Mining IoT Network Traffic in Smart Homes:

Traffic Measurement, Pattern Recognition, and Security Applications

by

Yinxin Wan

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved July 2023 by the
Graduate Supervisory Committee:

ARIZONA STATE UNIVERSITY

August 2023

ABSTRACT

Recent advances in cyber-physical systems, artificial intelligence, and cloud computing have driven the widespread deployment of Internet-of-Things (IoT) devices in smart homes. However, the spate of cyber attacks exploiting the vulnerabilities and weak security management of smart home IoT devices have highlighted the urgency and challenges of designing efficient mechanisms for detecting, analyzing, and mitigating security threats towards them.

In this dissertation, I seek to address the security and privacy issues of smart home IoT devices from the perspectives of traffic measurement, pattern recognition, and security applications. I first propose an efficient multidimensional smart home network traffic measurement framework, which enables me to deeply understand the smart home IoT ecosystem and detect various vulnerabilities and flaws. I further design intelligent schemes to efficiently extract security-related IoT device event and user activity patterns from the encrypted smart home network traffic. Based on the knowledge of how smart home operates, different systems for securing smart home networks are proposed and implemented, including abnormal network traffic detection across multiple IoT networking protocol layers, smart home safety monitoring with extracted spatial information about IoT device events, and system-level IoT vulnerability analysis and network hardening.

DEDICATION

To my family.

ACKNOWLEDGMENTS

First and foremost, I would like to express my profound gratitude to my two advisors, Dr. Guoliang Xue and Dr. Kuai Xu, for their consistent support, invaluable guidance, and continuous encouragement throughout my Ph.D. study. Without their insightful feedback and generous help, I could not have completed this dissertation. I believe that the knowledge and skills I have gained under their expert supervision will continue to shape my personal and professional growth for the remainder of my life.

I would like to extend my heartfelt thanks to Dr. Yezhou Yang and Dr. Yanchao Zhang for serving on my dissertation committee. Their constructive feedback and suggestions have helped me a lot.

I am honored to have worked with an incredible group of coauthors and collaborators. I would like to express my gratitude to Dr. Feng Wang for her generous help, valuable insights, and guidance. I am also grateful to Dr. Kaiping Xue, who led me onto the path of academic research. I would like to thank Dr. Ruozhou Yu for his kind assistance and support throughout my Ph.D. studies, career, and personal life. I would like to acknowledge my colleagues and friends: Alena Chang, Dr. Xiao Chen, Dr. Xinxin Feng, Dr. Jiangping Han, Jiawei Li, Xuanli Lin, Abdulhakim Sabur, Hang Wang, Dr. Haiqin Wu, and Yanbo Zhang for their help.

I want to truly thank my family for their love and encouragement. I am very thankful to my parents, Qin Gu and Jianyong Wan for their unconditional love and uninterrupted support throughout my life. I am also grateful to my friends who have consistently offered their encouragement and unwavering belief in me over time.

I gratefully acknowledge the financial support I received from NSF grants 2007469, 2007083, 1816995, 1717197, and 1704092.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

The recent decade has witnessed the rapid development and wide deployment of Internet-of-Things (IoT) devices in different aspects of our lives, such as home automation, smart city, connected healthcare, industry 4.0, etc. Among them, the smart home is one of the most important applications of IoT technology [4, 136, 137, 151]. Heterogeneous smart home IoT devices are now available on the market and have driven various innovative services to bring convenience to users.

However, weakly protected and coarsely managed smart home IoT devices can be easily targeted by cyber attacks [5, 9, 42, 70, 113, 114, 127, 159]. The pervasive smart home IoT devices can be turned into bots for launching distributed denial-of-service (DDoS) attacks [9, 57, 70] or can be compromised for projecting cyber security hazards and physical risks, e.g., unlocking the front door of a smart home via a hacked smart lock [47, 58, 113, 114, 127].

In this dissertation, we focus on addressing the security and privacy challenges of modern smart home systems by mining IoT network traffic in smart homes. We start from understanding the communication model of smart home networks and analyzing network behaviors of different IoT devices. Specifically, we propose an IoT traffic measurement framework to automatically collect, process, characterize, and profile communication patterns of IoT devices. we build the behavioral profiles of different IoT devices based on a wide spectrum of their traffic features from IP-spatial, temporal, cloud, and internal traffic dimensions. By carefully investigating the smart home network traffic from different dimensions, we also identified several flaws and vulnerabilities.

We next devote our research efforts to extracting critical security-related patterns from the encrypted smart home network traffic, which can help us learn what is happening in smart homes. We identified that each IoT device event will generate a static and unique packet-level signature. However, some device events' signatures may share overlapped network packets, making it challenging to differentiate between them. To address this problem, we design an efficient time-sensitive subsequence matching algorithm which utilizes the critical inter-packet time interval information to accurately extract device event sequences.

We then demonstrate that user activity information can also be inferred from the extracted device event sequences because different user activities will generate unique device event sequences in a smart home deployed with heterogeneous IoT devices. In light of the existence of cyberattacks and device failures which can result in missing and out-of-order IoT device events, we design a polynomial-time approximate matching algorithm to infer what user activities happened in our smart home.

With our extensive knowledge of smart home network communication patterns as well as device event and user activity information, we propose several systems for securing smart home networks. We first design a multi-layer security monitoring framework for addressing the broad attack vectors across the entire IoT protocol stack. The two-stage machine-learning based anomaly detection system designed by us can effectively identify both known attacks and zero-day attacks.

We then reveal that critical spatial information about whether an IoT device event is triggered locally or remotely can be extracted by comprehensively analyzing the network traffic of all devices collected in the home network. We further demonstrate the importance of the spatial information of each IoT device event by exploring its applications in smart home safety monitoring, including abnormal device event detection and home entrance safety monitoring.

In addition, We study the problem of systematically profiling and analyzing IoT system security using the extended weighted attack graph model. From the attacker's perspective, a novel algorithm named `SAT` for computing a shortest attack trace in a weighted attack graph is designed. Our `SAT` algorithm is *robust* as it can properly deal with cycles in attack graphs and is guaranteed to find a shortest attack trace in polynomial time if the attack graph contains at least one attack trace. In case there is no attack trace in the attack graph, `SAT` will stop properly without entering the infinite loop. Our algorithm is also *fast*. It has a worst-case running time of $O(m + n \log n)$, where n and m are the numbers of vertices and edges in the attack graph, respectively.

Based on `SAT`, we examine the network hardening problem from the defender's perspective, where a subset of network elements within a budget constraint is selected to be hardened in order to maximally increase the height of the shortest attack trace in the hardened attack graph. We prove that the network hardening problem is NP-hard. We design an exact algorithm for computing an optimal solution using a novel bounding technique as the baseline. We also design a polynomial-time heuristic algorithm. While the heuristic algorithm does not guarantee to find an optimal solution, our extensive experimental evaluation on different datasets demonstrates that it can efficiently produce relatively good results compared with the exact algorithm.

Part I

Smart Home Network Measurement

Chapter 2

MULTIDIMENSIONAL SMART HOME NETWORK TRAFFIC MEASUREMENT

2.1 Background

Recent advances in embedded systems have enabled the wide deployment of IoT devices in edge networks. The major players in IoT domains have also developed their own smart IoT platforms such as Samsung's SmartThings [119], Google's Nest [51], and Amazon's Alexa [6] to support broad IoT compatibility and rapid application development. These IoT platforms adopt similar system architectures consisting of IoT devices, cloud-based servers, and IoT applications.

These existing IoT platforms are primarily function-driven and feature-driven, thus leaving security and privacy concerns as the secondary or optional goals. As a result, today's IoT devices are often vulnerable to various security threats, and many of them have already been compromised [72, 81]. For example, the insecure configuration and design flaws of IoT devices have contributed to one of the largest botnets, the Mirai botnet [9, 52, 70], which commands and controls over 600,000 IoT devices at its peak. In addition, the coarse access control policy, malicious applications, and exposures in open wireless channels have created broad attack vectors towards heterogeneous IoT devices [15, 42, 58, 114].

Protecting and securing millions of vulnerable IoT devices is a complicated and challenging task. As a recent security evaluation study [5] pointed out, IoT measurement and monitoring is an important early step towards this goal. Specifically, the first step of IoT security lies in measuring, monitoring, and analyzing communication patterns and behavioral profiles of IoT devices. For example, what do remote *hosts* on the Internet talk with the smart speakers or thermostats, at what *time*, for what *reasons*? Answering these questions is of great importance to understanding *if*, *when*, and *how* the connected IoT devices in edge networks are targeted, compromised, and controlled by cyber attacks.

2.2 Smart Home IoT System Communication Model and Network Traffic Collection

As illustrated in Figure 2.1, a smart home IoT device either directly connects to a home router via WiFi or Ethernet, or indirectly through a hub device via low-energy wireless protocols such as Bluetooth, Zigbee, or Z-Wave for Internet access. To control a smart home IoT device such as turning a smart bulb on, a controlling device (with companion apps installed) needs to connect to the Internet through WiFi or cellular network and then authenticate itself with the cloud services. After connections with the cloud servers are established, the controlling devices can send commands to the servers, where the integrity and validity of the commands will be verified. The command messages will be routed inside the cloud service providers' AS (autonomous system) and then forwarded to the IoT devices through the home router. Upon receiving commands from the cloud servers, the IoT devices will validate them and then take the corresponding actions.

When the controlling devices are physically close to IoT devices, they can directly communicate with the IoT devices and send commands using Bluetooth if supported. In fact, we found that August Smart lock's mobile companion app always prefers using the Bluetooth protocol when the controlling device is within the communication range of the lock, even if WiFi or cellular network is available. We also found that

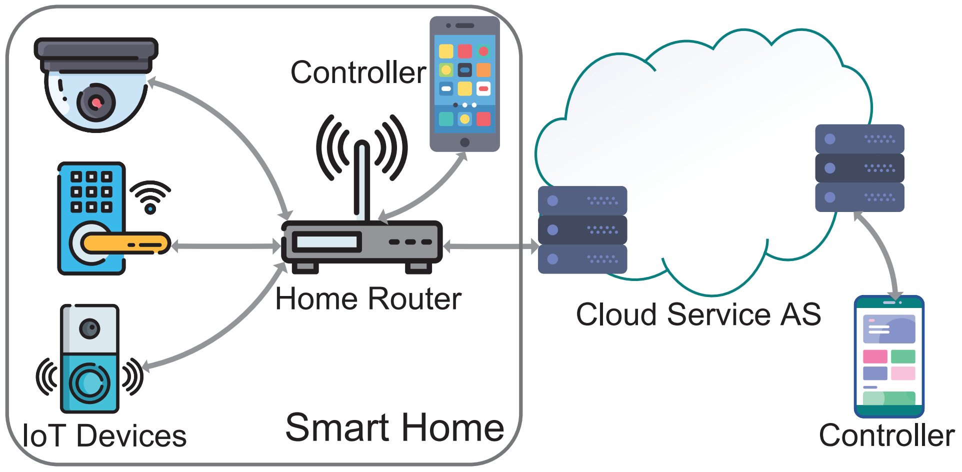


Figure 2.1. Communication Model of the Smart Home IoT System

some IoT devices, such as Reolink cameras, directly communicate with the controlling devices connected to the same home network to exchange commands and data.

As observed from Figure 2.1, the home router is an ideal place for comprehensively collecting the network traffic of IoT devices in smart homes because all IoT devices are directly or indirectly connected to the home router for Internet access. Thus in this study, we advocate a home router-based IoT traffic measurement and monitoring platform for continuously monitoring the incoming and outgoing traffic between smart home networks and the Internet as well as the internal traffic within smart home networks.

More specifically, the home router can see the non-translated incoming and outgoing traffic to and from an IoT device. The internal LAN traffic in the smart home network is also visible to the internal interface of the home router. In addition, the home router-based solutions are transparent to IoT devices and therefore there is no need for the users to install or update additional packages and applications on IoT devices.

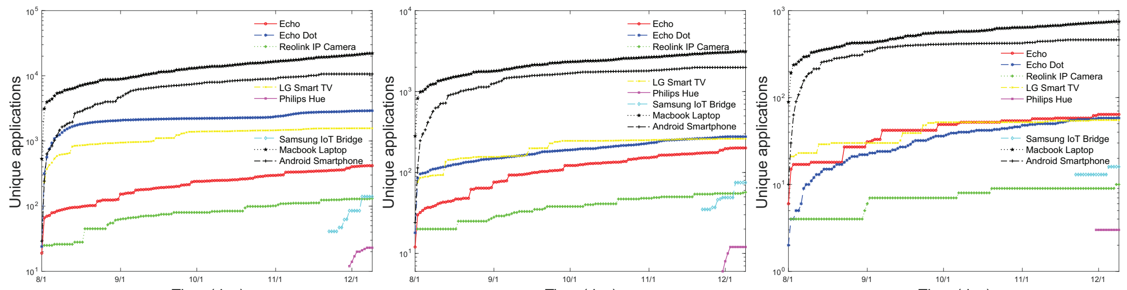
We propose a IoT traffic measurement framework via programmable routers at edge networks where the programmable edge router continuously captures, stores, and analyzes the incoming, outgoing, and internal network traffic flow records of all IoT devices in the edge network. For each flow record, we collect the well-known 5-tuples of a network conversation or session, i.e., source IP address (`srcIP`), source port number (`srcPort`), destination IP address (`dstIP`), destination port number (`dstPort`), and protocol, as well as the start and end timestamps, duration, byte count, and packet count. The availability of millions of network traffic flow data allows us to characterize and model the multidimensional behavioral profiles of heterogeneous IoT devices.

2.3 Multidimensional Behavioral Profiling of IoT Devices

After collecting comprehensive network traffic in smart homes, we further design a multidimensional approach to characterizing the behaviors of IoT devices from IP-spatial, temporal, entropy, cloud, and internal dimensions.

2.3.1 IP-Spatial Behavior of IoT Devices

We first characterize the IP-spatial behaviors of IoT devices by analyzing whom the IoT devices talk to. We propose to aggregate remote IP addresses that IoT devices communicate with into *BGP network prefixes* and *ASNs* in order to gain an in-depth understanding of "clustered" IP-spatial behaviors for IoT devices and make sense of the remote IP addresses. For example, the IP address of the DNS server for Google home smart voice assistant, `8.8.8.8`, is from the BGP prefix `8.0.0.0/9` and ASN 15169 owned by Google based on the latest snapshot of the BGP routing table [97] and the official registry records from Internet assigned numbers authority (IANA). Our experiments following this strategy reveal an interesting observation. Even though most IoT devices communicate with a large number of remote hosts, they typically only engage with a very small subset of BGP network prefixes and ASNs, which are likely from the same server pool by the same service providers for efficient load balancing and content distributions.



(a) IP addresses (b) BGP prefixes (c) ASNs

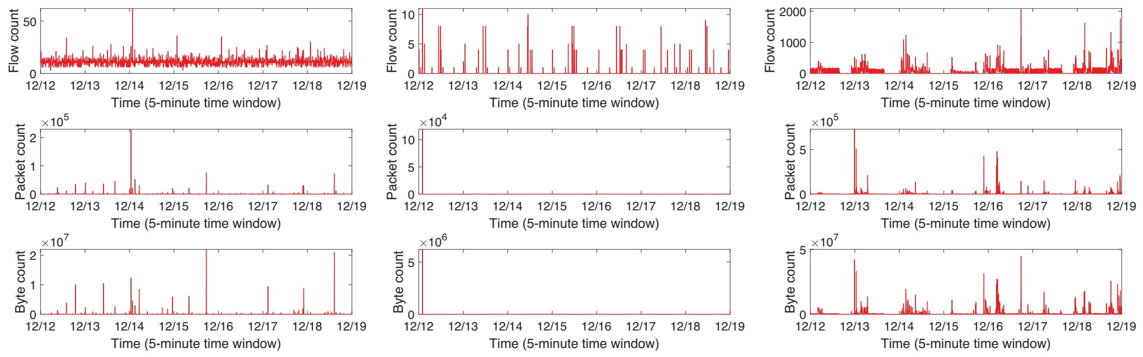
Figure 2.2. The Convergence of IP Addresses, Prefixes, and ASNs for IoT and Non-IoT Devices Over the Longitudinal Measurement Period (Samsung IoT Bridge and Philips Hue Were Added Late)

Figure 2.2 demonstrates the convergence of unique remote IP addresses, their network prefixes, and ASNs for a variety of IoT and non-IoT devices deployed in one edge network over a 4-month time span. This longitudinal measurement study for the IP-spatial behavior confirms that most IoT devices engage with a much smaller set of destination IP addresses, prefixes, and ASNs than smartphones and laptops.

2.3.2 Temporal Behavioral Dynamics of IoT Devices

We study the temporal behavior of IoT devices by measuring the number of distinct time slots in which IoT devices exhibit traffic activities. We select a 5-minute time window in the experiment in order to balance the computation overhead and monitor real-time traffic activities. Figure 2.3 depicts the flow, packet, and byte count of three different kinds of devices over a one-week time span. As shown in Figure 2.3, the Echo

Dot, LG Smart TV, and Macbook Laptop exhibit distinct traffic characteristics over time. These features reflect the activities of different devices. For example, LG smart TV has high peaks in both packet count and byte count at the very beginning, which corresponds to the activity that this device was turned on for network streaming at that time. The diversity of temporal patterns on flow, packet and byte count inspires us to measure and quantify the *variability* over the entire data collection period.



(a) Echo Dot (b) LG Smart TV (c) Macbook Laptop

Figure 2.3. Traffic Characteristics of IoT Devices and Non-IoT Devices Over One-Week Time-Span

For each IoT device d in the edge network, let $W_{d,i}$ denote the number of time windows in which the device d is observed with network traffic on the i-th day. Considering that the connected devices are randomly added into the edge network, we use the average time window μ_d for each device rather than the total number of time windows during the entire measurement period, which is derived as $\mu_d = \frac{\sum_{i=1}^{N} W_{d,i}}{N}$, where N is the number of the days since device d is observed in the edge network and $1 \leq i \leq N$. So the temporal variability on time windows, measured by coefficient of variance, can be calculated as $CoV_d = \frac{\mu_d}{\sigma_d}$, where σ_d, the standard deviation, is derived as $\sigma_d = \sqrt{\frac{1}{N} \sum_{i=1}^{N} W_{d,i} - \mu_d}$.

Figure 2.4 is a scatter graph on the mean μ and coefficient of variance CoV of

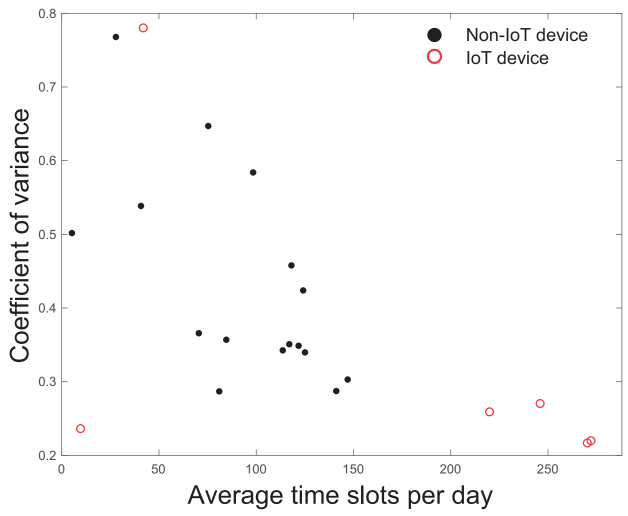


Figure 2.4. The Mean and Coefficient of Variance of Time Slots Observed With Traffic Activities for IoT and Non-IoT Devices

time slots observed with network activities for different IoT and non-IoT devices. In Figure 2.4, four out of the six IoT devices exhibit traffic activities during the majority of time windows in each and every day, and their *variability* on the number of time windows is much smaller compared with non-IoT devices. One of the IoT devices, i.e., an IP camera, is only active for a small number of time slots per day, but exhibits low variability on the time window as well. The only IoT device exhibiting a high variability is a smart TV, and the main reason is that it is often turned on and off in an unpredictable fashion. Based on these observations, we can easily classify connected networked devices in edge networks into three categories: always-on IoT devices (e.g. Echo Dot), on-demand IoT devices (e.g. IP camera and smart TV), and non-IoT devices.

The self-similarity traffic patterns of IoT devices also inspire us to analyze the autocorrelation on network traffic generated by all connected devices in edge networks. Autocorrelation is a metric that quantifies the correlation of the same variable across different and lagged periods of times, thus it is also often referred to as serial correlation and lagged correlation. The autocorrelation metric, $\rho_{d,k}$, for the IoT device d, between network traffic activity time series $X_{d,i}$ and a k-lagged copy of itself $X_{d,i+k}$ is captured by the autocorrelation function (ACF) as $\rho_{d,k} = \frac{\sum_{i=1}^{n-k}(X_{d,i}-\mu)(X_{d,i+k}-\mu)}{\sigma^2}$, where μ and σ are the mean and standard deviation of network traffic activity time series $X_{d,1}, X_{d,2}, \ldots, X_{d,n}$ for the device d, respectively. An autocorrelation value of 0 suggests independent and random observations on the traffic time series of connected devices in edge networks, while a significant autocorrelation reveals substantial correlations among adjacent observations or determines predictable seasonality in the time series [84, 104].

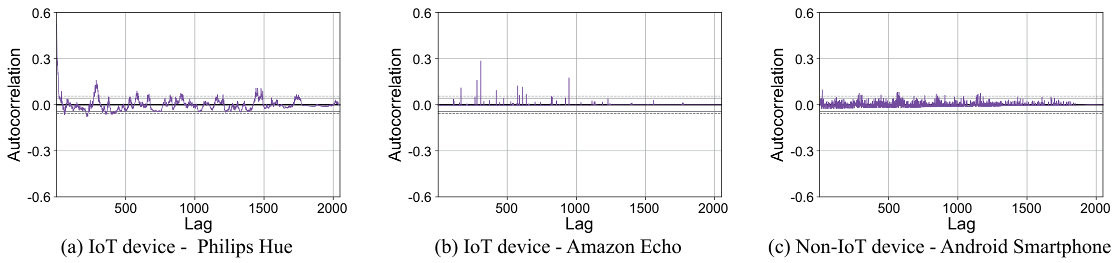


Figure 2.5. Autocorrelation Plots of Network Traffic Time Series for Selected IoT and Non-IoT Devices

Figure 2.5 illustrates the autocorrelation plots, also referred to as correlograms, of network traffic time series for three selected IoT and non-IoT device. These plots reflect distinct repeating patterns of different devices. We can see noticeable peaks at the beginning where time lag is short for both Philips Hue and Amazon Echo. This indicates that communication patterns of IoT devices are typically stable and predictable. On the other hand, for Android smartphone, there is no significant peak in the autocorrelation plot, which corresponds to our intuition that non-IoT devices like smartphones often have messy and random network traffic.

2.3.3 Characterizing Traffic Predictability via Sample Entropy

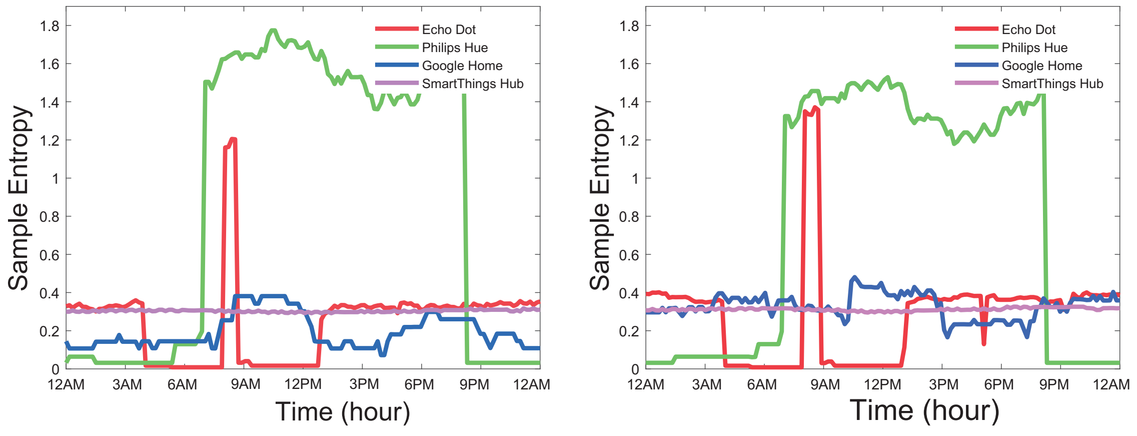
To further study temporal dynamics and predictability of IoT network traffic in edge networks, we explore *sample entropy*, denoted as $\mathcal{SE}$, to quantify the randomness, uncertainty, or determinism of network traffic for IoT device over time due to the inherent ability of the sample entropy measure in capturing the complexity and predictability of time series data [112]. Given a traffic feature f, our continuous data collection generates a unique time series observation $f(t_1), f(t_2), \ldots, f(t_M)$ over M consecutive time windows. Let $Y(t_i)$ denote a vector of m continuous observations at time t_i, i.e., $\{f(t_i), f(t_{i+1}), \ldots, f(t_{i+m-1})\}$. For $1 \leq i \leq M-m+1$, $B_i^m(r)$ represents the number of $Y(t_j)$ such that $D[Y(t_i), Y(t_j)] \leq r$ $(j \neq i)$, where $D[Y(t_i), Y(t_j)] = \max\limits_k |f_i(t_k) - f_j(t_k)|$ where $f_i(t_k) \in Y(t_i), f_j(t_k) \in Y(t_j)$, and r specifies how much two sequences are expected to exhibit strong similarity, which is usually set as proportional to the standard deviation of the original time series.

The sample entropy $\mathcal{SE}$ is defined as $\mathcal{SE} = -\ln(\Phi^{m+1}(r)/\Phi^m(r))$, where $\Phi^m(r)$ is the mean average value of $B_i^m(r)$, i.e., $\Phi^m(r) = (M-m+1)^{-1}\sum_{i=1}^{M-m+1} B_i^m(r)$. In other words, the sample entropy reflects the conditional probability for two subsequences of $f(t_1), f(t_2), \ldots, f(t_M)$ that are similar along m consecutive observations continue to share similarity for $m+1$ observations.

Given a time series of data $x(1), x(2), \ldots, x(N)$, SampEn defines two positive parameters $m \in \mathbb{N}$ and $r \in \mathbb{R}$ to represent the length of compared run and the filtering level respectively. Vectors $X(1), X(2), \ldots, X(N-m+1)$ are constructed, where $X(i) = \{x(i), x(i+1), \ldots, x(i+m-1)\}$. Each of these vectors contains m consecutive time series values starting from $x(t)$. For $1 \leq i \leq N-m+1$, B_i^m equals to the number of $X(j)$ such that $d[X(i), X(j)] \leq r$ $(j \neq i)$, where $d[X(i), X(j)] = \max\limits_a |x_i(a) - x_j(a)|$

and $x_i(a) \in X(i), x_j(a) \in X(j)$. Here r specifies how much we expect two sequences to be similar with other, and it is usually set as proportional to the standard deviation of the original time series. We define $\Phi^m(r)$ as the mean average value of $B_i^m(r)$: $\Phi^m(r) = (N-m+1)^{-1}\sum_{i=1}^{N-m+1} B_i^m(r)$. We have $SampEn = -\ln(\Phi^{m+1}(r)/\Phi^m(r))$.

Applying the sliding window approach, we can estimate the entropy values for all traffic features of IoT devices in edge networks. In our experiments, we set each observation time window as 10 minutes and the overall time period as 4 hours, and then calculate the sample entropy of the time series traffic data collected in the past four hours to balance the computational overhead and real-time responses to traffic fluctuations.



(a) Incoming traffic of selected IoT devices (b) Outgoing traffic of selected IoT devices

Figure 2.6. Distinct Sample Entropy Measures of Network Traffic by Different IoT Devices Over Time

We set the parameters m as 2 and r as 0.2 times the standard deviation of time series $f(t_1), f(t_2), \ldots, f(t_M)$ [110]. Based on our experimental results, such parameter settings will best estimate the time series entropy and depict the traffic and activity patterns of IoT devices, which confirms the findings in [110]. Figure 2.6 illustrates the distinct sample entropy measures on packet count of four different IoT devices in the same edge network. Specifically, the entropy of Echo Dot exhibits a spike at 9AM due to the music playing on Spotify and a preconfigured weather forecast service during this time period, while Philips Hue communicates with the cloud server actively during the daytime and remains only the heart-beat communications with servers at night. Compared with Echo Dot and Philips Hue, Google Home and SmartThings Hub have more stable entropy over time, as our in-depth analysis reveals that most of their network traffic are predicable short-term connections with NTP, DNS, and cloud servers. In other words, the simple yet effective sample entropy measure is able to capture, characterize, and distinguish the temporal dynamics and predictability of network traffic for IoT devices, thus potentially could help develop new event detection and intrusion prevention algorithms for monitoring and securing IoT devices in edge networks.

2.3.4 Cloud Behavior of IoT Devices

The objective of cloud behavior analysis is to understand why and how IoT devices communicate with remote cloud servers. Specifically, we profile cloud behaviors of IoT devices based on the *dominant* applications or services observed from `dstPort` and protocol of their outgoing network traffic flows. Table 2.1 demonstrates all the observed 5 applications for 5 IoT devices deployed in one edge network during a 24-hour time window. These 5 applications are HTTP, HTTPS, DNS, NTP, and Spotify music streaming. As a comparison, the Andriod Smartphone and the Macbook laptop in the same edge network engage with 11 and 15 distinct applications, respectively, during the same time period.

The limited and consistent set of common applications used by IoT devices

Table 2.1. The Dominant Applications Used by IoT Devices in Edge Networks

Application	Service	Echo	Camera	Echo Dot	Philips Hue	Smart TV
443/TCP	HTTPS	Y	Y	Y	Y	Y
80/TCP	HTTP	Y		Y	Y	Y
53/UDP	DNS	Y	Y	Y		Y
123/UDP	NTP	Y	Y	Y	Y	
4070/TCP	Spotify	Y				

again confirms that IoT devices are typically designed for very specific functions and dedicated utilities. Figure 2.7 illustrates the convergence of cloud applications for IoT and non-IoT devices, where the number of applications for IoT devices converges rapidly.

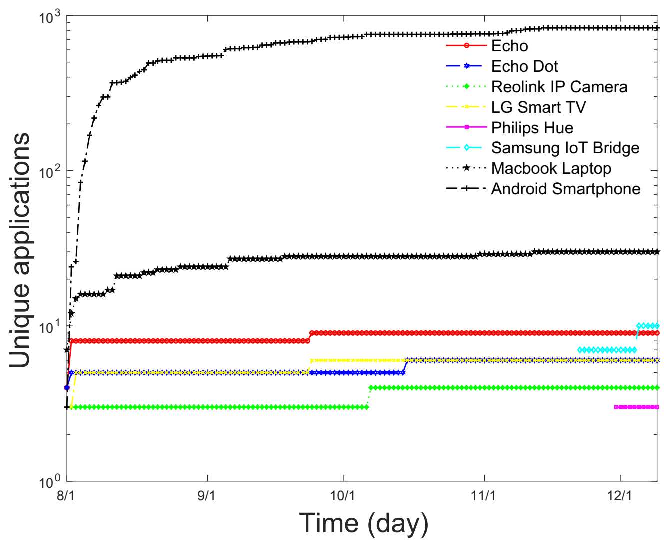


Figure 2.7. The Convergence of Applications for IoT and Non-IoT Devices

We continue to characterize the remote servers and their aggregated network prefixes or ASNs via analyzing the *fanouts*, i.e. unique numbers of destination IP address, BGP prefixes, and ASNs, for each application. In addition, we measure the distribution of network traffic across these remote servers, prefixes and ASNs by calculating the entropy and standardized entropy of these fanouts. For a given application a of an IoT device d, let F and R denote the number of network traffic flows and the *unique* numbers of the remote servers represented as $s_1, s_2, \ldots, s_R$. The probability of each remote server P_{s_i} is calculated as $P_{s_i} = \frac{C_{s_i}}{F}$, where C_{s_i} denotes the number of flows between d and s_i. Clearly $\sum_{i=1}^{R} C_{s_i} = F$. The normalized entropy on the remote servers for application a of device d is then derived as $\mathcal{NE}_{d,a} = -(\log R)^{-1} \sum_{i=1}^{R} P_{s_i} \times \log P_{s_i}$.

Table 2.2. The Entropy of Destination IP Addresses, Prefixes and ASNs IoT Devices Have Sent HTTPS Requests Within a 24-Hour Time Window

Device	Flows	Fanout			Normalized Entropy		
		IP	Prefix	ASN	IP	Prefix	ASN
Echo	148	20	6	1	0.5529	0.3158	0.0000
IP Camera	32	12	9	2	0.6023	0.5422	0.1792
Echo Dot	228	40	10	2	0.6197	0.3365	0.0051
Philips Hue	96	4	2	1	0.2163	0.0221	0.0000
LG Smart TV	429	109	39	7	0.6574	0.2968	0.1733
IoT Hub	258	3	2	1	0.1969	0.1115	0.0000
Laptop	3831	832	340	90	0.6782	0.5191	0.3064
Smartphone	1497	353	131	21	0.6274	0.4964	0.3077

The normalized entropy is in the range of [0, 1], revealing the degree of uncertainty, randomness, or variations on the remote servers which communicate with IoT devices in edge networks. Clearly, a $\mathcal{NE}_{d,a}$ value of 0 or near 0 indicates the uniformity on the remote servers, which means that this device only communicates with one or few servers on application a. While a $\mathcal{NE}_{d,a}$ value of 1 or near 1 means the high randomness on the remote servers.

Following a similar process, we could also calculate the entropies and normalized entropies for their aggregated network prefixes or ASNs of remote servers.

Table 2.2 illustrates the entropy values of destination IP addresses, prefixes and ASNs of the hosts which IoT devices have sent HTTPS requests to within a 24-hour time window. In our eperiments, all IoT devices exhibit some uncertainty on network prefixes and ASNs for their HTTPS traffic, while the laptop and smartphones exhibit much higher variations on the remote prefixes and ASNs for HTTPS traffic.

2.3.5 IoT Traffic Behaviors within Edge Networks

2.3.5.1 Communication between IoT Devices and Edge Routers

Based on our longitudinal measurement, we have observed two dominant applications in this category, dynamic host configuration protocol (DHCP) and DNS. All of the IoT devices exchange the DHCP information with their respective edge routers periodically via UDP ports 67 and 68. This observation is consistent with the common DHCP configurations on today's home routers, which act as DHCP servers and automatically assign the IP address to all of the devices in edge networks. After the lease time is over, home edge routers will renew it if the IoT device is still active. It is interesting to note that all of the IoT devices in our study have their IP addresses renewed every 12 hours by exchanging DHCP packets with the edge routers, which indicates that the DHCP lease time is set as 12 hours on the router side.

Similar to many non-IoT devices, such as laptops and smartphones, the majority of IoT devices leave the choice of DNS servers to the edge routers for the consideration of short DNS query and reply latency. Routers usually set themselves as the DNS server and add their IP addresses in the "DNS servers" field of the DHCP Offer packets. Among the 20 types of IoT devices in our study, Google Home is the only device that prefers external DNS services, i.e., Google's own public IPv4 DNS servers `8.8.8.8` and `8.8.4.4` over the edge router-based DNS services. As DNS traffic is often triggered by many IoT applications such as HTTPS and NTP for retrieving IP addresses of the cloud servers, there tends to exist strong correlations between the internal DNS traffic of IoT devices and the actual incoming wide area networks (WAN) network traffic of these devices.

2.3.5.2 Communication among IoT Devices

The data communication among IoT devices happens primarily during the *operation* stage between paired IoT devices in the same edge network such as an Amazon Echo and a Philips Hue bridge. Due to the improved network latency and simplified management and operations, different IoT devices in the same edge network can be "paired" with each other for better communication and cooperation. We notice that during the initial pairing stage, all IoT devices contact their respective vendors' cloud servers for authentication and registration. After a paired relationship is established, many of the paired devices continue to rely on the cloud servers as a proxy to communicate with each other for security and trust considerations.

```
SrcIP           DstIP           SrcPort  DstPort  Size
192.168.1.216   192.168.1.195   59337    80       678B
192.168.1.195   192.168.1.216   80       59337    2634B
192.168.1.216   192.168.1.195   59338    80       574B
192.168.1.195   192.168.1.216   80       59338    2542B
192.168.1.216   192.168.1.195   59339    80       750B
192.168.1.195   192.168.1.216   80       59339    1650B
```

Figure 2.8. First 6 Network Flows Captured When Sending an "open" Command From Amazon Echo Dot (192.168.1.216) to Philips Hue Bridge (192.168.1.195)

We also noticed one pair of IoT devices, Amazon Echo Dot and Philips Hue bridge, directly talking with each other using the HTTP protocol, after the Philips Hue bridge is added into the trusted device list on the Amazon Echo Dot. Figure 2.8 illustrates network packets captured by the router when we press the "open" button in the Amazon Alexa App. The direct internal communication significantly improves the efficiency and latency of operating the Philips Hue bulbs via controlling Amazon Echo Dot in the same edge network, since the commands are not required to transfer through the long-latency path from mobile Apps on smartphones, cloud servers, the Philips Hue bridge, to the light bulbs. On the other hand, the direct communication using the insecure HTTP protocol could potentially leave both IoT devices vulnerable to attacks. Therefore, whether retaining the cloud servers as a communication proxy is a system design trade-off between security and efficiency.

2.3.5.3 Communication between IoT and Non-IoT Devices

We discover three communication protocols, multicast DNS (mDNS), simple service discovery protocol (SSDP), and HTTP/HTTPS in this category. Many Apple devices, Linux-based networked systems, and Windows computers with Apple iTunes all periodically broadcast mDNS packets to identify and resolve the IP addresses of other devices in the same edge network. In our study, Philips Hue bridge is the only IoT device leveraging mDNS protocol to identify itself via replying mDNS queries but many IoT devices broadcast mDNS packets in order to find other devices.

