the privacy labels to a specific IoT string after considering both the correct and the incorrectly applied privacy labels for every value of th.
8. *Specificity* is the ability of our tool to discriminate the privacy labels for every different th.

The performance metrics are defined by the following equations:

$$Accuracy = \frac{(T_P + T_N)}{(T_P + T_N + F_P + F_N)}, \tag{6.1}$$

$$Recall = \frac{T_P}{(T_P + F_N)}, \tag{6.2}$$

$$Precision = \frac{T_P}{(T_P + F_P)}, \tag{6.3}$$

$$Specificity = \frac{T_N}{(T_N + F_P)}. \tag{6.4}$$

6.7.2 Assigning Privacy Labels

Table 6.3 summarizes the distribution of the privacy labels during evaluation. One can notice that, out of 228 different labels assigned, 52.3% corresponds to device-info, followed by a 29.3% of user-behavior information. We also show in Table 6.3 the number of labels assigned to the different call types (i.e., messaging and Internet).

For instance, 83 labels were assigned to strings extracted from messaging that do not leak information, while 98 labels were assigned to strings extracted from messaging that leaked information. On the other hand, Internet communications without privacy concerns received 15 privacy labels while 32 were assigned to Internet communications that leak data. With this distribution, one can notice that the majority of labels were assigned to leaked data. These results reflect on the fact that privacy leakage constitute a serious problem in IoT. Indeed, the more privacy-sensitive labels assigned to potential leaks, the higher the risk of users' and system's sensitive information being leaked by IoT apps.

6.7.3 Performance of IoT String Classification

IoTWatch's analyzer classifies IoT app strings into four privacy labels. The classification results include the assigned label and the confidence scores through a threshold th. If the classification score is over the predefined threshold, the privacy label is assigned to the string. For instance, the string "the front door at home is unlocked" resulted in classification scores of device-info=0.94, user-behavior=0.03, location=0.86, and date-time=0.3. By design, the classification scores are independent of each other, that is, if th value is set to 0.7, IoTWatch classifies the string into privacy information of type device-info and location. In total, IoTWatch classified 146 IoT strings extracted from 95 different IoT apps. Out of these, 112 strings were extracted from messaging communications; 54 messages from 44 market IoT apps and 58 messages from 30 malicious apps. The remaining 34 strings were extracted from Internet communications; 12 Internet calls extracted from 10 market IoT apps, and 22 extracted from 11 malicious IoT apps.

Classification of Encrypted IoT Strings. As detailed in Section 6.5.1, the selective instrumentation of IoTWatch enables NLP analysis on encrypted data. We

App Type	Total Apps Analyzed	Messaging Ccommunications Analyzed	No. of Data Leaks	No. of Privacy Concerns	IoTWatcH Effectiveness
Market	120	54	0	–	–
Malicious	40	58	29	–	100%
Total	160	112	29	–	100%

Table 6.4: Effectiveness of IoTWatcH in detecting sensitive data leaks using messaging.

study the effectiveness of IoTWatcH in classifying IoT strings that have been previously encrypted. Out of 160 apps analyzed, we found seven benign apps executing nine encrypted Internet communications. Similarly, we found 11 malicious apps executing 23 Internet communications with encrypted content. Despite the use of encryption techniques, IoTWatcH effectively collected and classified the information sent as encrypted strings in 100% of the cases. Finally, we did not find any IoT application using encrypted messaging.

Accuracy by Threshold Values. We study how IoTWatcH's classifier performs for different threshold values th. The goal of this analysis is to find the value th that leads to the highest performance overall. Figure 6.9 illustrates the accuracy, recall, precision, and specificity of the NLP model. Overall, IoTWatcH yields the best accuracy with the threshold values between 0.1 to 0.5. Figure 6.9(e) summarizes the average metrics for different values of th. We observe that $th = 0.4$ yields the best classification results for all metrics. Finally, IoTWatcH classifies IoT strings to correct privacy labels with 94.25%, 85.17%, 95.01%, and 97.34% average accuracy, recall, precision, and specificity, respectively.

Accuracy by Privacy Label. We further study the sensitivity of IoTWatcH's classifier to each privacy label. The goal is to determine the effectiveness of IoTWatcH in classifying strings to the different privacy labels. IoTWatcH achieves the highest accuracy for privacy labels of type location and date-time. This is because date, time,

App Type	Total Apps Analyzed	Internet Communications Analyzed	No. of Data Leaks	No. of Privacy Concerns	IoTWatcH Effectiveness
Market	120	12	11	3†	100%
Malicious	40	22	22	3‡	100%
Total	160	34	33	6	100%

† Includes one privacy concern that does not constitute a data leak and two privacy concerns that were also flagged as data leaks.

‡ Includes three privacy concerns that were also flagged as data leaks.

Table 6.5: Effectiveness of IoTWatcH in detecting sensitive data leaks via Internet communications.

and location can be easily inferred from semantically-simple strings. Also, it is very common to find information related to these privacy labels embedded in the same string (e.g., "he arrived home 5 minutes ago"). In contrast, IoTWatcH obtained the lower accuracy results for privacy labels of type user-behavior. This is because user-behavior information is harder to infer from simple strings. In spite of these results, IoTWatcH achieved the lowest accuracy of 90.79% for user-behavior labels, which is comparable with the best classification results of other similar tools in the market [PCD+18]. Table 6.6 details average metric values in classifying IoT strings to privacy labels for every different th. Also, we present the average metric values after combining results from all considered privacy labels and thresholds. One can verify that IoTWatcH obtains the best performance for threshold values of 0.4, which supports the results showed in Section 6.7, Figure 6.9(e). For th values higher than 0.5, the accuracy decreases for the privacy labels of device-info and user-behavior. This is mainly because the evaluation of semantically-limited strings related to these two privacy labels requires more sophisticated analysis. For instance, user-behavior achieved incorrect or lower classification scores as the string "IoT switched to sleep mode" cannot be easily related to user activities. We obtained similar results for recall metrics; however, recall values of date-time are lower compared to other labels. We found that date-time information could be easily missed from short strings, which

Evaluation Metric	Classification Thresholds									
	0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**Average**
Accuracy	0.9537	0.9623	0.9692	0.9743	0.9623	0.9469	0.9296	0.9092	0.8750	0.9425
Recall	0.9754	0.9400	0.9379	09341	0.8821	0.8419	0.7919	0.7359	0.6263	0.8517
Precision	0.8470	0.8776	0.9163	0.9587	0.9803	0.9833	0.9929	0.9975	0.9835	0.9501
Specificity	0.9283	0.9498	0.9682	0.9772	0.9823	0.9855	0.9876	0.9907	0.9907	0.9734

Table 6.6: Evaluation metric results in classifying IoT strings for all classification thresholds. The rightmost column presents the average metrics.

Privacy Label	No. of Times Evaluated	TP	TN	FP	FN	Accuracy	Recall	Precision	Specificity
Device-info	1285	958	232	11	84	0.9260	0.9214	0.9890	0.9547
Date-time	1314	99	1161	18	36	0.9589	0.7333	0.8918	0.9847
User-behavior	1313	508	684	26	95	0.9079	0.8425	0.9585	0.9633
Location	1314	221	1061	10	22	0.9756	0.9095	0.9610	0.9907

Table 6.7: Evaluation results of IoTWatch in classifying IoT strings to all the different privacy labels.

makes the label specially vulnerable to false negative events. The precision and specificity improves with th for all the labels, which denotes a remarkable confidence of the model for those results with the highest classification scores. Finally, Table 6.7 illustrates the performance of IoTWatch in classifying IoT strings for every different privacy label.

Findings on Privacy Analysis of IoT Strings. IoTWatch classifies IoT strings to privacy labels with an average accuracy of 94.25%. Out of 160 apps, IoTWatch identified 50 (31.25%) apps that transmit data related to device information via messaging, and 20 (12.5%) apps that do the same via Internet communications. Also, 11 (6.9%) apps handled data related to date and time in messages and only one transmit similar information using the Internet. IoTWatch also identified 38 (23.75%) apps transmitting information related to the user behavior in their messages and nine (5.6%) including similar type of information in Internet communications. Further, 20 (12.5%) apps sent information related to location via messaging, while six (3.7%) apps did the same via Internet. Finally, we evaluated how the privacy analysis of IoT

strings benefited from the use of NLP. IoTWatcH assigned multiple privacy labels to classify more semantically-complex IoT strings, which guaranteed completeness in the privacy analysis. Out of 146 strings analyzed, IoTWatcH applied multi-labeling to 68 (46.5%). Specifically, 54 IoT strings (36.9%) received two privacy labels and 14 (9.6%) strings received three privacy labels.

6.7.4 Analysis of Data Leaks in IoT Apps

We evaluated the correctness of IoTWatcH in identifying data leaks. Table 6.8 presents examples of data leaks in IoT apps, and the associated privacy labels assigned to the leak.

Findings on Data Leaks via Messaging. Table 6.4 shows IoTWatcH's findings after analyzing messaging recipients. IoTWatcH extracted 54 recipients in messages from market apps and 58 from malicious apps. We found no data leaks via messaging in market apps, meaning, all recipients were defined (or authorized) by the user at install-time. We believe this is due the strict review process enforced by SmartThings IoT market. Further, we found 29 leaks from 14 different malicious apps [IoT17], meaning, all these recipients were hard-coded by a developer and their intent were not defined in the app's description block. For instance, the User Event app [IoT17] leaks privacy labels of type device-info, location, and user-behavior ("Everyone is away and hub ID is #") to a hard-coded phone number. We manually reviewed the app source codes and verified that all IoTWatcH's findings were correct. Also, we verified that all data leaks via messaging were properly flagged.

Findings on Data Leaks via Internet. Table 6.5 details the effectiveness of IoTWatcH's analyzer in finding data leaks via Internet communications. IoTWatcH analyzed 34 recipients from Internet communications, 12 from market and 22 from malicious apps. Our tool flagged 11 Internet communications from seven market apps

Type*	App Name	Leak Type**	Recipient	Content	Behavior	Privacy Labels
	Squeeze Box Controller [Smac]	I	http://$ip:$port	"Open box"	●	Device-info
	StatHat Quick Start [Smac]	I	http://api.stathat.com/ez	Thermostat1:thermostat	●	Device-info
	ThingSpeak Logger [Smac]	I	http://api.thingspeak.com	DeviceID:333:Key:123	●	Device-info
❶	User Lock Manager [Smac]	M	defined by user	User no longer has access to the door	○	Device-info User-behavior
	Smart Lock [Smac]	M	defined by user	SmartLock disabled. Door will remain unlocked indefinitely	○	Device-info User-behavior
	Fire Alarm [IoT17]	I	http://141.212.110.244/stmalware/maliciousServer.php	–	●	–
	Ransomware [IoT17]	I	http://141.212.110.244/stmalware/maliciousServer.php	–	●	–
❷	Remote Command [IoT17]	I	http://141.212.110.244/stmalware/maliciousServer.php	–	●	–
	Spyware [IoT17]	M	123 − 456 − 7890	Doors locked after everyone departed	●	Device-info User-behavior
	User Event [IoT17]	M	123 − 456 − 7890	Everyone is away and Hub ID is 123	●	Device-info Location User-behavior

* ❶ is for Market IoT apps and ❷ is for Malicious IoT apps (Handcrafted)

** I is for Internet and M is for Messaging. ● is for Privacy Risk and ○ is for Privacy Preference of a user.

Table 6.8: Examples of privacy risks and the use of sensitive information in market and malicious IoT apps. IoTWatcH identified six privacy-violating behaviors and 62 leaks from market and malicious IoT apps.

that leak privacy data without user consent. That is, the user is neither informed about the recipient of the data in the app description block nor enters the URL or domain name herself. For instance, the ThingSpeak Logger app [Smac] transmits a device ID to a remote server through an HTTP call, which is identified through a device-info privacy label. For malicious apps, IoTWatch flagged 22 Internet communications from 14 different malicious apps as leaks. We manually verified that all IoTWatch's findings were correct. Also, we verified that 100% of data leaks via Internet were properly flagged.

Findings on Privacy Behaviors via Internet. IoTWatch further verifies whether an IoT app protects the sensitive information before sending it to an external party. The use of encryption gives the sensitive data extra protection against unauthorized disclosure to passive eavesdroppers. Out of 12 Internet communications analyzed from market apps, IoTWatch flagged three from three different apps which make unsecured `HTTP` requests. Interestingly, one of these calls was authorized by the user at install-time through the app description block. Coincidentally, our tool also flagged three Internet communications from three different malicious apps that constituted potential privacy behaviors. Finally, IoTWatch did not detect any privacy behavior via messaging.

6.7.5 Overhead Analysis

We evaluated the performance overhead of IoTWatch in terms of runtime and storage overhead.

Runtime Overhead. Latency refers to the time elapsed from the moment the IoT app sends relevant data via IoTWatch's API (see Section 6.5.4) to the moment that the user receives IoTWatch's notifications. Latency overhead is calculated as the average difference in the execution time of the original and instrumented apps. On

the one hand, IoTWatcH required 75 ms to perform the NLP-based classification of the sink-call contents on average. On the other hand, we implemented IoTWatcH's API using asynchronous `HTTPS` communications via the *asynchttp_v*1 class defined in SmartThings [Smah]. With this, we obtained a communication latency of 35 ms on average while supporting encrypted communications that protected the sensitive information sent to IoTWatcH's server. In the end, we found that the total latency introduced by IoTWatcH was 105 ms on average.

Storage Overhead. We measured the storage overhead imposed by IoTWatcH. Our tool does not store app information after the analysis is completed; thus, the storage cost is determined by the total storage size of the JSON object used to exchange information between the IoT apps and IoTWatcH's analyzer (Section 6.6). We evaluated the storage overhead imposed by the analysis of 160 IoT apps. On average, IoTWatcH imposes 1 KB of storage overhead, which we consider negligible.

6.8 Discussion

IoTWatcH is the first dynamic tool that performs NLP-based real-time privacy analysis in IoT apps to (1) classify IoT strings to privacy labels that are easy to understand by the user, and to (2) flag IoT apps that represent privacy concerns for the user. We implemented IoTWatcH for SmartThings IoT platform, and we plan to extend our analysis to other IoT platforms. Additionally, we analyzed 380 IoT apps and constructed a dataset to study how these apps use privacy-sensitive information. While our corpus included IoT strings extracted from SmartThings market apps, we plan to investigate other IoT platforms to construct similar datasets.

We designed and built IoTWatcH by first understanding the privacy needs of IoT users. We plan to conduct an additional study to evaluate the usability of IoT-

WatcH which is outside the scope of the current manuscript. Also, IoTWatcH's analysis would benefit from mapping the app descriptions to privacy labels. However, this is challenging as the description block of an IoT app does not explicitly state the app's privacy behavior but its functionality. We plan to use more advanced NLP techniques to address this challenge. Finally, IoTWatcH's execution requires the collection and analysis of privacy-sensitive information. We use secure `HTTPS` communications to protect the communication between IoT apps and IoTWatcH's server. In addition, IoTWatcH does not keep record of any collected information nor share this information with any third party. As a future work, a complete privacy assessment of IoTWatcH may be conducted to guarantee that user's privacy is completely preserved.

6.9 Conclusion

IoT apps access sensitive data that, if leaked, can compromise the privacy of the users. IoT platforms do not offer real-time privacy analysis that informs users about how the IoT apps use sensitive information and what they do with it. In this chapter, we introduced IoTWatcH, a novel dynamic privacy analysis tool for current IoT apps. We designed and built IoTWatcH based on a privacy survey with 123 IoT users. IoTWatcH uses NLP techniques to classify IoT strings extracted from messaging and Internet communications to a set of privacy labels at runtime. Additionally, IoTWatcH enables users to select their privacy preferences and reports its findings based on their privacy profiles. This allows users to make informed decisions about their privacy and reject apps. We analyzed 540 real IoT apps to train the NLP model and evaluate its effectiveness. IoTWatcH classifies IoT strings to correct privacy preferences with an average accuracy of 94.25% and flags 35 apps that leak data.

Finally, IoTWatch imposes minimal overhead to an IoT app's execution, introducing on average 105 ms additional latency.

