

EFFECTIVE SECURITY SCHEMES FOR SMART HOME INTERNET OF THINGS

A Dissertation
Submitted to
the Temple University Graduate Board

in Partial Fulfillment
of the Requirements for the Degree
DOCTOR OF PHILOSOPHY

by
Chenglong Fu
May, 2022

Dissertation Committee:

Dr. Xiaojiang Du , Advisor, Dept. of Computer & Information Sciences
Dr. Yu Wang, Dept. of Computer & Information Sciences
Dr. Yan Wang, Dept. of Computer & Information Sciences
Dr. Qiang Zeng, External Reader, University of South Carolina

ABSTRACT

Recent advancements in low-cost and low-power hardware facilitate the prevalence of Internet of Things (IoT) to be adopted in various smart environments. Although our works mainly take the smart home as the target application scenario, they can be easily applied to other smart environments such as smart office and industry. These devices and systems bring higher-level context sensing capability and enable intelligent and autonomous actions. However, these advantages are greatly undermined by concerns over the systems' security issues. Due to constraints on cost, power, and size, a large number of high-risk vulnerabilities of IoT/CPS (Cyber Physical System) systems are being discovered every year, which allow attackers to exploit them and expand their attacks from cyberspace to the physical world and causes severe damages. To cope with these emerging threats, we systematically investigate constituent components of IoT systems. Based on our exploration, we reveal new vulnerabilities and propose effective defense and detection mechanism.

In one work, we propose a novel semantic-aware anomaly detection scheme for smart home IoT systems, which provides timely anomaly alerts with a very low false alarm rate. In this scheme, we innovatively present the inter-device correlations as an uniform representation for profiling normal behaviors of smart home IoT systems. We utilize semantic information such as automation rules and device attributes to generate hypothetical correlations and then test them using collected event logs. The accepted correlations are then applied to the real-time events stream which can raise alarms when there is violation. We build a prototype anomaly detection system and evaluate it on four real-world testbeds. The evaluation results show that the accuracies are higher than 97%. In another work, we design an audio adversarial examples (AEs) detection system that can protect any system that uses audio speech

recognition (ASR), such as smart speakers. We observe that existing audio AEs cannot transfer among different speech recognition model. Based on this observation, we propose to apply multiple speech recognition models to an input audio samples concurrently and detect AEs as those have inconsistent transcripts. We evaluate the our prototype detector on 1,125 benign audio samples and 1,125 AE samples. The results show an detection accuracy of over 98%.

We conduct detailed investigation of IoT messaging protocols. We discover a new vulnerability of the IoT message timeout handling mechanism that broadly affects a large number of IoT devices and is not due to implementation flaws. We summarize our discovery as two attack primitives that can stealthily delay event and command messages, respectively. Further, we build three types of attacks based on them, which can manipulate the execution of automation rules to trigger erroneous actions or disable actions of safety-critical devices. We extensively evaluate the vulnerability on 50 IoT devices (belonging to 8 types) construct 11 proof-of-concept attacking cases that are collected from real-world user forums. The results show the effectiveness of the attack. We have reported the discovered vulnerability to major IoT vendors, including Apple, Google, Amazon, Ring, SmartThings, and SimpliSafe, and some vendors have discussed their counter measurements.

We also develop a blockchain-assisted distributed relay sharing scheme for smart home IoT systems. The scheme can effectively resolve the single points of failure (SPOF) problem of existing cloud based service architecture and prevent malicious behaviors through financial punishment.

ACKNOWLEDGEMENTS

First, I want to thank my advisor, Dr. Xiaojiang Du, for his constant and patient guidance and encouragement throughout my Ph.D. program studies. His dedication to academic research motivates me to pursue a lifelong career as a computer science researcher and contribute to the computer security community. I thank Dr. Qiang Zeng for his help on my research. His instructions help me significantly improve my research and writing skills and give me the confidence on challenging topics. I also want to thank my colleague Haotian who constantly discusses research ideas and technical solutions and gives a lot of precious comments. I thank Dr. Yu Wang and Dr. Yan Wang for their help on my preliminary and dissertation exams. I am very grateful for the support from other faculty members and students in the CIS department, which has contributed immensely to my growth at Temple University. Finally, I thank my family members for their unconditional support during all these years.

TABLE OF CONTENTS

	Page
ABSTRACT	ii
ACKNOWLEDGEMENTS	iv
LIST OF FIGURES	xi
LIST OF TABLES	xiii
CHAPTER	
1. INTRODUCTION	1
1.1 Motivation and Challenge	2
1.2 Major Contributions	3
1.3 Dissertation Overview	5
2. SMART HOME IOT ANOMALY DETECTION	6
2.1 Introduction	7
2.2 Background: Appified Smart Homes	9
2.3 Motivation and Threat Model	11
2.3.1 IoT Device Malfunctions	11
2.3.2 Attacks on IoT Devices	12
2.3.3 Threat Model	13
2.4 Correlations	14
2.4.1 Correlation Channels	14
2.4.2 Representation of Correlations	16
2.5 HAWatcher Design and Implementation	16
2.5.1 Workflow of Anomaly Detection	17
2.5.2 Semantic Analysis	18
2.5.3 Correlation Mining	20

2.5.3.1	Prepossessing Event Logs	20
2.5.3.2	Hypothetical Correlation Generation	21
2.5.3.3	Hypothesis Testing	24
2.5.4	Correlation Refining	25
2.5.5	Anomaly Detection	26
2.6	Evaluation	27
2.6.1	Experimental Setup	27
2.6.2	Training	30
2.6.3	Anomaly Generation	35
2.6.4	Performance of Anomaly Detection	37
2.6.5	Performance upon Smart App Changes	42
2.7	Literature Review	43
2.8	Chapter Summary	45
3.	SMART HOME IOT MESSAGE TIMEOUT VULNERABILITY AND DEFENSE	46
3.1	Introduction	47
3.2	Background	51
3.2.1	IoT Servers	51
3.2.2	Automation Rules	51
3.2.3	Inferring Private Information from Traffic	52
3.3	System Model and Attack Model	52
3.3.1	System Model	52
3.3.2	Attack Model	54
3.3.3	Attack of Unseen Devices	54
3.4	Demystifying IoT Timeout Behaviors	55
3.4.1	IoT Network Protocol Analysis	55

3.4.1.1	Transport Layer Protocols	55
3.4.1.2	Transport Layer Security Protocol	56
3.4.1.3	Application Layer Protocols	56
3.4.2	Description of Device Timeout Behaviors	57
3.4.3	Attack Primitives	58
3.5	Exploiting IoT Timeout Behaviors	61
3.5.1	Attack Procedure	61
3.5.2	State-Update Delay Attack	62
3.5.3	Action Delay Attack	62
3.5.4	Erroneous Execution Attack	63
3.5.4.1	Spurious Execution.	64
3.5.4.2	Disabled Execution.	64
3.6	Evaluation	65
3.6.1	Testbed Setup	65
3.6.2	Device Behavior Measurement	67
3.6.2.1	Target Events and Commands Recognition	67
3.6.2.2	Parameter Measuring Method	67
3.6.3	Device Timeout Measurement Results	68
3.6.3.1	Results of Cloud-based IoT Devices	68
3.6.3.2	Results of Local-based IoT Devices	70
3.6.3.3	Interesting Findings	71
3.6.4	Proof-of-Concept Attacks	72
3.6.4.1	State-Update Delay Attack	72
3.6.4.2	Action Delay Attack	73
3.6.4.3	Erroneous Execution Attack	74
3.7	Delay Attack Defense	75

3.7.1	Requiring Message ACK and Shortening ACK Timeout . . .	75
3.7.2	Timestamp Checking	75
3.7.3	Two-way Liveness Checking	76
3.7.4	Summary.	76
3.8	Literature Review	76
3.8.1	Device Blind Attacks	76
3.8.2	WiFi Jamming Attacks	77
3.8.3	Other IoT Vulnerabilities	78
3.9	Chapter Summary	79
4.	EFFECTIVE DETECTION OF AUDIO ADVERSARIAL EXAMPLE . . .	80
4.1	Introduction	80
4.2	Transferability	83
4.3	Diverse ASRs	85
4.4	System Design	86
4.5	Evaluation	88
4.5.1	Experimental Settings	89
4.5.2	Dataset Preparation	89
4.5.3	Feasibility Analysis	90
4.5.4	Comparison of Different Similarity Measurement Methods	91
4.5.5	Single-Auxiliary-Model System	92
4.5.6	Multiple-Auxiliary-Models System	93
4.5.7	Robustness against Unseen Attack Methods	94
4.6	Chapter Summary	95
5.	BLOCKCHAIN-ASSISTED RELAY SHARING OF SMART HOME IOT .	100
5.1	Introduction	100
5.2	Background	103

5.2.1	N-version Programming	103
5.2.2	Non-Repudiable TLS	104
5.2.3	Smart Contract	104
5.3	Blockchain and IoT Integration	105
5.3.1	Retrospection of Existing Works	106
5.3.1.1	Blockchain as a Ledger	106
5.3.1.2	Blockchain as a Service	107
5.3.2	Rethinking Blockchain and IoT Integration	108
5.4	Case Study of Efficient Blockchain and IoT Integration: IoT Remote Access	109
5.4.1	Threat Model	110
5.4.2	State-of-art Solutions	111
5.4.2.1	Port forwarding	111
5.4.2.2	Cloud-based Endpoint	111
5.5	Design of RS-IoT	112
5.5.1	Relay Sharing Model	113
5.5.2	Relay Workflow	113
5.5.2.1	Registration	113
5.5.2.2	Commission	114
5.5.2.3	Relay	115
5.5.2.4	Decommission	116
5.5.3	Billing of Relay Service	116
5.6	Proof-of-delivery	117
5.6.1	Components	118
5.6.1.1	Cover Stream Generator	119
5.6.1.2	Bytes Selector	120

5.6.2 Proof-of-Delivery Workflow	121
5.6.2.1 Bytes Commitment by the Controller Client	121
5.6.2.2 Bits Covering by the Relay Server	122
5.6.2.3 Bits Commitment by the IoT Device	122
5.6.2.4 Asynchronous Delivery Verification	122
5.7 Penalty & Disputation Solving	123
5.7.1 Reporting	123
5.7.2 Rebutting	124
5.7.3 Executing	125
5.8 Security Analysis	126
5.8.1 Possible Attacks	126
5.8.1.1 Random Scanning Attack	126
5.8.1.2 Attacks against the Relay Server	128
5.8.1.3 Malicious Relay Server Attack	128
5.8.2 Fairness Analysis	129
5.8.2.1 Cheating	129
5.8.2.2 Malicious Reporting	130
5.9 Experiment	131
5.10 Related Work	133
5.11 Chapter Summary	134
6. DISSERTATION CONCLUSION	135
BIBLIOGRAPHY	137

LIST OF FIGURES

Figure	Page
2.1 Examples of anomalies	7
2.2 The SmartThings architecture	10
2.3 Correlation channels	14
2.4 Architecture of HAWatcher	17
2.5 Examples of detection principle	18
2.6 Code snippet of the app <i>LightUpTheNight</i>	19
2.7 Floor plans of four testbeds.	27
2.8 Recall and precision of HAWatcher.	38
3.1 System model of smart home deployments	53
3.2 Attack model	53
3.3 Delaying IoT messages using a TCP proxy	59
3.4 Example attacks	61
4.1 An audio AE	81
4.2 Overview of the detection system	87
4.3 Similarity scores of transcriptions generated by DS0 and one of DS1, GCS and AT.	90
4.4 The ROC curves of three single-auxiliary-model systems.	92
5.1 Abstract architectures of IoT services.	108
5.2 Workflow of RS-IoT.	114
5.3 The workflow of proof-of-delivery.	119
5.4 The workflow of reporting suspicious packets.	124
5.5 The workflow of rebutting a pending report record.	125
5.6 Demonstration of possible attacks.	127

5.7 Reporting & Rebutting Gas Cost. 132

LIST OF TABLES

Table	Page
2.1 Part of the adjacency table	22
2.2 Numbers of rooms, devices and apps in each testbed.	28
2.3 IoT devices used in the four testbeds	28
2.4 Automation rules used in Testbed 1.	30
2.5 A portion of refined correlations acquired from Testbed 1.	31
2.6 Impact of Different Training-Phase Duration	33
2.7 HAWatcher’s detection performance	36
2.8 The number of false alarms caused by smart app changes.	42
3.1 Test results of cloud devices	66
3.2 Measurement results of local	70
3.3 Cases of event delay attacks	74
4.1 Recognition results of an AE by multiple ASRs	87
4.2 Datasets.	89
4.3 Comparison of 6 similarity metrics	96
4.4 The training results of three single-auxiliary-model systems	97
4.5 The testing results of three single-auxiliary-model systems	97
4.6 The testing results of four multiple-auxiliary-model systems when dif- ferent binary classifiers are adopted.	98
4.7 The detection results of unseen-attack AEs for three single-auxiliary-models.	99
4.8 The detection results of unseen-attack AEs for four multiple-auxiliary- models.	99
5.1 Keys of content in the serviceList.	115
5.2 Notations	118

5.3 Contracts Execution Cost.	131
---------------------------------------	-----

Chapter 1

INTRODUCTION

The development of low-cost embedded hardware and advanced network technologies facilitates the flourishing of the IoT market, which is expected to reach \$1,319 billion dollars by 2026 [1]. As the year of 2021, there are 12.3 billion connected IoT devices around the world and this number is projected to reach 27 billion in 2025 [2]. Among different types of IoT applications, smart home IoT draws the most attention as they are approachable by common users and bring significant convenience for their daily life. As in the year of 2021, 43 percent of U.S. households own smart home IoT devices [3]. Compared to traditional computing devices, smart home IoT devices are designed to interact with a wide range of physical properties such as sound, temperature, and light and equipped with wireless transmission capability. Depending on their applications, there could see either cheap IoT devices like smart home motion sensors and high-end specialized devices such as wireless insulin pumps. Along with the popularity of IoT devices, smart home IoT systems are evolving from remote controlled devices to integrated platforms. A series of smart home automation platforms such as Google Home [4], Alexa [5], and SmartThings [6] are rolled out. These platforms provide the broad interoperability among IoT devices from different vendors, and allow them to work autonomously according to user-specified automation rule. A typical smart home IoT platform comprises IoT devices, an automation server, and an mobile application. The user can access their devices through the mobile application and deploy automation rules on the automation server, which interacts with

IoT devices through event and command messages. The emerging of smart home automation platforms brings further convenience for ordinary users' life. For example, people are getting to rely on smart speakers for controlling home appliances, making online orders, and checking personal calendars and smart locks are widely adopted for rental properties owners to dynamically manage their tenants' access.

1.1 Motivation and Challenge

Despite the convenience brought by the emerging smart home IoT platforms, concerns about smart home IoT system's safety and security also arise as IoT platforms exposes new vulnerabilities to be exploited by attackers [7, 8, 9]. Due to smart home IoT devices' inherent capability to interact with the physical environment, these attacks can impose much greater threats to users' safety than traditional cyber-attacks by extending the effect of attack from the cyber-world to the physical-world. For example, there have been reported that home security systems can be exploited to make hoax calls to police[10] and a series of vulnerabilities are found in smart locks [11, 12] that allows attackers to maliciously unlock doors for burglary. However, IoT devices usually have inferior in security implementations to other computing devices like PCs and smart phones because of constraints on power and cost. There are dozens of large-scale IoT attacks such as Mirai [13] and Hajime [14] being reported each year recently and each of them can take down millions of IoT devices in short time.

To make things worse, the deployment of automation platforms and automation rules also attackers to manipulate IoT devices' operation without necessarily compromising them. Existing research works have shown that attackers can exploit vulnerabilities of automation servers [7], mobile applications [8], or network commu-

nication [9] to launch remote attacks. Utilizing these vulnerabilities, attackers can trigger devices' hazardous actions directly with fake command messages or indirectly with spoofed events [15]. As smart home IoT devices are becoming increasingly connected, discovery of new attack vectors will become more and more frequent in the future.

Although story of smart home IoT attacks grows longer and longer, few counter-measurements can be made by common users. Currently, security solutions to smart home IoT vulnerabilities depend heavily on the device vendor's effort of distributing security patches, which means users can only passively wait for vendors' remedies even after the disclosure of vulnerabilities. However, most of the device vendors provide support for their products for a limited time due to the cost constraint and then leave newer vulnerabilities to stay unpatched forever [16].

1.2 Major Contributions

In this dissertation, we take a systematical investigation about all components of modern smart home IoT systems and improve their security by inventing novel attack/intrusion detection methods, introducing secure-by-design IoT service architectures, and discovering new vulnerabilities. Targeting the existing security challenges, we build practical solutions that are depolyable by common users. Our solutions are based on existing smart home IoT systems such as SmartThings, Alexa, Google Home, and HomeKit and requires minimal effort from users and device vendors. The contributions of this dissertation can be summarized as follows:

- We propose an effective anomaly detection system for the smart homes IoT system, which innovatively infuse smart home semantic information with data mining methods. We develop a NLP-based method to extract semantic in-

formation in the form of inter-device hypothetical correlations and combine it with data mining methods in the correlation verification process. The collected correlations are explainable, and can be refined easily to resolve conflicts with smart apps and updated conveniently when apps change. We test our prototype on four real-world testbeds. Our detection system reaches a high precision of 97.83% and a recall of 94.12%, significantly outperforming prior approaches.

- We systematically investigate IoT devices' behaviors on handling network timeouts and discover the widely existed vulnerability among IoT devices. By exploiting the vulnerability, attacker will be able to manipulate IoT automation rules' execution without triggering alert at any layer. We propose a series of effective counter measurements for possible exploitation.
- We explore the generation and defense mechanism of audio adversarial examples (AEs). We generate a large audio AE dataset and empirically investigate their transferability. Based on our exploration, we propose a novel audio AE detection approach that concurrently uses multiple voice recognition models. The proposed detection approach achieves an accuracy of 99.88% on our dataset.
- We conduct a comprehensive literature review about blockchain and IoT system and rethink their integration. Based on the investigation, we propose a novel blockchain-assisted decentralized relay sharing system as a solution to the IoT remote accessing problem. We design a smart contract-based relay service trading system where disputes are resolved by using smart contracts.

1.3 Dissertation Overview

In this dissertation, we first introduce the background and motivation and summarize our contributions. In Chapter 2, we present our work on smart home IoT anomaly detection. In Chapter 3, we describe our study on IoT device timeout exploitation and defense. In Chapter 4, we explore the audio adversarial example and present our audio AE detection method. In Chapter 5, we propose a blockchain-based relay sharing system for IoT remote access. The conclusion and future works are discussed in Chapter 6.

Chapter 2

SMART HOME IOT ANOMALY DETECTION

As IoT devices are integrated via automation and coupled with the physical environment, anomalies in an appified smart home, whether due to attacks or device malfunctions, may lead to severe consequences. Prior works that utilize data mining techniques to detect anomalies suffer from high false alarm rates and missing many real anomalies. Our observation is that data mining-based approaches miss a large chunk of information about automation programs (also called smart apps) and devices. We propose Home Automation Watcher (HAWatcher), a semantics-aware anomaly detection system for appified smart homes. HAWatcher models a smart home’s normal behaviors based on both event logs and semantics. Given a home, HAWatcher generates hypothetical correlations according to semantic information, such as apps, device types, relations and installation locations, and verifies them with event logs. The mined correlations are refined using correlations extracted from the installed smart apps. The refined correlations are used by a Shadow Execution engine to simulate the smart home’s normal behaviors. During runtime, inconsistencies between devices’ real-world states and simulated states are reported as anomalies. We evaluate our prototype on the SmartThings platform in four real-world testbeds and test it against totally 62 different anomaly cases. The results show that HAWatcher achieves high accuracy, significantly outperforming prior approaches.

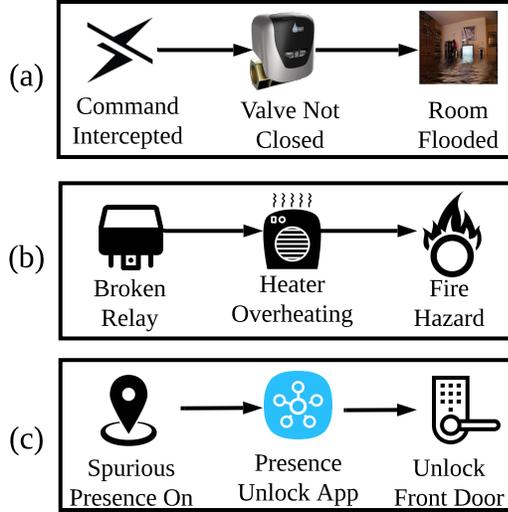


Figure 2.1. Examples of anomalies.

2.1 Introduction

With the rapid growth of Internet of Things (IoT), smart homes gain booming popularity. As predicted by Gartner, there will be more than 500 IoT devices deployed in a typical household by 2022 [17]. IoT devices become increasingly integrated, thanks to IoT platforms such as SmartThings [6], Homekit [18], and OpenHAB [19]. These platforms provide interoperability among home IoT devices by different vendors, and allow them to work according to user-specified automation programs (also called smart apps).

Despite advances in appified smart home, there are growing concerns about its safety and security [7]. First, IoT devices make it possible for cyber-space attacks to be extended to the physical world. As shown in Figure 2.1(a), the command of “close the valve” is maliciously intercepted, which may cause room flooding. Second, very often a device malfunction is hardly noticeable until certain consequences arise. As shown in Figure 2.1(b), an electronic heater controlled by a smart app “*It’s too*

cold” [20] could result in fires because of a broken relay (an electronically operated switch), which prevents the plug from shutting the power for the heater. Third, as IoT devices are chained together via automation [21, 22, 23], abnormal behaviors of one device might trigger undesired actions of another, which further exaggerates the impact of anomalies. As shown in Figure 2.1(c), a smart lock that automatically unlocks upon the resident’s presence is unlocked due to a fake event of the presence sensor.

To address these concerns, many anomaly detection systems [24, 25, 26, 27, 28, 29, 30] utilize data mining techniques to profile the system’s normal behaviors and report events that deviate from profiles as anomalies. However, these works usually take event logs as inputs without fully considering each event’s semantics, which actually may be acquired from smart apps, device types, and device functionalities. The limitations are threefold. First, the logic of some smart apps is too complex to be mined accurately, causing false negatives and positives. For example, the event pattern introduced by the smart app logic “Turn off a smart plug 30 minutes after two motion sensors in the living room are both motionless” is difficult to be mined considering the ‘AND’ logic between two motion sensors and the 30 minutes action delay. As a result, an anomaly “the smart plug fails to turn off” may not be detected. Second, the learning results are typically difficult to interpret; thus, they can hardly be explained and often confuse users. Third, the learning results cannot be updated quickly when smart apps or configuration changes. A long re-training process is then needed to adapt to the changes and many false alarms arise before the re-training is done.

Intuitively, incorporating semantic information, such as automation logic, device types, relations and installation locations, can help improve the accuracy of anomaly detection. However, there are a number of challenges to overcome in order to realize

this idea: 1) Standard data mining methods take event logs as inputs; however, it is unknown how to represent the diverse semantic information in the form of event logs. 2) System behavior patterns derived from smart apps and those mined from events logs may conflict. It is challenging to identify and resolve these conflicts. 3) When smart apps change, there are no effective methods to update the system profiling accordingly.

To fill the gap, we present Home Automation Watcher (HAWatcher), a novel anomaly detection system for appified home automation systems. We propose a semantics-assisted mining method that exploits diverse semantic information to construct hypothetical correlations (where a correlation describes how a device state or event correlates with another), and use event logs as evidence to verify them. Second, as the correlations are explainable according to the semantics, they can be easily refined to resolve conflicts with smart apps. Third, still thanks to explainability, they can be updated conveniently according to smart app changes. The correlations are then used by our *shadow execution* module to simulate normal behaviors in the *virtual world*. The simulated states are compared to those in the *real world* through both contextual checking and consequential checking, and inconsistencies during comparison are reported as anomalies.

2.2 Background: Appified Smart Homes

IoT devices in smart homes have become increasingly integrated via IoT platforms for rich automation. IoT integration platforms, such as SmartThings, Amazon Alexa, and OpenHAB, support trigger-action automation programs. On these platforms, despite the huge number of IoT devices, they are abstracted into a small number of *abstract devices*. For example, a smart light, regardless of its brand, shape, size, and wireless

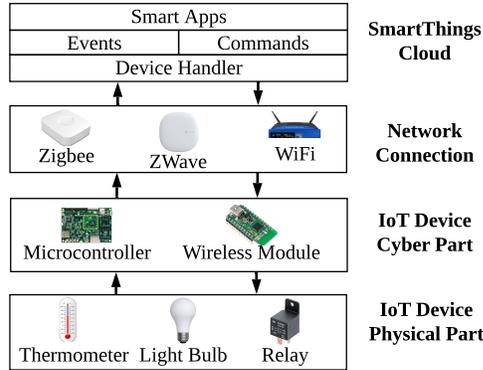


Figure 2.2. The SmartThings architecture.

technology, is abstracted into the same abstract device, *light*. Each abstract device has its associated *events* and *commands*. Device vendors can have their products support integration by realizing the events and commands.

We choose SmartThings [6] as an example IoT integration platform to present our design, as SmartThings is one of the leading platforms and supports sophisticated automation logic. Other integration platforms, such as Amazon Alexa, have similar structures. As illustrated in Figure 2.2, a typical SmartThings deployment has a cloud-centric architecture of four layers. On the top is the SmartThings cloud, where smart apps run and interact with abstracted capabilities. The cloud communicates with IoT devices through the network connection layer that uses various communication techniques such as WiFi, Zigbee, and ZWave. An IoT devices can be partitioned into the cyber part and the physical part. The cyber part manages interfaces for humans and bridges the communication between the cloud and the physical part, and the latter fulfills its functions in the physical world. Taking the Philips’ Hue smart light bulb as an example, the physical part is the LED light bulb and the cyber part is the embedded micro-controller with a built-in wireless component.

Next, we describe some terms used in SmartThings. A device has one or multiple *capabilities*, each categorized as an *actuator* or *sensor*. Each capability defines one or more attributes. For example, a smart plug device has an attribute “*switch*” and, optionally, an attribute “*power*.” Each attribute’s *state* (i.e., value) is stored on the cloud and updated due to events sent from the IoT device. For example, the SmartThings multipurpose sensor has a capability *contact sensor*, whose attribute “*contact*” changes from “*open*” to “*closed*” when SmartThings receives an event of “*contact closed*” from the sensor. In addition, the state of an actuator’s attribute is updated due to a *feedback event*, which is sent by the device after a command is executed by the actuator.

2.3 Motivation and Threat Model

IoT devices are notorious for their unreliability and insecurity [31, 32, 33]. Numerous anomalies in appified homes have been reported by users [34]. Below, we first discuss anomalies due to IoT device malfunctions and attacks as the motivation, and then present our goals and threat model.

2.3.1 IoT Device Malfunctions

We survey real-world anomalies frequently reported in the SmartThings user forum [34]. IoT devices interact with the IoT platform via events and commands; thus, we categorize malfunctions according to problematic events and commands.

Faulty Events. Faulty events refer to incorrect values reported by IoT devices. They can be caused by sensor defects or physical interference, such as mysterious door-knocking events [35] and motion events [36, 33, 37]. Faulty events may incorrectly trigger actuator actions and cause user confusions.

Ghost Commands. They are widely discussed in SmartThings’ user forum, dubbed ‘*poltergeists*’ [38, 39, 40]. For example, a smart plug was turned on itself at night, which overheated the connected waffle maker and electrical grill [41]. Users frequently reported their lights were turned on during midnight mysteriously [38].

Event Losses (or Large Delays). They refer to events that fail to be reported to the IoT cloud (in a timely manner). For example, mobile phone presence sensors were reported to suffer from a large delay on status update [42], which was confirmed by SmartThings [43]. Event losses may prevent the execution of related automation and leave the home in risky states. For example, the loss of a *presence-off* event could leave the door unlocked after the resident leaves home.

Command Failures. They correspond to commands issued by the IoT platforms that fail to be executed by the target devices. Command failures may be caused by malfunctions of a cyber part or physical part. (1) ***Cyber-part*** malfunctions that cause commands to fail to execute, such as system crashes and unstable network connections, are considered in our work. For example, the TP-Link smart plug often goes irresponsive [44]. (2) A ***physical-part*** malfunction is equivalent to a malfunction in a traditional (i.e., non-smart) device. For example, a broken electrical relay inside a smart plug can prevent the plug from cutting off the power supply [45], although from the perspective of the IoT platform, the plug has been turned off.

2.3.2 Attacks on IoT Devices

We survey the recent work on attacks against IoT devices, and find HAWatcher has the potential to detect the following five different types of attacks.

Fake Events. They are events maliciously injected by attackers. Fake events [8] may cause severe consequences by triggering actuator’s actions. As illustrated in Figure 2.1(c), a fake presence-on event can unlock the door.

Fake Commands. An attacker may inject fake commands to IoT devices. For example, Sonos smart speaker [46] and WeMo Smart switch [47] accept commands from the local network without authenticating their origins [48, 49].

Event Interceptions. Events can be intercepted and discarded by attackers. E.g., the home security system can be muted by intercepting the window and door sensors’ wireless connections to stop them from sending sensor events [50].

Command Interceptions. Similar to event interceptions, an attacker can also intercept a command and prevents it from being delivered to the device [51].

Compromised Devices. An attacker can compromise an IoT device and, at least, launch the following attacks. (1) *Stealthy Commands*. The attacker can control the device to execute commands [52] and, to keep stealthy, stops the corresponding feedback events from being sent out.¹ (2) *Denial of Executions (DoE)*. When a legitimate command is sent to the device, it does not execute the command but sends back a feedback event reporting the command has been executed.

2.3.3 Threat Model

We aim to detect both IoT device malfunctions described in Section 2.3.1 and attacks in Section 2.3.2. We clarify that HAWatcher can only detect attacks that violate correlations. Attackers who have knowledge of the correlations may construct attacks that do not violate any correlations and thus evade our detection.

We assume the IoT platform is not compromised. Like other anomaly detection work [24, 53, 30], we assume there are no or very few anomalies during training. We assume there are no malicious or conflicting rules in the installed smart apps; how to detect malicious logic [54] and conflicting rules [55, 23] are two separate research problems, and there are existing solutions to them [54, 23], including our

¹If feedback events are not muted, it is much like a Fake Command.

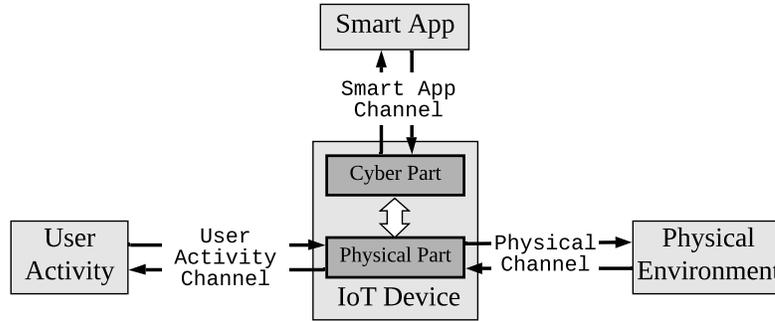


Figure 2.3. Correlation channels.

prior work [55, 56]. Gartner predicts that a typical household could have more than 500 IoT devices by 2022 [17]. Given the dense deployment in the near future, we exploit scenarios where an IoT device has one or more other devices nearby to interact with, and propose to leverage them to detect a device’s anomalous physical behaviors. Jamming that blocks communications reporting IoT events can be easily detected due to session timeout or missing sequence numbers; we thus do not further discuss it.

2.4 Correlations

Devices deployed in the same home may correlate in the form of co-present or temporally related events [57, 25, 24, 21]. These correlations can be attributed to the execution of smart apps [22], physical interactions [21] or users’ activities [57]. As shown in Figure 2.3, we investigate the causes of these correlations and categorize them into three channels below.

2.4.1 Correlation Channels

Smart App Channel. Smart apps not only directly cause correlations between triggers and actions as programmed, but also imply some extra correlations that

should be considered. For example, the smart App “*light follows me*” [58] leads to the correlation between the motion sensor and the light, and also implies a possible correlation worth verification, that is, “*if the light is turned on, then the motion should be in the active state*”. The implied correlation is true if the light is exclusively turned on by the smart app.

Physical Channel. Two devices can correlate via a certain physical property. First, an actuator device’s action can change a physical property, which is captured by nearby sensor devices observing that property. For example, a smart light’s action can affect an illuminance sensor nearby. Second, different sensor devices can be affected by the same physical event and generate temporally correlated IoT events. For instance, opening a door inevitably involves the door’s movement, which could be captured by both a contact sensor and an acceleration sensor installed on the door and results in two consecutive events. With increasing types of IoT devices deployed, physical-channel correlations can be pervasively observed on many physical properties, such as illuminance, power, sound, and temperature [21].

User Activity Channel. While user activities impose changes on devices, device states also reflect user activities. Thus, the user activity channel causes correlations between devices. For example, a TV being turned on typically implies that the user is nearby, which should be captured by the motion sensor. When a user returns home, there should be consecutive events, such as “*presence on*” showing the user’s proximity and “*contact-sensor open*” for door opening.