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: 4
ST: urn:schemas-upnp-org:device:basic:1
```

Figure 2.9. An Example of SSDP Requests Sent by SmartThings Hub

The SSDP protocol is designed for the advertisement and discovery of network services and device existence. Many IoT devices adopt SSDP protocol to bootstrap the device discovery services. For example, Samsung SmartThings hub broadcasts SSDP messages whenever a user tries to pair a new IoT device to the hub using the

SmartThings App on a smartphone. Figure 2.9 shows an SSDP message sent from the Samsung SmartThings hub when the hub is requested to pair with a Philips Hue bridge. The Mandatory Extensions in HTTP (MAN) in Figure 2.9 defines the scope of the extension and carries the value of "ssdp:discover" to indicate a device search request, and the maximum wait time in seconds (MX) is used for load balance when the hub processes the SSDP responses. Search target (ST) is in the format of `urn:schemas-upnp-org:device:DeviceType:version` in the case of searching for a particular device, specified by the device type and version.

The corresponding device type and version of Philips Hue bridge is `basic` and `1`, respectively, as included in the SSDP response messages. However, we notice that the Samsung SmartThings hub is actually enumerating all the device types and versions by sending out different SSDP requests, which explains why our IoT measurement framework captures a large number of SSDP network flows during every device pairing process. These SSDP requests also flood in the Wi-Fi networks even if the selected device pairs with the hub using other wireless communication protocols such as ZigBee and Z-Wave. In other words, the drive-by attackers, if receiving the broadcast SSDP packets sent by the hub, could pair with and potentially compromise the corresponding IoT devices. These findings confirm the design flaws in the paring stage of these wireless protocols reported in the prior research in [90, 114, 115, 159]. Our study also discovers an interesting behavior of Philips Hue bridge which proactively sends out the SSDP packets targeting a Windows PC in the same edge network every two minutes. Such unique traffic pattern could help effectively detect this type of IoT device.

The third type of communication protocol between IoT and non-IoT devices is HTTP/HTTPS, which is used by smartphones to directly communicate with a variety of IoT devices in the same edge network. Many IoT devices are controlled, configured, and monitored by smartphone-based apps on Android or Apple iOS platforms, and these devices e.g. Philips Hue Bridge, Google Home, and Reolink Camera, often allow the smartphones to directly communicate with them for reduced network latency using HTTP and HTTPS protocols if and only if the device has been registered in the corresponding application on the smartphone and the IoT device and smartphone are in the same edge network. On the other hand, some IoT devices such as Echo Dot, SmartThings Hub, and Ring Video Doorbell strictly require that all data packets of command and control must first go through the trusted cloud servers and then be forwarded to the devices for security reasons.

2.4 Related Work

IoT behavioral profiling and fingerprinting is one of the crucial topics where we have witnessed a lot of recent research efforts with the recent rapid development and deployment of IoT devices in smart homes[11, 65, 88, 103, 118, 131]. The fingerprinting techniques cover nearly all protocol layers of TCP/IP stacks such as applying wavelet transform on the sequence of packet inter-arrival time (IAT) of wireless access points for device profiling [49, 54, 125] or characterizing packet headers and IP payload [11, 86].

Most of the existing studies on IoT behavioral fingerprinting are centered on the protocols of physical and link layers for the applications of device classification [49, 54, 64, 125]. For example, IoTScanner [125] introduces a real-time system that passively scans and analyzes the data communication over WiFi, Bluetooth, and Zigbee for classifying IoT devices and detecting privacy threats. While BF-IoT [54] proposes to extract the unique features from the link and service layers of Bluetooth low energy

(BLE) protocol stack for generating the IoT fingerprint for authenticating devices and defending against spoofing attacks. In addition, a wireless device identification platform for distinguishing legitimate and adversarial IoT devices based on radio frequency (RF) fingerprinting over different ranges of signal-to-noise ratio (SNR) levels is proposed in [64].

A few recent studies have shifted traffic data collection and analysis to the network, transport, and application layers for device behavioral modeling and characterization [11, 86]. For example, IoT SENTINEL [86] achieves IoT device fingerprints with 20 binary features of protocol fields extracted from packet headers collected from link, network, transport and application layers to reflect the protocol engagement of IoT devices headers such as ARP, IP, ICMP, TCP, UDP, NTP, DNS, DHCP, HTTP and HTPPS, and 3 numerical features including packet size, destination IP counter, source and destination port numbers. In [11], behavioral fingerprints of IoT devices are characterized with a subset of binary features identified in [86], and 3 payload-based features including the entropy of payload, TCP payload size, and TCP window size. Complement to these studies, our efforts are focused on behavioral fingerprinting of IoT devices in edge networks based on network flow records rather than the raw IP data packets which raise privacy concerns of IoT users and incur expensive computational and storage cost on resource-constrained commodity edge routers such as off-the-shelf home routers.

A very recent paper studying the IoT devices on home networks [72] provides a large-scale empirical analysis with the ISP level network traffic data. Different from [72], our study explores programmable edge routers to build an IoT measurement framework from the perspective of edge networks and sheds light on multidimensional traffic patterns of IoT devices from incoming and outgoing network traffic as well as from the local LAN traffic within edge networks.

2.5 Conclusions

As the wide adoption of IoT devices continues to accelerate in smart homes, cities, and industries, it becomes increasingly urgent to design and implement Internet traffic measurement platforms to effectively monitor, characterize, and profile communications patterns of IoT devices with remote end hosts on the Internet and local systems on the same edge networks. Towards this end, this study develops a systematic measurement framework for establishing multidimensional behavioral profiles of connected IoT devices based on a wide spectrum of traffic features from IP-spatial, temporal, entropy, and cloud dimensions. We also leverage the benefits of our programmable router based scheme to take a deep look into the LAN network patterns of different IoT devices. Based on our deep analysis, we have discovered a number of important and interesting findings. We notice that IoT devices typically communicate with cloud servers from a very small number of prefixes and ASNs, which belong to IoT manufactures, the cloud service providers, NTP service providers, and public DNS service providers. IoT devices also often exhibit repeated and predictable traffic activities over time due to heart-beat signals between IoT devices and cloud servers. Unlike laptops, desktops, or smartphones, IoT devices often engage with a limited and common number of applications such as DNS, HTTPS, HTTP, and NTP. These behavioral fingerprints not only characterize communication patterns of IoT devices with end systems on the Internet, but also benefit a range of security applications for IoT devices.

Part II

Extracting Critical Patterns from Smart Home Network Traffic

Chapter 3

IOTATHENA: UNVEILING IOT DEVICE EVENTS FROM NETWORK TRAFFIC

3.1 IoTAthena System Overview

Developing effective techniques to understand and report IoT device events, *e.g.*, *the smart lock of the home's main entrance is unlocked remotely with a smartphone app*, is crucial for ensuring the physical and property safety of these devices' homeowners. Our real-world experiments with August Lock and other IoT devices demonstrated the feasibility of developing an automated system to learn and generate signatures of IoT device events and use them for unveiling IoT device events from network traffic logs. Such a system is urgently needed for understanding what is happening to IoT devices in millions of smart homes and for detecting suspicious and unauthorized behaviors towards critical home devices.

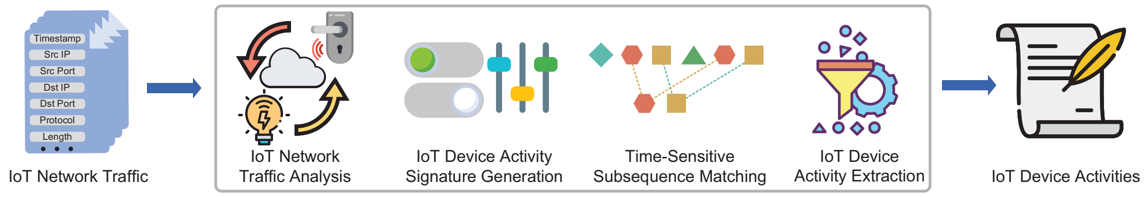

Timestamp
Src IP
Src Port
Dst IP
Dst Port
Protocol
Length
IoT Network Traffic
IoT Network Traffic Analysis
IoT Device Activity Signature Generation
Time-Sensitive Subsequence Matching
IoT Device Activity Extraction
IoT Device Activities

Figure 3.1. Overall Architecture of the IoTAthena System for Unveiling IoT Device Events From IoT Network Traffic

In this study, we propose a new system, named IoTAthena, to automatically and accurately unveil IoT device events from smart home network traffic logs. Figure 3.1 illustrates the overall architecture of IoTAthena, which includes four key system modules: i) IoT network traffic analysis, ii) IoT device event signature generation, iii) time-sensitive subsequence matching, and iv) IoT device event extraction.

The *IoT network traffic analysis* module takes IoT network traffic during the intentionally "silent" period, and characterizes background network traffic for each IoT device. The *IoT device event signature generation* module collects the corresponding network traffic of each IoT device event by intentionally triggering the event and collecting the traffic. The collected IoT network traffic along with the labeled event logs serve as the ground truth for generating the signature of each IoT device event consisting of an ordered sequence of IP packets with inter-packet time intervals. The *time-sensitive subsequence matching* module relies on the `sigMatch` algorithm to capture all matches of each IoT device event signature in the network traffic log, while the *IoT device event extraction* module relies on `actExtract` to unveil the sequence of IoT device events from the network traffic log.

In summary, IoTAthena adopts a white-box approach to first generate signatures of IoT device events consisting of ordered sequences of IP packets with inter-packet time interval information. Subsequently, IoTAthena applies efficient matching algorithms for deterministically unveiling the sequence of IoT device events from the network traffic log, unlike black-box machine learning classification models [1, 11, 86, 126].

3.2 Network Traffic Collection and Analysis

Network traffic of IoT devices embeds rich information on device types and their behavioral patterns [109, 126]. In this section, we describe how to collect and analyze IoT network traffic in order to characterize and generate signatures of IoT device events.

3.2.1 IoT Network Traffic Collection

Figure 3.2 illustrates the data flows initiated from an IoT device or destined to an IoT device in a smart home environment. For clarity, we use two IoT devices as examples: a smart lock and a security camera. A user usually interacts with an IoT device using the device's companion app on the smartphone in the home or outside the home, e.g., in the office or on the road. The app first communicates with the cloud server which in turn generates traffic between the cloud server and the device, as illustrated by the solid red line between the smart lock and the cloud server. Sometimes, the smartphone directly communicates with the device without involving the cloud server, such as streaming requests on the security camera, illustrated by the solid green line between the smartphone and the security camera.

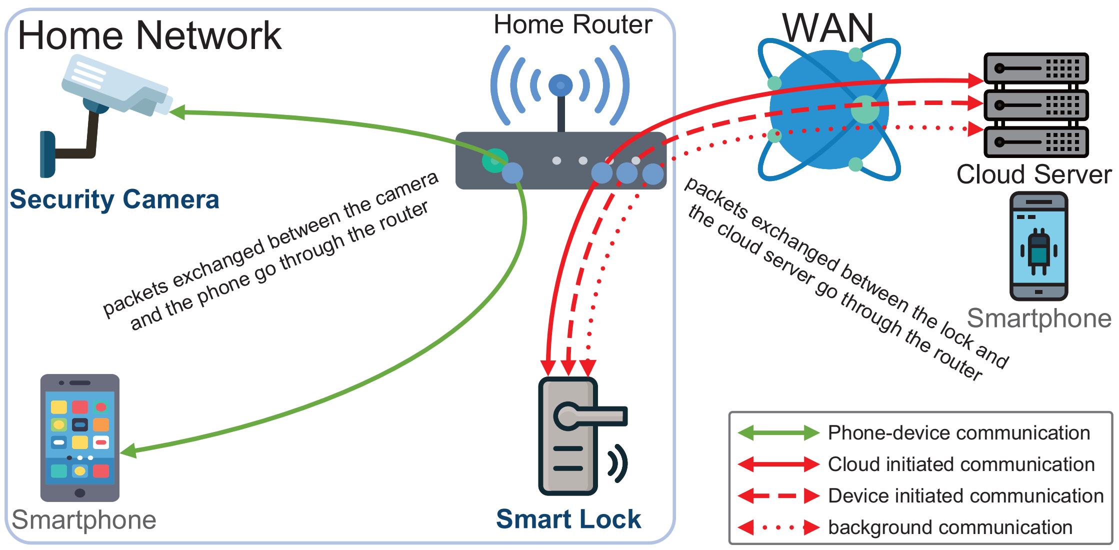

Home Network
Home Router
WAN
Security Camera
Cloud Server
packets exchanged between the camera and the phone go through the router
packets exchanged between the lock and the cloud server go through the router
Smartphone
Smartphone
Smart Lock
Phone-device communication
Cloud initiated communication
Device initiated communication
background communication

Figure 3.2. Illustration of Data Flows Initiated From or Destined to IoT Devices, Using Smart Lock and Security Camera As Examples

The user can also manually operate the device in the traditional way, such as locking the smart lock manually. This action causes the device to update its status to the cloud server immediately following the action. In addition, the user can communicate directly with the device locally through a non-WiFi communication channel, such as Bluetooth or ultrawideband (UWB) when the user is in the vicinity of the device. This action also causes device initiated status updates. Furthermore, automatic device operations such as the smart lock's autolocking function also introduce status update traffic. These types of device initiated communications are illustrated by the dashed red line. There also exists traffic introduced by device background operations such as device firmware update checks. We use the dotted red line between the smart lock and the cloud server to illustrate these data flows.

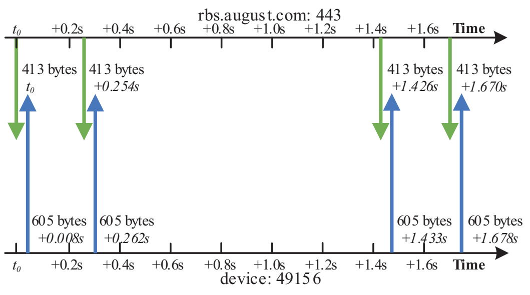
IoTAthena collects the network traffic at the programmable home router, which enables the capture of incoming and outgoing packets of all above mentioned device-related operations. The smart home router is a desirable centralized location for data collection, considering its switching and routing function, sufficient computational and processing capacities, and the design transparency to IoT devices and apps.

3.2.2 IoT Network Traffic Analysis

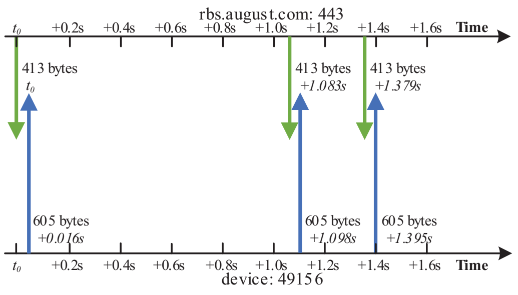
The network traffic collected at the home router can be classified into two parts: the first part consists of traffic between the IoT devices and the cloud servers, while the second part consists of internal LAN traffic such as address resolution protocol (ARP) requests and SSDP broadcast packets. In order to separate the logs of an individual IoT device from the mixed home network traffic, IoTAthena first identifies each device's unique IP address via the mapping of its media access control (MAC) address and host name in DHCP packets. It subsequently uses the device IP address as the unique *cluster* key to separate IoT network traffic into individual traffic clusters to simplify further analysis.

We carefully studied the network traffic within each individual traffic cluster of an IoT device. We can clearly observe the network traffic one would anticipate for normal IoT device events, e.g., users issuing locking or unlocking commands for August Lock via the smartphone app. Surprisingly, we also discovered a significant amount of network traffic when there is no human-triggered or environment-triggered IoT device event. We use the term *background traffic* to denote such network traffic, i.e., network traffic not triggered by human or environment. In order to gain a thorough understanding of IoT background traffic, we left the devices in our controlled smart home environment without any human interactions for one week and consider the network traffic cluster of each IoT device during this "silent" period as background traffic. By separating IP data packets based on the destination (and source) ports of the outgoing (and incoming) traffic, we observed that these IoT devices typically exchange messages with the remote cloud servers on the well-known application ports such as 22/TCP (SSH), 53/UDP (DNS), 80/TCP (HTTP), 123/UDP (NTP), 5353/UDP (mDNS). This observation leads us to classify IoT background traffic into three categories: management and service, signal and update, and random noise.

The management and service traffic is mainly used to manage and maintain the devices, e.g., periodical time synchronizations with NTP servers. The signal and update traffic corresponds to keep-alive signals and regular firewall update checks between IoT devices and cloud servers. The random noise traffic is mostly generated by other IoT or non-IoT devices in the local home network for a variety of reasons, e.g., ARP requests, SSDP broadcasts, and mDNS traffic from Apple Bonjour protocol for automatic device and service discovery.



(a) August Lock *WiFi (un)locking*.



(b) August Lock *Bluetooth (un)locking*.

Figure 3.3. Packet Sequences of August Lock's Device Events: The Pattern in (b) Seems Like a Subsequence of the Pattern in (a)

3.3 IoT Device Event Signatures

Consistent with the findings of PingPong [135], we observed repetitive network packet sequences that correspond to repeated device events in the network traffic collected at the router of the smart home network. We also observed certain August Lock device events resulting packet sequences that are challenging for PingPong to recognize. Figure 3.3 illustrates such an example. The Bluetooth (un)locking[1] event's packet sequence (3 pairs as illustrated in Figure 3.3(b)) is a subset of the WiFi (un)locking event's packet sequence (4 pairs as illustrated in Figure 3.3(a)). The clustering of re-occurring packet pairs approach in PingPong cannot distinguish WiFi (un)locking from Bluetooth (un)locking. In fact, it is difficult to distinguish these two events in the network traffic solely based on request/reply patterns, which leads us to consider more information (the full detailed packet sequence) and inter-packet time intervals to characterize IoT device events. The time intervals between consecutive

[1]The locking event and the unlocking event exhibit the same packet sequences and inter-packet time intervals because of the simple lock/unlock state transitions. The encrypted application data prevents us for further differentiating these two events with network traffic only. We generate a unique signature for each *indistinguishable event group*. For example, we use `(un)locking` for short to denote either the `locking` event or the `unlocking` event. Similarly, we use `on or off` to denote either the `on` event or the `off` event.

packets provide critical information to effectively and accurately differentiate IoT device events such as those in Figure 3.3 that share overlapping packet sequences and happen very closely in time.

3.3.1 Inter-Packet Time Interval Measurement

Because IoTAthena collects network traffic at the home router, the inter-packet time intervals are essentially the round-trip time (RTT) between the home router and IoT devices in the smart home plus the processing time at the device (LAN), or the RTT between the home router and cloud servers across the Internet plus the processing time at the cloud server (WAN).

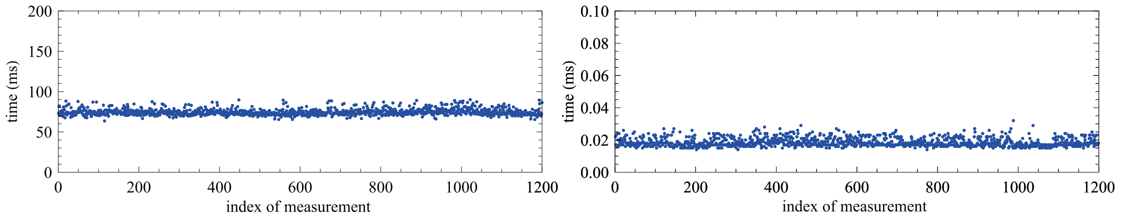


Figure 3.4. Inter-Packet Time Intervals: Large Values in WAN (left) vs Small Values in LAN (right)

Figure 3.4 illustrates the time interval between the first and second packets (left plot), and the time interval between the second and third packets (right plot), of $1,200$ repeated *on* device events of TP-Link Plug over a 24-hour span. We observe that the inter-packet time intervals exhibit stable and consistent patterns, with small variances. However, the time interval between one pair of consecutive packets may significantly differ from that between another pair of consecutive packets. Specifically, the interval between the first and second packets has a mean (μ) of $76.73ms$ and a standard deviation (σ) of $0.003995ms$, while the interval between the second and third packets has a mean (μ) of $0.02ms$ and a standard deviation (σ) of $0.000003ms$ for TP-Link Plug's *on* event. The unstable wireless channel between the IoT devices and the home router could result in packet loss and retransmission which contribute to the fluctuation in the LAN inter-packet time intervals. The uncertain number of retransmissions in the MAC layer affects the inter-packet time intervals in our collected traffic log. However, compared with the short wireless transmission delay, the local processing time at the IoT device still dominates the LAN inter-packet time intervals, as we observe from the right plot in Figure 3.4. These key observations inspire us to include inter-packet time intervals as an important component in characterizing the signatures of IoT device events.

3.3.2 IoT Device Event Signature Definition

A network data packet p collected at the home router is an 8-tuple, where the first through eighth fields are *timestamp*, *IoT device internal IP address*, *canonical remote cloud server name*, *remote application port*, *protocol*, *traffic direction*, *packet length*, and *application-layer data*, respectively. It is important to note that each TCP/IP data packet carries a variety of traffic features including those in the 8-tuple. However, this study only selects the features that provide additional information on identifying and differentiating IoT device events, while skipping the features, e.g., Time to Live (TTL), sequence and acknowledgement numbers with redundant or little contributions towards device event identification.

We use $p.t$ to denote the timestamp of packet p, and use $\widehat{p}$ to denote the 7-tuple obtained by deleting the first field (timestamp) in p. We call $\widehat{p}$ the *base packet* of packet p.

Definition 3.1. *The signature of an IoT device event is given by an ordered sequence of n base packets $(\widehat{q_1}, \widehat{q_2}, \ldots, \widehat{q_n})$, together with an ordered sequence of $n-1$ inter-packet time intervals $(\tau_1, \tau_2, \ldots, \tau_{n-1})$, where $\tau_j > 0$ is the time interval between the jth packet and the $j+1$th packet, $j = 1, 2, \ldots, n-1$. The number of base packets, n, in each device event signature is determined by the observed TCP/IP data packets triggered by the device event minus the protocol-specific packets, e.g., TCP three-way handshake, and the regular heart-beat signals between the device and the remote cloud server.* □

Instead of using $(\widehat{q_1}, \widehat{q_2}, \ldots, \widehat{q_n})$ and $(\tau_1, \tau_2, \ldots, \tau_{n-1})$ to represent a signature, we can equivalently represent the same signature using a sequence of n packets $(\rho_1, \rho_2, \ldots, \rho_n)$, where $\widehat{\rho_j} = \widehat{q_j}$ for $j = 1, 2, \ldots, n$ and $\rho_{j+1}.t - \rho_j.t = \tau_j$ for $j = 1, 2, \ldots, n-1$. In this representation, the time interval between the jth packet and the $j+1$th packet can be uniquely computed by $\tau_j = \rho_{j+1}.t - \rho_j.t$.

The signature of an IoT device event is a constant, as defined in **Definition** 3.1. The above *alternative* representation of the signature, however, does not look like a constant *in format*. For example, for any given real number c, $(p_1, p_2, \ldots, p_n)$ and $(\rho_1, \rho_2, \ldots, \rho_n)$ denote exactly the same signature, provided that $\widehat{p_j} = \widehat{\rho_j}$, $p_j.t = \rho_j.t + c$, for $j = 1, 2, \ldots, n$. Since $(p_1, p_2, \ldots, p_n)$ and $(\rho_1, \rho_2, \ldots, \rho_n)$ define exactly the same sequence of n base packets $(\widehat{p_1}, \widehat{p_2}, \ldots, \widehat{p_n}) = (\widehat{\rho_1}, \widehat{\rho_2}, \ldots, \widehat{\rho_n})$ and exactly the same sequence of $n-1$ inter-packet time intervals $(p_2.t - p_1.t, p_3.t - p_2.t, \ldots, p_n.t - p_{n-1}.t)$ $= (\rho_2.t - \rho_1.t, \rho_3.t - \rho_2.t, \ldots, \rho_n.t - \rho_{n-1}.t)$, *we can use this alternative representation without losing any accuracy.*

Given the above discussions, we will denote a signature of an IoT device event using an ordered sequence of packets $(q_1, q_2, \ldots, q_n)$, where the timestamp fields are only used to compute the inter-packet time intervals $\tau_j = q_{j+1}.t - q_j.t$. For this reason, we also call the timestamp fields in a signature *relative timestamps*. We set $q_1.t$ to 0 for simplicity.

3.3.3 Automated Signature Generation

Towards automatically generating signatures of IoT device events, we first follow the same practice as [158] to compile a complete list of any given IoT device's events from the AndroidManifest.xml file of the device's companion app. We then write scripts using command-line tool and scripting feature in Android Debug Bridge (ADB) to automate the user interactions with IoT devices such as turning on/off Philips Hue and (un)locking of August Lock. For all IoT devices in our lab, we trigger each of their device events 100 times in order to remove randomness and gain statistically meaningful understanding of the device event packet sequence.

Filtering the background traffic described in Section 3.2.2, which happens in parallel with the device event, leads to an ordered sequence of timestamped IP packets exchanged between IoT devices and the cloud servers. Each packet in the sequence carries a variety of traffic features such as the timestamp of each packet, local IP address and port number of the IoT device, remote IP address and port number of the cloud server, protocol, packet length, and the actual application payload of IoT applications which are mostly encrypted for security and privacy reasons. For each packet, we continue to remove features with random and dynamic values due to the protocol designs, e.g., the random local port number at IoT devices in TCP connections with cloud servers and TCP sequence and acknowledge numbers. In addition, we transform certain traffic features to retain the stable values, e.g., converting dynamic

IP addresses of load-balanced cloud servers to the canonical remote cloud server names.

The inter-packet time interval τ_j between the jth packet and the $j+1$th packet in the signature is set to the mean (over the 100 tries) of the inter-packet time intervals. To simplify notations, we set $q_1.t$ to 0, and set $q_{j+1}.t = q_j.t + \tau_j$, $j = 1, 2, \ldots, n$.

3.4 Algorithms for Unveiling IoT Device Events from Network Traffic

Having discussed network traffic in Section 3.2 and device event signatures in Section 3.3, we are now ready to present our algorithms for unveiling IoT device events from network traffic logs. In Section 3.4.1, we formally define the IoT device event signature matching problem and the IoT device event extraction problem. In Section 3.4.2, we present the `sigMatch` algorithm for identifying all matches of a given signature in the network traffic log. In Section 3.4.3, we present the `actExtract` algorithm for unveiling the sequence of events of an IoT device from the network traffic log. In Section 3.4.4, we discuss the limitations and extensions of our algorithms.

3.4.1 Problem Formulation

An IoT *network traffic log* (denoted by $\mathbb{L}$) is an ordered sequence of packets $(p_1, p_2, \ldots, p_m)$ with increasing timestamps (i.e., $p_{i'}.t < p_{i''}.t$ for $i' < i''$). As discussed in Section 3.3.2, a *signature* of an IoT device event (denoted by $\mathbb{S}$) is an ordered sequence of packets $(q_1, q_2, \ldots, q_n)$ with increasing relative timestamps (i.e., $q_{j'}.t < q_{j''}.t$ for $j' < j''$). Recall that $\widehat{p}$ and $\widehat{q}$ denote the 7-tuple obtained by deleting the timestamp in p and the relative timestamp in q, respectively. A *signature set* of an IoT device (denoted by $\mathbb{SS}$) is a set of distinct signatures $\{\mathbb{S}^1, \mathbb{S}^2, \ldots, \mathbb{S}^K\}$, one signature per event of the device.

In light of the end-to-end network latency variations on the Internet [28, 59, 105], we allow an inter-packet time interval tolerance $\epsilon_j > 0$ as the "safety margin" for the measurement of $q_{j+1}.t - q_j.t$ when trying to find a match of a signature in the network log.

Let j satisfy $1 < j \leq n$ and $\delta > 0$ be a given tolerance. Let i' and i'' satisfy $1 \leq i' < i'' \leq m$. We say that $(p_{i'}, p_{i''})$ is a δ-valid match of (q_{j-1}, q_j), if

1. $\widehat{p_{i'}} = \widehat{q_{j-1}}$, $\widehat{p_{i''}} = \widehat{q_j}$;
2. $|(p_{i''}.t - p_{i'}.t) - (q_j.t - q_{j-1}.t)| \leq \delta$.

Let $\mathbb{S} = (q_1, q_2, \ldots, q_n)$ be a signature. Let $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_{n-1})$ be the matching tolerance vector, where ϵ_j is the tolerance for the matching of (q_j, q_{j+1}). Let $(l[1], l[2], \ldots, l[n])$ be an increasing sequence of integers indicating the index of the location of a packet in the network log. We say that $(p_{l[1]}, p_{l[2]}, \ldots, p_{l[n]})$ is an ϵ-valid match of signature $\mathbb{S}$ in log $\mathbb{L}$, if $(p_{l[j]}, p_{l[j+1]})$ is an ϵ_j-valid match of (q_j, q_{j+1}), for $j = 1, 2, \ldots, n-1$.

We study the following two related problems:

IoT device event signature matching: Given network traffic log $\mathbb{L}$, signature $\mathbb{S}$, and tolerance vector ϵ for $\mathbb{S}$, identify all ϵ-valid matches of signature $\mathbb{S}$ in log $\mathbb{L}$.

IoT device event extraction: Given network traffic log $\mathbb{L}$ and signature set $\mathbb{SS}$, find a sequence of IoT device events $\mathbb{A}_1, \mathbb{A}_2, \ldots$, whose execution leads to the network traffic log $\mathbb{L}$.

3.4.2 Signature Matching via Time-Sensitive Subsequence Matching

The IoT device event signature matching problem is different from the traditional subsequence matching problem [80] and the longest common subsequence problem [10, 21] due to the inter-packet time interval constraint. We solve the signature matching problem using a time-sensitive subsequence matching approach, called `sigMatch`, as presented in Algorithm 3.1.

```
Algorithm 3.1: sigMatch($\mathbb{L}, \mathbb{S}, \epsilon$)
Input: Network traffic log $\mathbb{L} = (p_1, p_2, \ldots, p_m)$, Signature $\mathbb{S} = (q_1, q_2, \ldots, q_n)$,
       tolerance vector $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_{n-1})$.
Output: A DAG $G_{\mathbb{LS}} = (V_{\mathbb{LS}}, E_{\mathbb{LS}})$ that captures all $\epsilon$-valid matches of signature $\mathbb{S}$
        in $\mathbb{L}$.
1 $V_{\mathbb{LS}} \leftarrow \emptyset$; $E_{\mathbb{LS}} \leftarrow \emptyset$;
2 for $i := 1$ to $m$ do
3     if $\widehat{p_i} == \widehat{q_1}$ then
4         $V_{\mathbb{LS}} \leftarrow V_{\mathbb{LS}} \cup \{v_{i,1}\}$;
5     for $j := 2$ to $n$ do
6         for $k := 1$ to $i - 1$ do
7             if $v_{k,j-1} \in V_{\mathbb{LS}}$ and $(p_k, p_i)$ is an $\epsilon_{j-1}$-valid match of $(q_{j-1}, q_j)$ then
8                 $V_{\mathbb{LS}} \leftarrow V_{\mathbb{LS}} \cup \{v_{i,j}\}$; $E_{\mathbb{LS}} \leftarrow E_{\mathbb{LS}} \cup \{(v_{i,j}, v_{k,j-1})\}$;
9 output DAG $G_{\mathbb{LS}}$.
```

For a given network traffic log $\mathbb{L} = (p_1, p_2, \ldots, p_m)$ and signature $\mathbb{S} = (q_1, q_2, \ldots, q_n)$, together with a inter-packet time interval tolerance vector ϵ, we compute a DAG $G_{\mathbb{LS}} = (V_{\mathbb{LS}}, E_{\mathbb{LS}})$ that captures all ϵ-valid matches of signature $\mathbb{S}$ in log $\mathbb{L}$. The vertex set $V_{\mathbb{LS}}$ contains vertices in the form of $v_{i,j}$, where p_i is a *potential match* of q_j. The edge set $E_{\mathbb{LS}}$ contains directed edges in the form of $(v_{i,j}, v_{k,j-1})$, where (p_k, p_i) is an ϵ_{j-1}-valid match of (q_{j-1}, q_j) for $1 \leq k \leq i-1$, and there is a directed path from vertex $v_{i,j}$ to a vertex $v_{i',1} \in V_{\mathbb{LS}}$ (for some $i' \leq i - j + 1$).

If $\widehat{p_i} \neq \widehat{q_j}$, vertex $v_{i,j}$ *does not* exist. If $\widehat{p_i} = \widehat{q_j}$, vertex $v_{i,j}$ *may* exist. Each edge has the form $(v_{i,j}, v_{k,j-1})$ for some $k < i$. Hence we have $|V_{\mathbb{LS}}| \leq mn$ and $|E_{\mathbb{LS}}| \leq \frac{m(m-1)(n-1)}{2}$.

In Line 1 of Algorithm 3.1, both $V_{\mathbb{LS}}$ and $E_{\mathbb{LS}}$ are initialized to $\emptyset$. The algorithm then populates the vertex set and the edge set while looping over the packets $p_1, p_2, \ldots, p_m$. For each i, the algorithm loops over the packets $q_1, q_2, \ldots, q_n$. When $\widehat{p_i} = \widehat{q_1}$, $v_{i,1}$ is a vertex in the DAG. For $j = 2, 3, \ldots, n$, $v_{i,j}$ is a vertex if and only if $\widehat{p_i} = \widehat{q_j}$ **and** (p_k, p_i) is an ϵ_{j-1}-valid match of (q_{j-1}, q_j) for some $k < i$. In this case, $(v_{i,j}, v_{k,j-1})$ is an edge in the DAG.

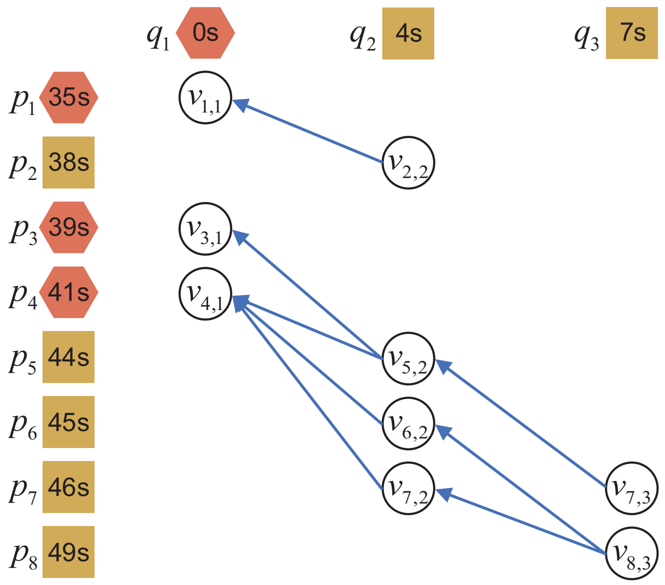


Figure 3.5. Running Example of `sigMatch`: Row Index Corresponds to the Traffic Log, Column Index Corresponds to the Signature

We use Figure 3.5 to illustrate a running example of `sigMatch`. The goal is to identify all ϵ-valid matches of signature $\mathbb{S} = (q_1, q_2, q_3)$ in log $\mathbb{L} = (p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8)$ with tolerance vector $\epsilon = (1, 1)$. In this example, we have $\widehat{p_1} = \widehat{p_3} = \widehat{p_4} = \widehat{q_1}$, denoted by the *hexagon* shape; we also have $\widehat{p_2} = \widehat{p_5} = \widehat{p_6} = \widehat{p_7} = \widehat{p_8} = \widehat{q_2} = \widehat{p_3}$, denoted by the *square* shape. The timestamps (for traffic log) and relative timestamps (for signature) are inside the corresponding shape.

We start from p_1. Since $\widehat{p_1} = \widehat{q_1}$, vertex $v_{1,1}$ is added to $V_{\mathbb{LS}}$; Then we move to p_2. Since $\widehat{p_2} = \widehat{q_2}$, $v_{1,1} \in V_{LS}$, and $|(p_2.t - p_1.t) - (q_2.t - q_1.t)| = |(38 - 35) - (4 - 0)| \leq 1$, vertex $v_{2,2}$ is added to $V_{\mathbb{LS}}$ and directed edge $(v_{2,2}, v_{1,1})$ is added to $E_{\mathbb{LS}}$. Similarly, vertices $v_{3,1}$ and $v_{4,1}$ are added to $V_{\mathbb{LS}}$. Next, we pay attention to the row corresponding to p_5. We found $\widehat{p_5} = \widehat{q_2}$. For $k = 1$, we found $v_{1,1} \in V_{\mathbb{LS}}$, but the time interval does not match. For $k = 3$, we found $v_{3,1} \in V_{\mathbb{LS}}$, and the time interval matches. Hence vertex $v_{5,2}$ is added to $V_{\mathbb{LS}}$, and edge $(v_{5,2}, v_{3,1})$ is added to $E_{\mathbb{LS}}$. For $k = 4$, we found $v_{4,1} \in V_{\mathbb{LS}}$, and the time interval matches. At this moment, vertex $v_{5,2}$ is already in $V_{\mathbb{LS}}$, and edge $(v_{5,2}, v_{4,1})$ is added to $E_{\mathbb{LS}}$. We obtain the DAG as shown in Figure 3.5 by continuing the above process.

Theorem 3.1. Algorithm `sigMatch` has a worst-case time complexity of $O(m^2 n)$, where n is the number of packets in the signature $\mathbb{S}$, and m is the number of packets in the network traffic log $\mathbb{L}$. Furthermore,

(a) If $(p_{l[1]}, p_{l[2]}, \ldots, p_{l[n]})$ is an ϵ-valid match of $\mathbb{S}$, then $(v_{l[n],n}, v_{l[n-1],n-1}, \ldots, v_{l[1],1})$ is a directed path in $G_{\mathbb{LS}}$, and $l[1] < l[2] < \cdots < l[n]$.

(b) If $(v_{l[n],n}, v_{l[n-1],n-1}, \ldots, v_{l[1],1})$ is a directed path in $G_{\mathbb{LS}}$, then $(p_{l[1]}, p_{l[2]}, \ldots, p_{l[n]})$ is an ϵ-valid match of $\mathbb{S}$ in $\mathbb{L}$, and $l[1] < l[2] < \cdots < l[n]$.

Proof. The loop over i runs m times. The loop over j runs n times. The loop over k runs $i - 1$ times, for each i. This leads to the worst-case time complexity of $O(m^2 n)$.

From the condition in Line 7 of the algorithm, we notice that *there is an edge in the form* $(v_{i,j}, v_{k,j-1})$ **if and only if** *there is an* $\epsilon_{[j]}$*-valid match of* $(q_1, q_2, \ldots, q_j)$ *in* $\mathbb{L}$

that matches (q_{j-1}, q_j) *to* (p_k, p_i), where $\epsilon_{[j]} = (\epsilon_1, \epsilon_2, \ldots, \epsilon_{j-1})$. This leads to claims (a) and (b). □

We point out that the total number of ϵ-valid matches of signature $\mathbb{S}$ in $\mathbb{L}$ may be exponential. However, all of them are captured by a polynomial sized DAG $G_{\mathbb{L}\mathbb{S}}$, which can be computed in polynomial time.

3.4.3 Unveiling IoT Device Events from Network Traffic Log

We investigate how to unveil the events of an IoT device using `sigMatch` in Algorithm 3.1 as a building block. Note that we can separate the traffic of a specific IoT device from all network traffic using the IoT device's distinct IP address. For a given IoT device, we first extract its signature set $\mathbb{S}\mathbb{S} = \{\mathbb{S}^1, \mathbb{S}^2, \ldots, \mathbb{S}^K\}$. For each signature $\mathbb{S}^k$, using its corresponding tolerance vector ϵ^k, we can apply `sigMatch` to construct the corresponding DAG $G_{\mathbb{L}\mathbb{S}^k}$ in $O(m^2 n_k)$ worst-case time, where n_k is the number of packets in $\mathbb{S}^k$. We can compute all K DAGs in $O(Km^2 n_{\max})$ worst-case time, where $n_{\max} = \max\{n_1, n_2, \ldots, n_K\}$.

For each $k = 1, 2, \ldots, K$, there may be zero or more ϵ^k-valid matches of signature $\mathbb{S}^k$. Making use of $G_{\mathbb{L}\mathbb{S}^k}$, we can either confirm that there is no ϵ^k-valid match (when there is no vertex v_{i,n_k} in $V_{\mathbb{L}\mathbb{S}^k}$) or compute *the earliest* ϵ^k-valid match $(p_{l[1]}, p_{l[2]}, \ldots, p_{l[n_k]})$, in the sense that $(p_{l[1]}, p_{l[2]}, \ldots, p_{l[n_k]})$ is *lexicographically smallest*, in $O(m+n_k)$ worst-case time.

Given the network traffic $\mathbb{L}$, and the valid matches of signatures in $\mathbb{S}\mathbb{S}$, how do we decide which IoT device event happened first? Through extensive experiments, we found that *in normal situations, each network packet corresponding to an earlier IoT device event proceeds every network packet corresponding to a later IoT device event.*

Therefore the signature that has the earliest match happens first. Once this decision is made, we can delete each packet with a timestamp no later than that of the last packet in the match of the found signature from the network traffic. Repeating the above process, we can unveil the sequence of IoT device events from the given network traffic. We formally describe this process called `actExtract` in Algorithm 3.2.

Algorithm 3.2: actExtract$(\mathbb{L}, \mathbb{SS}, \varepsilon)$