CHAPTER 7

FORENSICS ANALYSIS OF RESOURCE-LIMITED DEVICE DATA

7.1 Introduction

This chapter introduces IoTDots, a novel digital forensic framework for smart settings. IoTDots has two main components: IoTDots-Modifier (ITM) and IoTDots-Analyzer (ITA). The ITM analyzes and instruments smart applications to detect, log, and store forensically-relevant information into a secure IoTDots Database (ITD) at runtime. Later, in the case of a forensic investigation, the ITA applies data processing and machine learning techniques on the ITD data to learn the overall state of the smart environment. IoTDots considers the events inferred from the ITD data and the security policies defined for the smart environment to detect security violations from users, devices, or smart apps. We evaluate IoTDots in a real-life smart environment containing 22 different off-the-shell smart devices and sensors including smart lights, thermostats, and motion sensors. Also, we considered 10 different cases of forensically-relevant activities and behavior from users and apps analyzing a total of 96209 forensics incidents. Our experimental results demonstrate the effectiveness of IoTDots in logging, storing, and processing forensically-relevant data from the environment to infer activities and behavior. Specifically, for activity detection, the framework achieves over 99% accuracy for both time-dependent (i.e., activities allowed only over certain specific time frame `t`) and time-independent (i.e., activities performed freely without considering any specific time frame) user activities. On the other hand, for the case of forensic behavior detection, the framework achieves over 96% accuracy. Finally, the analysis shows that *IoTDots* yields minimal overhead to the smart devices ans apps.

Summary of Contributions: The contributions of this chapter are as follows:

- *IoTDots:* We introduce IoTDots, a novel digital forensic framework for smart settings. The framework automatically analyzes and instruments smart apps to detect and stores forensically-relevant data from the smart environment. Then, in the event of a forensic investigation, IoTDots detects valuable forensic evidence from smart devices, apps, and users.
- *ITM:* We made the IoTDots-Modifier freely available online at `https://IoTDots-modifier.appspot.com/`. IoTDots users may utilize our automated system to perform source code analysis and instrumentation of their smart apps to enable IoTDots.
- *ITA:* Our ITA applies data processing and machine learning techniques to infer forensic activities and behaviors related to careless and malicious users, malicious apps, and tampered devices over large sets of data extracted from the smart environment.
- *Realistic Evaluation:* We evaluated IoTDots in a real smart environment with 22 off-the-shell smart devices and sensors. Our results demonstrate that IoTDots achieves very high accuracy on revealing forensic-relevant activities and behaviors with minimal overhead.

7.1.1 Differences from Existing Works

Table 7.1 summarizes the major differences between IoTDots and other IFA tools. In general, solutions prior to IoTDots are limited to policy enforcement or data provenance in IoT. IoTDots achieves the capabilities of all the considered tools with high accuracy and low overhead. In addition, the framework also offers the capability of applying deep data analysis to solve a comprehensive forensic model

Tool Name	Cross App Analysis	Consider Devices	No Platform Modification	Freely Available Online	Consider Tampered Devices	Deep Data Analysis	Tool Goals and Comments
FlowFence [FPR+16]	•	○	○	•	○	○	Protects data by enforcing data flow policies from users.
ContextIoT [JCW+17]	○	○	•	○	○	○	Detects malicious data from app context in a simulation environment.
SaINT [CBS+18]	○	•	•	•	○	○	Detects data exfiltrations from smart apps.
ProvThings [WHBG18]	•	•	•	○	○	○	Detects malicious data flow by analyzing app context.
IoTDots	•	•	•	•	•	•	Enables forensics analysis in smart settings. Considers tampered devices.

Table 7.1: Comparison between IoTDots and other IFA tools.

that includes challenges related to careless users, malicious users, malicious apps, and tampered devices. Compared to these prior works, IoTDots presents a lightweight solution that automatically analyses smart apps source code and performs app instrumentation to collect forensically-relevant data from a smart environment. Also, IoTDots includes tampered devices in its analysis, something that has not been considered by previous works. Additionally, the IoTDots framework provides the means to apply machine learning algorithms to analyze the logged data and detect anomalous activities and malicious behavior from users, devices, and smart apps. IoTDots flags the actions that potentially violate the security policies of the smart environment and may help to hold the perpetrators accountable in a holistic manner during forensic investigations.

7.2 Problem and Threat Model

In this section, we introduce the problem through a use case. Then, we provide assumptions and definitions of different terms in the context of IoTDots. Finally, we articulate the threat model.

7.2.1 Problem Definition

This dissertation assumes that there exists an office O. The office has deployed several devices to create a fully equipped smart environment. The topology of the smart environment in O includes devices like smart thermostats, locks, lights, presence sensors, security cameras, and smoke detectors. We also assume that the general manager, Bob, is the only person in O with administrative rights to handle the apps that control the smart devices. By policy, these are IoTDots-modified apps, and they are the only ones authorized to manage the devices inside O. Bob, however,

is not authorized to modify the security policies in place for the smart environment. Finally, the security policies of O prohibit the presence of any person between 8:00 pm to 7:00 am from Monday through Friday and anytime during the weekends. At some point, a fire incident inside O has caused the loss of sensitive information along with important economic consequences. After the incident, the insurance company requests a forensic investigation.

We introduce IoTDots as a novel framework that utilizes the logs extracted from the IoT-modified apps to perform the forensic analysis of the events in O. Indeed, IoTDots can be used in conjunction with traditional forensic analysis tools and techniques to hold the person, smart app, or device (if any) accountable if a case of negligence or deliberate violation of security policies is detected. By using IoTDots, we can answer several forensic questions: (1) What was happening inside O right before the fire incident had occurred (e.g., right before the smart smoke detector started sensing the smoke presence)? (2) Was anyone inside the room (e.g., presence sensor state changed)? (3) Was the door opened/closed anytime before the fire incident (e.g., smart lock state changed)? Further, the forensic framework would be able to evaluate the different states of connected devices and the overall smart environment. Them, this information can be matched with the security policies in place to detect any (intentional or involuntary) violation of the security policies. Some of these violations could be: workers accessing the office at night time (i.e., careless unauthorized activities), Bob tampering the security camera to avoid video recording (i.e., device tampering), or smart apps running malicious code to modify the values of the presence sensor (i.e., malicious behavior).

7.2.2 Assumptions and Definitions

We assume and define some terms that we use to explain IoTDots threat model and further sections.

- **Assumption 1–Smart Environment Security Policies:** We assume that a smart environment is centrally-managed by well-defined security policies, and users are not authorized/allowed to change its pre-defined settings. For instance, configuration settings on thermostat's and smart lock's cannot be changed unless management requests and approves a modification in the smart environment settings. With this, we assume that security policies regulate both the access level and settings of the smart environment.
- **Definition 1–Forensically-relevant Data:** We define *Forensically-relevant Information* as the smart apps events, device's actions, user-defined inputs, device's information, and time and location information that IoTDots extracts/logs from the smart apps. This information typically reflects on the state of the different smart devices and sensors as a function of the users' and smart apps' activities.
- **Definition 2–Authorized User:** We use the term *authorized user* to define users that are authorized to perform activities or access specific locations and devices inside the smart environment, but only during the permitted hours specified by the security policies in place.
- **Definition 3–Unauthorized User:** We use the term *unauthorized user* to define those users that carelessly or deliberately try to change the pre-defined settings of the smart environment. Also, unauthorized users are those who perform activities and access the smart environment locations during not-permitted hours. Depending on the specific time, date, the context of the activity, and

the smart environment policies, authorized users can act in an unauthorized manner.

- **Definition 4–Attacker (insider or outsider):** We consider the *attacker* as an insider or outsider that maliciously tries to disrupt or take control over the smart environment or any of its entities (i.e., devices and sensors) to learn user behaviors, steal sensitive information, gain access to the systems, or even interrupt the normal operations of IoTDots.
- **Definition 5–Malicious App:** We call *malicious app* a smart app with malicious code intended to leak sensitive information from users and the smart environment to attackers, change/replace legitimate IoTDots logs with the intention to hide malicious behavior, or perform any other malicious activity inside the smart setup (e.g., side channel attacks).
- **Definition 6–Tampered Device:** We define as *tampered device* a smart entity that is forced to change its expected (based on the user activity) state by a different one that does not reflect the real overall state of the smart environment. Also, this group includes devices that are deliberately relocated, disabled, or turned off without the required authorization.
- **Definition 7–Regular User Activities:** Any action performed by the authorized users inside the smart environment that does not violate the security policies in place.

7.2.3 Threat Model

For IoTDots, we consider two different sets of forensic threats: (1) *anomalous user activities* and (2) *malicious behavior from users, smart apps, and devices.*

Anomalous User Activity. We consider threats related to user activities based on their time dependency, that is, we explicitly consider time-based actions to evaluate IoTDots in detecting anomalous user activities. Anomalous user activity defines any careless and unintentional action performed by an unauthorized user inside the smart environment that is considered a violation of the security policies in place. Since IoTDots considers these policies to perform its analysis, we categorize user activities as *time-independent* and *time-dependent*. In the first group, we include all the user actions whose execution time is irrelevant to IoTDots. On the other hand, the second group gathers those actions whose execution time is strictly regulated by the security policies of the smart environment (e.g., the presence of an authorized user in O is only a violation between 8:00 pm and 7:00 am).

Malicious Behavior. We consider as malicious behavior any intentional action performed by authorized users, unauthorized users, smart apps, and/or smart devices that clearly violates the security policies and can be considered a threat for other users and/or the overall state of the smart environment. These forensic behaviors include *impersonation attacks*, *false data injection attacks*, *reply attacks*, *Denial-of-Service*, and *side channel attacks*. IoTDots considers all these attacks as they may affect the smart environment and the overall performance of the forensic framework.

We provide details of specific forensic activities and behaviors considered in our threat model in Table 7.2. For every activity/behavior, we specify the attack method utilized, time dependency, and specific examples in the context of the problem considered in this chapter. Later, these activities and behavior are utilized to test the efficacy and performance of IoTDots (Section 7.5).

Threat	Time-dependency	Attack Method	Specific Attack Example
Activity-1	Time-independent	Tampered device	Bob changes the orientation of the presence sensor to fit in a new equipment.
Activity-2	Time-independent	Careless unauthorized user	Bob manually lowers the temperature of smart thermostat from home, the night before of a big meeting with the stakeholders
Activity-3	Time-dependent	Careless unauthorized user	Bob is inside a restricted area (servers room) at 8:45 pm.
Activity-4	Time-dependent	Careless unauthorized user	Bob is getting into the office at 8:45 pm.
Activity-5	Time-dependent	Careless unauthorized user	Bob is using the secure pin to unlock the smart lock.
Behavior-1	Time-independent	Tampered device	Bob disables the smart camera to stop recording while he is performing unauthorized activities inside the office O.
Behavior-2	Time-independent	Impersonation Attack	Alice gets access to the office O using the smart lock pin that she obtained through a malicious app that leaks information.
Behavior-3	Time-independent	False Data Injection Reply Attack	The presence sensor app reports inverted states to IoTDots.
Behavior-4	Time-independent	Denial-of-Service Attack	The smoke detector app triggers the smart windows and locks open by reporting false user presence.
Behavior-5	Time-independent	Side Channel Attack	The smart light app disable the compromised smart camera by creating a specific light on/off pattern.

Table 7.2: Summary of the threat model (forensically-valuable activities and behaviors) considered in this chapter. In Section 7.5), we utilize these specific activities and behaviors to evaluate the efficacy of IoTDots.

7.3 IoTDots

Figure 7.1 depicts the general architecture of IoTDots, which includes two main components: IoTDots-*Modifier* (ITM) and IoTDots-*Analyzer* (ITA). The first part of IoTDots involves the ITM. Here, the IoTDots user downloads the original smart app source from one of the freely available online repositories ❶. Then, the ITM automatically analyzes and instruments the smart apps at compile time to allow logging of forensically-relevant data to the secure ITD ❷. The ITM processes include (1) the analysis of the source code of the smart apps [CBS+18, SaI] and (2) the smart app instrumentation. Then, upon utilization, the IoTDots-modified apps send forensic logs containing states and actions occurring in the smart environment to the ITD ❸.

The second part of the architecture involves the ITA (Figure 7.1). This stage performs data processing and applies machine learning techniques on the IoTDots-collected data ❹. The purpose of this analysis is to extract forensically-relevant information from the IoTDots logs. This information includes devices' states and actions, configuration changes, notifications, etc. Collecting this data may potentially allow learning the state of the smart environment during specific incidents and provide insights about the users', apps', and devices' activities. Finally, IoTDots correlates these activities with the security policies defined for the smart environment to detect anomalous user activities and potential malicious behaviors from users, smart apps, and devices ❺. In the next sub-sections, we detail essential aspects of these operations.

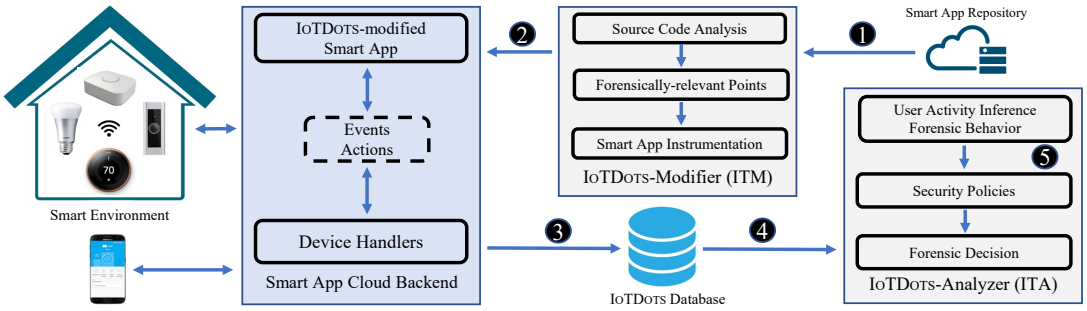


Figure 7.1: The architecture of the ITM. IoTDots-Modifier analyzes the smart apps to detect and send forensic-relevant data logs to the ITD at runtime. Later, during the event of a forensic investigation, the ITA analyzes the data and infers forensically-relevant activities and behavior from users and smart apps.

7.3.1 Forensically-valuable features in IoTDots

In Chapter 2 (Section 2.7), we described smart app features that could potentially contain relevant information for forensic purposes. We consider *events* since they define physical changes in the smart environment setup based on the user's activities (e.g, the door's sate was "unlocked" at an unauthorized time). Also, our architecture considers *actions* extracted from smart apps because they potentially contain valuable timing information that can define changes in forensic timelines (e.g., the fire event occurred after the door was unlocked). These actions are correlated by IoTDots to detect tampered devices in the smart environment. Additionally, we consider *user-defined inputs* in our architecture. Changes in user-defined inputs are critical during forensic analysis since unauthorized modifications directly impact the execution of the apps' events, actions, and notification mechanisms (e.g., Bob changed the notification recipient of the smoke detector app right before the fire event). Other features considered by IoTDots are *time* and *location* information. Including time and location information in our analysis not only provides valuable timing information from forensic actions but also avoid reply attacks from malicious users trying to inject fake activity data into the smart environment (e.g., it prevents Alice from using

old benign logs to hide current potentially dangerous activities). Finally, IoTDots tracks smart apps exfiltrations to correlate the logged data with information obtained from notifications. For instance, the device state obtained through specific IoTDots logs can be confirmed or disproved by checking the notifications sent by the smart app. This process may help to identify tampered devices or attackers trying to inject fake device state data into the system to disrupt the normal operations of IoTDots.

7.3.2 IoTDots Modifier (ITM)

As mentioned before, the ITM analyzes the source code from the original smart applications to detect forensically-relevant points and automatically inserts specific code for logging purposes.

Source Code Analysis and Instrumentation

The purpose of the source code analysis in IoTDots is to automatically detect the previously defined forensically-relevant points inside the smart app.