2.4.2 Representation of Correlations

An *event* reporting that the device A 's attribute α should be changed to the value a is denoted as $\mathcal{E}_a^{\alpha(A)}$, while a *state* which indicates that the device B 's attribute β has the value b is denoted as $\mathcal{S}_b^{\beta(B)}$.² We define two types of correlations.

- The *event-to-event* (*e2e*) correlation. It means that one event should be followed by (denoted as \rightarrow) another. For example, given a motion sensor A and a light B , the e2e correlation $\langle \mathcal{E}_{active}^{motion(A)} \rightarrow \mathcal{E}_{on}^{switch(B)} \rangle$ means the event $\mathcal{E}_{active}^{motion(A)}$ should be followed by the event $\mathcal{E}_{on}^{switch(B)}$.
- The *event-to-state* (*e2s*) correlation. It means that one event arising implies (denoted as \rightsquigarrow) a state is true. For example, $\langle \mathcal{E}_{high}^{power(plug)} \rightsquigarrow \mathcal{S}_{on}^{switch(heater)} \rangle$ means that, when the event $\mathcal{E}_{high}^{power(plug)}$ arises, the state $\mathcal{S}_{on}^{switch(heater)}$ should be true.

For the representation of a correlation involving conditions, its anterior event is combined with the conditions using the “ \wedge ” symbol. For example, $\langle \mathcal{E}_{active}^{Motion} \wedge \mathcal{S}_{present}^{Presence} \rightarrow \mathcal{E}_{on}^{switch(Light)} \rangle$ means the event $\mathcal{E}_{active}^{Motion}$, if the condition $\mathcal{S}_{present}^{Presence}$ is true, should be followed by $\mathcal{E}_{on}^{switch(Light)}$.

We show in Section 2.5 that the two types of correlations, despite their simplicity, are very effective in capturing rich semantic information and modeling the relations of devices that correlate via different channels.

2.5 HAWatcher Design and Implementation

We first introduce the workflow of anomaly detection (Section 2.5.1), and then describe the major modules in HAWatcher, as shown in Figure 2.4: 1) Semantic Analysis

²For simplicity of description, without causing confusion we sometimes omit the device IDs and use the simplified notations \mathcal{E}_a^α and \mathcal{S}_b^β .

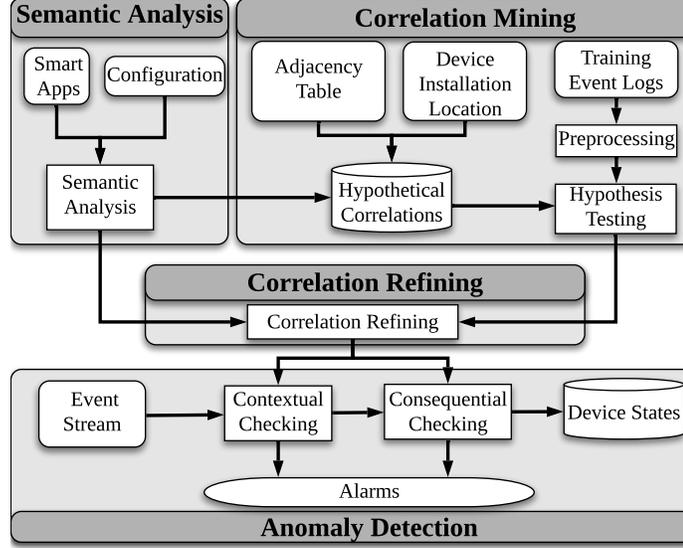


Figure 2.4. Architecture of HAWatcher.

(Section 2.5.2), 2) Correlation Mining (Section 2.5.3), 3) Correlation Refining (Section 2.5.4), and 4) Anomaly Detection (Section 2.5.5).

2.5.1 Workflow of Anomaly Detection

The Anomaly Detection module runs parallel with the appified home automation, and checks the events received from IoT devices against the learned correlations to detect anomalies. Figure 2.5 illustrates how this module detects anomalies, using anomalies depicted in Figure 2.1 as examples.

In case (a), the smart app automatically shuts the valve when water is detected. By applying semantic analysis to the app, HAWatcher extracts an e2e correlation $\langle \mathcal{E}_{detected}^{water} \rightarrow \mathcal{E}_{closed}^{valve} \rangle$. Since attackers intentionally intercept the command “close the valve” towards the valve, there is no feedback event $\mathcal{E}_{closed}^{valve}$, which contradicts the correlation. Furthermore, if it is a Command Failure caused by the valve’s cyber-part malfunction, HAWatcher can detect it the same way.

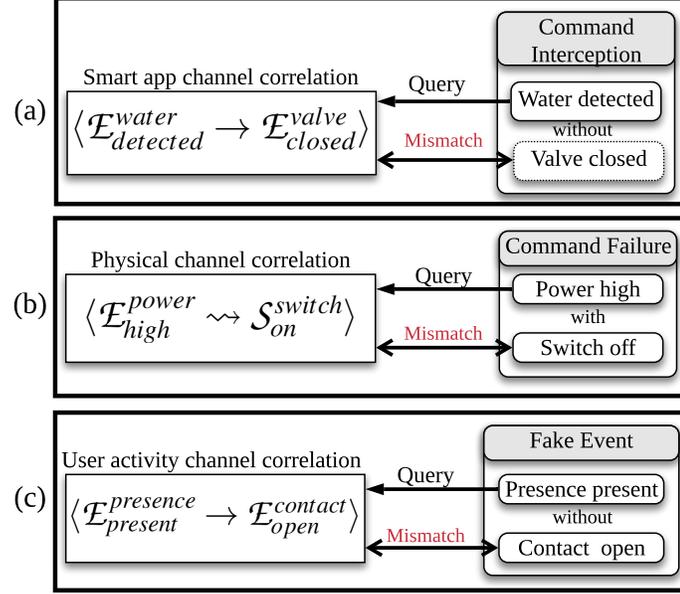


Figure 2.5. Examples of detection principle.

In case (b), the hypothetical e2s correlation $\langle \mathcal{E}_{high}^{power} \rightsquigarrow \mathcal{S}_{on}^{switch} \rangle$ is first proposed based on the physical channel and then gets confirmed using the training event logs. After a turning-off command is sent to the plug and executed by its cyber part (hence, its $Switch=off$), however, due to its broken relay, the plug still supplies power and thus the power meter reports events of high power usage, which violates the aforementioned correlation and triggers an alarm.

In case (c), as the resident does not actually return home, there is no event $\mathcal{E}_{open}^{contact}$ that follows the fake event $\mathcal{E}_{present}^{presence}$. This deviates from the user activity channel correlation $\langle \mathcal{E}_{present}^{presence} \rightarrow \mathcal{E}_{open}^{contact} \rangle$ and is thus reported as an anomaly.

2.5.2 Semantic Analysis

The Semantic Analysis module executes two steps: (1) extract semantics from smart apps and their configuration, such as the temperature threshold for turning on AC

```

def installed() {
  subscribe(lightSensor, "illuminance", illuminanceHandler)
}

def updated() {
  unsubscribe()
  subscribe(lightSensor, "illuminance", illuminanceHandler)
}

def illuminanceHandler(evt) {
  if (evt.integerValue < 30)
    lights.on()
  else if (evt.integerValue > 50)
    lights.off()
}

```

Figure 2.6. Code snippet of the app *LightUpTheNight*.

and which IoT devices are bound to which app, and (2) convert the semantics to correlations.

Semantic analysis has been used to detect malicious or risky smart apps as in [59, 7, 60]. We use the method described in our prior work [56, 55] to extract semantics in Step (1). It applies symbolic execution to the Intermediate Representation of apps and captures the configuration information, achieving precise semantics extraction. The extracted semantics of each app is represented as one or more *rules*, each in the form of a tuple *trigger(T)-condition(C)-action(A)*, which means that “if *T* occurs, when *C* is true, execute *A*.”

Step (2), which converts rules to correlations, is straightforward. Assuming *T* is reflected by the event E_1 , and E_2 is the feedback event due to executing *A*, the rule above is converted to a correlation $\langle E_1 \wedge C \rightarrow E_2 \rangle$.

Taking a SmartThings official app *LightUpTheNight* [61] shown in Figure 2.6 as an example, the Semantic Analysis module converts it into two e2e correlations: $\langle \mathcal{E}_{<30}^{Illuminance} \rightarrow \mathcal{E}_{on}^{Light} \rangle$ and $\langle \mathcal{E}_{>50}^{Illuminance} \rightarrow \mathcal{E}_{off}^{Light} \rangle$. Here, note that the condition (“*Illuminance* $\dot{<$ 30” or “*Illuminance* $\dot{>$ 50”) and the trigger event in each rule refer to the same attribute of the same device; we thus merge the trigger and the condition to derive a concise representation of the trigger events.

Moreover, as described in Section 2.4.1, given an e2e correlation $\langle \mathcal{E}_a^{\alpha(A)} \rightarrow \mathcal{E}_b^{\beta(B)} \rangle$ extracted from the smart app, we further propose a hypothetical e2s correlation $\langle \mathcal{E}_b^{\beta(B)} \rightsquigarrow \mathcal{S}_a^{\alpha(A)} \rangle$, which means that the event $\mathcal{E}_b^{\beta(B)}$ only arises when $\mathcal{S}_a^{\alpha(A)}$ is true. Such hypothetical e2s correlations are *not* necessarily true, and have to be verified using event logs (Section 2.5.3).

2.5.3 Correlation Mining

While there exist many pattern mining methods, few achieve both good usability and high accuracy in the context of applied home automation. Supervised mining methods [53, 62] are more accurate but require well annotated datasets or users' interventions. Unsupervised methods [24, 26, 25, 63] can be applied to unannotated data, but are less accurate.

Instead of relying on annotated datasets, we propose a semantic-based mining method. Semantic information includes devices' types and installation locations, which can be obtained from home automation platforms. Based on this information, HAWatcher proposes hypothetical correlations (in addition to those e2s correlations from smart apps) corresponding to physical channels and user activity channels. Each hypothetical correlation is then verified independently. Like other anomaly detection works [24, 53, 30], we assume there are no or very few anomalies during the training phase.

2.5.3.1 Preprocessing Event Logs

Preprocessing of event logs is necessary for two reasons: 1) Raw event logs are noisy with repetitive sensor readings. For example, some power meters periodically report similar (but slightly fluctuating) readings. 2) Devices' numeric readings cannot be

incorporated into logical calculations. We thus design a preprocessing scheme for redundancy removal and numeric-to-binary conversion.

For each device that generates numeric readings, we add up its readings from the entire training dataset and calculate its mean μ and standard deviation σ . Readings that fall outside the range $[\mu - 3\sigma, \mu + 3\sigma]$ are excluded as extreme values (i.e., the three-sigma rule [64]).³ Then, we apply the Jenks natural breaks classification algorithm [65]⁴ to the remaining readings and classify them as either ‘*low*’ or ‘*high*’. Next, for each device’s given attribute, we traverse the events and remove those that do not change the state (e.g., consecutive $\mathcal{E}_{high}^{Illuminance}$). Now, each two temporally adjacent events about the same attribute of a device have opposite values.

2.5.3.2 Hypothetical Correlation Generation

Besides those generated from the smart app channel, hypothetical correlations can be generated from the physical and user activity channels with other semantic information, such as device attributes and relations between attributes. We first utilize the semantic information to construct a table marking correlated attribute pairs; then, we fill each pair with devices that have matching attributes to generate hypothetical correlations.

For physical channel correlations, we consider seven *physical properties* that are related to many smart home IoT devices: illuminance, sound, temperature, humidity, vibration, power, and air quality. To determine whether two IoT device attributes may relate via a physical property, we develop an NLP (Natural Language Processing) based approach. Specifically, for each attribute of an abstract IoT device, we obtain its description from the SmartThings’ developer website [68] and parse it into a list

³Event exclusion is for training only; the anomaly detection module does not eliminate events.

⁴Jenks natural breaks algorithm and K-means algorithm give the same results for one-dimension data [66]

Table 2.1. Part of the adjacency table.

	Acceleration	CarbonDioxide	Contact	Illuminance	Motion	Power	Presence	Humidity	Sound	Button	Switch
Acceleration			✓		✓		✓		✓		✓
CarbonDioxide					✓		✓				✓
Contact	✓				✓		✓		✓		✓
Illuminance					✓		✓				✓
Motion	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
Power					✓		✓				✓
Presence	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
Humidity					✓		✓				✓
Sound	✓		✓		✓		✓				✓
Button					✓		✓				✓
Switch	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

A cell marked with ✓ means the corresponding attribute in the column may correlate with the one in the row head. The full table of 73*73 is in our technical report [67].

of separate words. To objectively evaluate the relatedness between an attribute and a physical property, we use Google’s pre-trained word2vec model [69] to calculate the semantic similarity scores between each word in the list and the physical property, and use the highest score as the *relatedness score* between the physical property and the attribute. For each physical property, we select the top ten attributes with the highest scores, which are considered mutually correlated via that physical property.

This way, we are able to find all correlated attribute pairs and mark them in an *adjacency table*, part of which is shown in Table 2.1. As SmartThings stipulates 73 attributes [68], the table is 73*73. A cell with ✓ means that the attributes in its row head and column head correlate.

While most of the cells are automatically generated, an exception is the *switch* attribute: as all actuator devices have the switch attribute, we mark it as correlated

with all other attributes. For user activity channel correlations, we use *presence* and *motion* as the two special attributes that directly reflect users’ activities. As a user’s activity may affect all the attributes, in the adjacency table we mark *presence* and *motion* as correlated with all other attributes.

For a specific smart home, all attributes of the installed devices are checked against this adjacency table to find pairs that may correlate. Given a pair of correlated attributes α and β in the adjacency table, the device A with the attribute α , and B with β , we generate four hypothetical e2e correlations $\langle \mathcal{E}_a^{\alpha(A)} \rightarrow \mathcal{E}_b^{\beta(B)} \rangle$, $\langle \mathcal{E}_{a'}^{\alpha(A)} \rightarrow \mathcal{E}_b^{\beta(B)} \rangle$, $\langle \mathcal{E}_a^{\alpha(A)} \rightarrow \mathcal{E}_{b'}^{\beta(B)} \rangle$, $\langle \mathcal{E}_{a'}^{\alpha(A)} \rightarrow \mathcal{E}_{b'}^{\beta(B)} \rangle$, and four e2s ones ($\langle \mathcal{E}_a^{\alpha(A)} \rightarrow \mathcal{S}_b^{\beta(B)} \rangle$, $\langle \mathcal{E}_{a'}^{\alpha(A)} \rightarrow \mathcal{S}_b^{\beta(B)} \rangle$, $\langle \mathcal{E}_a^{\alpha(A)} \rightarrow \mathcal{S}_{b'}^{\beta(B)} \rangle$, $\langle \mathcal{E}_{a'}^{\alpha(A)} \rightarrow \mathcal{S}_{b'}^{\beta(B)} \rangle$, where a and a' (b and b' , resp.) are values of the attribute α (β , resp.) after numeric-to-binary conversion; symmetrically, we generate another eight hypothetical correlations with the events of B as anteriors.

Moreover, we propose to combine semantics from smart apps with semantics from the adjacency table. The intuition behind the combination is that when an action command in a smart app is executed, it usually imposes certain changes on one or more attributes. Given an e2e correlation containing a condition extracted from a smart app, we create a *virtual device*, which reports an event when both the trigger event arises and the condition is true. For instance, a virtual motion sensor is created according to the conditional trigger $\mathcal{E}_{active}^{Motion(M)} \wedge \mathcal{S}_{present}^{presence(PS)}$, which becomes active only when $\mathcal{E}_{active}^{Motion(M)}$ arises and PS is present. Next, the virtual device is used, just like the corresponding real device, to generate hypothetical correlations according to the adjacency table.

Our current prototype only considers devices installed in the same room for generating hypothetical correlations. While this can be relaxed by considering any two devices in the home, our current implementation makes a trade-off between the com-

prehensiveness of hypothetical correlations and the meaningfulness of the mined correlations.

2.5.3.3 Hypothesis Testing

It is worth emphasizing that hypothetical correlations are not necessarily true. That is why we need hypothesis testing, the process of verifying hypothetical correlations using event logs. Given a hypothetical correlation, we traverse event logs to find all events that match its anterior, and take each of them as a testing case. Then, we check whether the hypothetical correlation’s posterior event or state is consistent with the physical ground truth as recorded in event logs. For example, an event instance of $\mathcal{E}_{active}^{Motion}$ constitutes a testing case for the hypothetical correlation $\langle \mathcal{E}_{active}^{Motion} \rightarrow \mathcal{E}_{on}^{switch(Light)} \rangle$. This case is counted as a success if $\mathcal{E}_{on}^{switch(Light)}$ occurs within a short duration d after $\mathcal{E}_{active}^{Motion}$. In our implementation, $d = 60s$, which is long enough to wait for the feedback event to arrive but not too long as to accept an event not related to $\mathcal{E}_{active}^{Motion}$. Note the scheduling granularity of SmartThings is at per-minute level [70].

Checking these testing cases can be considered as a sequence of independent Bernoulli trials. We use the one-tail test [71] to evaluate each hypothetical correlation’s correctness. For a given correlation, we set the alternative hypothesis H_α as “*the correlation succeeds with a probability higher than P_0* ”. Correspondingly, the null hypothesis H_0 is “*the correlation succeeds with a probability no higher than P_0* ”. We choose the 95% fiducial probability as in common practices [72], which means that the correlation can only be accepted if the null hypothesis’s p-value is smaller than 5%.

2.5.4 Correlation Refining

The accepted hypothetical correlations should not be used directly for two reasons. First, conditions of smart apps may be overlooked if they remain unchanged during training. For instance, assume there is a smart app that, upon the front door opening, turns on the porch light after sunset. If the residents always come back home after sunset, the inaccurate correlation $\langle \mathcal{E}_{open}^{contact} \rightarrow \mathcal{E}_{on}^{switch(PorchLight)} \rangle$ could be accepted by hypothesis testing and cause false alarms of “*porch light not turned on*” when the residents return before sunset. Second, when apps change, accepted hypothetical correlations may become outdated and contradict with the e2e correlations newly derived from apps. This can also cause false alarms, as confirmed by our experiments (Section 2.6.5).

We thus propose to refine mined correlations using e2e correlations extracted from smart apps, and launch the refining process whenever smart app changes or there are hypothetical correlations accepted by hypothesis testing. We first define the *cover* relation between two correlations: an e2e correlation $\mathbb{C}_s = \langle \mathcal{E}_a^{\alpha(A)} \rightarrow \mathcal{E}_b^{\beta(B)} \rangle$ extracted from a smart app *covers* a correlation $\mathbb{C}_h = \langle \mathcal{E}_c^{\gamma(C)} \rightarrow \mathcal{E}_d^{\delta(D)} \rangle$ that passes hypothesis testing if they meet two conditions: 1) they have the same posterior event (i.e., $\mathcal{E}_b^{\beta(B)} = \mathcal{E}_d^{\delta(D)}$); and 2) $\mathcal{E}_a^{\alpha(A)}$ (logically) implies $\mathcal{E}_c^{\gamma(C)}$ (i.e., $\mathcal{E}_a^{\alpha(A)} \Rightarrow \mathcal{E}_c^{\gamma(C)}$). If \mathbb{C}_s covers \mathbb{C}_h , the latter is removed. In the example mentioned above, a smart app derived e2e correlation $\langle \mathcal{E}_{open}^{contact} \wedge \mathcal{S}_{sunset}^{location} \rightarrow \mathcal{E}_{on}^{switch(PorchLight)} \rangle$ covers the mined correlation $\langle \mathcal{E}_{open}^{contact} \rightarrow \mathcal{E}_{on}^{switch(PorchLight)} \rangle$ because they have the same posterior event and $(\mathcal{E}_{open}^{contact} \wedge \mathcal{S}_{sunset}^{location}) \Rightarrow \mathcal{E}_{open}^{contact}$; thus, the latter correlation is removed.

2.5.5 Anomaly Detection

SmartThings does not provide access to its internal content, such as device states. To overcome the barrier, we design a *shadow execution engine*, which subscribes to the events of the installed IoT devices. It keeps track of all devices' states and simulates a smart home's legitimate behaviors based on obtained correlations.

For each incoming event, the shadow execution engine performs the *Contextual* and *Consequential* checking successively. The contextual checking verifies whether the event occurs in a valid context specified in e2s correlations. After that, the consequential checking searches for its consequential events as predicted by e2e correlations.

Below, we use the same example correlation (between a motion sensor and a light) as in Section 2.4.2. When an event $\mathcal{E}_{active}^{Motion(A)}$ is received, the shadow execution engine first conducts the contextual checking. It traverses all e2s correlations and locates those with the event $\mathcal{E}_{active}^{Motion(A)}$ at their anterior places. Among the located e2s correlations, if any of them have states in their posterior places that are inconsistent with the real-world devices' states, an alarm is raised reporting the event $\mathcal{E}_{active}^{Motion(A)}$ as invalid. Otherwise, the event is accepted and the shadow execution engine changes its simulated motion sensor's state to "active" accordingly. Then, for each accepted event (motion A turns "active" in the example), the shadow execution engine performs the consequential checking. It searches all e2e correlations that have $\mathcal{E}_{active}^{Motion(A)}$ at their anterior places and caches events at their posterior places in a waiting list. If any event in the list is not received within 60 seconds (consistent with d in hypothesis testing), the shadow execution engine reports an anomaly of a missing event. Moreover, an event from a real device also induces an event from its derived *virtual device* (defined in Section 2.5.3.2) if the involved condition is true, and the event of the *virtual device*

is handled in the same way as that from the real device through contextual and consequential checking.

2.6 Evaluation

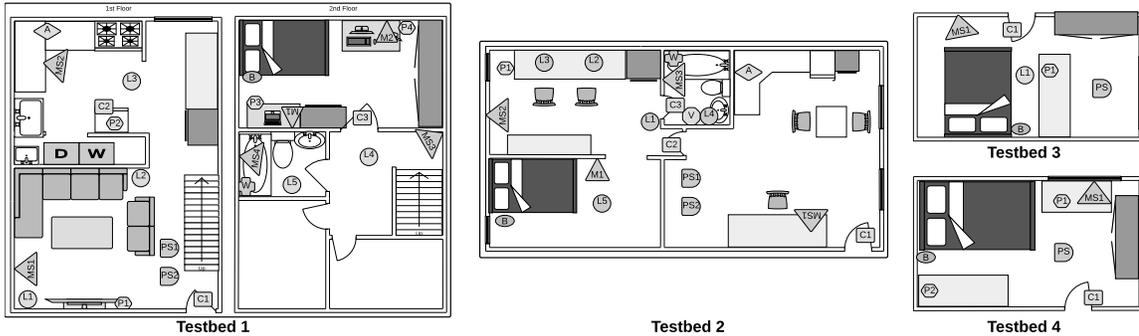


Figure 2.7. Floor plans of four testbeds..

We evaluate HAWatcher with datasets collected from 4 different real-world testbeds as shown in Figure 2.7. On each testbed, we spend three weeks collecting dataset for training and one week for testing. We apply collected correlations to each event from the testing datasets to evaluate HAWatcher’s performance. We compare HAWatcher with other anomaly detectors.

2.6.1 Experimental Setup

While there are several existing datasets from smart homes or home activity learning researches, such as [73, 74], none of these are collected from appified home testbeds. In addition, these testbeds contain mainly sensor devices but very few actuator devices. These make them unsuitable for evaluating HAWatcher, which is designed to work with appified homes. Next, we describe how we set up our testbeds.

Table 2.2. Numbers of rooms, devices and apps in each testbed..

Testbed	#Rooms	#Devices	#Smart apps
1	5	23	17
2	4	19	11
3	1	6	7
4	1	6	4

Table 2.3. IoT devices used in the four testbeds .

Abbr.	Device Name	Attributes	Deployment
<i>M</i>	<i>SmartThings Motion Sensor</i>	motion	on wall
<i>MS</i>	<i>Zooz 4-in-1 Sensor</i>	motion, illuminance, humidity	on wall
<i>W</i>	<i>SmartThings Waterleak Sensor</i>	water	on bathroom floor
<i>C</i>	<i>SmartThings Contact Sensor</i>	contact, acceleration	on doors
<i>B</i>	<i>SmartThings Button</i>	button	bedside
<i>L</i>	<i>SmartThings Light Bulb</i>	switch	as ceiling light, lamp
<i>PS</i>	<i>SmartThings Arrival sensor</i>	presence	in wallet
<i>P</i>	<i>SmarThings Smart Plug</i>	switch, power	to control fan, computer, and lamp
<i>A</i>	<i>Netatmo Air Station</i>	carbonDioxide, sound, hu- midity	on kitchen countertop
<i>V</i>	<i>ThreeReality Smart Switch</i>	switch	to control fan

Testbeds and Participants. We deploy SmartThings systems in four homes and Table 2.2 lists their basic information. Testbeds 1 and 2 each have two residents, and testbeds 3 and 4 have one resident each. The six (6) participants consist of 5 graduate students and 1 undergraduate student including two females and four males. Two of them are members of our research lab and none are authors of this work. None of them had prior experience of using home automation systems. For each testbed, we let the resident(s) propose desired automation, which is fulfilled by us with off-the-shelf IoT devices and smart apps from the SmartThings official repository. We then give them sufficient time to get familiar with the installed home automation before starting data collection.

Device Deployment. The device deployment is depicted in Figure 2.7. We deploy 10 different types of IoT devices as listed in Table 2.3, including their abbreviation labels. Note that the ThreeReality Smart Switch (denoted as **V**) can be attached to a wall switch to control traditional devices, such as lights and fans. The smart plug (denoted as **P**) can be used to control electrical appliances with power plugs; for example, in Testbed 1, **P1** and **P2** are connected to a TV and a fan, respectively, and **P3** and **P4** are connected to lamps.

Automation Rules. We extract automation rules from the installed smart apps in the form of “If *trigger* when *condition*, then *action*”. The extracted rules of Testbed 1 are listed in Table 2.4.

Ethical Concerns and Mitigation. We obtained the IRB approval for the study. All participants are fully aware of all the installed devices and apps. We do not use any sensitive devices such as cameras and microphones. The sound sensor of Device *A* in Table 2.3 only reports the sound level rather than the raw audio. All data is considered sensitive and personal identifiable information (PII) is removed right after collection for long-term storage. We store all the data in an encrypted

Table 2.4. Automation rules used in Testbed 1..

Index	Smart app rules
R1	If M1(active) when Mode(home), then P3(on)
R2	If M2(active) when Mode(home), then P4(on)
R3	If MS1(active), then L1(on) and L2(on)
R4	If MS1(inactive) for 15 minutes, then L1(off) and L2(off)
R5	If MS2(active), then L3(on)
R6	If MS2(inactive) for 10 minutes, then L3(off)
R7	If MS3(active), then L4(on)
R8	If MS3(inactive) for 5 minutes, then L4(off)
R9	If MS4(active), then L5(on)
R10	If MS4(inactive) for 15 minutes, then L5(off)
R11	If B(pressed), then toggle P3 and P4
R12	If B(held), then turn off all L and P and Mode(night)
R13	If B(double pressed), turn on P3 and P4 and Mode(home)
R14	If A($CO_2 \geq 950$), then P2(on)
R15	If A($CO_2 \leq 950$), then P2(off)
R16	If PS1 and PS2 (away), then turn off all L and P and Mode(away)
R17	If PS1 or PS2 (present), then turn on L1, L2, and P1 and Mode(home)

hard drive mounted to our lab’s server, which is only accessible to accounts of the authors.

For the purpose of testing, we need to inject anomalies (see Section 2.6.3). To avoid safety issues, the injected anomalies do not target any safety-sensitive devices, such as heaters. We notify participants of incoming testing one day ahead but do not disclose the details (e.g., device and time) of the anomaly cases. We also ask participants to keep their normal living habits and do not panic if they notice any anomalies. The purpose is to avoid their behavioral bias during testing. Details of the injected anomalies are presented to participants after the testing.

2.6.2 Training

Training HAWatcher. From Testbed 1, we generate 46 e2e correlations from the automation rules. In addition, we generate totally 2,398 hypothetical correlations,

Table 2.5. A portion of refined correlations acquired from Testbed 1..

ID	Correlation	ID	Correlation
C1	$\langle \mathcal{E}_{low}^{illumination(MS3)} \rightsquigarrow \mathcal{S}_{off}^{switch(L4)} \rangle$	C2	$\langle \mathcal{E}_{active}^{motion(MS1)} \rightarrow \mathcal{E}_{on}^{switch(L1)} \rangle$
C3	$\langle \mathcal{E}_{present}^{presence(PS1)} \rightarrow \mathcal{E}_{open}^{contact(C1)} \rangle$	C4	$\langle \mathcal{E}_{present}^{presence(PS1)} \rightarrow \mathcal{E}_{closed}^{contact(C1)} \rangle$
C5	$\langle \mathcal{E}_{present}^{presence(PS2)} \rightarrow \mathcal{E}_{open}^{contact(C1)} \rangle$	C6	$\langle \mathcal{E}_{high}^{power(P2)} \rightsquigarrow \mathcal{S}_{on}^{switch(P2)} \rangle$
C7	$\langle \mathcal{E}_{present}^{presence(PS2)} \rightarrow \mathcal{E}_{active}^{motion(MS1)} \rangle$	C8	$\langle \mathcal{E}_{pushed}^{button(B)} \rightarrow \mathcal{E}_{active}^{motion(M1)} \rangle$
C9	$\langle \mathcal{E}_{open}^{contact(C1)} \rightarrow \mathcal{E}_{active}^{acceleration(C1)} \rangle$	C10	$\langle \mathcal{E}_{on}^{switch(P4)} \rightarrow \mathcal{E}_{high}^{power(P4)} \rangle$
C11	$\langle \mathcal{E}_{active}^{acceleration(C1)} \rightarrow \mathcal{E}_{closed}^{contact(C1)} \rangle$	C12	$\langle \mathcal{E}_{on}^{switch(L4)} \rightarrow \mathcal{E}_{high}^{illumination(MS3)} \rangle$
C13	$\langle \mathcal{E}_{off}^{switch(L4)} \rightarrow \mathcal{E}_{low}^{illumination(MS3)} \rangle$	C14	$\langle \mathcal{E}_{on}^{switch(L3)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS2)} \rangle$
C15	$\langle \mathcal{E}_{on}^{switch(L3)} \rightarrow \mathcal{E}_{high}^{illumination(MS2)} \rangle$	C16	$\langle \mathcal{E}_{on}^{switch(P2)} \rightarrow \mathcal{E}_{high}^{power(P2)} \rangle$
C17	$\langle \mathcal{E}_{active}^{acceleration(C3)} \rightarrow \mathcal{E}_{active}^{motion(MS3)} \rangle$	C18	$\langle \mathcal{E}_{closed}^{contact(C1)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS1)} \rangle$
C19	$\langle \mathcal{E}_{on}^{switch(L4)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS3)} \rangle$	C20	$\langle \mathcal{E}_{closed}^{contact(C1)} \rightsquigarrow \mathcal{S}_{active}^{acceleration(C1)} \rangle$
C21	$\langle \mathcal{E}_{closed}^{contact(C3)} \rightsquigarrow \mathcal{S}_{active}^{acceleration(C3)} \rangle$	C22	$\langle \mathcal{E}_{active}^{motion(MS3)} \rightarrow \mathcal{E}_{on}^{switch(L4)} \rangle$
C23	$\langle \mathcal{E}_{high}^{illumination(MS3)} \rightsquigarrow \mathcal{S}_{on}^{switch(L4)} \rangle$	C24	$\langle \mathcal{E}_{low}^{illumination(MS1)} \rightsquigarrow \mathcal{S}_{off}^{switch(L1)} \rangle$
C25	$\langle \mathcal{E}_{present}^{presence(PS1)} \rightarrow \mathcal{E}_{active}^{motion(MS1)} \rangle$	C26	$\langle \mathcal{E}_{active}^{motion(MS1)} \rightsquigarrow \mathcal{S}_{on}^{switch(P1)} \rangle$
C27	$\langle \mathcal{E}_{active}^{acceleration(C1)} \rightsquigarrow \mathcal{S}_{open}^{contact(C1)} \rangle$	C28	$\langle \mathcal{E}_{active}^{acceleration(C2)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS2)} \rangle$
C29	$\langle \mathcal{E}_{on}^{switch(L5)} \rightarrow \mathcal{E}_{high}^{illumination(MS4)} \rangle$	C30	$\langle \mathcal{E}_{on}^{switch(P2)} \rightsquigarrow \mathcal{S}_{>950}^{CO_2(A)} \rangle$
C31	$\langle \mathcal{E}_{on}^{switch(P3)} \rightsquigarrow \mathcal{S}_{active}^{motion(M1)} \rangle$	C32	$\langle \mathcal{E}_{high}^{power(P3)} \rightsquigarrow \mathcal{S}_{on}^{switch(P3)} \rangle$
C33	$\langle \mathcal{E}_{open}^{contact(C2)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS2)} \rangle$	C34	$\langle \mathcal{E}_{>950}^{CO_2(A)} \rightarrow \mathcal{E}_{on}^{switch(P2)} \rangle$
C35	$\langle \mathcal{E}_{high}^{CO_2(A)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS2)} \rangle$	C36	$\langle \mathcal{E}_{high}^{sound(A)} \rightsquigarrow \mathcal{S}_{active}^{motion(MS2)} \rangle$
C37	$\langle \mathcal{E}_{open}^{contact(C1)} \rightsquigarrow \mathcal{S}_{present}^{presence(PS1)} \vee \mathcal{S}_{present}^{presence(PS2)} \rangle$		
C38	$\langle \mathcal{E}_{active}^{motion(M2)} \wedge \mathcal{S}_{home}^{mode} \rightarrow \mathcal{E}_{on}^{switch(P4)} \rangle$		

including 46 e2s correlations from the smart app channel, 544 from the physical channel, and 1,808 from the user activity channel. Then, the hypothetical correlations are checked using 22,655 events collected from the three weeks' training phase. In total, 146 correlations are accepted by hypothesis testing, and 130 remain after refining. On other three testbeds, the portion of smart app channel correlations are 32/109, 15/55, and 8/26, respectively. Table 2.5 lists a portion of the correlations after refining. Some correlations reveal interesting facts that are confirmed by the residents.