```
Input: Network traffic $\mathbb{L} = (p_1, p_2, \ldots, p_m)$, signature set $\mathbb{SS} = \{\mathbb{S}^1, \mathbb{S}^2, \ldots, \mathbb{S}^K\}$,
       $\varepsilon = (\epsilon^1, \epsilon^2, \ldots, \epsilon^K)$ where $\epsilon^k$ is the match tolerance vector for $\mathbb{S}^k$.
Output: A sequence of IoT device events $\mathbb{A}_1, \mathbb{A}_2, \ldots$.
1 for $k := 1$ to $K$ do
2     $G_{\mathbb{S}^k} \leftarrow sigMatch(\mathbb{L}, \mathbb{S}^k, \epsilon^k)$;
3 while some signature $\mathbb{S}^k$ has a match in $G_{\mathbb{S}^k}$ do
4     Let $\mathbb{S}^{k'}$ have the earliest match;
5     output Device event corresponding to $\mathbb{S}^{k'}$;
6     Remove all packets in $\mathbb{L}$ with timestamp no later than that of the last matched
        packet for $\mathbb{S}^{k'}$.
7     for $k := 1$ to $K$ do
8         $G_{\mathbb{S}^k} \leftarrow sigMatch(\mathbb{L}, \mathbb{S}^k, \epsilon^k)$;
```

Theorem 3.2. The worst-case time complexity of Algorithm 3.2 is $O(Km^3 n_{\max})$, where K is the number of signatures, $n_{\max}$ is the maximum number of packets in any of the signatures, and m is the number of packets in network traffic log $\mathbb{L}$. In normal situations (i.e., each packet for an earlier device event precedes every network packet of a later device event), `actExtract` correctly outputs a sequence of IoT device events $\mathbb{A}_1, \mathbb{A}_2, \ldots$ whose sequential execution will generate a network traffic log that may be different from $\mathbb{L}$ only in the timestamp fields.

Proof. Initially, the K DAGs can be computed in $O(Km^2 n_{\max})$ time. The earliest match of $\mathbb{S}^k$ can be computed in $O(m + n_k)$ time, $\forall k$. Selecting the signature with the earliest match requires $O(Kn_{\max})$ time. This process is repeated for no more than m times, hence the time complexity.

Next, we prove the correctness of the algorithm. Assuming that the sequence of IoT device events that generated the network traffic $\mathbb{L}$ is $\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_x$. By our *normal* assumption, each network packet of $\mathbb{A}_1$ must happen earlier than every network packet of $\mathbb{A}_\lambda$, for any $\lambda > 1$. Since `actExtract` uses the earliest match, it will output $\mathbb{A}_1$ as the first device event, and all of the packets in the computed match for $\mathbb{A}_1$ have timestamps earlier than the timestamp of any packet in other IoT device event $\mathbb{A}_\lambda$, with $\lambda > 1$. Hence, when we delete the packets matched for $\mathbb{A}_1$, we delete all of the packets generated for $\mathbb{A}_1$, but none of the packets generated by $\mathbb{A}_\lambda$ with $\lambda > 1$. Therefore `actExtract` will next output $\mathbb{A}_2$, then $\mathbb{A}_3$, and so on. This proves the correctness of the algorithm. □

When we execute the computed sequence of IoT device events, the network traffic observed may be different from $\mathbb{L}$, but only in the timestamp field. The sequence of packets will have increasing timestamps. Ignoring the timestamp field, two sequences of packets will be identical with $\mathbb{L}$. Note that we can divide the network traffic log into multiple sublogs where each sublog corresponds to a unique IoT device. We can apply `actExtract` to each sublog in parallel to *unveil the IoT device events.*

3.4.4 Discussions

Our proposed `actExtract` algorithm can unveil the event sequence of an IoT device with no ambiguity and guarantee correctness, assuming there is no ongoing attack and the device can only carry out one event at a time, which is true for most devices. For devices that allow two or more concurrent events, such as IP cameras, we can modify `sigMatch` algorithm to record only non-overlapping matches in network traffic for a signature in a DAG. We can then build the DAG for the same device events' signatures independently and output all the identified events. In case when there is attacking traffic, it is possible that one packet is matched to two different signatures. We can add a variable in `sigMatch` to record all the signatures that a packet is matched to. If such a conflict happens, our algorithm can report it as an anomaly and raise an alarm.

3.5 Experimental Evaluations

We evaluate the performance of IoTAthena using two different settings: 1) our own smart home testbed, and 2) a large public IoT network traffic dataset [109]. We first describe these settings in Section 3.5.1. In Section 3.5.3, we present experimental results on the sensitivity of IoTAthena's accuracy on the matching tolerance. In Section 3.5.4, we present experimental results for homogeneous IoT device events. In Section 3.5.5, we present experimental results for mixed IoT device events, together with a case study.

3.5.1 Experiment Setting

Our smart home testbed has 16 widely-used IoT devices, including multiple models of *IP cameras, smart bulbs, smart doorbells, smart locks, and smart plugs.* These IoT devices are all ranked as popular by Smart Home DB [128]. Our experiments have identified a total of 44 different IoT device events by using these 16 devices. The numbers of IoT devices and device events in this study are comparable to existing studies on understanding IoT device events in smart home network environments [1, 135]. This controlled smart home environment was used to create the "silent" week for collecting, analyzing, and characterizing background network traffic, as discussed in Section 3.2. For each IoT device event, we repeatedly generate the event while collecting the associated network traffic as well as recording the event logs which are used for establishing the ground truth at the same time. Using the signature extraction technique introduced in Section 3.3, we were able to extract signatures for most of the IoT devices event of all 16 representative IoT devices except the *stream off* event of Amcrest ProHD camera, which does not have the deterministic traffic pattern to form a signature.

In addition to network traffic and event logs collected from our own smart home testbed, we also evaluated IoTAthena's performance using a large public IoT network traffic dataset [109], known as the MON(IOT)R dataset. The MON(IOT)R dataset includes raw IP data traffic and the labeled event logs of 25 IoT devices[2]. Among these devices, 6 of them are also included in our smart home testbed, while the other 19 devices are unique to the dataset. The IoT network traffic and labeled device events in the dataset allow us to evaluate the performance of IoTAthena.

3.5.2 Evaluation Metrics

To evaluate and quantify the performance of our proposed algorithms, we use the widely used metrics, i.e., true positives ($\mathcal{TP}$), false positive ($\mathcal{FP}$), false negative ($\mathcal{FN}$), and true negative ($\mathcal{TN}$). In addition to these four measures from the confusion

[2]We evaluated IoTAthena on the IoT device events with at least 30 samples in the MON(IOT)R dataset in order to have statistically meaningful results.

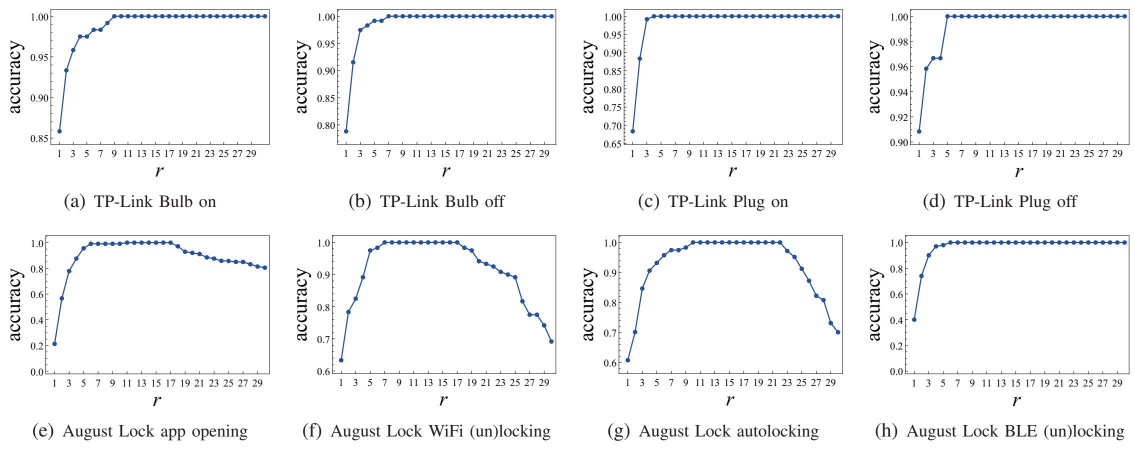
matrix, we also use precision ($\mathcal{P}$), recall ($\mathcal{R}$), and accuracy ($\mathcal{A}$) to understand the overall quality of our user activity inferring experiments. The precision is calculated as $\mathcal{P} = \frac{\mathcal{TP}}{\mathcal{TP}+\mathcal{FP}}$, while the recall is $\mathcal{R} = \frac{\mathcal{TP}}{\mathcal{TP}+\mathcal{FN}}$. The accuracy can be calculated as $\mathcal{A} = \frac{\mathcal{TP}+\mathcal{TN}}{\mathcal{TP}+\mathcal{TN}+\mathcal{FP}+\mathcal{FN}}$.

3.5.3 Sensitivity Analysis on the Tolerance Parameter

Our time-sensitive subsequence matching algorithm `sigMatch` uses the tolerance vector $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_{n-1})$ for accommodating inter-packet time intervals' variations. In our experiment, ϵ_j is set to $r \times \sigma_j$ for $j \in [1, n-1]$, where σ_j is the standard deviation of the inter-packet time interval between two consecutive packets q_j and q_{j+1} and $r \geq 1$ is a tunable parameter.

The accuracy of IoTAthena depends on the matching tolerance parameter. Intuitively, when the matching tolerance is very small, IoTAthena tends to unveil fewer device events due to the strict checking of inter-packet time intervals, leading to low accuracy. On the other hand, when the matching tolerance is very large, IoTAthena tends to have more false negatives due to the loose checking of inter-packet time intervals, also leading to low accuracy. In order to have a deeper understanding of this dependency, we carried out sensitivity analysis. For each IoT device event, we repeatedly triggered it 120 times with random delays between two consecutive experiments. We then ran 6-fold cross validation using the data collected. In each of the 6 rounds, we observe the accuracy of IoTAthena on 20 of the experiments, while using the remaining 100 to generate the signature. Figure 3.6 illustrates some representative results.

Figure 3.6(a)-(d) show the accuracy changes for unveiling the *on* and *off* events of


(a) TP-Link Bulb on (b) TP-Link Bulb off (c) TP-Link Plug on (d) TP-Link Plug off

(e) August Lock app opening (f) August Lock WiFi (un)locking (g) August Lock autolocking (h) August Lock BLE (un)locking

Figure 3.6. The Impact of r in the Inter-Packet Time Interval Tolerance Parameter on the Accuracy of IoT Device Event Signature Matching

TP-Link Bulb and TP-Link Plug as r increases from 1 to 30, while Figure 3.6(e)-(h) show the accuracy dynamics of unveiling August Lock's app opening, Wifi (un)locking, autolocking, and Bluetooth (un)locking events. In Figure 3.6(a)-(d), we observe that the accuracy of IoTAthena exhibits a non-decreasing trend for the *on* and *off* events of both TP-Link Bulb and TP-Link Plug, when r increases from 1 to 30. These observations are not surprising since the increasing value of r leads to a higher tolerance value to allow larger inter-packet time intervals.

However, Figure 3.6(e)-(g) contradict such conjectures as increasing r to a particular value leads to decreasing accuracy in matching August Lock's app opening, WiFi (un)locking, and autolocking events. Our in-depth investigation discovered that the accuracy decrease for the larger r values is due to the interference of August Lock's background traffic noise. The background traffic happens to shares overlapping packets with the signatures of app opening, WiFi (un)locking, and autolocking events when we allow bigger tolerance of inter-packet intervals. Unlike Figure 3.6(e)-(g), the accuracy of unveiling August Lock's Bluetooth (un)locking events changes in a non-decreasing fashion in Figure 3.6(h) as r increases from 1 to 30. The underlying reason of this distinct observation is the unique traffic patterns of the network traffic collected when triggering August Lock's Bluetooth (un)locking events, where no background noise that cannot be filtered out has been observed.

In summary, our sensitivity analysis on the tolerance parameter confirmed the importance and impact of choosing appropriate tolerance values during the IoT device event extraction process. More importantly, the observations in Figure 3.6 highlight the rationale and necessity of our *full* packet sequences with inter-packet time intervals as the IoT device event signature and our time-sensitive subsequence matching algorithm for unveiling IoT device events.

3.5.4 Performance of IoTAthena on Homogeneous IoT Device Events

Having done the sensitivity analysis, we focused on evaluating IoTAthena's performance of unveiling homogeneous IoT device events. We first present experimental results on our smart home testbed. We then present results on the MON(IOT)R dataset [109].

For each IoT device event, we repeated it 120 times and collected the network traffic on the router in our smart home testbed. We again ran the 6-fold cross validation. Table 3.1 shows the accuracy ($\mathcal{A}$), precision ($\mathcal{P}$), and recall ($\mathcal{R}$) measures of IoTAthena for various events of the 16 IoT devices in our smart home testbed, with r set to 3, 11, and 23, respectively.

From Table 3.1, we observe that the performance of IoTAthena depends on r. For $r = 3$, IoTAthena achieves a minimum accuracy of 0.78, a minimum precision of 0.98, and a minimum recall of 0.78. When r is increased to 11, the performance of IoTAthena improves, with precision of 1.00, accuracy of 0.99 or better, and recall

Table 3.1. Accuracy, Precision, and Recall of IoTAthena for 16 Devices with r Set as 3, 11, 23 Respectively.

Type	Device	Event	$r = 3$			$r = 11$			$r = 23$		
			$\mathcal{A}$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{R}$
bulb	Philips Hue	on or off	0.98	1.00	0.98	1.00	1.00	1.00	1.00	1.00	1.00
		brightness	0.95	1.00	0.95	1.00	1.00	1.00	1.00	1.00	1.00
	Sengled LED	on	0.88	1.00	0.88	1.00	1.00	1.00	1.00	1.00	1.00
		off	0.89	1.00	0.89	1.00	1.00	1.00	1.00	1.00	1.00
		brightness	0.92	1.00	0.92	1.00	1.00	1.00	1.00	1.00	1.00
	TP-Link Bulb	on	0.96	1.00	0.96	1.00	1.00	1.00	1.00	1.00	1.00
		off	0.97	1.00	0.97	1.00	1.00	1.00	1.00	1.00	1.00
		color	0.99	1.00	0.99	1.00	1.00	1.00	1.00	1.00	1.00
		brightness	0.95	1.00	0.95	1.00	1.00	1.00	1.00	1.00	1.00
camera	Amcrest ProHD	stream on	0.86	1.00	0.86	1.00	1.00	1.00	1.00	1.00	1.00
	Arlo - Q Indoor	stream on	0.96	1.00	0.96	1.00	1.00	1.00	1.00	1.00	1.00
		stream off	0.94	1.00	0.94	1.00	1.00	1.00	1.00	1.00	1.00
		motion detection	0.98	1.00	0.98	1.00	1.00	1.00	1.00	1.00	1.00
	Arlo Ultra	stream on	0.88	1.00	0.88	1.00	1.00	1.00	1.00	1.00	1.00
		stream off	0.92	1.00	0.92	1.00	1.00	1.00	1.00	1.00	1.00
		motion detection	0.95	1.00	0.95	1.00	1.00	1.00	1.00	1.00	1.00
	Blink XT2	stream on	0.92	1.00	0.92	0.99	1.00	0.99	0.99	1.00	0.99
		stream off	0.95	1.00	0.95	1.00	1.00	1.00	1.00	1.00	1.00
		motion detection	0.96	1.00	0.96	1.00	1.00	1.00	1.00	1.00	1.00
	Reolink Camera	stream on	0.98	1.00	0.98	1.00	1.00	1.00	1.00	1.00	1.00
		stream off	0.96	1.00	0.96	1.00	1.00	1.00	1.00	1.00	1.00
		motion detection	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
doorbell	August Doorbell	stream on	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
		stream off	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
		ringing	0.94	1.00	0.94	1.00	1.00	1.00	1.00	1.00	1.00
		motion detection	0.95	1.00	0.95	1.00	1.00	1.00	1.00	1.00	1.00
	Ring Doorbell	stream on	0.98	1.00	0.98	1.00	1.00	1.00	1.00	1.00	1.00
		stream off	0.96	1.00	0.96	1.00	1.00	1.00	1.00	1.00	1.00
		ringing	0.96	1.00	0.96	1.00	1.00	1.00	1.00	1.00	1.00
		motion detection	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
lock	August Lock Pro	app opening	0.78	1.00	0.78	1.00	1.00	1.00	0.77	0.86	0.88
		WiFi (un)locking	0.80	0.98	0.82	1.00	1.00	1.00	0.75	0.81	0.91
		Bluetooth (un)locking	0.90	1.00	0.90	1.00	1.00	1.00	0.99	0.99	1.00
		autolocking	0.85	1.00	0.85	1.00	1.00	1.00	0.97	1.00	0.97
		manual (un)locking	0.93	1.00	0.93	1.00	1.00	1.00	1.00	1.00	1.00
	Schlage Deadbolt	WiFi (un)locking	0.89	1.00	0.89	1.00	1.00	1.00	1.00	1.00	1.00
		autolocking	0.92	1.00	0.92	1.00	1.00	1.00	1.00	1.00	1.00
		manual (un)locking	0.90	1.00	0.90	1.00	1.00	1.00	1.00	1.00	1.00
plug	Amazon Plug	on	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
		off	0.96	1.00	0.96	1.00	1.00	1.00	1.00	1.00	1.00
	Gosund Socket	on or off	0.98	1.00	0.98	1.00	1.00	1.00	0.99	0.99	1.00
	TP-Link Plug	on	0.99	1.00	0.99	1.00	1.00	1.00	1.00	1.00	1.00
		off	0.97	1.00	0.97	1.00	1.00	1.00	1.00	1.00	1.00
	WeMo Plug	on or off	0.98	1.00	0.98	1.00	1.00	1.00	0.99	0.99	1.00

of 0.99 or better, across all IoT device events in our experiments. When r is further increased to 23, the performance of IoTAthena drops, with a minimum accuracy of 0.75, a minimum precision of 0.81, and a minimum recall of 0.88. Based on this empirical evidence, we choose $r = 11$ as the "optimal" value for the tolerance parameter.

Table 3.2. Accuracy, Precision, and Recall of IoTAthena in the External MON(IOT)R Dataset with r Set as 11

Device	Event	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{R}$
Amcrest Camera Wired	watch	1.00	1.00	1.00
Blink Camera	watch	1.00	1.00	1.00
Blink Security Hub	watch or photo	1.00	1.00	1.00
Bulb1	on or off	1.00	1.00	1.00
Fire TV	menu	0.95	1.00	0.95
Google Home Mini	volume or voice	1.00	1.00	1.00
Insteon Hub	on or off	1.00	1.00	1.00
Invoke	volume or voice	1.00	1.00	1.00
Lefun Camera Wired	watch or photo	1.00	1.00	1.00
LG TV Wired	menu	1.00	1.00	1.00
Lightify Hub	on or off	1.00	1.00	1.00
	color	1.00	1.00	1.00
Luohe Spycam	watch	1.00	1.00	1.00
Magichome Strip	on	0.98	0.98	1.00
	off	1.00	1.00	1.00
Microseven Camera	watch	1.00	1.00	1.00
Philips Bulb	on or off	0.97	1.00	0.97
Samsungtv Wired	menu	1.00	1.00	1.00
Sengled Hub	on or off	0.98	1.00	0.98
Smartthings Hub	on or off	1.00	1.00	1.00
T-philips Hub	on or off	1.00	1.00	1.00
TP Link Bulb	on	1.00	1.00	1.00
	off	1.00	1.00	1.00
	color	1.00	1.00	1.00
	dim	1.00	1.00	1.00
TP Link Plug	on	1.00	1.00	1.00
	off	1.00	1.00	1.00
Wink Hub2	on	1.00	1.00	1.00
	off	1.00	1.00	1.00
Xiaomi Hub	on or off	0.98	0.98	1.00
Xiaomi Strip	on or off	1.00	1.00	1.00
Zmodo Doorbell	watch	1.00	1.00	1.00

We also evaluated the performance of IoTAthena using the MON(IOT)R dataset [109]. Due to the relatively small sample size (between 30 and 40), we ran 4-fold cross validation instead of 6-fold cross validation. Table 3.2 illustrates the accuracy, precision, and recall measures of running IoTAthena against 25 IoT devices in the MON(IOT)R dataset with r set to 11. We observe that IoTAthena achieve a minimum accuracy of 0.95, a minimum precision of 0.98, and a minimum recall of 0.95.

The prior study [135] also evaluates the algorithm with the same the MON(IOT)R dataset [109]. Table VI in [135] reports an average accuracy of 99.12% on 3 IoT device events for WAN Sniffer, and an average accuracy of 99.06% on 4 IoT device events for WiFi sniffer. As shown in Table 3.2, our approach achieves an average accuracy of 99.57% on 33 IoT device events on the same dataset. Therefore, our proposed IoTAthena system is able to generate signatures for more IoT device events than [135] while achieving slightly better accuracy in identifying the signatures with the same public dataset.

In summary, experimental evaluations with our smart home testbed and the MON(IOT)R dataset demonstrate that IoTAthena can successfully unveil homogeneous IoT device events from network traffic logs.

3.5.5 Performance of IoTAthena on Mixed IoT Device Events

A significant benefit of our IoTAthena system lies in the realtime security monitoring of IoT devices in smart homes, which has become an increasingly important research topic. Given the IoT network traffic logs, IoTAthena can accurately unveil the sequence of IoT device events over time and potentially detect anomalous traffic patterns and behaviors towards IoT devices.

As a case study, we applied IoTAthena to unveil the device event of 5 IoT devices in our smart home during a 24-hour span. The 5 devices in this case study consist of Arlo Ultra Camera, August Lock, Ring Doorbell, TP-Link Bulb, and TP-Link

Plug. Figure 3.7 visualizes the time-series events of these 5 IoT devices discovered by IoTAthena during a 24-hour time span in our smart home environment.

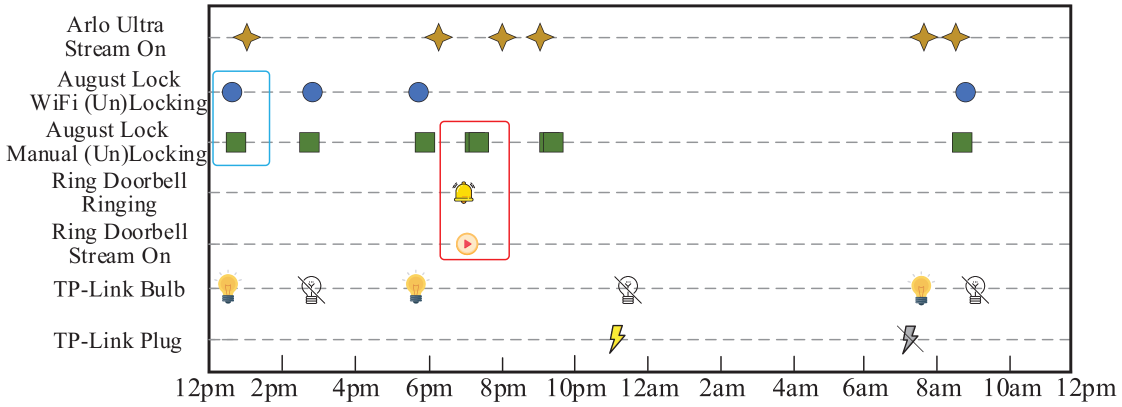


Figure 3.7. IoT Device Events Discovered by IoTAthena in the Small Home Environment During a 24-Hour Span

The two device events highlighted by the light blue box near the left end of Figure 3.7 capture two consecutive user-triggered events at around 12:45pm: i) (the homeowner) unlocked the August Lock with app (from outside), indicated by the dark blue disk inside the light blue box, and ii) manually locked the August Lock (after entering home), indicated by the green square inside the light blue box. Similarly, the four IoT device events highlighted by the red box at around 7:20pm in Figure 3.7 reflect four consecutive events: i) (a visitor) pressed the button on the ring doorbell, which generated a push notification to the homeowner's smartphone; ii) (the homeowner) watched the video streaming feed on the ring doorbell to check the visitor's identity; iii) (the homeowner) manually unlocked the August Lock to let the visitor in; iv) the August Lock was manually locked (from inside).

To evaluate IoTAthena's ability in unveiling sequences of mixed IoT device events, we used IoTAthena to unveil the mixed IoT device event of all 16 devices in our smart home from the network traffic, in a 24-hour span, from 12:00pm to 11:59am.

Figure 3.8 illustrates IoTAthena's performance by comparing the ground truth event sequences with the unveiled event sequences of 16 IoT devices in the smart home testbed, where a blue dot represents a successful match, while a red cross represents a failed match. The actual dates for running different IoT device event experiments might vary, so the x-axis only denotes the time of the day from 12:00pm to 11:59am. As can be seen from the figure, IoTAthena correctly unveiled all but one of the events. The only missed event occurs with the Blink XT2 Camera. Our root cause analysis revealed that IoTAthena missed one streaming event due to the unseen variation in packet length.

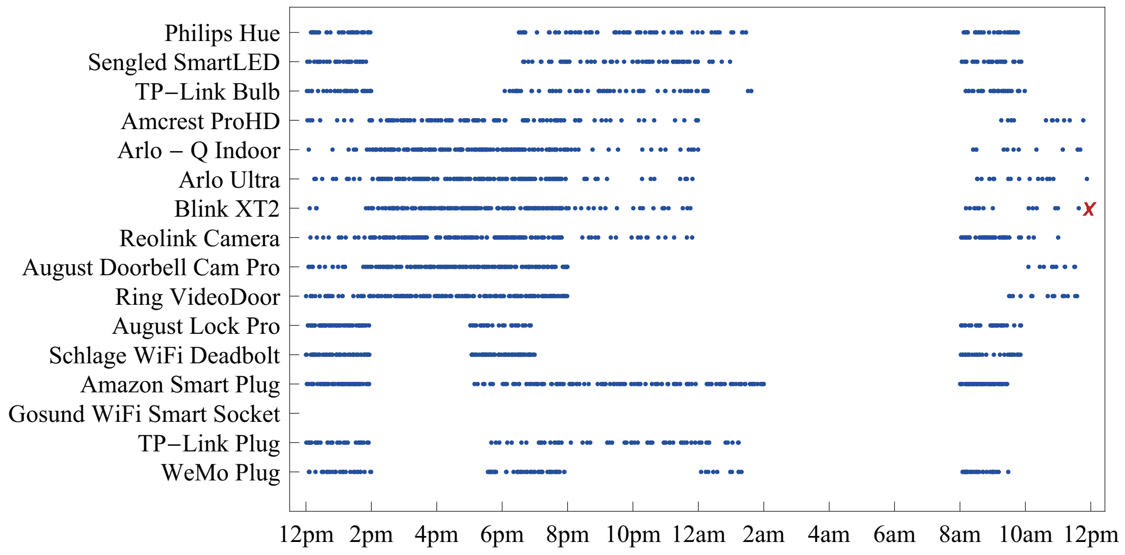


Figure 3.8. Device Event Sequence Extraction Results of 16 IoT Devices in the Smart Home Environment Where X-Axis Denotes the Time of the Day From 12:00pm to 11:59am

In summary, our experimental evaluations based on a variety of heterogeneous IoT devices demonstrated that IoTAthena can effectively and accurately unveil individual IoT device events as well as unveil IoT device event sequences over time.

3.6 Related Work

The recent growth and deployment of IoT devices in smart homes have attracted the networking research community to study the traffic characterization and behavioral fingerprinting of IoT devices, and explore network traffic to discover IoT devices' types and events. Most of the existing studies [49, 53, 54, 62, 64, 79, 86, 103, 109, 118, 125, 155, 160] in IoT traffic characterization and fingerprinting are interested in a wide range of traffic features from TCP/IP protocols as well as from IoT wireless communication channels. For example, the research in [62] utilizes the wireless radio propagation patterns of IoT devices for secure authentication. Two recent researches [155, 160] explore the captured WiFi signals in home network for applications of localization and positioning. These prior researches provide critical insights for understanding traffic patterns of heterogeneous IoT devices and identifying IoT device models or types for IoT device discovery and management, IoT application performance monitoring, and vulnerability and security analysis.

In light of the recent IoT Botnets exploiting and control thousands of vulnerable IoT devices [5, 9, 24, 42, 44, 57, 72, 89, 91, 101, 133, 141, 154], some research efforts have proposed innovative methods of classifying IoT devices based on machine learning, statistical inference, or passive traffic measurement [11, 86, 126]. For example, IoTSentinel [86] first extracts 23 traffic features of IoT network traffic, and subsequently builds Random Forest classifiers to identify IoT device types. Similarly, IoTSense [11] fingerprints the behaviors of IoT device types with feature vectors from packet headers and payload, and builds several machine learning classifiers for effectively detecting IoT device types based on the trained behavioral fingerprinting. The research in [126] first monitors a smart IoT environment with various IoT devices for six months for extensive IoT network traffic analysis, and then builds a machine learning framework for classifying IoT device types.

As homeowners continue to deploy smart home IoT devices such as smart locks and security cameras for mission-critical applications, accurately identifying IoT device events via supervised machine learning models [1, 96] and deterministic inference [135, 156] becomes an urgent research problem. For example, HomeSnitch [96] constructs bidirectional application data unit exchanges for representing IoT application behaviors and applies supervised machine learning classifiers to classify IoT application behaviors and identifying unknown behaviors. Similarly, Peek-a-Boo [1] demonstrates the feasibility of identifying the types, states, and IoT device events via machine learning techniques from an attacker's perspective. The closest work to ours is PingPong [135], which explores the sequential and directional "ping/pong" behavioral patterns between cloud servers and IoT devices or between cloud servers and smartphones. The experiments in [135] have shown that the simple ping/pong packet-pairs with payload size and traffic directions can effectively detect many IoT devices events. HoMonit [156], another work of deterministically detecting IoT device event, monitors encrypted wireless traffic of some home apps and infers smart app activities based on the deterministic finite automaton (DFA) model of smart app behavior and wireless side-channel analysis. IoTGaze [53] also builds up a system to identify IoT device events using the sniffed wireless traffic.

Inspired and motivated by these studies on identifying IoT device types and/or events, our proposed IoTAthena system is focused on understanding traffic signatures of IoT device events and accurately extracting device events from IoT network traffic. The insights from the unveiled IoT device events have a broad range of applications such as anomaly detection, e.g., an unauthorized user is watching the video stream of the surveillance camera, IoT device malfunction detection, e.g., a smart plug shows two consecutive *on* events, and smart home safety, e.g., the smart lock was unlocked remotely by an unauthorized user.

Note that our work is significantly different from [11, 86, 126] in the way that our objective is generating signatures for concrete IoT device events such as *on* or *off* events of a smart plug and unveiling these events from network traffic, instead of identifying IoT device models or types. Different from machine learning based solutions [1, 96], IoTAthena adopts a *white-box* approach to programmatically generate signatures of IoT device events consisting of ordered sequences of IP data packets with relative timestamps. IoTAthena's signature generation module is inspired by PingPong [135], but it generates a *full* signature for each IoT device event and introduces a novel time-sensitive subsequence matching approach for unveiling IoT device events from new IoT network traffic logs.

3.7 Conclusions

This study introduces IoTAthena to effectively and accurately unveil IoT device events from network traffic in smart homes. We first recognize and generate event signatures of IoT device events consisting of ordered sequences of IP data packets by repeated and controlled experiments. Subsequently, we design two polynomial time algorithms, `sigMatch` and `actExtract`. The `sigMatch` algorithm captures all matches of any given IoT device event signature from real network traffic logs. The `actExtract` algorithm unveils the full event sequences of all IoT devices from the network traffic log. Through experimental evaluations based on a wide range of heterogeneous IoT devices from a real smart home environment and a public IoT dataset, we demonstrated that IoTAthena is able to accurately unveil IoT device events from raw network traffic logs.

Chapter 4

IOTMOSAIC: INFERRING USER ACTIVITIES IN SMART HOMES

4.1 System Overview and Architecture

In this section, we first discuss the rationale for building a system of user activity inference from IoT network traffic in smart homes. Subsequently, we present the overall architecture of our proposed IoTMosaic system and its main components.

4.1.1 Towards A User Activity Inference System

With the prevalence and popularity of IoT devices, a wide range of innovative services and applications such as home automation, remote healthcare, and voice assistants are available for homeowners. These dedicated IoT devices mostly work independently for specific functions, e.g., a smart camera starts recording once sensing sound or motions, and a smart lock is opened or closed remotely via a companion smartphone app.

Many real-world user activities trigger a sequence of temporally and spatially *correlated* events involving multiple IoT devices. The individual device event information of each of these IoT devices often lacks sufficient evidence to infer the user activities and tell homeowners what happened in their homes. For example, a delivery personnel dropping off a package on the front door and ringing the Ring Doorbell could trigger the `motion detection` event and the doorbell's `ringing` event. However, exploring correlated events and collective insights from heterogeneous IoT devices in the same home could reveal many important user activities. Such automated user activity monitoring and inference system could have many practical benefits in home safety and security, assisted living, and remote healthcare.

Towards this end, this study proposes the IoTMosaic system for automatically and algorithmically inferring user activities based on the underlying network traffic of IoT devices in smart homes. Rather than inferring user activities directly from the IoT network traffic, IoTMosaic first detects IoT device events by analyzing network traffic in smart homes by adopting the solution introduced in Chapter 3. Based on these extracted IoT device events, IoTMosaic generates the signatures of diverse user activities which consist of IoT device event sequences. IoTMosaic then infers user activities using approximate matching algorithms to accommodate missing or out-of-order IoT device events due to device malfunctions or varying latencies.

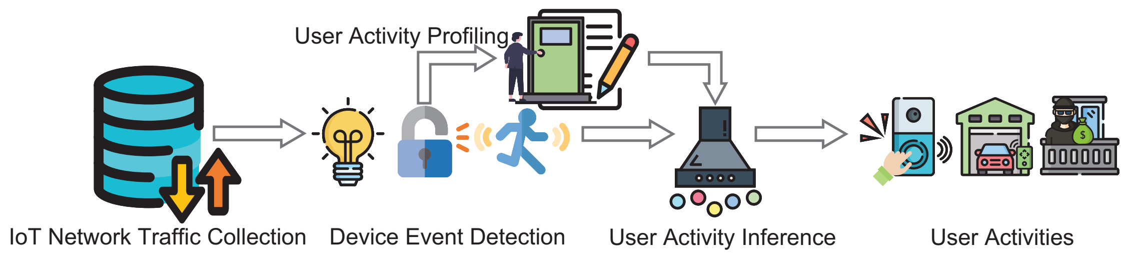


Figure 4.1. System Architecture of IoTMosaic for Inferring User Activity From IoT Network Traffic in Smart Homes

4.1.2 System Architecture and Components

Figure 4.1 illustrates IoTMosaic's overall system architecture for inferring user activity from IoT network traffic in smart homes. IoTMosaic consists of four main components: i) IoT network traffic collection, ii) IoT device event detection, iii) user activity profiling, and vi) user activity inference.

The first key system component, *IoT network traffic collection*, leverages programmable home routers to continuously collect, process, and analyze outgoing and incoming timestamped TCP/IP data packets of smart home IoT devices. For the second component, *IoT device event detection*, this work adopts the solution introduced in Chapter 3 for extracting IoT device events from network traffic collected by the home routers in the first component.

The primary focus of this study is to design and implement the last two system components in Figure 4.1, i.e., *user activity profiling* and *user activity inference.* For learning and profiling user activities, we first collect IoT network traffic in smart homes while repeatedly running and labeling each user activity as ground truth. Subsequently, we extract the sequence of IoT device events as the signature for each user activity. The *user activity profiling* component is responsible for recognizing and generating the signatures of different user activities which are the input of the next *user activity inference* component.

Towards developing the *user activity inference* system component, we propose simple yet effective algorithms for identifying user activities from IoT device events with the tolerance of missing events. Our proposed approximate matching algorithms effectively infer user activities with varying missing device events. In case of multiple matches to different user activities which share overlapping IoT device events, we devise a heuristic trimming step to strategically remove some inferred activities based on the number of missing device events and the dependency among the signatures of user activities.

To evaluate the performance of our proposed system, we set up an experimental smart home environment in a two-bedroom apartment with heterogeneous IoT devices. We systematically evaluate our system with different user activities, mostly related to home safety and security, in the smart home testbed. Our extensive experiments on the labeled user activities and network traffic data collected span over two months show that IoTMosaic is able to accurately infer diverse user activities in smart homes.

4.2 User Activities Profiling

4.2.1 Detecting IoT Device Events with IoT Network Traffic

The network traffic of IoT devices plays a crucial role in classifying the IoT device types, e.g., `LG smart TV`, and detecting IoT device events, e.g., switching Philips Hue smart lighting `on` or `off`. These individual and distinguished events generated by IoT devices are referred to as IoT device events.

In this study, we adopt the approach in Chapter 3 to first generate the *unique* signature of each IoT device event with diverse traffic features including the crucial inter-packet time interval information in IP packets and then extract IoT device events in our smart home environments. Table 4.1 lists the smart home IoT devices deployed in this study and their respective device events for learning and inferring a wide range of user activities related to home safety and security, e.g., a person with smart lock app access entering the home from the front door.

As IoT devices continue to be deployed in smart home applications such as smart locks and surveillance cameras, the knowledge and awareness of IoT device events have become increasingly important for understanding the statuses of IoT devices and detecting anomalous behaviors and attacks towards them. More importantly,

Table 4.1. Smart Home IoT Devices Deployed in This Study and Their Respective Device Events

Device Name	Device Event	Abbreviation
Arlo Q Camera (AQ)	motion detection	AQ_{mot}
August Lock (AL)	WiFi (un)locking	AL_{wlk}
	manual (Un)locking	AL_{mlk}
	auto locking	AL_{alk}
D-Link Water Sensor (DW)	water detected	DW_{wtr}
	water not detected	DW_{nwtr}
Kangaroo Motion Sensor (KM)	motion detection	KM_{mot}
Reolink Camera (RC)	motion detection	RC_{mot}
	stream on / off	RC_{on} / RC_{off}
Ring Doorbell (RD)	motion detection	RD_{mot}
	stream on / off	RD_{on} / RD_{off}
	ringing	RD_{ring}
Ring Spotlight (RS)	motion detection	RS_{mot}
	motion light on / off	RS_{on} / RS_{off}
Smart Life Contact Sensor (SC)	open / close	SC_{open} / SC_{close}
Tessan Contact Sensor (TC)	open / close	TC_{open} / TC_{close}
TP-Link Bulb (TB)	on / off	TB_{on} / TB_{off}
TP-Link Plug (TP)	on / off	TP_{on} / TP_{off}

multiple IoT device events, happening very closely in time and space, could *collectively* provide valuable knowledge for inferring user activities in smart homes. Inspired by this critical insight, IoTMosaic explores IoT device events for understanding, profiling, and inferring user activities in smart homes.

4.2.2 Profiling User Activities with IoT Device Event Sequences

Recognizing and tracking user activities and behaviors in smart environments have been a long-standing research problem [107] due to its importance in assisted living, remote healthcare, and home safety. In this study, we consider a user activity as the interaction between a person and the smart home environment, e.g., an e-commerce delivery personnel pushing the smart doorbell and leaving a package on the porch. The direct interaction between users and IoT devices, e.g., a user opening or closing the smart lock via the companion smartphone App, is only considered as IoT device events (or actions) instead of user activities. In other words, a user activity, consisting of several human actions, could trigger one or more IoT device events, determined by the availability and deployment of IoT devices as well as the layout of the smart home.

Our real-world experiments with heterogeneous IoT devices deployed in the smart home testbed have discovered that many user activities trigger an ordered sequence of device events from several adjacent IoT devices. For example, a person with the physical key entering our smart home environment from the front door triggers a series of IoT device events related to August smart lock, Tessan Contact Sensor, Ring doorbell, Ring Spotlight, and Alro Q Camera. It should be noted that a single IoT device event is often unable to infer the underlying user activities independently. However, combining and correlating the time and space of multiple events from adjacent IoT devices that are deployed at the nearby locations where user activities happen potentially provide sufficient information for extracting these user activities.

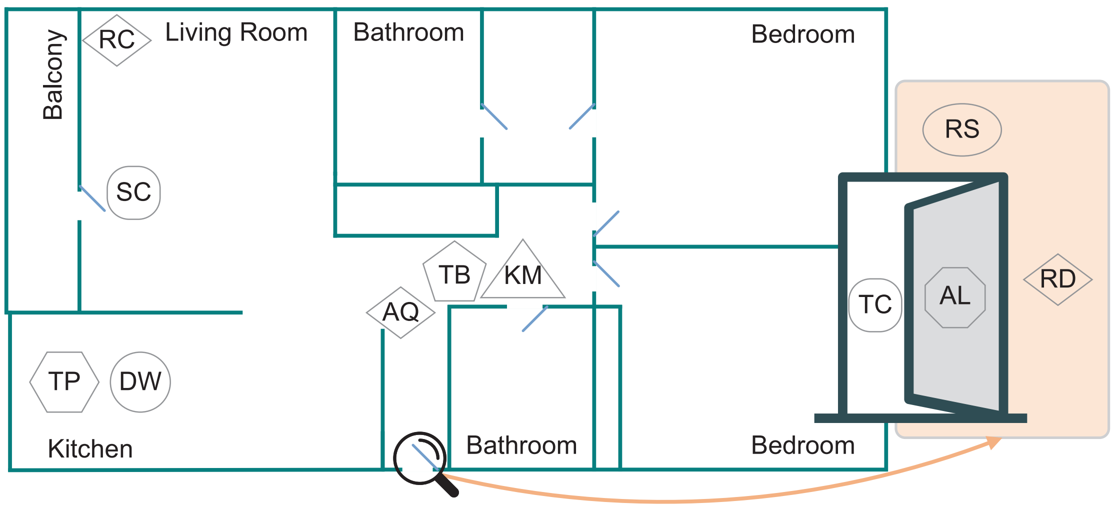


Figure 4.2. Layout of IoT Device Deployment in a Real-World Smart Home Where Each IoT Device Is Represented With Its Abbreviation Name

Figure 4.2 illustrates the layout of our smart home experimental environment in a two-bedroom apartment. Given the list of IoT devices in Table 4.1 and the deployment of devices in Figure 4.2, we use repeated and controlled experiments to discover and generate distinct signatures of user activities with the sequences of IoT device events.

Table 4.2. User Activities and Their Signatures of IoT Device Event Sequences

No.	User Activity	Device Event Sequence
1	A person without key entering the home from the front door (day)	$RD_{mot}, RS_{mot}, RD_{ring}, RD_{on}, RD_{off}, AL_{mlk}, TC_{open}, TC_{close}, AL_{mlk}, AQ_{mot}$
2	A person without key entering the home from the front door (night)	$RD_{mot}, RS_{on}, RD_{ring}, RD_{on}, RD_{off}, AL_{mlk}, TC_{open}, TC_{close}, AL_{mlk}, AQ_{mot}, RS_{off}$
3	A person with app access entering the home from the front door (day)	$RD_{mot}, RS_{mot}, AL_{wlk}, TC_{open}, TC_{close}, AL_{alk}, AQ_{mot}$
4	A person with app access entering the home from the front door (night)	$RD_{mot}, RS_{on}, AL_{wlk}, TC_{open}, TC_{close}, AL_{alk}, AQ_{mot}, RS_{off}$
5	A person with key entering the home from the front door (day)	$RD_{mot}, RS_{mot}, AL_{mlk}, TC_{open}, TC_{close}, AL_{alk}, AQ_{mot}$
6	A person with key entering the home from the front door (night)	$RD_{mot}, RS_{on}, AL_{mlk}, TC_{open}, TC_{close}, AL_{alk}, AQ_{mot}, RS_{off}$
7	A person ring the doorbell and leave (day)	$RD_{mot}, RS_{mot}, RD_{ring}$
8	A person ring the doorbell and leave (night)	$RD_{mot}, RS_{on}, RD_{ring}, RS_{off}$
9	A person checking the front door of the home (day)	RD_{mot}, RS_{mot}
10	A person checking the front door of the home (night)	$RD_{mot}, RS_{on}, RS_{off}$
11	A person with key leaving the home from the front door (door locked) (day)	$AQ_{mot}, AL_{mlk}, TC_{open}, TC_{close}, RD_{mot}, RS_{mot}, AL_{mlk}$
12	A person with key leaving the home from the front door (door locked) (night)	$AQ_{mot}, AL_{mlk}, TC_{open}, TC_{close}, RD_{mot}, RS_{on}, AL_{mlk}, RS_{off}$
13	A person with key leaving the home from the front door (door not locked) (day)	$AQ_{mot}, AL_{mlk}, TC_{open}, TC_{close}, RD_{mot}, RS_{mot}$
14	A person with key leaving the home from the front door (door not locked) (night)	$AQ_{mot}, AL_{mlk}, TC_{open}, TC_{close}, RD_{mot}, RS_{on}, RS_{off}$
15	A person appearing in the hallway of the home	KM_{mot}, TB_{on}
16	A person leaving the hallway of the home	KM_{mot}, TB_{off}
17	A person checking the living room's camera streaming	RC_{on}, RC_{off}
18	A person entering the balcony from living room or entering the living from balcony (contact sensor alarm off)	RC_{mot}
19	An person entering the home from the balcony (contact sensor alarm on)	SC_{open}, RC_{mot}
20	An person leaving the home to enter the balcony ((contact sensor alarm on)	RC_{mot}, SC_{close}
21	A person checking water leakage	$DW_{wtr}, TP_{off}, DW_{nwtr}, TP_{on}$

Table 4.2 summarizes the list of 21 user activities in our smart home testbed which are common and essential to home safety and security. For each user activity,

we repeatedly trigger it 20 times, while simultaneously collecting TCP/IP packets using the programmable home router. To only include IoT device events that are actually triggered by the user activity, we run all of the experiments at this stage in a controlled environment and ensure no other interfering IoT device events or user activities happen at the same time. To avoid the mutual inference of user activities, we separate the experiments of two consecutive user activities for at least 10 minutes.

We first extract all device events from the IoT network traffic during the experiment period where user activity U is triggered. For each IoT device event e, we compare its timestamp, i.e., the timestamp of the first packet captured by the smart home router which is matched to e, with the manually-recorded timestamp of the user activity U. Given the existence of clock synchronizations, varying end-to-end network latencies, and automatic events, e.g., August smart lock automatically closes the door if the door remains open for 30 seconds after an `unlock` event, we consider the device event e is actually generated by the user activity U if and only if $|e.t - U.t| \leq \Omega$. In our experiments, we choose Ω as 60 seconds as the values from 60 seconds to 5 minutes achieve similar results in mapping the relevant device event e to U.

Sorting all IoT device events associated with each user activity U based on their timestamps leads to an ordered sequence of IoT device events $(e_1, e_2, \ldots, e_n)$. If two or more sequences of IoT device events happen for the same user activity during the 20 experiments, we select the sequence with the most occurrences. Such disparity is not observed in our experiments, as the same user activity always triggers the same sequences of IoT device events. We refer to the sequence of IoT device events, $(e_1, e_2, \ldots, e_n)$, as the *signature* of the user activity U, as well as the user activity itself.

Table 4.2 summarizes the signatures of all 21 user activities with the corresponding

IoT device event sequences generated by following the above process. These signatures confirm the feasibility of inferring user activities related to physical safety and security in smart homes from the device events extracted from the IoT network traffic.

4.3 User Activity Inference

Inferring interleaving user activities from device event sequence includes two steps: first, unveil the non-overlapped occurrences of each user activity individually; second, orchestrate a global view of what user activities happened at what time for all user activities. In this section, we first define the problem of k-approximate signature matching to unveil the occurrences of an individual user activity in the device event sequence, followed by our proposed k-approximate signature matching algorithm. Lastly, we present an activity inference algorithm to provide the global view of user activities when multiple user activities happen in parallel.

4.3.1 User Activity Inference Problem

Given a sequence of IoT device events $\mathbb{S} = (s_1, s_2, \ldots, s_m)$ and a set of user activity signatures $\mathbb{U} = \{U_1, U_2, \ldots, U_r\}$ where U_i is the signature of the user activity i, we define the **user activity inference problem** as inferring all user activities that generate events in $\mathbb{S}$.

The unpredictability of user activities and the missing and out-of-order device events have created substantial challenges for solving the user activity inference problem. In this study, we present the first attempt to give a heuristic solution of this problem via finding approximate matches of the user activities' signatures from

$\mathbb{S}$. Specifically, we formulate a problem called $k_{\leq}$*approximate signature matching* and develop an optimal solution for user activity inference based on algorithms designed for solving $k_{\leq}$approximate signature matching problem.

4.3.2 $k_{\leq}$Approximate Signature Matching Problem

The *edit distance* (ED) between a user activity signature U and a sequence of device events $\mathbb{S}$ is defined as the minimum cost of changing $\mathbb{S}$ into U where 1) only the operation of deletion from $\mathbb{S}$ and U is allowed, 2) deletion of an event from U has cost 1, and 3) deletion of an event from $\mathbb{S}$ has cost 0. This definition of ED is different from the classic definition in literature in that the substitution is not a valid operation here because a device event could be missing or out-of-order but should not change into another event. In addition, the costs of deleting different events from U are different due to the varying importance of these events to U. For ease of presentation, we set the cost of deleting any event from U uniformly as 1.

Given a sequence of device events $\mathbb{S} = (s_1, s_2, \ldots, s_m)$, a user activity signature $U = (e_1, e_2, \ldots, e_n)$, and an integer $k \geq 0$, we define a $k_{=}$**single-approximate signature match** as a minimal-length subsequence of $\mathbb{S}$ with the start index i and end index j and $ED((s_i, \ldots, s_j), U) = k$, where k is the approximation parameter.

The $k_{=}$**approximate signature matching problem** is to reveal the *maximum* number of non-overlapping $k_{=}$single-approximate matches of U in $\mathbb{S}$, which is referred to as $k_{=}$approximate match.

The $k_{\leq}$**approximate signature matching problem** is to find all the approximate matches that:

1) exist in an $i_{=}$approximate match of U in $\mathbb{S}$ where $i \leq k$;

2) if there exists a match M in $i_{=}$approximate match, then there does not exist a match M' in $j_{=}$approximate match where $j > i$ and the start and end indices overlap with those of M.

For example, if $\mathbb{S} = (a, a, b, a, c, a, b, a, a, b, a, b, c)$ and $U = (a, b, a, c)$, a $1_{\leq}$approximate match of U in $\mathbb{S}$ includes two $0_{=}$single-approximate signature matches (s_2, s_3, s_4, s_5) and $(s_9, s_{10}, s_{11}, s_{13})$ and one $1_{=}$single-approximate signature match (s_6, s_7, s_8).

4.3.3 $k_{\leq}$Approximate Signature Matching Algorithm

In this section, we first present Algorithm 4.1 to solve the $k_{=}$approximate match problem which aims at identifying all $k_{=}$single-approximate matches of U in $\mathbb{S}$. In Algorithm 4.1, *start* records the starting index for the chunk of $\mathbb{S}$ that U is matched against. C is a two-dimensional cost matrix of size $(m+1) \times (n+1)$ and C records the edit distance between two sequences $\mathbb{S}_{start,i}$ and $U_{0,j}$. The first row of C is initialized as 0 to n since the cost of matching U to an empty device event sequence equals to cost of deleting events from U. The first column of C is initialized as all 0s because the match of U can start from anywhere in $\mathbb{S}$. The last column of C indicates the edit distance between U and $\mathbb{S}_{start,i}$ where $i = start, \ldots, m-1$.

The main idea of this algorithm is that the cost of matching sequence $U_{0,j}$ in $\mathbb{S}_{start,i}$ is the same as the cost of matching $U_{0,j-1}$ in $\mathbb{S}_{start,i-1}$ if $s_i = e_j$. Otherwise, the cost either equals to the cost of matching $U_{0,j}$ in $\mathbb{S}_{start,i-1}$ due to the cost 0 of deleting s_i, or equals to the cost of matching $U_{0,j-1}$ in $\mathbb{S}_{start,i}$ plus 1 due to the cost 1 of deleting e_j.

If $C_{i,n} = k$, we find a match of U in $\mathbb{S}$ that ends at s_i with the approximate

Algorithm 4.1: $k_{=}$appxMatch($\mathbb{S}, U, k$)

Input: Device event sequence $\mathbb{S} = (s_1, s_2, \ldots, s_m)$, a user activity signature $U = (e_1, e_2, \ldots, e_n)$, approximation parameter k

Output: A list L_k of all $k_{=}$single-approximate match U in $\mathbb{S}$. Each match is represented by an ordered sequence of the indices of device events in $\mathbb{S}$ matched to U

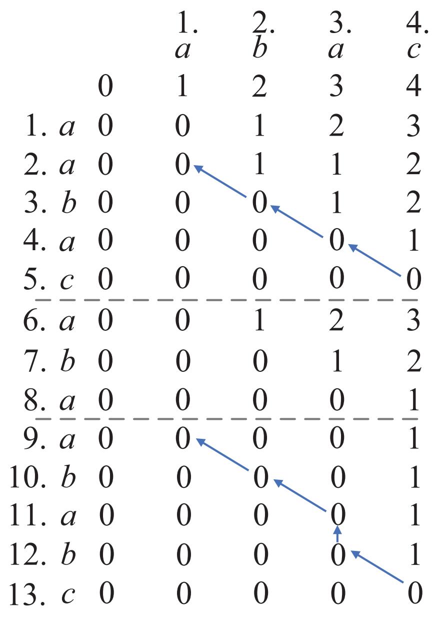
```
1  $L_k \leftarrow \emptyset$; $start \leftarrow 1$;
2  for $i := 0$ to $m$ do
3      $C_{i,0} \leftarrow 0$;
4  for $j := 1$ to $n$ do
5      $C_{0,j} \leftarrow j$;
6  for $i := 1$ to $m$ do
7      for $j := 1$ to $n$ do
8          if $s_i == e_j$ then
9              $C_{i,j} \leftarrow C_{i-1,j-1}$;
10         else if $C_{i-1,j} < C_{i,j-1} + 1$ then
11             $C_{i,j} \leftarrow C_{i-1,j}$;
12         else
13             $C_{i,j} \leftarrow C_{i,j-1} + 1$;
14     if $j == n$ and $C_{i,j} == k$ then
15         $M \leftarrow \emptyset$; $p \leftarrow i$; $q \leftarrow n$;
16         while $q > 0$ do
17             if $C_{p,q} == C_{p-1,q-1}$ and $s_p == e_q$ then
18                 $p \leftarrow p - 1$; $q \leftarrow q - 1$;
19                 $M.insert(p)$;
20             else if $C_{p-1,q} < C_{p,q-1} + 1$ then
21                 $p \leftarrow p - 1$;
22             else
23                 $q \leftarrow q - 1$;
24         $L_k.insert(M)$;
25         for $j := 1$ to $n$ do
26             $C_{i,j} \leftarrow j$; $start \leftarrow i$;
27 output List $L_k$.
```

parameter k. We then output the indices of the current match by backtracking the costs saved in C. Next, we reset the cost matrix by setting the i-th row of C from 0 to n and then continue to find matches starting from $\mathbb{S}_{i+1}$.

Algorithm 4.2: $k_{\leq}$appxMatch$(\mathbb{S}, U, k)$