The first step toward analyzing the smart app source code is to create an intermediate representation (IR) of the app and model the application's structure. In general, a smart app's structure follows the sensor-computation-actuator paradigm, which means, smart solutions are generally designed the same way regardless of their specific application and complexity [CBS+18]. This facilitates the design of a generalized solution through the IR to integrate apps and devices from multiple IoT platforms into IoTDots's analysis. Furthermore, modeling the smart app permits the extraction of smart apps' entry points, events, and control flow of data. Also, it allows for identifying the data sources and exfiltrations methods (i.e., sink functions) that are used to define (1) the origin of the forensic-relevant information and

(2) how this information is sent out from the smart apps and to which destination, respectively.

We use an Abstract Syntax Tree (AST) representation of the smart apps for building algorithms to find forensically-relevant points inside the source code's IR. Once IoTDots generates the AST, the source code analysis starts by constructing an Interprocedural Control Flow Graph (ICFG) of the apps and by extracting the specific nodes that define events handlers, actions, and user inputs. Particularly, we implement the `ASTTransformationCollector-CodeVisitor` [Gro] at compile-time and create a customizer that analyzes each node of the ICFG and looks for forensically-relevant points. While analyzing the ICFG, IoTDots considers the following smart apps' distinctiveness:

App Permissions. In smart apps, permissions include the smart device information, the device capabilities, and the user inputs defined at install time. The ITM considers the permissions to flag nodes from the ICFG that potentially contain device and user-defined information. We noticed that, in addition to install-time, permissions are modified through updates in the smart apps. For that reason, the ITM flags both, `installed()`- and `updated()`-related events in the smart app. Also, we flag `subscribe()`-related methods utilized to define the different device's events.

Entry Points. Smart apps define multiple entry points to subscribe to the different events occurring in the smart environment. In general, to subscribe to such events, smart apps consider the subscribed event and the *event handler* invoked every time the event occurs. Hence, such event handlers can be utilized to extract the device actions. As a result, the ITM flags the ICFG nodes that define the event handlers to extract the different device's actions from the smart app.

Handler-dependent Methods. Once IoTDots flags the event handlers, the ITM visits all the methods invoked by every specific handler. With this, the ITM constructs

the different call graph branches of the ICFG starting from the event handlers to the different smart app's sinks.

Sinks. The ITM flags the smart app's sinks invoked in every ICFG branch. Sink functions contain considerable forensic information. By analyzing the sinks, IoTDots can confirm that notifications were sent to the authorized recipients right after forensic events occur in the smart environment. Additionally, sink's information can be used to correlate device states acquired via IoTDots logs which may potentially flag malicious activities trying to disrupt the normal operations of the IoTDots. For instance, the logs from a smart app controlling a motion sensor can be modified to hide user presence. However, by analyzing the notification received by the user, IoTDots can detect malicious behavior if the content from the sink function does not match with the recorded log. Also, sink information may flag unauthorized app settings (e.g., modification of the sink's recipients). Finally, in case of missing sinks (e.g., notifications never received by the authorized users) IoTDots analysis may provide evidence of devices that were disabled or turned off.

Once the source code analysis is completed, IoTDots performs smart app instrumentation to insert custom-defined code at every forensically-relevant point location to enable logging. In Listing 7.1 and Listing 7.2, we show how a smart application is modified by IoTDots for forensic purposes. In Listing 7.1, IoTDots flags a user-defined input `recipients`. This specific input defines the phone number to which all the notification from the smart apps are sent to. As explained earlier in this section, these types of inputs are very critical for forensic purposes since careless or malicious users can modify this info anytime without being noticed, which can negatively impact the final purpose of tracking the events and actions executed by apps. Going back to the previous example, after labeling the user input, IoTDots modifies the

app by inserting a forensic log (Listing 7.2). Here, the `log.IoTDots` function defines an `https` request that securely pushes the log information to the remote ITD.

Listing 7.1: A sample code for a smart app

```
1 /* A section of a code block of an original smart app */
2
3 section("Via a push notification and/or an SMS message") {
4     input("recipients", "contact", title: "Send notifications to") {
5         input "phone", "phone", title: "Enter a phone number to get SMS", required: false
6     }
7 }
```

Listing 7.2: A sample code for an IoTDots-modified smart app

```
1 /* A section of a code block of an \namefor-modified smart app */
2
3 section("Via a push notification and/or an SMS message") {
4     input("recipients", "contact", title: "Send notifications to") {
5         input "phone", "phone", title: "Enter a phone number to get SMS", required: false
6     }
7     log.IoTDots ("New recipient defined: $phone") //IoTDots log
8 }
```

Further, Algorithm 6 details the processes performed by the ITM to analyze and modify a smart app. During *Analysis*, the application source code is loaded into the IoTDots modifier (Line 1). Recall that in this chapter, we are targeting open source smart app application platforms, so the availability of the source code is assumed. Then, the IR and AST are generated and extracted in Lines 2 and 3, respectively.

From here, we extract forensically-relevant features to construct the ICFG of the app, starting from the event-handler methods (Line 4). Once the ICFG is obtained, the nodes are explored and all the forensic-relevant points are flagged in Line 7. This step concludes the Analysis process in Algorithm 6. Finally, the *Instrumentation* process customizes the smart apps by inserting the IoTDots logs in Line 12.

Runtime Data Collection. Source code analysis and app instrumentation do not guarantee completeness of the smart environment taxonomy and description based solely on the device activities. While most of the data flow paths may be inferred from the source code analysis, some IoT platforms utilize programming languages that allow dynamic method invocation and the use of closures which may generate

Algorithm 6: Steps in the IoTDots-Modifier (ITM).

```
appSC ← app source code
Analysis:
IR ← generateAST(appSC)
AST ← generateAST(IR)
ICFG ← createICFG(AST)
if Exists ICFG then
    for nodes in ICFG do
        forensicPT ← forensic-relevant points
    end for
end if
Instrumentation:
if forensicPT then
    for points in forensicPT do
        Insert IoTDots Logs
    end for
end if
```

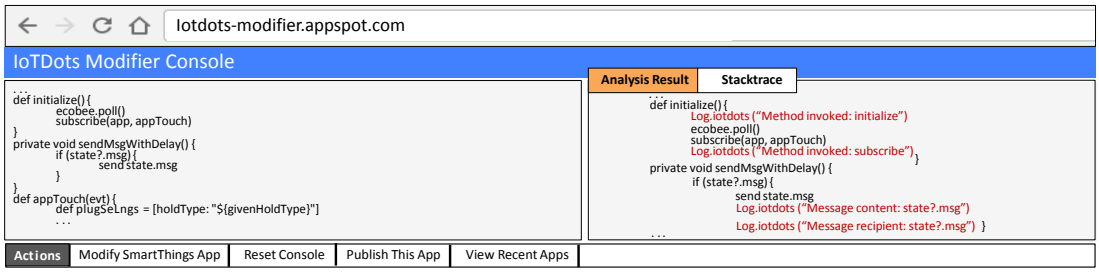


Figure 7.2: IoTDots is available online at https://iotdots-modifier.appspot.com/.

different data flow at runtime [CBS+18]. By combining handler-dependent methods and sink logging, IoTDots performs data collection at runtime which guarantees, based on the device activities, accuracy on the collected data and completeness on the smart environment characterization.

ITM Online Web App. We made the ITM available online at: `https://IoTDots-modifier.appspot.com/`. Figure 7.2 depicts details of the online version of the ITM. On the left side, the user simply inputs the original source code of the app that needs to be modified to enable IoTDots analysis. Then, on the right panel, the tool returns the modified app that permits logging of all the forensically-relevant data.

7.3.3 IoTDots Analyzer (ITA)

The ITD stores logs obtained from smart apps at runtime, which permits that the information from events and actions in a smart environment can be later utilized for forensic purposes. For successfully extracting information from the logs, we implement an ITA that performs the following actions on the ITD data.

Labeling. This step classifies and stamps the data in the ITD. Once collected, the data is used to feed machine learning models to complete the analysis. Here, IoTDots labels the ITD data based on timing information, the specific device generating the data, the specific exfiltraiting app, and the location of devices and controlling apps. For IoTDots, labeling constitutes an essential pre-processing step towards the classification process that extracts forensic information from the data.

Detection. Labeling prepares the ITD data for classification. For simple smart environments, data provenance and cross-app analysis using the labelled data may provide information with forensic content. However, more complex scenarios (i.e., higher number of devices or scenarios that include tampered devices) may require deeper analysis on the collected data. IoTDots is capable of not only labeling the logs based on forensic criteria (i.e., timestamp, location, device info), but also analyzing the data to infer user activity and detect forensic behavior of users, smart apps, and devices. For this purpose, the ITA applies machine learning techniques on the ITD-labeled data to classify and extract legitimate and red-flagged forensic logs. Specifically, IoTDots utilizes Markov-Chain-based detection mechanisms which we describe in details in Section 7.4.

Device Cooperation. IoTDots infers tampered devices based on the analysis of collected logs from multiple devices. For this, the Markov Chain model analyses the state of all the devices in the smart environment at any given time and detects malicious or unexpected states by comparing data from similar devices sharing the

same context (e.g., similar locations). We call this process *device cooperation*. During device cooperation analysis, if one device is compromised or tampered, the information collected from other trusted devices inside the environment is used to detect the one reporting fake or unexpected data. For instance, consider a smart light which function is controlled by a motion sensor. During normal operations, if the sensor detects any motion, the light is turned on. However, in the event of a compromised state of the smart light, the light may not operate appropriately nor follow the motion sensor states. In this case, a third device (e.g., a smart thermostat that also includes a presence sensor) can act as an *authority* to decide which device is misreporting states. If both the motion sensor and the thermostat achieves similar states, then the smart light can be considered as tampered. On the other hand, if the smart light successfully follows the smart thermostat actions, then the motion sensor is flagged as tampered. In summary, our classification model relies on third-party entities to resolve conflicts between device states inside the smart environment. However, for the cases where compromised devices are majority, device cooperation losses effectiveness. For instance, going back to the previous example, if both the motion sensor and the smart light are tampered, it would be hard to define which device is hampering the normal operation of the system.

Multi-class Approach. In this chapter, we utilize a multi-class classification approach to infer forensic activities and behaviors. The ITA classifies a log as legitimate if, as a result of the detection process, IoTDots infers a regular user activity from the log. On the contrary, red-flagged logs may help to infer events and actions related to anomalous or unauthorized user activities (e.g., careless device replacement or relocation, changes on user-defined inputs, unauthorized changes on setup topologies, or misplace or disable of any part of the smart setup). Also, red-flagged logs can be used by the ITA to infer malicious behavior of users. For instance, these logs

may detect users carelessly or maliciously accessing to restricted areas of the smart environment after work hours, malicious apps forcing the smart setup to behave differently than expected, or tampered devices that maliciously report false states to the smart environment. Domains like IoT, with multiple apps, devices, sensors, and users, requires a multi-classifier approach rather than binary classification (i.e., benign/malicious). Additionally, any industry standard or government guidelines (e.g., NIST) expect multi-level outcome in lieu of simple benign/malicious decisions from a forensic analysis.

Algorithm 7 details the processes performed by the ITA to analyze the forensics data. In Line 1, the *iotLogs* variable is initialized with the content of the ITD. Additionally, the initialization step includes updating the variable *policy* with the security policies that were valid for the smart environment by the time that the logs were acquired. Then, in Line 5 the IoTDots's logs are organized and labeled based on the timing information, the type of devices that generated the logs, and the location information of the devices. With this information, a Markov Chain-based machine learning model is applied on the data to (1) detect user actions in the smart environment (Line 9) and (2) detect behaviour of users and smart apps, which are correlated with the security policies in *policy* (Line 10). Finally, if a forensic violation is detected, the flag *anomaly* is set to TRUE.

7.4 Forensic Evidence Detection in IoTDots

In this section, we describe the analysis techniques used to identify activities and behaviors in a smart environment from the collected forensic data. Particularly, we implement a Markov Chain-based detection technique in IoTDots.

Algorithm 7: Steps in the IoTDots-Analyzer (ITA).

```
iotdLogs ← ITD
policy ← smart environment security policies
anomaly ← FALSE
Labeling:
for each Log in iotdLogs do
    label Log by Time, Device, Location
    MLdata ← labels
end for
Detection:
for data in MLdata do
    Evaluates user activity userAct
    ML ← MLanalyzer(userAct, policy)
    if Anomaly detected in ML then
        anomaly ← TRUE
    end if
end for
```

7.4.1 IoTDots Data Characterization

As noted earlier, IoTDots collects data from a smart environment in an ITD that includes timing information, sensor information, device state, location, and log's timestamp. For a specific time slot t, the collected data can be represented by:

$$Data\ array, E_t = \{T, S, D, M, L, t\}, \tag{7.1}$$

Where T represents the timing features, S represents the set of sensors' features, D is the set of device features, M are the features extracted from the controlling devices (smartphone/tablet), L is the set of location features of the controlling devices, and t represents the IoTDots log's timestamp information. We describe the characteristics of these features below.

Timing Features (T). A smart environment consists of several sensors and devices that change their states based on different user activities and controlling commands. In this context, some devices perform instantaneous tasks (e.g., switching lights on and off based on motion) while some devices perform a task over a period (e.g.,

changing temperature over time). IoTDots considers this timing information as a feature to infer the overall state of the smart environment at a specific time.

Sensor Features (S). Sensors in a smart environment work as a trigger to different actions. A smart environment may contain several different sensors (e.g., motion sensor, temperature sensor, presence sensor) attached to multiple devices. These sensors sense the changes in devices' proximity and help the devices to take autonomous decisions such as switching lights on motion or triggering fire alarm after smoke is detected. Depending on the nature of the sensors, sensor data can be both logical (active/inactive) or numerical values. For IoTDots, we collect both numerical values and logical states of the sensors and create the state of the smart environment at a specific time. We represent the change in both logical states and a numerical value of a sensor as a binary input (1 if active/change and 0 otherwise) to create a forensic data matrix to train the detection algorithm.

Device Features (D). A smart environment supports different devices that may be or may be not connected to a smart hub (Chapter 2.6), to different sensors, or other devices. These devices can perform multiple tasks as standalone (e.g., smart thermostat) or as ad-hoc entities (e.g., automatic door controlled by smart lock, camera, and presence sensor). For different user activities and input commands, these devices change their states (active/inactive) autonomously. IoTDots considers the device state data as part of its analysis.

Controller Device Features (M) In a smart environment, users can use smartphones or tablets to control devices from the associated smart apps (Chapter 2.6). IoTDots collects the control command generated from controller devices to understand the smart environment settings and the user intended operations in the smart environment at any moment.

Location Features (L). The smart environment allows controlling the connected devices from both inside and outside of the location. Location information of a given command to the devices has very high forensic value and is critical to infer illegal activities either from insiders or outsiders. IoTDots considers the location of both the controller and the smart devices as a feature to understand activities occurring in the smart environment.

Timestamp (t). Finally, our model includes specific timestamps in all the data collected by IoTDots. The timestamps provide accurate information about the exact date and time of occurrence of the events. Since IoTDots considers malicious apps and tampered devices, timing information collected from the smart environment cannot be completely trusted. Additionally, the use of timestamps prevents from reply attacks from malicious insiders or outsiders.