Observation 1: While $C1$ and $C3$ are both contact sensors, $C1$ has one additional correlation $C11 = \langle \mathcal{E}_{active}^{acceleration(C1)} \rightarrow \mathcal{E}_{closed}^{contact(C1)} \rangle$, which means the event $\mathcal{E}_{active}^{acceleration(C1)}$ should be followed by $\mathcal{E}_{closed}^{contact(C1)}$. This is because the front door (with $C1$) is typically closed right after being opened, while the bedroom door (with $C3$) does not have this pattern.

Observation 2: The e2s correlation $C23$ means that $MS3$'s illuminance goes *high* only when $L4$ is on. This is because there are no other light sources near $MS3$. Other illuminance sensors do not have such a correlation as the high illuminance value can be caused by multiple lights or natural lights.

Observation 3: Smart plugs $P2$ and $P4$ are to turn on/off a fan and a lamp, respectively. Whenever $P2$ and $P4$ are turned on, higher power use is observed (see e2e correlations $C16$ and $C10$ in Table 2.5). However, for $P1$ that is connected to a TV, $\mathcal{E}_{on}^{switch(P1)}$ is not followed by a power-high event, as the TV needs to be further turned on manually by the residents.

Observation 4: Physical- and user activity-channel correlations cannot be obtained without mining, since they are not included in any smart apps. On the other hand, some correlations can be easily extracted from smart apps but difficult to mine. For example, correlations that involve delays are difficult to be mined accurately, but can be precisely derived from rules, such as **R4**, **R6**, **R8**, and **R10**.

Training Baseline Approaches. We select the Association Rule Mining (ARM) [75] and the One-class Support Vector Machine (OCSVM) [76] based detectors as two baseline approaches. We choose OCSVM because it is widely used for anomaly detection and trained with one class of input data, which is suitable for our training data containing no or few anomalies [77]. ARM is selected because it is a well-established method for mining correlations/rules, and HAWatcher is also based on correlation mining.

Table 2.6. Impact of Different Training-Phase Duration.

Training phase (days)	Precision	Recall	# of false alarms	# of correlations
3	63.63%	78.69%	212	183
6	75.35%	85.78%	147	141
9	94.57%	94.12%	15	135
12	97.25%	94.12%	8	132
15	97.83%	94.12%	4	130
18	97.83%	94.12%	4	130
21	97.83%	94.12%	4	130

We perform ARM [75] on the same training dataset for comparison. Since ARM algorithms require transaction-form inputs, we segment the training dataset at places where the time interval between two consecutive events is longer than 60s (the same as the threshold d used for hypothesis testing). By using the library *pymining* [78], we mine 221 association rules with the confidence threshold of 0.95. Unlike our correlation mining method that covers various attributes and devices, rules produced by the association rule mining are dominated by motion sensors $MS3$ and $MS4$. All the 221 rules have either $MS3$ or $MS4$'s motion attributes in their consequent event set and 214 of them have that in their antecedent event set. There are 80 rules involving lights $L4$ and $L5$, 32 with illuminance sensors in $MS3$ and $MS4$, and 14 with the CO_2 sensor in A . Other attributes are not seen in any rules, as events involving them are overshadowed by those involving the four aforementioned attributes. In contrast, with our mining method, each attribute is involved in at least four (4) correlations and has an average of 10.5 correlations.

For the OCSVM-based detector, it takes a snapshot of all devices' states as a *frame* each time a new event arises and concatenates four consecutive frames as one input data *vector* [79]. We use the open source OCSVM implementation in sklearn [80] and the default kernel (Radial Basis Function).

Impact of Training-Phase Duration We study the impact of the duration of the training phase on the performance of HAWatcher. As Testbed 1 is the most complex one among the four testbeds, we select it in this experiment. As illustrated in Table 2.6, we start from using the first three (3) days of data as a training dataset, and then use the first six (6) days by increasing three days of data, and so on until we use all the 21 days of data. With each of the seven (7) training datasets, we train a system and evaluate its performance using the fourth week of testing data.

Based on the study and the results shown in Table 2.6, we have the following observations. (1) Nine (9) days of training data is enough for HAWatcher to achieve the highest detection recall, but its number of false alarms has not reached the lowest, which means some false correlations are obtained. (2) For the first two training datasets, although they lead to more correlations than the subsequent ones, the overall quality of correlations is not high. The reason is that we use the one-tail test (Section 2.5.3.3), which has two impacts. On the one hand, even a very small number of abnormal behaviors in the small datasets will cause some true correlations to be rejected. On the other hand, due to the small amount of data, many false correlations are not rejected yet. (3) Starting from the dataset of 15 days, the performance (including the number of false alarms) does not change anymore, which means that amount of data is sufficient for the testbed. (4) Those true correlations which have been rejected in the small datasets are recovered in the larger datasets. This shows the robustness of the design of HAWatcher. Even if very few anomalies arise during the training phase, true correlations can survive given sufficient training data. (5) We examine the different sets of correlations mined based on different duration and find that some false correlations remain there until more data is available. For example, $\langle \mathcal{E}_{high}^{humidity(MS3)} \rightsquigarrow \mathcal{S}_{closed}^{contact(C3)} \rangle$ remains until behaviors that fail the correlation appear on Days 11 and 12.

2.6.3 Anomaly Generation

To evaluate HAWatcher, we simulate 24 cases of anomalies on Testbed 1 listed in Table 2.7 (totally 62 cases on the four testbeds). We follow two criteria to select anomaly cases: (1) the attacks are discussed in the literature about IoT attacks; and (2) the malfunctions are frequently discussed in the SmartThings community. To simulate an anomaly case, we either modify the testing event logs (collected in the fourth week) or interfere with the home automation, and the resulting logs are used for anomaly detection. For each case, multiple instances (see the “#inst.” column) are injected.

If an attack has the same impact on the event logs as a malfunction, we group and simulate them as one case. Taking Case 1 as an example, we randomly inject a total of 50 motion events of MS1 into the testing event logs to simulate the effect of both Faulty Events (due to sensor malfunctions) and Fake Events (due to attacks).

Faulty/Fake Events. We simulate them by inserting events of devices, such as motion sensors [36], presence sensor [81], and contact sensors [35], as they are reportedly unreliable.

Event Losses/Interceptions. To simulate them, we randomly remove events of some devices from the testing event logs. We select various types of devices that users complain about event losses, such as presence sensors [43], contact sensors [82], and motion sensors [83].

Ghost/Fake Commands Both smart lights and plugs have been frequently reported by users for turning on/off unexpectedly [39, 40, 41]. We write a *ghost* smart app, which is not known by HAWatcher, and use the app randomly issue commands to turn on smart lights and plugs.

Table 2.7. HAWatcher’s detection performance.

Case	Type	Anomaly Description	Creation Method	#inst.	Precision	Recall	Correlations
1	Faulty/Fake Events	false motion(MS1) active	insert events into the dataset	50	97.77%	86.00%	C26
2		false contact(C1) open		50	100.00%	100.00%	C9
3		false acceleration(C1) active		50	97.87%	92.00%	C27
4	false presence(PS1, PS2) present	false presence(PS1, PS2) present		50	96.15%	100.00%	C3, C5, C25, C7
5		false button(B) pushed		50	100.00%	100.00%	C8
6	Event Losses/ Interceptions	missing motion(MS2) active	remove events from the dataset	57	100.00%	92.98%	C28, C14
7		missing motion(MS3) active		38	100.00%	100.00%	C17
8		missing contact(C1) open		11	78.57%	100.00%	C3, C5, C27
9		missing presence(PS1, PS2) present		9	77.78%	77.78%	C37
10	Ghost/Fake Commands	missing illuminance(MS3) events	toggle from the ghost smart app	46	100.00%	43.47%	C12, C13
11		turn on switch(P2):fan		50	100.00%	100.00%	C30
12		turn on switch(P3):lamp		50	100.00%	100.00%	C31
13	Stealthy Commands	turn on switch(L4):light	toggle from the ghost smart app and remove feedback events	50	100.00%	100.00%	C19
14		stealthily turn on switch(P2):fan		50	100.00%	100.00%	C6
15		stealthily turn on switch(P3):lamp		50	100.00%	100.00%	C32
16	Command	stealthily turn on switch(L4):light	cut off device power supply	50	100.00%	100.00%	C23
17		fail to turn on switch(L1):light		9	100.00%	100.00%	C2
18	Failures (cyber)/ Command	fail to turn on switch(L4):light	cover bulbs with paper	12	100.00%	100.00%	C22
19		fail to turn on switch(P2):fan		10	100.00%	100.00%	C34
20	Interceptions	fail to turn on switch(P4):lamp	unplug connected appliances	53	100.00%	100.00%	C38
21		Command		fail to turn on switch(L1):light	9	100.00%	66.67%
22	Failures (physical)/ Denial of Executions	fail to turn on switch(L4):light		12	100.00%	100.00%	C12, C1
23		fail to turn on switch(P2):fan		10	100.00%	100.00%	C16
24	fail to turn on switch(P4):lamp	53	100.00%	100.00%	C10		
Avg	-	-	-	-	97.83%	94.12%	-

“#inst.” indicates the number of instances for one testing case. As switch is a common attribute for all actuators, we point out the specific appliance controlled by each switch after the colon.

Stealthy Commands With compromised smart lights [52] and plugs [49], attackers can control them to make stealthy but hazardous actions. We simulate this type of attacks using the same method as ghost/fake commands but remove the feedback event of each fake command.

Command Failures (cyber)/Command Interceptions We simulate Command Failures (cyber-part malfunctions) and Command Interceptions on smart plugs [44] and smart lights [84]. We cut the power of target devices to make them unresponsive. For each target device, we conduct the experiment multiple times during one day.

Command Failures (physical)/Denial of Executions Command Failures (physical part malfunctions) and Denial of Executions are simulated on lights [52] and smart plugs [45]. We cover smart lights with a lightproof paper, and unplug appliances from smart plugs. The smart lights and plugs still respond to commands with feedback events, but those commands would not have any physical effect. For each case, we conduct the experiment multiple times during one day.

2.6.4 Performance of Anomaly Detection

We first evaluate HAWatcher’s precision and recall in detecting anomalies, and compare them with two baseline detectors. We then measure the false alarm rate of HAWatcher.

Evaluation Metrics. Given an anomaly case (see Table 2.7), *precision* is the number of correctly detected instances of that case divided by the number of alarms reporting that anomaly case (i.e., ratio of true anomalies to alarms), *recall* is the number of correctly detected instances of that case divided by the number of injected instances of that case (i.e., percentage of anomalies that can be detected), and the false alarm rate is the number of false positives divided by the number of IoT events.

$$\begin{aligned}
Precision &= \frac{True\ Positive}{True\ Positive + False\ Positive} \\
Recall &= \frac{True\ Positive}{True\ Positive + False\ Negative} \\
False\ Alarm\ Rate &= \frac{False\ Positive}{All\ Events}
\end{aligned}
\tag{2.6-1}$$

Detectors for Comparison. We compare the performance of HAWatcher with that of two baseline approaches described in Section 2.6.2, *ARM* and *OCSVM*. For the ARM-based detector, we segment the testing dataset as during the training phase, and check each segment against all mined rules to detect anomalies. For the OCSVM-based detector, as in [79], we take a snapshot of all devices’ states as a *frame* each time a new event arises and concatenate four consecutive frames as one data *vector*, which is fed into the trained OCSVM for detecting anomalies.

In addition, to evaluate the effect of semantic analysis of smart apps and correlation mining each and also to measure the benefit brought by the combination of the two, we build two variants of HAWatcher: *HAWatcher (Apps Only)*, which extracts correlations from smart apps only, and *HAWatcher (Mining Only)*, which mines correlations without using apps.

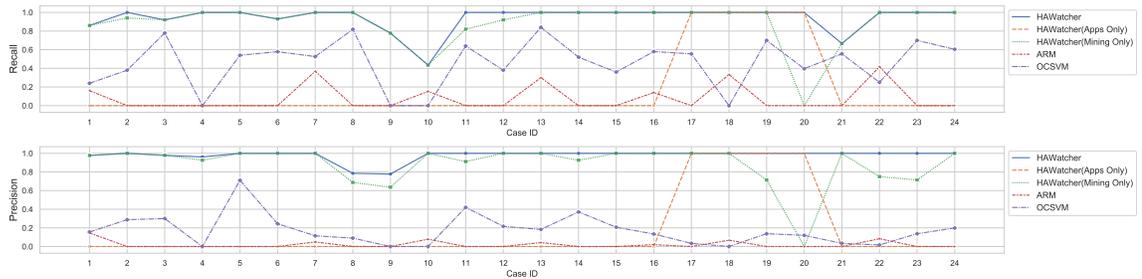


Figure 2.8. Recall and precision of HAWatcher..

Detection Results of HAWatcher. As shown in Table 2.7, HAWatcher has an average detection precision of 97.83% and a recall of 94.12% across the 24 diverse anomaly cases. For 18 out of 24 cases, HAWatcher successfully detects all the instances. Below we describe some examples to illustrate how HAWatcher detects anomalies.

Detecting Case 7. Residents entering/leaving the bedroom open the door, which is installed with an acceleration sensor $C3$, and cause the motion-active event of $MS3$. However, as motion-active events of $MS3$ are intercepted/lost, the user activity e2e correlation $C17 = \langle \mathcal{E}_{active}^{acceleration(C3)} \rightarrow \mathcal{E}_{active}^{motion(MS3)} \rangle$ is violated and the anomaly is hence detected.

Detecting Case 11. Ghost/Fake Commands that try to turn on $P2$ are detected due to a violation of the correlation $C30 = \langle \mathcal{E}_{on}^{switch(P2)} \rightsquigarrow \mathcal{S}_{>950}^{CO_2(A)} \rangle$, which is derived from the smart app rule **R14** and accepted by the hypothesis testing. The threshold 950 is easily extracted via semantic analysis of apps, but it would be difficult, if not impossible, for pure mining based approaches to learn it.

Detecting Case 14. A stealthy command in Case 14 tries to turn on the plug $P2$ to start the connected fan, which causes the event $\mathcal{E}_{high}^{power(P2)}$. However, Since the feedback event $\mathcal{E}_{on}^{switch(P2)}$ is intercepted by attackers, the switch of $P2$ is still at the state $\mathcal{S}_{off}^{switch(P2)}$. Thus, the physical channel e2s correlation $C6 = \langle \mathcal{E}_{high}^{power(P2)} \rightsquigarrow \mathcal{S}_{on}^{switch(P2)} \rangle$ is violated.

Detecting Case 20. Command Failures (cyber)/Command Interceptions are detected because of violation of the smart app channel e2e correlation $C38 = \langle \mathcal{E}_{active}^{motion(M2)} \wedge \mathcal{S}_{home}^{mode} \rightarrow \mathcal{E}_{on}^{switch(P4)} \rangle$: the commands are intercepted or not processed by the cyber part, so there are no feedback events $\mathcal{E}_{on}^{switch(P4)}$. In contrast, HAWatcher (Mining Only) cannot learn this correlation and thus misses all instances of this case.

Detecting Case 21. *L1* accepts the turning-on command and sends the feedback event, but due to a physical-part failure or DoE, the light is not on. While most of the instances of Case 21 can be detected as violation of the correlation $C24 = \langle \mathcal{E}_{low}^{illuminate(MS1)} \rightsquigarrow \mathcal{S}_{off}^{switch(L1)} \rangle$ (since the illuminance keeps low but the light-switch state is on), 3 instances are missed, because the room has been brightened up by natural light (hence, illuminance has already been high) when the anomaly arises.

For Cases 1, 3, 6, 9, and 10, some instances are missed, which should be attributed to *imperfection of anomaly simulation* (rather than the inability of HAWatcher). For example, seven instances of Case 1 are missed, because the fake motion-active events of MS1 happen to be injected during the time when there are real events of $\mathcal{E}_{active}^{motion(MS1)}$; such missed instances should not impose hazards, as the events are consistent with the fact that the residents are active during the time. Similarly, the 26 missed instances of Case 10 are illuminance readings which have similar values with real readings at the time. For Case 9, two instances are missed because two residents are back home together when one of their presence sensors' events get intercepted. In this situation, smart app **R17** will be triggered without difference by the other presence sensor and no hazard is caused.

Comparison. (1) As shown in Figure 2.8, HAWatcher achieves the best performance across all the 24 cases. (2) HAWatcher (Apps Only) merely obtains e2e correlations from smart apps, and can only detect anomalies, such as Command Failures (cyber)/Command Interceptions. It gets 16.67% for both the average precision and recall. (3) HAWatcher (Mining Only) has the *second best* performance. On average, its precision is 88.42% and recall 88.62%, showing the effectiveness and importance of our mining approach. However, due to the lack of knowledge of smart apps, it misses many instances of Cases 2, 11, 12, and 20. (4) The ARM-based detector has an average precision 2.03% and recall 7.79%. It fails to detect any anomaly

instances for 17 of the 24 cases, as its rules cover very few attributes (Section 2.6.2). (5) OCSVM performs slightly better with precision 17.15% and recall 45.19%. It fails for Cases 4, 9, 10, and 18, as events related to these cases do not fall inside the same input vector.

False Alarm Rate. We measure the false alarm rate of HAWatcher using the testing event logs (collected during the fourth week). We consider any alarms that are not due to our anomaly injection and cannot be categorized as any of the anomaly types listed in Section 2.3 as *false alarms*. HAWatcher reports totally 13 anomalies other than those injected by us. Among them, six (6) are due to violations of correlations C12, C13, C29, and C15, because of the large delays of some events from the illuminance sensors; three (3) are due to violations of correlations C20 and C21, because of the large delays of some events from the acceleration sensors. Such anomalies are categorized as true positives due to *Event Losses or Large Delays* (Section 2.3.1). They should be reported to users, as the large delay may confuse users and even cause undesired automation (e.g., an unlock-door command arrives late after the user has locked the door).

The other four (4) are due to user behavioral deviations: two are due to violation of C4 and C5, because there is one time that the residents stayed outside the door for a while (longer than 60 seconds) before opening the front door; C11 and C18 each cause one false alarm, and the reason is that the residents left the front door open for quite a while and then closed it. While it is arguable whether anomalies due to user behavioral deviations should be categorized as false alarms, we consider them false alarms, as they are not due to attacks or device malfunctions.

In total, HAWatcher reports four (4) false alarms from 9,756 events collected during a week, which makes 0.57 false alarms per day and a false alarm rate of 0.04%.

Table 2.8. The number of false alarms caused by smart app changes..

Rule	Type	Rule after change	HAWatcher	OCSVM	ARM
R3	Action change	If MS1(active), then L2(on) and L1(on)	0	14	0
R5	New rule	If MS2(active) B2(click), then L3(on) L3(toggle)	0	10	0
R8	Condition change	If MS3(inactive) for 5 15 minutes, then L4(off)	0	30	67
R10	Condition change	If MS4(inactive) for 15 30 minutes, then L5(off)	0	17	75
R14	Trigger change	If A(CO2 > 950 1000), then P2(on) for 15 minutes	0	17	0

In comparison, ARM and OCSVM cause 722 and 1,116 false alarms, respectively; that is, 103 and 159 per day and false alarm rates 7.40% and 11.44%, respectively.

2.6.5 Performance upon Smart App Changes

In an appified home, it is common that users change the smart apps, such as installing new apps and changing the configuration. However, traditional mining based anomaly detection needs a long time to adapt to the changes and, during the adaptation time, may trigger many false alarms. Handling such changes for anomaly detection in appified homes has been challenging. We conduct smart app change experiments to evaluate HAWatcher’s performance and compare it with other systems, OCSVM and ARM.

As listed in Table 2.8, we create five cases of smart app changes, which cover changes of trigger, condition, action, and the whole rule. For each case, we use one day to collect the data, and then apply HAWatcher, OCSVM, and ARM to the collected data. The results show that HAWatcher does not trigger any alarms, while OCSVM triggers many alarms for all the five cases and ARM for the changes of **R8**

and **R10**. We manually inspect the alarms and confirm that they are all false alarms caused by app changes.

ARM does not trigger false alarms for the changes of **R3**, **R5**, and **R14** because it does not include any association rules covering the devices, such as *L1* and *L3*, involved in the updated rules. For the OCSVM-based detector, each vector contains four consecutive snapshots of device states. In the case of **R3**, for example, the missing $\mathcal{E}_{on}^{switch(L1)}$ causes unseen vectors and thus triggers false alarms. For HAWatcher, upon app changes, the semantics of the updated apps are extracted and an updated set of correlations obtained. Thus, it is able to handle the changes without triggering false alarms.

2.7 Literature Review

With the emerging development of IoT devices and appified home automation, their security and privacy issues have drawn great attention [85, 55, 22, 23, 86, 59, 87, 60, 88, 89]. Most of them are focused on detecting threats, attacks and malware, rather than IoT malfunctions. For example, HomeGuard [56, 55] presents the first systematic categorization of threats due to interference between different automation apps, dubbed *cross-app interference* (CAI) threats, such as automation conflicts, chained execution, and loop triggering; it is also the first that uses SMT solvers to systematically detect such threats. It conducts symbolic execution to extract automation rules from apps, which is used in this work.

PFirewall [90] is a unique work that notices excessive IoT device data continuously flows to IoT automation platforms. It enforces data minimization, without changing IoT devices or platforms, to protect user privacy from platforms.

IoTSan [86] statically analyzes smart apps to predict whether the resulting automation may violate any safety properties. IoTGuard [22] instruments smart apps. Before an app issues a sensitive command, the action has to pass the policies defined by users. Both rely on pre-defined policies, while HAWatcher does not. Unlike our work, which detects IoT device anomalies, HoMonit [59] is focused on detecting misbehaving smart apps. Given a physical event, Orpheus [63] checks the system call trace due to the event against an automaton to detect attacks; it cannot detect anomalies such as fake events, event interceptions, etc.

Many anomaly detection detectors learn normal behaviors of a smart home from its historical data[91, 30, 24, 92, 26, 53, 29, 15]. For example, SMART [53] trains multiple user activity classifiers based on different subsets of sensor readings, and further trains another classifier that takes the vector of activity-classification results as its input to detect sensor failures. DICE [24] detects anomalies during state transitions by checking the context. Peeves [15] makes use of data from an ensemble of sensors to detect spoofed events.

The main difference of these existing anomaly detectors and our work is that HAWatcher extracts various semantics (device types, device relations, smart apps and their configuration), and infuses the semantics into the mining process. Not only is the detection more accurate, but each detected anomaly can be interpreted as a violation of a correlation, which itself is explainable. Prior to our work, it is unclear how a mining based approach is able to accurately learn complex behaviors in an appified home (e.g., Testbed 1 with 17 apps). HAWatcher provides an effective solution.

2.8 Chapter Summary

In an appified smart home, there exists rich semantic information, such as smart apps, configurations, device types, and installation locations. It is a promising direction to combine such semantic information with mining for anomaly detection. We presented a viable and effective approach in this direction: it exploits semantics on different channels (smart-app, physical, and user-activity) to propose explainable hypothetical correlations, which are tested using event logs and refined by smart apps. We built a prototype *HAWatcher* and evaluated it on four real-world testbeds against various (totally 62) anomaly cases, demonstrating its high accuracy and low false alarm rate. We view this work as a first step, rather than the final solution, in the direction of semantics-aware anomaly detection for appified smart homes.

Chapter 3

SMART HOME IOT MESSAGE TIMEOUT VULNERABILITY AND DEFENSE

This work unveils a set of new attacks against Internet of Things (IoT) automation systems. We first propose two novel attack primitives: IoT Event Message Delay and Command Message Delay. Our insight is that the timeout detection in the TCP layer is decoupled from the data protection in the Transport Layer Security (TLS) layer. As a result, even when a session is protected by TLS, its IoT event and/or command messages can be significantly delayed without triggering timeout alerts. It is worth highlighting that, by compromising/controlling one WiFi device in a smart environment, the attacker can delay the IoT messages of other non-compromised IoT devices; we thus call the attacks IoT Phantom-Delay Attacks. Our study shows the attack primitives can be used to build rich attacks and cause a variety of attack effects. The presented attacks are very different from jamming. 1) Unlike jamming, our attacks do not discard any packets and thus do not trigger re-transmission. 2) Our attacks do not cause disconnection or timeout alerts. 3) Unlike reactive jamming, which usually relies on specific hardware, our attacks can be launched from an ordinary WiFi device. Our evaluation involves 50 popular IoT devices and demonstrates that they are all vulnerable to the phantom-delay attacks. Finally, we discuss the countermeasures. We have contacted multiple IoT platforms regarding the vulnerable IoT timeout behaviors, and Google, Ring and SimpliSafe have acknowledged the problem.

3.1 Introduction

The global IoT market size was valued at \$212 billion in 2018 and is expected to reach \$1,319 billion by 2026 [1]. Many IoT platforms support integration of heterogeneous IoT devices and installation of powerful automation in a smart environment. While prior works have described attacks that exploit vulnerabilities of IoT devices or platforms [7, 60, 9], this work unveils a set of IoT attacks without relying on any implementation vulnerabilities.

The attacks are built on two *attack primitives*, dubbed *IoT Event Message Delay* (E-DELAY, for short) and *IoT Command Message Delay* (C-DELAY, for short). An IoT *event* is raised by an IoT device to report the device state to an IoT server (such as a “*motion active*” event), and an IoT *command* is issued by an IoT server to control a device (such as a “*lock front door*” command). Due to packet transmission time, a delay is inevitable from the moment an event or command is raised to the moment it is delivered. Such delays are typically sub-seconds and usually do not cause issues.

However, our study discovers that attackers who have the capability to sniff IoT devices’ traffic can maliciously increase the delay from sub-seconds to minutes or even hours without cause any alarms. As a result, when an IoT cloud server receives an IoT event that actually was raised quite a while ago, it incorrectly assumes the corresponding IoT device state update is real-time. Few works have studied the following issue: In smart environments (e.g., smart homes), outdated knowledge (held by the cyber-world) about the physical world could cause intriguing problems. Thus, this work investigates two critical questions. (1) Why are such large delays possible? (2) What attacks can be built?

To study the first question, we analyze the IoT network protocol stack to demystify IoT timeout behaviors. Our analysis starts from the TCP layer and moves upwards.

A key observation is that the timeout detection implemented in the TCP layer is decoupled from the data protection provided by the TLS layer. As a result,, an attacker can fool both the IoT device and server to believe that a session is healthy, while the attacker actually delays IoT messages (although the data integrity remains protected by TLS).

Consequently, the allowed delay of event/command messages is only bounded to the timeout behaviors implemented in the application layer, such as MQTT or HTTP. Our investigation shows that, despite the diversity of IoT devices, most of them allow C-DELAY from multiple seconds to sub-minutes, while the other attack primitive, E-DELAY, ranges from sub-minutes to hours.

To explore the second question, we exploit the two attack primitives, E-DELAY and C-DELAY, to build a family of attacks against smart homes. (They leverage recent advances in inferring smart home privacy by sniffing packets, and the inferred information, such as packet semantics, installed automation, and user activity patterns, can be very accurate [93, 94, 95, 96].) While some attacks merely incur automation delays, others can disrupt, disable, and override automation. They are categorized into three types.

Type-I: State-Update Delay Attack. Given a critical IoT device, a home owner should be notified of its state update as soon as possible (e.g., a pop-up notification on her smartphone). However, an attacker can apply E-DELAY to packets of an event message that reports an IoT state update. Despite the simplicity of the attack, it can cause severe consequences if the state update from certain IoT devices is delayed for minutes. Such critical IoT devices include: a water sensor that detects room flooding, a smoke detector that reports fires, a contact sensor that detects home invasion, and so forth. In these situations, every second matters.

Type-II: Action Delay Attack. With automation, an event can trigger a critical action. Hence, by applying E-DELAY to an event, the triggered critical action is also delayed. For example, a high carbon-monoxide level triggers an open-window action, an attacker can hold the message that reports “*CO high*” to delay the open-window action, which may be fatal to the residents. The attacker can also apply the C-DELAY primitive to the critical action to further delay the action. When multiple automation programs interplay in the same physical space, the Action Delay Attack can even be used to override a critical action (detailed in Section 3.5.3).

Type-III: Erroneous Execution Attack. An attacker may leverage the attack primitives to launch two kinds of Erroneous Execution. (1) ***Spurious Execution*** means that an action command that should not be issued is, in fact, issued. For instance, an attacker delays the state-update event that could have turned the automation condition to be *false* (i.e., the condition remains *true*); as a result, execution of the rule will trigger a spurious action. Given an example automation rule, “***trigger: the storm door is pulled, condition: presence is on, action: unlock the front door***”, which is to automatically unlock the front door when the storm door is opened, if and only if the owner’s presence sensor is “on”, then by delaying the “*presence-off*” state-update message, an attacker who pulls the storm door can trigger the spurious action of unlocking the front door. (2) ***Disabled Execution*** means an action command that should be issued is not issued. An attacker can apply E-DELAY to the message that could have turned an automation condition to be *true*, and hence the automation is disabled until after the delay.

The presented attacks are different from **jamming** (and other Denial of Service attacks) in three aspects: 1) The attacks do not cause timeout alarms in any layers of the IoT protocol stack on the device’s or the server’s side and, hence, existing IoT device and server implementations can hardly notice that they are being attacked. 2)

Unlike the jamming attack that discards packets, the messages carried by the delayed packets are eventually delivered, making our attacks less conspicuous to humans (e.g., a heater turn-off action is delayed, but when the user returns home, she finds that the heater is indeed turned off). 3) As the attacker has a fine control over the delay, the delivery of delayed messages can be used to construct more subtle attacks (such as attacks overriding IoT actions) that are impossible via jamming.

We measure the timeout behavior and delay range of 50 popular IoT devices, the measurement result shows that the two attack primitives are generally applicable. The possible E-DELAY varies among IoT devices, ranging from sub-minutes to hours, which allows a considerable time window for attackers. The results alarm that numerous IoT devices can be exploited by the attacks. They have the most significant impact on Apple’s smart home users, as HomeKit Accessory Protocol’s design allows event messages to be theoretically delayed with an infinite upper bound. Moreover, through proof of concept exploits, we demonstrate how an attacker can leverage the primitives to launch sophisticated attacks in the real-world environment.

This work studies a largely omitted topic—*IoT timeout behaviors*. It not only demonstrates that current IoT timeout behaviors are exploitable in cyber-physical systems like smart homes, but also reveals critical flaws in the existing IoT network protocol stack. We discuss several ways to handle the attacks. An intuitive counter measurement to our attacks is to use shorter timeout and keep-alive period settings. However, this would significantly increase the traffic load between IoT devices and cloud servers, especially when some users’ home networks suffer from large packet transmission delays. This causes a dilemma between undermining user experiences and being exploited by delay attackers. Checking messages’ timestamp at the application layer is another potential solution. However, timestamp cannot stop the

erroneous execution attack. More detailed discussions of potential counter measurements are given in Section 3.7.

3.2 Background

3.2.1 IoT Servers

The emerging home automation platforms facilitate convenient IoT automation in smart home systems. A home automation platform is composed of multiple entities, including *IoT servers* for executing automation. Based on their locations, IoT servers can be roughly categorized into two types: (1) *Cloud IoT servers*, such as those of SmartThings and Amazon, host IoT services on cloud servers. (2) *Local IoT servers*, such as an Apple’s HomeKit hubs, operate on a local device in users’ home. The cloud IoT servers can be further categorized into *endpoint servers* and *integration servers*. An endpoint server is operated by an IoT device vendor and directly interacts with their own devices. An integration server, on the other hand, indirectly interacts with third-party IoT devices via their endpoint servers using cloud-to-cloud communication. Some cloud-based servers combine the two functionalities. For example, SmartThings’s IoT servers can both connect IoT devices directly and integrate third-party devices via their endpoint cloud servers.