```
Input: Device event sequence $\mathbb{S}$, a user activity signature $U$, approximation
       parameter $k$
Output: $\mathbb{L} = (L_0, L_1, \ldots, L_k)$ where $L_k$ is the $k_{=}$approximate match of $U$ in $\mathbb{S}$
1  Pre-process $\mathbb{S}$ to remove all the events in $\mathbb{S}$ but not in $U$;
2  $L_0, L_1, \ldots, L_k \leftarrow \emptyset$;
3  Initialize $Q$ with $(0, m-1)$ as its only element;
4  for $i := 0$ to $k$ do
5      $T \leftarrow null$;
6      while $Q$ do
7          $H \leftarrow Q.head()$; $D \leftarrow \mathbb{S}[H.start, H.end]$;
8          $L_k$.append($k_{=}$appxMatch$(D, U, i)$);
9          for each $k_{=}$approximate match in $L_k$ do
10             $B \leftarrow$ the event sequence before $M$ in $D$;
11             $D \leftarrow$ the event sequence after $M$ in $D$;
12             if $B$ is not empty then
13                 start $\leftarrow$ starting index of $B$;
14                 end $\leftarrow$ ending index of $B$;
15                 $T$.append(start, end);
16         if $D$ is not empty then
17             start $\leftarrow$ starting index of $D$;
18             end $\leftarrow$ ending index of $D$;
19             $T$.append(start, end);
20     $Q \leftarrow T$;
21 return $\mathbb{L}$.
```

Figure 4.3 shows a running example of Algorithm 4.1 with $U = (a, b, a, c)$, $\mathbb{S} = (a, a, b, a, c, a, b, a, a, b, a, b, c)$, and $k = 0$, respectively. In this example, s_1 equals to e_1 which are both a, thus $C_{1,1} = 0$. s_1, i.e., a, does not equal to e_2, i.e., b, therefore, $C_{1,2} = 1$ because $C_{1,1} + 1 = 1$ and $C_{0,2} = 2$. Similarly, we determine the other values in the cost matrix C until comparing s_5 with e_4 which are both c, so $C_{5,4} = C_{4,3} = 0$.



$$\begin{array}{l|ccccc}
 & & 1.\ a & 2.\ b & 3.\ a & 4.\ c \\
 & 0 & 1 & 2 & 3 & 4 \\
1.\ a & 0 & 0 & 1 & 2 & 3 \\
2.\ a & 0 & 0 & 1 & 1 & 2 \\
3.\ b & 0 & 0 & 0 & 1 & 2 \\
4.\ a & 0 & 0 & 0 & 0 & 1 \\
5.\ c & 0 & 0 & 0 & 0 & 0 \\
\hdashline
6.\ a & 0 & 0 & 1 & 2 & 3 \\
7.\ b & 0 & 0 & 0 & 1 & 2 \\
8.\ a & 0 & 0 & 0 & 0 & 1 \\
\hdashline
9.\ a & 0 & 0 & 0 & 0 & 1 \\
10.\ b & 0 & 0 & 0 & 0 & 1 \\
11.\ a & 0 & 0 & 0 & 0 & 1 \\
12.\ b & 0 & 0 & 0 & 0 & 1 \\
13.\ c & 0 & 0 & 0 & 0 & 0
\end{array}$$

Figure 4.3. A Running Example of Algorithm 4.1

As we reach the last column of the cost matrix, and the cost value equals to $k = 0$, we find a match of U in $\mathbb{S}$ with the approximation parameter 0. Then we start from $C_{5,4}$ and revisit the cost matrix to output this match as $(5, 4, 3, 2)$.

We continue on searching for matches from s_6 until we find another $0_=$single-approximate signature match ending at s_{13} as $(13, 11, 10, 9)$. As a result, Algorithm 1 outputs $0_=$approximate match $L_0 = ((5, 4, 3, 2), (13, 11, 10, 9))$.

Algorithm 4.2 presents the solution for the $k_\leq$approximate signature matching problem. It outputs $\mathbb{L} = (L_0, L_1, \ldots, L_k)$, where L_i records the $i_=$approximate match of U in $\mathbb{S}$. Each element on L_i stores the indices of the device events in $\mathbb{S}$ that form the $i_=$single-approximate match of U.

The main idea of Algorithm 4.2 is to start from $0_=$approximate match, repeatedly discover the $i_=$approximate match of U in $\mathbb{S}$, and remove all the device events in the $i_=$approximate match from $\mathbb{S}$ until $i = k$. Indices of events in $\mathbb{S}$ are carefully recorded that only portions of device events in $\mathbb{S}$ that do not overlap with the starting and ending indices of matches to U in the current round will be considered in the next round. In this way, only non-overlapping matches are selected for output and approximate matches with smaller k are discovered before approximate matches with bigger k. List Q is designed to save a list of a pair of starting and ending indices for a continuous portion of $\mathbb{S}$ that have not been matched in the approximate match with smaller k. The output of Algorithm 4.2 with $k = 3$ on the running example in Figure. 4.3 is $L_0 = ((2, 5), (9, 13))$, $L_1 = ((6, 8))$, $L_2 = \emptyset$, and $L_3 = ((1, 1))$.

Theorem 4.1. The time complexity of $k_=$appxMatch algorithm is $O(mn)$ and the time complexity of $k_\leq$appxMatch algorithm is $O(kmn)$ where m is the number of events in $\mathbb{S}$ and n is the number of events in U.

Proof Sketch. This can be derived from the execution of the algorithms. □

Theorem 4.2. The $k_=$appxMatch algorithm outputs the maximum number of non-overlapping $k_=$single-approximate signature matches of U in $\mathbb{S}$.

Proof Sketch. It has been proved in literature [71] that given a sequence of overlapped time intervals, the greedy algorithm that chooses an interval with the earliest finish time and excludes other overlapping intervals, outputs the maximum number of non-overlapping intervals. The $k_=$appxMatch algorithm indeed outputs the match with the earliest finish time and only considers non-overlapping matches. □

4.3.4 Inferring User Activities via Approximate Matching

With the $k_\leq$appxMatch algorithm, we can discover approximate matches of each user activity independently. However, it is possible that two matches of different user activities share overlapped device events which cause collisions and ambiguity.

Therefore, we further develop a trimming step to remove the collisions in the device event sequences that are matched to different user activities.