7.4.2 Analytical Model used in IoTDots

ITA collects the information above and creates a binary state array (1 for active status and 0 for inactive status) to represent the state of the smart environment at any specific time t. Thus, we represent the state of the smart environment as a n-bit binary number, where n is the number of features extracted from the logs. The total number of possible states of a smart environment with n number of features (sensors' states, devices' states, controller devices, and locations) would be $m = 2^n$. IoTDots utilizes the timed binary state of the smart environment to train a Markov Chain classification model to detect forensically-valuable behavior from users, smart apps, and devices. The Markov Chain model benefits from two main assumptions: (1) the occurrence probability of a specific state s_i only depends only on the previous state s_{i-1} and (2) the transition between two consecutive states is independent of time and does not depend on any previous condition. Based on these assumptions, we illustrate

the Markov Chain model by the following equation [SAU17].

$$P(X_{t+1} = x | X_1 = x_1, X_2 = x_2..., X_t = x_t) = P(X_{t+1} = x | X_t = x_t), \quad when,\ P(X_1 = x_1, X_2 = x_2..., X_t = x_t) > 0, \tag{7.2}$$

where X_t and X_{t+1} denotes state of the smart environment at time t and $t+1$, respectively. Lets assume the smart environment has the state i at time t and j at time $t+1$. If P_{ij} illustrates the transition probability between state i to j, the state transition matrix for m number of states of a smart environment can be represented by the following matrix.

$$P = \begin{bmatrix} P_{11} & P_{12} & P_{13} & \dots & \dots & P_{1m} \\ P_{21} & P_{22} & P_{23} & \dots & \dots & P_{2m} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ P_{m1} & P_{m2} & P_{m3} & \dots & \dots & P_{mm} \end{bmatrix}, \tag{7.3}$$

To calculate each element of the transition matrix, lets assume the smart environment has $X_0, X_1, \dots, X_T$ states at a given time $t = 0, 1, \dots, T$. Then, each element of the transition matrix can be represented by the following equation [SAU17]:

$$P_{ij} = \frac{N_{ij}}{N_i}, \tag{7.4}$$

where N_{ij} is the total number of transition from X_t to X_{t+1} over a certain period. From the state transition matrix, Markov Chain model calculates the probability of occurring a state or sequence of states. To predict the probability of occurring a state, lets assume the initial probability distribution of the Markov Chain model as follows:

$$Q = \begin{bmatrix} q_1 & q_2 & q_3 & \dots & \dots & q_m \end{bmatrix}, \tag{7.5}$$

where, q_m is the probability that the model is in state m at time 0. The probability of observing a sequence of states can be calculated by the following equation:

$$P(X_1, X_2, \ldots, X_T) = q_{x1} \prod_{2}^{T} P_{X_{t-1}Xt}. \tag{7.6}$$

7.4.3 Data Binarization in IoTDots

Due to the nature of the IoTDots-collected data, the acquired logs are not always Markov Chain-ready. In some cases, one needs to implement a *binarizer* that converts numeric data (e.g., sensor readings) into binary values that can be directly interpreted by the Markov Chain model. Table 7.3 summarizes the use of the binarizer for different types of data in IoTDots. We found that, for the types of devices and sensors utilized in our evaluation, only a few cases of sensor readings require *binarization*. In these specific cases, IoTDots utilizes user-defined inputs in the smart apps to define threshold values that automatically convert numeric readings into binary values. For instance, for a temperature sensor, IoTDots logs the sensor value for every device state change. Then, during analysis, these values are compared against the temperature value that the user set at install time.

IoTDots Logs	Data Type	Requires Binarizer	Comments
Location	Binary	○	–
Device	Binary	○	–
Sensor states	Binary	○	–
Sensor values	Numeric	●	user-defined inputs
Controller	Binary	○	–

Table 7.3: IoTDots implements a binarizer to transform multi-class numeric logs to binary numeric values. Logs of type Location refers to the Location Modes "Office" or "Other".

For sensor readings over the threshold, IoTDots feeds a value 1 to the Markov Chain model for that specific variable. On the other hand, for sensor readings below

the user-defined threshold, IoTDots feeds a value 0 to the Markov Chain model. *The use of user-defined inputs to implement the binarizer provides information to the* IoTDots *framework that may help to determine if the devices were compromised or were behaving unexpectedly during the forensic incident.*

7.5 Performance Evaluation

We evaluate the efficacy of IoTDots to detect a diverse set of forensic incidents including anomalous user activities and behaviors from users, smart apps and devices. Specifically, we aim to answer the following research questions:

- **RQ1: Forensic Activities.** How does IoTDots perform in detecting regular activities in the smart environment? (Section 7.5.3).
- **RQ2: Forensic Behavior from Users.** How does IoTDots perform in detecting unauthorized and malicious behaviors from users? (Section 7.5.4).
- **RQ3: Forensic Behavior from Devices and Apps.** How does IoTDots perform in detecting unauthorized and malicious behaviors from devices and apps? (Section 7.5.5).
- **RQ4: Overhead.** What is the overhead introduced by IoTDots? (Section 7.5.6).

For a better analysis, we obtained Institutional Review Board (IRB) approval to test IoTDots in real scenarios. We implemented a smart office environment with specific security rules enforced in the system (e.g., time-restricted location access for users and restricted device re-configuration) where real users perform regular activities. Further, we consider a group of users that carelessly violate the security rules of the environment by performing anomalous activities such as accessing office

locations at an unauthorized time or changing the configuration/topology of the smart environment. Finally, we consider specific forensic behavior like poisoning data from a specific number of devices using malicious apps. As detailed in Section 7.3, our framework first detect the forensically-valuable user activities to classify them as regular or anomalous based on the security policies in place. In parallel, IoTDots detects forensic behaviors. We built a Markov Chain-based detection method for this purpose and trained our model with data collected by IoTDots from the real-life smart office setup. Also, we evaluated the general overhead introduced by IoTDots to the smart devices and the cloud-based servers.

7.5.1 IoTDots Implementation

The initial version of IoTDots implements solutions for Samsung SmartThings platform. With this, we focus on a smart platform that (1) defines the highest amount of different smart devices and apps in the market [FJP16]; (2) it is open source, so the apps' source code is freely available online; and (3) it makes extensive API's documentation available to developers [Smah].

Training Environment Setup

For training purposes, we built a real-life smart office environment with 22 different smart devices and popular smart home apps available in the SmartThings App Market (Table 7.4). Then, we designed several forensically-valuable activities that the users typically perform in most smart settings. The set of considered activities included the following scenarios:

- **Time-dependent access**: We allowed a single user to access all the office locations during office hours to observe his/her movements and activity patterns inside the smart environment.

- **Restricted access**: We repeated the same activities performed during time-depended access, but this time while enforcing restricted access policies for specific places in the smart environment, again for one single user.
- **Multi-user environment**: We combined the two previous scenarios with multiple users interacting with the smart setting. In the multi-user scenario, we again considered time-dependent and restricted access to emulate the real-life smart office setup.
- **Reconfiguration rule**: We considered a smart office environment as a controlled environment where reconfiguration of the smart environment is possible only in a timely and supervised manner. Any reconfiguration outside specific pre-defined time slots is considered anomalous. We provided time-specific rules to collect data while re-configuring the smart environment.

Data Collection

For data collection, we used the ITM to automatically instrument the SmartThing apps and insert forensic logs. Then, while utilizing the apps, IoTDots-collected data was sent to the ITD. For this phase, we explicitly considered scenarios where 10 different authorized users interacted with a smart environment consisting of 22 smart devices and sensors (Table 7.4) during a period of seven days. Our dataset comprises 84209 forensically-valuable incidents that include different regular user activities only. For these, we asked the users to perform their daily tasks inside the smart office setup. On the other hand, for the anomalous activities, we asked the users (considered now as an unauthorized user for this specific experiment) to implicitly violate the security policies defined for the smart environment in specific ways as described in Section 7.2 (Table 7.2). In total, we collected forensic data from 30 different experiments resulting in over 4500 instances from the five different anomalous activities.

Device Type	Model	Total
Smart Home Hub	Samsung SamrtThings Hub	1
Smart Light	Philips Hue Light Bulb	5
Smart Lock	Yale B1L Lock with Z-Wave Push Button Deadbolt	2
Fire Alarm	First Alert 2-in-1 Z-Wave Smoke Detector and Carbon Monoxide Alarm	1
Smart Monitoring System	Arlo by NETGEAR Security System	2
Smart Thermostat	Ecobee 4 Smart Thermostat	1
Motion Sensor Light Sensor Temperature Sensor	Fibaro FGMS-001 ZW5 Motion Sensor with Z-Wave Plus Multi-sensor	6
Door Sensor	Samsung Multipurpose Sensor	4

Table 7.4: List of smart devices and sensors used during the data collection stage in IoTDots's evaluation.

We also collected data related to forensically-relevant behavior from the users, smart apps, and devices. Forensic behavior from users' actions considered a user that tries to modify, tamper, or remove the devices from the smart environment. This behavior matches Behavior-1 from our threat model (Section 7.2) and intends to explicitly change the original configuration and prevent IoTDots from sending incriminating logs to the ITD. We allowed the users to tamper (e.g., disable, relocate) and remove smart devices to simulate this scenario and collect data for IoTDots. On the other hand, for forensic behavior from the smart apps, we installed the modified version of malicious smart apps collected from the IoTBench [IoT] and ContextIoT [Con17] projects. For Behavior-2, we created two different apps for a smart lock that leak the access code to an unauthorized person. For Behavior-3, we built an app that injects forged data in a fire alarm and triggers the alarm maliciously. For Behavior-4, we developed an app that can power down the smart thermostat after setting a

specific input temperature. Finally, for Behavior-5, we created an app through which a specific light pattern can trigger the smart camera. To collect the behavior-related data, we performed the attack scenarios multiple times. We also considered multi-attacker scenarios where multiple attackers perform different attacks at once. In total, for these malicious scenarios, we collected 50 different forensic datasets comprised into 7500 different data instances. Table 7.5 summarizes the distribution of the datasets during the collection process based on the three different experiments.

Type of Experiment	**Dataset Size**	**% of Total**
Regular	84209	87.5
Anomalous	4500	4.7
Forensic Behavior	7500	7.8
Total	96209	100

Table 7.5: Distribution of IoTDots evaluation data among three different types of experiments.

Finally, we used 75% of both user activity data (regular and anomalous) to train the Markov Chain model and then combined the remaining 25% of the data along with the behavior-related data to test the performance of the ITA.

7.5.2 Performance Metrics

We chose six different performance metrics to evaluate the effectiveness of IoTDots in a smart environment: True Positive rate or Recall rate (TPR), False Negative rate (FNR), True Negative rate or Specificity (TNR), False Positive rate (FPR), Accuracy, and F-score. To define these metrics, we use following four terms:

- *True Positive (TP):* True positive refers to the total number of correctly identified benign activities.
- *True Negative (TN):* True negative specifies the number of correctly detected malicious activities.

- *False Positive (FP):* False positive defines the number of instances when a malicious activity is mistaken as benign activities.
- *False Negative (FN):* False negative states the number of benign activities that are mistaken as malicious activities.

Then, the performance metrics can be calculated as follows:

$$TP\ rate\ (Recall\ Rate) = \frac{TP}{TP + FN}, \tag{7.7}$$

$$TN\ rate\ (Precision\ rate) = \frac{TN}{TN + FP}, \tag{7.8}$$

$$FP\ rate = \frac{FP}{TN + FP}, \tag{7.9}$$

$$FN\ rate = \frac{FN}{TP + FN}, \tag{7.10}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{7.11}$$

$$F - score = \frac{2 * Recall\ rate * Precision\ rate}{Recall\ rate + Precision\ rate}. \tag{7.12}$$

7.5.3 Forensic Activity Detection from Users

The state of the interconnected devices inside the smart environment depends on the on-going user activities. For example, while a user moves from one place to another, several devices and sensors become active. The changes in device states can be an instantaneous event (a specific event at a specific time such as switching on a light) or a combination of subsequent events over a certain period (motion from one place to another).

Table 7.6 shows the evaluation results of IoTDots for user activity inference. One can observe that, for time-independent activities (i.e., Activity-1 and Activity-2), IoTDots achieved accuracy (i.e., ACC) and F-score values of over 98% and 96%, respectively. True Positive Rate (TPR) and True Negative Rate (TNR) are also high

User Activity	TPR	FNR	TNR	FPR	ACC	F-Score
Activity-1	0.9926	0.0074	0.9562	0.0438	0.9907	0.9740
Activity-2	0.9904	0.0096	0.9438	0.0562	0.9880	0.9665
Activity-3	0.9860	0.0140	0.8623	0.1377	0.9739	0.9197
Activity-4	0.9721	0.0279	0.8614	0.1386	0.9664	0.9133
Activity-5	0.9584	0.0416	0.8861	0.1139	0.9547	0.9208

Table 7.6: Performance evaluation of IoTDots for inferring forensically-valuable user activities.

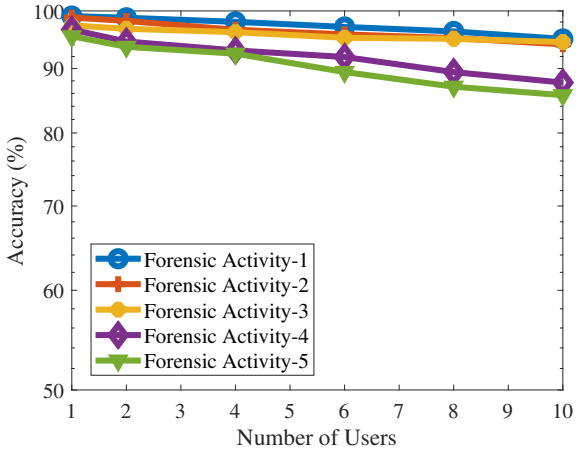


Figure 7.3: Accuracy of IoTDots in inferring activities in multi-user scenarios.

(over 99% and 94%, respectively). On the other hand, for time-dependent activities (i.e., Activity-3, Activity-4, and Activity-5), IoTDots achieved accuracy and F-score values of over 95% and 91%, respectively. Since the time-dependent actions are related to the user motion, different users may have different motion patterns which increase the False Positive rate (FPR) and False Negative rate (FNR). In summary, IoTDots obtained accuracy values of over 95% on average for detecting different forensically-relevant user activities.

For the case of a multi-user smart environment, since users can perform different tasks at once, this may directly impact the accuracy of the analysis. Figure 7.3 shows the accuracy of IoTDots in inferring user activities in multi-user scenarios.

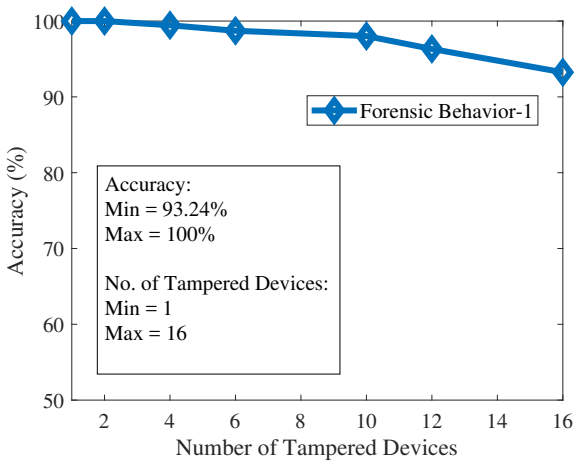


Figure 7.4: Accuracy of IoTDots in detecting forensic behavior from users versus the number of tampered devices.

As explained, one can observe how the accuracy values decreased with the number of users. For time-independent activities (i.e., Activity-1 and Activity-2), IoTDots achieved accuracy in the range of 98% to 95%. For time-dependent activities (i.e., Activity-3, Activity-4, and Activity-5), the accuracy of IoTDots varied from 96% to 86% as the number of users increased.

7.5.4 Detection of Forensic Behavior from Users

Devices installed in a smart environment are vulnerable to tampering which can lead to malicious events. In previous works that proposed the use of IoT data for forensic investigations, trusted devices were always assumed as part of the analysis, resulting in threat models that were vulnerable to insiders [WHBG18].

Figure 7.4 depicts the accuracy of IoTDots in detecting tampered or compromised devices in a smart environment. For this, we installed 22 different devices (including smart sensors) in a smart office environment. One can notice from Figure 7.4 that IoTDots can achieve near 100% of accuracy in cases with 2 tampered devices in the environment. In general, the accuracy of IoTDots decreased as the num-

ber of tampered devices increased in the system. Finally, Figure 7.4 demonstrates how device cooperation (Section 7.3.3) is affected by the number of compromised devices. One can observe that the accuracy values significantly dropped when the experiments considered more than 10 compromised devices (near 50% of the total number of devices).

Behavioral Model	TPR	FNR	TNR	FPR	ACC	F-Score
Behavior-2	0.9612	0.0388	0.8652	0.1348	0.9533	0.9106
Behavior-3	0.9651	0.0349	0.9289	0.0711	0.9621	0.9466
Behavior-4	0.9730	0.0270	0.9317	0.0683	0.9696	0.9518
Behavior-5	0.9687	0.0313	0.9012	0.0988	0.9631	0.9336

Table 7.7: Performance evaluation of IoTDots in detecting forensic behaviors from smart apps.