3.2.2 Automation Rules

An increasing number of IoT platforms support user-customized automation programs to allow devices to work autonomously. Despite the different syntax of automation programs among IoT platforms, an automation program is equivalent to one or more *automation rules*, each of which can be represented as a tuple of ***trigger-condition-action*** (TCA, for short) [97, 55, 98]: When the trigger event is received by an IoT

server, if the condition is evaluated as *true*, the specified action is taken. An IoT *event* is usually generated by an IoT device and sent to an IoT server; a delivered event triggers the execution of all automation programs that subscribe to the event. To take an action on an IoT device, an IoT server generates an IoT *command* and sends it to the destination device. Not all rules have *conditions*; in this case, a rule can be regarded as a special case of the TCA rule whose condition is always true [99, 100].

3.2.3 Inferring Private Information from Traffic

Recent research has made significant progress in inferring private information of a smart home from its *encrypted* traffic. Some work utilizes the metadata in encrypted traffic, such as packet headers, lengths, and frequencies, to recognize the device's identity [101, 96]. By incorporating the traffic patterns, recent work achieves high precision in inferring events [94, 93] and automation rules [95] from the traffic. Finally, from a sequence of recognized IoT events, patterns of user activities can also be inferred [102, 103, 104, 105]. This work does not advance information inference methods, but uses the inferred information to build *active* attacks.

3.3 System Model and Attack Model

3.3.1 System Model

A typical smart home deployment consists of IoT devices, a home WiFi router, and one or more IoT servers. The home router establishes and maintains a home area network and serves for both Internet and LAN access. As illustrated in Figure 3.1(a), for a *cloud-based IoT deployment*, each WiFi device has an individual network connection with the device vendor's endpoint server. These endpoint servers communicate with an integration server (e.g., SmartThings's server), which stores and executes

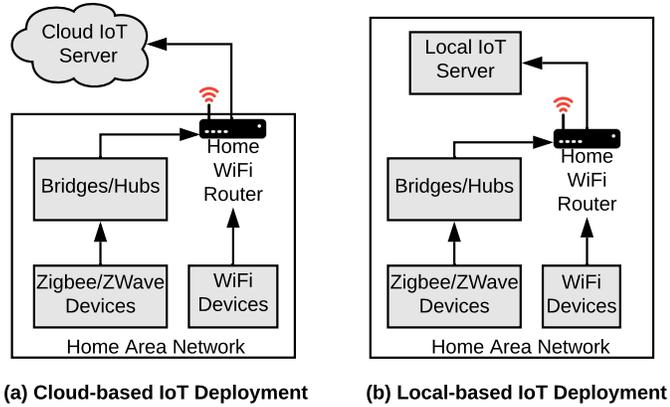


Figure 3.1. System model of smart home deployments.

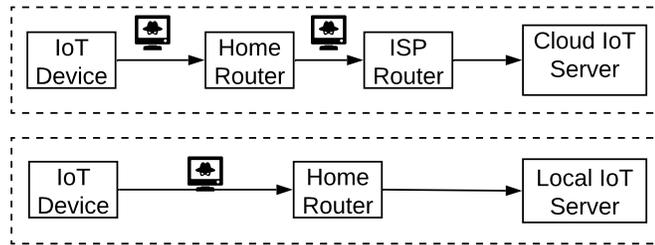


Figure 3.2. Attack model.

automation rules. As shown in Figure 3.1(b), for a *local-based deployment*, IoT devices connect with a local server (e.g., Apple’s HomePod) that executes automation programs.

This study focuses on IoT devices/hubs/bridges) that use the TCP/IP suite. Note that Zigbee/ZWave devices also need an IoT hub/bridge, which mostly use TCP/IP, to exchange data with IoT servers. For example, Philips Hue lights are connected to a Hue bridge via Zigbee communication, and the bridge maintains TCP connections with an IoT server.

3.3.2 Attack Model

Attacker’s Goal The *goal* of attackers is to go beyond the passive sniffing/eavesdropping attack and actively manipulate the smart home automation’s execution (which could put the smart home in an unsafe state, e.g., door is open), while not triggering alarms in any layers of the protocol stack on the IoT device’s or server’s side. This is very different from attacks that disrupt automation by jamming or discarding packets [106, 107], which can be detected by multiple layers of the protocol stack.

Attacker’s Capabilities As illustrated in Figure 3.2, we adopt the same assumptions as in several IoT message sniffing works [93, 96, 94, 108], which assume the attacker is able to monitor and access IoT devices’ IP traffic. It can be achieved by either a local attacker that exploits a compromised device or a home router [96, 94] or a remote attacker that has access to the network traffic from outside the home LAN (e.g., a malicious ISP) [108, 93].

Moreover, the proposed attacks do *not* rely on specific implementation vulnerabilities of IoT devices or servers. That is, the attacks have impacts on most IoT devices and smart homes. Communications between IoT devices and servers are protected by TLS. It is assumed that the attacker *cannot* obtain the encryption key, and cannot decrypt, forge, modify, or disorder any data protected by TLS.

3.3.3 Attack of Unseen Devices

For an unseen device whose timeout behavior is unknown, delay attempts targeting it could cause device offline alerts. However, by measuring only a few popular devices, attackers gain the capability to attack devices in many smart homes. For example, in the market of home security cameras, the top five brands take more than 30% of market share [109]. Also, products from the same vendor usually share similar

timeout behaviors. Considering the total sales of 42.5 million home security cameras in 2020, this means attackers can delay messages of more than 12.5 million home security cameras by only measure a few cameras.

3.4 Demystifying IoT Timeout Behaviors

In this section, we analyze IoT network protocols in different layers regarding their timeout behaviors. Our insight is that, the timeout detection in the TCP layer is decoupled from the data protection in the Transport Layer Security (TLS) layer, which allows attackers to delay messages without breaking data integrity in the TLS layer. Based on this insight, we present a practical method to delay IoT messages without triggering timeout at any layers. The attack method leads to our two attacking primitives: Command Message Delay (C-DELAY) and Event Message Delay (E-DELAY).

3.4.1 IoT Network Protocol Analysis

In Figure 3.3, we present a typical IoT network protocol stack. The layers beneath the transport layer are not end-to-end and do not impose barriers on attempts to delay IoT messages. Thus, our analysis starts from the transport layer and moves upwards.

3.4.1.1 Transport Layer Protocols

For the transport layer, out of the popular UDP and TCP protocols, we focus on the latter one because the former is rarely adopted for communicating with IoT servers [110, 31] and does not provide any timeout detection. As a connection-oriented protocol, TCP is designed to handle delayed, out-of-order, and lost segments by

requiring each transmitted segment to be acknowledged by the receiver. For this purpose, either side of a TCP connection maintains a *retransmission timer* and a *keep-alive timer* [111]. By default, if the ACK is not received before the retransmission timer expires, the sender will make several attempts to re-transmit the packet (with random backoff intervals). If all retransmission attempts fail, the sender will terminate the TCP connection and notify upper-layer protocols of the timeout. Besides, a TCP probing segment is sent if a session stays idle for a period that makes the keep-alive timer expire. Since the TCP ACK is not encrypted and is independent of the segment payload, attackers that have access to the TCP connection can avoid TCP timeout by generating fake ACKs for either normal and probing segments, which fool both sides to believe the connection remains to be healthy.

3.4.1.2 Transport Layer Security Protocol

The TLS protocol is adopted by most of today's WiFi-based IoT devices to provide security services [31]. An implicit sequence number is maintained by two parties of a TLS session and incremented after each successful transmission. The sequence number is included in the generation of message authentication code (MAC) for each record, which is protected by the TLS session key. Without the TLS session key, attackers cannot forge a valid MAC for disordered or replayed records. However, TLS does not provide timeout detection.

3.4.1.3 Application Layer Protocols

In the application layer, device vendors have the flexibility to choose existing protocols or build their proprietary protocols. Message Queuing and Telemetry Transport

(MQTT) and Hypertext Transfer Protocol (HTTP) are two of the most commonly used protocols for smart home IoT devices [31, 110].

The MQTT protocol, which is one of the most popular protocols for IoT devices [9], is based on a long-live session between a device and an IoT server, for full-duplex message pushing and subscribing [112]. The protocol requires the IoT device to send *PINGREQ* messages if the connection stays idle for a time longer than a predefined *keep-alive interval*. If a *PINGREQ* message is not received within 1.5 times of the keep-alive period [113], the server should reset the TCP connection with the device and raise a “*device offline*” alarm.

Unlike MQTT, HTTP-based protocols are connectionless and highly customizable [110]. Usually, the sender of an HTTP request waits for the response from the receiver. If the response is not received before the pre-defined period, the sender raises 408 ‘request timeout’ error [114] and drops the session. The pre-defined period is the message’s timeout threshold, which is configurable by both the server and the device sides. Besides normal messages that carry events and commands, some devices also exchange keep-alive messages with their server to maintain *long-live* TCP and TLS sessions. Excessive delay or missing keep-alive messages trigger timeout and cause the session dropped. In contrast, other devices only establish *on-demand* sessions with their servers upon sending normal messages and terminate the sessions once the transmission is completed.

3.4.2 Description of Device Timeout Behaviors

According to our analysis of IoT network protocols, timeout in the TCP layer can be completely avoided with spoofed ACKs. Hence, we focus on the timeout behavior at the application layer. We first classify IoT messages into two major types: (i) normal *IoT event/command* messages, and (ii) *keep-alive* (also known as heartbeat) messages

that are used to check the connection quality and IoT device or server’s liveness. We can describe an IoT device’s timeout behaviors using three parameters:

- **Timeout threshold of keep-alive messages.** This parameter is applicable to devices that use long-live session because the purpose of keep-alive messages is to detect and terminate non-responsive sessions.
- **Pattern of keep-alive messages.** This parameter describes what condition will keep-alive messages be exchanged. Keep-alive messages are exchanged either in a *fixed* period or non-periodically when the session stays *idle* longer than a interval. The keep-alive message pattern comprises the period and strategy (fixed or on-idle) of keep-alive messages.
- **Timeout threshold of normal IoT messages.** This parameter is only applicable to a portion of devices, as some devices do not implement timeout for event and command messages. For example, MQTT protocol does not require timeout for normal messages.

With the three parameters, an attacker can accurately predict the happening of the incoming timeout of an IoT device, and he can fine-tune the delay imposed on the IoT messages without causing timeouts.

3.4.3 Attack Primitives

With the analysis of IoT network protocols, we obtain two major insights: (1) We find that the timeout detection provided by insecure layers (such as TCP) can be fooled. (2) By observing the target device’s history traffic, attackers can derive the parameters to model the session timeout behaviors. Based on the parameters, attackers can accurately predict when timeout is to occur and achieve maximum delay without

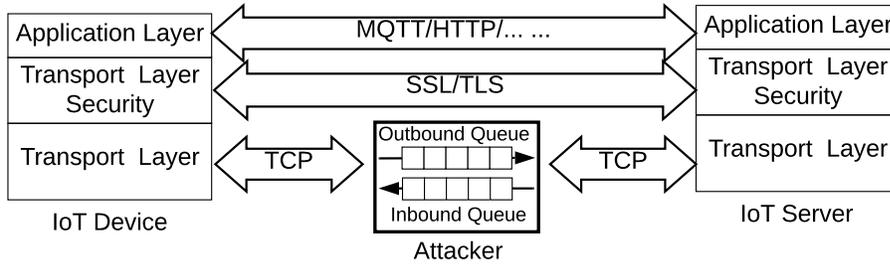


Figure 3.3. Delaying IoT messages using a TCP proxy.

causing device timeout. Following these insights, we build two attack primitives, E-DELAY and C-DELAY, for delaying event and command messages, respectively.

TCP timeouts can be easily bypassed with forged TCP ACKs. As shown in Figure 3.3, for each device-to-server TCP connection, attackers can split the connection between the IoT device and the corresponding cloud server into two separate TCP connections, with a transparent TCP forwarder in the middle. For each side’s connection, the attacker can hold the received packets before forwarding them to the destination but acknowledge the receiving immediately. Even if the delay is long enough (*i.e., connections are idle*) to trigger the TCP’s keep-alive timeout, the probing segments can also be acknowledged to avoid connection termination. After the end of the delay, all held packets are sent to their destination in their original order so that the TLS MAC verification is never violated. This way, transport layer protocols no longer have any restrictions on the delay time.

Because of the encryption provided by TLS, we cannot bypass application layer protocols’ timeout by forging device’s or server’s responses. Instead, we hold target event messages until the moment right before an application layer protocol timeout occurs. During the period of delay, any following messages are also delayed accordingly to avoid breaking the sequence number checking of TLS.

We present the concrete experiment steps to derive a device’s timeout behavior parameters. Attackers can perform these steps on his own devices to collect parameters, and then apply them to delay other devices of the same model. (1) By monitoring the device’s traffic on idle state, we can distinguish devices using on-demand sessions by their intermittent TCP/TLS sessions. For devices that are using long-live sessions, the packet length and period of keep-alive messages can be observed. (2) By manually triggering normal messages of keep-alive devices, we can confirm the keep-alive pattern. If the next keep-alive message is postponed accordingly, the device has non-periodically keep-alive messages that are exchanged when the session is idle. Otherwise, the keep-alive messages are exchanged in a fixed period. (3) We measure the timeout threshold of keep-alive messages by delaying a keep-alive message in idle state until the timeout happens. The interval between the beginning of the delay and the occurring of timeout is recorded as timeout period of keep-alive messages. (4) We measure the timeout threshold of normal (i.e., event and command) messages using the same method as keep-alive messages. We manually trigger the message right after a successful exchange of a keep-alive message and delay it until timeout happens. If the timeout occurs earlier than the anticipated timeout of the next keep-alive message (which is also delayed accordingly), we record this interval as the timeout of the corresponding message. Otherwise, it means the device does not implement a timeout for that type of message and session timeout is solely triggered by keep-alive messages.

Finally, we verify the collected parameters by randomly delaying a message and predicting the happening of timeout. The parameters are proved to be correct if the predicted timeout matches the real-world timeout.

In Section 3.6.3, we conduct evaluation experiments to measure delay times of 50 popular smart home IoT devices. According to our evaluation results (see Table 3.1

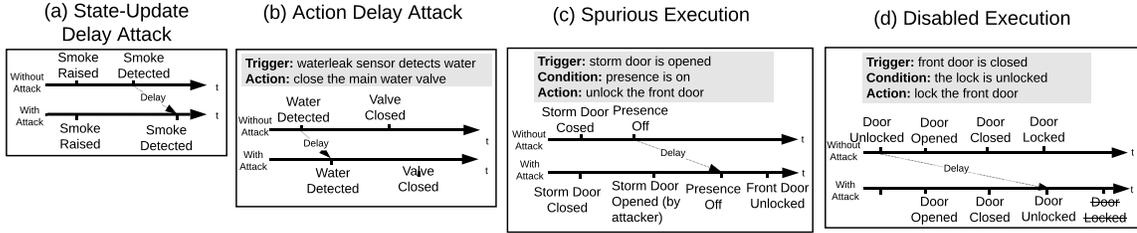


Figure 3.4. Example attacks .

and Table 3.2), both C-DELAY and E-DELAY are feasible on all 50 measured devices and allow a delay of 15 seconds. For most of the devices, E-DELAY can be longer than 30 seconds. A straightforward way to understand this is, for example, when a “*smoke-detected*” event arises, our E-DELAY attack primitive can delay it for more than 30 seconds without raising any alarms on the device’s or server’s side.

3.5 Exploiting IoT Timeout Behaviors

Although the adoption of TLS defeats event and command spoofing attacks, smart home automation system can still be manipulated using our delay primitives. We propose three types of attacks, based on C-DELAY and E-DELAY, that can delay critical state updates and actions, override or disable smart home automation, and even incur spurious actions.

3.5.1 Attack Procedure

We first overview the procedure to employ our delay attack primitives, which are summarized as three steps. In the first step, attackers obtain some popular IoT devices that they have full control of and measure their timeout behaviors and packet characteristics. In the second step, the attacker passively eavesdrops the network traffic in the victim’s home network and localize the target IoT device’s IP address

and packets by referring to the measurement results. In the third step, the attacker hijacks the target devices' traffic and use the TCP proxy to delay messages as described in Section 3.4.3. There are different ways to achieve the third step: 1) LAN attackers can hijack and redirect the target devices' traffic to a compromised or attacker's device using ARP spoofing [115]. ARP spoofing, although old and mature, is still shown to be effective on wide variety of IoT devices [116]. 2) For attackers that can access ISP's facilities or the victim's home router, this step can be done without using ARP spoofing.

3.5.2 State-Update Delay Attack

The attack utilizes the E-DELAY primitive to delay critical notifications, which defers users' awareness of hazardous situations. For instance, smart home safety monitoring systems have been used in millions of households for providing alerts regarding safety-related incidents [117]. For such IoT devices, *timeliness* is critical as a hazardous situation may occur unexpectedly and develop very fast. Figure 3.4(a) shows an example where a smart smoke detector is installed in the kitchen. Normally, the “*smoke-detected*” alert is sent to users' smartphones instantly. By applying E-DELAY to the smoke detector's event, even for only dozens of seconds, serious damages can be caused when users finally receive the delayed smoke alert.

3.5.3 Action Delay Attack

Given an automation rule, by applying E-DELAY to its trigger event and/or C-DELAY to its action command, the action can be delayed. Some automation rules have critical actions for IoT devices to respond to hazardous situations automatically. For example, as illustrated in Figure 3.4(b), with a water leak sensor installed in bathroom, water

leaking can trigger the shut-off action of a smart valve. Normally, in the case of a water leak, the valve can be shut off timely, which prevents flooding. By delaying the water leak sensor’s event and/or the *shut-off* command towards the valve, the leaked water can cause severe damages.

In particular, if a device is controlled by two separate automation rules that have opposite actions, delaying one action can *override* the effect of the other, effectively disordering the two actions. For example, a smart lock that is automatically unlocked when the user’s presence sensor becomes on, and then is locked when the contact sensor on the same door turns closed. When the user returns home, by delaying the door-unlock action until after the door-lock action is executed, the door stays unlocked overnight.

3.5.4 Erroneous Execution Attack

More than temporarily delaying device’s state-update and actions, the Erroneous Execution Attack imposes more severe safety risks by maliciously invoking or disabling automation rules. Exploiting the delay attack primitives, attackers can selectively delay an IoT event while leaving others untouched. As a result, the delayed event arrives at the IoT server later than other events even if it actually happens earlier in the physical world. The *disordered* events can be used to launch erroneous automation attacks. We first give a formal representation of automation rules that takes delays into consideration, and then describe two types of erroneous automation. Given an automation rule $R = \langle T, C, A \rangle$, where T , C and A represent the trigger, condition, and action of the rule R . A trigger event instance is denoted as \mathcal{E}_T , an event that changes the condition’s boolean value \mathcal{E}_C , and the evaluation function of the condition C is $f(C)$. The time an event \mathcal{E} is generated by an IoT device is denoted as $I(\mathcal{E})$ and the time it is received by the IoT server is denoted as $S(\mathcal{E})$. When there are no attacks,

we assume all events from a home to the IoT server use the same transmission time. Thus, if $I(\mathcal{E}_1) < I(\mathcal{E}_2)$, then $S(\mathcal{E}_1) < S(\mathcal{E}_2)$ without attacks. When \mathcal{E}_T is received by the IoT server, if $f(C) = true$, the action A is taken; otherwise, no action is taken. There are two kinds of erroneous execution, discussed as follows.

3.5.4.1 Spurious Execution.

It means an automation rule's condition is falsely satisfied when the rule gets triggered, which leads to an action that should not have been issued in the first place. This can be conducted by applying E-DELAY to either the trigger or condition event.

(1) Delaying the trigger event \mathcal{E}_t . Assume $I(\mathcal{E}_t) < I(\mathcal{E}_c)$; due to the delay attack applied to \mathcal{E}_t , $S(\mathcal{E}_c) < S(\mathcal{E}_t)$, and the arrival of \mathcal{E}_c turns $f(C)$ from *false* to *true*. Then, the action of the rule will be spuriously taken when the delayed event \mathcal{E}_t is received.

(2) Delaying the event \mathcal{E}_c . Assume $I(\mathcal{E}_c) < I(\mathcal{E}_t)$. Due to the delay attack applied to \mathcal{E}_c , $S(\mathcal{E}_t) < S(\mathcal{E}_c)$; thus, the action of the rule will be spuriously executed when \mathcal{E}_t is received, as the arrival of \mathcal{E}_c should have turned $f(C)$ from *true* to *false*.

For example, in Figure 3.4(c), the automation rule is as follows: when a user pulls the storm door, if the user is home, the front door is automatically unlocked. As the user leaves home, the event *presence-sensor off* event \mathcal{E}_c is delayed by the attacker, who pulls the storm door to generate the *storm-door pulled* event \mathcal{E}_t ; as a result, $S(\mathcal{E}_t) < S(\mathcal{E}_c)$ and the front door gets unlocked due to the spurious execution.

3.5.4.2 Disabled Execution.

Similarly, *disabled execution* can be conducted by delaying trigger or condition events. For the sake of brevity, we use an example to illustrate it. As shown in Figure 3.4(d), the automation rules are as follows: when the front door is closed, if the lock is

unlocked, the front door is locked automatically. When the user leaves home, he first unlocks and opens the door. However, by delaying the *door-unlocked* event \mathcal{E}_c , when the *door-closed* event \mathcal{E}_t is received, the condition evaluation $f(c)$ is still *false*; as a result, the lock-door action is not taken. We denote the \mathcal{E} and $\hat{\mathcal{E}}$ as two opposite events for the same device and attribute. This attack can be achieved by either delaying the trigger event \mathcal{E}_t until the $f(c)$ is no longer *true* (i.e., $I(\mathcal{E}_t) < I(\hat{\mathcal{E}}_c)$ and $S(\mathcal{E}_t) > S(\hat{\mathcal{E}}_c)$) or delaying the condition event \mathcal{E}_c that turns $f(c)$ as *true* to be later than the trigger event (i.e., $I(\mathcal{E}_t) > I(\mathcal{E}_c)$ and $S(\mathcal{E}_t) < S(\mathcal{E}_c)$).

3.6 Evaluation

In this section, we evaluate our attack primitives on 50 off-the-shelf commercial IoT devices. For each device, we analyze its timeout behavior and conduct 20-trial experiments to find the range of delay on commands and events. We demonstrate all three types of attacks by reproducing them in real-world smart home testbeds.

3.6.1 Testbed Setup

In our experiment, we test 50 popular IoT devices of 8 types as listed in Table 3.1. The popularity of each device is indicated by the number of downloads of its mobile app in the Google Play store. All devices are connected to a home Verizon G3100 WiFi 6 router. Low energy devices that do not have WiFi communication capabilities are connected to the router via their compatible hub/bridge devices. A Raspberry Pi 3B is used to simulate an attacker who is in the same local network as the IoT devices.

Table 3.1. Test results of cloud devices.

No.	Device Type	Device Model	App Install	Long-live Session	Keep-alive Messages			Event Messages		Command Messages	
					Period(s)	Pattern	Timeout(s)	Timeout(s)	Range(s)	Timeout(s)	Range(s)
L1	Smart	Wyze White A19	1M+	Yes	62	fixed	60	60	[60, 60]	60	[60, 60]
L2	Light	Philips Hue white A19	1M+	Yes	120	fixed	60	∞	[60, 180]	21	[21, 21]
P1	Smart Plug	Wyze Plug	1M+	Yes	62	fixed	60	60	[60, 60]	60	[60, 60]
P2		Amazon Plug	50M+	Yes	30	fixed	30	30	[30, 30]	30	[30, 30]
P3		SmartThings WiFi Plug	100M+	Yes	110	on-idle	110	∞	[110, 220]	∞	[110, 220]
P4		SmartThings Zigbee Plug	100M+	Yes	31	on-idle	16	∞	[16, 47]	∞	[16, 47]
P5		SmartLife Gosound Plug	5M+	Yes	60	on-idle	32	∞	[32, 92]	∞	[32, 92]
P6		KASA HS103P2 Plug	1M+	Yes	150	fixed	15	55	[15, 55]	15	[15, 15]
P7		Cync	100K+	Yes	21	on-idle	84	∞	[84, 105]	∞	[84, 105]
P8		iHome iSP6X Plug	100K+	Yes	30	fixed	18	32	[18, 32]	32	[18, 32]
P9		Aqara Plug	50K+	Yes	150	fixed	30	60	[30, 60]	30	[30, 30]
P10		Wemo Mini Plug	1M+	No	-	-	-	52	[52, 52]	15	[15, 15]
P11		Geeni Plug	1M+	No	-	-	-	90	[90, 90]	25	[25, 25]
M1	Motion Sensor	SmartThings Motion	100M+	Yes	31	on-idle	16	∞	[16, 47]	-	-
M2		Philips Hue Motion	1M+	Yes	120	fixed	60	∞	[60, 180]	-	-
M3		Wyze Motion	1M+	Yes	62	fixed	60	60	[60, 60]	-	-
M4		Ring Motion	5M+	Yes	30	fixed	35	∞	[35, 65]	-	-
M5		Nest Motion	5M+	Yes	120	on-idle	60	∞	[60, 180]	-	-
M6		Ecobee Smart Sensor	500K+	Yes	60	on-idle	30	∞	[30, 90]	-	-
M7		SmartLife Sonew Motion	5M+	No	-	-	-	260	[260, 260]	-	-
M8		iHome iSB01 Motion	100K+	No	-	-	-	70	[70, 70]	-	-
M9		Aqara Motion	50K+	Yes	150	fixed	30	60	[30, 60]	-	-
M10		Govee Motion	50K+	Yes	90	fixed	35	55	[35, 55]	-	-
M11		Amazon Echo Flex	50M+	Yes	30	on-idle	30	60	[30, 60]	-	-
C1	Contact Sensor	SmartThings multipurpose	100M+	Yes	31	on-idle	16	∞	[16, 47]	-	-
C2		Wyze Contact	1M+	Yes	62	fixed	60	60	[60, 60]	-	-
C3		Nest Contact	5M+	Yes	120	on-idle	60	∞	[60, 180]	-	-
C4		Ecobee Smartsensor	50K+	Yes	60	on-idle	30	∞	[30, 90]	-	-
C5		SmartLife Towode Contact	5M+	No	-	-	-	130	[130, 130]	-	-
C6		iHome iSB04 Contact	100K+	No	-	-	-	70	[70, 70]	-	-
C7		Aqara Contact	50K+	Yes	150	fixed	30	60	[30, 60]	-	-
C8		Ring Contact	5M+	Yes	30	fixed	35	∞	[35, 65]	-	-
C9		Geeni Door & Window	1M+	No	-	-	-	90	[90, 90]	-	-
C10		Govee door	500K+	Yes	90	fixed	35	55	[35, 55]	-	-
HS1	Home Security	Ring Keypad	5M+	Yes	30	fixed	35	∞	[35, 65]	-	-
HS2		Nest Keypad	5M+	Yes	120	on-idle	60	∞	[60, 180]	-	-
HS3		SimpliSafe Keypad	5M+	Yes	55	fixed	30	20	[20, 20]	-	-
S1	Smart Switch	SmartThings button	100M+	Yes	31	on-idle	16	∞	[16, 47]	-	-
S2		Philips Hue Dimmer	1M+	Yes	120	fixed	60	∞	[60, 180]	-	-
S3		ThirdReality Switch	1K+	Yes	31	on-idle	16	∞	[16, 47]	∞	[16, 47]
S4		Aqara Button	50K+	Yes	150	fixed	30	60	[30, 60]	-	-
CM1	Smart Camera	Arlo Q	1M+	No	-	-	-	60	[60, 60]	-	-
CM2		Wyze Cam Indoor	1M+	Yes	62	fixed	60	60	[60, 60]	-	-
CM3		Ring Doorbell	5M+	Yes	55	fixed	25	31	[29, 31]	-	-
CM4		Foscam R2C	1M+	Yes	150	fixed	45	30	[30, 30]	-	-
CM5		YiHome Cam Indoor	1M+	Yes	45	on-idle	30	∞	[30, 74]	-	-
LC1	Smart Lock	August Pro Gen2	500K+	Yes	70	on-idle	30	58	[30, 58]	58	[30, 58]
LC2		Kwikset Smartcode 913	50K+	Yes	31	on-idle	16	∞	[16, 47]	∞	[16, 47]

3.6.2 Device Behavior Measurement

First we measure all 50 devices in Table 3.1 to profile their message characteristics and timeout behaviors. For a device, we use the ARP spoofing to redirect its traffic to the Raspberry Pi where we setup the TCP proxy as shown in Figure 3.3. Then, we trigger its commands and events manually to analyze its characteristics and attempt to make different delays to explore its timeout behavior. Note that the measurement is supposed to be conducted by attackers on their own devices and is prior to performing attacks. To launch the attack in a target smart home, attackers only need to redirect the traffic of the target device and leave others' traffic unchanged, which is shown to be feasible on a wide variety of devices without affecting network stability [116].

3.6.2.1 Target Events and Commands Recognition

To delay a specific event or command message, first, we need to locate the packet that carries the target event or command. In our work, we use the cloud server's domain name and length of TLS record as the characteristic to identify messages. We manually trigger events and commands and check the corresponding network packets that are received by the raspberry pi. If packets with the same length can always be found right after events or commands, we take it as the length of the target message. Then, we confirm this by delaying messages with the same length for a short time of 5 seconds and check whether the device's status update is also delayed by the same amount of time.

3.6.2.2 Parameter Measuring Method

Also, to check whether the delayed events are accepted by IoT servers, we install automation rules that use the delayed events as triggers to show visible actions on other

devices (e.g., turning on a light). For each device, we first set the proxy program to the pass-through mode to allow any packet pass without delay and keep the device being idle to observe the device’s keep-alive period clearly. After that, we conduct a 20-trials experiment for delaying each of the keep-alive, event, and command messages. We have a two-minute interval between We log all TCP segments that flow through our proxy, and analyze them to get the three parameters by following the procedures in Section 3.4.3

3.6.3 Device Timeout Measurement Results

3.6.3.1 Results of Cloud-based IoT Devices

We first evaluate delays on connections between IoT devices and cloud-based IoT servers. As shown in Table 3.1, we present the parameters of timeout behavior for cloud-based device, which include keep-alive message’s period and pattern, and timeout threshold for all three types of messages. A cell marked as ‘ ∞ ’ means we only observe timeouts caused by keep-alive messages among all experiment trials. This indicates that the device has equal or no timeout for event and command messages, and the timeout is solely triggered by the delay of keep-alive messages.

From Table 3.1, we can see event messages of all tested devices can be delayed for longer than 30 seconds except the SimpliSafe keypad (HS3), which is enough to cause severe consequences if applied on safety-sensitive events such as smoke alerts. In particular, some WiFi-enabled sensor devices (e.g., M7 and C5) do not use long-live sessions and show delay time windows longer than 2 minutes. Due to the lack of keep-alive message, session timeout that is caused by delaying event messages will not be noticed by the cloud server because the session is never established. Even after

the session resumes, this anomaly is not reported to the cloud server. Here, we use some example devices to illustrate the procedure to measure the results in Table 2.3.

For devices that are using the SmartThings hub, we start the by monitoring the hub's network traffic with no device attached. From the long-live TLS session between the SmartThings hub and the SmartThings cloud server, we find that they exchange messages with fixed lengths of 40 bytes (hub to cloud) and 42 bytes (cloud to hub) in every 31 seconds. When an event or command message happens, the next keep-alive messages are postponed accordingly to 31 seconds later. When we attempt to delay keep-alive messages, we observe a constant timeout threshold of 16 seconds. After that, we manually trigger and delay event and command messages right after keep-alive messages and find the session timeout still happens 16 seconds after the next keep-alive message. This implies that the SmartThings session timeout is solely triggered by the delay of keep-alive messages. We confirm that by randomly triggering and delaying event and command messages. The timeout always happen 16 seconds after a keep-alive message. For Philips Hue Light bulb (L2) and dimmer switch (S2) that are using the Philips Hue bridge, we observe a fixed keep-alive period of 120-second period, which is independent from the event and command messages. During the 20-trials experiment, delay of command messages consistently causes session timeout after 21 seconds. While, in trials of event messages delays, timeout always happens 60 seconds after a keep-alive message, which means there is no dedicated timeout for event messages. In summary, Philip Hue bridge associated devices can be delayed in the range of $[60s, 180s]$ depends on the interval between the event message and the next keep-alive message.

To verify the collected parameters in Table 3.1, we also conduct a verification test. For each testing device, we randomly trigger and delay its messages and predict the timeout occurrence according to the collected parameters. At 2 seconds before the

Table 3.2. Measurement results of local.

Label	Device Model	Event Messages	
		Max (s)	Min (s)
L2	Philips Hue white A19	420	223
L3	LIFX Mini White A19	412	179
P8	iHome iSP6X Plug	341	115
M2	Philips Hue Motion	290	67
M6	Ecobee Smart Sensor	679	337
M9	Aqara Motion	1310	421
C4	Ecobee Smartsensor	854	211
C7	Aqara Contact	1345	683
S2	Philips Hue Dimmer	275	170
S4	Aqara Button	1453	302
S5	Insignia Garage Controller	343	196
CM1	Arlo Q	200	129

predicted timeout, we end the delay and release the messages that are on hold. The results show that not only the timeout is 100% avoided, but also the delayed messages are accepted by the device or cloud server.