```
Algorithm 4.3: actInfer($\mathbb{S}, \mathbb{U}, k$)
  Input: A device event sequence $\mathbb{S}$, an ordered sequence of signatures of all user
         activities by their lengths $\mathbb{U} = (U_1, U_2, \dots, U_r), k$
  Output: $L_{U_1}, L_{U_2}, \dots, L_{U_r}$ where collisions in approximate matches have been
          removed
1 for $i := 0$ to $r$ do
2     $L_{U_i} \leftarrow k_{\leq}$appxMatch($\mathbb{S}, U_i, k$);
3 for $i := 0$ to $k$ do
4     for $j := 1$ to $r$ do
5         for $M \in L^i_{U_j}$ do
6             if $\exists M' \in L^{i'}_{U_{j'}}$ ($i' < i$) and $M \cap M' \neq \emptyset$ then
7                 remove $M$ from $L^i_{U_j}$;
8             else if $\exists M' \in L^i_{U_{j'}}$ ($j \neq j'$) and $M \subseteq M'$ then
9                 remove $M$ from $L^i_{U_j}$;
10 output $L_{U_1}, L_{U_2}, \dots, L_{U_r}$.
```

The trimming heuristics in the last step of inferring user activities prefer a match with a smaller approximate parameter k or a longer sequence of IoT device events in its signature. This trimming step is designed based on the following empirical observations from our real-world experiments:

1. The chance of missing device events in the signatures of user activities during the matching is marginal. Thus we always prefer matches with smaller values of k.
2. If the signature of one user activity is a subset of another one and both of them are matched independently, the user activity with the superset signature is preferred if both activities are inferred at the same time.
3. If two user activities from different users have overlapping device events, and

these two activities happen at the same time, then it is possible that the overlapped events are shared by both of them. For example, if one user is entering the home from the hallway while another user is heading towards the kitchen through the hallway, these two user activities will only generate one light `on` event since the light will be turned on only once. Thus this device event is shared by these two user activities and can be mapped to both of them during the matching.

Algorithm 4.3 presents the pseudocode of the final algorithm of user activity inference and outputs the set of inferred user activities from the IoT device event streams with the simple yet effective trimming heuristic step.

4.4 Performance Evaluations

In this section, we first describe the experiment setup of a real-world smart home environment for evaluating our proposed user activity inference algorithms. Subsequently, we systematically evaluate the performance of our algorithms in inferring a diverse of user activities from IoT network traffic collected from the smart home environments.

4.4.1 Experiment Setup of Smart Home Environments

To evaluate our proposed algorithms of inferring user activities from IoT network traffic, we have designed and set up a real-world smart home environment, as illustrated in Figure 4.2, where we deployed a number of heterogeneous IoT devices. In this smart home environment, each user activity enumerated in Table 4.2 will trigger at least one IoT device event, thus leading us to observe and collect IoT network traffic via the programmable home router.

In our experiment, we use the Linksys WRT1900AC router running the OpenWrt operating system to collect TCP/IP packets for all the outgoing and incoming network traffic between IoT devices and remote hosts as well as internal traffic in the LAN traffic between IoT devices. For each IoT device event, we adopt the method in Chapter 3 for generating its signature consisting of an ordered sequence of IP packets with inter-packet time intervals. Based on these signatures, we run the signature matching and event extraction algorithms in Chapter 3 on the collected traffic to detect all of the IoT device events that happened in the smart home.

For each of the 21 user activities in Table 4.2, we first learn and build its signature via capturing and studying the underlying IoT network traffic while intentionally repeating the activity with time logs, which serve as the labeled ground truth. Extracting IoT device events from the network traffic and correlating them with the labeled and repeated user activities allow us to recognize and generate the signatures for all 21 user activities. To evaluate the performance of our proposed user activity inference algorithms, we also repeatedly run thousands of user activities at different times and days with time logs over a two-month evaluation period. In addition, our smart home environment continuously collects IoT network traffic, even during the time that we are not actively running the experiments of triggering the 21 user activities. The labeled user activities and collected IoT network traffic provide the valuable dataset for evaluating the correctness and accuracy of our algorithms for inferring user activities.

4.4.2 Evaluation Metrics

We use the same evaluation metrics metrics, i.e., true positives ($\mathcal{TP}$), false positive ($\mathcal{FP}$), false negative ($\mathcal{FN}$), true negative ($\mathcal{TN}$), precision ($\mathcal{P}$), recall ($\mathcal{R}$), and accuracy ($\mathcal{A}$), as introduced in Section 3.5.2. In addition, We also use the F1 score ($\mathcal{F}_1$) matric, which is the harmonic mean of precision and recall and can be calculated as $2 \times \frac{\mathcal{P}\times\mathcal{R}}{\mathcal{P}+\mathcal{R}}$.

Specifically, for a given *inferred* user activity α', if a matching *real* user activity α is found at the same time from the ground truth, we consider this activity as a true positive. However, if there is no matching user activity from the ground truth, we consider it as a false positive. For a given *real* user activity α, if the user activity inference algorithms report a different *inferred* user activity, e.g., β', or simply fail to report an *inferred* user activity, we consider it as a false negative. When the user inference algorithms do *not* report an *inferred* user activity α' for any *real* user activity other than α in the ground truth, it is a true negative. In our experiments, we always observe perfect true negative results, thus we leave the true negative measures out of the experiment results.

4.4.3 Experimental Results

Using performance evaluation metrics introduced in Section 4.4.2, we present the performance of our proposed user activity inference system in three phases. In the first phase, we present the experimental results of running the $0_{\leq}$approximate matching algorithm, i.e., requiring the *exact* matching of device event sequences for all user activities. In the second phase, we present the result of running the $1_{\leq}$approximate matching algorithm, allowing at most one missing device event due to packet delays and losses or device malfunctions. In the third and final phase, we add the optimization rules to the $1_{\leq}$approximate matching algorithm for substantially improving the performance of user activity inference.

4.4.3.1 Phase I: $0_{\leq}$approximate matching

Our experiments of running $0_{\leq}$approximate matching algorithm for inferring thousands of user activities in the real-world smart home environment have shown that the overall value of accuracy, precision, and recall are 0.96, 0.99, and 0.97, respectively, as illustrated in columns 6 to 8 of the last row in Table 4.3. We can observe that most of the user activities can be detected correctly. However, the user activity inference algorithms occasionally fail to detect user activities $\#2, \#4, \#6, \#8, \#10, \#11, \#12,$ and $\#14$, leading to a small number of false negatives, as shown on the fifth column of Table 4.3.

Our in-depth investigation has discovered that nearly all of these false negatives are caused by the "missing" Ring Doorbell's motion detection event from the device event sequences of these user activities. The underlying root cause of these "missing" events is the multiple-second long delay of reporting these motion events by the battery-powered Ring Doorbell [19], leading to the *out-of-order* device events for these user activities. In other words, these motion events are simply delayed, actually not missing. However, as the $0_{\leq}$approximate matching algorithm requires the exact matching of device event sequences for all user activities, these cases with out-of-order device events are reported as false negatives.

In addition to these false negatives, there are also 16 cases of false positive for user activity #18. Our follow-up analysis reveals that the real user activities indeed have

Table 4.3. Experimental Results of Our Proposed User Activity Inference Algorithms in the Real-World Smart Home Environment

User	#	Phase I						Phase II						Phase III					
Act.		$\mathcal{TP}$	$\mathcal{FP}$	$\mathcal{FN}$	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{TP}$	$\mathcal{FP}$	$\mathcal{FN}$	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{TP}$	$\mathcal{FP}$	$\mathcal{FN}$	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{R}$
1	112	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00
2	115	105	0	10	0.91	1.00	0.91	115	0	0	1.00	1.00	1.00	115	0	0	1.00	1.00	1.00
3	112	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00
4	115	106	0	9	0.92	1.00	0.92	115	0	0	1.00	1.00	1.00	115	0	0	1.00	1.00	1.00
5	113	113	0	0	1.00	1.00	1.00	113	0	0	1.00	1.00	1.00	113	0	0	1.00	1.00	1.00
6	115	97	0	18	0.84	1.00	0.84	115	0	0	1.00	1.00	1.00	115	0	0	1.00	1.00	1.00
7	112	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00
8	112	110	0	2	0.98	1.00	0.98	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00
9	121	121	0	0	1.00	1.00	1.00	121	78	0	0.61	0.61	1.00	121	0	0	1.00	1.00	1.00
10	112	105	0	7	0.94	1.00	0.94	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00
11	350	347	0	3	0.99	1.00	0.99	350	0	0	1.00	1.00	1.00	350	0	0	1.00	1.00	1.00
12	294	269	0	25	0.91	1.00	0.91	294	0	0	1.00	1.00	1.00	294	0	0	1.00	1.00	1.00
13	116	116	0	0	1.00	1.00	1.00	116	0	0	1.00	1.00	1.00	116	0	0	1.00	1.00	1.00
14	113	98	0	15	0.87	1.00	0.87	113	0	0	1.00	1.00	1.00	113	0	0	1.00	1.00	1.00
15	138	138	0	0	1.00	1.00	1.00	138	33	0	0.81	0.81	1.00	138	0	0	1.00	1.00	1.00
16	132	132	0	0	1.00	1.00	1.00	132	33	0	0.81	0.81	1.00	132	0	0	1.00	1.00	1.00
17	112	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00
18	224	224	16	0	0.93	0.93	1.00	224	16	0	0.93	0.93	1.00	224	16	0	0.93	0.93	1.00
19	112	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00	112	0	0	1.00	1.00	1.00
20	113	113	0	0	1.00	1.00	1.00	113	0	0	1.00	1.00	1.00	113	0	0	1.00	1.00	1.00
21	116	116	0	0	1.00	1.00	1.00	116	0	0	1.00	1.00	1.00	116	0	0	1.00	1.00	1.00
all	2959	2870	16	89	0.96	0.99	0.97	2959	158	0	0.95	0.95	1.00	2959	16	0	0.99	0.99	1.00

happened as inferred, but are not recorded in our controlled experiments. This observation suggests that our user activity inference algorithms could potentially monitor real-time IoT network traffic and alert homeowners on unexpected or anomalous user activities for home safety and other applications.

4.4.3.2 Phase II: $1_{\leq}$approximate matching

Running the $0_{\leq}$approximate matching algorithm for inferring thousands of user activities achieves high accuracy, but returns a non-trivial of false negatives. Therefore, in phase II we explore the increase of the value of k in the matching algorithm for accommodating the missing and out-of-order device events when searching and matching the signatures of user activities. To understand and quantify the performance tradeoff, specifically false positives and false negatives with varying k, we run the $k_{\leq}$approximate matching with the value of k increasing from 0 to 5. As illustrated in Figure 4.4, the increase of k leads to lower false negatives but higher false positives. In addition, increasing k from 1 to $2, 3, 4, 5$ achieves marginal improvements in false negatives, but incurs significant penalties in false positives. Therefore, we choose $1_{\leq}$approximate matching algorithms for the remaining experiment evaluations.

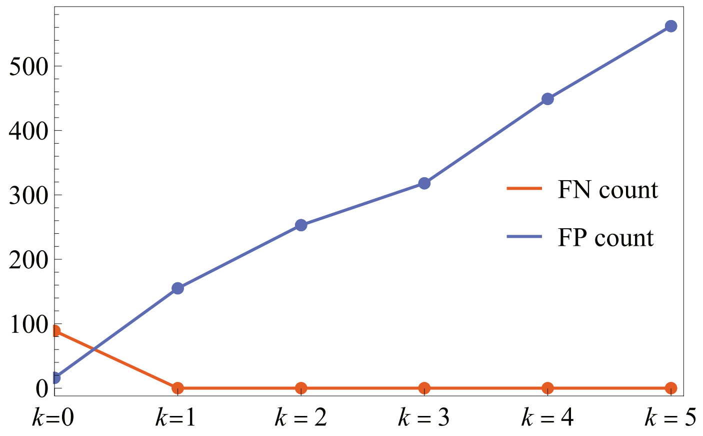


Figure 4.4. Tradeoff Analysis of False Positive vs. False Negatives With Varying k

By setting k as 1, the approximate matching algorithm allows at most one missing or out-of-order device event when matching the signatures of user activities. Such flexibility effectively addresses the challenge of the long delay of the Ring Doorbell's motion detection event, and significantly improves the performance of our user activity inference system in the smart home environment. As shown in the column 11 of Table 4.3, the false negative measures of user activities #2, #4, #6, #8, #10, #11, #12, and #14 drop to 0.

However, in column 10 of Table 4.3, we also observe the increase of false positive measures from 0 to 78, 33, 33 for user activities #9, #15, and #16, respectively. Our follow-up analysis discovers that the signatures of all of these three user activities

(#9, #15, and #16) have a length of 2 device events. The short length of 2 device events in the signatures explains the high false positives, since $1_{\leq}$approximate matching algorithm will report the matching success of one of these activities whenever finding one single device event in their signatures. Given the above observations, we continue to explore optimization rules in the approximate matching algorithm to reduce high false positives while maintaining low false negatives during the matching process.

4.4.3.3 Phase III

$1_{\leq}$*approximate matching with optimization.* Inspired by the findings of high false positives for the short device event sequences, we add a simple yet effective optimization rule for the approximate matching algorithm. Specifically, when the length of the device event sequence for a user activity is equal to or less than a certain threshold, referred to as θ, we only run the $0_{\leq}$approximate matching algorithm to infer such activities. The intuition of such optimization rule lies in the observation that when the length of the device event sequence in the user activity signature is short, the probability of finding false positives is very high. Based on our empirical results, we set θ as 3 in our experiments.

As shown in columns 16 and 17 in Table 4.3, the false positives of user activities #9, #15, and #16 drop to 0 thanks to the simple yet effective optimization rule. The only remaining false positives correspond to user activity #18, which are caused by real but uncontrolled user activities that were not recorded in our experiments. Figure 4.5 compares the overall accuracy, precision, recall, and F1 scores of running our proposed user activity inference algorithms over three phases: $0_{\leq}$approximate matching, $1_{\leq}$approximate matching, $1_{\leq}$approximate matching with optimization, respectively.

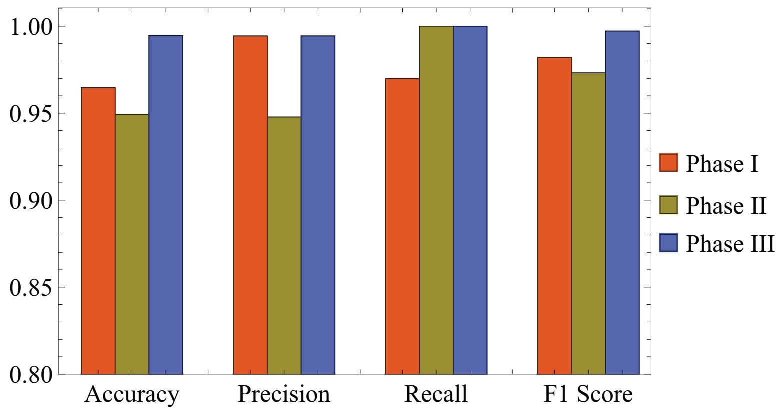


Figure 4.5. Overall Accuracy, Precision, Recall, and F1 Scores of Phase I, Phase II, and Phase III, Respectively

Clearly, $1_{\leq}$approximate matching with the simple and intuitive optimization rule achieves the best performance.

In summary, our two-month long experiments have shown that our proposed user activity inference system is able to effectively and accurately detect and infer user activities from IoT network traffic in smart homes. By applying the approximation matching algorithm with the simple optimization rule, we achieve the overall values of accuracy, precision, and recall as 0.99, 0.99, and 1.00, respectively.

4.5 Related Work

Given the prevalent threats and attacks targeting IoT devices [5, 9, 57, 75, 90, 91, 114, 127, 159], detecting various user activities in smart homes has become a crucial task for IoT security. A few recent studies have explored the events and statuses of IoT devices for recognizing user activities and behaviors [1, 47, 107]. For example, the researchers of [107] deploy numerous sensors in a three-bedroom apartment and develop a hidden Markov model (HMM) based system to recognize and track user activities. Peek-a-Boo [1] is able to launch privacy attacks in smart homes by passively sniffing the encrypted network traffic over the air and utilizing machine learning approaches to recognize six different user activities. Similarly, the study in [47] proposes HAWatcher, a semantics-aware anomaly detection system for appified smart homes, for discovering the correlation according to semantic information in smart homes, and exploits these correlations for modeling a smart home's normal behaviors.

Different from these prior work, IoTMosaic first detects IoT device events from smart home network traffic and then recognizes and profiles user activities to map them to their triggered IoT device event sequences. Subsequently, IoTMosaic develops an effective approximate matching algorithm for inferring user activities in smart homes, which could provide homeowners critical insights on what is happening in their homes instantly and automatically.

4.6 Conclusions

This study proposes IoTMosaic for inferring user activities from IoT network traffic. Based on the IoT device events detected from smart home network traffic, IoTMosaic generates the signatures of diverse user activities consisting of IoT device event sequences. Given the observation of missing and out-of-order device events due to device malfunctions and varying network latencies, we design the approximate matching algorithms to capture the exact or approximate matches of user activities' signatures in the device event sequence. In addition, we devise a heuristic trimming strategy to resolve the conflicts in multiple matches of user activities due to overlapping device events in their signatures. Our two-month experimental results with thousands of user activities in a real-world smart home show that our proposed algorithms can infer different user activities with accuracy, precision, and recall of 0.99, 0.99, and 1.00, respectively.

Part III

Securing Smart Home Networks

Chapter 5

IOTARGOS: MULTI-LAYER MACHINE-LEARNING BASED INTRUSION DETECTION FOR SMART HOME IOT DEVICES

5.1 Background

In this section, we first describe the rising deployment, applications, and services of IoT devices in millions of smart homes, and then discuss the existing vulnerabilities of heterogeneous and weakly-protected IoT devices and the challenges to secure them. Finally, we shed light on multi-layer behavioral fingerprints left by real-world IoT attacks and threats.

5.1.1 Internet-of-Things at Smart Homes

The recent years have witnessed the explosive deployment of Internet-connected IoT devices in smart homes. A smart home today can connect a wide range of IoT devices such as IP surveillance cameras, smart thermostats and smoke detectors, voice assistants, and air quality monitors via different communication protocols for home automation, home security and safety, energy efficiency, and healthcare.

Smart home IoT devices connect to the Internet either *directly* via running TCP/IP protocol stacks on themselves or *indirectly* via relying on a smart home platform such as Samsung's SmartThings [119], Google's Nest [51], and Amazon's Alexa [6]. In smart homes, the Internet-capable IoT devices such as smart TVs, IP surveillance cameras, Amazon Echo, and Google Home typically use wired cables or WiFi to connect to home routers, while many embedded IoT devices such as Philips Hue smart bulb, Samsung multipurpose sensor, and August smart lock use low-power wireless protocols such as Zigbee, BLE, and Z-Wave to connect to the Internet via a smart gateway, hub, or bridge. For ease of presentation, we refer the Internet-capable IoT devices as *ic-IoT devices*, and the gateway-supported embedded IoT devices as *em-IoT devices*.

5.1.2 Security Challenges of Heterogeneous IoTs

As connected IoT devices in smart homes continue to grow in size and complexity, the security issue has become one of the top challenges in IoT research community. Securing IoT systems in smart home is a daunting task due to the heterogeneity of IoT systems, the prevalence of vulnerabilities in IoT devices and applications, as well as the broad attack vector across the entire IoT protocol stack. For example, the recent Mirai botnet [9, 70], formed by hundreds of thousands of IoT devices, launched an aggregated 600 Gbps DDoS attack towards Brian Krebs's security blog. Thousands of home IP cameras, as part of Mirai botnet, are remotely exploited by attackers via universal plug and lay (UPnP) enabled home routers which allow IoT devices behind network address translation (NAT) protection to automatically bind a service port for communicating with remote networked systems [127]. Another innovative attack [90, 114] discovers and leverages the software implementation bugs in the Zigbee light link (ZLL) protocol [159], to potentially control all the lights in a city via spreading IoT worms from a single infected bulb, i.e., patient zero, to all compatible IoT lights using their built-in Zigbee wireless connectivity and physical adjacency. Similarly, misbehaving and malicious smart home applications could explore the design flaws of smart home programming platforms such as Samsung SmartThings to gain over-privileged access and control of IoT systems and to launch event spoofing attacks e.g., triggering fake fire alarms [42, 156].

5.1.3 Multi-Layer Behavioral Fingerprint of IoT Attacks

Many cyber attacks towards IoT devices leave traffic and behavioral fingerprints at different TCP/IP layers and IoT protocol stacks. For example, the Mirai botnet employs three stages including infiltration, infection, and operation for scanning, controlling, and exploiting vulnerable IoT devices. These steps trigger Internet data traffic between the attacking devices and the UPnP-enabled home router as well as wireless packets between the router and wireless IP cameras or other IoT devices. Similarly, if an attacker with unauthorized access to an August smart lock account via a Web interface or a smart phone app remotely opens and closes the smart lock via August cloud services, the events will leave IP, WiFi, and Bluetooth data traffic between the cloud servers and the home router, the router and the August connect bridge, the bridge and the August smart lock, respectively. Therefore, the broad attack vector over the entire IoT protocol stack calls for a multi-layer security monitoring and analysis platform.

5.2 IoTArgos System Overview and Design

In this section, we first present the system overview and architecture of IoTArgos. Subsequently, we discuss each key component of IoTArgos for monitoring and measuring IoT data communications via a multi-layer approach.

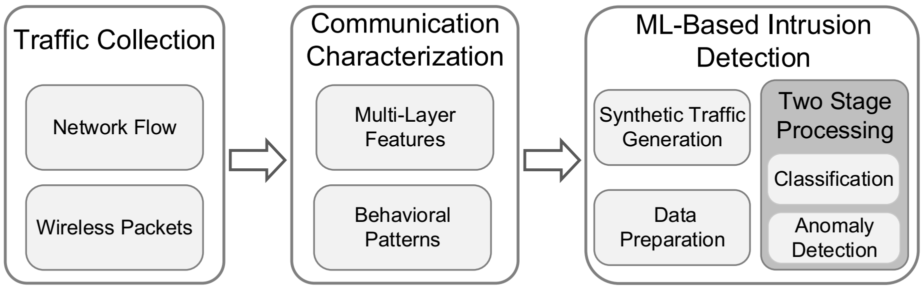


Figure 5.1. The Overall System Architecture of IoTArgos.

5.2.1 IoTArgos System Overview

IoTArgos is a multi-layer smart home security monitoring system for monitoring and analyzing data communications of IoT systems in smart homes and detecting and mitigating intrusions and anomalous activities. The design and implementation of IoTArgos are router-centered. Our intuition of developing IoTArgos on the home router originates from the network architecture of smart homes. Specifically, consumer-grade home routers serve as the residential gateway to route and forward data packets between internal IoT devices in smart homes and external cloud servers of IoT vendors or other remote servers on the Internet. The physical connection from IoT devices to the router is established either directly through wired cables or WiFi, or indirectly via a hub or bridge. In recent years, a number of research studies have explored the computation, storage, and bandwidth resources on commodity home routers to characterize end-to-end performance and troubleshoot performance anomalies and mis-configurations in home networks [2, 29].

Figure 5.1 illustrates the overall system architecture of IoTArgos, which consists of four key components: data collection, IoT communication characterization, ML-based intrusion detection, and real-time mitigation and defense. The primary objective of the *data collection* component is to configure and setup data collection instruments on programmable home routers, while the *IoT communication characterization* component is devoted to characterizing and profiling communication patterns of IoT devices. The *ML-based intrusion detection* component first explores supervised classification algorithms to classify known attacks of IoT data communications and then relies on unsupervised anomaly detection algorithms to detect unknown suspicious activities or zero-day attacks.

5.2.2 Key System Components

5.2.2.1 Multi-Layer IoT Data Collection

A key strength of our proposed router-centered security monitoring system lies in its flexibility of collecting all data communications at a centralized location. Such simple yet effective deployment and configuration is crucial for millions of regular home users to adopt the system, as existing application-driven or device-specific solutions require non-trivial skills and efforts for home users to manage and secure IoT devices with diverse operating systems and interfaces in smart homes. As the residential gateway of broadband home networks, many programmable home routers including bare-bone Raspberry Pi models have the computational resources and open-source packages to capture and store raw IP data packets and aggregated network traffic flows as well as the Ethernet and WiFi frames. Many battery-operated IoT devices in smart homes such as smart locks only communicate with low energy wireless protocols such as Zigbee and Bluetooth, which have been proven to be insecure by design. Therefore, in order to collect the entire data frames from heterogeneous IoT devices adopting different link layer protocols, we setup Texas Instrument CC2531 and CC2540 USB dongles for the capture of Zigbee and Bluetooth frames respectively. In this study, IoTArgos collects both network flow records and wireless packets of Zigbee, Bluetooth and WiFi protocols at programmable home routers.

5.2.2.2 Characterizing IoT Data Communications

The collected multi-layer data by programmable home routers allow us to systematically characterize data communication behaviors of all IoT devices in smart homes, e.g., when, how, and why IoT devices in smart homes communicate with cloud servers, other remote networked systems, mobile applications that control the devices, home routers, and local IoT hubs. Specifically, IoTArgos profiles data communications of IoT devices with a broad range of basic *raw* communication and traffic features such as the IP address and domain name of remote end hosts, inter-packet arrival time, packet size, flow duration, source port, destination port, protocol, link layer protocol, as well as aggregated features such as the number and dynamics of remote hosts, and the dominant applications. These traffic features not only characterize IoT data communications, but also play a crucial role in the ML-based intrusion detection component.

5.2.2.3 ML-based Intrusion Detection

In general, we classify cyber attacks towards IoT systems as known and unknown attacks. The signature and patterns of known attacks are often public knowledge, while unknown attacks, e.g., new or zero-day attacks are typically not discovered or reported yet. Similar to traditional firewalls, detecting known intrusions and attacks often requires signature-based techniques or supervised machine-learning based methods which are often unable to uncover new attacks. In order to capture both known and unknown attacks, IoTArgos adopts a two-stage approach for ML-based intrusion detection: i) supervised classification stage, and ii) unsupervised anomaly detection stage. The first stage applies one supervised machine learning algorithm on the collected multi-layer data for classifying IoT attacks or normal IoT data communications, while the second stage applies one unsupervised anomaly detection algorithm to uncover anomalous behaviors that are not detectable by the supervised classification stage due to the unavailability of attack signatures or training data-sets.

5.3 Multi-Layer Feature Characterization of Smart Home IoT Devices

IoT devices in smart homes often exhibit various functions and diverse computation, storage, and communication capabilities. For example, smart TVs and Amazon Echo are often powered by electric power and have wired or wireless connections to home routers for Internet connections, while battery-operated motion and water leak sensors rely on low energy wireless communication protocols such as Zigbee, Z-Wave, or Bluetooth to connect with the specific bridges (also called hubs or gateways) which support both TCP/IP protocols and IoT wireless protocols and standards.

To characterize data communication for *all* IoT devices in smart homes with a *centralized* solution, IoTArgos explores a wide range of multi-layer features from TCP/IP-based network flow records extracted by *softflowd* and *nfcapd* packages installed on programmable home routers and wireless packets captured by wireless sniffers installed on the routers. The majority of IoT applications and services in smart homes have employed encryption in data communications, thus IoTArgos focuses on behavioral features such as the size, durations, protocols, and remote end systems of data communications.

Mining and correlating multi-layer features of IoT data communications allow us to gain a deep understanding on behavioral patterns of these IoT devices in smart homes, which is a critical first step for securing these devices and detecting anomalies. For example, Figure 5.2 shows the numbers of Zigbee wireless packets exchanged between a Samsung outlet and Samsung SmartThings hub as well as IP data packets between the hub and Samsung servers hosted on Amazon cloud over a 24-hour time window.

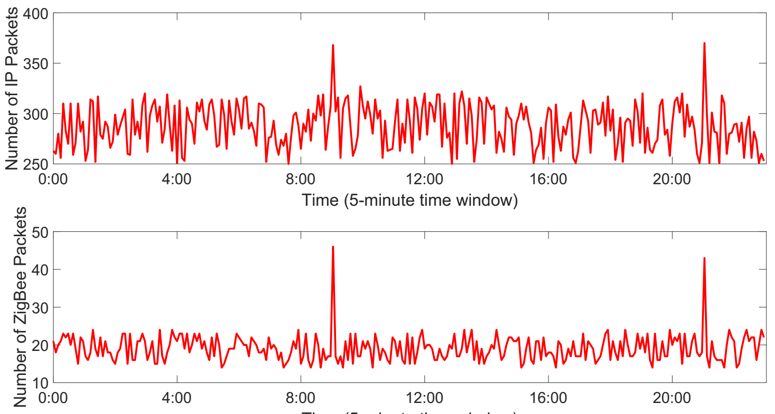


Figure 5.2. The Numbers of ZigBee Wireless Packets Exchanged Between a Samsung Outlet and Samsung SmartThings Hub As Well as IP Packets Between the Hub and Samsung Cloud Servers

The objective of building behavioral patterns of IoT devices is to understand *what*, *when*, *how*, *if*, and *why* the devices communicate with other systems including their *local* bridges, hubs, or gateways in the same home and *remote* cloud servers. Towards this end, IoTArgos extracts the *basic* traffic features of network flows and wireless packets originating from or destined to the IoT devices including source MAC address, destination MAC address, source IP address, destination IP address, source port, destination port, protocol, flow volumes in packets and bytes, flow duration, and then derives the *advanced* features such as inter-packet arrival time, average packet size, domain and autonomous system number (ASN) of cloud server.

The multi-layer data communication features allow IoTArgos to establish a behavioral profile to summarize the communication patterns of IoT devices over time. Such behavioral profiles and communication patterns, if captured in a controlled smart home laboratory environment, can serve as a baseline of normal IoT communications. To identify multi-layer features of IoT data communications, we rely on permanent physical layer addresses of IoT devices and their hubs as well as the temporal sequences to associate network flow records with wireless packets for generating *augmented* network flow records which summarize individual conversations or events of IoT applications and services, e.g., remotely opening an August smart lock via data communications among the lock owner's smart phone, August cloud server, the smart home router, and the August connect bridge.

In summary, IoTArgos characterizes and profiles IoT data communications with a broad range of features from multi-layer IoT protocol stacks. Mining these multi-layer features also leads us to discover distinct communication patterns of IoT devices in smart homes. These features and patterns provide critical insights and valuable inputs for exploring ML algorithms to detect intrusion activities and anomalous behaviors towards or from IoT devices.

5.4 Machine-Learning Based Intrusion Detection

A number of recent studies and surveys have reported that the prevalent and exploitable vulnerabilities of IoT systems in smart homes have enabled the attackers to compromise and control thousands of IoT systems for launching large scale DDoS attacks or listening to the private conversations in hacked smart voice assistants. Hence, detecting intrusion activities and anomalous behaviors of IoT systems in smart homes is an important design goal of IoTArgos.

In this section, we present our two-stage ML-based intrusion detection design in IoTArgos, which relies on the supervised classification algorithms in the first stage to classify known attacks, and explores the unsupervised anomaly detection in the second stage to detect unknown or zero-day attacks towards IoT devices in smart homes.

5.4.1 First Stage: Supervised Classification

The intuition of our proposed two-stage intrusion detection strategy is driven by the diversity and complexity of the existing and potential attacks and attacks towards heterogeneous IoT devices in smart homes. Traditional signature-based detection methods or emerging ML-based classifications approaches are very efficient for classifying attacks whose signatures are available or whose prior instances are captured and labelled. However, such methods often have challenges in recognizing zero-day attacks that are created by attackers via exploiting newly discovered or exposed vulnerabilities from one or more types of IoT devices.

Therefore, as a first step, IoTArgos explores supervised classification algorithms to detect and filter a subset of attacks via training a suite of classification algorithms and selecting the *desired* algorithm that balances the intrusion detection performance and system consumption such as CPU and memory cost on resource-constrained home routers. Specifically, we choose five well-known and computationally lightweight classification algorithms including k-nearest neighbors (k-NN), logistic regression (LR), naïve bayes (NB), RF, and support vector machine (SVM).

5.4.2 Second Stage: Unsupervised Anomaly Detection

As several prior studies on Internet intrusion and anomaly detection have pointed out, cyber attackers often employ new attack techniques thanks to the newly discovered zero-day vulnerabilities in the compromised systems, they often exhibit similar behavioral patterns, e.g., such as port scanning, penetration testing, and brute-force password attempts as existing attacks. Given the likely new attacks that are misclassified as "normal" in the first stage, IoTArgos develops the second-stage with anomaly detection algorithms for identifying new attacks from the remaining "normal" data communications. In this stage, we also select and evaluate computationally lightweight anomaly detection algorithms such as clustering-based local outlier factor (CBLOF), fast angle-based outlier detection (FastABOD), feature bagging (FB), isolation forest (IForest), local outlier factor (LOF), and PCA.

Therefore, the goal of the second stage is to uncover anomaly behaviors that are not detectable by the supervised intrusion classification technique due to the unavailability of attack signatures or training data-sets or the emerging new vulnerabilities or weakness of IoT systems to be discovered by attackers. In other words, each data communication with various multi-layer features of IoT systems in smart homes will be initially inspected by the the first-stage classification algorithms in IoTArgos. If the classifier reports normal, the record will go through the second stage anomaly detection model.

5.5 Performance Evaluations

We have implemented the IOTArgos system and deployed the system across 22 real-world smart home networks. In this section, we present results of our extensive performance evaluations along with our key observations. We first describe our experiment setup, synthetic IoT data communication generations, and evaluation metrics. Subsequently, we systematically evaluate the performance of our proposed two-stage intrusion detection technique.

5.5.1 Experiment Setup and Synthetic IoT Traffic Generation

To detect and classify attacks and intrusion activities towards IoT devices in smart homes, we built a simple yet effective ML-based intrusion detection component into the IoTArgos system. To demonstrate the performance, benefit, and feasibility of our approach, the IoTArgos system not only collects the normal multi-layer data communications of IoT devices in distributed smart homes in real-time, but also captures the simulated attack traffic towards selected IoT devices. To comprehensively simulate the existing cyber attacks against smart home IoT systems, we referred to recent studies [5, 9, 102, 114] on security attacks and threats towards IoT devices and simulate and replay a wide range of cyber attacks, as summarized in Table 5.1, across multiple layers of IoT communication protocol stack.

For each type of attacks, we wrote dedicated scripts and followed the state-of-the-art penetration test procedure to replay the attacks with varying instances and intensity towards selected IoT devices. In order to replay link layer attacks against devices running the Zigbee and Bluetooth protocols [32, 90, 114, 115, 159], we rely on the widely-used HackRF One transceiver, a software defined radio (SDR) device capable of transferring and receiving radio signals ranging from 1MHz to 6GHz, and run the open-source radio frequency monitoring and injecting tools such as GNU Radio and Scapy-radio for customizing the frequency or the type of the link layer frames.

For collecting the *normal* IoT data communication, we built a smart home sandbox in a laboratory environment that deploys the IoTArgos system on a OpenWrt-supported Linksys WRT1900ACS home router equipped with 1.6GHz dual-core processor and 512MB memory. In the smart home laboratory, all IoT devices are configured behind the NAT-enabled router. In addition, we disabled UPnP from the security setting on the home router to prevent outside attackers in close proximity from directly targeting all IoT devices in the smart home sandbox. The smart home sandbox, consisting of a variety of IoT devices and the programmable home router, has been continuously running for over six months. We consider the data collected by the router in the sandbox during these six months as the *normal* IoT data communications, and combine with the simulated attacks to generate large scale *synthetic* IoT communication data-sets. We eventually built a labelled data-set consisting of over 6 million normal network flow records and over 300 thousand attack flows for our evaluation experiment. Once acquiring the synthetic IoT data communications consisting of normal IoT data communications and simulated attacks, we evaluate the performance and cost of a suite of ML models for determining the most optimal model to balance the intrusion detection quality and the cost of computational and memory resources.

Table 5.1. The List of Simulated Attacks Towards IoT Devices in Smart Homes

Category	Attack Strategy	Description
Scanning Attacks	host scanning [9, 148] port scanning [9, 102] nexpose scanning [5] nessus scanning [5]	identifying IoT devices and scanning for vulnerabilities
Flooding Attacks	HTTP flooding [9]	application-layer
	DNS flooding [9]	application-layer
	GRE-IP flooding [9]	application-layer
	UDP flooding [9]	volumetric
	UDP plain flooding [9]	volumetric
	SYN flooding [9]	TCP state exhaustion
	ACK flooding [9]	TCP state exhaustion
	IP AH flooding [9]	IPSec
	IP ESP flooding [9]	IPSec
Brute Force Attacks	SSH brute force [102] Telnet brute force [9]	brute forcing user credentials
Data Link Layer Attacks	ACK spoofing [90, 114]	fake scan response
	blind attack [90]	malicious identify request
	DoS attack [90]	junk traffic
	force re-pairing [115]	manipulate pairing request

5.5.2 Evaluation Metrics

We use the similar true positives ($\mathcal{TP}$), false positive ($\mathcal{FP}$), false negative ($\mathcal{FN}$), true negative ($\mathcal{TN}$), precision ($\mathcal{P}$), recall ($\mathcal{R}$), accuracy ($\mathcal{A}$), and F1 score ($\mathcal{F}_1$) metrics as introduced in Section 3.5.2 and Section 4.4.2. In our synthetic IoT data traffic, we label the simulated attacks as *positive* and normal traffic flows and wireless packets as *negative*. During the performance evaluation of ML-based algorithms, the correctly detected attacks are denoted by $\mathcal{TP}$, while the attacks detected as normal scenarios are considered as $\mathcal{FN}$. Similarly, $\mathcal{TN}$ refers to the cases when normal IoT data communications are recognized as normal, while $\mathcal{FP}$ represents the cases when normal IoT communications are incorrectly detected as attacks.

5.5.3 Evaluation of the First Stage

In the first stage for IoT intrusion detection, we select five widely-used classification algorithms, i.e., k-NN, LR, NB, RF, and SVM, and apply k-fold cross validation with k set as 10 to evaluate and compare their performance. Table 5.2 illustrates accuracy, precision, recall, and $\mathcal{F}_1$ score of the five classification algorithm. As shown in Table 5.2, all classification algorithms achieve over 0.90 across all metrics except the precision and $\mathcal{F}_1$ score for NB classification.

Table 5.2. The Performance Metrics of Detecting IoT Intrusions With Five Classification Algorithms

Model	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{F}_1$
k-NN	0.9833	0.9878	0.9649	0.9762
LR	0.9573	0.9718	0.9076	0.9386
NB	0.9195	0.9413	0.8292	0.8817
RF	0.9858	0.9893	0.9703	0.9797
SVM	0.9707	0.9806	0.9365	0.9580

While all these classification algorithms are very effective, the effectiveness of these algorithms depends on the complete knowledge of the attacks. In reality, there are always the possibility of *new* and *unknown* attacks. In order to study the impact of *new* and *unknown* attacks, we conducted a study in which we *intentionally* treat certain types of attacks as "normal" data in the training phase. As a result, the performance of all classification algorithms decreases significantly due to the existence of *new attacks*. Table 5.3 shows the decreased performance metrics of detecting IoT intrusions with five classification algorithms with 25% types of IoT attacks with the lowest instances in the synthetic data-set as new or unknown during model training process. For example, the accuracy, precision, recall and $\mathcal{F}_1$ score of RF classification drops 0.1088, 0.1004, 0.0950, 0.0977, respectively. This simple study demonstrates that the classification algorithm alone, albeit efficient in detecting IoT attacks with high quality labelled data-sets, is not sufficient enough to detect the rising new attacks and threats towards IoT systems in smart homes.

Table 5.3. The Performance Metrics of Detecting IoT Intrusions With Five Classification Algorithms With 25% Types of IoT Attacks As New or Unknown During Model Training Process

Model	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{F}_1$
k-NN	0.8945	0.8958	0.8781	0.8869
LR	0.8388	0.8620	0.8388	0.8502
NB	0.7802	0.8668	0.7636	0.8119
RF	0.8770	0.8889	0.8753	0.8820
SVM	0.8766	0.8886	0.8642	0.8762

5.5.4 The Benefits of the Second Stage

Anomaly detection algorithms have been extensively studied to identify unknown attacks or anomalies towards networked systems. Thus, we introduce anomaly detection algorithms in IoTArgos as the second stage for discovering those “new” types of IoT attacks that are incorrectly detected as normal data communications by the classification algorithm in the first stage. Considering the rare nature of the new or zero-day IoT attacks, we only consider the types of IoT attacks with the lowest instances from our labelled data-set as the “new” types of attacks in the experiments.

The intuition and rationale of introducing anomaly detection algorithms lie in our observations that all outlier IoT data communications are likely anomalous and suspicious activities towards IoT systems since these data communications likely deviate from common and normal IoT data communication patterns. In this study, we select six algorithms i.e., CBLOF, FastABOD, FB, IForest, LOF, and PCA which are widely used for anomaly or outlier detections, and run each of these algorithms against all the “normal” IoT data communications to generate distinct clusters of various sizes for grouping similar patterns or to assign anomaly scores for each communication flow.

Table 5.4. Performance Metrics of Combining RF Classification and the Second Stage for Detecting Known and New IoT Attacks

Model	$\mathcal{A}$	$\mathcal{P}$	$\mathcal{R}$	$\mathcal{F}_1$
CBLOF	0.9757	0.9798	0.9661	0.9729
FastABOD	0.9241	0.9365	0.9212	0.9288
FB	0.9467	0.9521	0.9353	0.9436
IForest	0.9876	0.9897	0.9750	0.9823
LOF	0.9444	0.9500	0.9342	0.9420
PCA	0.9818	0.9876	0.9763	0.9819

Table 5.4 illustrates the performance of combining RF classification with the second-stage anomaly detection algorithms for detecting new IoT intrusions that are misclassified as *normal* communications in the first stage. To systematically evaluate the benefit of adding anomaly detection algorithms in the second stage, we run all the 30 combinations of the (classification, anomaly detection) pair on our labelled data-set of synthetic IoT data communications. Table 5.5 shows the substantial improvement on AUC of combining two-stages over the first classification stage alone, where each entry shows the result of the corresponding (classification, anomaly detection) pair.

Table 5.5. The AUC Improvement by Combining Two-Stages Over the First Classification Stage Alone

Model	None	CBLOF	FastABOD	FB	IForest	LOF	PCA
k-NN	0.87	0.97(+10.6%)	0.92(+4.9%)	0.96(+9.3%)	0.97(+10.7%)	0.95(+8.8%)	0.97(+10.8%)
LR	0.89	0.96(+8.9%)	0.91(+2.9%)	0.94(+6.0%)	0.97(+9.2%)	0.96(+8.2%)	0.96(+7.7%)
NB	0.82	0.92(+11.1%)	0.90(+9.7%)	0.92(+11.4%)	0.96(+16.6%)	0.92(+12.0%)	0.92(+12.0%)
RF	0.87	0.97(+11.5%)	0.92(+6.0%)	0.95(+8.8%)	0.97(+11.3%)	0.96(+10.0%)	0.97(+11.4%)
SVM	0.88	0.94(+7.3%)	0.91(+3.8%)	0.94(+7.0%)	0.96(+9.7%)	0.94(+7.5%)	0.95(+9.0%)

We observe from Table 5.5 that running anomaly detection after classification improves accuracy in all cases, and the improvement is significant except for a few combinations. For example, applying PCA anomaly detection after running the RF classification improves 11.4% on the AUC metric, i.e., from an AUC of 0.87 in the first stage to the final AUC of 0.97. To have a better view of the improvement, we use Figure 5.3 to illustrate both the average improvement and maximum improvement of AUC via combining anomaly detection stage and classification for all 30 combinations. All of the models have significant AUC increment on average and the isolation forest model managed to improve the raw naïve bayes model's AUC over 16%.

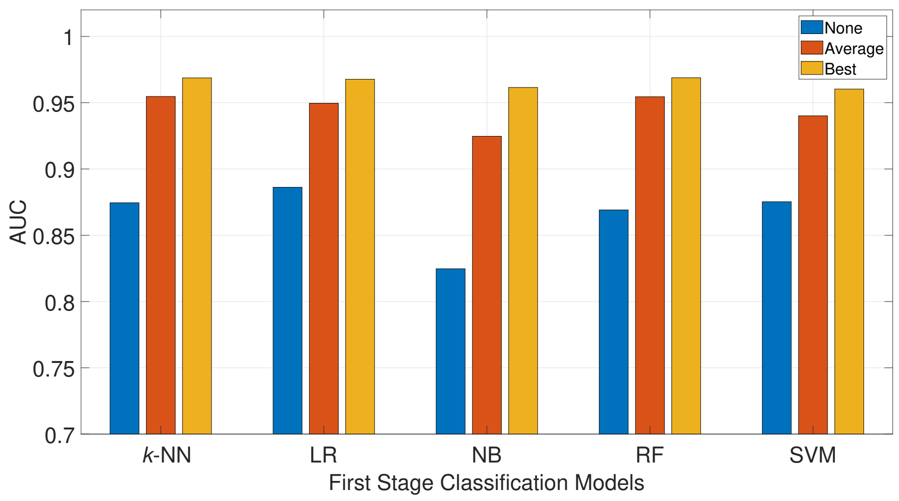


Figure 5.3. The Average and Best AUC Improvement by Combining Two-Stages Over the First Classification Stage Alone

In summary, these experimental results demonstrate that our proposed two-stage intrusion detection algorithm is very effective and has significant advantages over classification alone. In addition, we have run a series of experiments with varying ratios, e.g., 95%/5%, 90%/10%, 85%/15%, 80%/20%, 75%/25%, and 70%/30% of normal and attack traffic flows. All the experiments show similar performances of our proposed two-stage ML-based method in detecting simulated attacks towards smart home IoT devices. Given the overall performance and robustness of our proposed method, it is safe to expect significant improvement of the two-stage algorithm when new/better classification or anomaly detection algorithms are designed and employed by our two-stage algorithm.

5.6 Related Work

Smart home IoT security has been of recent interest to many researchers due to the wide deployment of IoT devices in home networks. Research effort has been devoted to empirically study the current status of IoT deployment, to discover and raise awareness of the potential vulnerabilities and their growing security implications, to design more secure IoT application frameworks, to analyze the firmware and software security, and to understand the behavior of IoT components and further detect anomaly [5, 15, 17, 39, 41, 44, 48, 55, 60, 65–69, 76, 82, 83, 92, 100, 108, 114, 122, 124, 129, 144, 157].

State of Home IoT Deployment and Security: A recent SoK paper [5] categorizes the home IoT security research into device, mobile application, cloud endpoint, and communication, describes the attacks and mitigation, and proposes recommendations to stakeholders for each category. In addition, the paper evaluates the security properties of 45 home IoT devices and their applications. A recent paper [72] carries out a large-scale empirical analysis of IoT devices in real-world homes, covering 83 million devices in 16 million homes, and presents methodologies to identify the types of IoT devices in home networks and provides their regional distributions. The paper also analyzes the status of services and weakness in the IoT devices and the scanning behavior of smart homes. As the first paper investigating the security problems of Smart Home IoT applications, the work in [42] identifies over 50% of the applications on Samsung's SmartThings platform with serious over-privilege problems. The similarities and differences between IoT security research and classic IT security research from hardware, system software, network, and application layers are discussed in [44]. A recent paper [91] provides a comprehensive survey on security issues and defense mechanisms for various IoT applications in smart homes, vehicles, cities and buildings. Human Factors in smart home IoT security are discussed in [36–38, 56, 73, 74, 89, 130, 149].

Attack Techniques: The popularity of home IoT devices has introduced a new spectrum of attacks towards all the components in smart homes. The SOK paper [5] compiles a comprehensive list of attacks towards different types of IoT devices, popular services supported by the devices, weak trust management and weak credentials, mobile application development, and communication channels. With an example attack where worms automatically spread over a large area among physically adjacent lamps in a chain reaction using only the standard Zigbee wireless interface, a new attack paradigm where IoT devices with ad hoc networking capabilities can spread malware to their physically adjacent neighbors bypassing the Internet is investigated in [114]. In addition, vulnerable IoT devices have been recently exploited to form high profile botnets. The detail insight of Mirai botnet is studied in [9, 52, 70], which is a high profile DDoS threat sourced from hundreds of thousands of IoT devices, and how the insecurity of IoT devices contributes to the growth of this largest ever botnet. The latest botnet consisting of IoT devices managed in a peer-to-peer fashion named Hajime is discussed in [57].

Application Security: A recent paper [15] gives a thorough analysis of smart home IoT applications' security and privacy issues. SmartAuth [134] is a framework that identifies required permissions for IoT applications running on platforms like SmartThings and Apple Home. SAINT [14] is a static information flow tracking and analysis tool for evaluating privacy risks in IoT implementation. Their results show that 60% SmartThings market apps include sensitive data flow. Soteria [16] presents a static analysis system for validating whether an IoT app or environment is secure and operates correctly by automatically extracting a state model from a SmartThings IoT app and applying model checking to identify property violations. IoTSan [93] uses model checking to reveal interaction level flaws by identifying events that can lead the system to unsafe states. FlowFence [43] proposed a framework that splits application codes into sensitive and non-sensitive modules and orchestrates the execution through opaque handlers. SIFT [77] is a safety-centric programming platform which leads to more robust and reliable IoT apps.

Behavior Modeling and Intrusion Detection: HoMonit [156] proposes a third-party defender which monitors the smart home side-channel traffic and detects misbehavior in smart apps such as privileged accesses and event spoofing. Their approach leverages wireless fingerprints to detect mis-behaviors in a resource-constraint IoT environment. The current encryption status of four popular medical devices is investigated in [148] by capturing and analyzing network traffic and retrieving clear-text information, which reveals sensitive medical conditions and behaviors. Applying machine learning techniques for detecting intrusion targeting IoT devices are discussed in [95, 144, 152]. [11, 86, 150] model the device behavior at network, transport and application layers, while [54, 64, 125] model device behavior with linker layer and physical layer traffic.

A recent position paper [139] summarizes the attacks and security functions in device, network, and service layers, and introduces a cross layer framework to connect and bridge the gap between different layers. Another relevant paper [26] demonstrates smart home devices are vulnerable to attacks from malicious mobile apps running on authorized phones and implements and evaluates a HanGuard system where the home router enforces role based access control between mobile apps and IoT devices with the help of a user-space monitoring app running on the mobile phone. Their adoption of router for enforcing security policy is similar to this work. Leveraging smart home applications on activity recognition, health monitoring, and automation for detecting abnormal home and user behavior in the homes is proposed in [23]. In contrast to these research effort, our proposed IoTArgos system focuses on characterizing IoT data communications with multi-layer behavioral features, and detecting intrusion and anomalous activities towards IoT with a two-staged ML-based method.

5.7 Conclusions

The recent high-profile cyber attacks towards vulnerable and insecure IoT devices have highlighted the prevalent security threats towards millions of smart homes and the great risks of data and user privacy. These attacks call for systematic approaches for protecting IoT devices from the broad attack vector which spans the entire IoT protocol stacks due to design flaws of rapidly developed and deployed protocols, weak credential management, and lack of cryptographic functions on resource-constrained IoT devices. As a first step of securing IoT devices in smart homes, this paper designs, develops, and evaluates IoTArgos, a multi-layer security monitoring system on programmable home routers. Based on data captured from hundreds of IoT devices in real-world smart homes, IoTArgos characterizes and models data communication behaviors of heterogeneous IoT devices with a broad range of communication and traffic features. To detect intrusions towards IoT devices, IoTArgos develops a two-stage method to first explore supervised classification algorithms for identifying known attacks based on trained labelled data-sets and then rely on unsupervised anomaly detection algorithms for capturing emerging attacks without prior attack labels or signatures. Our extensive experiments based on synthetic IoT data traffic with normal communications collected from a smart home sandbox and simulated attacks have shown the two-stage method is very effective in detecting a wide range of IoT attacks.

Chapter 6

EXTRACTING SPATIAL INFORMATION OF IOT DEVICE EVENTS FOR SMART HOME SAFETY MONITORING

6.1 Motivation and System Architecture

In this section, we first discuss the motivation of this study and then present the overall architecture of the proposed system.

6.1.1 Motivation

Our motivation comes from real-world attacking scenarios where smart home IoT devices are compromised for executing malicious device events triggered by attackers [45, 78, 127]. However, these attacks could reveal that the controlling devices triggering the events are in abnormal or suspicious places. Thus, extracting the spatial information of each IoT device event is crucial to better understand the behaviors of smart home IoT devices and users as well as to secure smart homes.

In this work, we study the problem of extracting spatial information of IoT device events by first discussing the obstacles and challenges. We propose IoTDuet for determining whether a device event is triggered locally or remotely. IoTDuet makes efforts in extracting spatial information of IoT device events by collecting and analyzing only the home network traffic, while the outputs of IoTDuet are still informative and crucial for understanding IoT devices' behaviors and detecting safety risks in smart homes. We then utilize the extracted spatial information for applications in home safety monitoring.

6.1.2 System Architecture

Figure 6.1 presents the overall architecture of our system which detects whether a device event is triggered locally or remotely. Our system consists of five major components, which are (i) home network traffic collection, (ii) IoT device event extraction, (iii) controlling device's command and data traffic identification, (iv) traffic analysis of device event and controlling device, and (v) home safety monitoring.

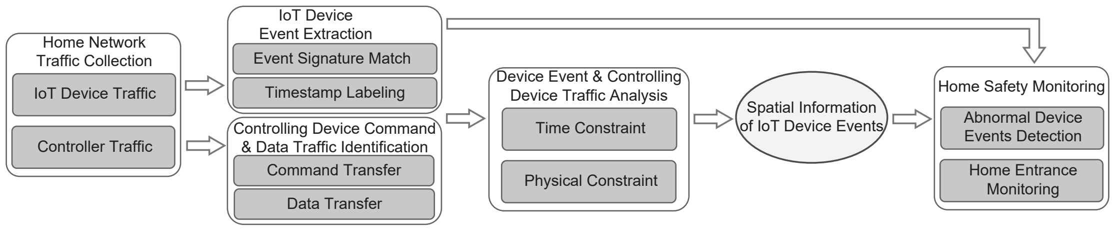

Home Network Traffic Collection
IoT Device Traffic
Controller Traffic
IoT Device Event Extraction
Event Signature Match
Timestamp Labeling
Controlling Device Command & Data Traffic Identification
Command Transfer
Data Transfer
Device Event & Controlling Device Traffic Analysis
Time Constraint
Physical Constraint
Spatial Information of IoT Device Events
Home Safety Monitoring
Abnormal Device Events Detection
Home Entrance Monitoring

Figure 6.1. The Overall Architecture of Our Proposed System

The first component of our system is in charge of collecting all incoming and outgoing network traffic of the smart home at the home router. The second component is implemented by adopting the solution introduced in Chapter 3 for accurately inferring the device event logs from the collected smart home network traffic. IoTDuet corresponds to the third and fourth components in Figure 6.1 where the third component focuses on identifying the command and data transfer traffic of the controlling devices in the home network and building profiles of the domain name information of the cloud servers that the controlling devices send commands and data to. The fourth component combines information captured by the second and third components to determine whether a device event is triggered locally or remotely. The last component leverages the extracted spatial information of IoT device events for critical home safety monitoring applications. We focus on detecting abnormal device events and comprehensively monitor all possible home entrance activities, which are closely related to the cyber and physical security of smart homes.

6.2 Challenges of Device Events' Spatial Information Extraction

As we can observe from the communication model of smart home IoT devices discussed in Chapter 2, IoT devices and controlling devices do not directly communicate with each other for sending and receiving commands, data, and status update information in most cases. In particular, the vendors' cloud servers act as a proxy between IoT devices and controlling devices for forwarding messages. From the smart home users' perspective, capturing the network traffic sent and received by the IoT devices at the home router is practical for most off-the-shelf home routers with little modifications. However, by analyzing the network traffic collected at the home router, we can only observe network packets sending to or receiving from the cloud servers. Thus, no information about the controlling devices' IP addresses can be observed by just analyzing the home network traffic.

We thus turn our attention to exploring the IP addresses of the cloud servers to check whether they can help us determine the location of the controlling devices. We examine the DNS traffic of all IoT devices deployed in our smart home testbed and notice that when IoT devices send DNS queries to get the cloud servers' IP addresses, the DNS resolution policies will pick the host that is close to the IoT devices geographically. The traffic will be routed inside the cloud service provider's AS to the host that is close to the controlling device to minimize network latency. On the other hand, such policy also prevents us from inferring the location of the controlling device by examining the location of the cloud servers the IoT devices communicate with.

On the cloud servers' side, they can learn the IP addresses of both the controlling devices and the IoT devices but such information cannot be directly accessed by smart home users. The lack of network traffic captured at the cloud servers side prevents us from directly knowing the IP addresses of the controlling devices which carry important spatial information.

The inter-packet time intervals of packets sent between the IoT devices and the cloud servers captured at the home router were also considered for inferring the locations of the controlling devices. It can reveal whether the controlling devices are far away from the IoT devices if network latency instead of computation latency at the cloud server dominates the inter-packet time intervals. In fact, we notice that for some devices such as August Lock, sending out commands using a controlling device located on a different continent from where the IoT device is deployed will result in noticeably larger intervals compared to the case where the controlling device is in the same city as the IoT device. However, we can only tell whether the controlling device is significantly far away by analyzing the intervals without knowing the exact geographical location. The variations in network conditions also prevent us from quantifying how far the controlling device is from the IoT device. This strategy also does not apply to all IoT devices because inter-packet intervals of some devices are dominated by the computation latency of the cloud server.

6.3 IoTDuet Design

6.3.1 Home Network Traffic Collection

The first component of IoTDuet is in charge of collecting smart home network traffic at the home router. We first flash the Linux-based OpenWRT operating system to an off-the-shelf home router. Then we collect all raw incoming and outgoing traffic of all devices connected to the home router using software such as tcpdump or tshark. The network traffic of different devices can be grouped by filtering with IP addresses and MAC addresses. Specifically, we identify the network traffic generated by different IoT devices using the hostname field in the header of DHCP response packets, which contain string values that can be mapped to the names of the device vendors, device names, and device models.

6.3.2 Device Event Inference from Home Network Traffic

Existing studies have revealed that different IoT device events always generate unique network traffic traces which can be modeled as device event signatures [1, 109, 135]. After extracting device event signatures via controlled experiments, they can be used for inferring device events by matching them in home network traffic. IoTDuet adopts the solution introduced in Chapter 3 for generating the traffic signature of each IoT device event with inter-packet interval information, and extracting IoT device events from network traffic collected by the home routers using time-sensitive subsequence matching. We label the timestamp of each extracted device event using the timestamp of the first packet that is matched to it.

6.3.3 Identifying Controller's Command and Data Transfer Traces from Home Network Traffic

We start with an example of an unlocking command packed in JSON format sent from an iPhone controller with the Schlage Home mobile app to the cloud server whose domain name is 'api.branch.io' to open the Schlage smart lock. From the key-value pairs in the JSON object illustrated in Figure 6.2, we notice that there are many values that could change in different settings for keys such as 'device_carrier', 'connection_type', 'latest_update_time', and 'app_version'. Thus the payload contents and the packet lengths could vary when sent from different types of controlling devices in different network conditions at different times which makes the packet-level signature hard to be extracted from the controller's command messages, unlike the device event traces of the IoT devices.