7.5.5 Detection of Forensic Behavior from Apps

IoT programming platforms offer customized apps to control smart devices. In recent years, researchers have reported several malicious apps that can perform malicious activities in a smart environment [FJP16, And, IoT, Con17]. In this section, we test the efficacy of IoTDots in detecting behaviors in a smart environment caused by malicious apps installed in the system. As noted earlier, we considered four different scenarios to evaluate app behavior in IoTDots (Section 7.2). To evaluate these scenarios, we installed malicious IoTDots-modified apps in a real-life smart environment (smart office). Table 7.7 shows the evaluation results of IoTDots in detecting app's behavior in the smart environment. One can observe that IoTDots achieved high accuracy and F-score for all the cases above (over 95% and 91%, respectively).

In Figure 7.5, the accuracy of IoTDots is shown for different forensically-relevant behaviors when the number of devices controlled by malicious apps increases in the

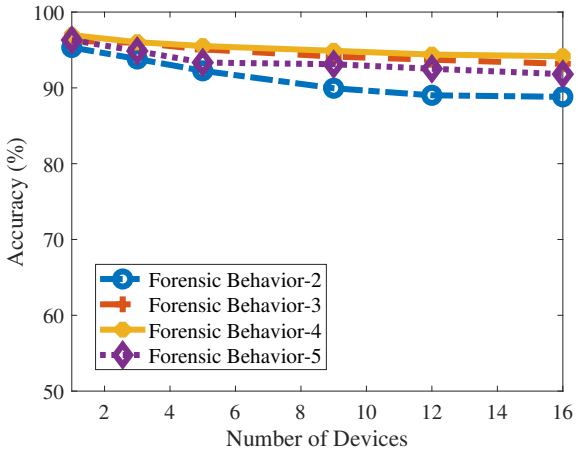


Figure 7.5: Accuracy of IoTDots in detecting forensic behavior from smart apps versus different number of devices.

system. IoTDots achieved the highest accuracy value of 97% for Behavior-4 and the lowest accuracy value of 95% for Behavior-2, for the case of only one device installed in the system. As the number of devices increased, the accuracy decreased to 95% and 89% for Behavior-4 and Behavior-2, respectively. The accuracy of detecting Behavior-3 and Behavior-5 varied between 96% to 94% and 96% to 92% with the number of devices in the system, respectively.

7.5.6 System Overhead

Since most of the smart apps are cloud-based, we do not expect that IoTDots imposes any overhead on the smart devices. However, it is necessary to evaluate the impact of IoTDots on the computing resources of the IoT smart apps servers in the cloud. To assess this, we selected two metrics which relate to (1) the amount of physical storage occupied by the IoTDots-modified apps and (2) the latency of the smart apps in their operations and responses, if compared to the original apps.

Physical Storage

Since server space is costly, we must evaluate how much more physical space the IoTDots-modified apps require from the cloud servers if compared with the unmodified smart apps. On average, the ITM added around 110 new lines of code to the original smart apps source code, which represents an increase of around 5KB of physical memory space per app. Additionally, we evaluated the physical memory space that the logs acquired from the smart environment during evaluation occupied in the ITD. We acquired data from 3 different types of experiments: regular user activities, anomalous activities, and forensic behavior from users and apps. In total, the datasets from 7 days of activities occupied a total size of 20.55 MB.

Latency

For this chapter, we define *latency* as the additional delay imposed by the IoTDots-modified apps source code to the original execution time of the smart apps. In general, the latency depends on (1) the number of times that the IoTDots sends logs to the cloud and (2) the IoTDots logging function execution time. Regarding latency, the worst case scenario occurred when an individual synchronous `HTTPS` request was called every single time a IoTDots log was sent to the ITD. In that case, around 2.4s of extra latency was added to the app execution every time the IoTDots communicates with the ITD, which represented an important overhead to the app operation. To reduce this latency, IoTDots adopted asynchronous `HTTPS` request to send data to ITD. Since we evaluated apps from the Samsung Smartthings platform, we utilized the beta version of the *asynchttp_v*1 class [Smah] to asynchronously send the logs to the database. Further, we reduced the latency overhead even more by avoiding single `HTTPS` calls for every single log. For this, we utilized the `AtomicState` variable in SmartThings to queue and map several logs together before communicating with

the ITD. Finally, with all these modifications, the average latency overhead imposed by IoTDots decreased to around 30ms for every `HTTPS` call sent to the ITD (Figure 7.6).

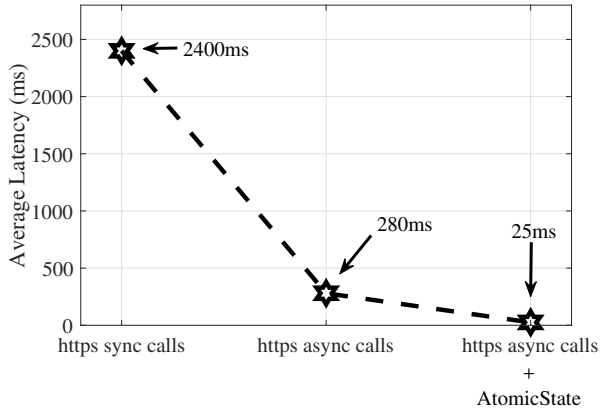


Figure 7.6: Average latency imposed by IoTDots to smart apps' execution times. The minimum latency (25ms) is obtained after combining the asynchronous `HTTPS` request with AtomicState queuing.

7.6 Summary and Benefits

There are several benefits associated with the use of IoTDots. Also, we highlight some challenges and limitations.

- **Forensic Framework for Smart Settings:** Smart settings include a myriad of different smart devices and sensors. They represent a new domain of relevant data related to the users', devices', and apps' activities that has high forensic value. In this context, IoTDots emerges as a practical solution that collects, stores, and processes smart environment data to instrument forensic analysis.
- **Automatic App Analysis and Instrumentation:** The ITM automatically analyzes smart apps source code to identify and log forensically-relevant data. Then, it sends the logs to a database for future analysis. The current version

of IoTDots performs automatic analysis and instrumentation for Samsung SmartThings apps. A new version that supports other IoT programming platforms is currently under development. Despite most IoT apps follow similar architectural paradigms, differences in the programming languages and other platform-specific features make the expansion of IoTDots a challenge (see Applicability below).

- **Deep Data Analysis:** IoTDots-Analyzer incorporates data processing and machine learning techniques to label the IoTDots-collected data and detect forensically-valuable anomalous activity and behavior from users, smart apps, and devices. The initial evaluation results showed excellent results on accuracy metrics for different cases of numbers of users and devices. However, the analysis of new evaluation cases may be necessary. For instance, new devices can be added to the analysis and different machine learning algorithms can be considered.

- **Effectiveness:** Our evaluation results demonstrated the effectiveness of IoTDots in detecting forensically-relevant activities and behavior from users and smart apps. In general, our framework achieved over 95% accuracy for all the anomalous activities and forensic behaviors included in the analysis. Additionally, IoTDots demonstrated very high effectiveness in detecting tampered devices by using device cooperation.

- **Applicability:** One of the most important characteristics of IoT is the high integration among different systems. Rather than platform-specific, current smart environments integrate devices from different IoT platforms. To solve this specific challenge, we implement the ITM to work over the IR of the smart apps. With this, IoTDots can translate apps from different programming

platforms to a common representation which benefits the analysis and permits the utilization of IoTDots in multi-platform environments.

- **Scalability:** The approach used in IoTDots can be easily generalized to any IoT platform and smart settings. For the current version, IoTDots takes advantage of open source platforms to enable simplicity. Nowadays, some of the most important IoT platforms in the market are open source and make the source code of their application available online through specific repositories [Opea, Smag]. However, IoTDots can be also adapted to closed-source IoT systems like HomeSeer [Homb] and HomeOS [Homa]. For this, the current version of IoTDots may require additional modifications on the ITM to allow source code analysis of the closed-source apps and the implementation of an effective logging system. Additionally, smart settings from domains other than smart offices (e.g., smart home scenarios) may require different attack vectors in the adversary model.

- **Privacy-aware Operations:** IoTDots has access to potentially sensitive information from the smart environment. We apply well-known cloud-based data protection schemes to protect the IoTDots logs and prevent compromising the privacy of the users and the smart environment's integrity. For instance, we use encryption (i.e., HTTPS calls) to protect the data exchange between the IoTDots-enabled apps and the ITD. We also use encryption to protect the data at rest in the database (considering it may potentially be stored for long periods), restrictive access control configurations, and periodical backup policies. Additionally, IoTDots does not acquire or process information that is directly related or can contain sensitive information from the users (e.g., Protected Personal Information (PPI)) and that can compromise their privacy. Indeed, even though it is possible, IoTDots does not perform any analysis intended to af-

fect the privacy of the users via, for instance, behavioral fingerprinting or user profiling. As a future work, a complete privacy assessment on IoTDots may be necessary to guarantee that user's privacy is completely preserved.

- **Usability:** We utilized the obtained IRB to survey potential users regarding IoTDots's usability. We asked two specific questions related to (1) the willingness of the users to implement security solutions like IoTDots into their smart settings and (2) the willingness of the users to install instrumented apps that enable security solutions like IoTDots. Out of 118 responses, 88.2% of the users recognized that there is need of security solutions like IoTDots while 82.4% answered "Yes" to the use of IoTDots-modified apps to enable such a solution, while 14.3% expressed their willingness to do it if more information is provided.

7.7 Conclusion

Devices and sensors deployed in a smart environment such as smart home or office have access to data with high forensic value. Nonetheless, current smart app programming platforms do not provide any digital forensic capability to keep track of such data. Additionally, current forensic analysis solutions do not use information from smart apps and/or smart devices to perform forensic investigations and do not consider a comprehensive threat model. In this chapter, we introduce IoTDots, a novel framework used to extract forensically-relevant logs from the smart environment and automatically analyze them later for forensic purposes. The framework has two main components: IoTDots-Modifier and IoTDots-Analyzer. The Modifier performs smart apps' source code analysis, detects forensically-relevant data points inside the smart app's source code, and inserts specific logs at compile time. Then, at runtime, the log data is sent to a remote IoTDots server. In the case of a forensic

investigation, the Analyzer applies data processing and machine learning techniques to extract valuable and usable forensic information from the IoTDots logs. We tested IoTDots in a realistic, smart office environment with a total of 22 devices and sensors. Also, we considered 10 different cases of forensically-relevant activities and behaviors, analyzing a total of 96209 forensic data. As per the evaluation results, IoTDots achieves over 99% of accuracy in detecting user activities and over 96% accuracy in detecting the behavior of users and smart apps in the smart environment. Additionally, IoTDots performance yields no overhead to the smart devices and very low overhead to the IoT cloud server resources. Finally, we made the IoTDots-Modifier freely available online at `https://IoTDots-modifier.appspot.com/`.

CHAPTER 8

CONCLUDING REMARKS AND FUTURE WORK

In this dissertation, we introduced a bottom-up approach to provide comprehensive on-device security and privacy mechanisms for resource-limited devices and their applications. These mechanisms tackled five main security and privacy challenges from the CPS, IoT, and IIoT domains. First, we presented a host-based fingerprinting solution for device identification that is complementary to other security services like device authentication and access control. Also, we designed a kernel- and user-level detection framework that used behavioral analysis to discover compromised resource-limited devices. Further, we applied dynamic analysis to smart devices applications to uncover security and privacy risks in real-time. Then, we implemented a solution to enable digital forensics analysis on data extracted from interconnected resource-limited devices that form a smart environment. Finally, we offered to researchers from industry and academia a collection of benchmark solutions for the evaluation of the described security mechanisms on different smart domains, including CPS, IoT, and IIoT.

From bottom to top, the lower layer of the described security framework introduced Stop-and-Frisk (S&F), a combination of techniques to identify different types of resource-limited devices via host-based fingerprinting. With this, our layered framework targeted the security issues related to spoofing or unauthorized devices that may be used to gain access to critical CPS, IIoT, or IoT networks. S&F combined hardware performance analysis, system and function call tracing techniques, and statistical analysis to create unique signatures of different devices-types using a challenge-response fashion. Then, the signatures were used to identify types of unknown devices present in the network. We evaluated our fingerprinting framework with a realistic testbed that included CPS and IIoT devices of different classes. S&F achieved ex-

cellent identification rates while imposing minimal overhead to the resource-limited devices.

Further, the second layer of our security framework applied behavioral analysis at OS- and kernel-level to detect if devices in critical CPS and IIoT networks have been compromised. We utilized a use case that embodies smart grid devices to evaluate our detection mechanisms against six types of compromised devices. These compromised devices included systems with different combinations of computational resources and attack payloads (i.e., threats). Specifically, the detection mechanisms here introduced incorporated system and function call tracing techniques and statistical analysis to detect unexpected behaviors that can be deemed malicious. We tested our detection approaches on a testbed that featured devices communicating using the IEC61850 protocol, and performing realistic smart grid operations. Our evaluation results demonstrated that behavioral analysis based on kernel and OS behavior analysis can be effectively used to detect devices performing unauthorized operations in CPS and IIoT networks. Also, such an analysis was performed while imposing minimal overhead to the resource-limited CPS and IIoT devices.

The third analysis included in the multi-layered security framework implemented a tool to performed dynamic analysis on IoT applications. With this analysis, we aimed to uncover apps that represented a security or privacy risk for the users and the critical smart systems. First, we analyzed and modified real IoT applications to offer the users the opportunity of creating their own privacy profile, based on specific privacy references. Also, we inserted additional code into the apps to collect application data at runtime. We then used the data extracted from the apps to perform application analysis in real time. First, we implemented an NLP model to classify the app data into privacy labels that are easy to understand by the user. Second, we verified if the recipients of the sensitive information were authorized or

acknowledged by the users of the IoT apps. Third, we certified that IoT applications did not make the sensitive data available to passive eavesdroppers. Finally, the users were informed about our findings in real time. In total, we analyzed 540 real IoT apps to train the NLP model and evaluate its effectiveness. Our privacy tool classified IoT strings to correct privacy preferences with an average accuracy of 94.25%, and flagged 35 apps that leak data. Finally, the tool imposed minimal overhead to an IoT apps execution, introducing on average 105 ms additional latency.

Finally, we evaluated the security of resource-limited devices and their apps at system level. With this analysis, we provided protection not only for individual devices and apps but for the smart environment as a whole. Specifically, we designed a novel forensic framework that extracted data from smart environments at runtime and also provided mechanisms to infer relevant forensic activities and malicious behaviors occurring in the smart environment. First, we analyzed the IoT applications to detect forensically-relevant points within the apps. Then, we instrumented the apps to insert logs and extract the forensic data at runtime. Later, in the case of a forensics investigation, we used Markov-Chain models to match the data extracted from different sensors, devices, and IoT apps to events that potentially occurred in the smart environment. We tested the introduced forensics framework on real smart offices settings. Our evaluation results demonstrated very high accuracy in detecting user activities and forensic behaviors of users, devices, and IoT apps. These results were obtained while adding no overhead to the smart devices and very low overhead to the IoT apps' execution.

For every security layer included in our bottom-up framework, we designed and implemented benchmark tools and solutions used to test and evaluate the security mechanisms here introduced. These benchmarks offer evaluation mechanisms and metrics for devices and apps at different levels: (1) hardware, (2) operating system-

s/kernel, (3) application, and (4) system. Therefore, the benchmarks here introduced were meant to be highly effective, while imposing minimal overhead to the devices and apps. Also, as devices and apps may have different application domains with specific characteristics (e.g., CPS, IIoT, IoT), the benchmark solutions were designed both scalable and flexible, so they can be easily adapted to target new security domains. Finally, the collection of frameworks and tools discussed in this dissertation were made freely available to other security researchers.

We present several key directions for future research.