3.6.3.2 Results of Local-based IoT Devices

For measuring devices' delays with local IoT servers, we choose Apple's HomeKit as the representative IoT server and connect compatible devices, as listed in Table 3.2, to a HomePod speaker using the Home+ mobile application [118] from Apple's App Store. Even though these devices are using long-live sessions to communicate with the HomePod speaker, we could not find a single keep-alive message. If there is no transmission of events or commands, then the connection simply stays idle. Moreover, for each event message sent by devices, the HomePod speaker does not have any response. On the other hand, command messages issued by the HomePod speaker have a fixed timeout of 10 seconds for the receiving device to respond, which is consistent with the specifications of HomeKit Accessory Protocol (HAP) [119]. As a result, the delay of event messages will only be interrupted by sporadic commands and

status polling messages. Otherwise, the delay of events could be unlimited. During our 10-trial experiment, device status polling messages appear on the voice activation of the HomePod speaker and mobile app operations rather than in any explicit period or pattern.

3.6.3.3 Interesting Findings

Below, we present several interesting findings that reveal IoT system design flaws:

Finding 1: Lack of Device Offline Reporting. Although we reasonably assume the “device offline” alarms can be raised when a device’s connection with the IoT server is terminated because of timeout, it is not always the case, according to our experiment. Some long-live connection devices such as Nest security system (HS2), and most on-demand devices (M7, C5, and C9) do not raise a “device offline” alarm immediately after the connection timeout. For Nest security system, the base station will try to reconnect the server; as long as the reconnecting is successful within three attempts, no alert will be raised. Sensors with on-demand connections (e.g., M7 and C9) do not have offline detection. No matter how many times they are delayed to timeout, these devices do not report the history offline events.

Finding 2: Duplicated Connections. On delaying event messages from devices, even after a timeout occurs and connection closes between a device and our Raspberry Pi, the connection on the other side with the IoT cloud server can still be maintained, which gives the server an illusion that the device is still online and prevents the “device offline” alarms from being raised, temporarily. After that, the disconnected device will try to reconnect to the server. Surprisingly, even after a new connection has been established, cloud servers still maintain the duplicated half-open connection and do not raise any alerts. Moreover, as long as the new connection is established before we close the half-open connection, the server will never raise the “device offline”

alarm. This implies that whenever an IoT device raises timeout and terminates the connection because of delay, attackers can still avoid device-offline alarms by maintaining the half-open connection until the device reconnects to the IoT server. This flaw can be exploited to further extend the time of delay.

Finding 3: Unidirectional Liveness Checking. Although most of the tested devices adopt long-live sessions with keep-alive messages, keep-alive requests are always initiated by IoT devices, while IoT servers only passively receive and reply to the keep-alive messages. Even when a device’s events are being delayed by attackers, from the perspective of an IoT server, it seems as if the device is just idle. This offers opportunities for attackers to make longer delays on events, until the IoT server can proactively send command messages towards the IoT device.

3.6.4 Proof-of-Concept Attacks

We conduct end-to-end experiments in real-world users’ homes with automation rules that are collected from IoT user forums (as listed in Table 3.3). We simulate the attacking device using a raspberry pi that is connected to the participant’s home network. We have obtained our university’s IRB approval for the experiment involving real-world participants.

3.6.4.1 State-Update Delay Attack

State-update delay attacks are applied on devices’ event messages. For Case 1 in Table 3.3, we deploy an Amazon Echo Speaker and a Ring security base station with a contact sensor on the front door. We add a routine that issues voice alerts when the front door is opened. We use the Nmap [120] to discover the Ring base station and hijack it’s TCP connection using ARP spoofing. We can locate the TCP connect

with a cloud domain at “*.prd.ring.solution” as carrying events. By referring to our measurement results, we can accurately identify messages of keep-alive (48 bytes) and contact sensor (986 bytes) and delay them for up to 60 seconds. To be noticed, Ring base station’s backup cellular network connection, which can defeat the jamming attacks, are not activated in our experiment since the base station is never aware of the attack.

3.6.4.2 Action Delay Attack

We take the Case 3 in Table 3.3 to demonstrate the action delay attack. We install an August Smart Lock along with the Ring security system into Amazon’s Alexa platform and install an automation rule (also called routine) to lock the front door upon door closing. Using a similar method as the state-update delay attack demonstration, we identify the August Lock’s connection and message packets with the cloud server. With only one day’s event, we can reasonably infer this automation rule by observing the behavior pattern between the lock’s locking commands and the events of door closing. We can proactively verify this hypothesis by adding small delays of five seconds on events of front door closing, and check whether the “door locking” actions are also delayed by five seconds. After that, on detecting “door closed” events, we can delay the following locking command for a period between 30 seconds to 58 seconds. By combining the E-DELAY on the contact sensor’s event, the length of the attacking window can be extended to be at least 60 seconds, which is enough for burglars to unhurriedly break into a victim’s house after the victim leaves home. Moreover, for Case 4, we have another interesting finding that delayed event messages from the Ring security system, even without causing the base station’s disconnection, will be discarded by the Alexa if the delay time is longer than 30 seconds. And no notification

or alerts are raised about this anomaly. This makes it possible for attackers to disable the execution of safety critical routines (e.g., turn off a electric heater) forever.

3.6.4.3 Erroneous Execution Attack

Table 3.3. Cases of event delay attacks.

Case	Type	Trigger	Condition	Action	Consequence	Reference
Case1	State-Update	<i>Front door opened</i>	-	Voice notification	late burglary alerts	[121]
Case2	Delay	<i>Motion active</i>	-	Mobile notification	late burglary alerts	[122]
Case3	Action	<i>Front door closed</i>	-	<i>Lock the door</i>	door not locked in time	[123]
Case4	Delay	<i>Home security system armed</i>	-	<i>Turn off heater</i>	heater not turned off	[123]
Case5	Erroneous Execution	<i>Front door unlocked</i>	Entrance motion inactive	Disarm security system	security system disarmed	[124]
Case6		<i>Bedroom motion active</i>	Bedroom door closed	Turn on bedroom heater	heater maliciously turned on	[125]
Case7		<i>Study motion active</i>	Study door closed	Open the study window	window maliciously opened	[126]
Case8		Storm door opened	<i>Presence on</i>	Unlock the interior door	door maliciously unlocked	[127]
Case9		Presence away	<i>Front door open</i>	Send text message	door open notification muted	[128]
Case10		Front door closed	<i>Front door unlocked</i>	Lock the front door	door not locked	[129]
Case11		Presence away	<i>Heater is on</i>	Turn off Heater	heater not turn off	[130]

Rules in these cases are collected from smart home user forums as referred in the last column. In each case, the event being delayed is highlighted in italic font.

For Erroneous Execution Attacks, we collect seven real-world conditional automation rules from the IoT user forum, as Case 5 to Case 8 listed in Table 3.3, and reproduce them using our testing devices. For the subtype attack of Spurious Execution, we reproduce four rules that can be manipulated to make safety-critical actions such as disarming a home security system and unlocking the front door. We reproduce the Case 8 on SmartThings platform with a SmartThings presence sensor, August Smart Lock, and SmartLife contact sensor. Whenever the attacker observes the sequential events of ‘storm door opened’, ‘interior door locked’, and ‘storm door closed’, he can delay the following ‘presence off’ event by 40 seconds in average, during which attackers can break in by simply pulling the storm door.

For the disabled execution attack (Case 9 to Case 11), we reproduce Case 10 in Table 3.3 with an August Smart Lock and a SmartThings contact sensor, both of which are installed on the same door. There is a clear event sequence here: ‘door unlocked’, ‘door opened’, ‘door closed’, and ‘door locked’. Whenever we see the event

of ‘door unlocked’ from the connection between the August Connect bridge to the August server, we hold the event until we receive and forward the ‘door closed’ event through another connection between the SmartThings hub and SmartThings cloud server. According to our measurement, the delay time window has a length of at least 16 seconds, which is long enough to cover the interval between the door unlocking and closing. We confirm that the door has been left unlocked for the entire day while the participant was away.

3.7 Delay Attack Defense

3.7.1 Requiring Message ACK and Shortening ACK Timeout

As analyzed in Section 3.4, the delay time range is closely related to the configuration of timeout for acknowledgement of commands and events. The measurement results in Table 3.1 also indicate that a shorter message timeout value can effectively restrict the attack primitives and reduce the length of the attack window. According to our measurement, existing IoT devices have an overlong timeout for event message acknowledgement, which is usually longer than 30 seconds; we suggest shortening it significantly. The HomeKit Accessory protocol does not require acknowledgement of IoT event messages, which leaves an almost unbounded window for our attacks, showing a serious design flaw that should be fixed. The current MQTT protocol does not mandate timeout for an acknowledgement (PUBACK message), but we consider it to be a critical requirement for a secure MQTT implementation.

3.7.2 Timestamp Checking

Currently, commands and events are accepted if they are generated a long time ago, which shows another critical design flaw of existing IoT protocol stack. We suggest

it should be enforced in standard protocols (such as MQTT) by adding timestamps as message fields and checking them at the receiver's side. There should be a mechanism that allows devices and servers to determine whether the received messages are outdated or not.

3.7.3 Two-way Liveness Checking

As discussed in Section 2.6, lacking server-initiated device-liveness checking offers attackers the opportunity to delay event messages for a longer period. As a remedy, IoT servers can also send keep-alive requests to check IoT devices' liveness with a timeout value shorter than the device's idle timeout value. As a result, on receiving server-initiated keep-alive requests, attackers have to stop delaying event messages to allow the keep-alive responses to be sent timely, which significantly constrains the range that events can be delayed.

3.7.4 Summary.

Each countermeasure aforementioned has its limitations. Besides, all of them require modifications of IoT protocol and firmware/software. Thus, the path to handle the attacks described in this work is not easy. How to handle the attacks without updating the firmware of numerous IoT devices is an interesting research question.

3.8 Literature Review

3.8.1 Device Blind Attacks

IoT attacks and defenses are actively studied [7, 60, 131, 55, 59, 8]. There have been many attacks that exploit IoT device and server vulnerabilities. The attacks

presented by Hariri *et al.* [106] discard IoT event messages from an IoT device’s long-live session. They find that, due to certain implementation vulnerabilities, as long as the keep-alive messages are transmitted normally, devices would not raise device-offline alarms, even when some messages are discarded. In [132], OConnor *et al.* explore the possibility of blocking an IoT device’s events that are transmitted through on-demand connections. They find that, by blocking the device’s event message, the event gets lost permanently. Even after the connection is resumed, the missing event will not be retransmitted.

Attacks proposed by these works are only applicable to certain devices that have implementation vulnerabilities. For example, attacks in [106] do not take the TLS record sequence number into consideration and are only applicable to devices with defective TLS implementations. As admitted by the authors, on devices with a secure implementation, their attack will result in connection termination and raising the device-offline alarms immediately. Attacks in [132] only work on devices that use on-demand connections or have their heartbeat and event messages transmitted through different channels. According to our evaluation, most IoT devices use long-live connections and have their heartbeat and normal event messages transmitted through the same TCP connection. Unlike these works, our attacks do not depend on any implementation vulnerabilities and are widely applicable to existing WiFi IoT devices.

3.8.2 WiFi Jamming Attacks

WiFi jamming attacks have been widely investigated and evaluated. Traditional proactive jamming attacks continuously block the radio frequency channel by sending junk data constantly or randomly. Since devices are using the identical communication channel when connected to the same WiFi access point, this method will

inevitably disconnect all devices in a smart home and hence it can be easily noticed by users. The state-of-art jamming attacking methods can reactively jam the radio frequency bursts to destroy packets transmission [133, 134, 135, 136]. The reactive jamming attacks, although reduce the risk of exposure, are more challenging to implement due to the strict requirement of frame detecting and jamming to be finished within hundreds of nanoseconds. As a result, most of these attacks requires special dedicated hardware such as software-defined radio (SDR) [137]. Jamming attacks are also inferior to our delay attacks in terms of effects. Unlike our delay attack that can accurately control the time of delay to facilitate the erroneous attack, jamming attacks can only wait for the device to retransmit the lost messages.

3.8.3 Other IoT Vulnerabilities

There are attacks that exploit IoT network protocols and IoT platforms. Jia *et al.* [9] systematically study the MQTT protocol and reveal critical design flaws. They demonstrate that MQTT can be exploited to either maliciously control victim devices or impersonate devices to send spoofed events. However, their attack is based on the assumption that attackers can access user accounts on the device vendor’s cloud, which limits the impact of the attack to certain situations such as device sharing. Yuan *et al.* [138] investigate the security problem caused by an IoT device’s inter-platform integration and delegation. They find that existing IoT devices have chained trust on IoT cloud platforms. However, the revocation of delegation between two platforms does not break the trust chain accordingly, leaving the devices exploitable by unauthorized users. Our work studies a very different topic—IoT timeout behaviors—and builds attacks that exploit these behaviors.

3.9 Chapter Summary

This work studies an important but largely omitted topic—IoT timeout behaviors. Due to the diverse network conditions of smart homes, IoT network protocols have no choice but to allow delays when IoT event and command messages are transmitted. This study finds that the delay limits are loosely defined and lack standardization. As a result, an attacker who leverages simple attack primitives can delay IoT messages for tens of seconds without triggering alarms in any layers of the existing IoT network protocol stack. With the very simple attack primitives, we constructed attacks that can delay critical state updates or maliciously disable/incure/delay/override automation actions. The evaluation on 50 popular IoT devices shows that the attacks are widely applicable to IoT devices, and we demonstrate a variety of Proof-of-Concept attacks. The study reveals critical design flaws, regarding timeout behaviors, in existing IoT network protocol stack, and we hope this study can draw attentions of IoT vendors and platform designers.

Chapter 4

EFFECTIVE DETECTION OF AUDIO ADVERSARIAL EXAMPLE

Adversarial examples (AEs) are crafted by adding human-imperceptible perturbations to inputs such that a machine-learning based classifier incorrectly labels them. They have become a severe threat to the trustworthiness of machine learning. While AEs in the image domain have been well studied, audio AEs are less investigated. Recently, multiple techniques are proposed to generate audio AEs, which makes countermeasures against them an urgent task. Our experiments show that, given an AE, the transcription results by different Automatic Speech Recognition (ASR) systems differ significantly, as they use different architectures, parameters, and training datasets. Inspired by Multiversion Programming, we propose a novel audio AE detection approach, which utilizes multiple off-the-shelf ASR systems to determine whether an audio input is an AE. The evaluation shows that the detection achieves accuracies over 98.6%.

4.1 Introduction

Automatic Speech Recognition (ASR) is a system that converts speech to text. ASR has been studied intensively for decades. Many core technologies, such as gaussian mixture models and hidden Markov models, were developed. In particular, recent advances [139] based on deep neural network (DNN) have improved the accuracy significantly. DNN-based speech recognition thus has become the mainstream technique

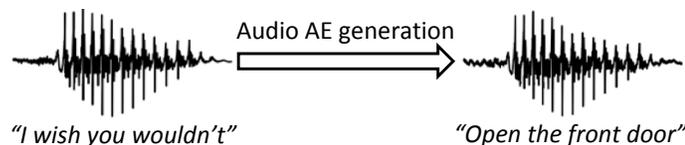


Figure 4.1. An audio AE.

in ASR systems. Companies such as Google, Apple and Amazon have widely adopted DNN-based ASRs for interaction with IoT devices, smart phones and cars. Gartner [140] estimates that 75% of American households will have at least one smart voice-enabled speaker by 2020.

Despite the great improvement in accuracy, recent studies [141, 142] show that DNN is vulnerable to adversarial examples. An *adversarial example* (AE) x' is a mix of a host sample x and a carefully crafted, human-imperceptible perturbation δ such that a DNN will assign different labels to x' and x . Figure 4.1 presents an audio AE, which sounds as in the text shown on the left to human, but is transcribed by an ASR system into a completely different text as shown on the right.

Several techniques have been proposed to generate audio AEs, and novel attack vectors based on audio AEs have been demonstrated [143]. There are two *state-of-the-art* audio AE generation methods: (1) **White-box attacks:** Carlini et al. proposed an optimization based method to convert an audio to an AE that transcribes to an attacker-designated phrase [144]. It is classified as white-box attacks because the target system’s detailed internal architecture and parameters are required to perform the attack. (2) **Black-box attacks:** Taori et al. [145] combined the genetic algorithm [146] and gradient estimation to generate AEs. It does not require the knowledge of the ASR’s internal parameters, but imposes a larger perturbation (94.6% similarity on average between an AE and its original audio, compared to 99.9% in [144]). The rapid development of audio AE generation methods makes counter-

measures against them an urgent and important problem. The goal of our work is to detect audio AEs.

Existing work on audio AE detection is rare and limited. Yang et al. [147] hypothesized that AEs are fragile: given an audio AE, if it is cut into two sections, which are then transcribed section by section, then the phrase by splicing the two sectional results is very different from the result if the AE is transcribed as a whole. However, as admitted by the authors, this method cannot handle “adaptive attacks”, which may evade the detection by embedding a malicious command into one section alone. Rajaratnam et al. [148] proposed detection based on audio pre-processing methods. Yet, if an attacker knows the detection details, he can take the pre-processing effect into account when designing the loss function used for generating AEs; this attack approach has been well demonstrated in [149]. An effective audio AE detection method that can handle adaptive attacks is missing.

Our key observation is that existing ASRs are diverse with regard to architectures, parameters and training processes. Our hypothesis is that an AE that is effective on one ASR system is highly likely to fail on another, which is verified by our experiments. How to generate transferable audio AEs that can fool multiple heterogenous ASR systems is still an open question [144] (discussed in Section 4.2). This inspires us to borrow the idea of *multiversion programming* (MVP) [150], a method in software engineering where multiple functionally equivalent programs are independently developed from the same specification, such that an exploit that compromises one of them is ineffective on other programs. We thus propose to run multiple ASR systems in parallel, and an input is determined as an AE if the ASR systems generate very dissimilar transcription results.

Our contributions are as follows. (1) We empirically investigate the transferability of audio AEs and analyze the reasons behind the poor transferability. (2) We propose

a novel audio AE detection approach inspired by multiversion programming, which achieves accuracy rates of over 98.6%. (3) The detection method dramatically reduces the flexibility of the adversary, in that audio AEs cannot succeed unless the host text is highly similar to the malicious command.

4.2 Transferability

There are two types of AEs: *non-targeted* AEs and *targeted* AEs. A *non-targeted* AE is considered successful as long as it is classified as a label different from the label for the host sample, while a *targeted* AE is successful only if it is classified as a label desired by the attacker. In the context of audio based human-computer interaction, a non-targeted AE is not very useful, as it cannot make the ASR system issue an attacker-desired command. It explains why state-of-the-art audio AE generation methods all generate targeted AEs [144, 145]. Thus, our work focuses on targeted audio AEs, although the proposed detection approach should be effective in detecting non-targeted AEs as well.

An intriguing property of AEs in the image domain is the existence of transferable adversarial examples. That is, an image AE crafted to mislead a specific model M can also fool a different model M' [142, 141, 151]. By exploiting this property, Papernot et al. [152] proposed a reservoir-sampling based approach to generate transferable non-targeted image AEs and successfully launch black-box attacks against both image classification systems from Amazon and Google. But a recent study [153] points out that the transferability property does not hold in some scenarios. The experiment shows a failure of AE’s transferability between a linear model and a quadratic model. For targeted image AEs, Liu et al. [154] proposed an ensemble-based approach to craft

AEs that could transfer to ResNet models, VGG and GoogleNet with success rates of 40%, 24% and 11% respectively. Thus, an image AE can fool multiple models [155].

To the best of our knowledge, no systematic approaches are available for generating transferable audio AEs. Carlini et al. [144] found that the attack method derived from Fast Gradient Sign Method [142] in image domain is not effective for audio AEs because of a large degree of non-linearity in ASRs, and further stated that the transferability of audio AEs is an *open question*.

CommanderSong [143] represents a recent advance in audio AE generation, in that it is able to generate AEs that can fool an ASR system in the presences of background noise. This work also slightly explored transferability of audio AEs (as we discussed in Section 5.3). Specifically, to create AEs that can transfer from “Kaldi to DeepSpeech” (both *Kaldi* and *DeepSpeech* are open source ASR systems, and *Kaldi* is the target ASR system of CommanderSong), a two-iteration recursive AE generation method is described in CommanderSong: an AE generated by CommanderSong, embedding a malicious command c and able to fool *Kaldi*, is used as a host sample in the second iteration of AE generation using the method [144], which targets *DeepSpeech* v0.1.0 and embeds the same command c . We followed this two-iteration recursive AE generation method to generate AEs, but our experiment results [156] showed that the generated AEs could only fool *DeepSpeech* but not *Kaldi*. That is, AEs generated using this method are not transferable.

Furthermore, we adapted the two-iteration AE generation method by concatenating the two aforementioned state-of-the-art attack methods [144] and [145] targeting *DeepSpeech* v0.1.0 and v0.1.1, respectively, expecting to generate AEs that can fool both *DeepSpeech* v0.1.0 and v0.1.1. But none of the generated AEs showed transferability [156].

Moreover, by changing the value of “`--frame-subsampling-factor`” from 1 to 3, which is a parameter configuration of the *Kaldi* model, we derived a variant of *Kaldi*. The AEs generated by CommanderSong did not show transferability on the variant, even given the fact that the variant was only slightly modified from the model targeted by CommanderSong. Here, we clarify that CommanderSong did not claim their AEs could transfer across the *Kaldi* variants.

Based on our detailed literature review and empirical study, we find that so far there are no systematic AE generation methods that can generate transferable audio AEs effective across two ASR systems, not to mention three or more. This is consistent with the statement by Carlini et al. [144] that transferability of audio AEs is an open question,

4.3 Diverse ASRs

Yu and Li [157] summarized the recent progress in deep-learning based ASR acoustic models, where both recurrent neural network (RNN) and convolutional neural network (CNN) come to play as parts of deep neural networks. Standard RNN could capture sequence history in its internal states, but can only be effective for short-range sequence due to its intrinsic issue of exploding and vanishing gradients. This issue is resolved by the introduction of long short-term memory (LSTM) RNN [158], which outperforms RNNs on a variety of ASR tasks [159, 160, 161]. As to CNNs, its inherent translational invariability facilitates the exploitation of variable-length contextual information in input speech. The first CNN model proposed for ASRs is time delay neural network [162] that applies multiple CNN layers. Later, several studies [163, 164, 165] combine CNN and Hidden-Markov Model to create hybrid models that

are more robust against vocal-tract-length variability between different speakers. But there is *no* single uniform structure used across all ASRs.

In addition to several well-known ASR systems, multiple companies have independently developed their own ASRs, such as *Google Now*, *Apple Siri* and *Microsoft Cortana*. In our system, we use *DeepSpeech*, *Google Cloud Speech*, and *Amazon Transcribe*. *DeepSpeech* is an end-to-end speech recognition software open-sourced by Mozilla. Its first version [166] uses a five-layers neural network where the fourth layer is a RNN layer, while the second version [167] contains a mix of CNN and RNN layers. Our experiments used the first version, as the second version was not publicly available at the time of our experiment. Unlike *DeepSpeech*, the DNN behind Google Cloud Speech is a LSTM-based RNN according to the source [168]. Each memory block in Google’s LSTM network [160] is a standard LSTM memory block with an added recurrent projection layer. This design enables the model’s memory to be increased independently from the output layer and recurrent connections. No public information about the internal details of *Amazon Transcribe* is available.

In short, existing ASR systems are diverse with regard to architectures, parameters and training datasets. Compared to opensourced systems, propriety ASR systems provide little information that can be exploited by attackers. Given the diversity of ASR systems (and proprietary networks), it is unclear how to propose a generic AE generation method that can simultaneously mislead all of them.

4.4 System Design

In software engineering, *multi-version programming* (MVP) is an approach that independently develops multiple software programs based on the same specification, such that an exploit that compromises one program probably fails on other programs. This

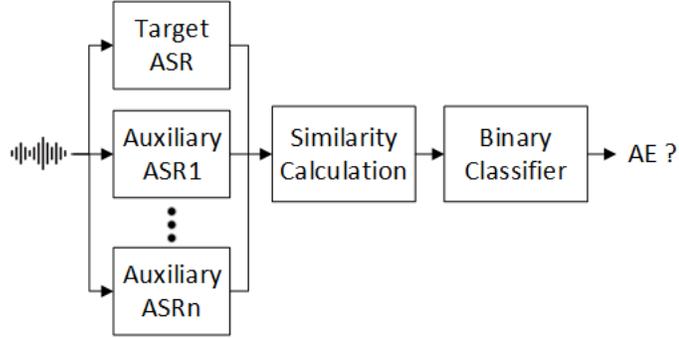


Figure 4.2. Overview of the detection system.

Table 4.1. Recognition results of an AE by multiple ASRs.

ASR	Transcribed Text
DeepSpeech v0.1.0	<i>A sight for sore eyes</i>
DeepSpeech v0.1.1	<i>I wish you live</i>
Google Cloud Speech	<i>I wish you wouldn't.</i>
Amazon Transcribe	<i>I wish you wouldn't.</i>

The host transcription is “I wish you wouldn’t”, while the embedded text is “a sight for sore eyes”.

inspires us to propose a system design that runs multiple ASRs in parallel to detect audio AEs as shown in Figure 4.2, where the *target* ASR is the system targeted by the adversary (e.g., the speech recognition system at a smart home) and *auxiliary* ASRs are models different from the target ASR.

The intuition behind this design is that different ASRs can be regarded as “independently developed programs” in MVP. Since they follow the same specification, that is, to covert audios into texts, given a benign sample, they should output very similar recognition results. On the other hand, an audio AE can be regarded as an “exploit”, and cannot fool all ASRs as verified in Section 4.2. Thus, the recognition result by the target ASR differs *significantly* from those by the auxiliary ASRs. Table 4.1 shows a typical example, which can only fool one ASR and fails on others.

The system works as follows: 1) the target and auxiliary ASRs simultaneously conduct speech-to-text conversion, 2) then the transcriptions are used to calculate similarity score(s), which are passed into a binary classifier to determine whether the input audio is adversarial.

Similarity scores are calculated in two steps. First, each transcription is converted into its phonetic-encoding representation. Phonetic encoding converts a word to the representation of its pronunciation [169]. This helps handle variations between ASRs, as they may output different words for similar pronunciations. The validity of using phonetic encoding will be demonstrated in Section 4.5.4. Second, for each auxiliary ASR, a similarity score is calculated to measure the similarity between the text recognized by the target ASR and that by the auxiliary ASR. We adopt the Jaro-Winkler distance method [170] to calculate the similarity score (see Section 4.5.4). It gives a value between 0 and 1, where 0 indicates totally dissimilar and 1 very similar.

4.5 Evaluation

We evaluate our system on its accuracy and robustness. We first describe the experiment setup (Section 4.5.1) and discuss the dataset used in our evaluation (Section 4.5.2). Next, we investigate the feasibility of our idea (Section 4.5.3), and examine different methods for calculating similarity scores and select the one that provides the best results (Section 4.5.4). After that, we evaluate the accuracy of our system when one auxiliary ASR is used (Section 4.5.5) and more than one auxiliary ASR is used (Section 4.5.6). Finally, we evaluate the robustness of our system against AEs generated by unseen attack methods (Section 4.5.7)

Table 4.2. Datasets..

Dataset Name		# of Samples
<i>Benign</i>		1125
<i>AE</i>	White-box AEs	1025
	Black-box AEs	100

4.5.1 Experimental Settings

Our experiments were performed on a 64-bit Linux machine with an Intel(R) Core(TM) i9-7980XE CPU @ 2.60GHz, NVIDIA GeForce GTX 1080 Ti, and 32GB DDR4-RAM.

Target Model. We select *DeepSpeech v0.1.0*, called DS0 [171] as the target model. The main reason we select DS0 is that its model architecture and parameters are publicly available—making it possible to generate white-box AEs [145]. But our system should also work if any other ASR model is selected as the target model.

Auxiliary Models. There are three auxiliary models: (1) *Google Cloud Speech*, called GCS [172], (2) *Amazon Transcribe*, called AT [173], and (3) *DeepSpeech v0.1.1*, called DS1 [171]. The first two auxiliary ASRs are on-line services, while the last one runs locally with an officially pre-trained model.

We use $X+\{Y_1, \dots, Y_n\}$ to denote a system using X as the target model and Y_i ($\forall i \in \{1, \dots, n\}$) as the auxiliary models. When only one auxiliary model is included, $n = 1$.

4.5.2 Dataset Preparation

We consider two audio AE generation techniques: white-box based [144] and black-box based [145] methods, and build two datasets: a *Benign* dataset and an *AE* dataset, each of which contains 1125 audio samples, as shown in Table 4.2. The audio

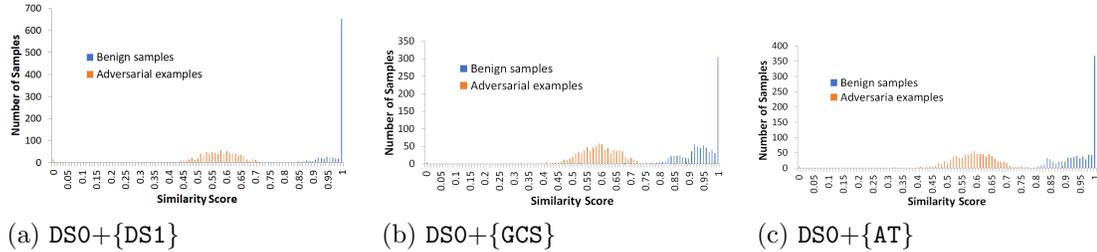


Figure 4.3. Similarity scores of transcriptions generated by DS0 and one of DS1, GCS and AT..

samples of the *Benign* dataset are randomly selected from the *dev_clean* dataset of LibriSpeech [174].

The *AE* dataset is composed of the following two parts: (1) 1025 white-box AEs, including 990 AEs provided by [144] and 35 ones created by us; (2) 100 black-box AEs constructed by selecting the first 100 audio files in *Common Voice* dataset [175] as host speeches, each of which is converted into an AE by applying the black-box approach [145], such that the AE’s transcription contains only two words. All the AEs can successfully fool the target model DS0.

For each dataset, 80% of its samples are used for training, and the remaining 20% for testing.

4.5.3 Feasibility Analysis

To detect whether an audio is an AE, the basic idea is to compare its transcriptions generated from different ASRs. Our intuition is that the transcriptions generated from different ASRs for a benign sample should be similar. On the contrary, if the transcriptions generated from different ASRs are dissimilar, the input audio is likely to be adversarial.

To this end, for each audio sample in the training dataset, we use the *phonetic encoding* technique and *Jarro-Winkler* distance method to calculate the similarity

score between the transcriptions generated by DS0 and one of the three auxiliary models (i.e., DS1, GCS and AT). Figure 4.3 confirms our intuition visually by comparing the similarity scores for benign samples and AEs—the similarity scores for AEs and those for benign samples form two distinguishable clusters.

4.5.4 Comparison of Different Similarity Measurement Methods

Many methods can calculate the similarity score of two strings, such as *Jaccard index* [176], *Cosine similarity*, and *edit distance* (e.g., *Jaro-Winkler* [170]).

To analyze a transcription, some previous works first apply the phonetic encoding technique to convert the transcription to its phonetic-encoding representation (where each word is converted to the representation of its pronunciation), and then perform further analysis on the phonetic-encoding representation [169]. Thus, here we come with three other methods to measure the similarity of two strings, denoted as *PE-Jaccard*, *PE-Cosine* and *PE-JaroWinkler*, representing that the phonetic encoding technique is first applied on each transcription and then *Jaccard index*, *Cosine similarity* and *Jaro-Winkler* are applied later to computing the similarity score of two transcriptions, respectively.

Each system uses a SVM classifier. We examine the aforementioned six methods and evaluate the system performance with respect to each of them. The results are shown in Table 4.3. It can be observed that when *PE-JaroWinkler* is adopted, all the systems achieve the best performance: for example, the detection accuracy of the system DS0+{DS1, AT}—using DS0 as the target model and both DS1 and AT as the auxiliary models—is 99.78% when *PE-JaroWinkler* is used. Thus, in our later experiments *PE-JaroWinkler* is adopted to measure the similarity between transcriptions.

4.5.5 Single-Auxiliary-Model System

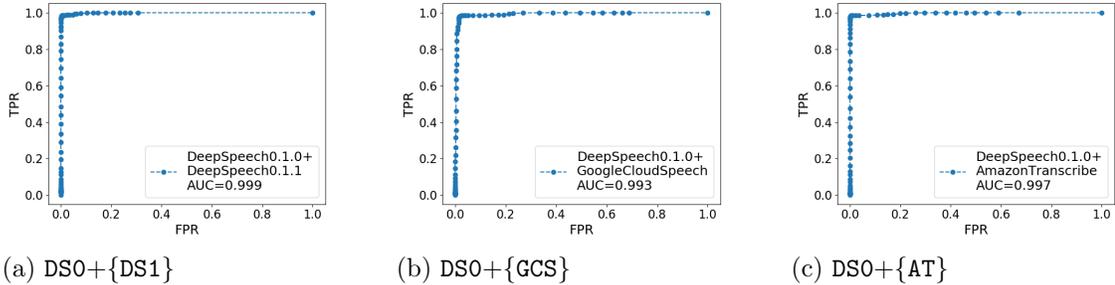


Figure 4.4. The ROC curves of three single-auxiliary-model systems..