```
"device_carrier" : "Verizon",
"connection_type" : "wifi",
"latest_update_time" : 1641964009951,
"app_version" : "4.4.0",
```

Figure 6.2. Examples of the Keys in the JSON Object Corresponding to the Unlocking Command Sent From an iPhone Controller to the Cloud Server With the Domain Name 'api.branch.io' To Unlock the Schlage Smart Lock Where the Keys 'Device_carrier', 'Connection_type', 'Latest_update_time', and 'App_version' Can Change if the Command Is Sent From a Different Controller Device in Different Network Conditions at Different Times

However, we notice that the domain name of the remote server that the controller communicates with is always 'api.branch.io' even if we use different controlling devices such as an Android phone or a tablet. Thus, we use the domain name or part of the domain name of the remote host that command messages and data are sent to as the signature of the controller's command transfer. As the example in Figure 6.3 illustrates, if a controlling device inside the home network initiates Arlo Q camera's stream on event, we would observe the network traffic of both the packet-level signature of Arlo Q camera's stream on event and controlling device's command message which is sent to 'myapi.Arlo.com' at the home router. However, when a controlling device outside the home network initiates the same device event, we can only observe the packet-level signature of Arlo Q camera's stream on device event at the home router. Such differences in the network traffic data collected by the home router can help differentiate whether an IoT device event is triggered by the controlling device connected to the home network or not.

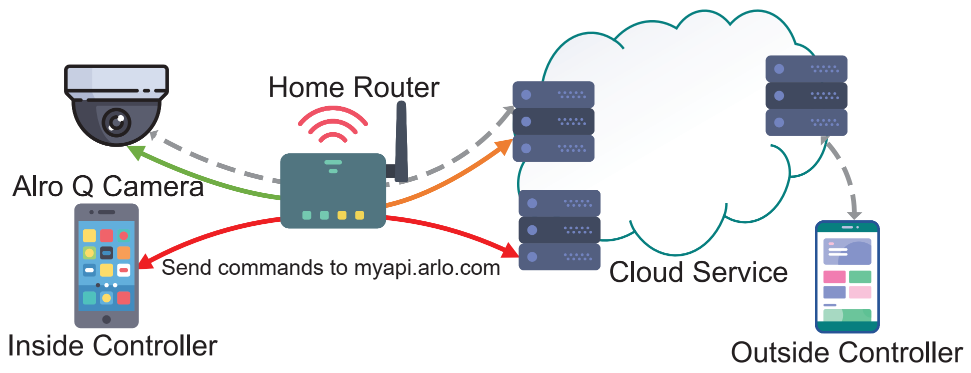


Figure 6.3. An Example of the Network Traffic Collected at the Home Router in the Scenarios of a Controller Inside the Home Network Initiating a Stream on Event and a Controller Outside the Home Network Initiating a Stream on Event of the Arlo Q Camera

To correctly extract the domain name information, we installed a MITM (Man-in-the-Middle) root certificate on the controlling devices and applied the MITM proxy for collecting and decrypting the network traffic generated by the controlling device. We then repeat each device event for at least 5 times at different times of the day for each kind of controlling device such as smartphone, tablet, and browser. After verifying that the domain name is consistent over different runs and different controlling platforms, we set it as the signature of the controlling command.

Some device events such as video streaming have different kinds of network traffic traces on the controlling devices side because a large number of video stream packets instead of simple reply packets are sent to the controlling devices from the cloud. We observe that for some of the camera and doorbell devices in our testbed, video streams could be sent from the cloud server with the same domain name as the cloud server that the IoT device communicates with for uploading the videos. So, we can apply the same strategy used for command messages to extract the domain name information of the controller's data transfer traffic.

After building signatures of the controllers' command and data transfer messages, we can easily look up the packets sent to servers with these domain names in the traffic collected by the home router.

6.3.4 Extracting Spatial Information of IoT Device Event

For each device event e_i, we can efficiently extract the spatial information about whether it is triggered locally or remotely, as described in Algorithm 6.1. We first examine the device event name to check whether it contains keywords in the set H which includes names of the low-energy wireless communication protocols such as Zigbee, Bluetooth, and Z-Wave. Due to the limited wireless communication range of these protocols, when we observe the above keywords, we can claim that such device event is triggered by the controlling device that is close to the home and label the location of e_i as 'inherently local'.

We then check the device event name to verify if the device event can only be

Algorithm 6.1: IoTDuet(e_i, P, H, D)

```
Input: An IoT device event $e_i$, network packets of all devices connected to the
       home network $P = \{p_1, p_2, \ldots, p_r\}$, a set $H$ of device event names that are
       inherently local, a set $D$ of the domain name controlling device
       communicate with when triggers $e_i$
Output: Spatial attribute of $e_i$
1  $e_i.location \leftarrow$ 'remote';
2  if $e_i.name \in H$ then
3      $e_i.location \leftarrow$ 'inherently local';
4  else
5      $start \leftarrow \arg\min_j(p_j.t, p_j.t > e_i.time - \delta)$;
6      $end \leftarrow \arg\max_j(p_j.t, p_j.t < e_i.time + \delta)$;
7      for $j := start$ to $end$ do
8          if $p_j.domainName \in D$ then
9              $e_i.location \leftarrow$ 'local';
10 output $e_i.location$.
```

triggered by a person or an object that is physically close to the device. In our smart home environment, these device events include motion detection events of all camera and doorbell devices, ringing event of all doorbells, and manual locking or unlocking and autolocking events of all lock devices. These device names are also included in the set H and when observing these device events, we label the location of e_i as 'inherently local'.

If $e_i.name \notin H$ and e_i is extracted from the IoT device's network traffic with timestamp $e_i.time$, we further check the network packets P of all devices connected to the home network. When we observe the packet sent to the host of the domain name in set D which contains domain names that the controlling device will communicate with for sending the command messages or data during the time period of $[e_i.time - \delta, e_i.time + \delta]$, we claim that the device event e_i is triggered by a controlling device in the home network and label the spatial attribute of e_i as 'local'. In our environment, we set $\delta = 5s$ to cope with network latency while minimizing interference. Otherwise, if only the device event e_i is extracted from the IoT device's network traffic and there are no packets sent to the server with the domain name in D during $[e_i.time-\delta, e_i.time+\delta]$, then the device event e_i is considered to be triggered by a controlling device in a different network. Most smartphones, tablets, and laptops automatically connect to the home WiFi network when in the communication range unless deliberately disconnected by the user, which is a rare case. So in this condition, we will label the location of e_i as 'remote'.

6.4 Home Safety Monitoring

The spatial information about smart home IoT device events is critical in smart home security. We explore home safety monitoring applications of detecting abnormal device events and home entrance monitoring.

6.4.1 Abnormal Device Event Detection

The spatial information about where a device event is triggered helps us gain a deeper understanding of the device events that happen at a smart home. Thus we can apply this information for detecting if a device event is abnormal or not. We assume that when we observe the network traffic of a controlling device connected to the home network, the device events observed during this time period should be triggered locally. On the contrary, a device event that is triggered remotely during this time should be labeled as anomalies and reported to the smart home users because there is a high possibility that this device event is triggered by attackers who compromised the device remotely.

Algorithm 6.2 describes how we identify the list of abnormal device events L_e in sequence $\mathbb{E}$. If we observe a device event e_i at time $e_i.time$ triggered by a controlling device that is connected to the home network, and by observing the network traffic in the smart home collected at the home router, we can notice that the controlling device is connected to the home network until t'. In this case, we will check the spatial information of all device events $e_{i+1}, e_{i+2} \ldots, e_u$ where $e_u.time \leq t'$ and $e_{u+1}.time > t'$ extracted using IoTDuet as described in Algorithm 6.2. If we observe a device event e_j which is triggered by remotely, we will flag it as 'potentially abnormal' and include it in the output list L_e.

Algorithm 6.2: AbnormalEventsExtract($\mathbb{E}, P$, IoTDuet)

```
Input: Device event sequence $\mathbb{E} = (e_1, e_2, \ldots, e_m)$, network packets of all devices
       connected to the home network $P$
Output: A list $L_e$ of all potential abnormal device events $L_e = \{e_{a_1}, e_{a_2}, \ldots, e_{a_r}\}$
1  $L_e \leftarrow \emptyset$;
2  for $i := 0$ to $m$ do
3      if $IoTDuet(e_i, P, H, D)$ == 'local' then
4          check $P$ for $t'$ of time that the controlling device disconnect the home
             network;
5          for $j := i + 1$ to $m$ do
6              if $e_j.time < t'$ then
7                  if $IoTDuet(e_j, P, H, D)$ == 'remote' then
8                      $L_e.insert(e_j)$;
9              else
10                 $i \leftarrow j + 1$; break;
11 output List $L_e$.
```

6.4.2 Home Entrance Monitoring

The extracted spatial information of smart home device events can help us better profile the entrance activities in a smart home. In this work, we aim to comprehensively monitor all possible home entrance activities as it is critical to the physical safety and privacy of the smart home residents. We first identify all possible entrance points in the smart home. For each entrance point, we enumerate all possible home entrance activities and build up signatures of them which consist of sequences of device events using the techniques proposed in Chapter 4. The key difference is that we embedded the spatial information into the signatures of home entrance activities, which means two occurrences of a device event with the same name but different spatial attributes will be treated differently when building up signatures for the entrance activities, unlike IoTMosaic introduced in Chapter 4.

We also comprehensively consider all kinds of entrance activities at different entrance points which not only include the common ones listed in IoTMosaic, but also activities that rarely happen or are potentially 'abnormal'. Based on whether an entrance activity is common for the normal user, we build up a set N consisting of potentially abnormal entrance activities which are seldom triggered by legitimate users. We also build a set T consisting of time-dependent activities which are considered to be abnormal during the specific time period, i.e., midnight time M.

By applying the $k_{\leq}$appxMatch algorithm on the device events $\mathbb{E}$, we can extract all home entrance activities $\mathbb{A} = (A_1, A_2, \ldots, A_n)$ by ordering them based on the timestamp of the first device event in each sequence that is matched to the signature of a home entrance activity. Each extracted home entrance activity A_i has two attributes, activity name $A_i.name$ and its timestamp $A_i.time$. Utilizing spatial and temporal information of each device event, we can identify a list of abnormal home entrance activities L_a which should be reported to the smart home users using Algorithm 6.3. We first check the name of each entrance activity A_i and add A_i to L_a if $A_i.name \in N$. We then check the timestamp of A_i and if $A_i.name \in T$ and $A_i.time \in M$, we also include A_i in L_a. In our experiment, we set M =[1:00am, 5:00am] when the users in the smart home and the neighborhood are less active.

Algorithm 6.3: EntranceActMon($\mathbb{A}, N, T, M$)

Input: Home entrance activity sequence $\mathbb{A} = (A_1, A_2, \ldots, A_n)$, a set of potentially abnormal entrance activities N, a set of time-dependent abnormal entrance activities T, midnight time period M

Output: A list L_a of all potentially abnormal home entrance activities $L_a = \{e_{g_1}, e_{g_2}, \ldots, e_{g_q}\}$

```
1 $L_a \leftarrow \emptyset$;
2 for $i := 0$ to $n$ do
3     if $A_i.name \in N$ then
4         $L_a.insert(A_i)$;
5     else if $A_i.name \in T$ and $A_i.time \in M$ then
6         $L_a.insert(A_i)$;
7 output List $L_a$.
```

6.5 Experimental Evaluation

In this section, we first present the setup of our experimental environment. Then we discuss our identified hostnames that controllers communicate with for command and data transfer. We then evaluate the performance of IoTDuet in extracting the spatial information of each IoT device event. Subsequently, we evaluate the frameworks of abnormal device event detection and home entrance safety monitoring.

6.5.1 Experimental Setup

To systematically and extensively evaluate IoTDuet, we set up a real-world smart home environment where 17 different types of IoT devices are deployed as illustrated in Table 6.1. All of these IoT devices are ranked as popular based on Smart Home DB [128]. For each of these IoT devices, we also identify the different device events it supports and there are 53 different IoT device events in total as listed in Table 6.1.

We identified 3 entrance points in our smart home testbed which are the window, the front door, and the back door. Figure 6.4 demonstrates the layout of our smart home environment and the deployment of the IoT devices at the 3 entrance points for home entrance monitoring.

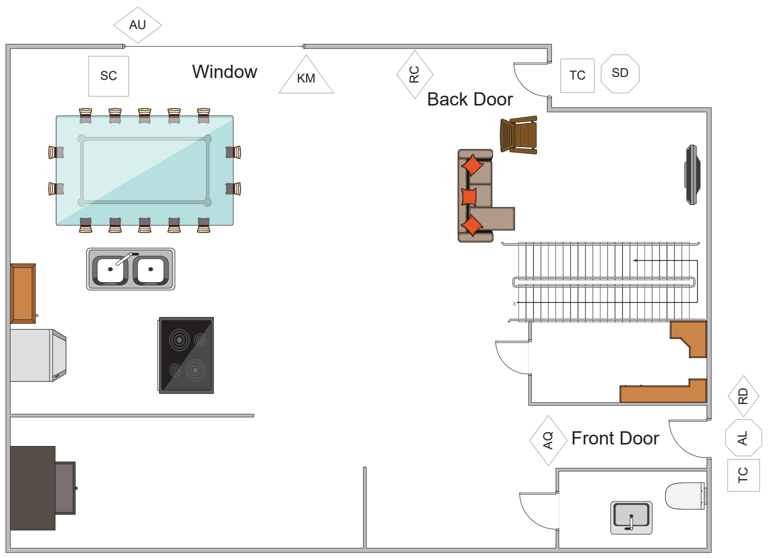


Figure 6.4. The Layout of Our Smart Home Testbed With 3 Entrance Points and the Deployment of IoT Devices at the Entrance Points for Home Entrance Monitoring

Table 6.1. The Domain Names of the Cloud Server Which the Controlling Device of Different Platforms Communicates With for Command and Data Exchange of 17 IoT Devices With 53 Different IoT Device Events

Type	Device Name	Device Event	Cloud Server's Domain Name
bulb	Philips Hue	on or off (PH_{onoff})	api2.amplitude.com
		brightness (PH_{br})	api2.amplitude.com
	Sengled LED	on (SS_{on})	*.cloud.sengled.com
		off (SS_{off})	*.cloud.sengled.com
		brightness (SS_{br})	*.cloud.sengled.com
	TP-Link Bulb	on (TB_{on})	api.tplinkra.com
		off (TB_{off})	api.tplinkra.com
		color (TB_{cl})	api.tplinkra.com
		brightness (TB_{br})	api.tplinkra.com
camera	Amcrest ProHD	stream on (AP_{on})	*.compute.amazonaws.com
	Arlo - Q Indoor	stream on (AQ_{on})	myapi.arlo.com
		stream off (AQ_{off})	myapi.Arlo.com
		streaming (AQ_{str})	arlo*.*.amazonaws.com
		motion detection (AQ_{mot})	N/A
	Arlo Ultra	stream on (AU_{on})	myapi.arlo.com
		stream off (AU_{off})	myapi.arlo.com
		streaming (AU_{str})	arlo*.*.amazonaws.com
		motion detection (AU_{mot})	N/A
	Blink XT2	stream on (BX_{on})	*.immedia-semi.com
		stream off (BX_{off})	*.immedia-semi.com
		streaming (BX_{str})	*.compute.amazonaws.com
		motion detection (BX_{mot})	N/A
	Reolink Camera	stream on (RC_{on})	apis.reolink.com
		stream off (RC_{off})	apis.reolink.com
		streaming (RC_{str})	direct
		motion detection (RC_{mot})	N/A
	Yi Camera	stream on (YC_{on})	gw-us.xiaoyi.com
		stream off (YC_{off})	gw-us.xiaoyi.com
		streaming (YC_{str})	*.aliyun.com
		motion detection (YC_{mot})	N/A
doorbell	August Doorbell	stream on (AD_{on})	api-production.august.com
		stream off (AD_{off})	api-production.august.com
		streaming (AD_{str})	*.compute.amazonaws.com
		ringing (AD_{ring})	N/A
		motion detection (AD_{mot})	N/A
	Ring VideoDoorbell	stream on (RD_{on})	clientapigw.ring.com
		stream off (RD_{off})	clientapigw.ring.com
		streaming (RD_{str})	*.compute.amazonaws.com
		ringing (RD_{ring})	N/A
		motion detection (RD_{mot})	N/A
lock	August Lock Pro	WiFi (un)locking (AL_{wlk})	api-production.august.com
		Bluetooth (un)locking (AL_{blk})	N/A
		autolocking (AL_{alk})	N/A
		manual (un)locking (AL_{mlk})	N/A
	Schlage Deadbolt	WiFi (un)locking (SD_{wlk})	api2.branch.io
		autolocking (SD_{alk})	N/A
		manual (un)lock (SD_{mlk})	N/A
plug	Amazon Plug	on (AP_{on})	api.amazon.com
		off (AP_{off})	api.amazon.com
	Gosund Socket	on or off (GS_{onoff})	device-provisioning.googleapis.com
	TP-Link Plug	on (TP_{on})	api.tplinkra.com
		off (TP_{off})	api.tplinkra.com
	WeMo Plug	on or off (WP_{onoff})	appapis.xwemo.com

6.5.2 Domain Name Extraction of Cloud Servers

For each of the 53 IoT device events, we repeated it for at least 10 times using all possible controlling devices. We marked traffic types and remote cloud servers' host names of the device events that are 'inherently local' as 'N/A' in Table 6.1. The rest of the IoT device events can be triggered by different kinds of controllers such as smartphones and tablets with the mobile companion apps installed. Some of them can also be triggered using a web browser or the PC software provided by the vendors. We tested on all of these controlling devices to extract the domain names of the remote cloud servers that the controlling commands of data are sent to.

To accurately identify domain names, we installed a MITM root certificate on the controlling devices and deploy the MITM proxy to collect and decrypt the network traffic sent from it. For mobile applications employing the certificate pinning technique, we patched them using Frida to bypass the certificate pinning. We investigated the packets whose payloads carry commands that are sent to the cloud servers for triggering the corresponding device events or data received from the cloud servers for device events such as video streaming. Since all the TLS packets are already decrypted using MITM, we identified the packets corresponding to the command or data exchange by matching keywords in the payloads such as 'open', 'off', and 'lock'.

We identified domain names of the remote cloud servers for each of the device events which involve the controlling device and we found that the domain name is consistent in all 10 runs in the experiments across all different controlling platforms. The results are illustrated in Table 6.1.

6.5.3 Performance Evaluation of IoTDuet

With the domain name of the cloud servers that controllers communicate with, we further evaluated the performance of IoTDuet in detecting whether an IoT device is triggered locally or remotely. We collected the data over a 2-week period where device events are triggered by both controllers connected to the smart home network and by controllers in three different remote places including the same city as the smart home tested, a different city in US west coast, and a different city in US east coast respectively. For each of the triggered device event, we recorded its name and labeled the timestamp.

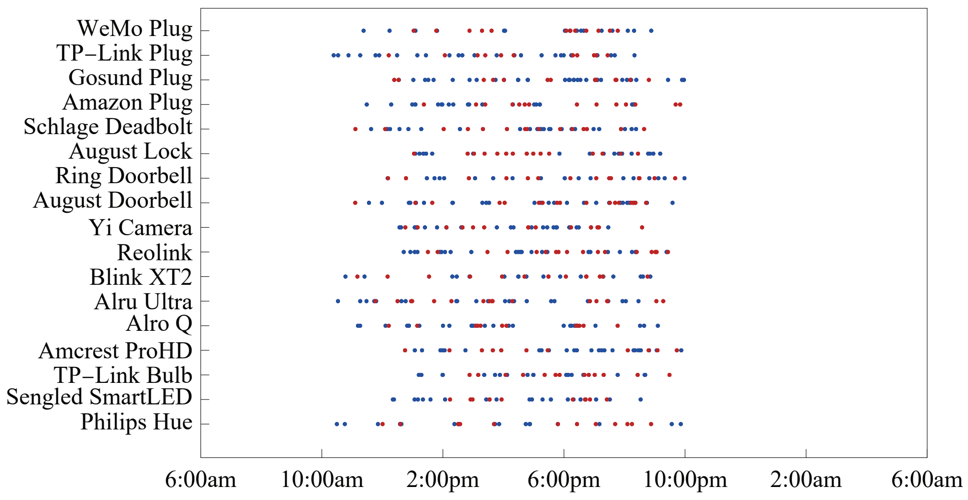


Figure 6.5. Device Events Triggered by the Controller Locally and Remotely Over the 2 Weeks That Mapped to a 24-Hour Window Where the Blue Points Indicate the Device Events Triggered Locally While the Red Points Indicate the Device Events Triggered Remotely

Figure 6.5 illustrates the extracted device events by mapping them to a 24-hour time window. The blue points indicate the device events triggered locally the red points indicate the device events triggered remotely. We notice that IoTDuet can accurately and efficiently determine where a device event is triggered by analyzing the smart home network traffic.

6.5.4 Performance Evaluation of Home Safety Monitoring

The spatial information of device events extracted by IoTDuet enables us to detect potential anomalies that are critical to the safety of smart home users. We systematically evaluated the performance of our proposed Algorithm 6.2 and Algorithm 6.3 in detecting abnormal device events and home entrance activities using the data collected in our real-world smart home testbed.

6.5.4.1 Abnormal Device Event Detection

We detect the abnormal device events using the spatial information of each device event extracted with IoTDuet by setting alarms to the device events that are triggered remotely while the controlling devices are connected locally to the home network. To evaluate the performance of Algorithm 6.2 on extracting abnormal device events, we trigger each device event normally while synthetically injecting device events that are triggered remotely when the controlling device is connected to the home network.

Figure 6.6 illustrates an example of detected abnormal Blink XT2 stream on and off device events. These two device events are triggered remotely and are marked in red in Figure 6.6. We also noticed Phillips Hue on event and TP-Link Plug on event which are triggered locally and the controlling device is connected to the home network during the time that Blink XT2 stream on and off device events are triggered remotely. So, we raise an alarm for these two device events to notify the smart home

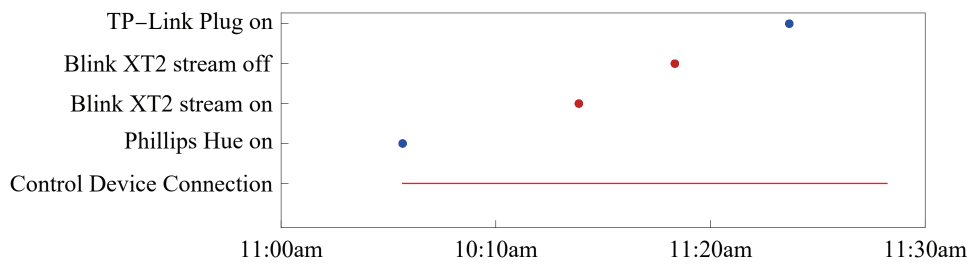


Figure 6.6. An Example of Detected Abnormal Blink XT2 Stream on and off Device Events Which Are Triggered Remotely (Marked in Red) When the Controlling Device Is Connected to the Home Network

user that the Blink XT2 may be compromised for generating abnormal device events. Our evaluation on the data collected at the smart home with synthetic abnormal data shows that our framework can correctly identify abnormal device events with suspicious spatial patterns which are triggered remotely while the controlling device still connects to the home network.

6.5.4.2 Home Entrance Monitoring

In our smart home testbed, we identified 3 critical entrance points including the window, front door, and back door. For each of the 3 entrance points, we further enumerated all possible entrance activities in order to comprehensively monitor the home entrance safety. Table 6.2 lists all 90 home entrance activities that could happen at our smart home. Different from IoTMosaic introduced in Chapter 4, which considers only the common user activities, we study all kinds of home entrance activities that could happen including intrusive activities and abnormal activities. For example, we not only consider the opening window from the inside activity of normal users, but also include the activity that a person breaks into the house through the window from the outside, as listed in Table 6.2. In addition, we also take the spatial information of

Table 6.2. Home Entrance Activities With Labels and Their Corresponding Device Event Sequence Signatures

Entrance Point	Entrance Activity	Event Sequence	Label
Window	A person or an object outside the house passes by the window	AU_{mot}	time-dependent
	A person opens the window from the inside	KM_{mot}, SC_{open}	potentially normal
	A person closes the window from the inside	KM_{mot}, SC_{close}	potentially normal
	A person opens the window from the outside	AU_{mot}, SC_{open}	potentially abnormal
	A person closes the window from the outside	AU_{mot}, SC_{close}	potentially abnormal
	A person breaks into the house through the closed window from outside and leaves the window open	$AU_{mot}, SC_{open}, KM_{mot}$	potentially abnormal
	7 breaking into the house activities are omitted due to page limit	...	...
Back Door	A person enters the house through back door by manually unlocking a locked door and locks it manually	$SD_{mlk}, TC2_{open}, RC_{mot}, TC2_{close}, SD_{mlk}$	potentially normal
	29 back door entrance activities are omitted due to page limit	...	...
Front Door	A person or an object outside the house passes by the front door	RD_{mot}	time-dependent
	A person outside the house rings the doorbell and then leaves	RD_{mot}, RD_{ring}	time-dependent
	A person outside the house rings the doorbell and gets inside with door unlocked manually and leaves door opened	$RD_{mot}, RD_{ring}, AL_{mlk}, TC1_{open}, AQ_{mot}$	potentially abnormal
	14 entrance activities by ringing doorbell first with different ways of opening and closing front door omitted due to page limit	...	...
	A person enters the house through front door by manually unlocking a locked door and locks the door manually	$RD_{mot}, AL_{mlk}, TC1_{open}, AQ_{mot}, TC1_{close}, AL_{mlk}$	potentially normal
	14 entrance activities of a person with lock control with different ways of opening and closing front door omitted due to page limit	...	...
	A person exits the house through front door by unlocking a locked door manually and leaves the door unlocked	$AQ_{mot}, AL_{mlk}, TC1_{open}, RD_{mot}$	potentially abnormal
	14 exiting activities of a person with lock control with different ways of opening and closing the front door omitted due to page limit	...	...

each device event into consideration when identifying the entrance activities and treat the device events with same name but different spatial information as unique when building up the signatures.

As shown in Table 6.2, we categorize each entrance activity as ‘potentially normal’, ‘potentially abnormal’, and ’time-dependent’ based on whether the entrance activity is considered to be different from smart home users’ normal behaviors. If an entrance activity is performed by legitimate users most of the time, we consider it as ‘potentially normal’. If an entrance activity is intrusive or not common, we consider it as ‘potentially abnormal’. If an entrance activity is suspicious during a particular time period, i.e., midnight, we label it as ‘time-dependent’ because we need the timestamp information of the activity to help us make the decision.

We applied the signature extraction and approximate matching algorithms proposed in Chapter 4 with k set as 0 on the dataset collected in our smart home testbed where each home entrance activity was repeated for at least 10 times over 2 weeks. For each matched entrance activity, we first check if its name corresponds to a ‘potentially abnormal’ activity and then check the timestamp to see if it is ’time-dependent’ and it happens during midnight following Algorithm 6.3. For both of the above cases, we will mark the home entrance activity as a potential anomaly and report it to the users. Our experiments of applying Algorithm 6.3 on the dataset which contains all 90 different kinds of entrance activities confirm that our system can effectively identify intrusive and abnormal home entrance activities for home safety monitoring.

6.6 Related Work

The raising security and privacy threats of smart home IoT devices have drawn great research attention [1, 8, 12, 13, 18, 20, 25, 30, 31, 47, 61, 68, 69, 72, 84, 85, 94, 96, 100, 106, 107, 109, 111, 120, 121, 132, 138, 140, 145, 147, 156, 157]. Most of the existing work focuses on either detecting abnormal flows of known attacks such as port scanning and DoS attacks at the network traffic level [20, 94] or applying the domain knowledge such as smart home layout and IoT devices' deployment location for anomaly detection at the device event level with the assumption that the event logs of all IoT devices are available [13, 30, 31, 47]. Some researchers have been working on inferring the device event information from the home network traffic by utilizing the fact that IoT devices generate unique signatures when device events are triggered [1, 135]. However, little effort has been devoted to extracting the controller spatial information from the home network traffic.

DÏoT [94] proposes a federated learning model for detecting abnormal network traffic. Survey paper [20] reviews the literature work in network traffic level anomaly detection using machine learning. These solutions achieve high accuracy and efficiency in detecting cyber attacks such as port scanning, brute forcing credential, and DoS attacks, but fail to work if the IoT devices have already been compromised by attackers.

HAWatcher [47] is a semantics-aware anomaly detection system for appified smart homes by generating and enforcing hypothetical correlations based on semantic information. The work in [30] identifies device event constraints in physical channels. Peeves [13] verifies the validity of a device event based on the state and sensory data of nearby IoT devices. However, all of them rely on the strong assumption that the complete device event logs or device state information are always available.

IoTDuet, to the best of our knowledge, is the first effort in studying the spatial information of IoT device events from the home network traffic. IoTDuet is able to determine whether a device event is triggered locally and remotely by monitoring the traffic of all devices connected to the home network. Such spatial information is crucial for understanding the behavior of the smart home ecosystem and home safety monitoring.

6.7 Conclusions

In this study, we propose a system named IoTDuet for extracting spatial information of IoT device events about whether they are triggered by controlling devices connected to the home network. We explore the extracted spatial information of each IoT device event for applications of abnormal device event detection and home entrance safety monitoring. We extensively evaluated the performance of IoTDuet and applications of home safety monitoring on a real-world smart home testbed.

Chapter 7

IOT SYSTEM VULNERABILITY ANALYSIS AND NETWORK HARDENING WITH SHORTEST ATTACK TRACE IN A WEIGHTED ATTACK GRAPH

7.1 Basic Concepts

In this section, we first present the concept of attack graphs in network security. We then discuss how to conduct the system-level security analysis of IoT systems using attack graphs. We show that the concept of attack trace [40] is critical in analyzing the attack graphs. Subsequently, we define weighted attack graphs and discuss attack traces and shortest attack traces in a weighted attack graph.

7.1.1 Attack Graph

The concept of attack graphs has been widely adopted in cyber security to provide a systematic view of network security and to analyze the vulnerabilities in the system. State-based attack graphs [123] model the network system as a finite state machine where state transitions correspond to the intruder's atomic attacks. However, the size of the state-based attack graphs could scale exponentially to the input size, thus limiting their applications. Logical attack graphs [98, 99], on the other hand, are more efficient as the logical attack graphs can be generated in polynomial time. In this study, we focus on the logical attack graph model, and we refer to logical attack graphs as attack graphs in the rest of this chapter.

We use `MulVAL` [99] to generate attack graphs. The inputs to `MulVAL` are system configurations and interaction rules in Datalog tuples and Datalog rules. Then the `MulVAL` reasoning engine will call the Prolog system XSB [117] to evaluate the interaction rules on input facts and subsequently output the logical attack graph.

The `MulVAL`-generated logical attack graphs are directed graphs and could contain *cycles* [98]. There are three kinds of vertices in the logical attack graph: *primitive fact* vertices, *derived fact* vertices, and *rule* vertices. The attack goal is a special *derived fact* vertex. The edges in the graph represent the "depends on" relation. Each derived fact vertex is dependent on any one of its incoming neighbors. Hence a derived fact vertex is also called an `OR` vertex. Each rule vertex is dependent on all of its incoming neighbors. Hence a rule vertex is also called an `AND` vertex.

7.1.2 System-Level IoT Security Analysis

An attack graph conveys critical information regarding the system vulnerabilities, making it a powerful model for methodically measuring and analyzing IoT system security. IOTA [40] formally develops a framework for efficiently constructing attack graphs given the IoT system configurations.

In IOTA, the attack graph of an IoT system is generated using MulVal [99] with Prolog clauses representing the exploit models and device dependencies as inputs. The exploit models are built by scanning the IoT system configurations for individual vulnerability. Three types of IoT device dependencies are identified by IOTA, including app-based dependency, indirect physical dependency, and direct physical dependency. The app-based dependencies are extracted by applying natural language processing (NLP) techniques to the descriptions of smart home apps. IOTA [40] proposes the metrics of the *shortest attack trace* to an attack goal and the *blast radius* of a vulnerability for interpreting the generated attack graph. A recursive algorithm is proposed to calculate the shortest attack trace to the attack goal in the attack graph.

The authors of [3] study the IoT device deployment problem systematically by considering the whole IoT system with the attack graph. Both the problem of deploying all required IoT devices with minimal security implications and computing the maximal number of IoT devices to be deployed without increasing the security risk of the network are studied in [3]. A heuristic search algorithm for solving both optimization problems is designed by [3].

7.1.3 Weighted Attack Graph

We extend the concept of attack graphs [98] and propose the concept of weighted attack graphs.

Definition 7.1 (**Weighted Attack Graph**)**.** A weighted attack graph is a directed graph $G = (V_p, V_d, V_r, E, w, g)$, where V_p, V_d and V_r denote the set of *primitive fact* vertices (`source` vertices), the set of *derived fact* vertices (`OR` vertices), and the set of *rule* vertices (`AND` vertices), respectively; $E \subseteq \{(V_p \cup V_d) \times V_r\} \cup \{V_r \times V_d\}$ is the set of directed edges; $w(v) \geq 0$ is the *weight* for a vertex v; $w(e) \geq 0$ the *weight* for an edge $e \in E$; $g \in V_d$ is the attacker's goal. $\square$

In the above definition, the notation $w(\cdot)$ is *overloaded*: $w(v)$ denotes the weight of vertex $v \in V_p \cup V_d \cup V_r$, while $w(e)$ denotes the weight of edge $e \in E$. It should be noted that this notation overloading simplifies the presentation without creating ambiguity, as its meaning is clear from the context.

Let G be a weighted attack graph. We use V_p^G to denote the set of primitive fact vertices in G, V_d^G to denote the set of derived fact vertices in G, V_r^G to denote the set of rule vertices in G, E^G to denote the set of edges in G, g^G to denote the goal vertex in G, and w^G to denote the weight function in G. We also use V^G to denote the set of all vertices, i.e., $V^G = V_p^G \cup V_d^G \cup V_r^G$. When the graph G is clear from the context, we may drop the superscript and use the notations $V_p, V_d, V_r, V, E, g, w$.

The concept of *weighted attack graph* contains the concept of (unweighted) *attack graph* studied in the literature [40, 98, 116, 123] as a special case. When we restrict $w(v) = 0, \forall v \in V$ and $w(e) = 1, \forall e \in E$, the weighted attack graph reduces to the classic (unweighted) attack graph. For this reason, we will use the notations of weighted attack graph and (unweighted) attack graph interchangeably, unless specified otherwise.

Note that not all vulnerabilities are the same in terms of penetrability and ubiquity. Therefore, the inclusion of vertex and edge weights in attack graphs can more accurately characterize the vulnerabilities of the IoT system. This is the main motivation for us to study weighted attack graphs. We will discuss the impact and cost of networking hardening in Section 7.3.

7.1.4 Attack Trace

Given an attack graph, an attacker can have different attacking strategies in order to achieve the attack goal. To model the attacker's attacking strategies, we adopt the concept of attack trace introduced in IOTA [40], which can accurately profile the dependency relationships between vertices in an attack graph. Note that IOTA [40] defines attack traces for unweighted attack graphs. In this study, we extend the concept of attack traces to weighted attack graphs in Definition 7.2.

Definition 7.2 (**Attack Trace in a Weighted Attack Graph**). Let G be a weighted attack graph. Let $t \in V^G$ be any vertex in G. An *attack trace to vertex t in G*, denoted by $T^{G,t}$, is a subgraph of G that satisfies the following properties:

1. Let v be any vertex in $T^{G,t}$. If $v \in V_d^G$, then the in-degree of v in $T^{G,t}$ is 1. In other words, for any `OR` vertex v in $T^{G,t}$, $T^{G,t}$ contains exactly one of the edges in $\{(u,v) | (u,v) \in E^G\}$.
2. Let v be any vertex in $T^{G,t}$. If $v \in V_r^G$, then the in-degree of v in $T^{G,t}$ is equal to the in-degree of v in G. In other words, for any `AND` vertex v in $T^{G,t}$, $T^{G,t}$ contains all edges in the set $\{(u,v) | (u,v) \in E^G\}$.
3. Let v be any vertex in $T^{G,t}$. If the in-degree of v in $T^{G,t}$ is 0, then $v \in V_p^G$. In other words, every source vertex in $T^{G,t}$ is a primitive fact vertex.
4. Vertex t is the only vertex in $T^{G,t}$ with out-degree 0 in $T^{G,t}$.

The *height* of the attack trace $T^{G,t}$, denoted by $\mathcal{H}(T^{G,t})$, is the length of the longest path in $T^{G,t}$, where the length of a path is the summation of the weights of the vertices and edges on the path. An attack trace to vertex g is called an attack trace of G. We may use T^G to denote $T^{G,g}$ since the goal vertex g is unique in G. $\square$

Example. To illustrate the concept of attack trace in a weighted attack graph, consider the attack graph G in Figure 7.1(a). We assume that $w(v) = 0$ for each vertex v and $w(e) = 1$ for each edge in G. There are two attack traces to node g in the attack graph G:

1. Attack trace $T_1^{G,g}$ with the set of vertices $\{v_p^2, v_p^3, v_r^2, v_d^2, v_r^4, g\}$, and the set of edges $\{(v_p^2, v_r^2), (v_p^3, v_r^2), (v_r^2, v_d^2), (v_d^2, v_r^4), (v_r^4, g)\}$. This attack trace is highlighted in orange in Figure 7.1(a).
2. Attack trace $T_2^{G,g}$ with the set of vertices $\{v_p^1, v_r^1, v_d^1, v_r^3, v_d^2, v_r^4, g\}$, and the set of edges $\{(v_p^1, v_r^1), (v_r^1, v_d^1), (v_d^1, v_r^3), (v_r^3, v_d^2), (v_d^2, v_r^4), (v_r^4, g)\}$

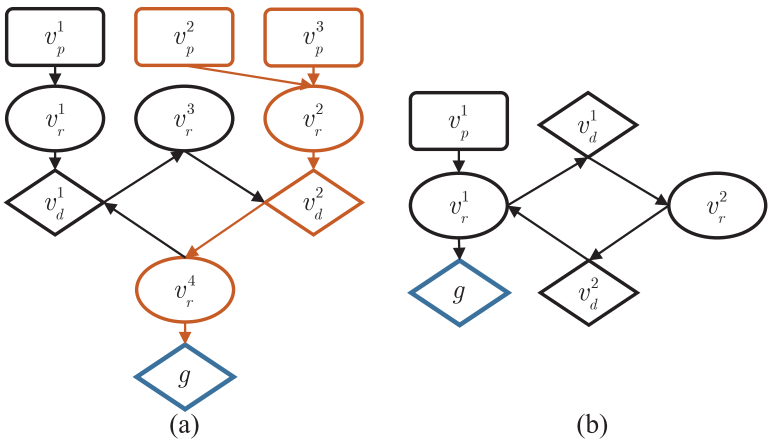


Figure 7.1. (a) An Attack Graph With Two Attack Traces (b) An Attack Graph With No Attack Trace

The height of attack trace $T_1^{G,g}$ is 4, while the height of attack trace $T_2^{G,g}$ is 6.

Definition 7.3 (**Shortest Attack Trace**)**.** Let G be a weighted attack graph. Let $t \in V^G$ be any vertex in G. Let $T_{sh}^{G,t}$ be an attack trace in G. $T_{sh}^{G,t}$ is said to be a shortest attack trace to vertex t, if $\mathcal{H}(T_{sh}^{G,t}) \leq \mathcal{H}(T^{G,t})$ for any attack trace $T^{G,t}$ to vertex t. A shortest attack trace to vertex $g \in V^G$ (the attacker's goal) is called a shortest attack trace of G. $\square$

The concept of (unweighted) attack trace and shortest attack trace was introduced in [40] for unweighted attack graphs, where the height of an attack trace is defined as the longest path (measured by hop-count) in the attack trace. One can verify that the height of an attack trace defined above is the same as the height of an attack trace defined in [40] when $w(v) = 0, \forall v \in V$ and $w(e) = 1, \forall e \in E$.

A given attack graph may have multiple nonidentical shortest attack traces. However, for any two shortest attack traces T_1 and T_2 of attack graph G, their heights must be equal, i.e., $\mathcal{H}(T_1) = \mathcal{H}(T_2)$. To simplify the presentation without confusion, we will use T_{sh} to denote a shortest attack trace.

Following a shortest attack trace is the *optimal* strategy for the attacker, i.e., it requires the *least* effort/time for the attacker to gain control of the attack goal. Therefore $\mathcal{H}(T_{\text{sh}})$ is the least effort/time that the attacker needs to spend to gain control of the attack goal.

7.2 Computing a Shortest Attack Trace: Perspective of the Attacker

In this section, we present a novel polynomial time algorithm for computing a shortest attack trace in a weighted attack graph. Our algorithm is inspired by the approach in IOTA [40].

7.2.1 A Novel Algorithm for Computing an SAT

Our algorithm for computing a shortest attack trace is Algorithm 7.1, named $\texttt{SAT}(G)$. $\texttt{SAT}(G)$ takes an attack graph G as input. It either stops in Line 26, claiming that there is no attack trace in G, or stops in Line 36, outputting a shortest attack trace T_{sh}.

Before illustrating $\texttt{SAT}(G)$ with examples, we explain its main steps in the following. Line 1 creates an empty priority queue PQ such as the Fibonacci heap [46]. We use a color system for the vertices. A vertex v is `WHITE` before it is inserted into PQ, `GRAY` when it is on PQ, and `BLACK` after it is extracted from PQ. The in-degree of v is recorded in $v.in$. The vertex attribute $v.done$ denotes the number of edges into v that have been traversed by the algorithm. If v is an `OR` vertex, $v.height$ is initialized to ∞, and decreased to the *minimum* of the heights of all attack traces to v computed so far. If v is an `AND` vertex, $v.height$ is initialized to $-\infty$, and increased to the *maximum* of

Algorithm 7.1: SAT(G)