- Throughout this dissertation, we performed hardware-, OS- and kernel-based analysis on resource-limited devices that are representative of real CPS and IIoT infrastructures. However, we believe that implementing these security frameworks on real devices would carry new implementations challenges not considered in this dissertation. These challenges are mainly related with the extraction of system and function calls from devices running different types of operating systems, including very-limited embedded OS for which several system or function calls might not be available. To overcome these challenges, the security tools here described must implement an adaptive approach so the appropriate analysis is performed to guarantee high accuracy for every different OS type. In some critical cases were the number of system and function calls available for analysis is too limited, alternative approaches to perform behavioral analysis must be explored.
- We also performed application- and system-level analysis targeting specific IoT programming platforms. However, different programming languages may generate new and different design and implementation challenges. Even though we partially solved this problem by proposing the use of intermediate representa-

tion of code into our analysis, we believe that code analysis and instrumentation for different languages need to be explored.

- Finally, even though we consider far-reaching threat models in every layer, we also believe that new threats can be included and evaluated. For instance, new forensic behaviors can be considered at system level as different smart environments define new specific user activities and possible behaviors.

BIBLIOGRAPHY

[A. 15] A. Kanovsky, P. Spanik and M. Frivaldsky. Detection of Electronic Counterfeit Components. In *2015 16th Int. Scientific Conf. on Electric Power Engineering (EPE)*, pages 701 – 705, Kouty and Desnou, May 2015. IEEE.

[ABC+18] H. Aksu, L. Babun, M. Conti, G. Tolomei, and A. S. Uluagac. Advertising in the IoT Era: Vision and Challenges. *IEEE Communications Magazine*, pages 1–7, 2018.

[And] Android-based Smart TVs Hit By Backdoor Spread Via Malicious App, Ju Zhu. https://blog.trendmicro.com/trendlabs-security-intelligence/android-based-smart-tvs-hit-by-backdoor-spread-via-malicious-app/. [Online; accessed January-2020].

[Appa] Apple's Home Kit. `https://www.apple.com/ios/home/`. [Online; accessed January-2020].

[Appb] Apple's Home Kit Security and Privacy on iOS. `https://www.apple.com/business/docs/iOS_Security_Guide.pdf`. [Online; accessed January-2020].

[Appc] Apple's HomeKit Submission Guideline. `https://developer.apple.com/app-store/review/guidelines`. [Online; accessed January-2020].

[ARF+14] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. *ACM SIGPLAN Notices*, 2014.

[Asb09] Bjørn Egil Asbjørnslett. *Assessing the Vulnerability of Supply Chains*, pages 15–33. Springer US, Boston, MA, 2009.

[Auta] AutoML. `https://www.ml4aad.org/automl/`. [Online; accessed 10-February-2020].

[Autb] AutoML Natural Language Google. `https://cloud.google.com/natural-language/automl/docs/`. google[Online; accessed 10-February-2020].

[AVJA15] J. D. Ansilla, N. Vasudevan, J. JayachandraBensam, and J. D. Anunciya. Data security in smart grid with hardware implementation against dos attacks. In *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*, pages 1–7, March 2015.

[AYSM17] M. A. B. Ahmadon, S. Yamaguchi, S. Saon, and A. K. Mahamad. On service security analysis for event log of iot system based on data petri net. In *2017 IEEE International Symposium on Consumer Electronics (ISCE)*, pages 4–8, Nov 2017.

[Ayy18] V. Kishore Ayyadevara. *Word2vec*, pages 167–178. Apress, Berkeley, CA, 2018.

[AZZ+17] Mussab Alaa, AA Zaidan, BB Zaidan, Mohammed Talal, and Miss Laiha Mat Kiah. A Review of Smart Home Applications based on Internet of Things. *Journal of Network and Computer Applications*, 97:48–65, 2017.

[Bab18] Babun, Leonardo (Miami, FL, US), Aksu, Hidayet (Miami, FL, US), Uluagac, Selcuk A. (Miami, FL, US). Detection of Counterfeit and Compromised Devices using System and Function Call Tracing Techniques. https://www.osti.gov/biblio/1463864, July 2018.

[Bab19] Babun, Leonardo (Miami, FL, US), Aksu, Hidayet (Miami, FL, US), Uluagac, Selcuk A. (Miami, FL, US). Method of resource-limited device and device class identification using system and function call tracing techniques, performance, and statistical analysis. `https://patents.google.com/patent/US10242193B1/en`, March 2019.

[BAU17] L. Babun, H. Aksu, and A. S. Uluagac. Identifying Counterfeit Smart Grid Devices: A Lightweight System Level Framework. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2017.

[BAU19] Leonardo Babun, Hidayet Aksu, and A. Selcuk Uluagac. A System-Level Behavioral Detection Framework for Compromised CPS Devices: Smart-Grid Case. *ACM Trans. Cyber-Phys. Syst.*, 4(2), November 2019.

[BCKP08] Sergey Bratus, Cory Cornelius, David Kotz, and Daniel Peebles. Active Behavioral Fingerprinting of Wireless Devices. In *Proceedings of the First ACM Conference on Wireless Network Security*, WiSec '08, pages 56–61, New York, NY, USA, 2008. ACM.

[BCMU19] Leonardo Babun, Z. Berkay Celik, Patrick McDaniel, and A. Selcuk Uluagac. Real-time Analysis of Privacy-(un)aware IoT Applications. `https://arxiv.org/abs/1911.10461`, 2019.

[BGK16] N. Boumkheld, M. Ghogho, and M. El Koutbi. Intrusion Detection System for the Detection of Blackhole Attacks in a Smart Grid. In *2016 4th International Symposium on Computational and Business Intelligence (ISCBI)*, pages 108–111, Sept 2016.

[BGV18] François Bouchaud, Gilles Grimaud, and Thomas Vantroys. IoT Forensic: Identification and Classification of Evidence in Criminal Investigations. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, page 60. ACM, 2018.

[Blu13] Reverend Bill Blunden. *The Rookit arsenal: Escape and Evasion in the Dark Corners of the System.* Cathleen Sether, Burlington, MA, 2nd edition, 2013.

[BSAU18] Leonardo Babun, Amit Kumar Sikder, Abbas Acar, and A. Selcuk Uluagac. IoTDots: A Digital Forensics Framework for Smart Environments. `https://arxiv.org/abs/1809.00745`, 2018.

[C. 13] C. Kriger, S. Behardien and J. Retonda-Modiya. A Detailed Analysis of the GOOSE Message Structure in an IEC 61850 Standard-Based Substation Automation System. *Int. Journal Comp. Comm.*, 8(5):708–721, Oct. 2013.

[C+07] James Clause et al. Dytan: a Generic Dynamic Taint Analysis Framework. In *ACM Software Testing and Analysis*, 2007.

[CBS+18] Z. Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A. Selcuk Uluagac. Sensitive Information Tracking in Commodity IoT. In *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, 2018. USENIX Association.

[CDS16] Sudhir Chitnis, Neha Deshpande, and Arvind Shaligram. An Investigative Study for Smart Home Security: Issues, Challenges and Countermeasures. *Wireless Sensor Network*, page 61, 2016.

[CFP+19] Z Berkay Celik, Earlence Fernandes, Eric Pauley, Gang Tan, and Patrick McDaniel. Program Analysis of Commodity IoT Applications for Security

and Privacy: Challenges and Opportunities. *ACM Computing Surveys (CSUR)*, 2019.

[CGJ15] Henry Corrigan-Gibbs and Suman Jana. Recommendations for Randomness in the Operating System or, How to Keep Evil Children out of Your Pool and Other Random Facts. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*, HOTOS'15, pages 25–25, Berkeley, CA, USA, 2015. USENIX Association.

[Ch.15] Ch. Wong and M. Wu. A Study on PUF Characteristics for Counterfeit Detection. In *2015 IEEE Int. Conf. on Image Processing (ICIP)*, pages 1643 – 1647, Quebec City, QC, Sept. 2015. IEEE.

[CMT18] Z. Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated IoT Safety and Security Analysis. In *USENIX Annual Technical Conference (USENIX ATC)*, 2018.

[CMT+19] Z. B. Celik, P. McDaniel, G. Tan, L. Babun, and A. S. Uluagac. Verifying Internet of Things Safety and Security in Physical Spaces. *IEEE Security Privacy*, 17(5):30–37, Sep. 2019.

[Con] Control4. https://www.control4.com/blog/440/can-smart-home-technology-reduce-home-insurance-rates. [Online; accessed August-2018].

[Con17] ContexIoT attacks for SmartThings programs using existing adversary techniques. `https://sites.google.com/site/iotcontextualintegrity/home`, 2017. [Online; accessed 09-January-2020].

[CPL17] Hyunji Chung, Jungheum Park, and Sangjin Lee. Digital Forensic Approaches for Amazon Alexa Ecosystem. *Digital Investigation*, 22:15–25, 2017.

[CPLK12] Hyunji Chung, Jungheum Park, Sangjin Lee, and Cheulhoon Kang. Digital Forensic Investigation of Cloud Storage Services. *Digital investigation*, 9(2):81–95, 2012.

[CPS18a] Y. Chen, C. M. Poskitt, and J. Sun. Learning from Mutants: Using Code Mutation to Learn and Monitor Invariants of a Cyber-Physical System. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 648–660, May 2018.

[CPS18b] Y. Chen, C. M. Poskitt, and J. Sun. Learning from Mutants: Using Code Mutation to Learn and Monitor Invariants of a Cyber-Physical System. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 648–660, May 2018.

[CTM19] Z. Berkay Celik, Gang Tan, and Patrick McDaniel. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. In *Network and Distributed System Security Symposium (NDSS)*, 2019.

[D. 12] D. van Opstal, U.S. Resilience Project. Supply Chain Solutions for Smart Grid Security: Building on Business Best Practices. `http://usresilienceproject.org/wp-content/uploads/2014/09/report-Supply_Chain_Solutions_for_Smart_Grid_Security.pdf`, Sep 2012. [ONLINE-Accessed: January-2020].

[DBC14] Anupam Das, Nikita Borisov, and Matthew Caesar. Do You Hear What I Hear?: Fingerprinting Smart Devices Through Embedded Acoustic Components. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 441–452, New York, NY, USA, 2014. ACM.

[DBCM16] Anupam Das, Nikita Borisov, Edward Chou, and Muhammad Haris Mughees. Smartphone Fingerprinting Via Motion Sensors: Analyzing Feasiblity at Large-Scale and Studing Real Usage Patterns. *CoRR*, abs/1605.08763, 2016.

[DBU20] Kyle Denney, Leonardo Babun, and A. Selcuk Uluagac. Usb-watch: a generalized hardware-assisted insider threat detection framework. *J Hardw Syst Secur*, 2020.

[DEB+19] Kyle Denney, Enes Erdin, Leonardo Babun, Michael Vai, and Selcuk Uluagac. Usb-watch: A dynamic hardware-assisted usb threat detection framework. In Songqing Chen, Kim-Kwang Raymond Choo, Xinwen Fu, Wenjing Lou, and Aziz Mohaisen, editors, *Security and Privacy in Communication Networks*, pages 126–146, Cham, 2019. Springer International Publishing.

[DJ17] Asish Kumar Dalai and Sanjay Kumar Jena. WDTF: A Technique for Wireless Device Type Fingerprinting. *Wireless Personal Communications*, 97(2):1911–1928, Nov 2017.

[DS12] Y. Deng and S. Shukla. Vulnerabilities and Countermeasures - A Survey on the Cyber Security Issues in the Transmission Subsystem of a Smart Grid. *Journal of Cyber Security and Mobility*, 1:251–276, 2012.

[E. 01] E. Eskin, W. Lee and S. J, Stolfo. Modeling System Calls for Intrusion Detection with Dynamic Window Sizes. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01*, pages 165–171, Anaheim, CA, Jun. 2001. IEEE.

[EGH+14] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *ACM Transaction on Computer Systems*, 2014.

[Eur12] European Network and Information Security Agency (enisa). Smart Grid Security. Annex II: Security Aspects of the Smart Grid. `https://www.enisa.europa.eu/topics/critical-information-infrastructures-and-services/smart-grids/smart-grids-and-smart-metering/ENISA_Annex%20II%20-%20Security%20Aspects%20of%20Smart%20Grid.pdf`, 2012. [Online; accessed February-2020].

[FHDK16] A. Farraj, E. Hammad, A. A. Daoud, and D. Kundur. A Game-Theoretic Analysis of Cyber Switching Attacks and Mitigation in Smart Grid Systems. *IEEE Transactions on Smart Grid*, 7(4):1846–1855, July 2016.

[FJP16] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security Analysis of Emerging Smart Home Applications. In *IEEE Security and Privacy (SP)*, 2016.

[FKF+03] Henry Hanping Feng, Oleg M. Kolesnikov, Prahlad Fogla, Wenke Lee, and Weibo Gong. Anomaly Detection Using Call Stack Information. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, SP '03, pages 62–, Washington, DC, USA, 2003. IEEE Computer Society.

[FPR+16] 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.

[FSL+16] David Formby, Preethi Srinivasan, Andrew Leonard, Jonathan Rogers, and Raheem A. Beyah. Who's in Control of Your Control System? Device

Fingerprinting for Cyber-Physical Systems. In *23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016.

[Gar03] Tal Garfinkel. Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools. In *In Proc. Network and Distributed Systems Security Symposium*, pages 163–176, 2003.

[GE11] P. J. Guo and D. Engler. CDE: Using System Call Interposition to Automatically Create Portable Software Packages. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, pages 21–21, Berkeley, CA, USA, 2011. USENIX Association.

[GFT13] Ujjwal Guin, Domenic Forte, and Mohammad Tehranipoor. Anti-counterfeit Techniques: From Design to Resign. In *Proceedings of the 2013 14th International Workshop on Microprocessor Test and Verification*, pages 89–94, Washington, DC, USA, 2013. IEEE Computer Society.

[GKGK16] I. Graja, S. Kallel, N. Guermouche, and A. H. Kacem. BPMN4CPS: A BPMN Extension for Modeling Cyber-Physical Systems. In *2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 152–157, June 2016.

[GKP+15] Michael I Gordon, Deokhwan Kim, Jeff H Perkins, Limei Gilham, Nguyen Nguyen, and Martin C Rinard. Information Flow Analysis of Android Applications in DroidSafe. In *NDSS*, 2015.

[GLL+13] B. Gu, X. Li, G. Li, A. C. Champion, Z. Chen, F. Qin, and D. Xuan. D2Taint: Differentiated and Dynamic Information Flow Tracking on Smartphones for Numerous Data Sources. In *INFOCOM*, 2013.

[Gooa] Google App Engine. `https://cloud.google.com/appengine/`. [Online; accessed January-2020].

[Goob] Google Books NGrams. `https://aws.amazon.com/datasets/google-books-ngrams/`. [Online; accessed January-2020].

[Gooc] Google Forms. `https://www.google.com/forms/about/`. [Online; accessed January-2020].

[Gro] GroovyCodeVisitor: An Implementation of the Groovy Visitor Patterns. `http://docs.groovy-lang.org/docs`. [Online; accessed January-2018].

[GTGZ14] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. Checking App Behavior Against App Descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 1025–1035, New York, NY, USA, 2014. ACM.

[HB99] G. Hunt and D. Brubacher. Detours: Binary Interception of Win32 Functions. In *Proceedings of the 3rd Conference on USENIX Windows NT Symposium - Volume 3*, WINSYM'99, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

[HCG17a] D. He, S. Chan, and M. Guizani. Cyber Security Analysis and Protection of Wireless Sensor Networks for Smart Grid Monitoring. *IEEE Wireless Communications*, 24(6):98–103, Dec 2017.

[HCG17b] D. He, S. Chan, and M. Guizani. Win-Win Security Approaches for Smart Grid Communications Networks. *IEEE Network*, 31(6):122–128, November 2017.

[HLM+16] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. Smart Locks: Lessons for Securing Commodity Internet of Things Devices. In *ACM AsiaCCS*, 2016.

[Homa] HomeOS, Microsoft. `https://www.microsoft.com/en-us/research/project/homeos-enabling-smarter-homes-for-everyone/`. [Online; accessed January-2020].

[Homb] HomeSeer. `https://homeseer.com/`. [Online; accessed January-2020].

[Hon14] Honeywell. RTU2020 Remote Terminal Unit Specifications. `https://www.honeywellprocess.com/library/marketing/tech-specs/SC03-300-101-RTU-2020.pdf`, Oct 2014. [ONLINE; accessed January-2020].