A *single-auxiliary-model* system contains a target model and *only one* auxiliary model. As three ASRs are selected as the auxiliary model, we have three different single-auxiliary-model system, denoted as $DSO+\{DS1\}$, $DSO+\{GCS\}$, and $DSO+\{AT\}$, using DS1, GCS, and AT as the auxiliary model, respectively.

A single-auxiliary-model system computes a similarity score for an input audio. If the similarity score is lower than a threshold T , the corresponding audio is detected as adversarial, otherwise benign. We use the Receiver Operating Characteristic (ROC) curve to select the threshold T , such that the sum of TPR (true positive rate) and TNR (true negative rate) is maximized.

The training results are shown in Table 4.4 and the corresponding ROC curves are plotted in Figure 4.4.

We show testing results in Table 4.5. All of the three single-auxiliary-model systems achieve TPR higher than 98.6% and FPR (false positive rate) lower than 1.34%. We further investigate the results, and have the following interesting findings: (1) all 100 black-box AEs are correctly detected; and (2) for one FP (false positive) case, the transcriptions recognized by DSO and GCS have the same meaning, but are in different text formats. The recognition of DSO is the phrase “*SEVENTEEN SEVENTEEN*”,

while that of **GCS** is the digit “1717”. The reason is that the alphabet model of **DS0** does not include digits; as a result, any digit is represented as a word. However, if the alphabet model is integrated into **DS0**, such a FP case could be eliminated.

4.5.6 Multiple-Auxiliary-Models System

A *multiple-auxiliary-model* system contains a target model and *more than one* auxiliary models. We design four different multiple-auxiliary-model systems, denoted as **DS0**+{**DS1**, **GCS**, **AT**}, **DS0**+{**DS1**, **GCS**}, **DS0**+{**DS1**, **AT**}, and **DS0**+{**GCS**, **AT**}, each of which use **DS0** as the target model, and the other included ones as the auxiliary models. For example, **DS0**+{**DS1**, **GCS**} has two auxiliary models, **DS1** and **GCS**.

For an multiple-auxiliary-model system with n auxiliary models, n similarity scores are computed for a given input audio, which are grouped together as a feature vector. The feature vector is then fed into a binary classifier (e.g., **SVM**) to predict whether the input audio is benign or adversarial.

For each multiple-auxiliary-models system, we train a binary classifier on all the 1800 feature vectors (900 benign samples and 900 AE samples), and test it on over 450 test samples. We use three different binary classifiers, including **SVM**, **KNN** and **Random Forest**, and configured each classifier as follows: (1) **SVM** use a 3-degree polynomial kernel; (2) **KNN** use 10 neighbors to vote; and (3) **Random Forest** use a seed of 200 as starting random state.

Table 4.6 shows the results. All the accuracy results are higher than 99%, and FPR and FNR are lower than 0.5%, regardless of auxiliary models and binary classifiers. The results also show that the three-auxiliary-models system outperforms the two-auxiliary-models systems, probably due to more information learned by the three-auxiliary-models system. As the detection accuracy of the three-auxiliary-models

system is already 100% for `KNN` and `Random Forest`, we stop at three and did not include more auxiliary models.

4.5.7 Robustness against Unseen Attack Methods

The last experiment aims to examine whether a system trained on AEs generated by a particular attack method is able to detect AEs generated by other kinds of attack methods—such an AE is called an *unseen-attack AE*. We use the *defense rate*, defined as the ratio of the number of successfully detected AEs among the total number of AEs, to measure the robustness. The higher the defense rate, the better robustness of the system.

Single-auxiliary-model systems. We first examine the three single-auxiliary-model systems. We train each system using only the benign samples, and test each one on all the 1125 AEs (see Table 4.2). All the AEs can be considered as unseen-attack AEs. The results are presented in Table 4.7.

For each system, the threshold is determined by maximizing FPR under an upper-bounded constraint, which is set as 5%. We can see that the lowest defense rate is 98.67% for the `DS0+{GCS}` system. We further try different upper bound values for FPR, including 4%, 3% and 2%. The defense rate slightly drops with 2% as the upper-bound value, and the `DS0+{GCS}` system has the lowest defense rate of 98.49%.

Multiple-auxiliary-model systems. We next examine the four multiple-auxiliary-model systems. As presented in Section 4.5.2, two different methods are used to generate the AEs: the white-box approach and black-box approach. There are totally 100 black-box AEs and 1025 white-box AEs.

We conduct two different experiments to evaluate each of the four multiple-auxiliary-model systems. (1) We first use all the 1025 white-box AEs and 1025 benign samples to train each system, and use all the black-box AEs to test each

trained system. The results are showed in the second column in Table 4.8. It can be observed that the defense rates of all the systems are 100%—all the black-box AEs can be successfully detected. (2) We next use all the 100 black-box AEs and 100 benign samples to train each system, and use all the white-box AEs to test each trained system. The results are showed in the third column in Table 4.8. We can see that all the systems perform very well, and the lowest defense rate is 98.34% for the DS0+{GCS, AT} system.

Therefore, we can conclude that our detection method is very robust against unseen-attack AEs.

4.6 Chapter Summary

Work on handling audio AEs is still very limited. Considering that ASRs are widely deployed in smart homes, smart phones and cars, how to detect audio AEs is an important problem. Inspired by Multiversion Programming, we propose to run multiple different ASR systems in parallel, and an audio input is determined as adversarial if the multiple ASRs generate very dissimilar transcriptions. Detection systems with one single auxiliary ASR achieve accuracies over 98.6%, while designs with more than one ASR achieve even higher accuracies as more features are provided to the classifier. Moreover, the research results invalidate the widely-believed claim that an adversary can embed a malicious command to *any* host audio.

Table 4.3. Comparison of 6 similarity metrics.

Similarity Metric	Performance	System			
		DS0+{DS1, GCS}	DS0+{DS1, AT}	DS0+{GCS, AT}	DS0+{DS1, GCS, AT}
<i>Jaccard</i>	Accuracy	447/450 (99.33%)	447/450 (99.33%)	435/450 (96.67%)	448/450 (99.56%)
	FPR	3/225 (1.33%)	3/225 (1.33%)	15/225 (6.67%)	2/225 (0.89%)
	FNR	0/225 (0.00%)	0/225 (0.00%)	0/225 (0.00%)	0/225 (0.00%)
<i>Cosine</i>	Accuracy	448/450 (99.56%)	448/450 (99.56%)	444/450 (98.67%)	448/450 (99.56%)
	FPR	2/225 (0.89%)	2/225 (0.89%)	6/225 (2.67%)	2/225 (0.89%)
	FNR	0/225 (0.00%)	0/225 (0.00%)	0/225 (0.00%)	0/225 (0.00%)
<i>JaroWinkler</i>	Accuracy	447/450 (99.33%)	447/450 (99.33%)	444/450 (98.67%)	446/450 (99.11%)
	FPR	3/225 (1.33%)	3/225 (1.33%)	5/225 (2.22%)	4/225 (1.78%)
	FNR	0/225 (0.00%)	0/225 (0.00%)	1/225 (0.44%)	0/225 (0.00%)
<i>PE_Jaccard</i>	Accuracy	447/450 (99.33%)	448/450 (99.56%)	445/450 (98.89%)	447/450 (99.3%)
	FPR	3/225 (1.33%)	2/225 (0.89%)	5/225 (2.22%)	3/225 (1.33%)
	FNR	0/225 (0.00%)	0/225 (0.00%)	0/225 (0.00%)	0/225 (0.00%)
<i>PE_Cosine</i>	Accuracy	448/450 (99.56%)	448/450 (99.56%)	443/450 (98.44%)	447/450 (99.33%)
	FPR	2/225 (0.89%)	2/225 (0.89%)	6/225 (2.67%)	3/225 (1.33%)
	FNR	0/225 (0.00%)	0/225 (0.00%)	1/225 (0.44%)	0/225 (0.00%)
<i>PE_JaroWinkler</i>	Accuracy	449/450 (99.78%)	449/450 (99.78%)	448/450 (99.56%)	449/450 (99.78%)
	FPR	1/225 (0.44%)	0/225 (0.00%)	1/225 (0.44%)	1/225 (0.44%)
	FNR	0/225 (0.44%)	1/225 (0.44%)	1/225 (0.44%)	0/225 (0.00%)

Legend: X+{ Y_1, \dots, Y_n }, stands for a system using X as the target model and Y_i ($Y_i \in \{1, \dots, n\}$) as the auxiliary models.

Table 4.4. The training results of three single-auxiliary-model systems.

System	Threshold	# of TPs	# of FNs	# of TNs	# of FPs	TPR	FPR	AUC
DS0+{DS1}	0.78	885	15	898	2	98.33%	0.22%	0.9990
DS0+{GCS}	0.75	881	19	885	15	97.89%	1.67%	0.9930
DS0+{AT}	0.77	885	15	898	2	98.33%	0.22%	0.9973

Table 4.5. The testing results of three single-auxiliary-model systems.

System	Threshold	# of TPs	# of FNs	# of TNs	# of FPs	TPR	FPR	Accuracy
DS0+{DS1}	0.78	223	2	224	1	99.11%	0.44%	99.33%
DS0+{GCS}	0.75	222	3	222	3	98.67%	1.33%	98.67%
DS0+{AT}	0.77	224	1	225	0	99.56%	0.00%	99.78%

TP, FN, TN and FP represent True Positive, False Negative, True Negative, and False Positive, respectively.

Table 4.6. The testing results of four multiple-auxiliary-model systems when different binary classifiers are adopted..

Classifier	Performance	System			
		DS0+{DS1, GCS}	DS0+{DS1, AT}	DS0+{GCS, AT}	DS0+{DS1, GCS, AT}
SVM	Accuracy	449/450 (99.78%)	449/450 (99.78%)	448/450 (99.56%)	449/450 (99.78%)
	FPR	1/225 (0.44%)	0/225 (0.00%)	1/225 (0.44%)	1/225 (0.44%)
	FNR	0/225 (0.00%)	1/225 (0.44%)	1/225 (0.44%)	0/225 (0.00%)
KNN	Accuracy	449/450 (99.78%)	447/450 (99.33%)	448/450 (99.56%)	450/450 (100%)
	FPR	1/225 (0.44%)	1/225 (0.44%)	1/225 (0.44%)	0/225 (0.00%)
	FNR	0/225 (0.00%)	2/225 (0.89%)	1/225 (0.44%)	0/225 (0.00%)
Random Forest	Accuracy	448/450 (99.56%)	449/450 (99.78%)	449/450 (99.78%)	450/450 (100%)
	FPR	0/225 (0.00%)	0/225 (0.00%)	0/225 (0.00%)	0/225 (0.00%)
	FNR	2/225 (0.89%)	1/225 (0.44%)	1/225 (0.44%)	0/225 (0.00%)

Table 4.7. The detection results of unseen-attack AEs for three single-auxiliary-models..

System	Threshold	FPR	FNs	FNR	Defense rate
DS0+{DS1}	0.88	4.44%	10	0.89%	99.11%
DS0+{GCS}	0.81	4.53%	15	1.33%	98.67%
DS0+{AT}	0.83	3.73%	14	1.24%	98.76%

Table 4.8. The detection results of unseen-attack AEs for four multiple-auxiliary-models..

System	Defense rate	
	Black-box AEs	White-box AEs
DS0+{DS1, GCS}	100.00%	98.63%
DS0+{DS1, AT}	100.00%	98.63%
DS0+{GCS, AT}	100.00%	98.34%
DS0+{DS1, GCS, AT}	100.00%	98.54%

Chapter 5

BLOCKCHAIN-ASSISTED RELAY SHARING OF SMART HOME IOT

The centralized cloud-based IoT service architecture has the limitation in terms of security, usability, and scalability, and is subject to the single points of failure (SPOF). Recently, accommodating IoT services on blockchains becomes a trend for better security. However, blockchain's shortcomings on cost, throughput, and latency beset its integration with IoT. In this work, we take a retrospection of exiting blockchain-based IoT solutions and propose a framework for efficient blockchain and IoT integration. Following the framework, we design the first blockchain-assisted IoT accessing system, RS-IoT, which can defend IoT devices against zero-day attacks without relying on any trusted third-party. By introducing incentives and penalties enforced by smart contracts, our work enables “*an economic approach to cyber security*”, thwarting attackers who aim to achieve monetary gains. We prove the security of RS-IoT via detailed security analyses and demonstrate its efficiency and usability through a proof-of-concept implementation on the Ethereum testnet blockchain.

5.1 Introduction

The IoT market is flourishing. According to Gartner's report in 2018 [177], there is predicted to have 14.2 billion connected things in use in 2019 and 25 billion by 2021. However, security concerns are raised along with the growth of the IoT market. A series of horrible IoT-related attacks have been seen during the past years such

as the Mirai botnet attack [178], BrickerBot attack [179], Deutsch Telekom TR-069 attack [178]. Newly developed fancy pwns and hacks targeting IoT devices like [180, 181] are emerging every day. Considering the scale of IoT devices, securing them becomes a non-trivial task.

Securing these vulnerable IoT devices is challenging. Not only because those low-cost IoT devices are lacking of computing resources and I/O peripherals, but also due to the IoT vendors' lax on implementing secure softwares. Many manufacturers are busy rolling out products with novel features while leaving security flaws unpatched for years [182, 181, 183, 184]. Given the actuality that IoT devices are vulnerable, it is reasonable to have the assumption of *access-to-attack*, which means any attacker that gains direct access to an IoT device's open port is supposed to be able to compromise the device. Modern IoT vendors try to address this problem by anchoring the access entries of their products on endpoint instances hosted by cloud servers [31]. They migrate critical services such as authentication, remote administration, and data collection from vulnerable IoT devices to the more secure cloud server. Unfortunately, the insecurity of cloud servers is never a piece of news with prominent examples of high-profile compromises including Equifax [185], Dropbox, and the US voter database [186]. Since the Cloud endpoint serves as the entry and is trusted by a large number of IoT devices, it may, on the contrary, give adversaries an additional arsenal for launching large scale attacks [187, 188].

Recent advancements on the blockchain and smart contracts inspire researchers to seek blockchain-based solutions for its intrinsic advantages on decentralization, faulty tolerance, and data integrity. However, incorporating blockchain and IoT is not easy due to the blockchain's characteristics and the IoT's requirements. On the one hand, billions of IoT devices are running 24/7 and produce enormous amounts of data to be timely stored and processed. On the other hand, blockchain usually

has limited throughput and is costly. Although the smart contract is theoretically turing-complete, the Proof-of-Work (PoW) consensus mechanism makes it not only expensive but also slow.

Currently, most research works on blockchain-based IoT solutions are still using the blockchain as a substitution of the cloud server. Their approaches to solving the aforementioned challenges can be roughly categorized into three types: 1) They choose to study specific services that are latency insensitive and bring low overhead, for example, IoT authentication and identity management. 2) They use private or permissioned blockchains instead of public blockchains to avoid cost and throughput issues. 3) They turn to edge computing where edge servers are introduced to mitigate the IoT's resource constrains for interacting with the blockchain. These workarounds are either making a trade-off between the usability and security or are limited on specific tasks. A general framework to efficiently integrate blockchain into IoT systems is still an open question.

In this work, we try to fill this gap by rethinking the blockchain's role in the IoT service architecture. Instead of being the host to accommodate services directly, blockchain is more suitable to be a service trading platform where service users and third-party providers can discover each other, establish commission relationships, and settle service fees. The service itself is undertaken by independent third-party providers. The participation of third-party providers decouples the relationship between IoT devices and vendor operated cloud servers, which allows IoT devices switch to any provider for better service security and quality. To realize it, the following questions need to be answered:

Q1: How to motivate the participation of third-party service providers?

Q2: How to establish a mutual trust between service users and providers?

Q3: How to prevent malicious behaviors like cheating, attacking, and denial of service?

We propose RS-IoT, a novel blockchain assisted IoT relay sharing system as a case study of the blockchain-assisted IoT remote access system. In the system, we introduce third-party relay servers to substitute the centralized message broker [189] for enabling two-way relayed communications between the user's controller device and IoT device. IoT device owners on this platform can freely commission any relay servers for their devices instead of using those designated by device vendors. The decentralized nature of the proposed technique resolves the SPOF and scalability issues. Leveraging the power of the smart contract [190], we design a transparent, self-governing relay service trading platform where *monetary incentives* are used to motivate third-party relay providers' participation and deter potential malicious behaviors. Furthermore, we propose the proof-of-delivery scheme for fair and objective disputation arbitration and attack handling. With the scheme, proofs generated in off-chain settings can be verified on the smart contract, which significantly reduces the execution of smart contract functions. As a result, third-party relay servers get the incentive to shield their customer IoT devices, which gives vulnerable IoT devices additional protection against zero-day attacks.

5.2 Background

5.2.1 N-version Programming

N-version programming (NVP) [191] is a method used in software engineering in which multiple versions of software are created from the same copy of initial specifications. These equivalent copies of softwares are developed independently in different approaches. It is proved that NVP can greatly reduce the influence from identical

software faults. The concept has already been used as an effective defense method against software flaws [192]. For our proposed relay sharing system, the variety of IoT devices' software implementation makes it impossible for attackers to launch universally applicable attacks and imposes risks when they make unsuccessful attempts.

5.2.2 Non-Repudiable TLS

TLS-N [193] is an extension of the current TLS protocol by adding non-repudiation features. Unlike the standard TLS protocol, which only uses HMAC in application data packets for message integrity checking, TLS-N enables traffic signatures between communication peers. By adding a verifiable signature created with the private key, the sender cannot deny sending certain content. The non-repudiable feature perfectly solves the requirement in the smart contract, where objective evidence is required for the contract's execution. In our work, TLS-N signatures of malicious packets are used by our reporting system as the evidence of relay servers' misbehavior.

5.2.3 Smart Contract

Smart contract [190] consists of pieces of script code on blockchain to deal with digital assets. It is executed by all miner nodes in the isolated virtual environment. The accepted execution results are recorded on the decentralized ledger through consensus mechanisms which ensures it to be trustworthy and tampering-resistant. Smart contracts provide two types of accounts: the personal account is owned by the participating node and protected by the private key; the contract account points to the instance of smart contract code without private keys. Both types of accounts have the account addresses and are able to hold digital assets.

5.3 Blockchain and IoT Integration

Cloud platforms have been proved to help enhance resource constraint IoT devices against security threats [194, 195, 196]. However, the risk of single point of failure and the problem of scalability bottleneck entice IoT vendors to shift their services from the centralized architecture to the decentralized form. The blockchain's intrinsic natures of decentralization, tamper-resistance, and autonomy make it a promising candidate to be used in the IoT paradigm. Furthermore, the emergence of smart contracts provides a practical way for Turning-complete trusted global computer, which inspires the proposal of autonomous IoT system [197].

However, these wonderful security benefits of the blockchain are not to be taken for granted. The 'world computer' is achieved by using numerous blockchain nodes as redundant backups. This brings high cost and latency for operations on public blockchains. Moreover, despite blockchain's good scalability of accommodating unlimited participating nodes, the whole network throughput of transaction processing is limited. For instance, Bitcoin has a constant throughput of 10 minutes per block, which is equivalent to 7 transactions per second and Ethereum allows 15 seconds per block. Another issue is cost. All operations that modify the public blockchain's state are subject to transaction fees. Even simple operations like storing or changing a byte in the blockchain can be expensive, let alone complicated tasks such as data processing. Considering the IoT's requirement of large scale and low cost, the use of blockchain becomes unpractical.

To find a general and viable solution, we first review the existing research works of blockchain-based IoT security mechanisms and analyze their strengths and weaknesses on addressing the two mentioned constraints. Based on the retrospection, we propose our efficient integration framework.

5.3.1 Retrospection of Existing Works

Since the proposal of smart contracts in 2014, there have been many research works regarding the paradigm of IoT and blockchain integration. We choose three well-presented survey papers [198, 199, 200] as indices to collect notable works related to this topic for further analyses and discussions. In contrast to these survey papers, we focus on categorizing and evaluating the technical methods used by these papers to solve the aforementioned problems.

5.3.1.1 Blockchain as a Ledger

In this type, the blockchain is used as an immutable ledger or a distributed database to store critical information. This is the most straightforward way for the integration of IoT and blockchain and is adopted by many early works. [201] proposes to use the blockchain to store and deliver out-of-band authentication credentials. Since authentication only occurs when devices request sensitive information from the cloud server, it has low requirements for throughput and latency. The authors indicate that they use the Eris blockchain, but they do not specify any details about the implementation, nor a solution to reduce the involved transaction fees. [202] designs a multi-layer architecture for blockchain powered smart home access control. It employs a centralized device to process all transactions and generate new blocks to get rid of the overhead brought by the Proof-of-Work. However, this design contradicts the core concept of decentralization of blockchain and undermines the security. Some other works straightforwardly leverage the blockchain's immutable feature to facilitate distributed data storage and sharing as discussed in [203] and [204]. They use public blockchain to store either raw data or the hash of data. These works avoid the cost and

throughput constraints by either choosing specific services that have low requirements for transaction frequency and latency or using a single centralized miner.

5.3.1.2 Blockchain as a Service

Inspired by the concept of decentralized applications [205], some works regard the smart contract as a trusted computing platform and build more applications on it. The first category is decentralized PKIs based on the smart contract. [206] introduces an additional party named ‘bookkeeper’ to form the backbone of a permissioned blockchain and store the certificate revocation list. But it does not specify the origin and the incentive of ‘bookkeepers’, which implies a limited number of available bookkeepers. Certcoin [207] chooses to build a decentralized trust system based on Namecoin, a public blockchain for DNS services. Although the author finds a solution to mitigate end users’ storage burdens, all certificates and public keys are still held on the blockchain. Access control and authorization is another hot topic for blockchain and IoT integration. ‘WAVE’ [208] proposes a city-wide blockchain to store the metadata of permissions for supporting a variety of access control patterns. The authors craft a set of smart contracts to automatically handle complicated out-of-order access permission delegations. A bunch of other IoT authorization solutions [209, 210, 211] basically use the similar architecture that employs smart contracts to automatically enforce access control policies. They try to address the cost and throughput issues either by using permissioned blockchains [206, 211, 210], or delicately designing the contract code to minimize the frequency of modifying the global state of blockchain [209, 208].

In summary, the aforementioned works basically follow two approaches. First, they choose specific services that require a low frequency of issuing transactions to avoid intensive on-chain operations which are subject to fees and latencies. Reading

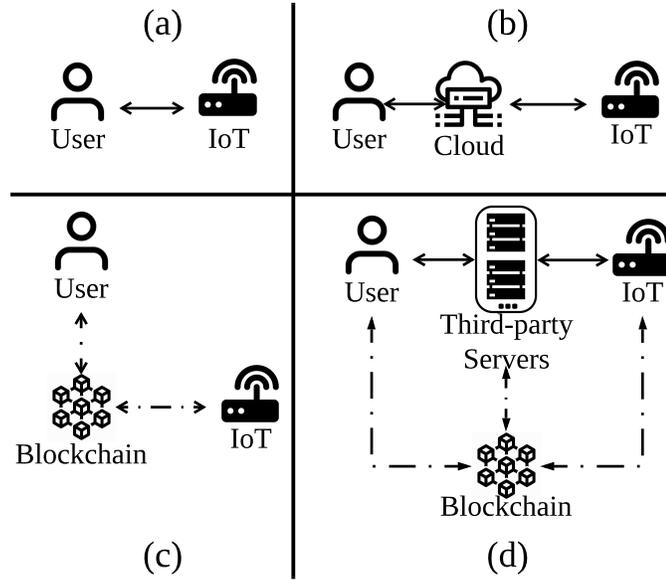


Figure 5.1. Abstract architectures of IoT services. .

the blockchain data would not cause a state change and has no cost or minimal overhead. Therefore, the blockchain can host data sharing, PKI services, and access controls, which have asymmetric requirements of reading and writing. This also explains why a considerable portion of blockchain-IoT papers in the survey [198] focus on these topics.

Another approach is using private or permission blockchains instead of the public blockchain, where the burden of computing-intensive proof-of-work is relieved or avoided. However, it also undermines the security with a much smaller network scale. Some new blockchain techniques such as hyperledger [212] and IoTa [213] achieve better performance by making trade-offs either on decentralization or on security.

5.3.2 Rethinking Blockchain and IoT Integration

To explore a feasible way to integrate IoT and blockchain, we first make an abstract model of IoT services as depicted in Figure 5.1. In a conventional IoT services sce-

nario, with client devices (e.g., smartphone), users can either access their IoT devices directly (case (a)) or through the cloud server for advanced functionalities like cloud storage and automation (case (b)). The blockchain, as mentioned earlier enables IoT services to replace the cloud server with the blockchain (case (c)). One promising architecture is proposed in [214] as shown in (case (d)) that combines the ordinary distributed system with blockchain by moving heavy-load services from the blockchain to third-party servers. The blockchain serves as the coordinator to enforce correctness and fairness. However, this work mainly focuses on the theoretical model of secure multi-party computing and does not offer more details of how to realize it in the IoT domain.

We follow this insight and dive into more details of blockchain-assisted distributed IoT services to find the answer to the three questions as proposed in the Section 5.1. Many blockchains provide the functionality of cryptocurrencies. So, the blockchain can be used as a service trading platform where service users (usually IoT devices) use services provided by third-party service providers who join for service fees paid by cryptocurrencies. To avoid possible disputations, the blockchain can also serve as the intermediary between two parties that collects pre-paid service fee payments from service users and compensates providers when objective proofs are provided. The blockchain can also punish malicious service providers by forfeiting their deposits of cryptocurrency to deter possible attacking attempts.

5.4 Case Study of Efficient Blockchain and IoT Integration: IoT Remote Access

Smart home IoT systems enable homeowners to manage their IoT sensors and appliances from both inside and outside their homes. However, accessing IoT outside the

home’s private network is challenging due to the isolation of Network Address Translation (NAT) and the firewall. As a result, IoT remote access requires a relay server with a public accessible IP address for bridged communication. In this section, we study the use case of IoT remote accessing to demonstrate the proposed framework.

5.4.1 Threat Model

According to the background knowledge and the aforementioned challenges, we present our threat model here. Following the *access-to-attack* assumption, we assume very strong attackers who have capabilities to 1) take down any IoT devices by exploiting application layer software vulnerabilities as long as they can get access to them; 2) take down arbitrary relay server with non-negligible time 3) install malware on compromised IoT devices for botnet attacks.

These assumptions are practical because of the principle of multi-version programming. Considering the diversity of IoT devices, there is no universally applicable vulnerability. Besides, compared to vendor self-developed application layer programs, the underlying components such as operating system kernel and TLS-N library are well tested by many developers, which makes them far more difficult to be exploited by attackers. So, those extremely strong attacking vectors exploiting fundamental system-level software vulnerabilities are considered to be out of our scope. Particularly, to achieve our claimed protection, we do not need to assume our proposed system can be perfectly implemented by device vendors.

5.4.2 State-of-art Solutions

5.4.2.1 Port forwarding

Direct access to IoT devices may be obstructed since home routers with NAT (Network Address Translation) and firewall shield IoT devices in private networks and filter out inbound connections. The intuitive workaround is configuring port forwarding to expose devices' ports to the router's public IP Address. However, this solution exposes IoT devices to the attacking traffic from the Internet. For example, a surprisingly high number of IoT devices are compromised by the Mirai botnet malware because they are exposed to the public Internet with the UPnP IGD protocol [215].

5.4.2.2 Cloud-based Endpoint

Even shielded by the gateway, vulnerable IoT devices behind NAT are still threatened by other malicious devices in the same private network as described in [48]. Modern IoT vendors try to address this problem with further isolation. They anchor the device access entry of their devices on cloud endpoint instances and deploy end-to-end encryption in between [216]. Since the keep-alive session is proactively launched by the IoT device located behind the NAT, access requests encapsulated in the response packets can freely pass through the NAT and firewall, which is dubbed as "piggybacking." A typical real-world example is the message broker service [217] provided by Amazon Web Service's IoT core module. It maintains a long-term session with subscribing IoT devices where messaging protocols such as MQTT are carried for real-time device control.

Counterintuitively, endpoints hosted on the centralized cloud server could provide attackers with a better arsenal. As the cloud server usually hosts endpoints for similar

devices from the same vendor, attackers may exploit it [218] to acquire direct access to a large number of IoT devices that share similar vulnerabilities. This will save attackers much effort and time to scan for potential victim hosts randomly on the Internet. As examples in [31, 8], flawed cloud endpoints APIs allow attackers to impersonate or even hijack IoT devices. These vulnerable cloud endpoints could be utilized by IoT botnet attackers [180, 219] to spread malware quickly. Besides, with the scale of IoT devices increasing rapidly, hosting centralized cloud service imposes the risk of the single point of failure (SPOF), which may cause large-scale IoT device blackout.

5.5 Design of RS-IoT

In this section, we present our design of the blockchain based relay-sharing IoT remote access system (RS-IoT), which utilizes the framework we propose in Section 5.3. Since a relay server is necessary for accessing IoT devices behind the NAT, a feasible solution is to decouple the fixed relationship between relay server and the IoT devices by replacing the centralized cloud server with a large number of third-party relay servers. Accordingly, a smart contract based service trading platform is designed for transaction management and dispute arbitration.

Our design brings four prominent benefits: 1) The management platform is completely self-governing and distributed without relying any third-party authorities; 2) Compromised or misbehaving relay servers will be reported and excluded from the relay platform to avoid further damage; 3) Risk of economic loss are introduced to deter malicious attacking attempts.

5.5.1 Relay Sharing Model

There are four roles involved in RS-IoT: the IoT device (D), the controller (C), the relay server (R), and the smart contract (SC). Among them, the controller and the IoT device are grouped as the party of the service user and share secret keys since the same owner owns them. The relay server constitutes the party of the service provider. The smart contract is script code stored on the public blockchain as an arbiter.

IoT devices, controller devices, and relay servers join the blockchain by generating their public and private keys. The smart contract is published on the blockchain with only the account address. We denote their addresses on as $addr(D)$, $addr(C)$, and $addr(R)$ respectively. After that, the IoT device, the controller, and the relay server top up their account with cryptocurrency as a deposit or service fee. The unit of relay service is a relayed session between an IoT device and its controller device through one relay server. Both the IoT device and the controller are connected to the relay server via TLS-N sessions, and all packets passed through are packaged as blockchain transactions and signed using their private keys.

5.5.2 Relay Workflow

This subsection describes the workflow of relay sharing by going over the procedure of forwarding a packet. As described in Figure 5.2, the workflow consists of the registration phase, the commission phase, and the relay phase.

5.5.2.1 Registration

With blockchain accounts established, all three parties register their account addresses on the smart contract to indicate their participation by calling a registration function of the smart contract SC and creating new items in two tables in blockchain: *userInfo*

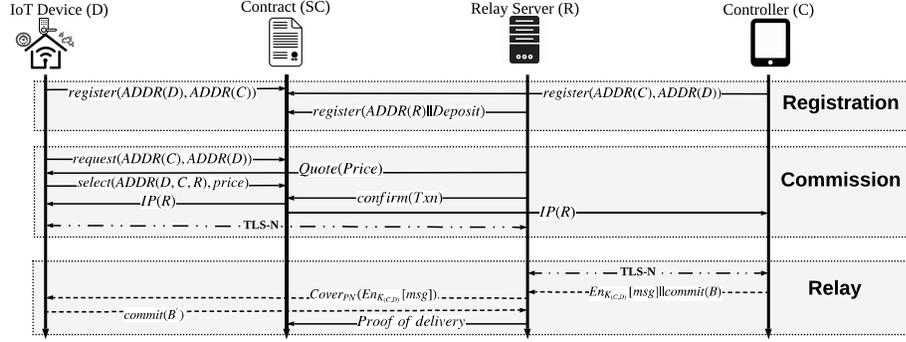


Figure 5.2. Workflow of RS-IoT. .

and *serverInfo*. Since a service user is uniquely identified by its pair of IoT device ID and controller ID, both of them need to register by providing the other party's address as arguments. After either one of them calls the registration function, a new item in *userInfo* is created with the confirmation flag set to false. Then, the function call by the other party would flip the confirmation flag to indicate a successful user registration. For the relay server, the deposit of cryptocurrency is required to be paid to the smart contract along with the registration function call transaction. The amount of deposit is stored in *serverInfo* together with the relay server's blockchain address.

5.5.2.2 Commission

The commission phase is used for mutual discovery between service users and providers, as well as setting up service relationships. The commission phase begins with an IoT device calling the **service request** function which broadcasts a global event containing the user's registration information. Upon receiving the event, interested relay servers respond with their IP addresses and quotes of service via direct transactions towards the requesting IoT's blockchain address. After waiting for some time, the IoT device evaluates the received quotes by the deposit and price, and finally makes a de-

cision by calling the **service select** function with the chosen relay server’s blockchain address and price as arguments. Similar to the user registration, this function call would generate an item into the table *serviceList* consisting the following values as shown in Table 5.1 and set the confirmation flag to pending.

Table 5.1. Keys of content in the serviceList..