```
    Input: $G = (V_p, V_d, V_r, E, w, g)$: an attack graph
    Output: $T_{\text{sh}}$: a shortest attack trace to $g$ in $G$.
 1  Create an empty priority queue $PQ$;
 2  for $\forall v \in V_d$ do
 3      $v.type \leftarrow 1$; $v.in \leftarrow 0$; $v.done \leftarrow 0$; $v.height \leftarrow \infty$; $v.color \leftarrow$ WHITE;
 4  for $\forall v \in V_r$ do
 5      $v.type \leftarrow 2$; $v.in \leftarrow 0$; $v.done \leftarrow 0$; $v.height \leftarrow -\infty$; $v.color \leftarrow$ WHITE;
 6  for $\forall e = (x, y) \in E$ do
 7      $y.in \leftarrow y.in + 1$;
 8  for $\forall v \in V_p$ do
 9      $v.type \leftarrow 0$; $v.height \leftarrow w(v)$; Insert $v$ to $PQ$; $v.color \leftarrow$ GRAY;
10  while $PQ \neq \emptyset$ do
11      $u \leftarrow$ ExtractMin($PQ$); $u.color \leftarrow$ BLACK;
12      if ($u == g$) goto 27;
13      for $\forall v \in u.adj$ with $v.color \neq$ BLACK do
14          $temp \leftarrow u.height + w(u, v) + w(v)$;
15          if ($v.type == 1$) then
16              if ($v.done == 0$) then
17                  $v.height \leftarrow temp$; $v.parent \leftarrow u$;
18                  Insert $v$ to $PQ$; $v.color \leftarrow$ GRAY;
19              else if ($temp < v.height$) then
20                  $v.height \leftarrow temp$; $v.parent \leftarrow u$;
21              $v.done \leftarrow v.done + 1$;
22          if ($v.type == 2$) then
23              $v.height \leftarrow \max\{v.height, temp\}$; $v.done \leftarrow v.done + 1$;
24              if $v.done == v.in$ then
25                  Insert $v$ to $PQ$; $v.color \leftarrow$ GRAY;
26  stop: There is no attack trace in the attack graph $G$;
27  Create an empty FIFO queue $Q$; Insert $g$ to $Q$;
28  Create an empty set $T_{\text{sh}}$ of edges in the shortest attack trace;
29  while $Q \neq \emptyset$ do
30      $v \leftarrow Dequeue(Q)$;
31      if ($v.type == 1$) then
32          $u \leftarrow v.parent$; Insert $u$ to $Q$; $T_{\text{sh}} \leftarrow T_{\text{sh}} \cup \{(u, v)\}$;
33      if ($v.type == 2$) then
34          for $\forall u \in V$ such that $(u, v) \in E$ do
35              Insert $u$ to $Q$; $T_{\text{sh}} \leftarrow T_{\text{sh}} \cup \{(u, v)\}$;
36  Output $T_{\text{sh}}$;
```

$\{u.height + w(u,v) + w(v) | u.color = \texttt{BLACK}\}$. When all incoming neighbors of v are extracted from PQ, a shortest attack trace to vertex v is computed, and its height is stored in $v.height$. These attributes for $v \in V_d \cup V_r$ are initialized in Lines 2-7.

In Lines 8-9, each vertex $v \in V_p$ is inserted into PQ, with $v.height$ set to its final value of $w(v)$. The main body of the algorithm is the while-loop in Lines 10-25, where the vertex on PQ with the minimum height value is extracted. Whenever a vertex u is extracted from PQ, edges in the form (u,v) are traversed and the attributes of v are updated accordingly. An `OR` vertex v is inserted into PQ as soon as an attack trace $T^{G,v}$ is computed (note that $T^{G,v}$ does not have to be a shortest attack trace to v). An `AND` vertex v is inserted into PQ as soon as a shortest attack trace $T^{G,v}$ is computed. If the goal vertex g is extracted, the algorithm goes to Line 27 to output the computed shortest attack trace T_{sh} and exits in Line 36. If PQ becomes empty before g is inserted into it, the algorithm exits in Line 26, claiming that G does not contain an attack trace.

7.2.2 Walk-through Examples

We use the examples in Figure 7.1 to illustrate Algorithm 7.1. The attack graph in Figure 7.1(a) has two attack traces. The attack graph in Figure 7.1(b) has no attack trace. We assume that $w(v) = 0$ for each vertex v and $w(e) = 1$ for each edge in Figure 7.1. Hence, the weighted attack graphs reduce to unweighted attack graphs.

When Algorithm 7.1 is applied to the attack graph in Figure 7.1(a), the main steps are as follows. (S1) An empty priority queue PQ is initialized and the primitive fact vertices v_p^1, v_p^2, v_p^3 are inserted into PQ with $v_p^1.height = 0$, $v_p^2.height = 0$, $v_p^3.height = 0$. (S2) v_p^1 is extracted from PQ. Edge (v_p^1, v_r^1) is traversed,and $v_r^1.height$

is increased from $-\infty$ to $v_p^1.height + w(v_p^1, v_r^1) + w(v_r^1) = 1$. Since we have traversed all edges going into v_r^1, the `AND` vertex v_r^1 is inserted into PQ. (S3) v_p^2 is extracted from PQ. Edge (v_p^2, v_r^2) is traversed, and $v_r^2.height$ is increased from $-\infty$ to $v_p^2.height + w(v_p^2, v_r^2) + w(v_r^2) = 1$. Unlike the previous step, the `AND` vertex v_r^2 is not inserted into PQ at this moment, since we have not traversed all edges going into v_r^2 yet. (S4) v_p^3 is extracted from PQ. Edge (v_p^3, v_r^2) is traversed, $v_r^2.height$ remains unchanged since $v_p^3.height + w(v_p^3, v_r^2) + w(v_r^2)$ is not larger than $v_r^2.height$. However, since we have now traversed all edges going into v_r^2, the `AND` vertex v_r^2 is inserted into PQ. (S5) v_r^1 is extracted from PQ. Edge (v_r^1, v_d^1) is traversed, and $v_d^1.height$ is decreased from ∞ to 2. The `OR` vertex v_d^1 is inserted into PQ. (S6) v_r^2 is extracted from PQ. The `OR` vertex v_d^2 is inserted into PQ with $v_d^2.height = 2$. (S7) v_d^1 is extracted from PQ. The `AND` vertex v_r^3 is inserted into PQ with $v_r^3.height = 3$. (S8) v_d^2 is extracted from PQ. The `AND` vertex v_r^4 is inserted into PQ with $v_r^4.height = 3$. (S9) v_r^3 is extracted from PQ. The edge (v_r^3, v_d^2) is traversed. Since v_d^2 has been extracted from PQ, the attributes at v_d^2 are not updated. (S10) v_r^4 is extracted from PQ. The `OR` vertex g is inserted into PQ with $g.height = 4$. (S11) g is extracted from PQ. The algorithm outputs the attack trace $\{(v_r^4, g), (v_d^2, v_r^4), (v_r^2, v_d^2), (v_p^2, v_r^2), (v_p^3, v_r^2)\}$ and stops in Line 36. The height of the computed shortest attack trace is 4. This example is representative of cases where the attack graph contains at least one attack trace.

When Algorithm 7.1 is applied to the attack graph in Figure 7.1(b), the main steps are as follows. (S1) An empty priority queue PQ is initialized and the primitive fact vertex v_p^1 is inserted into PQ with $v_p^1.height = 0$. (S2) v_p^1 is extracted from PQ. The edge (v_p^1, v_r^1) is traversed, and $v_r^1.height$ is increased from $-\infty$ to $v_p^1.height + w(v_p^1, v_r^1) + w(v_r^1) = 1$. Since we have not traversed all edges going into v_r^1 (edge (v_d^2, v_r^1) has not been traversed) yet, the `AND` vertex v_r^1 is not inserted into PQ at this moment. (S3) Algorithm 7.1 finds $PQ = \emptyset$ and goes to Line 26. It stops, claiming that there is no attack trace in the attack graph shown in Figure 7.1(b). This example is representative of cases where the attack graph does not contain any attack trace.

7.2.3 Analysis of the Algorithm

In this section, we analyze the properties of the algorithm. In Theorem 7.1, we analyze the worst-case running time of the algorithm. In Theorems 7.2-7.3, we prove the correctness of the algorithm.

Theorem 7.1. Let n be the number of vertices in G and m be the number of edges in G. The worst-case running time of Algorithm 7.1 is $O(m + n \log n)$. □

Proof. The time spent on Lines 1-9 is $O(n + m)$ as we spend $O(1)$ time for each vertex and each edge. The while-loop from Line 10 to Line 25 performs at most n ExtractMin operations, at most n Insertion operations, and at most m DecreaseKey operations. If we use the Fibonacci heap [46] to implement the priority queue, this portion has a worst-case running time $O(m + n \log n)$. Line 27 takes $O(1)$ time. The while-loop from Line 29 to Line 35 takes $O(m)$ time, as there are at most m edges in T. Hence the worst-case time complexity of Algorithm 7.1 is $O(m + n \log n)$. □

Remarks. At the high level, our algorithm for computing a shortest attack trace follows the same principle as the algorithm in [40]. However, there are subtle differences that may affect the worst-case running time. The algorithm in [40] uses a top-down approach (without memoization). Our algorithm uses a bottom-up approach, where no instance is solved more than once. Our algorithm can compute a shortest attack trace whenever an attack trace exists, and can recognize if an attack trace does not exist, in polynomial time (refer to Theorems 7.1 and 7.2).

Theorem 7.2. If the attack graph G contains an attack trace, then Algorithm 7.1 correctly computes a shortest attack trace, with the edges in the shortest attack trace stored in T_{sh}. In this case, the algorithm exits in Line 36. □

Proof. Our proof relies on the following claims:

(a) Let $v \in V^G$ be any vertex in G. The following is always true throughout the execution of the algorithm: $v.color$ is `WHITE` if and only v has not been inserted into the priority queue PQ, $v.color$ is `GRAY` if and only v is on the priority queue PQ, and $v.color$ is `BLACK` if and only v has been extracted from the priority queue PQ.

(b1) Let v be any `OR` vertex in G. The value $v.height$ is monotonically non-increasing during the execution of the algorithm.

(b2) Let v be any `OR` vertex in G. The first time $v.height$ is reduced from ∞ to a real number, v is inserted into PQ, and $v.height$ is the height of *some* (not necessarily the shortest) attack trace to vertex v; When v is extracted from PQ, $v.height$ is the height of a shortest attack trace to vertex v.

(c1) Let v be any `AND` vertex in G. The value $v.height$ is monotonically non-decreasing during the execution of the algorithm.

(c2) Let v be any `AND` vertex in G. When v is inserted into PQ, $v.height$ is the height of a shortest attack trace to vertex v. The value $v.height$ will not be changed again after v is inserted into PQ. When v is extracted from PQ, $v.height$ is the height of a shortest attack trace to vertex v.

(d) If vertex α is extracted from PQ before vertex β is extracted, then $\alpha.height$ (at the time α is extracted) is less than or equal to $\beta.height$ (at the time β is extracted).

Claim (a) follows directly from the algorithm. We note that vertex v becomes `GRAY` and gets inserted into PQ only in Line 9 (for $v \in V_p$), Line 18 (for $v \in V_d$), and Line 25 (for $v \in V_r$); and vertex v gets extracted from PQ and becomes `BLACK` only in Line 11.

To prove Claim (b1), we note that for $v \in V_d^G$, $v.height$ is initialized to ∞ in Line 5, and changed (to a smaller value) in Line 17 and Line 20. To prove Claim (c1), we note that for $v \in V_r^G$, $v.height$ is initialized to $-\infty$ in Line 7, and changed (to a larger value) in Line 23.

To prove Claim (d), we note that vertices is extracted from PQ by the ExtractMin operation. Therefore, when α is extracted, the height of α is the smallest among all vertices on PQ. Since the vertex weights and edge weights are all non-negative, for any vertex v that has not been extracted from PQ before vertex α, we will not have $v.height < \alpha.height$ when $v.done \geq 1$. This proves Claim (d).

Claims (b2) and (c2) can be proved using mathematical induction. Let v be an `OR` vertex. The first time an incoming neighbor of v is extracted from PQ, the height of v is decreased from ∞ to a real number (which is the height of an attack trace to v). This value may be further reduced when other incoming neighbors of v are extracted from PQ. When all incoming neighbors of v are extracted from PQ, the height of v will no longer be decreased.

Let v be an `AND` vertex. $v.height$ is initialized to $-\infty$ in Line 5. Every time an incoming neighbor v' of v is extracted, $v.height$ will be set to the maximum of its current value and $v'.height + w(v', v) + w(v)$. Therefore, when all incoming neighbors of v are extracted from PQ, the $v.height$ is the height of a shortest attack trace to vertex v. The above analysis, together with (d), proves (b2) and (c2).

Now assume that there is an attack trace in G. Since the number of attack traces in G is finite, there must be a shortest attack trace. Let h_{opt} be the height of the shortest attack trace to g. Following the above analysis, vertex v will be inserted into and extracted from PQ provided that the height of the shortest attack trace to v is *smaller than* the h_{opt}. Hence, the attack goal g will be inserted into and extracted from PQ. Therefore, Algorithm 7.1 exits at Line 36. □

Theorem 7.3. If the attack graph G contains no attack trace, then Algorithm 7.1 stops in Line 26. □

Proof. The algorithm stops after $O(m + n \log n)$ basic operations. If it stops at Line 36, it must have computed a shortest attack trace. The only other place for the algorithm to stop is Line 26. Therefore if there is no attack trace to g, Algorithm 7.1 must stop at Line 26. □

7.3 Optimal Network Hardening: Perspective of the Defender

In Section 7.2, we presented an efficient algorithm for computing a shortest attack trace. The best strategy for the attacker to gain access to the attack goal g is to launch an attack along a shortest attack trace. Therefore, the height of a shortest attack trace, $\mathcal{H}(T_{\mathrm{sh}})$, is a good metric to measure the *hardness* for the attacker to gain access to its attack goal g. The larger the value of $\mathcal{H}(T_{\mathrm{sh}})$, the less vulnerable the system is.

Assume that we can harden a network element z (a vertex or an edge) to increase the value of $w(z)$. Then we can increase the height of the shortest attack trace by hardening some selected network elements. However, hardening a network element comes with a cost. Also, some network elements are hardenable, while others are not. Therefore, a natural question to ask is: *What is the best strategy to harden the* *network with a given budget constraint*? This section is devoted to answering the above question.

7.3.1 The Network Hardening Problem

A *network element* in this study refers to either a vertex or an edge in G. Since there are $n = |V|$ vertices and $m = |E|$ edges in G, we have a total of $N = n + m$ network elements. We define a one-to-one mapping η from the set of integers $\{1, 2, \ldots, N\}$ to the set of network elements $V \cup E$ such that $\eta(k) \in V$ for $k = 1, 2, \ldots, n$, and $\eta(k) \in E$ for $k = n + 1, n + 2, \ldots, n + m$. We use a binary-valued array $\phi[1 : N]$ to indicate whether a network element is hardenable. In particular, $\phi[k] = 1$ if $\eta(k)$ is hardenable, and $\phi[k] = 0$ otherwise.

Let $k \in [1, N]$ be an integer. If $\phi[k] = 1$, we can harden network element $\eta(k)$ with a cost of $c[k] > 0$ to increase the weight of $\eta(k)$ from $w(\eta(k))$ to $w(\eta(k)) + \delta[k]$. If $\phi[k] = 0$, $\eta(k)$ is not hardenable. For convenience, we define $\delta[k] = 0$ and $c[k] = \infty$ in this situation.

Definition 7.4. Let the attack graph G be given, together with mapping η and network hardening information ϕ, δ, and c. A binary-valued array $X[1 : N]$ is said to be a feasible hardening strategy for budget $B \geq 0$, if $X[k] \leq \phi[k],\ k = 1, 2, \ldots, N$ and $\sum_{k=1}^{N} X[k] \times c[k] \leq B$. □

The meaning of the array $X[1 : N]$ as a hardening strategy is the following. For $k = 1, 2, \ldots, N$, network element $\eta(k)$ is hardened if and only if $X[k] = 1$. Since $X[k] \leq \phi[k]$, only hardenable network elements will be hardened. Since $\sum_{k=1}^{N} X[k] \times c[k] \leq B$, the total cost for hardening does not exceed the given budget B.

We use $G(X)$ to denote the hardened attack graph corresponding to X, and use $T_{\text{sh}}(X)$ and $\mathcal{H}(T_{\text{sh}}(X))$ to denote the shortest attack trace of $G(X)$ and the height of $T_{\text{sh}}(X)$, respectively. The decision version and the optimization version of the network hardening problem are formally defined in the following.

Definition 7.5 (Decision Hardening Problem)**.** Let the attack graph G be given, together with network hardening information η, ϕ, δ, and c. Let $\mathbb{B}$ be the budget for network hardening, and $\mathbb{H}$ be the desired level of network hardness. The decision network hardening problem (**DHP**) asks whether there exists a feasible hardening strategy X such that $\mathcal{H}(T_{\text{sh}}(X)) \geq \mathbb{H}$. When the answer is `YES`, the problem also asks for the corresponding hardening strategy X. □

Definition 7.6 (Optimization Hardening Problem)**.** Let the attack graph G be given, together with network hardening information η, ϕ, δ, and c. Let $\mathbb{B}$ be the budget for network hardening. The optimization network hardening problem (**OHP**) asks for a feasible hardening strategy X_{opt} such that $\mathcal{H}(T_{\text{sh}}(\mathrm{X}_{\text{opt}})) \geq \mathcal{H}(T_{\text{sh}}(X))$ for every feasible hardening strategy X. □

OHP can be formulated as the following optimization problem.

$$\texttt{maximize}\ \mathcal{H}(T_{\text{sh}}(X)) \tag{7.1}$$

$$\texttt{s.t.}\ X[k] \in \{0,1\},\ k = 1, 2, \ldots, N, \tag{7.2}$$

$$X[k] \leq \phi[k],\ k = 1, 2, \ldots, N, \tag{7.3}$$

$$\sum_{k=1}^{N} X[k] \times c[k] \leq \mathbb{B}. \tag{7.4}$$

In Section 7.3.2, we will study the computational complexity of the network hardening problem. In Sections 7.3.3 and 7.3.4, we will present optimal and heuristic algorithms, respectively, for solving the **OHP** problem.

7.3.2 Hardness of the Problem

We present a polynomial-time reduction from **Knapsack** [50] to **DHP**. An instance of **Knapsack** is given by a finite set U, a *size* $s(u) \in Z^+$ and a *reward* $r(u) \in Z^+$ for each $u \in U$, a *size constraint* $\mathbb{B} \in Z^+$, and a *reward goal* $\mathbb{R} \in Z^+$. It asks for a subset $S \subseteq U$ such that

$$\sum_{u \in S} s(u) \leq \mathbb{B} \text{ and } \sum_{u \in S} v(u) \geq \mathbb{R}. \tag{7.5}$$

Theorem 7.4 (Hardness of Network Hardening)**.** The network hardening problems **DHP** and **OHP** are both NP-hard. □

Proof. Let $\mathcal{I}_1 = (U, s(\cdot), r(\cdot), \mathbb{B}, \mathbb{R})$ be an arbitrary instance of **Knapsack**, where $U = \{u_1, u_2, \ldots, u_K\}$. If K is an even integer, we construct a corresponding instance $\mathcal{I}'_1 = (U', s'(\cdot), r'(\cdot), \mathbb{B}', \mathbb{R}')$ with $|U'| = |U| + 1$ such that

- $U' = U \cup \{u_{K+1}\}$,
- $s'(u_k) = s(u_k)$, $1 \leq k \leq K$,
- $s'(u_{K+1}) = \min\{s(u_k) | 1 \leq k \leq K\}$,
- $r'(u_k) = r(u_k)$, $1 \leq k \leq K$,
- $r'(u_{K+1}) = 1 + \sum_{k=1}^{K} r(u_k)$,
- $\mathbb{B}' = \mathbb{B} + s(u_{K+1})$,
- $\mathbb{R}' = \mathbb{R} + r(u_{K+1})$.

Since $s(u_{K+1}) \leq s(u_k)$ for $1 \leq k \leq K$ and $r(u_{K+1}) > \sum_{k=1}^{K} r(u_k)$, $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}'_1$ has a solution. In addition, any solution to $\mathcal{I}'_1$ must be the union of S and $\{u_{K+1}\}$, where S is a solution to $\mathcal{I}_1$. Therefore, without loss of generality, we may assume that for instance $\mathcal{I}_1$, K is an odd integer.

Given instance $\mathcal{I}_1$ of **Knapsack**, we construct a corresponding instance $\mathcal{I}_2$ of **DHP** in the following. The weighted attack graph is $G = (V_p, V_d, V_r, E, g, w)$, where $V_p = \{v_1\}$, $V_d = \{v_3, v_5, v_7, \ldots, v_K\}$, $V_r = \{v_2, v_4, v_6, \ldots, v_{K-1}\}$, $E = \{(v_{k-1}, v_k) | k = 2, 3, \ldots, K\}$, $g = v_K$, and $w(v_k) = 1$ for each $k = 1, 2, \ldots, K$ and $w(e) = 1$ for each $e \in E$. There are K vertices and $K - 1$ edges in G, leading to $N = 2K - 1$ network elements. Define η, ϕ, δ, and c such that

$$\eta(k) = v_k,\ \phi[k] = 1, \qquad k = 1, 2, \ldots, K, \tag{7.6}$$

$$\delta[k] = r(u_k),\ c[k] = s(u_k), \qquad k = 1, 2, \ldots, K, \tag{7.7}$$

$$\eta(K + k) = (v_k, v_{k+1}),\ \phi[K + k] = 0, \qquad k = 1, 2, \ldots, K - 1, \tag{7.8}$$

$$\delta[K + k] = 0,\ c[K + k] = \infty, \qquad k = 1, 2, \ldots, K - 1. \tag{7.9}$$

Define the hardening budget to be $\mathbb{B}$, and the desired hardness level to be $\mathbb{H} = \mathbb{R} + (K + 1)$. Then $\mathcal{I}_1$ has a solution if and only if **DHP** has a solution. In addition, if X is a solution to **DHP**, then $S = \{\eta(k) | X[k] = 1\}$ is a solution to **Knapsack** and vice versa. Since **Knapsack** is NP-hard, we have proved that **DHP** is NP-hard. Since **OHP** is the optimization version of **DHP**, **OHP** is also NP-hard. □

7.3.3 An Exact Algorithm

We design a branch and bound algorithm for computing an optimal solution to **OHP**. The algorithm is listed in Algorithm 7.2. While the branch and bound algorithm paradigm has been known for a long time, the bounding technique in our algorithm is novel. Its effectiveness will be demonstrated in our evaluation results.

Theorem 7.5. Algorithm 7.2 (`ExactBnB`) always computes an optimal hardening strategy. □

```
Algorithm 7.2: ExactBnB($G, N, \eta, \phi, \delta, c, \mathrm{H}_{\mathrm{best}}, \mathrm{X}_{\mathrm{best}}, \mathbb{B}$)
  Input: $G = (V, E, g)$ is a weighted attack graph; $N = |V| + |E|$ is the number of
         network elements; $\eta : \{1, 2, \ldots, N\} \mapsto V \cup E$ is a one-to-one mapping; $\phi[k]$
         is 1 if $\eta[k]$ is hardenable, 0 otherwise; $\delta[k]$ and $c[k]$ are the added strength
         and hardening cost for network element $\eta[k]$ when $\phi[k]$ is 1; $\mathbb{B}$ is the budget
         constraint for network hardening; $\mathrm{X}_{\mathrm{best}}$ is some hardening strategy; $\mathrm{H}_{\mathrm{best}}$ is
         the height of the shortest attack trace after the network is hardened using
         $\mathrm{X}_{\mathrm{best}}$.
  Output: An optimal hardening strategy given by $\mathrm{X}_{\mathrm{best}}[k]$, $j = 1, 2, \ldots, N$ together
          with the height of the shortest attack trace after hardening using the
          optimal strategy.
1  Create an array $X[1 : N]$ and an array $c_{\mathrm{sum}}[1 : N + 1]$;
2  $c_{\mathrm{sum}}[N + 1] \leftarrow 0$;
3  for $k = N, N - 1, \ldots, 1$ do
4      if ($\phi[k] == 1$) then
5          $c_{\mathrm{sum}}[k] \leftarrow c_{\mathrm{sum}}[k + 1] + c[k]$;
6      else
7          $c_{\mathrm{sum}}[k] \leftarrow c_{\mathrm{sum}}[k + 1]$;
8  $B \leftarrow \mathbb{B}$;
9  Branch1($B, 1, X, \mathrm{X}_{\mathrm{best}}, \mathrm{H}_{\mathrm{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}}$);
10 Branch0($B, 1, X, \mathrm{X}_{\mathrm{best}}, \mathrm{H}_{\mathrm{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}}$);
11 Output $\mathrm{H}_{\mathrm{best}}$ and $\mathrm{X}_{\mathrm{best}}$.
```

Proof. Algorithm 7.2 traverses a decision tree with 2^K leaf vertices, while cutting branches that do not contain a better solution whenever we know it. We use B to denote the *residual budget*, which is the initial budget $\mathbb{B}$ minus the sum of the costs of the subset of network elements selected to be hardened. In general, given the values of $X[j]$ for $1 \leq j \leq k - 1$ such that $X[j] \leq \phi[j]$ for $1 \leq j \leq k - 1$ and $\sum_{j=1}^{k-1} X[j] \times c[j] \leq \mathbb{B}$, we decide whether to explore or cut the branch with $X[k] = 1$, and the branch with $X[k] = 0$.

The Branch for $X[k] = 1$ does not exist if $\phi[k] = 0$ or $\sum_{j=1}^{k} X[j] \times c[j] > \mathbb{B}$. If $\sum_{j=1}^{k} X[j] \times c[j] + \sum_{j=k+1}^{N} c[j] \times \phi[j] \leq \mathbb{B}$, the best solution in the branch corresponding to setting $X[k] = 1$ is to set $X[j] = \phi[j]$ for $j = k + 1, \ldots, N$. In a nutshell, the

Algorithm 7.3: Branch1($B, k, X, \mathrm{X}_{\mathrm{best}}, \mathrm{H}_{\mathrm{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}}$)

```
Input: Current $\mathrm{H}_{\mathrm{best}}$ and feasible values for $X[1:k-1]$
Output: Explore the branch for $X[k] = 1$
1  if ($k > N$ or $\phi[k] == 0$ or $c[k] > B$) then return;
2  $X[k] \leftarrow 1$; $B \leftarrow B - c[k]$;
3  if ($k == N$ or $c_{\mathrm{sum}}[k+1] \leq B$) then
4      $X[j] \leftarrow \phi[j],\ j = k+1, k+2, \ldots, N$;
5      $\mathrm{H}_{\mathrm{new}} \leftarrow \mathcal{H}(T_{\mathrm{sh}}(X))$;
6      if ($\mathrm{H}_{\mathrm{new}} > \mathrm{H}_{\mathrm{best}}$) then
7          $\mathrm{X}_{\mathrm{best}} \leftarrow X$; $\mathrm{H}_{\mathrm{best}} \leftarrow \mathrm{H}_{\mathrm{new}}$;
8  else
9      $X[j] \leftarrow 0,\ j = k+1, k+2, \ldots, N$;
10     $\mathrm{H}_{\mathrm{new}} \leftarrow \mathcal{H}(T_{\mathrm{sh}}(X))$;
11     if ($\mathrm{H}_{\mathrm{new}} > \mathrm{H}_{\mathrm{best}}$) then
12         $\mathrm{X}_{\mathrm{best}} \leftarrow X$; $\mathrm{H}_{\mathrm{best}} \leftarrow \mathrm{H}_{\mathrm{new}}$;
13     Branch1($B, k+1, X, \mathrm{X}_{\mathrm{best}}, \mathrm{H}_{\mathrm{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}}$);
14     Branch0($B, k+1, X, \mathrm{X}_{\mathrm{best}}, \mathrm{H}_{\mathrm{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}}$);
15 $B \leftarrow B + c(k)$;
```

Algorithm 7.4: Branch0($B, k, X, \mathrm{X}_{\mathrm{best}}, \mathrm{H}_{\mathrm{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}}$)

```
Input: Current $\mathrm{H}_{\mathrm{best}}$ and feasible values for $X[1:k-1]$
Output: Explore the branch for $X[k] = 0$
1  if ($k > N$) then return;
2  $X[k] \leftarrow 0$;
3  if ($k == N$ or $c_{\mathrm{sum}}[k+1] \leq B$) then
4      $X[j] \leftarrow \phi[j],\ j = k+1, k+2, \ldots, N$;
5      $\mathrm{H}_{\mathrm{new}} \leftarrow \mathcal{H}(T_{\mathrm{sh}}(X))$;
6      if ($\mathrm{H}_{\mathrm{new}} > \mathrm{H}_{\mathrm{best}}$) then
7          $\mathrm{X}_{\mathrm{best}} \leftarrow X$; $\mathrm{H}_{\mathrm{best}} \leftarrow \mathrm{H}_{\mathrm{new}}$;
8  else
9      $X[j] \leftarrow 0,\ j = k+1, k+2, \ldots, N$;
10     $\mathrm{H}_{\mathrm{new}} \leftarrow \mathcal{H}(T_{\mathrm{sh}}(X))$;
11     if ($\mathrm{H}_{\mathrm{new}} > \mathrm{H}_{\mathrm{best}}$) then
12         $\mathrm{X}_{\mathrm{best}} \leftarrow X$; $\mathrm{H}_{\mathrm{best}} \leftarrow \mathrm{H}_{\mathrm{new}}$;
13     Branch1($B, k+1, X, \mathrm{X}_{\mathrm{best}}, \mathrm{H}_{\mathrm{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}}$);
14     Branch0($B, k+1, X, \mathrm{X}_{\mathrm{best}}, \mathrm{H}_{\mathrm{best}}, G, N, \eta, \phi, \delta, c, c_{\mathrm{sum}}$);
```

algorithm never cuts a branch that contains a better solution than the current best solution. Therefore, the algorithm always computes an optimal solution to **OHP**. □

7.3.4 A Heuristic Algorithm

Since **OHP** is NP-hard, we design a polynomial-time greedy heuristic algorithm for computing a solution to **OHP**. This is listed in Algorithm 7.5. While our greedy heuristic algorithm does not guarantee finding an optimal solution, extensive evaluation results show that the heuristic algorithm produces close-to-optimal solutions.