[HPK+14] J. Hao, R. J. Piechocki, D. Kaleshi, W. H. Chin, and Z. Fan. Optimal Malicious Attack Construction and Robust Detection in Smart Grid Cyber Security Analysis. In *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 836–841, Nov 2014.

[HSS17] Aaron Hansen, Jason Staggs, and Sujeet Shenoi. Security Analysis of an Advanced Metering Infrastructure. *International Journal of Critical Infrastructure Protection*, 18:3 – 19, 2017.

[HV17] Malek Harbawi and Asaf Varol. An Improved Digital Evidence Acquisition Model for the Internet of Things forensics: A Theoretical Framework. In *Digital Forensic and Security (ISDFS), 2017 5th International Symposium on*, pages 1–6. IEEE, 2017.

[IEC03a] IEC 61850-1. Communication networks and systems in substations Introduction and overview. `https://webstore.iec.ch/p-preview/info_iec61850-1%7Bed1.0%7Den.pdf`, 2003. [Online; accessed January-2020].

[IEC03b] IEC 61850-7-2. Communication networks and systems in substations - Basic communication structure for substation and feeder equipment Abstract Communication Service Interface (ACSI). `https://webstore.iec.ch/p-preview/info_iec61850-7-2%7Bed1.0%7Den.pdf`, 2003. [Online; accessed January-2020].

[IEC03c] IEC 61850-8-1. Communication networks and systems in substations - Specific Communication Service Mapping (SCSM) Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3. `https://webstore.iec.ch/p-preview/info_iec61850-8-1%7Bed1.0%7Den.pdf`, 2003. [Online; accessed January-2020].

[IEC03d] IEC61850-7-1. Communication networks and systems for power utility automation - Part 7-1: Basic communication structure - Principles and models. `https://webstore.iec.ch/publication/6014`, 2003. [Online; accessed January-2020].

[IFT] IFTTT (if this, then that). `https://ifttt.com/`. [Online; accessed August-2018].

[Int18] Interos Solutions, Inc. Supply chain vulnerabilities from china in u.s. federal information and communications technology. `https://www.uscc.gov/sites/default/files/Research/Interos_Supply%20Chain%20Vulnerabilities%20from%20China%20in%20U.S.%20Federal%20ICT_final.pdf`, 2018.

[IoT] IoTBench Repository, L. Babun, Z. Berkay Celik and A. Kumar Sikder. `https://github.com/IoTBench`. [Online; accessed January-2020].

[IoT17] IoTBench. https://github.com/IoTBench, 2017. [Online; accessed 09-October-2018].

[Iri] Iris by Lowe's, Lowe's. https://www.irisbylowes.com/. [Online; accessed January-2020].

[ISTC13] D. M. E. Ingram, P. Schaub, R. R. Taylor, and D. A. Campbell. Performance Analysis of IEC 61850 Sampled Value Process Bus Networks. *IEEE Transactions on Industrial Informatics*, 9(3):1445–1454, Aug 2013.

[J. 16] J. Ellperin and A. Entous. Russian Operation Hacked a Vermont Utility, Showing Risk to U.S. Electrical Grid Security, Officials Say. https://www.washingtonpost.com/world/national-security/russian-hackers-penetrated-us-electricity-grid-through-a-utility-in-vermont/2016/12/30/8fc90cc4-ceec-11e6-b8a2-8c2a61b0436f_story.html, 2016. [Online; accessed January-2020].

[JCW+17] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z Morley Mao, Atul Prakash, and Shanghai JiaoTong Unviersity. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In *NDSS*, 2017.

[JRSK17] P. Jafary, S. Repo, J. Seppl, and H. Koivisto. Security and Reliability Analysis of a Use Case in Smart Grid Substation Automation Systems. In *2017 IEEE International Conference on Industrial Technology (ICIT)*, pages 615–620, March 2017.

[JS99] K. Jain and R. Sekar. User-Level Infrastructure for System Call Interposition: A Platform for Intrusion Detection and Confinement. In *In Proc. Network and Distributed Systems Security Symposium*, 1999.

[K. 13] K. Huang, J. M. Carulli, and Y. Makris. Counterfeit Electronics: A Rising Threat in the Semiconductor Manufacturing Industry. In *ITC. IEEE Computer Society*, pages 1–4. IEEE, 2013.

[KAJM16] E. Kang, S. Adepu, D. Jackson, and A. P. Mathur. Model-Based Security Analysis of a Water Treatment System. In *2016 IEEE/ACM 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*, pages 22–28, May 2016.

[Kas16] Kaspersky. BlackEnergy APT Attacks in Ukraine. https://usa.kaspersky.com/resource-center/threats/blackenergy, 2016. [Online; accessed January-2020].

[KBAU18] C. Kaygusuz, L. Babun, H. Aksu, and A. S. Uluagac. Detection of Compromised Smart Grid Devices with Machine Learning and Convolution Techniques. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2018.

[KBC05] T. Kohno, A. Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, April 2005.

[KBKK12] E. C. Kara, M. Berges, B. Krogh, and S. Kar. Using Smart Devices for System-level Management and Control in the Smart Grid: A Reinforcement Learning Framework. In *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*, pages 85–90, Nov 2012.

[KCB10] Ke Gao, C. Corbett, and R. Beyah. A Passive Approach to Wireless Device Fingerprinting. In *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, pages 383–392, June 2010.

[KHLF10] H. Khurana, M. Hadley, N. Lu, and D. A. Frincke. Smart-grid Security Issues. *IEEE Security Privacy*, 8(1):81–85, Jan 2010.

[Kos16] A. M. Kosek. Contextual Anomaly Detection for Cyber-physical Security in Smart Grids based on an Artificial Neural Network Model. In *2016 Joint Workshop on Cyber- Physical Security and Resilience in Smart Grids (CPSR-SG)*, pages 1–6, April 2016.

[Kou12] G. Koutitas. Control of Flexible Smart Devices in the Smart Grid. *IEEE Transactions on Smart Grid*, 3(3):1333–1343, Sept 2012.

[KPJ16] K. Khanna, B. K. Panigrahi, and A. Joshi. Feasibility and Mitigation of False Data Injection Attacks in Smart Grid. In *2016 IEEE 6th International Conference on Power Systems (ICPS)*, pages 1–6, March 2016.

[KR16] Victor R Kebande and Indrakshi Ray. A Generic Digital Forensic Investigation Framework for Internet of Things (IoT). In *Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th International Conference on*, pages 356–362. IEEE, 2016.

[KSH16] M. Khan, B. N. Silva, and K. Han. Internet of Things Based Energy Aware Smart Home Control System. *IEEE Access*, 4:7556–7566, 2016.

[Kus13] D. Kushner. The Real Story of Stuxnet. *IEEE Spectrum*, 50(3):48–53, March 2013.

[KZ13] T. Kim and N. Zeldovich. Practical and Effective Sandboxing for Non-root Users. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, USENIX ATC'13, pages 139–144, Berkeley, CA, USA, 2013. USENIX Association.

[L. 17] L. Babun, H. Aksu and A. S. Uluagac. Identifying Counterfeit Smart Grid Devices: A Lightweight System Level Framework. In *Proceedings of the IEEE ICC Intern. Conf. on Communications*, Paris, France, May 2017. IEEE.

[LBAU17] Juan Lopez, Leonardo Babun, Hidayet Aksu, and A. Selcuk Uluagac. A Survey on Function and System Call Hooking Approaches. *Journal of Hardware and Systems Security*, 1(2):114–136, Jun 2017.

[LFPF18] L. D. Lago, O. Ferrante, R. Passerone, and A. Ferrari. Dependability Assessment of SOA-Based CPS With Contracts and Model-Based Fault Injection. *IEEE Transactions on Industrial Informatics*, 14(1):360–369, Jan 2018.

[LGGG07] M. LeMay, G. Gross, C. A. Gunter, and S. Garg. Unified Architecture for Large-Scale Attested Metering. In *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, pages 115–115, Jan 2007.

[LL15] M. Li and H. J. Lin. Design and Implementation of Smart Home Control Systems Based on Wireless Sensor Networks and Power Line Communications. *IEEE Transactions on Industrial Electronics*, 62(7):4430–4442, July 2015.

[LM14] Quoc Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, 2014.

[Log] Logs - MiCasaVerde. `http://wiki.micasaverde.com/index.php/Logs`. [Online; accessed January-2020].

[LPBZ12] F. Lanze, A. Panchenko, B. Braatz, and A. Zinnen. Clock Skew Based Remote Device Fingerprinting Demystified. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 813–819, Dec 2012.

[M. 13] M. Q. Saeed, Z. Bilal and C. D. Walter. An NFC Based Consumer-level Counterfeit Detection Framework. In *2013 Eleventh Annual Int. Conf. on Privacy, Security and Trust (PST)*, pages 135–142, Tarragona, July 2013. IEEE.

[M. 16] M. Sillgith. Open Source Library for IEC 61850: Release 0.9. `http://libiec61850.com/libiec61850/`, Feb 2016. [Online; accessed January-2020].

[Mal17] Malware Found in Surveillance Cameras Sold Through Amazon. `https://www.techworm.net/2016/04/malware-found-surveillance-cameras-sold-amazon.html`, 2017. [Online; accessed January-2020].

[MCCD13] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3781, 2013.

[Met] Metaprogramming. `http://docs.groovy-lang.org/docs/next/html/documentation/core-metaprogramming.html`. [Online; accessed January-2020].

[MIT] MIT Technology Review: Why Insurance Companies Want to Subsidize Your Smart Home. `https://www.technologyreview.com/s/602532/why-insurance-companies-want-to-subsidize-your-smart-home/`. [Online; accessed January-2020].

[MKLP12] Narcisa Andreea Milea, Siau Cheng Khoo, David Lo, and Cristian Pop. NORT: Runtime Anomaly-based Monitoring of Malicious Behavior for Windows. In *Proceedings of the Second International Conference on Runtime Verification*, RV'11, pages 115–130, Berlin, Heidelberg, 2012. Springer-Verlag.

[MMH+17] M. Miettinen, S. Marchal, I. Hafeez, T. Frassetto, N. Asokan, A. Sadeghi, and S. Tarkoma. IoT Sentinel Demo: Automated Device-Type Identification for Security Enforcement in IoT. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2511–2514, June 2017.

[MR16] C. Murguia and J. Ruths. CUSUM and Chi-squared Attack Detection of Compromised Sensors. In *2016 IEEE Conference on Control Applications (CCA)*, pages 474–480, Sep. 2016.

[N.] N. Iderhoff, "NLP-Datasets". `https://github.com/niderhoff/nlp-datasets/blob/master/README.md`. [Online; accessed January-2020].

[N. 14] N. Komninos, E. Philippou and A. Pitsillides. Survey in Smart Grid and Smart Home Security: Issues, Challenges and Countermeasures. *IEEE Communications Surveys and Tutorials*, 16:1933–1954, 2014.

[Nat18] National Cybersecurity & Communications Integration Center (NCCIC), Department of Homeland Security. Russian Activity Against Critical Infrastructure. `https://www.us-cert.gov/sites/default/files/c3vp/Russian_Activity_Webinar_Slides.pdf`, 2018. [Online; accessed January-2020].

[NHO12] C. Neumann, O. Heen, and S. Onno. An Empirical Study of Passive 802.11 Device Fingerprinting. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, pages 593–602, June 2012.

[NIS14] NIST Special Publication 1108r3. NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 3.0. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1108r3.pdf, Sep 2014. [Online; accessed January-2020].

[NM15] A. Nourian and S. Madnick. A Systems Theoretic Approach to the Security Threats in Cyber Physical Systems Applied to Stuxnet. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1–1, 2015.

[NMM+18] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Minh Hoang Dang, N. Asokan, and Ahmad-Reza Sadeghi. DÏoT: A Crowdsourced Self-learning Approach for Detecting Compromised IoT Devices. *CoRR*, abs/1804.07474, 2018.

[NSG+14] Sukhvir Notra, Muhammad Siddiqi, Hassan Habibi Gharakheili, Vijay Sivaraman, and Roksana Boreli. An Experimental Study of Security and Privacy Risks with Emerging Household Appliances. In *Communications and Network Security (CNS), 2014 IEEE Conference on*, pages 79–84. IEEE, 2014.

[OEV+16] M. Ozay, I. Esnaola, F. T. Yarman Vural, S. R. Kulkarni, and H. V. Poor. Machine Learning Methods for Attack Detection in the Smart Grid. *IEEE Transactions on Neural Networks and Learning Systems*, 27(8):1773–1786, Aug 2016.

[OJES13] Edewede Oriwoh, David Jazani, Gregory Epiphaniou, and Paul Sant. Internet of Things Forensics: Challenges and Approaches. In *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*, pages 608–615. IEEE, 2013.

[OKGP13] Temitope Oluwafemi, Tadayoshi Kohno, Sidhant Gupta, and Shwetak Patel. Experimental Security Analyses of Non-Networked Compact Fluorescent Lamps: A Case Study of Home Automation Security. In *USENIX LASER*, 2013.

[Opea] OpenHAB IoT App Market (Eclipse Market Place). `http://docs.openhab.org/eclipseiotmarket`. [Online; accessed January-2020].

[Opeb] OpenHAB IoT App Submission Guideline. `https://marketplace.eclipse.org/content/eclipse-marketplace-publishing-guidelines`. [Online; accessed January-2020].

[Opec] OpenHAB: Open Source Automation Software for Home. `https://www.openhab.org/`. [Online; accessed January-2020].

[Oped] OpenHAB Privacy Statement. `http://www.myopenhab.org/privacy`. [Online; accessed January-2020].

[PBL15] T. Park, S. Beack, and T. Lee. A Noise Robust Audio Fingerprint Extraction Technique for Mobile Devices Using Gradient Histograms. In *Consumer Electronics - Berlin (ICCE-Berlin), 2015 IEEE 5th International Conference on*, pages 287–290, Sept 2015.

[PCD+18] Xiang Pan, Yinzhi Cao, Xuechao Du, Boyuan He, Gan Fang, Rui Shao, and Yan Chen. FlowCog: Context-aware Semantics Extraction and Analysis of Information Flow Leaks in Android Apps. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1669–1685, Baltimore, MD, 2018. USENIX Association.

[PNR15] Sundresan Perumal, Norita Md Norwawi, and Valliappan Raman. Internet of Things (IoT) Digital Forensic Investigation Model: Top-down Forensic Approach Methodology. In *Digital Information Processing and Communications (ICDIPC), 2015 Fifth International Conference on*, pages 19–23. IEEE, 2015.

[PX17] M. Pendleton and S. Xu. A Dataset Generator for Next Generation System Call Host Intrusion Detection Systems. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pages 231–236, Oct 2017.

[PXY+13] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 527–542, Washington, D.C., 2013. USENIX.

[QRZ+14] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. AutoCog: Measuring the Description-to-permission Fidelity in Android Applications. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1354–1365, New York, NY, USA, 2014. ACM.

[R.] R. E Faith and M. Kerrisk. The Linux Man-pages Project: ptrace. `http://man7.org/linux/man-pages/man2/ptrace.2.html`. [Online; accessed January-2020].

[RB15] D. B. Rawat and Ch. Bajracharya. Cyber Security for Smart Grid Systems: Status, Challenges and Perspectives. In *Proceedings of the IEEE Southeast Conf*, pages 1–6, Fort Lauderdale, FL, USA, 2015. IEEE.

[RBAU19] Luis Puche Rondon, Leonardo Babun, Kemal Akkaya, and A. Selcuk Uluagac. Hdmi-walk: Attacking hdmi distribution networks via consumer electronic control protocol. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC 19, pages 650–659, New York, NY, USA, 2019. Association for Computing Machinery.

[Reu16] Reuters. U.S. firm blames Russian 'Sandworm' hackers for Ukraine outage. https://www.reuters.com/article/us-ukraine-cybersecurity-sandworm/u-s-firm-blames-russian-sandworm-hackers-for-ukraine-outage-idUSKBN0UM00N20160108, 2016. [Online; accessed January-2020].

[RFP16] Amir Rahmati, Earlence Fernandes, and Atul Prakash. Applying the Opacified Computation Model to Enforce Information Flow Policies in IoT Applications. In *IEEE Cybersecurity Development (SecDev)*, 2016.