<i>Txn</i>	index number for each pair of IoT device and relay server
<i>Serial</i>	counter to index the number of successfully relayed packets
<i>Address</i>	blockchain address of all involved parties
<i>Price</i>	cost of forwarding one packet
<i>Balance</i>	amount of pre-paid service fee payment

Finally, the chosen relay server confirms the service relationship by calling the **service confirm** function to change the confirmation flag in table *serviceList* to confirmed. At the same time, an event broadcast would be triggered as a log of relationship binding. Then, the commission is finished. The IoT device launches a TLS-N connection towards the commissioned relay server.

5.5.2.3 Relay

As all history transactions on blockchain are publicly readable, the controller client can easily recover the current serving relay server’s IP address and initiate a TLS-N connection towards it. With the shared secret key between the IoT device and the controller client, a long-term symmetric encryption key $K(C, D)$ can be derived to encrypt the messages exchanged between them. The controller client first generates a proof of signed transaction on the original packet which is sent to the relay server along with the packet itself. On receiving the packet, the relay server uses a random stream derived from a one-time key PN to cover the packet before forwarding it to the IoT device. Afterwards, the receiver (IoT device) generates another proof

transaction using the same algorithm but on the covered packet and sends it back to the relay server. Finally, the relay server creates a new transaction containing the cover key and broadcasts it along with two received proof transactions. With the cover key, a successful proof verification on the smart contract will trigger a transfer of cryptocurrency from the contract account to the relay server’s personal account. We will illustrate details of the proof generation algorithm and show its effect on preventing cheating in Section 5.6 and Section 5.8, respectively.

5.5.2.4 Decommission

As we stated, both the relay server and the IoT device can freely determine when to end the service relationship. The decommission process is provided to terminate a service relationship by either party by calling the function **decommission**. *Txn* is the only argument required to specify the service record to be cleared. After a decommission is initiated, an event would be emitted as the notification and the service record item in *serviceList* is deleted after a pre-defined block time for the relay server to finish billing. The remaining pre-paid service fee will be paid back to the IoT device’s personal account.

5.5.3 Billing of Relay Service

During the commission procedure, the pre-paid service fee is transferred to the smart contract’s account along with the function call of **service confirm**. As described in Section 5.5.2.3, a relay server gets remuneration by presenting the proof of each successfully relayed packet. However, considering the amount of relayed packets and blockchain’s aforementioned cost and throughput constraints, verifying each of them on the smart contract is unrealistic. We innovatively use the smart contract’s local

verifiability to make it unnecessary to verify each packet on the blockchain. That is, on receiving proof transactions, the relay server first runs the verification function locally instead of posting them on the blockchain. If the verification is successful, the relay server caches the proofs and for the current packet, and send the cover key to the IoT device. If both two parties are honest, the IoT device would be able to successfully recover the message with the received cover key. As long as no dispute occurs, this offline verification can be used for all following packets. When the relay server wants to cash the remuneration, it only needs to verify the proof of the last packet on the blockchain. The difference between the recorded serial number *serial* in the table *serviceList* and *serial* in the proof is accounted as the number of successfully relayed packets. The total amount of remunerations is then calculated as the product of the number of successfully relayed packets and the unit service price. After the transfer, the *serial* in *serviceList* is updated. Because the *serial* as a function call argument is signed by the IoT device, it can be regarded as the IoT's acknowledgment of successful relay for all packets before it.

5.6 Proof-of-delivery

In a service trading system, both the service user and the service provider have the incentive to cheat. The relay server may deliver broken, modified, or forged packets for making an extra profit; While, the relay user (including the controller and the IoT device) tend to deny the receipt of packets to avoid the payment. To solve this problem, we propose an autonomous proof-of-delivery mechanism to resolve possible disputes fairly by utilizing the smart contract as a decentralized trusted computing platform. First, we design an SHA-3 [220] based stream generator for the relay server to hide the content of the packet to be forwarded. Then, leveraging

the smart contract’s locally verifiable feature, we propose an innovative off-line blind proof generation algorithm to derive proofs of packet delivery on both the original and the covered packet. During the operation, the relay server holds the cover key while it asks the IoT device for the proof of the covered packet. On the one hand, without receiving a correct proof, the relay server would not reveal the cover key to let the receiver retrieve the message. On the other hand, the relay server is not able to receive the service fee if the packet is not delivered confidentially. This solution provides a mutual restriction between the service user and server so that neither of them can achieve their goal by cheating. Thanks to the smart contract’s off-chain verifiability, the proof only needs to be posted on the blockchain when disputes occur rather than every time a packet is relayed, which significantly reduces the cost of operation. For clarity, we list all symbols used in the following notation table:

Table 5.2. Notations.

$K_{(C,D)}$	pre-shared encryption key between C and D
$S_{(C,D)}$	pre-shared secret for bits selector
Ra & Ra'	byte select index list
PN	secret key for cover stream generator
B	selected bytes for uncovered packet
B'	selected bytes for covered packet
$Tx(B)$	contract function call with B as argument
N	commitment length
i	self-incrementing counter

5.6.1 Components

Considering the high cost of storing and processing data on the blockchain, it is not practical to verify the entire packet on the smart contract because of the unbearable latency and cost. Although existing cryptography primitives have already provided satisfying digest-based security features, hardly any of them are available as built-in

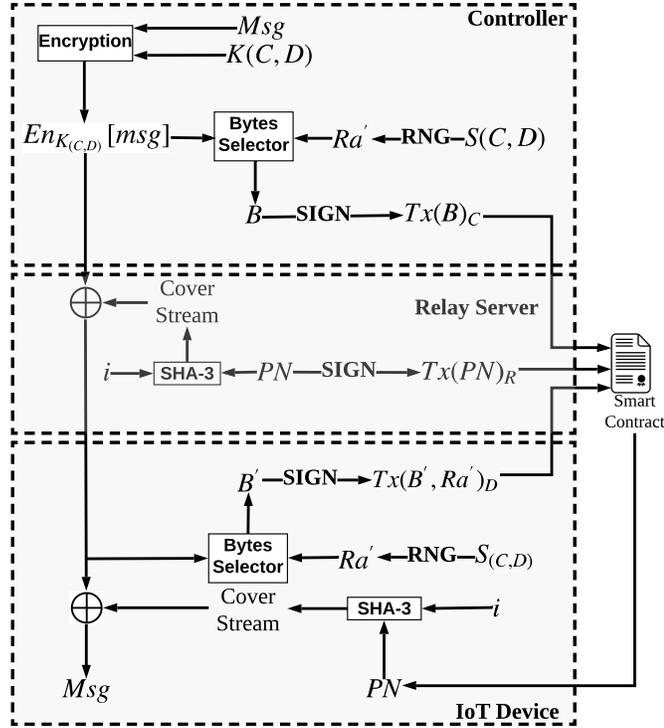


Figure 5.3. The workflow of proof-of-delivery. .

functions on popular public blockchain platforms such as the Ethereum [221]. If we implement them in the form of script code, the cost will become unbearable. To overcome these difficulties, we design the **Bytes Selector** and the **Cover Stream Generator** as new primitives based on Ethereum’s built-in functions. The Bytes Selector is used to extract fixed-length bytes streams from arbitrary packets, while the Cover Stream Generator is used to generate cover streams to hide the content of a packet. These new components provide comparable security while remains low cost.

5.6.1.1 Cover Stream Generator

To prevent service users maliciously denying the receiving of packets, the relay server covers the content of packets to be forwarded with a pseudo-random stream derived

from a cover key. Since there is no pre-compiled script function in the current version of Ethereum blockchain, implementing a standard stream cipher would be extremely expensive. So, we design our own pseudo-random stream generation algorithm, which is based on Ethereum’s built-in SHA-3 has function [221]. We concatenate hashes of sums of a secret key PN and a self-increasing counter i . To keep high entropy of randomness, we only retain the highest indexed byte of each 256-bit SHA-3 hash result. The cover stream is generated, as shown in the equation below. The ‘|’ means the concatenation of hash values.

$$cover(PN) = SHA3(PN)|SHA3(PN + 1)|\cdots \quad (5.6-1)$$

Aside from using the low-cost building block, this algorithm also reduces cost by allowing selective stream generation. That is, to encrypt/decrypt the content at the K th byte with a given key PN , we only need to calculate $SHA3(PN + K - 1)$ instead of producing the stream from the beginning.

5.6.1.2 Bytes Selector

As an analogy, the bytes selector is similar to the Hash Message Authentication Code (HMAC) function. The difference is that our bytes selector retains the commutativity when used with our cover stream generator (i.e., the digest of the covered packet equals the covered digest giving the same secure keys).

The bytes selector is driven by the secret $S(C, D)$ shared between the controller client C and the IoT device D . Both C and D use this secret as the seed of a pseudo random number generator. For each packet, C and D synchronously generate N random numbers of 16-bits, denoted as $Ra = \{ra_1, ra_2, ra_3, \cdots, ra_N\}$. Assuming the length of the packet is L , the bytes selection list $Ra' = \{ra'_1, ra'_2, ra'_3, \cdots, ra'_N\}$

is derived from Ra with $ra'_i = ra_i \bmod L$. Thereafter, a list of N bytes $B = \{b_1, b_2, b_3, \dots, b_N\}$ are extracted from targeting packet where b_i is the ra'_i th byte in the packet. The generated bytes list is totally random and there is no way to recover their locations in the packet without the knowledge of the random seed $S(C, D)$.

5.6.2 Proof-of-Delivery Workflow

The proof-of-delivery comprises four steps: 1) The sender (assuming it is the controller because control session is usually initiated by it) generates the first commitment with our bytes selector on the original packet to be sent to the relay server. 2) The relay server covers the packet and forwards it to the receiver (IoT device). 3) The receiver generates the second commitment on the covered packet using the same method and sends it back to the relay server. 4) The relay server verifies two received commitments with the cover stream key and reveals the key to the receiver if commitments are valid.

5.6.2.1 Bytes Commitment by the Controller Client

We assume the controller C wants to send a packet msg to the device D through the relay server R . C firstly encrypts the message with the pre-shared symmetric key $K_{(C,D)}$ and generates the encrypted packet $En_{K_{(C,D)}}[msg]$. Then, it rolls the random number generator with the pre-shared secret $S(C, D)$ to get the Ra' as indices of bytes to be selected. After that, it extracts the bytes from $En_{K_{(C,D)}}[msg]$ as indexed by Ra' to form the commitment B and use it as the argument of the transaction towards the function of **commitment**. Finally, the transaction $Tx(B)_C$ is signed by the controller client's private key and sent to the relay server R .

5.6.2.2 Bits Covering by the Relay Server

On receiving the encrypted packet $En_{K(C,D)}[msg]$ and the commitment B from the controller client C , the relay server R generates a random number PN as the seed of the cover stream generator to produce an pseudo random stream. It uses the cover stream to cover the packet with the exclusive-OR operation. which is illustrated as in the equation below. After that, the covered packet is forwarded to the IoT device.

$$Co(En_{K(C,D)}[msg]) = En_{K(C,D)}[msg] \oplus cover(PN)$$

5.6.2.3 Bits Commitment by the IoT Device

The received packet from the relay server is covered by the cover stream. By using the bytes selector, bytes at the same location are selected on the covered packet which is denoted as B' . Then, the IoT device packages B' together with the bytes selection list Ra' into a function call transaction $Tx(B', Ra')_D$. The signed transaction is sent back to the relay server R .

5.6.2.4 Asynchronous Delivery Verification

Upon receiving both commitment transactions $Tx(B)_C$ and $Tx(B', Ra')_D$, the relay server now has all the materials to verify the correctness of the commitments locally. Then, the relay server checks whether $B \oplus B'$ equals to $cover(PN)$ at the designated location as specified by Ra' . If the verification is successful, the relay server prepares another transaction $Tx(PN)_R$ signed by itself with the PN as the argument. These three commitment transactions are the proof of delivery, which are cached by the relay server. When commitments are presented to the smart contract SC , the

same verification is performed as the relay server. Payment for the relay service is transferred to the relay server upon successful commitment verifications.

To avoid the latency and the transaction fee caused by executing the smart contract, the relay server caches all commitments instead of verifying them immediately. It delivers the cover key PN through the relay connection to the IoT device after the successful verification. When it wants to withdraw the payment, it verifies the commitment of the newest packet. As described in Section 5.5.3, the verification of all previous commitments is unnecessary.

5.7 Penalty & Disputation Solving

Since the relay server serves as the only access entry for all its customer IoT devices, all traffic delivered to the IoT device should originate from its legitimate controller clients. However, relay service providers are anonymous according to the registration phase, which induces the risk of malicious relay servers delivering IoT malware. Based on the non-repudiable features of TLS-N, we develop a smart contract function for relay service users reporting unauthorized traffic from their commissioned relay server. After issuing a log event as notification, the reported relay server needs to present the TLS-N signature of the sender for the reported packet to claim its innocence. Otherwise, the relay server's registration will be revoked, and all its deposits will be confiscated.

5.7.1 Reporting

Upon receiving a packet, the IoT device firstly verifies the TLS-N signature. Packets with valid signatures are forwarded to application layer programs. If the accepted packet contains malicious content that does not originate from the controller, the IoT

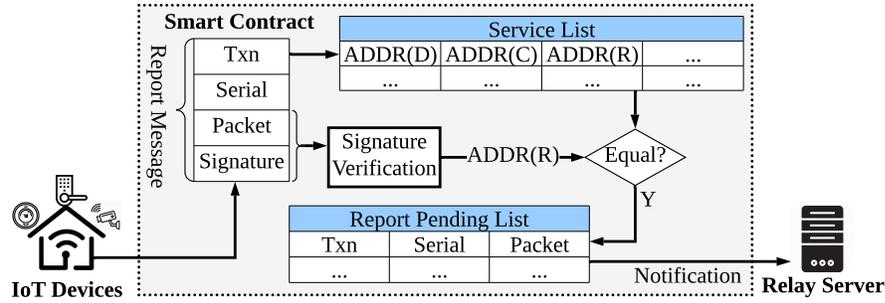


Figure 5.4. The workflow of reporting suspicious packets..

device has the chance to recognize it. Once malicious a relay server fails to compromise an IoT device, the attacking packets with its TLS-N signature will be reported by the IoT device to the smart contract as a proof of misbehaving. As shown in figure 5.4, to report the malicious relay server, the IoT device calls the reporting function in the smart contract with arguments of transaction number (*Txn*), packet serial (*Serial*), the received packet, and the relay server’s signature. The smart contract function verifies the signature and compares it with the address stored inside the *serviceList* as indexed by the given transaction number. If the recovered address matches that in the list, it means the reported packet really comes from the relay server, and the smart contract will take it as a valid report. Then the smart contract creates a new record in the report pending list, which contains the transaction number, the serial number, the reported packet, and the current latest block number. At the same time, a notification is broadcasted on the blockchain to inform the relay server to process this accusation.

5.7.2 Rebutting

The rebutting process is for relay servers to defend their innocence against allegations by proving the reported packet originates from the controller device. As shown in

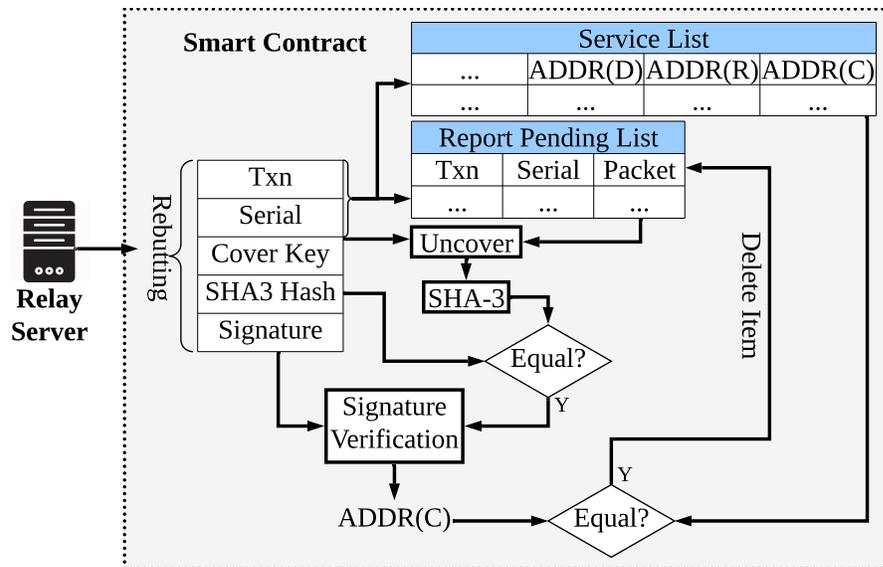


Figure 5.5. The workflow of rebutting a pending report record..

Figure 5.5, the reported relay server starts rebutting by calling the smart contract function. It provides the reported packet’s cover key and sender signature. The smart contract function repeats the proof-of-delivery verification to compute the SHA-3 digest of the uncovered packet. Finally, the digest is used to verify the validity of the provided TLS-N signature. If the verification result matches the sender device’s address, it means the reported relay server is innocent because the reported packet is sent by the legitimate sender. The smart contract will delete the record in the report pending list. Otherwise, the rebutting fails, and the record remains in the pending list.

5.7.3 Executing

Since the reporting notification takes time to broadcast, a grace period is given for the reported relay server to respond. The period is measured with the number of newly generated blocks because the generation of new blocks means most nodes in

the network reach a consensus on the smart contracts' execution. If the reporting record remains after the grace period, the IoT device which initiates the reporting can execute the penalty by calling the executing function in the smart contract. The smart contract will traverse the pending records to look for the one indexed by the transaction number and the serial number provided by the calling device. If the record exists and has the block number that is old enough, the relay server's account will be removed from the *serverInfo* list and its deposit in the smart contract account will be transferred to the reporting device's account. The smart contract firstly traverses the report pending list to check the existence of the referred reporting record. Then, it calculates the number of blocks that are newly generated after the reporting. If the difference is larger than the pre-defined grace period, the smart contract will execute the penalty that means the relay server's deposit stored inside the smart contract account will be transferred to the reporting IoT device's account. Even if the attacker can rejoin the relay network by creating another account, the nonnegligible financial loss makes further attacks unworthy.

5.8 Security Analysis

5.8.1 Possible Attacks

5.8.1.1 Random Scanning Attack

In a random scanning attack, the attacker scans for open ports as other botnet malwares do without joining the relay sharing system. Since IoT devices are shielded by NAT gateways, scanning traffic from the Internet will not be able to reach the IoT devices. Even if the attacker successfully compromises devices located in the same subnet, the non TLS-N traffic will be discarded by the packet filter that resides in the

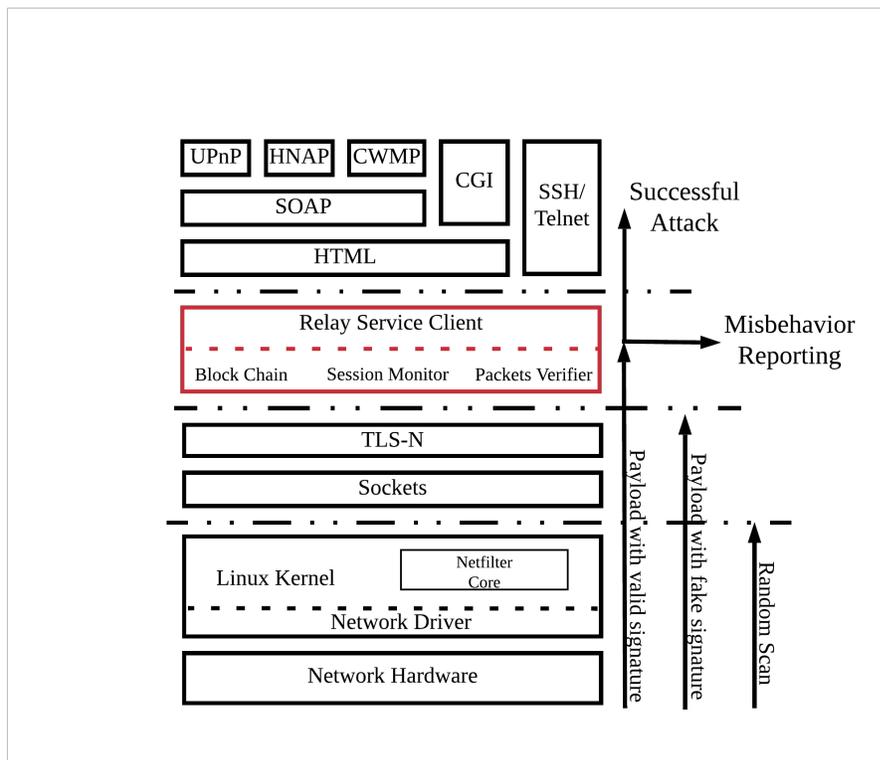


Figure 5.6. Demonstration of possible attacks..

IoT devices' operating system kernel. As depicted in Figure 5.6, this type of attack will never get a chance to exploit any vulnerabilities in the application layer.

5.8.1.2 Attacks against the Relay Server

In the centralized relay model, the relay server becomes an attractive target for attackers because once it gets compromised, attackers can efficiently access millions of its connected IoT devices. Differently, in our relay sharing system, attacking relay servers is much less efficient because each relay server only connects to a small number of IoT devices. In consequence, attackers need to spend significantly more efforts to take down many relay servers with different software implementations, while obtaining much less retribution. Even if some relay servers get compromised, they will be detected and excluded from the relay system when they are utilized for launching attacks.

5.8.1.3 Malicious Relay Server Attack

In this kind of attack, the malicious relay server attacks its connected IoT devices by delivering packets containing attacking vectors. According to our threat model, the malicious relay server must sign the packets with its blockchain private key to get it accepted by the target IoT device. As shown in Figure 5.6, the signed packet is then passed to our relay sharing client middleware. The middleware may be vulnerable which means a malicious relay server does have the chance to successfully bypass the middleware's inspection and takes down the device without getting reported. However, as one of the core benefits of our work, we propose to deter this kind of attack by imposing "*economic risk*" instead of relying on the unrealistic perfect software implementation. Launching attacks inevitably requires the target's platform

information which is acquired by sending some probing packets. If the target is not vulnerable to the probing attack, the unauthorized packets will be reported, which results in the malicious relay server losing all its deposit. Because the packet does not originate from the controller, the malicious relay server is not able to prove its innocence through the rebutting process. Although the attacker can rejoin RS-IoT with a new account, the risk of losing deposit still exists. Also, compared with the random scanning attack, it's very slow to traverse IoT devices on the RS-IoT platform by passively waiting to be commissioned. Finally, attackers get discouraged because this type of attack is not only risky but also inefficient.

5.8.2 Fairness Analysis

Considering there is no trust between relay users and servers, fairness of the service trading platform is required to prevent cheating behaviors of either party. We enumerate all possible cheating scenarios and show how to deal with them by using our proof-of-delivery scheme.

5.8.2.1 Cheating

First, the relay user has the incentive to cheat by denying that they have already received the relayed packet from the relay server. According to the commitment procedure described in Section 5.6, this can be achieved by sending an incorrect commitment back to the relay server. For this kind of cheating, the relay server cannot verify the commitment B and B' and thus will not reveal the cover stream key PN . Without PN , the cheating IoT device is not able to extract the desired content, which is equivalent to receiving nothing. Hence, the free ride is impossible due to the packet covering conducted by the relay server.

Second, the relay server has the motivation to reap without sowing. That is, it may deliver incomplete packets to IoT devices to reduce the cost. Since the delivered packet is covered by the cover stream, IoT cannot verify its integrity before the relay server reveals the cover stream key PN . However, the byte selecting list Ra' is not known to the relay server, and it has no idea about which bytes will be selected for composing the commitment. As a result, when an IoT device generates the commitment B' on the incompletely delivered packet, the relay server has no method to figure out a PN that can satisfy the relation of $Cover_{PN}(B) = B'$. Then, there will be no way to pass the checking of proof-of-delivery by the smart contract to obtain the service fee.

5.8.2.2 Malicious Reporting

Since joining the system as an IoT device requires no deposit, attackers who want to destroy the system may register a large amount of IoT device accounts and use them to report benign relay servers maliciously. Although we design the rebutting scheme, this method may still be able to overload relay servers and undermine the system's performance.

We prevent this kind of abuse by utilizing the high-cost nature of on-blockchain function calls, which is usually regarded as a problem, as we discussed before. According to our description of the reporting process, IoT devices need to pass the whole packet as one of the arguments to the function call, which will be very expensive considering the data storage and processing price on Ethereum [221]. For normal reporting of real malicious packets, this cost will be made up of the confiscated deposit of the relay server. However, if an IoT device reports a benign packet, there will be no makeup. So, malicious reporting leads to a high cost and is quite uneconomic.

5.9 Experiment

To validate the efficiency and usability of RS-IoT, we deploy the proposed RS-IoT on the Ethereum rinkeby testnet with solidity script language of version 0.4.21. After that, we use three Raspberry PIs with Geth (Golang Ethereum Client) and the web3.py package installed as an emulation of the relay server, IoT, and controller, respectively. Each of the raspberry PIs has one account set up in its geth client, and is topped up with test ethers from the rinkeby Faucet. To minimize the execution cost, we avoid using local variables and store all intermediate results in a memory location. We set the commitment length N to 32 bytes to avoid exceeding block gas limit. Based on the cost of gas defined in the Ethereum yellow paper [221], we can accurately evaluate the execution cost of every function that we use. Also, with the gas price set to 2 Gwei (1 Ether equals to $1 * E^9$ Gwei) and an Ether price of \$135, we can convert the execution cost to USD as listed in Table 5.3.

Table 5.3. Contracts Execution Cost..

Entity:Function	Cost in Gas	Cost in USD
<i>D</i> :register	47k	0.012
<i>C</i> :register	22k	0.005
<i>R</i> :register	40k	0.01
<i>D</i> :service request	1.8k	0.0003
<i>D</i> :service select	14.3k	0.003
<i>D</i> :service confirm	22.8k	0.005
<i>D</i> :commitment	175k	0.046
<i>C</i> :commitment	151k	0.039
<i>R</i> :commitment verify	40k	0.01
<i>C, D, R</i> :decommission	12k	0.003
<i>D</i> :execute	8k	0.002
Registration Total	109k	0.03
Commission Total	32.6k	0.009
Commit Total	366k	0.10

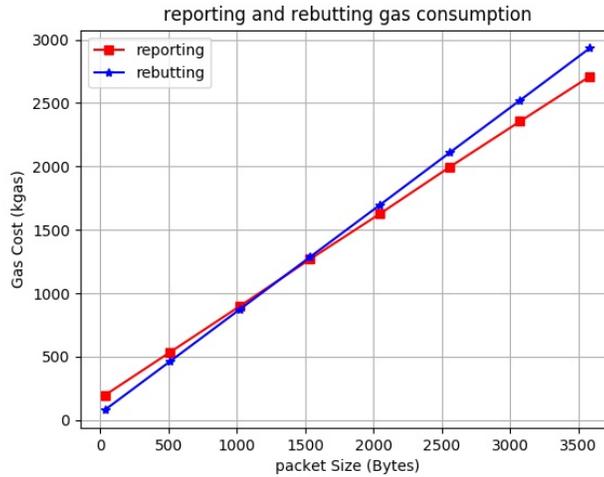


Figure 5.7. Reporting & Rebutting Gas Cost..

As we can see, since all the operations of the RS-IoT are asynchronous and none of them exceed the Ethereum gas limit (average 3,000,000 gas per block), there is no limitation on the number of concurrent online relay sessions. From the usage perspective, registration costs 109k Gas in total, which is a one-time cost. Commission/decommission cost 32.6k gas, but it only happens when the IoT device switches to a new relay server. Though the price for commitment is high, asynchronous verification as mentioned in Section 5.6.2.4 makes it unnecessary to be called for each delivered packet. Instead, by caching the commitment transactions, verification can be conducted in arbitrary long packet intervals as long as no disputation emerges. Thus, the cost is amortized as affordable.

The costs of reporting and rebutting scales along with the size of the reported packet are shown in Figure 5.7. Though the gas limit of one block is about 7 million gas, operations that consume more than 3.5 million gas become challenging to process. The largest packet size for successful reporting and rebutting in our experiment is 3.5k bytes.

5.10 Related Work

After the breakout of Mirai botnet, various defense schemes are proposed. They can be categorized into three types: The first type is the honeypot as in [222, 223, 178] which is usually for research purpose only. The honeypot is a computer with publicly accessible IP addresses and powerful traffic monitoring and logging systems that imitates normal IoT devices. It functions as a trap to detect botnet scanning traffic and after that entice the loading of executable malware files by pretending to be compromised. In this way, the honeypot operators can acquire samples of malwares at the first time. Analysis of the sample malware gives security experts clues about the address of the attacking master's command and control server and helps them make security patches. However, because of the lack of reliable firmware update methods, it's hard to push the security patch to vulnerable devices, which undermines the effect of this kind of defense.

The second type is the secure software implementation guideline, and modules like IDS (Intrusion Detection System) designed for IoT devices and IoT connected cloud servers. [224, 225] fall into this type. [224] proposed to enforce security policies on IoT devices to filter out abnormal or unnecessary packets and [225] offers some guidelines from perspectives of network configuration and device deployment. Both of these works require assumptions about the robustness of deployed softwares, which does not always hold.

The last type of defense as in [226, 227, 228] is actually inspired by Mirai botnet malware, which involves discovering vulnerable IoT devices by random scanning. On discovering new targets, they either dig into the system to expel the hidden malware or report the vulnerable device to authorities and the owner. However, this type of defense is limited on its usability that there is normally no reliable way to notify the

owner for discovered vulnerabilities. Purifying the IoT device without getting the owner’s approval causes legal concern.

5.11 Chapter Summary

In this work, we presented a general framework for efficient blockchain and IoT service integration. To solve the high cost and overhead of on-blockchain operations, we proposed the distributed architecture to host IoT services on third-party servers and use the blockchain and the smart contract as a naturally trusted authority to enforce fairness and punish attackers. We applied this architecture on the designed of RS-IoT, a blockchain-assisted IoT accessing system. RS-IoT provided secure and robust relay services for IoT users to access their IoT devices that are behind NAT. We utilized “*an economic approach to cyber security*” to deter malicious relay servers and achieved it with our novel proof-of-delivery mechanism. By verifying proofs off-chain, the costs and throughput issues of blockchain are overcome. We demonstrated the cost efficiency of our design with our prototype implementation on the Ethereum testnet.

Chapter 6

DISSERTATION CONCLUSION

In this dissertation, we systematically study the security issues of smart home IoT devices and systems and propose a series of effective security solutions. Our research reveals the increasingly severe threats of smart home automation platforms, which make it possible for attackers to manipulate IoT devices' actions without compromising them.

Targeting the discovered issues, we develop security schemes for different components of smart home IoT systems that are possibly exploitable by attackers. First, we present HAWatcher, the first semantic-assisted anomaly detection system for the execution engines of smart home automation systems. We innovatively introduce hypothetical correlations as the formal representation of unstructured semantic information. Then, training event logs are used as evidence for testing the hypothetical correlations. Our proposed method can effectively reduce the number of false alarms by more accurately profiling smart home IoT deployments' normal behaviors. Then, we explore the IoT messaging protocol and discover new attack vectors. The revealed vulnerabilities can be exploited by attackers to stealthily delay IoT event and command messages and manipulate the execution of automation rules. Targeting this, we design a series of counter measurements and share our findings with popular IoT vendors to help them make quick security patches. Based on our investigation of IoT messaging protocol and service topology, we propose the blockchain-based relay sharing system. The proposed system can provide a reliable message forwarding service

for users to remotely access their IoT devices. We utilize blockchain’s inherent, immutable feature to impose financial punishment on malicious or dishonest behaviors. We test our prototype system on Ethereum [221] testnet, and the results show that our system only slightly increases the cost and overhead of IoT access. Finally, we explore the defense of audio adversarial example attacks targeting automatic speech recognition models. We develop an effective detector by comparing audio samples’ transcription results on multiple recognition models. The detector achieves higher than 98% accuracy on a large dataset containing more than 1000 AE samples, which can be used to prevent IoT devices from being exploited by malicious AE voice commands.

Looking ahead, we believe that we can make deeper exploration based on our current works. For example, audio adversarial examples that can transfer among multiple speech recognition models have emerged, and we need to enhance our detector by exploring other weaknesses of adversarial examples. Also, we find there are several new research directions that remain unexplored. First, smart home IoT systems are shattered with more and more platforms and communication channels being adopted in a single household. The complicated smart home IoT topology brings significant difficulties for common users to understand and configure appropriate access control policies. We need to provide common users an easier way to visualize the effect of their deployed access control policies and fix those risking settings before damages are caused.