```
Algorithm 7.5: Greedy($G, N, \eta, \phi, \delta, c, \mathbb{B}$)
Input: $G, N, \eta, \phi, \delta, c$: as in Algorithm 7.2; $\mathbb{B}$: hardening budget.
Output: A feasible hardening strategy $\mathrm{X}_{\mathrm{grd}}$.
1  Create a binary-valued array $\mathrm{X}_{\mathrm{grd}}[1:N]$;
2  $\mathrm{X}_{\mathrm{grd}}[k] \leftarrow 0,\ k = 1, 2, \ldots, N$;
3  $\mathrm{H}_{\mathrm{grd}} \leftarrow \mathcal{H}(T_{\mathrm{sh}}(\mathrm{X}_{\mathrm{grd}}))$; $B \leftarrow \mathbb{B}$; $done \leftarrow 0$;
4  while ($done == 0$) do
5      $done \leftarrow 1$; $\mathrm{R}_{\mathrm{best}} \leftarrow 0$; $\mathrm{k}_{\mathrm{best}} \leftarrow 0$;
6      for $k = 1, 2, \ldots, N$ do
7          if ($\mathrm{X}_{\mathrm{grd}}[k] < \phi[k]$ and $c[k] \leq B$) then
8              $\mathrm{X}_{\mathrm{grd}}[k] \leftarrow 1$;
9              $\mathrm{H}_{\mathrm{new}} \leftarrow \mathcal{H}(T_{\mathrm{sh}}(\mathrm{X}_{\mathrm{grd}}))$; $\mathrm{R}_{\mathrm{new}} \leftarrow \frac{\mathrm{H}_{\mathrm{new}} - \mathrm{H}_{\mathrm{grd}}}{c[k]}$;
10             if ($\mathrm{R}_{\mathrm{new}} > \mathrm{R}_{\mathrm{best}}$) then
11                 $done \leftarrow 0$; $\mathrm{R}_{\mathrm{best}} \leftarrow \mathrm{R}_{\mathrm{new}}$; $\mathrm{k}_{\mathrm{best}} \leftarrow k$;
12             $\mathrm{X}_{\mathrm{grd}}[k] \leftarrow 0$;
13     if ($done == 0$) then
14         $\mathrm{X}_{\mathrm{grd}}(\mathrm{k}_{\mathrm{best}}) \leftarrow 1$; $B \leftarrow B - c[\mathrm{k}_{\mathrm{best}}]$; $\mathrm{H}_{\mathrm{grd}} \leftarrow \mathcal{H}(T_{\mathrm{sh}}(\mathrm{X}_{\mathrm{grd}}))$;
15 output $\mathrm{X}_{\mathrm{grd}}$ and $\mathrm{H}_{\mathrm{grd}}$;
```

Theorem 7.6. Algorithm 7.5 always finds a feasible hardening strategy. Its worst-case running time is $O(K^2(m + n \log n))$, where $n = |V|$, $m = |E|$, and K is the number of hardenable network elements. □

Proof. Algorithm 7.5 only hardens hardenable network elements, and never hardens a subset of network elements whose aggregated cost exceeds the given budget. Therefore, it always produces a feasible hardening strategy. The algorithm performs $O(K)$ iterations, where each iteration requires the computation of $O(K)$ shortest attack traces. Therefore, the worst-case running time is $O(K^2(m + n \log n))$. □

7.3.5 Related Work on Network Hardening

Network hardening is an important problem in cyber security, and the attack graph is an elegant tool to measure system-level vulnerabilities for performing network hardening [27, 33–35, 63, 143, 146, 153]. Islam *et al.* [63] proposes heuristic approaches to perform network hardening by patching a selected subset of vertices in the attack graph with the lowest cost. However, only primitive fact vertices can be patched in [63], thus their model is not general. Similarly, Dewri *et al.*[27] assumes that only primitive fact vertices are patchable at different costs. It models the network hardening problem as finding the optimal patching strategy that minimizes the cost and residual damage to the system. However, Dewri *et al.*[27] applies the attack tree model [7] instead of the attack graph, which has a simpler structure but makes strong assumptions about attackers' abilities. Durkota *et al.* [35] model the defender's hardening strategy as finding the optimal way to add honeypots to a networked system, by which the defender can detect and mitigate attacks at certain costs. They proposed several heuristic algorithms, albeit with limited scalability.

Another set of works takes the probabilistic metric [142] into account when studying the network hardening problem. The network hardening framework proposed by [146] assigns probabilities to the edges in attack graphs to incorporate the attack graph and HMM. Both attack cost and defense cost are modeled in [146] for conducting cost-benefit security analysis. The cost-aware IoT network hardening solution in [153] assigns a cost for removing exploits and initial conditions in the system, and applies a greedy algorithm for finding a cost-effective solution to harden the system. Since all paths to the attack goal need to be calculated to decide which exploits or initial conditions to be removed, it faces scalability issues when the attack graph is large.

Our work has similar motivations of assigning costs and weights to the elements in the attack graphs as that of existing research. However, our problem is more general where all vertices and edges are hardenable. Instead of measuring the attacker's capability as the probability of reaching a specific state, e.g., conquering the attack goal, we assume that the attacker always follows the shortest attack trace to launch an attack. This assumption is intuitive and reasonable, and profiles the attacker's attacking strategy more accurately compared to the probabilistic model.

7.4 Performance Evaluation

To evaluate our proposed network hardening algorithms, we implemented both algorithms and tested them in two types of scenarios. We introduce the network topology in Section 7.4.1 and the derivation of the parameters in the attack graph of the evaluation in Section 7.4.2. We present evaluation results in Section 7.4.3.

7.4.1 Network Topologies

Following the attack graph generation strategy in [40], we generated 11 (unweighted) attack graphs from the devices and app configurations in IoT systems using `MulVAL`

[99]. The basic flow of the attack graph generation is as follows. First, we profile an IoT system by obtaining the details of IoT devices installed in the system, including brand, model, network protocol, and firmware version. We also verify if there are companion apps to these devices. We then query the CVE database [22] and gather any vulnerabilities reported about the installed devices in the IoT system. To build dependency relationships in the attack graph, we use NLP techniques to process the descriptions and automation rules in the devices' companion apps. We then combine the vulnerability information and the dependency information and use `MulVAL` to generate the attack graph for the system.

We studied 9 synthetic IoT systems based on real IoT devices and apps (systems with numbered labels) and 2 real-world IoT systems from our smart home testbeds (System A and System B). The IoT devices deployed in Systems A and B are listed in Table 7.1. In total, 11 system configurations were studied and converted to attack graphs. With an attack graph generated for a system, we then apply the approaches discussed in Section 7.4.2 to obtain weighted attack graphs with both `vul-only` parameters and `random` parameters for evaluation.

7.4.2 Evaluation Scenarios

We evaluated our algorithms on two types of scenarios:

(α) Only vulnerability vertices are hardenable. A vulnerability vertex is signified by the `vulExists` keyword in the vertex's description. Their weights w, costs c, and added strength δ are systematically derived from the relevant CVE information, detailed below. Under this scenario, we study a variation of the OHP problem. Instead of trying to maximize the height of the shortest attack trace (SAT),

Table 7.1. IoT Devices Deployed in Smart Home Testbeds

Device Name	Device Type	Protocols	Home
Amazon Smart Plug	Plug	WiFi	A
Amcrest ProHD	Camera	WiFi	A
Arlo Q Camera	Camera	WiFi	A
Arlo Ultra	Camera	WiFi	A
August Doorbell Cam Pro	Doorbell	WiFi	A
August Lock	Lock	WiFi	A
Blink XT2	Camera	WiFi	A
Chamberlain Garage Control	Sensor	WiFi	B
D-Link Water Sensor	Sensor	WiFi	A & B
Gosund WiFi Smart Socket	Plug	WiFi	A
Kangaroo Motion Sensor	Sensor	WiFi	A & B
Philips Hue	Bulb	WiFi & Zigbee	A
Reolink Camera	Camera	WiFi	A
Ring Doorbell	Doorbell	WiFi	A & B
Ring Spotlight	Spotlight	WiFi	A
Schlage WiFi Deadbolt	Lock	WiFi	A
Sengled SmartLED	Bulb	WiFi	A
Smart Life Contact Sensor	Sensor	WiFi	A & B
SmartThings Hub	Hub	WiFi & Zigbee	A
Tessan Contact Sensor	Sensor	WiFi	A & B
TP-Link Bulb	Bulb	WiFi	A & B
TP-Link Plug	Plug	WiFi	A & B
WeMo Plug	Plug	WiFi	A

we aim to make the system secure by patching out critical vulnerabilities while minimizing the total costs associated with patching the system. We call this type of parameter assignment the `vul-only` type.

(β) Hardenable network elements are chosen randomly. Initial weights, costs, and added strength for all network elements are randomly generated. For this scenario, we follow OHP problem defined in Equations (7.1)-(7.4) to obtain a hardening strategy X under budget $\mathbb{B}$, where the SAT for the hardened attack graph $\mathcal{H}(T_{sh}(X))$ is maximized. We designate this type of parameter assignment as the `random` type.

We now detail our approach to parameter assignment in both types of scenarios.

Hardenable elements ϕ: For scenario (α), we define $\phi[i] = 0$ for $i = |V| + 1, |V| + 2, \ldots, |V| + |E|$, and $\phi[i] = 1$ if and only if $\eta(i)$ is a vulnerability vertex in the attack graph G, $i = 1, 2, \ldots, |V|$. For scenario (β), we choose hardenable elements randomly. $K = \sum_{i=1}^{N} \phi[i]$ is the number of hardenable elements. For each attack graph, we evaluate the algorithms with $K = 16, K = 24$, and $K = 32$.

Weights w: For scenario (α), weights for vulnerability vertices are calculated based on the CVSS exploitability score of the CVE number associated with each vertex. Let $CVSS_{expl}(v)$ be this exploitability score, $w(v) = (\max(CVSS_{expl}) - CVSS_{expl}(v)) \times 2.5$, where $\max(CVSS_{expl})$ is 4, the maximum exploitability subscore in the CVSS 3.x scoring system. Note that the higher the exploitability score is, the easier it is to breach a vulnerable device. Thus, to represent the relative difficulty to breach the device, it is necessary to deduct $CVSS_{expl}(v)$ from the maximum exploitability score. We multiply the final result by 2.5 to scale it to the range $[0, 10]$ to be consistent with the the weights in scenario (β). For network elements that are not hardenable, we assign weights of zero to them. For scenario (β), weights for vertices and edges are real numbers randomly chosen within the range $[0, 10]$.

Added strength δ: In scenario (α), hardening a network element means that the vulnerability is eliminated from the network and the attacker can no longer utilize the vulnerability. Therefore, we define the added strength for a hardened vulnerability vertex v to be a very large number $\mathbb{H}$, i.e. $\delta(v) = \mathbb{H}$. We seek a minimum-cost hardening strategy so that the height of a shortest attack trace in the hardened attack graph is greater than or equal to $\mathbb{H}$. For scenario (β), $\delta(z)$ for each hardenable element z is randomized to be a real number in $[0.05, 2.00]$ times $w(z)$.

Hardening cost c: In scenario (α), because every vulnerability is resulted from a weakness, we correlate the origin of the vulnerability, i.e., the weakness, and examine how many identified vulnerabilities are affected by the same weakness. The hardening cost $c(v)$ for a vulnerability vertex v is then defined as the ratio between the vertex's CWE [87] score relative to the number of all CWEs as follows: $c(v) = \frac{R_{CWE}(v)}{T_{CWE}} \times 10$, where $R_{CWE}(v)$ is the rank of the CWE for vulnerability v and T_{CWE} is the total number of CWEs. The resulting cost will be in the range [0-10]. This cost represents the amount of effort that is needed to remediate a vulnerability.

For scenario (β), $c(z)$ for a hardenable element z is chosen to be a real number in $[0.30, 1.50]$ times $\delta(z)$. Note that for scenario (β), the generated parameters are not meant to represent realistic network configurations or actual effects of hardening, but rather to test whether our algorithms are effective for any general case of network hardening problems. Also for a given attack graph, we fix the values of w, c, and δ once they are generated, and we vary the hardenable network elements for each test case in scenario (β).

7.4.3 Evaluation Results

We present the evaluation results for attack graphs generated in both scenarios (and their associated hardening problems). All evaluation was done on a workstation with i9-12900 16-core CPU, 64GB system memory, and Ubuntu 22.04 system.

For the `vul-only` type of graphs, we follow the parameter assignment described in Section 7.4.2. There is only one test case per attack graph, since the vulnerability vertices in an attack graph and their CVSS assessments are fixed. For the `random` type of graphs, define ρ the percentage of the budget $\mathbb{B}$ over the cost of all hardenable elements, i.e., $\rho = \frac{\mathbb{B}}{\sum_{i=1}^{N} \phi[i] \times c[i]}$. We further create two kinds of test suites.

(i) The suite with a fixed budget percentage ρ, but different attack graphs: We fix ρ to 0.3 so that the budget $\mathbb{B}$ will be 0.3 times the total cost of all hardenable elements in an attack graph, and we create test cases with all 11 attack graphs. For each attack graph, we further create cases where $K = 16, 24$, and 32, respectively. For each K we generate 10 different test cases. Recall that the network parameters are set once they are generated, so each test case with the same attack graph differs only in their selection of hardenable network elements and budget $\mathbb{B}$.

(ii) The suite with a fixed attack graph, but different budget percentages ρ: We fix System 9 as the target attack graph, but vary ρ to be one of the following values: $[0, 15, 30, 50, 70, 85, 100]$. We generate cases with zero budget assigned all the way up to the cases where the budget is enough to harden all network elements. Like in (i), we create test cases where $K = 16, 24$, and 32, respectively. For each K, we also generate 10 different test cases.

For each test suite, we run both the exact algorithm (Algorithm 7.2) and the heuristic algorithm (Algorithm 7.5). Note that the exact algorithm will give the optimal solution X_{opt}, where the height of the attack trace $\mathcal{H}(T_{sh}(X_{opt}))$ is maximized given the budget $\mathbb{B}$.

7.4.3.1 Results for Scenario (α)

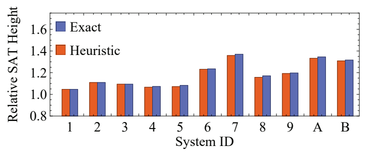
We apply the algorithms on `vul-only` graphs to find a hardening strategy X_{opt} that secures the system while having the lowest total cost. We say that a system is secure when the height of the SAT in the hardened attack graph is greater than or equal to $\mathbb{H}$. To ensure the hardening strategy X_{opt} has the minimum cost, we perform a bisection search on the budget $\mathbb{B}$ and find the minimal $\mathbb{B}$ that affords a strategy to secure the system.

Table 7.2 lists several key results from the experiments. We note that for all 11 systems, the procedure defined above was able to find a hardening strategy to secure the system within 0.01 seconds.

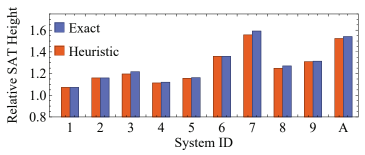
Table 7.2. Evaluation Results on Applying Network Hardening Techniques To Patch Vulnerabilities in the System

System ID	K	Running Time	Strategy Found?	Original Cost	Optimal Cost	Cost Saving
System 1	25	$0.0036s$	Yes	182.6	6.1	96.7%
System 2	17	$0.0009s$	Yes	127.7	8.2	93.6%
System 3	14	$0.0008s$	Yes	105.7	8.2	92.2%
System 4	9	$0.0006s$	Yes	74.5	19.8	73.4%
System 5	9	$0.0004s$	Yes	68.3	9.6	85.9%
System 6	4	$< 0.0001s$	Yes	31.2	9.5	69.6%
System 7	4	$< 0.0001s$	Yes	38.0	9.5	75.0%
System 8	4	$< 0.0001s$	Yes	28.3	5.5	80.6%
System 9	1	$< 0.0001s$	Yes	9.8	9.8	N/A
Testbed A	6	$< 0.0001s$	Yes	50.6	10.0	80.2%
Testbed B	2	$< 0.0001s$	Yes	15.1	6.6	56.3%

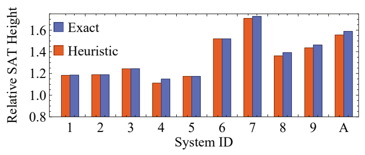
The results also reveal that in an attack graph, there can be a large number of hardenable elements (vulnerabilities), but not all of them affect the overall security of the system equally. In some cases, one can patch only a small subset of all vulnerable vertices and still be able to secure the system. We list Original Cost as the combined cost to harden all vulnerable vertices, and Optimal Cost as the optimized cost to secure the system after performing the cost optimization procedure. In all attack graphs in our test where $K > 1$, it is possible to cut down the costs to secure the system by a significant amount, resulting in as much as 96.7% savings in hardening cost as in the case of System 1.



(a) $K = 16$



(b) $K = 24$



(c) $K = 32$

Figure 7.2. Relative SAT Heights for the Exact and the Heuristic Algorithms Across Different Systems

Table 7.3. Evaluation Results of Our Exact and Heuristic Network Hardening Algorithm

System ID	n	m	Height	K	Budget	Heuristic Algorithm			Exact Algorithm			
						Cost	Height	Time	Cost	Height	Ratio	Time
System 1	338	577	124.83	16	23.20	5.78	130.60	0.0004s	17.12	130.60	21.70%	0.1121s
				24	36.34	9.88	133.90	0.0011s	32.08	133.90	12.03%	15.02s
				32	50.50	24.02	147.77	0.0020s	46.58	147.97	8.01%	2802s
System 2	209	310	77.94	16	21.14	11.58	86.46	0.0003s	17.50	86.46	21.21%	0.0527s
				24	34.01	19.90	90.38	0.0006s	27.49	90.38	8.86%	5.495s
				32	42.84	18.59	89.56	0.0010s	38.66	89.56	6.62%	1041s
System 3	162	234	79.06	16	22.04	7.55	86.52	0.0002s	18.56	86.52	17.44%	0.0328s
				24	28.40	15.73	94.57	0.0004s	25.63	96.25	12.54%	6.052s
				32	36.21	23.05	98.34	0.0007s	30.67	98.39	7.31%	873.7s
System 4	130	182	228.02	16	26.05	19.72	243.12	0.0002s	17.42	244.63	20.21%	0.0281s
				24	40.32	34.70	253.87	0.0004s	29.51	255.46	9.83%	3.571s
				32	51.44	43.28	257.55	0.0006s	46.68	260.47	5.21%	454.5s
System 5	117	173	110.29	16	27.91	12.55	118.20	0.0001s	18.86	119.40	25.68%	0.0338s
				24	47.07	22.78	127.49	0.0003s	38.07	128.20	15.95%	5.493s
				32	59.77	29.22	129.37	0.0005s	54.59	129.37	10.92%	900.9s
System 6	37	39	94.80	16	21.95	17.82	116.77	$<$ 0.0001s	17.40	117.07	8.72%	0.0028s
				24	33.31	28.47	128.84	0.0001s	29.03	128.90	3.60%	0.2950s
				32	44.68	41.18	114.07	0.0001s	41.59	144.07	1.81%	39.02s
System 7	36	35	72.11	16	24.19	21.59	97.96	$<$ 0.0001s	18.10	98.82	16.06%	0.0052s
				24	33.56	30.73	112.28	0.0001s	28.95	114.82	9.03%	0.7210s
				32	42.44	39.03	123.17	0.0002s	35.64	124.50	6.38%	34.40s
System 8	35	44	106.01	16	21.67	16.73	122.71	$<$ 0.0001s	17.88	124.12	27.07%	0.0076s
				24	31.31	28.33	128.84	0.0001s	26.18	134.68	12.22%	0.8846s
				32	44.78	40.79	145.04	0.0001s	41.71	147.30	7.32%	135.1s
System 9	25	26	165.84	16	24.25	22.82	197.94	$<$ 0.0001s	17.72	198.51	28.90%	0.0041s
				24	38.21	37.12	217.24	$<$ 0.0001s	31.28	217.95	22.78%	0.8392s
				32	52.42	51.69	238.24	0.0001s	50.96	242.62	16.78%	154.1s
Testbed A	37	42	61.91	16	15.22	13.38	82.55	$<$ 0.0001s	12.59	83.28	25.93%	0.0090s
				24	24.16	22.34	94.30	0.0001s	19.89	95.43	11.18%	0.9399s
				32	27.01	25.51	96.29	0.0002s	23.94	98.40	8.72%	12.00s
Testbed B	10	11	55.36	16	15.19	12.68	72.45	$<$ 0.0001s	12.71	72.87	34.38%	0.0026s
				21	18.87	17.16	80.75	$<$ 0.0001s	12.86	80.75	26.97%	0.0671s

7.4.3.2 Results for Scenario (β)-(i)

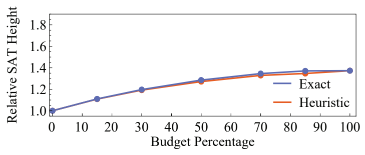
In Figure 7.2, we present the relative SAT heights for hardening strategies generated by both the exact and the heuristic algorithms. This illustrates the improvements of the heights of the SAT after running both algorithms and performing the hardening strategy returned by the algorithms. Note that for Testbed B, we have results for $K = 16$ and 21 only, as the system has only 21 network elements.

The exact algorithm is guaranteed to return an optimal hardening strategy X_{opt}

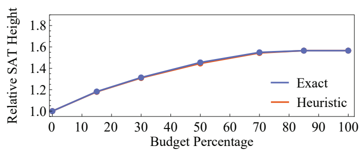
within the budget $\mathbb{B}$. Note that the optimal strategy is not necessarily unique, and the algorithm does not guarantee the strategy is of minimum cost. Due to the "bound" procedure in the algorithm, anytime when the residual budget is enough to harden all the remaining elements, the algorithm will do so if the resulting height of the SAT is better. This cuts down the running time but can result in more spending than necessary. This is the reason that the exact algorithm sometimes produces a more expensive strategy than the heuristic strategy, even if they achieve the same SAT height. The data shows that the algorithm exhibits an exponential running time with regard to K. However, due to the branch and bound strategy in the algorithm, the running time in general is much faster than a brute-force approach.

On the other hand, the heuristic algorithm achieves a comparable performance to the exact algorithm in many attack graphs. In some cases, the heuristic algorithm finds the same optimal solution that the exact algorithm finds. Averaging all test cases, the solution that the heuristic algorithm produces is approximately 96.81% as good as our exact algorithm produces. In other words, the heuristic strategy yields on average 96.81% of the increase in the height of SAT that an optimal strategy can do. In addition, the heuristic algorithm runs quickly even in the largest cases, finishing within 0.0002 seconds for all test cases.

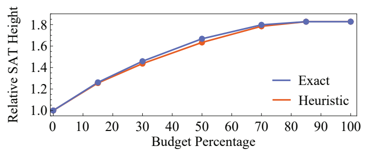
Table 7.3 lists the results in this test suite. For each system, we show the number of vertices and edges in the attack graph. The Height column indicates the height of the SAT in the graph before any hardening. For each algorithm, we show the cost of the hardening strategies along with the heights of the SAT in the hardened graph. We also list the time taken to execute one test case. For the exact algorithm, we added a ratio of the number of SAT queries made by the algorithm over that of the brute force solution (which is 2^K). Each (minor) row is averaged results over 10 test cases.



(a) $K = 16$



(b) $K = 24$



(c) $K = 32$

Figure 7.3. Relative SAT Heights for the Exact and the Heuristic Algorithms With Different Budget Percentages

7.4.3.3 Results for Scenario (β)-(ii)

We study also the behavior of the algorithms using the same attack graph (hence identical parameters) but different budget constraints. Figure 7.3 shows the relative heights of the SAT of both algorithms compared to that of the original graph when $K = 16, 24$, and 32. While reaffirming the results found in Section 7.4.3.2, the results also display that the difference between relative SAT heights is higher for medium budgets and tapers off towards the extremes.

Intuitively, when the budget is low, neither of the two algorithms can do much to harden the network. As the budget increases, the disparity between the two algorithms begins to manifest. However, once the budget is sufficiently large, the two algorithms can elect to harden most elements in the network, resulting again a similar performance.

The figures also suggest a diminishing return as more budget is added. Indeed, as the budget percentage ρ increases, the gains in SAT heights slow down, to a point where virtually no improvement is made between $\rho = 0.85$ and $\rho = 1.00$.

7.5 Conclusions

This study explores the problem of analyzing IoT system vulnerabilities and network hardening. We first design a novel algorithm for computing a shortest attack trace in a weighted attack graph. We demonstrate that our algorithm is more robust and faster than the state-of-the-art [40]. In particular, our algorithm can handle cycles in the attack graph properly, and works correctly regardless of whether the attack graph contains an attack trace or not. We then formulate the network hardening problem. We prove that the network hardening problem is NP-hard, and design an exact algorithm and a polynomial-time heuristic algorithm to solve it. Extensive evaluations show that our exact algorithm can compute optimal solutions for reasonably sized problems, and our heuristic algorithm can efficiently produce near optimal results.

REFERENCES

[1] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, “Peek-a-boo: I see your smart home activities, even encrypted!” In *Proc. of ACM WiSec*, 2020.

[2] B. Agarwal, R. Bhagwan, T. Das, S. Eswaran, V. N. Padmanabhan, and G. M. Voelker, “Netprints: Diagnosing home network misconfigurations using shared knowledge,” in *Proc. of USENIX NSDI*, 2009.

[3] N. Agmon, A. Shabtai, and R. Puzis, “Deployment optimization of IoT devices through attack graph analysis,” in *Proc. of ACM WiSec*, 2019.

[4] M. R. Alam, M. B. I. Reaz, and M. A. M. Ali, “A review of smart homes—past, present, and future,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 42, no. 6, pp. 1190–1203, 2012.

[5] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, “Sok: Security evaluation of home-based IoT deployments,” in *Proc. of IEEE S&P*, 2019.

[6] Amazon, *Alexa skills kit.* [Online]. Available: https://developer.amazon.com/alexa-skills-kit.

[7] P. Ammann, D. Wijesekera, and S. Kaushik, “Scalable, graph-based network vulnerability analysis,” in *Proc. of ACM CCS*, 2002.

[8] I. Angelakopoulos, G. Stringhini, and M. Egele, “Firmsolo: Enabling dynamic analysis of binary linux-based IoT kernel modules,” in *Proc. of USENIX Security*, 2023.

[9] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, “Understanding the mirai botnet,” in *Proc. of USENIX Security*, 2017.

[10] L. Bergroth, H. Hakonen, and T. Raita, “A survey of longest common subsequence algorithms,” in *Proc. of International Symposium on String Processing and Information Retrieval*, 2000.

[11] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, “Behavioral fingerprinting of IoT devices,” in *Proc. of ACM Workshop on Attacks and Solutions in Hardware Security*, 2018.

[12] V. Bhosale, L. De Carli, and I. Ray, “Detection of anomalous user activity for home IoT devices,” in *Proc. of IoTBDS*, 2021.

[13] S. Birnbach, S. Eberz, and I. Martinovic, “Peeves: Physical event verification in smart homes,” in *Proc. of ACM CCS*, 2019.

[14] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac, “Sensitive information tracking in commodity IoT,” in *Proc. of USENIX Security*, 2018.

[15] Z. B. Celik, E. Fernandes, E. Pauley, G. Tan, and P. McDaniel, “Program analysis of commodity IoT applications for security and privacy: Challenges and opportunities,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–30, 2019.

[16] Z. B. Celik, P. McDaniel, and G. Tan, “Soteria: Automated IoT safety and security analysis,” in *Proc. of USENIX ATC*, 2018.

[17] H. Chi, C. Fu, Q. Zeng, and X. Du, “Delay wreaks havoc on your smart home: Delay-based automation interference attacks,” in *Proc. of IEEE S&P*, 2022.

[18] H. Chi, Q. Zeng, and X. Du, “Detecting and handling IoT interaction threats in multi-platform multi-control-channel smart homes,” in *Proc. of USENIX Security*, 2023.

[19] B. Collins, *Ring video doorbells rescued by this great new feature*. [Online]. Available: https://www.forbes.com/sites/barrycollins/2020/11/29/ring-video-doorbells-rescued-by-this-great-new-feature/.

[20] A. A. Cook, G. Mısırlı, and Z. Fan, “Anomaly detection for IoT time-series data: A survey,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6481–6494, 2020.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2009.

[22] CVE, *Common vulnerabilities and exposures*. [Online]. Available: https://www.cve.org.

[23] J. Dahmen, D. J. Cook, X. Wang, and W. Honglei, “Smart secure homes: A survey of smart home technologies that sense, assess, and respond to security threats,” *Journal of Reliable Intelligent Environments*, vol. 3, no. 2, pp. 83–98, 2017.

[24] A. Davanian and M. Faloutsos, “Malnet: A binary-centric network-level profiling of IoT malware,” in *Proc. of ACM IMC*, 2022.

[25] P. De Vaere and A. Perrig, “Hey kimya, is my smart speaker spying on me? taking control of sensor privacy through isolation and amnesia,” in *Proc. of USENIX Security*, 2023.

[26] S. Demetriou, N. Zhang, Y. Lee, X. Wang, C. A. Gunter, X. Zhou, and M. Grace, “Hanguard: Sdn-driven protection of smart home WiFi devices from malicious mobile apps,” in *Proc. of ACM WiSec*, 2017.

[27] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley, “Optimal security hardening using multi-objective optimization on attack tree models of networks,” in *Proc. of ACM CCS*, 2007.

[28] A. Dhamdhere, D. D. Clark, A. Gamero-Garrido, M. Luckie, R. K. P. Mok, G. Akiwate, K. Gogia, V. Bajpai, A. C. Snoeren, and K. Claffy, “Inferring persistent interdomain congestion,” in *Proc. of ACM SIGCOMM*, 2018.

[29] L. DiCioccio, R. Teixeira, and C. Rosenberg, “Impact of home networks on end-to-end performance: Controlled experiments,” in *Proc. of ACM SIGCOMM Workshop on Home Networks*, 2010.

[30] W. Ding and H. Hu, “On the safety of IoT device physical interaction control,” in *Proc. of ACM CCS*, 2018.

[31] W. Ding, H. Hu, and L. Cheng, “Iotsafe: Enforcing safety and security policy with real IoT physical interaction discovery,” in *Proc. of NDSS*, 2021.

[32] J. Dunning, “Taming the blue beast: A survey of Bluetooth based threats,” *IEEE Security & Privacy*, vol. 8, no. 2, pp. 20–27, 2010.

[33] K. Durkota, V. Lisỳ, B. Bošanskỳ, and C. Kiekintveld, “Approximate solutions for attack graph games with imperfect information,” in *Proc. of International Conference on Decision and Game Theory for Security*, 2015.

[34] K. Durkota, V. Lisỳ, B. Bošanskỳ, and C. Kiekintveld, “Optimal network security hardening using attack graph games,” in *Proc. of IJCAI*, 2015.

[35] K. Durkota, V. Lisỳ, B. Bošanskỳ, C. Kiekintveld, and M. Pěchouček, “Hardening networks against strategic attackers using attack graph games,” *Computers & Security*, vol. 87, p. 101 578, 2019.

[36] P. Emami-Naeini, Y. Agarwal, L. F. Cranor, and H. Hibshi, "Ask the experts: What should be on an IoT privacy and security label?" In *Proc. of IEEE S&P*, 2020.

[37] P. Emami-Naeini, J. Dheenadhayalan, Y. Agarwal, and L. F. Cranor, "Are consumers willing to pay for security and privacy of IoT devices?" In *Proc. of USENIX Security*, 2023.

[38] P. Emami-Naeini, J. Dheenadhayalan, Y. Agarwal, and L. F. Cranor, "Which privacy and security attributes most impact consumers, risk perception and willingness to purchase IoT devices?" In *Proc. of IEEE S&P*, 2021.

[39] J. Fan, Y. He, B. Tang, Q. Li, and R. Sandhu, "Ruledger: Ensuring execution integrity in trigger-action IoT platforms," in *Proc. of IEEE INFOCOM*, 2021.

[40] Z. Fang, H. Fu, T. Gu, P. Hu, J. Song, T. Jaeger, and P. Mohapatra, "IOTA: A framework for analyzing system-level security of IoTs," in *Proc. of ACM/IEEE IoTDI*, 2022.

[41] X. Feng, R. Sun, X. Zhu, M. Xue, S. Wen, D. Liu, S. Nepal, and Y. Xiang, "Snipuzz: Black-box fuzzing of IoT firmware via message snippet inference," in *Proc. of ACM CCS*, 2021.

[42] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Proc. of IEEE S&P*, 2016.

[43] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging IoT application frameworks," in *Proc. of USENIX Security*, 2016.

[44] E. Fernandes, A. Rahmati, K. Eykholt, and A. Prakash, "Internet of things security research: A rehash of old ideas or new intellectual challenges?" *IEEE Security & Privacy*, vol. 15, no. 4, pp. 79–84, 2017.

[45] M. Fránik and M. Cermák, *Serious flaws found in multiple smart home hubs: Is your device among them?* [Online]. Available: https://www.welivesecurity.com/2020/04/22/serious-flaws-smart-home-hubs-is-your-device-among-them/.

[46] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[47] C. Fu, Q. Zeng, and X. Du, "Hawatcher: Semantics-aware anomaly detection for appified smart homes," in *Proc. of USENIX Security*, 2021.

[48] L. Fu, S. Ji, K. Lu, P. Liu, X. Zhang, Y. Duan, Z. Zhang, W. Chen, and Y. Wu, "Cpscan: Detecting bugs caused by code pruning in IoT kernels," in *Proc. of ACM CCS*, 2021.

[49] K. Gao, C. Corbett, and R. Beyah, "A passive approach to wireless device fingerprinting," in *Proc. of IEEE/IFIP International Conference on Dependable Systems and Networks*, 2010.

[50] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness.* W.H. Freeman and Company, 1979.

[51] Google, *Nest, create a connected home.* [Online]. Available: https://nest.com/.

[52] H. Griffioen and C. Doerr, "Examining mirai's battle over the internet of things," in *Proc. of ACM CCS*, 2020.

[53] T. Gu, Z. Fang, A. Abhishek, H. Fu, P. Hu, and P. Mohapatra, "Iotgaze: IoT security enforcement via wireless context analysis," in *Proc. of IEEE INFOCOM*, 2020.

[54] T. Gu and P. Mohapatra, "Bf-iot: Securing the IoT networks via fingerprinting-based device authentication," in *Proc. of IEEE MASS*, 2018.

[55] D. Han, A. Li, J. Li, Y. Zhang, T. Li, and Y. Zhang, "Dronekey: A drone-aided group-key generation scheme for large-scale IoT networks," in *Proc. of ACM CCS*, 2021.

[56] J. Haney and S. Furman, "User perceptions and experiences with smart home updates," in *Proc. of IEEE S&P*, 2023.

[57] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, "Measurement and analysis of hajime, a peer-to-peer IoT botnet," in *Proc. of NDSS*, 2019.

[58] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner, "Smart locks: Lessons for securing commodity internet of things devices," in *Proc. of ACM ASIACCS*, 2016.

[59] T. Høiland-Jørgensen, B. Ahlgren, P. Hurtig, and A. Brunstrom, "Measuring latency variation in the internet," in *Proc. of ACM CoNEXT*, 2016.

[60] B. Huang, A. A. Cardenas, and R. Baldick, "Not everything is dark and gloomy: Power grid protections against IoT demand attacks," in *Proc. of USENIX Security*, 2019.

[61] K. Huang, Y. Zhou, K. Zhang, J. Xu, J. Chen, D. Tang, and K. Zhang, "Homespy: The invisible sniffer of infrared remote control of smart tvs," in *Proc. of USENIX Security*, 2023.

[62] Y. Huang, W. Wang, H. Wang, T. Jiang, and Q. Zhang, "Authenticating on-body IoT devices: An adversarial learning approach," *IEEE Transactions on Wireless Communications (TWC)*, vol. 19, pp. 5234–5245, 2020.

[63] T. Islam and L. Wang, "A heuristic approach to minimum-cost network hardening using attack graph," in *New Technologies, Mobility and Security*, 2008.

[64] H. Jafari, O. Omotere, D. Adesina, H.-H. Wu, and L. Qian, "IoT devices fingerprinting using deep learning," in *Proc. of IEEE MILCOM*, 2018.

[65] Y. Jia, L. Xing, Y. Mao, D. Zhao, X. Wang, S. Zhao, and Y. Zhang, "Burglars' IoT paradise: Understanding and mitigating security risks of general messaging protocols on IoT clouds," in *Proc. of IEEE S&P*, 2020.

[66] Y. Jia, B. Yuan, L. Xing, D. Zhao, Y. Zhang, X. Wang, Y. Liu, K. Zheng, P. Crnjak, Y. Zhang, D. Zou, and H. Jin, "Who's in control? on security risks of disjointed IoT device management channels," in *Proc. of ACM CCS*, 2021.

[67] H. Jin, G. Liu, D. Hwang, S. Kumar, Y. Agarwal, and J. I. Hong, "Peekaboo: A hub-based approach to enable transparency in data processing within smart homes," in *Proc. of IEEE S&P*, 2022.

[68] X. Jin, S. Manandhar, K. Kafle, Z. Lin, and A. Nadkarni, "Understanding IoT security from a market-scale perspective," in *Proc. of ACM CCS*, 2022.

[69] Z. Jin, L. Xing, Y. Fang, Y. Jia, B. Yuan, and Q. Liu, "P-verifier: Understanding and mitigating security risks in cloud-based IoT access policies," in *Proc. of ACM CCS*, 2022.

[70] G. Kambourakis, C. Kolias, and A. Stavrou, "The mirai botnet and the IoT zombie armies," in *Proc. of IEEE MILCOM*, 2017.

[71] J. Kleinberg and E. Tardos, *Algorithm Design.* Pearson/Addison-Wesley, 2006.

[72] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: An analysis of IoT devices on home networks," in *Proc. of USENIX Security*, 2019.

[73] L. Kustosch, C. Gañán, M. van't Schip, M. van Eeten, and S. Parkin, "Measuring up to (reasonable) consumer expectations: Providing an empirical basis for holding IoT manufacturers legally responsible," in *Proc. of USENIX Security*, 2023.

[74] J. Li, K. Sun, B. S. Huff, A. M. Bierley, Y. Kim, F. Schaub, and K. Fawaz, "It's up to the consumer to be smart: Understanding the security and privacy attitudes of smart home users on reddit," in *Proc. of IEEE S&P*, 2023.

[75] T. Li, D. Han, J. Li, A. Li, Y. Zhang, R. Zhang, and Y. Zhang, "Your home is insecure: Practical attacks on wireless home alarm systems," in *Proc. of IEEE INFOCOM*, 2021.

[76] X. Li, Q. Zeng, L. Luo, and T. Luo, "T2pair: Secure and usable pairing for heterogeneous IoT devices," in *Proc. of ACM CCS*, 2020.

[77] C.-J. M. Liang, B. F. Karlsson, N. D. Lane, F. Zhao, J. Zhang, Z. Pan, Z. Li, and Y. Yu, "Sift: Building an internet of safe things," in *Proc. of ACM/IEEE Information Processing in Sensor Networks*, 2015.

[78] H. Liu, T. Spink, and P. Patras, "Uncovering security vulnerabilities in the belkin wemo home automation ecosystem," in *Proc. of IEEE PerCom Workshops*, 2019.

[79] X. Ma, J. Qu, J. Li, J. C. Lui, Z. Li, and X. Guan, "Pinpointing hidden IoT devices via spatial-temporal traffic fingerprinting," in *Proc. of IEEE INFOCOM*, 2020.

[80] D. Maier, "The complexity of some problems on subsequences and supersequences," *Journal of the ACM (JACM)*, vol. 25, no. 2, pp. 322–336, 1978.

[81] S. Manandhar, K. Kafle, B. Andow, K. Singh, and A. Nadkarni, "Smart home privacy policies demystified: A study of availability, content, and coverage," in *Proc. of USENIX Security*, 2022.

[82] S. Manandhar, K. Moran, K. Kafle, R. Tang, D. Poshyvanyk, and A. Nadkarni, "Towards a natural perspective of smart homes for practical security and safety analyses," in *Proc. of IEEE S&P*, 2020.

[83] A. M. Mandalari, H. Haddadi, D. J. Dubois, and D. Choffnes, "Protected or porous: A comparative analysis of threat detection capability of IoT safeguards," in *Proc. of IEEE S&P*, 2023.

[84] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "Profiliot: A machine learning approach for IoT device identification based on network traffic analysis," in *Proc. of ACM SAC*, 2017.

[85] Y. Meng, J. Li, M. Pillari, A. Deopujari, L. Brennan, H. Shamsie, H. Zhu, and Y. Tian, "Your microphone array retains your identity: A robust voice liveness detection system for smart speakers," in *Proc. of USENIX Security*, 2022.

[86] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *Proc. of IEEE ICDCS*, 2017.

[87] MITRE, *Cwe - common weakness enumeration.* [Online]. Available: https://cwe.mitre.org/.

[88] P. Moh, P. Datta, N. Warford, A. Bates, N. Malkin, and M. L. Mazurek, "Characterizing everyday misuse of smart home devices," in *Proc. of IEEE S&P*, 2023.

[89] P. Morgner, C. Mai, N. Koschate-Fischer, F. Freiling, and Z. Benenson, "Security update labels: Establishing economic incentives for security patching of IoT consumer products," in *Proc. of IEEE S&P*, 2020.

[90] P. Morgner, S. Mattejat, Z. Benenson, C. Müller, and F. Armknecht, "Insecure to the touch: Attacking ZigBee 3.0 via touchlink commissioning," in *Proc. of ACM WiSec*, 2017.

[91] A. Mosenia and N. K. Jha, "A comprehensive study of security of internet-of-things," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 586–602, 2016.

[92] Y. Nan, X. Wang, L. Xing, X. Liao, R. Wu, J. Wu, Y. Zhang, and X. Wang, "Are you spying on me? large-scale analysis on IoT data exposure through companion apps," in *Proc. of USENIX Security*, 2023.

[93] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. Colbert, and P. McDaniel, "Iotsan: Fortifying the safety of IoT systems," in *Proc. of ACM CoNEXT*, 2018.

[94] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "Dïot: A federated self-learning anomaly detection system for IoT," in *Proc. of IEEE ICDCS*, 2019.

[95] T.-A. Nguyen, J. He, L. T. Le, W. Bao, and N. H. Tran, "Federated pca on grassmann manifold for anomaly detection in IoT networks," in *Proc. of IEEE INFOCOM*, 2023.

[96] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, "Homesnitch: Behavior transparency and control for smart home IoT devicess," in *Proc. of ACM WiSec*, 2019.

[97] U. of Oregon, *Route views project.* [Online]. Available: http://www.routeviews.org/.

[98] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proc. of ACM CCS*, 2006.

[99] X. Ou, S. Govindavajhala, A. W. Appel, *et al.*, "Mulval: A logic-based network security analyzer," in *Proc. of USENIX Security*, 2005.

[100] M. O. Ozmen, X. Li, A. Chu, Z. B. Celik, B. Hoxha, and X. Zhang, "Discovering IoT physical channel vulnerabilities," in *Proc. of ACM CCS*, 2022.

[101] M. O. Ozmen, R. Song, H. Farrukh, and Z. B. Celik, "Evasion attacks and defenses on smart home physical event verification," in *Proc. of NDSS*, 2023.

[102] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: Analysing the rise of IoT compromises," in *Proc. of USENIX Workshop on Offensive Technologies*, 2015.

[103] M. T. Paracha, D. J. Dubois, N. Vallina-Rodriguez, and D. Choffnes, "Iotls: Understanding tls usage in consumer IoT devices," in *Proc. of ACM IMC*, 2021.

[104] K. Park and W. Willinger, *Self-Similar Network Traffic and Performance Evaluation.* John Wiley & Sons, 2002.

[105] V. Paxson and M. Allman, "Rfc2988: Computing tcp's retransmission timer," *Internet Engineering Task Force (IETF) Request for Comments*, 2000.

[106] L. Petzi, A. E. B. Yahya, A. Dmitrienko, G. Tsudik, T. Prantl, and S. Kounev, "Scraps: Scalable collective remote attestation for pub-sub IoT networks with untrusted proxy verifier," in *Proc. of USENIX Security*, 2022.

[107] P. Rashidi, D. J. Cook, L. B. Holder, and M. Schmitter-Edgecombe, "Discovering activities to recognize and rrack in a smart environment," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 23, no. 4, pp. 527–539, 2011.

[108] N. Redini, A. Continella, D. Das, G. De Pasquale, N. Spahn, A. Machiry, A. Bianchi, C. Kruegel, and G. Vigna, "Diane: Identifying fuzzing triggers in apps to generate under-constrained inputs for IoT devices," in *Proc. of IEEE S&P*, 2021.

[109] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach," in *Proc. of ACM IMC*, 2019.

[110] J. S. Richman and J. R. Moorman, "Physiological time-series analysis using approximate entropy and sample entropy," *American Journal of Physiology-Heart and Circulatory Physiology*, vol. 278, no. 6, H2039–H2049, 2000.

[111] P. Rieger, M. Chilese, R. Mohamed, M. Miettinen, H. Fereidooni, and A.-R. Sadeghi, "Argus: Context-based detection of stealthy IoT infiltration attacks," in *Proc. of USENIX Security*, 2023.

[112] J. Riihijarvi, M. Wellens, and P. Mahonen, "Measuring complexity and predictability in networks with multiscale entropy analysis," in *Proc. of IEEE INFOCOM*, 2009.

[113] E. Ronen and A. Shamir, "Extended functionality attacks on IoT devices: The case of smart lights," in *Proc. of IEEE EuroS&P*, 2016.

[114] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn, "IoT goes nuclear: Creating a ZigBee chain reaction," in *Proc. of IEEE S&P*, 2017.

[115] M. Ryan, *Bluetooth smart: The good, the bad, the ugly, and the fix.* [Online]. Available: https://lacklustre.net/bluetooth/bluetooth_smart_good_bad_ugly_fix-mikeryan-blackhat_2013.pdf.

[116] A. Sabur, A. Chowdhary, D. Huang, and A. Alshamrani, "Toward scalable graph-based security analysis for cloud networks," *Computer Networks*, vol. 206, p. 108 795, 2022.

[117] K. Sagonas, T. Swift, and D. S. Warren, "Xsb as a deductive database," in *Proc. of ACM SIGMOD*, 1994.

[118] S. J. Saidi, S. Matic, O. Gasser, G. Smaragdakis, and A. Feldmann, "Deep dive into the IoT backend ecosystem," in *Proc. of ACM IMC*, 2022.

[119] Samsung, *Smartthings, add a little smartness to your things.* [Online]. Available: https://www.smartthings.com/.

[120] R. A. Sharma, E. Soltanaghaei, A. Rowe, and V. Sekar, "Lumos: Identifying and localizing diverse hidden IoT devices in an unfamiliar environment," in *Proc. of USENIX Security*, 2022.

[121] T. Shekari, A. A. Cardenas, and R. Beyah, "Madiot 2.0: Modern high-wattage IoT botnet attacks and defenses," in *Proc. of USENIX Security*, 2022.

[122] T. Shekari, C. Irvene, A. A. Cardenas, and R. Beyah, "Mamiot: Manipulation of energy market leveraging high wattage IoT botnets," in *Proc. of ACM CCS*, 2021.

[123] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proc. of IEEE S&P*, 2002.

[124] X. Shi, S. Shi, M. Wang, J. Kaunisto, and C. Qian, "On-device IoT certificate revocation checking with small memory and low latency," in *Proc. of ACM CCS*, 2021.

[125] S. Siby, R. R. Maiti, and N. O. Tippenhauer, "Iotscanner: Detecting privacy threats in IoT neighborhoods," in *Proc. of ACM International Workshop on IoT Privacy, Trust, and Security*, 2017.

[126] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing (TMC)*, vol. 18, no. 8, pp. 1745–1759, 2018.

[127] V. Sivaraman, D. Chan, D. Earl, and R. Boreli, "Smart-phones attacking smart-homes," in *Proc. of ACM WiSec*, 2016.

[128] SmartHomeDB, *Smart home db - the smart home database.* [Online]. Available: https://www.smarthomedb.com/.

[129] S. Soltan, P. Mittal, and H. V. Poor, "Blackiot: IoT botnet of high wattage devices can disrupt the power grid," in *Proc. of USENIX Security*, 2018.

[130] N. Sombatruang, T. Caulfield, I. Becker, A. Fujita, T. Kasama, K. Nakao, and D. Inoue, "Internet service providers' and individuals' attitudes, barriers, and incentives to secure IoT," in *Proc. of USENIX Security*, 2023.

[131] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Open for hire: Attack trends and misconfiguration pitfalls of IoT devices," in *Proc. of ACM IMC*, 2021.

[132] S. Stephenson, M. Almansoori, P. Emami-Naeini, D. Y. Huang, and R. Chatterjee, "Abuse vectors: A framework for conceptualizing IoT-enabled interpersonal abuse," in *Proc. of USENIX Security*, 2023.

[133] E. Tekiner, A. Acar, and A. S. Uluagac, "A lightweight IoT cryptojacking detection mechanism in heterogeneous smart home networks," in *Proc. of NDSS*, 2022.

[134] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "Smartauth: User-centered authorization for the internet of things," in *Proc. of USENIX Security*, 2017.

[135] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Pingpong: Packet-level signatures for smart home device events," in *Proc. of NDSS*, 2019.

[136] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman, "Practical trigger-action programming in the smart home," in *Proc. of ACM CHI*, 2014.

[137] D. Valtchev and I. Frankov, "Service gateway architecture for a smart home," *IEEE Communications Magazine*, vol. 40, no. 4, pp. 126–132, 2002.

[138] S. Vetrivel, V. van Harten, C. H. Gañán, M. van Eeten, and S. Parkin, "Examining consumer reviews to understand security and privacy issues in the market of smart home devices," in *Proc. of USENIX Security*, 2023.

[139] A. Wang, A. Mohaisen, and S. Chen, "Xlf: A cross-layer framework to secure the internet of things (IoT)," in *Proc. of IEEE ICDCS*, 2019.

[140] F. Wang, J. Wu, Y. Nan, Y. Aafer, X. Zhang, D. Xu, and M. Payer, "Profactory: Improving IoT security via formalized protocol customization," in *Proc. of USENIX Security*, 2022.

[141] K. Wang, S. Xiao, X. Ji, C. Yan, C. Li, and W. Xu, "Volttack: Control IoT devices by manipulating power supply voltage," in *Proc. of IEEE S&P*, 2023.

[142] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, "An attack graph-based probabilistic security metric," in *Proc. of IFIP Annual Conference on Data and Applications Security and Privacy*, 2008.

[143] L. Wang, S. Noel, and S. Jajodia, "Minimum-cost network hardening using attack graphs," *Computer Communications*, vol. 29, no. 18, pp. 3812–3824, 2006.

[144] N. Wang, Y. Chen, Y. Hu, W. Lou, and Y. T. Hou, "Feco: Boosting intrusion detection capability in IoT networks via contrastive learning," in *Proc. of IEEE INFOCOM*, 2022.

[145] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *Proc. of NDSS*, 2018.

[146] S. Wang, Z. Zhang, and Y. Kadobayashi, "Exploring attack graph for cost-benefit security hardening: A probabilistic approach," *Computers & Security*, vol. 32, pp. 158–169, 2013.

[147] Z. Wang, Y. Yan, Y. Yan, H. Chen, and Z. Yang, "Camshield: Securing smart cameras through physical replication and isolation," in *Proc. of USENIX Security*, 2022.

[148] D. Wood, N. Apthorpe, and N. Feamster, "Cleartext data transmissions in consumer IoT medical devices," in *Proc. of ACM Workshop on IoT Privacy, Trust, and Security*, 2017.

[149] M. P. Woźniak, S. Vöge, R. Krüger, H. Müller, M. Koelle, and S. Boll, "Inhabiting interconnected spaces: How users shape and appropriate their smart home ecosystems," in *Proc. of ACM CHI*, 2023.

[150] L. Xiao, X. Wan, X. Lu, Y. Zhang, and D. Wu, "Iot security techniques based on machine learning: How do IoT devices use ai to enhance security?" *IEEE Signal Processing Magazine*, vol. 35, no. 5, pp. 41–49, 2018.

[151] K. Xu, X. Wang, W. Wei, H. Song, and B. Mao, "Toward software defined smart home," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 116–122, 2016.

[152] K. Yang, J. Ren, Y. Zhu, and W. Zhang, "Active learning for wireless IoT intrusion detection," *IEEE Wireless Communications*, vol. 25, no. 6, pp. 19–25, 2018.

[153] B. Yiğit, G. Gür, F. Alagöz, and B. Tellenbach, "Cost-aware securing of IoT systems using attack graphs," *Ad Hoc Networks*, vol. 86, pp. 23–35, 2019.

[154] L. Yu, B. Luo, J. Ma, Z. Zhou, and Q. Liu, "You are what you broadcast: Identification of mobile and IoT devices from (public) WiFi," in *Proc. of USENIX Security Symposium*, 2020.

[155] S. Zhang, W. Wang, S. Tang, S. Jin, and T. Jiang, "Robot-assisted backscatter localization for IoT applications," *IEEE Transactions on Wireless Communications (TWC)*, vol. 19, no. 9, pp. 5807–5818, 2020.

[156] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: Monitoring smart home apps from encrypted traffic," in *Proc. of ACM CCS*, 2018.

[157] X. Zhou, J. Guan, L. Xing, and Z. Qian, "Perils and mitigation of security risks of cooperation in mobile-as-a-gateway IoT," in *Proc. of ACM CCS*, 2022.

[158] H. Zhu, Y. Li, R. Li, J. Li, Z.-H. You, and H. Song, "Sedmdroid: An enhanced stacking ensemble of deep learning framework for android malware detection," *IEEE Transactions on Network Science and Engineering (TNSE)*, vol. 8, no. 2, pp. 984–994, 2020.

[159] T. Zillner, "ZigBee exploited: The good, the bad and the ugly," in *Proc. of the DeepSec Conferences in Depth Security*, 2017.

[160] H. Zou, M. Jin, H. Jiang, L. Xie, and C. J. Spanos, "Winips: WiFi-based non-intrusive indoor positioning system with online radio map construction and adaptation," *IEEE Transactions on Wireless Communications (TWC)*, vol. 16, no. 12, pp. 8118–8130, 2017.