[Ros01] Sheldon M. Ross. *Probability Models for Computer Science*. Academic Press, Inc., Orlando, FL, USA, 1st edition, 2001.

[RUB15] S. V. Radhakrishnan, A. S. Uluagac, and R. Beyah. GTID: A Technique for Physical Device and Device Type Fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 12(5):519–532, Sept 2015.

[S. 10] S. Fries, H. J. Hof and M. G. Seewald. Security of the Smart Grid - Enhancing IEC 62351 to Improve Security in Energy Automation Control. *Int. Journal on Advances in Security*, 3, 2010.

[SaI] SaINT Project, L. Babun, Z. Berkay Celik and A. Kumar Sikder. `http://saint-project.appspot.com/`. [Online; accessed January-2020].

[Sam] Samsung SmartThings. `https://www.smartthings.com/`. [Online; accessed January-2020].

[SAU17] Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac. 6thSense: A Context-aware Sensor-based Attack Detector for Smart Devices. In *USENIX Security*, 2017.

[SBAU19] Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A. Selcuk Uluagac. Aegis: A Context-Aware Security Framework for Smart Home Systems. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC 19, page 2841, New York, NY, USA, 2019. Association for Computing Machinery.

[SBDB01] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, SP '01, pages 144–, Washington, DC, USA, 2001. IEEE Computer Society.

[SC11] B. Sikdar and J. H. Chow. Defending Synchrophasor Data Networks Against Traffic Analysis Attacks. *IEEE Transactions on Smart Grid*, 2:819–826, 2011.

[SGLL13] Y. Sun, X. Guan, T. Liu, and Y. Liu. A Cyber-physical Monitoring System for Attack Detection in Smart Grid. In *2013 IEEE Conference*

on Computer Communications Workshops (INFOCOM WKSHPS), pages 33–34, April 2013.

[SHS12] Swati Sharma, Alefiya Hussain, and Huzur Saran. Experience with Heterogenous Clock-skew Based Device Fingerprinting. In *Proceedings of the 2012 Workshop on Learning from Authoritative Security Experiment Results*, LASER '12, pages 9–18, New York, NY, USA, 2012. ACM.

[Sim] Simple Event Logger, Kevin LaFramboise. `https://github.com/krlaframboise/SmartThings/blob/\master/smartapps/krlaframboise/simple-event-logger.src/simple-event-logger.groovy`. [Online; accessed January-2020].

[SKK+18] P. Sundaravadivel, K. Kesavan, L. Kesavan, S. P. Mohanty, and E. Kougianos. Smart-Log: A Deep-Learning Based Automated Nutrition Monitoring System in the IoT. *IEEE Transactions on Consumer Electronics*, 64(3):390–398, Aug 2018.

[Smaa] SmartThings Classic Documentation: Classes and JARs. `https://docs.smartthings.com/en/latest/getting-started/groovy-for-smartthings.html#allowed-classes`. [Online; accessed January-2020].

[Smab] SmartThings Code Review Guidelines and Best Practices. `http://docs.smartthings.com/en/latest/code-review-guidelines.html`. [Online; accessed January-2020].

[Smac] SmartThings Community Forum for Third-party Apps. `https://community.smartthings.com/`. [Online; accessed January-2020].

[Smad] SmartThings Groovy IDE. `https://graph.api.smartthings.com/`. [Online; accessed January-2020].

[Smae] SmartThings Logging, Matt J Frank. `https://github.com/krlaframboise/SmartThings/blob/\master/smartapps/krlaframboise/simple-event-logger.src/simple-event-logger.groovy`. [Online; accessed January-2020].

[Smaf] SmartThings Official API Documentation, Samsung. `http://docs.smartthings.com/en/latest/ref-docs/reference.html`. [Online; accessed January-2020].

[Smag] SmartThings Official App Repository. `https://github.com/SmartThingsCommunity`. [Online; accessed January-2020].

[Smah] SmartThings Official Developer Documentation, Samsung. [Online; accessed January-2020].

[Smai] SmartThings Official Logging, Samsung. `http://docs.smartthings.com/en/latest/tools-and-ide/logging.html`. [Online; accessed January-2020].

[Smaj] SmartThings Web-service App Overview. `http://docs.smartthings.com/en/latest/smartapp-web-services-developers-guide/overview.html`. [Online; accessed January-2020].

[SPJ15] Jan Spooren, Davy Preuveneers, and Wouter Joosen. Mobile Device Fingerprinting Considered Harmful for Risk-based Authentication. In *Proceedings of the Eighth European Workshop on System Security*, EuroSec '15, pages 6:1–6:6, New York, NY, USA, 2015. ACM.

[SS16a] A. Sanjab and W. Saad. Data Injection Attacks on Smart Grids With Multiple Adversaries: A Game-Theoretic Perspective. *IEEE Transactions on Smart Grid*, 7(4):2038–2049, July 2016.

[SS16b] H. Sedjelmaci and S. M. Senouci. Smart grid security: A new approach to detect intruders in a smart grid neighborhood area network. In *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pages 6–11, Oct 2016.

[SSG+16] Anibal Sanjab, Walid Saad, Ismail Güvenç, Arif I. Sarwat, and Saroj Biswas. Smart Grid Security: Threats, Challenges, and Solutions. `http://arxiv.org/abs/1606.06992`, 2016.

[SSIPD] Samsung SmartThings Supported IoT Products (Devices). `https://www.smartthings.com/products`. [Online; accessed January-2020].

[ST17] Biljana L Risteska Stojkoska and Kire V Trivodaliev. A Review of Internet of Things for Smart Home: Challenges and Solutions. *Journal of Cleaner Production*, 140:1454–1464, 2017.

[Sym18] Symantec. Sandworm Windows Zero-day Vulnerability Being Actively Exploited in Targeted Attacks.

https://www.symantec.com/connect/blogs/sandworm-windows-zero-day-vulnerability-being-actively-exploited-targeted-attacks, 2018. [Online; accessed January-2020].

[TAA15] K. Tazi, F. Abdi, and M. F. Abbou. Review on Cyber-physical Security of the Smart Grid: Attacks and Defense Mechanisms. In *2015 3rd International Renewable and Sustainable Energy Conference (IRSEC)*, pages 1–6, Dec 2015.

[TDS+19] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia, and M. Gurusamy. DEFT: A Distributed IoT Fingerprinting Technique. *IEEE Internet of Things Journal*, 6(1):940–952, Feb 2019.

[The10] The Smart Grid Interoperability Panel - Cyber Security Working Group. Introduction to NISTIR 7628: Guidelines for Smart Grid Cyber Security. `http://www.nist.gov/smartgrid/upload/nistir-7628_total.pdf`, Sept 2010. [Online; accessed January-2020].

[TZL+17] 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 *26th USENIX Security Symposium (USENIX Security 17)*, pages 361–378, Vancouver, BC, 2017. USENIX Association.

[U. 14] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor and Y. Makris. Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain. *Proceedings of the IEEE*, 102(8):1207 – 1228, July 2014.

[URC+13] A. S. Uluagac, S. V. Radhakrishnan, C. Corbett, A. Baca, and R. Beyah. A Passive Technique for Fingerprinting Wireless Devices with Wired-side Observations. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 305–313, Oct 2013.

[U.S10] U.S. Food & Drug Administration. Class II Special Controls Guidance Document: Tissue Adhesive with Adjunct Wound Closure Device Intended for the Topical Approximation of Skin - Guidance for Industry and FDA Staff. https://www.fda.gov/MedicalDevices/ucm233027.htm, Nov 2010. [Online; accessed January-2020].

[U.S18] U.S. Food & Drug Administration. Classify Your Medical Device. https://www.fda.gov/medicaldevices/deviceregulationandguidance/overview/classifyyourdevice/, Aug 2018. [Online; accessed January-2020].

[VBC14] J. Valente, C. Barreto, and A. A. Crdenas. Cyber-Physical Systems Attestation. In *2014 IEEE International Conference on Distributed Computing in Sensor Systems*, pages 354–357, May 2014.

[WD16] Steve Watson and Ali Dehghantanha. Digital Forensics: the Missing Piece of the Internet of Things Promise. *Computer Fraud & Security*, 2016(6):5–8, 2016.

[WDY+19] Qi Wang, Pubali Datta, Wei Yang, Si Liu, Adam Bates, and Carl A. Gunter. Charting the Atack Surface of Trigger-Action IoT Platforms. In *Proceedings of 26th ACM Conference on Computer and Communications Security*, 2019.

[WHBG18] Qi Wang, Wajih Ul Hassan, Adam J. Bates, and Carl Gunter. Fear and Logging in the Internet of Things. In *Network and Distributed Systems Symposium (NDSS)*, Feb 2018.

[Wik] Wikipedia. `https://dumps.wikimedia.org/wikidatawiki/entities/`. [Online; accessed January-2020].

[Win] Wink. `https://www.wink.com/`. [Online; accessed January-2020].

[WL13] W. Wang and Z. Lu. Survey Cyber Security in the Smart Grid: Survey and Challenges. *Computer Networks*, 57(5):1344–1371, April 2013.

[X. 12] X. Li, I. Lille, X. Liang, R. Lu, X. Shen, X. Lin and H. Zhu. Securing Smart Grid: Cyber Attacks, Countermeasures and Challenges. *IEEE Comm. magazine*, 50:38–45, 2012.

[XWP14] Teng Xu, James B Wendt, and Miodrag Potkonjak. Security of IoT Systems: Design Challenges and Opportunities. In *IEEE Computer-Aided Design*, 2014.

[XZSH16] Q. Xu, R. Zheng, W. Saad, and Z. Han. Device Fingerprinting in Wireless Networks: Challenges and Opportunities. *IEEE Communications Surveys Tutorials*, 18(1):94–104, Firstquarter 2016.

[Y. 12] Y. Yan, Y. Qian, H. Sharif and D. Tipper. A Survey on Cyber Security for Smart Grid Communications. *IEEE Communications Surveys and Tutorials*, 14:998–1010, 2012.

[Y. 16] Y. Obeng, C. Nolan and D. Brown. Hardware Security Through Chain Assurance. In *Design, Automation and Test in Europe Conf. and Exhibition (DATE)*, pages 1535 – 1537, Dresden, March 2016. IEEE.

[YCW+14] L. Yang, P. A. Crossley, A. Wen, R. Chatfield, and J. Wright. Design and Performance Testing of a Multivendor IEC61850 #x2013;9-2 Process Bus Based Protection Scheme. *IEEE Transactions on Smart Grid*, 5(3):1159–1164, May 2014.

[YMA+16] Q. Yang, Rui Min, D. An, W. Yu, and X. Yang. Towards Optimal PMU Placement Against Data Integrity Attacks in Smart Grid. In *2016 Annual Conference on Information Science and Systems (CISS)*, pages 54–58, March 2016.

[ZDLZ14] Zhe Zhou, Wenrui Diao, Xiangyu Liu, and Kehuan Zhang. Acoustic Fingerprinting Revisited: Generate Stable Device ID Stealthily with Inaudible Sound. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 429–440, New York, NY, USA, 2014. ACM.

[ZH15] Shams Zawoad and Ragib Hasan. FaIoT: Towards Building a Forensics Aware Eco System for the Internet of Things. In *Services Computing (SCC), 2015 IEEE International Conference on*, pages 279–284. IEEE, 2015.

[ZJS+11] David (Yu) Zhu, Jaeyeon Jung, Dawn Song, Tadayoshi Kohno, and David Wetherall. TaintEraser: Protecting Sensitive Data Leaks Using Application-level Taint Tracking. *SIGOPS Operating Systems Review*, 2011.

[ZM16] Y. Zhou and Z. Miao. Cyber Attacks, Detection and Protection in Smart Grid State Estimation. In *2016 North American Power Symposium (NAPS)*, pages 1–6, Sept 2016.

[ZM17] L. Zhou and Y. Makris. Hardware-based On-line Intrusion Detection via System Call Routine Fingerprinting. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1546–1551, March 2017.

[ZMR17] Eric Zeng, Shrirang Mare, and Franziska Roesner. End User Security & Privacy Concerns with Smart Homes. In *USENIX SOUPS*, 2017.

VITA

LEONARDO BABUN

2004	B.S., Telecomm. and Electronics Engineering Universidad de Oriente Santiago de Cuba, Cuba
2013 - 2015	M.S., Electrical Engineering Florida International University Miami, Florida
2017	NSF Cybercorps Scholarship for Service Florida International University Miami, Florida
2018	Cybersecurity Researcher Department of Homeland Security (DHS) Arlington, Virginia
2019	Cybersecurity Researcher Johns Hopkins APL Laurel, Maryland
2016 - 2020	M.S., Computer Engineering Florida International University Miami, Florida
2016 - 2020	Doctoral Degree Florida International University Miami, Florida

SELECTED PUBLICATIONS, PATENTS, AND INVENTION DISCLOSURES

L. Babun, H. Aksu, L. Ryan, E. Bentley, K. Akkaya, and A. S. Uluagac, "Z-IoT: Passive Device-class Fingerprinting of Zigbee and Z-wave IoT Devices", IEEE ICC, 2020.

L. Babun, H. Aksu, A. S. Uluagac, *A System-level Behavioral Detection Framework for Compromised CPS Devices: Smart-Grid Case*, ACM TCPS, 2019.

B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac, *Information Tracking in Commodity IoT*, Usenix Security Symposium, 2018.

A. K. Sikder, L. Babun, and A.S. Uluagac, *AEGIS: A Context-aware Security Framework for Smart Home Systems*, ACSAC, 2019.

L. Puche, L. Babun, and A. S. Uluagac, *HDMI-Walk: attacking HDMI configuration networks via CEC one step at a time*, ACSAC, 2019.

J. Myers, L. Babun, E. Yao, S. Helble, and P. Allen, *MAD-IOT: Memory Anomaly Detection for the Internet of Things*, Workshop on Impact of Artificial Intelligence on IoT, IEEE GLOBECOM, 2019.

F. Naseem, L. Babun, C. Kaygusuz, J. Moquin, C. Farnell, M. Alan, and A. S. Uluagac, *CSPoweR-Watch: A Cyber-resilient Residential Power Management System*, IEEE GreenCom, 2019.

K. Denney, E. Erdin, L. Babun, and A. S. Uluagac, *USB-based Insider Threat Prevention through Hardware-Assisted Anomaly Detection*, EAI SecureComm, 2019.

L. Babun, Z. B. Celik, A. S. Uluagac, G. Tang, and P. McDaniel, *Verifying IoT Safety and Security in Physical Spaces*, IEEE S&P Magazine, 2019.

L. Babun, H. Aksu, M. Conti, G. Tolomei, and A. S. Uluagac, *Advertising in the IoT Era: Vision and Challenges*, IEEE COMMAG, 2018.

J. Lopez, L. Babun, H. Aksu, and A. S. Uluagac, *A Survey on Function and System Call Hooking Approaches*, Springer HASS, 2017.

L. Babun, H. Aksu, and A. S. Uluagac, *Identifying Counterfeit Smart Grid Devices: A Lightweight System Level Framework*, IEEE ICC, 2017.

L. Babun, H. Aksu, A. S. Uluagac, *A Method of Resource-Limited Device and Device Class Identification using System and Function Call Tracing Techniques, Performance, and Statistical Analysis*, U.S. Patent and Trademarks Office (U.S. Patent 10,242,193).

L. Babun, H. Aksu, A. S. Uluagac, *Detection of Counterfeit and Compromised Devices using System and Function Call Tracing Techniques*, U.S. Patent and Trademarks Office (U.S. Patent 10,027,697).

C. Kaygusuz, L. Babun, H. Aksu, A. S. Uluagac, *A Method for the Detection of Compromised Computing Devices with Machine Learning and Convolution Techniques*, invention disclosure submitted to Florida International University, 2018.

L. Babun, A. K. Sikder, Abbas Acar, and A. S. Uluagac, *A Method for Automatic Evaluation and Modification of Smart Applications to Enable Comprehensive Forensic Analysis for Smart Settings*, invention disclosure submitted to Florida International University, 2018.