BIBLIOGRAPHY

- [1] V. Reports, “Global internet of things (iot) market size,” 2019.
- [2] S. Sinha, “State of iot 2021: Number of connected iot devices growing 9% to 12.3 b,” <https://iot-analytics.com/number-connected-iot-devices/>, 2021.
- [3] Statista, “Smart home device household penetration in the united states in 2019 and 2021,” 2021, <https://www.statista.com/statistics/1247351/smart-home-device-us-household-penetration/>.
- [4] “Do more with google home,” 2021.
- [5] “Alexa smart home,” 2021.
- [6] “Smarthings,” 2019, <https://www.smarthings.com>.
- [7] E. Fernandes, J. Jung, and A. Prakash, “Security analysis of emerging smart home applications,” in *IEEE Symposium on Security and Privacy (S&P)*, 2016, pp. 636–654.
- [8] W. Zhou *et al.*, “Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms,” in *28th USENIX Security Symposium (USENIX Security)*, 2019, pp. 1133–1150.
- [9] Y. Jia *et al.*, “Burglars’ iot paradise: Understanding and mitigating security risks of general messaging protocols on iot clouds,” in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 465–481.
- [10] B. Bracken, “Fbi warn hackers are using hijacked home security devices for swatting,” 2020.
- [11] A. Viderberg, “Security evaluation of smart door locks,” 2019.
- [12] M. Fuller, M. Jenkins, and K. Tjølsen, “Security analysis of the august smart lock,” *en. In:()*, p. 17, 2017.
- [13] M. Antonakakis *et al.*, “Understanding the mirai botnet,” in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110.
- [14] S. Herwig *et al.*, “Measurement and analysis of hajime, a peer-to-peer iot botnet,” in *Network and Distributed Systems Security (NDSS) Symposium*, 2019.
- [15] S. Birnbach, S. Eberz, and I. Martinovic, “Peeves: Physical event verification in smart homes,” in *Proceedings of the ACM Conference on Computer & Communications Security (CCS)*, 2019, pp. 1455–1467.
- [16] I. Ilascu, “D-link leaves severe security bugs in home router unpatched,” 2020.

- [17] R. van der Meulen and J. Rivera, “Gartner says a typical family home could contain more than 500 smart devices by 2022,” Tech. Rep., 2014, <http://www.gartner.com/newsroom/id/2839717>.
- [18] A. Homekit, “Homekit-apple developer,” 2019, <https://www.apple.com/ios/home/>.
- [19] K. Kreuzer, “Openhab-empowering the smart home,” 2013.
- [20] “It’s too cold,” 2018, <https://github.com/SmartThingsCommunity/SmartThingsPublic/tree/master/smartapps/smartthings/its-too-cold.src>.
- [21] W. Ding and H. Hu, “On the safety of iot device physical interaction control,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer & Communications Security (CCS)*, 2018, pp. 832–846.
- [22] Z. B. Celik, G. Tan, and P. D. McDaniel, “Iotguard: Dynamic enforcement of security and safety policy in commodity iot.” in *Network and Distributed System Security Symposium (NDSS)*, 2019.
- [23] Z. B. Celik, P. McDaniel, and G. Tan, “Soteria: Automated iot safety and security analysis,” in *2018 USENIX Annual Technical Conference (USENIX ATC)*, 2018, pp. 147–158.
- [24] J. Choi *et al.*, “Detecting and identifying faulty iot devices in smart home with context extraction,” in *48th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018.
- [25] A. K. Sikder, H. Aksu, and A. S. Uluagac, “6thsense: A context-aware sensor-based attack detector for smart devices,” in *26th USENIX Security Symposium (USENIX Security)*, 2017, pp. 397–414.
- [26] S. Munir and J. A. Stankovic, “Failuresense: Detecting sensor failure using electrical appliances in the home,” in *11th International Conference on Mobile Ad Hoc and Sensor Systems (MobiHoc)*, 2014, pp. 73–81.
- [27] W. Lee, S. J. Stolfo, and K. W. Mok, “A data mining framework for building intrusion detection models,” in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 1999, pp. 120–132.
- [28] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.
- [29] P. A. Kodeswaran, R. Kokku, S. Sen, and M. Srivatsa, “Idea: A system for efficient failure management in smart iot environments,” in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2016, pp. 43–56.

- [30] J. Ye, G. Stevenson, and S. Dobson, "Fault detection for binary sensors in smart home environments," in *Pervasive Computing and Communications (PerCom)*, 2015, pp. 20–28.
- [31] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2019.
- [32] N. ElHady and J. Provost, "A systematic survey on sensor failure detection and fault-tolerance in ambient assisted living," *Sensors*, vol. 18, no. 7, p. 1991, 2018.
- [33] T. W. Hnat *et al.*, "The hitchhiker's guide to successful residential sensing deployments," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2011, pp. 232–245.
- [34] "Tons of issues with smartthings," 2016, https://www.reddit.com/r/SmartThings/comments/4463eo/anyone_else_having_tons_of_issues_with_smartthings/.
- [35] "Door knocker going crazy," 2016, <https://community.smartthings.com/t/door-knocker-going-crazy/55570>.
- [36] "Motion detection false positive," 2018, <https://community.smartthings.com/t/motion-detection-false-positive/119816>.
- [37] "Motion sensor stuck on motion," 2017, <https://community.smartthings.com/t/motion-sensors-stuck-on-motion/46761>.
- [38] "What's wrong with smartthings now?" 2017, <https://community.smartthings.com/t/whats-wrong-with-smartthings-now-poltergeist-events/83889>.
- [39] "Are the poltergeists back?" 2017, <https://community.smartthings.com/t/october-2017-are-the-poltergeists-back-devices-randomly-turning-on/101402>.
- [40] "Undesired poltergeist lighting effect," 2017, <https://community.smartthings.com/t/undesired-poltergeist-lighting-effect/24132>.
- [41] "When st glitches become major safety fire hazard," 2016, <https://community.smartthings.com/t/when-st-glitches-become-major-safety-fire-hazard/43109>.
- [42] "Mobile device presence update delay," 2017, <https://community.smartthings.com/t/mobile-device-presence-update-delay/98672>.
- [43] "Known mobile presence issues and faq," 2019, <https://support.smartthings.com/hc/en-us/articles/204744424-Known-mobile-presence-issues-and-FAQ>.

- [44] “Tplink smart wi-fi plug fail,” 2017, <https://www.h3-digital.com/smarthomeblog/2017/5/23/tplink-smart-wi-fi-plug-fail>.
- [45] “Smart plug clicks but no power,” 2018, <https://community.smartthings.com/t/smart-plug-clicks-but-no-power/115252>.
- [46] S. P. Kavalaris and E. Serrelis, “Security issues of contemporary multimedia implementations: The case of sonos and sonosnet,” in *The International Conference in Information Security and Digital Forensics (ISDF)*, 2014, pp. 63–74.
- [47] S. Notra *et al.*, “An experimental study of security and privacy risks with emerging household appliances,” in *IEEE conference on communications and network security (CNS)*, 2014.
- [48] V. Sivaraman, D. Chan, D. Earl, and R. Boreli, “Smart-phones attacking smart-homes,” in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2016, pp. 195–200.
- [49] H. Liu, T. Spink, and P. Patras, “Uncovering security vulnerabilities in the belkin wemo home automation ecosystem,” in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2019, pp. 894–899.
- [50] L. Russell, “Wireless security monitoring versus a cellular jammer,” 2014.
- [51] M. Fránik and M. Čermák, “Serious flaws found in multiple smart home hubs: Is your device among them?” 2020, <https://www.welivesecurity.com/2020/04/22/serious-flaws-smart-home-hubs-is-your-device-among-them/>.
- [52] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O’Flynn, “Iot goes nuclear: Creating a zigbee chain reaction,” in *2017 IEEE Symposium on Security and Privacy (S&P)*, 2017, pp. 195–212.
- [53] K. Kapitanova *et al.*, “Being smart about failures: assessing repairs in smart homes,” in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (UbiComp)*, 2012, pp. 51–60.
- [54] Y. Tian *et al.*, “Smartauth: User-centered authorization for the internet of things,” in *26th USENIX Security Symposium (USENIX Security)*, 2017, pp. 361–378.
- [55] H. Chi, Q. Zeng, X. Du, and J. Yu, “Cross-app interference threats in smart homes: Categorization, detection and handling,” in *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020, pp. 411–423.

- [56] —, “Cross-app interference threats in smart homes: Categorization, detection and handling,” *arXiv*, pp. arXiv–1808, 2018.
- [57] J. Han *et al.*, “Do you feel what i hear? enabling autonomous iot device pairing using different sensor types,” in *2018 IEEE Symposium on Security and Privacy (S&P)*, 2018, pp. 836–852.
- [58] “Lights follows me,” 2015, <https://github.com/SmartThingsCommunity/SmartThingsPublic/tree/master/smartapps/smartthings/light-follows-me.src>.
- [59] W. Zhang *et al.*, “Homonit: Monitoring smart home apps from encrypted traffic,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1074–1088.
- [60] Y. J. Jia *et al.*, “Contexiot: Towards providing contextual integrity to appified iot platforms,” in *Proceedings of The Network and Distributed System Security Symposium (NDSS)*, 2017.
- [61] “Light up the night,” 2018, <https://github.com/infinitywings/SmartThingsPublic/blob/master/smartapps/smartthings/light-up-the-night.src/light-up-the-night.groovy>.
- [62] J. Ye, G. Stevenson, and S. Dobson, “Detecting abnormal events on binary sensors in smart home environments,” in *Pervasive and Mobile Computing*, 2016, pp. 32–49.
- [63] L. Cheng, K. Tian, and D. D. Yao, “Orpheus: Enforcing cyber-physical execution semantics to defend against data-oriented attacks,” in *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC)*, 2017, pp. 315–326.
- [64] F. Pukelsheim, “The three sigma rule,” *The American Statistician*, vol. 48, no. 2, pp. 88–91, 1994.
- [65] G. F. Jenks, “The data model concept in statistical mapping,” *International yearbook of cartography*, vol. 7, pp. 186–190, 1967.
- [66] B. Dent, “Cartography–thematic map design. 1999,” pp. 147–149.
- [67] C. Fu, Q. Zeng, and X. Du, “Hawatcher: Semantics-aware anomaly detection for appified smart homes (technical report),” 2020, <https://github.com/infinitywings/HAWatcher.git>.
- [68] “Smartthings capabilities,” 2018, <https://smartthings.developer.samsung.com/docs/api-ref/capabilities.html>.

- [69] T. Mikolov *et al.*, “Distributed representations of words and phrases and their compositionality,” in *Advances in neural information processing systems (NeurIPS)*, 2013, pp. 3111–3119.
- [70] “Smartapp execution scheduling,” <https://docs.smartthings.com/en/latest/ref-docs/smartapp-ref.html#smartapp-run-in>.
- [71] R. A. Fisher, “Statistical methods for research workers,” in *Breakthroughs in statistics*. Springer, 1992, pp. 66–70.
- [72] K. Calder, “Statistical inference,” *New York: Holt*, 1953.
- [73] D. J. Cook *et al.*, “Mavhome: An agent-based smart home,” in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2003, pp. 521–524.
- [74] D. J. Cook, A. S. Crandall, B. L. Thomas, and N. C. Krishnan, “Casas: A smart home in a box,” *Computer*, vol. 46, no. 7, pp. 62–69, 2013.
- [75] R. Agrawal, R. Srikant *et al.*, “Fast algorithms for mining association rules,” in *Proceedings of 20th International Conference of Very Large Data Bases (VLDB)*, vol. 1215, 1994, pp. 487–499.
- [76] B. Schölkopf *et al.*, “Estimating the support of a high-dimensional distribution,” *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.
- [77] S. S. Khan and M. G. Madden, “One-class classification: taxonomy of study and review of techniques,” *The Knowledge Engineering Review*, vol. 29, no. 3, pp. 345–374, 2014.
- [78] “Pymining - a collection of data mining algorithms in python,” 2020, <https://github.com/bartdag/pymining>.
- [79] J. Inoue *et al.*, “Anomaly detection for a water treatment system using unsupervised machine learning,” in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017, pp. 1058–1065.
- [80] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [81] “Your hotspot is a presence detector,” http://ficara.altervista.org/?p=3744&doing_wp_cron=1591921359.5108020305633544921875, 2017.
- [82] “Troubleshooting: Smartthings multipurpose sensor is stuck on ”open” or ”closed”,” 2020, <https://support.smartthings.com/hc/en-us/articles/200955940-Troubleshooting-SmartThings-Multipurpose-Sensor-is-stuck-on-open-or-closed->.

- [83] “Motion sensors losing connectivity,” 2017, <https://community.smartthings.com/t/smartthings-motion-multi-sensors-losing-connectivity-on-a-daily-basis/84512>.
- [84] “Command received but not executed,” 2017, <https://community.smartthings.com/t/command-received-but-not-executed/112045>.
- [85] R. Xu *et al.*, “Privacy leakage in smart homes and its mitigation: Ifttt as a case study,” *IEEE Access*, vol. 7, pp. 63 457–63 471, 2019.
- [86] D. T. Nguyen *et al.*, “Iotsan: fortifying the safety of iot systems,” in *Proceedings of the 14th International Conference on emerging Networking Experiments and Technologies (CoNEXT)*, 2018, pp. 191–203.
- [87] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, “Fear and logging in the internet of things,” in *Network and Distributed System Security Symposium (NDSS)*, 2018.
- [88] Q. Zeng *et al.*, “A multiversion programming inspired approach to detecting audio adversarial examples,” in *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 39–51.
- [89] X. Li, Q. Zeng, L. Luo, and T. Luo, “T2Pair: Secure and Usable Pairing for Heterogeneous IoT Devices,” in *Proceedings of the ACM Conference on Computer & Communications Security (CCS)*, 2020.
- [90] H. Chi, Q. Zeng, X. Du, and L. Luo, “PFirewall: Semantics-aware customizable data flow control for home automation systems,” *arXiv preprint arXiv:1910.07987*, 2019.
- [91] A. K. Sikder, L. Babun, H. Aksu, and A. S. Uluagac, “Aegis: a context-aware security framework for smart home systems,” in *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC)*, 2019, pp. 28–41.
- [92] M. Yahyazadeh, P. Podder, E. Hoque, and O. Chowdhury, “Expat: Expectation-based policy analysis and enforcement for appified smart-home platforms,” in *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2019, pp. 61–72.
- [93] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, “Packet-level signatures for smart home devices,” *Signature*, vol. 10, no. 13, p. 54, 2020.
- [94] A. Acar *et al.*, “Peek-a-boo: I see your smart home activities, even encrypted!” in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 207–218.

- [95] Y. Luo *et al.*, “Context-rich privacy leakage analysis through inferring apps in smart home iot,” *IEEE Internet of Things Journal*, 2020.
- [96] N. Apthorpe, D. Reisman, and N. Feamster, “Poster: A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic,” 2016.
- [97] J. Bae, H. Bae, S.-H. Kang, and Y. Kim, “Automatic control of workflow processes using ECA rules,” *IEEE transactions on knowledge and data engineering*, vol. 16, no. 8, pp. 1010–1023, 2004.
- [98] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman, “Practical trigger-action programming in the smart home,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2014, pp. 803–812.
- [99] “Create your own if this then that rule on IFTTT,” <https://ifttt.com/create>, 2019.
- [100] I. Bastys, M. Balliu, and A. Sabelfeld, “If this then what?: Controlling flows in iot apps,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 1102–1119.
- [101] M. Miettinen *et al.*, “Iot sentinel: Automated device-type identification for security enforcement in iot,” in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2177–2184.
- [102] B. Copos, K. Levitt, M. Bishop, and J. Rowe, “Is anybody home? inferring activity from smart home network traffic,” in *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 245–251.
- [103] N. Apthorpe *et al.*, “Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic,” *arXiv preprint arXiv:1708.05044*, 2017.
- [104] N. Apthorpe, D. Y. Huang, and D. Reisman, “Keeping the smart home private with smart (er) iot traffic shaping,” *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 3, pp. 128–148, 2019.
- [105] A. Subahi and G. Theodorakopoulos, “Detecting iot user behavior and sensitive information in encrypted iot-app traffic,” *Sensors*, vol. 19, no. 21, p. 4777, 2019.
- [106] A. Hariri, N. Giannelos, and B. Arief, “Selective forwarding attack on iot home security kits,” in *Computer Security*. Springer, 2019, pp. 360–373.
- [107] B. Visan *et al.*, “Vulnerabilities in hub architecture iot devices,” in *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2017, pp. 83–88.

- [108] N. Apthorpe *et al.*, “Keeping the smart home private with smart (er) iot traffic shaping,” *arXiv preprint arXiv:1812.00955*, 2018.
- [109] J. Narcotta and B. Ablondi, “Strategy analytics: Amazon’s ring claimed top spot in global home security camera market in 2020,” 2021.
- [110] D. Kumar *et al.*, “All things considered: an analysis of iot devices on home networks,” in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1169–1185.
- [111] I. RFC793, “Transmission control protocol,” *IETF, September*, 1981.
- [112] “Google cloud iot protocols,” 2019.
- [113] O. Standard, “Mqtt version 3.1. 1,” *URL <http://docs.oasis-open.org/mqtt/mqtt/v3>*, vol. 1, 2014.
- [114] R. Fielding and J. Reschke, “Rfc 7231-hypertext transfer protocol (http/1.1): Semantics and content,” *Internet Engineering Task Force (IETF)*, 2014.
- [115] S. Whalen, “An introduction to arp spoofing,” *Node99 [Online Document]*, April, 2001.
- [116] D. Y. Huang *et al.*, “Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 2, pp. 1–21, 2020.
- [117] “Home security systems market by home type, security, systems, services, region - global forecast 2025,” 2020.
- [118] “Home+,” 2020.
- [119] “Homekit,” 2020.
- [120] G. F. Lyon, *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.
- [121] “Ring security system integration with echo devices,” 2020.
- [122] “Receiving notifications of motion while home and disarmed,” 2018.
- [123] “Arming or disarming your ring alarm with your smart lock,” 2020.
- [124] “Routine to arm monitor based on lock and presence?” <https://community.smarththings.com/t/routine-to-arm-monitor-based-on-lock-and-presence/>, 2018.
- [125] “Piston in a stuck state,” <https://community.webcore.co/t/piston-in-a-stuck-state/>, 2018.

- [126] “Climate control guru,” <https://community.smartthings.com/t/climate-control-guru-beta>, 2017.
- [127] “Door unlock automation? how do you reduce security risks?” <https://community.smartthings.com/t/door-unlock-automation-how-do-you-reduce-security-risks>, 2018.
- [128] “Opening door lock with mobile presence: good idea or not?” <https://community.smartthings.com/t/opening-door-lock-with-mobile-presence-good-idea-or-not/>, 2018.
- [129] “Automation: Front door lock locking when its not supposed to,” <https://community.smartthings.com/t/automation-front-door-lock-locking-when-its-not-supposed-to>, 2013.
- [130] “Have i done it? - thermostat/motion for space heater,” <https://community.webcore.co/t/have-i-done-it-thermostat-motion-for-space-heater/>, 2019.
- [131] Q. Wang *et al.*, “Charting the attack surface of trigger-action iot platforms,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer & Communications Security (CCS)*, 2019, pp. 1439–1453.
- [132] T. OConnor, W. Enck, and B. Reaves, “Blinded and confused: uncovering systemic flaws in device telemetry for smart-home internet of things,” in *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, 2019, pp. 140–150.
- [133] M. Wilhelm, I. Martinovic, J. B. Schmitt, and V. Lenders, “Short paper: Reactive jamming in wireless networks: How realistic is the threat?” in *Proceedings of the fourth ACM conference on Wireless network security*, 2011, pp. 47–52.
- [134] M. Li, I. Koutsopoulos, and R. Poovendran, “Optimal jamming attacks and network defense policies in wireless sensor networks,” in *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 1307–1315.
- [135] R. Negi and A. Rajeswaran, “Dos analysis of reservation based mac protocols,” in *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, vol. 5. IEEE, 2005, pp. 3632–3636.
- [136] D. Nguyen *et al.*, “A real-time and protocol-aware reactive jamming framework built on software-defined radios,” in *Proceedings of the 2014 ACM workshop on Software radio implementation forum*, 2014, pp. 15–22.
- [137] H. Pirayesh and H. Zeng, “Jamming attacks and anti-jamming strategies in wireless networks: A comprehensive survey,” *arXiv preprint arXiv:2101.00292*, 2021.

- [138] B. Yuan, Y. Jia, L. Xing, and D. Zhao, “Shattered chain of trust: Understanding security risks in cross-cloud iot access delegation,” in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1183–1200.
- [139] D. Yu and L. Deng, *Automatic Speech Recognition: A Deep Learning Approach*, 2nd ed., ser. Signals and Communication Technology. London: Springer-Verlag London, 2015.
- [140] InsideRadio, “Microsoft hopes skype sets its smart speaker apart,” http://www.insideradio.com/free/microsoft-hopes-skype-sets-its-smart-speaker-apart/article_df50d874-1f52-11e7-a34f-eb5f9f355c22.html, 2017.
- [141] C. Szegedy *et al.*, “Intriguing properties of neural networks,” *CoRR*, vol. abs/1312.6199, 2013.
- [142] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.
- [143] X. Yuan *et al.*, “Commandersong: A systematic approach for practical adversarial voice recognition,” in *USENIX Security Symposium*. USENIX Association, 2018, pp. 49–64.
- [144] N. Carlini and D. Wagner, “Audio adversarial examples: Targeted attacks on speech-to-text,” in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 1–7.
- [145] R. Taori, A. Kamsetty, B. Chu, and N. Vemuri, “Targeted adversarial examples for black box audio systems,” *CoRR*, vol. abs/1805.07820, 2018.
- [146] M. Alzantot, B. Balaji, and M. Srivastava, “Did you hear that? adversarial examples against automatic speech recognition,” *arXiv preprint arXiv:1801.00554*, 2018.
- [147] Z. Yang, B. Li, P.-Y. Chen, and D. Song, “Towards mitigating audio adversarial perturbations,” *ICLR workshop*, 2018.
- [148] K. Rajaratnam, K. Shah, and J. Kalita, “Isolated and ensemble audio preprocessing methods for detecting adversarial examples against automatic speech recognition,” *arXiv preprint arXiv:1809.04397*, 2018.
- [149] N. Carlini and D. Wagner, “Adversarial examples are not easily detected: Bypassing ten detection methods,” in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017, pp. 3–14.

- [150] A. A. Liming Chen, “N-version programming: A fault-tolerance approach to reliability of software operation,” in *Twenty-Fifth International Symposium on Fault-Tolerant Computing*, 1995.
- [151] N. Papernot *et al.*, “The limitations of deep learning in adversarial settings,” in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, 2016, pp. 372–387.
- [152] N. Papernot, P. McDaniel, and I. Goodfellow, “Transferability in machine learning: from phenomena to black-box attacks using adversarial samples,” *arXiv preprint arXiv:1605.07277*, 2016.
- [153] F. Tramèr *et al.*, “The space of transferable adversarial examples,” *arXiv preprint arXiv:1704.03453*, 2017.
- [154] Y. Liu, X. Chen, C. Liu, and D. Song, “Delving into transferable adversarial examples and black-box attacks,” *CoRR*, vol. abs/1611.02770, 2016.
- [155] J. Monteiro, Z. Akhtar, and T. H. Falk, “Generalizable adversarial examples detection based on bi-model decision mismatch,” *arXiv preprint arXiv:1802.07770*, 2018.
- [156] “Test 10 commandersong aes over iffyteck, google cloud speech and kaldı,” <https://drive.google.com/open?id=1CMtDOIjtBpNrbaTknxSsmB5xjoNbjYXq>, 2018.
- [157] D. Yu and J. Li, “Recent progresses in deep learning based acoustic models,” *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 3, pp. 396–409, 2017.
- [158] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
- [159] A. Graves, A. Mohamed, and G. E. Hinton, “Speech recognition with deep recurrent neural networks,” in *ICASSP*, 2013, pp. 6645–6649.
- [160] H. Sak, A. Senior, and F. Beaufays, “Long short-term memory recurrent neural network architectures for large scale acoustic modeling,” in *Fifteenth annual conference of the international speech communication association*, 2014.
- [161] X. Li and X. Wu, “Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition,” *CoRR*, vol. abs/1410.4281, 2014.
- [162] K. J. Lang, A. Waibel, and G. E. Hinton, “A time-delay neural network architecture for isolated word recognition,” *Neural Networks*, vol. 3, no. 1, pp. 23–43, 1990.

- [163] O. Abdel-Hamid, L. Deng, and D. Yu, “Exploring convolutional neural network structures and optimization techniques for speech recognition,” in *INTERSPEECH*. ISCA, 2013, pp. 3366–3370.
- [164] L. Tóth, “Modeling long temporal contexts in convolutional neural network-based phone recognition,” in *ICASSP*, 2015.
- [165] T. Sercu and V. Goel, “Dense prediction on sequences with time-dilated convolutions for speech recognition,” *CoRR*, vol. abs/1611.09288, 2016.
- [166] A. Y. Hannun *et al.*, “Deep speech: Scaling up end-to-end speech recognition,” *CoRR*, vol. abs/1412.5567, 2014.
- [167] D. Amodei *et al.*, “Deep speech 2: End-to-end speech recognition in english and mandarin,” in *International Conference on Machine Learning*, 2016, pp. 173–182.
- [168] “Google ai blog: The neural networks behind google voice transcription,” <https://ai.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html>, 2015.
- [169] “Wikipage of phonetic algorithm,” https://en.wikipedia.org/wiki/Phonetic_algorithm, 2017.
- [170] “Jaro–winkler distance,” https://en.wikipedia.org/wiki/Jaro%E2%80%9393Winkler_distance, 2018.
- [171] “Deepspeech github repository,” <https://github.com/mozilla/DeepSpeech>, 2018.
- [172] “Google cloud speech homepage,” <https://cloud.google.com/speech-to-text>, 2018.
- [173] “Amazon transcribe homepage,” <https://aws.amazon.com/transcribe>, 2018.
- [174] “Librispeech asr corpus,” <http://www.openslr.org/12>, 2015.
- [175] “Download page for common voice dataset,” <https://voice.mozilla.org/en/data>, 2018.
- [176] “Jaccard index,” https://en.wikipedia.org/wiki/Jaccard_index, 2018.
- [177] Gargner, “Gartner identifies top 10 strategic iot technologies and trends,” 2018.
- [178] M. Antonakakis *et al.*, “Understanding the mirai botnet,” in *USENIX Security Symposium*, 2017, pp. 1092–1110.

- [179] C. Koliass, G. Kambourakis, A. Stavrou, and J. Voas, “Ddos in the iot: Mirai and other botnets,” *Computer*, vol. 50, no. 7, pp. 80–84, 2017.
- [180] P. Kim, “Pwning the dlink 850l routers and abusing the mydlink cloud protocol,” 2017.
- [181] B. Zhang, “Awesome iot hacks,” 2019.
- [182] J. E. Dunn, “Researcher reveals d-link router holes that might never be patched,” Sep 2017.
- [183] H. Tschofenig and S. Farrell, “Report from the internet of things software update (iotsu) workshop 2016,” Tech. Rep., 2017.
- [184] B. Schneier, “The internet of things is wildly insecure—and often unpatchable,” *Schneier on Security*, vol. 6, 2014.
- [185] T. S. Bernard, T. Hsu, N. Perlroth, and R. Lieber, “Equifax says cyberattack may have affected 143 million in the us,” *The New York Times*, p. A1, 2017.
- [186] R. Walters, “Cyber attacks on us companies in 2014,” *The Heritage Foundation*, vol. 4289, pp. 1–5, 2014.
- [187] B. Zhang *et al.*, “The cloud is not enough: Saving iot from the cloud,” in *7th {USENIX} Workshop on Hot Topics in Cloud Computing*, 2015.
- [188] J. Singh *et al.*, “Twenty security considerations for cloud-supported internet of things,” *IEEE Internet of things Journal*, vol. 3, no. 3, pp. 269–284, 2016.
- [189] D. Happ *et al.*, “Meeting iot platform requirements with open pub/sub solutions,” *Annals of Telecommunications*, vol. 72, no. 1-2, pp. 41–52, 2017.
- [190] V. Buterin *et al.*, “A next-generation smart contract and decentralized application platform,” *white paper*, 2014.
- [191] L. Chen, A. Avizienis *et al.*, “N-version programming: A fault-tolerance approach to reliability of software operation,” in *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995.*, IEEE, 1995, p. 113.
- [192] B. Cox *et al.*, “N-variant systems: A secretless framework for security through diversity.” in *USENIX Security Symposium*, 2006, pp. 105–120.
- [193] H. Ritzdorf *et al.*, “Tls-n: Non-repudiation over tls enable ubiquitous content signing.” in *NDSS*, 2018.
- [194] A. Sajid, H. Abbas, and K. Saleem, “Cloud-assisted iot-based scada systems security: A review of the state of the art and future challenges,” *IEEE Access*, vol. 4, pp. 1375–1384, 2016.

- [195] P. Parwekar, “From internet of things towards cloud of things,” in *2nd International Conference on Computer and Communication Technology*. IEEE, 2011, pp. 329–333.
- [196] P. Srivastava and N. Garg, “Secure and optimized data storage for iot through cloud framework,” in *International Conference on Computing, Communication & Automation*. IEEE, 2015, pp. 720–723.
- [197] P. Brody and V. Pureswaran, “Device democracy: Saving the future of the internet of things,” *IBM*, September, 2014.
- [198] A. Panarello *et al.*, “Blockchain and iot integration: A systematic survey,” *Sensors*, vol. 18, no. 8, p. 2575, 2018.
- [199] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the internet of things,” *Ieee Access*, vol. 4, pp. 2292–2303, 2016.
- [200] M. Conoscenti, A. Vetro, and J. C. De Martin, “Blockchain for the internet of things: A systematic literature review,” in *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*. IEEE, 2016, pp. 1–6.
- [201] L. Wu, X. Du, W. Wang, and B. Lin, “An out-of-band authentication scheme for internet of things using blockchain technology,” in *2018 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2018, pp. 769–773.
- [202] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, “Blockchain for iot security and privacy: The case study of a smart home,” in *Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2017, pp. 618–623.
- [203] G. Zyskind, O. Nathan *et al.*, “Decentralizing privacy: Using blockchain to protect personal data,” in *2015 IEEE Security and Privacy Workshops*. IEEE, 2015, pp. 180–184.
- [204] D. Wörner and T. von Bomhard, “When your sensor earns money: exchanging data for cash with bitcoin,” in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. ACM, 2014, pp. 295–298.
- [205] S. Raval, *Decentralized applications: harnessing Bitcoin’s blockchain technology*. ” O’Reilly Media, Inc.”, 2016.
- [206] J. Chen *et al.*, “Certchain: Public and efficient certificate audit based on blockchain for tls connections,” in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2060–2068.

- [207] C. Fromknecht, D. Velicanu, and S. Yakoubov, “Certcoin: A namecoin based decentralized authentication system,” *Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep.*, vol. 6, 2014.
- [208] M. P. Andersen *et al.*, “Wave: A decentralised authorization system for iot via blockchain smart contracts,” 2017.
- [209] A. Z. Ourad, B. Belgacem, and K. Salah, “Using blockchain for iot access control and authentication management,” in *International Conference on Internet of Things*. Springer, 2018, pp. 150–164.
- [210] N. Alexopoulos, S. M. Habib, and M. Mühlhäuser, “Towards secure distributed trust management on a global scale: An analytical approach for applying distributed ledgers for authorization in the iot,” in *Proceedings of the 2018 Workshop on IoT Security and Privacy*. ACM, 2018, pp. 49–54.
- [211] A. Stanciu, “Blockchain based distributed control system for edge computing,” in *2017 21st International Conference on Control Systems and Computer Science*. IEEE, 2017, pp. 667–671.
- [212] E. Androulaki *et al.*, “Hyperledger fabric: a distributed operating system for permissioned blockchains,” in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 30.
- [213] S. Popov, “The tangle,” *cit. on*, p. 131, 2016.
- [214] G. Zyskind, O. Nathan, and A. Pentland, “Enigma: Decentralized computation platform with guaranteed privacy,” *arXiv preprint arXiv:1506.03471*, 2015.
- [215] B. Krebs, “Who makes the iot things under attack,” *Krebs on Security*, 2016.
- [216] D. Ajitomi, H. Kawazoe, K. Minami, and N. Esaka, “A cost-effective method to keep availability of many cloud-connected devices,” in *2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 2015, pp. 1–8.
- [217] “Aws iot developer guide,” 2019.
- [218] C. Bradford, “7 Most Infamous Cloud Security Breaches,” 2018.
- [219] P. Kim, “Security research by pierre multiple vulnerabilities found in wireless ip cameras,” 2017.
- [220] M. J. Dworkin, “Sha-3 standard: Permutation-based hash and extendable-output functions,” *Tech. Rep.*, 2015.
- [221] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

- [222] Y. M. P. Pa *et al.*, “Iotpot: analysing the rise of iot compromises,” *EMU*, vol. 9, p. 1, 2015.
- [223] T. Luo *et al.*, “Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices,” *Black Hat*, 2017.
- [224] D. Barrera, I. Molloy, and H. Huang, “Idiot: Securing the internet of things like it’s 1994,” *arXiv preprint arXiv:1712.03623*, 2017.
- [225] B. R. Payne and T. T. Abegaz, “Securing the internet of things: Best practices for deploying iot devices,” pp. 493–506, 2018.
- [226] C. Cao *et al.*, “Hey, you, keep away from my device: remotely implanting a virus expeller to defeat mirai on iot devices,” *arXiv preprint arXiv:1706.05779*, 2017.
- [227] F. Lauria, “How to footprint, report and remotely secure compromised iot devices,” *Network Security*, vol. 2017, no. 12, pp. 10–16, 2017.
- [228] D. Goodin, “Brickerbot, the permanent denial-of-service botnet, is back with a vengeance,” *Ars Technica*, 2017.