Traditional Indicators of Compromise (IOCs) Meet Dynamic App-Device Interactions for IoT-specific Threat Intelligence

Sofya Smolyakova

A Thesis

in

The Department

of

Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Information Systems Security) at

Concordia University

Montréal, Québec, Canada

June 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Sofya Smolyakova**

Entitled: **Traditional Indicators of Compromise (IOCs) Meet Dynamic App-Device Interactions for IoT-specific Threat Intelligence**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Information Systems Security)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

______________________ Chair
Dr. Mohsen Ghafouri

______________________ External Examiner
Dr. Lianying Zhao

______________________ Examiner
Dr. Mohsen Ghafouri

______________________ Supervisor
Dr. Suryadipta Majumdar

Approved by ______________________
Dr. Chun Wang, Director
Department of Concordia Institute for Information Systems Engineering

__________ 2024

____
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Traditional Indicators of Compromise (IOCs) Meet Dynamic App-Device Interactions for IoT-specific Threat Intelligence

Sofya Smolyakova

While enjoying widespread popularity, IoT faces numerous threats with both traditional (e.g., Common Vulnerabilities and Exposures (CVEs) and Common Weakness Enumerations (CWEs)) and IoT-specific (e.g., device-application interactions) attack vectors. Therefore, gathering threat intelligence for an IoT environment is equally essential if not more (compared to many other IT environments). However, extracting threat intelligence from an IoT deployment poses several unique challenges. First, most IoT implementations are not logging threat-related information and even if they are, their logging mechanisms require significant additional effort to turn those logs to a threat intelligence. Second, there is no clear definition of IOCs (indicators of compromise), which are the key inputs to threat intelligence, in the context of IoT; including how to combine IoT-specific IOCs including that are involved with the dynamic app-device interactions.

In this thesis, we propose *IoTINT*, a solution to obtain IoT-specific threat intelligence while addressing the above-mentioned challenges. Specifically, our key ideas are to first enable logging in IoT devices and apps without requiring any code instrumentation (in contrast to existing approaches), then iteratively finding dynamic interactions between IoT devices and their apps that are defined by automation rules and result in various security threats, and finally, combine both app-device interactions with traditional IOCs (such as, CVEs and CWEs) to build a comprehensive threat intelligence for IoT. We implement IoTINT for Samsung SmartThings, a major smart home platform, and evaluate its performance (e.g., 100% coverage in extracting threat intelligence within 11 seconds for 10 realistic IoT attack scenarios).

Acknowledgments

I want to express my sincere gratitude to Dr. Suryadipta Majumdar, who has served as my supervisor and been a fantastic mentor to me throughout my graduate studies. His consistent encouragement, advice, and original ideas have improved my confidence and propelled me to the next level of competence. Dr. Majumdar was always open to new research ideas and welcomed them, which was crucial to the smooth development of this thesis. The financial and logistical support I have received from him is greatly appreciated. In a personal sense, he was kind, patient, and motivating.

I want to express gratitude to my family and friends for their unflagging love, support, and inspiration throughout this journey. Their encouragement has been my source of strength and motivation, propelling me forward throughout this journey. Without them, I could not have achieved this milestone.

I am grateful to be a member of the CISR Lab under the leadership of Dr. Majumdar. I express my gratitude to my talented, innovative, and hardworking colleagues at the CISR Lab for their support and collaboration throughout the different phases of this thesis. Their insightful recommendations and constructive feedback have consistently assisted me in raising the quality of my work.

Finally, I am also grateful to the department and university for providing me with the resources and opportunities that made this research possible. I appreciate your invaluable contributions to my academic and personal growth.

Contents

List of Figures

List of Tables

List of Abbreviations

CAPEC	Common Attack Pattern Enumeration
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumerations
CTI	Cyber Threat Intelligence
FAISS	Facebook AI Similarity Search
FN	False Negative
FP	False Positive
IOC	Indicator of Compromise
IoT	Internet of Things
LSTM	Long Short-Term Memory
NVD	National Vulnerability Database
NLP	Natural Language Processing
STIX	Structured Threat Information Expression
TTPs	Tactics, Techniques and Procedures

Chapter 1

Introduction

This chapter briefly describes the context, motivations, problem statements, related publications and contributions of this thesis.

1.1 Context

The usage of connected devices in smart environments (e.g., homes/offices, health facilities, factories and cities) follows an upward trend [1], opening doors for significant security threats and attacks in IoT, the number of which reached to 112 million in 2022 worldwide and keeps growing [2], [3]. Smart environments are typically managed by IoT platforms, such as Samsung SmartThings [4], AWS IoT Core [5], Google IoT Core [6], and openHAB [7]. These IoT platforms support the interconnection of diverse IoT devices via automations and assist in deploying customized smart applications that implement a wide range of user requirements. Thus, security attacks in IoT can follow various attack vectors encompassing traditional methods like vulnerability exploitation and IoT-specific approaches such as systems events and remote commands based on device-app interactions [8]. Therefore, threat intelligence gathering is critical to keep fresh knowledge about emerging attacks in IoT, define mitigating mechanisms and enhance security [9].

Existing research studies that gather IoT-specific threat intelligence (e.g., [10–14]) cover compromised IoT devices only from the network perspective, missing possible threats rooted in device-app interactions. These works extensively collect network traffic from compromised IoT devices using diverse methodologies such as telescopes, honeypots, machine learning, and deep learning. Subsequently, the collected traffic undergoes thorough analysis to extract various malware artifacts and attack patterns specific to these IoT devices, constituting IoT-specific threat intelligence. In addition, threat intelligence is usually produced from threat-related information or evidence, which is not always available in IoT. Other works on IoT (e.g., [15–17]) focus on providing raw logs from smart environments in an unstructured manner, and thus it is tedious to manually identify which log entries are threat-related. Also, traditional Cyber Threat Intelligence (CTI) studies (e.g., [18–23]) also focus on the network aspect and are not fully helpful for the IoT domain due to the limitations of IoT device's resources and the complexity of the smart environment.

Therefore, the IoT-specific challenges to obtain threat intelligence is unaddressed. Specifically, smart environments involve intricate interactions among IoT devices, applications, platforms, and users, which the traditional CTI cannot describe. Hence, there is no definition for Indicator of Compromise (IOCs), which are the major inputs to CTI, specific to the IoT context that could describe smart environment behavior. Furthermore, obtaining IoT-specific threat intelligence from the device-app interactions requires uncovering the artifacts that describe the smart environment behavior and establishing the chronological connectivity among various pieces of attack evidence. These limitations will be further illustrated in the following motivating example.

1.2 Motivating Example

Figure 1.1 shows our motivating example with a security problem (on the top), challenges to generate threat intelligence for smart environments using existing solutions (below) and our ideas to achieve the goal (on the bottom). Note that the scope of our work stays in the smart home while the problem and solution can be extended to multiple organizations, as mentioned in the motivating example.

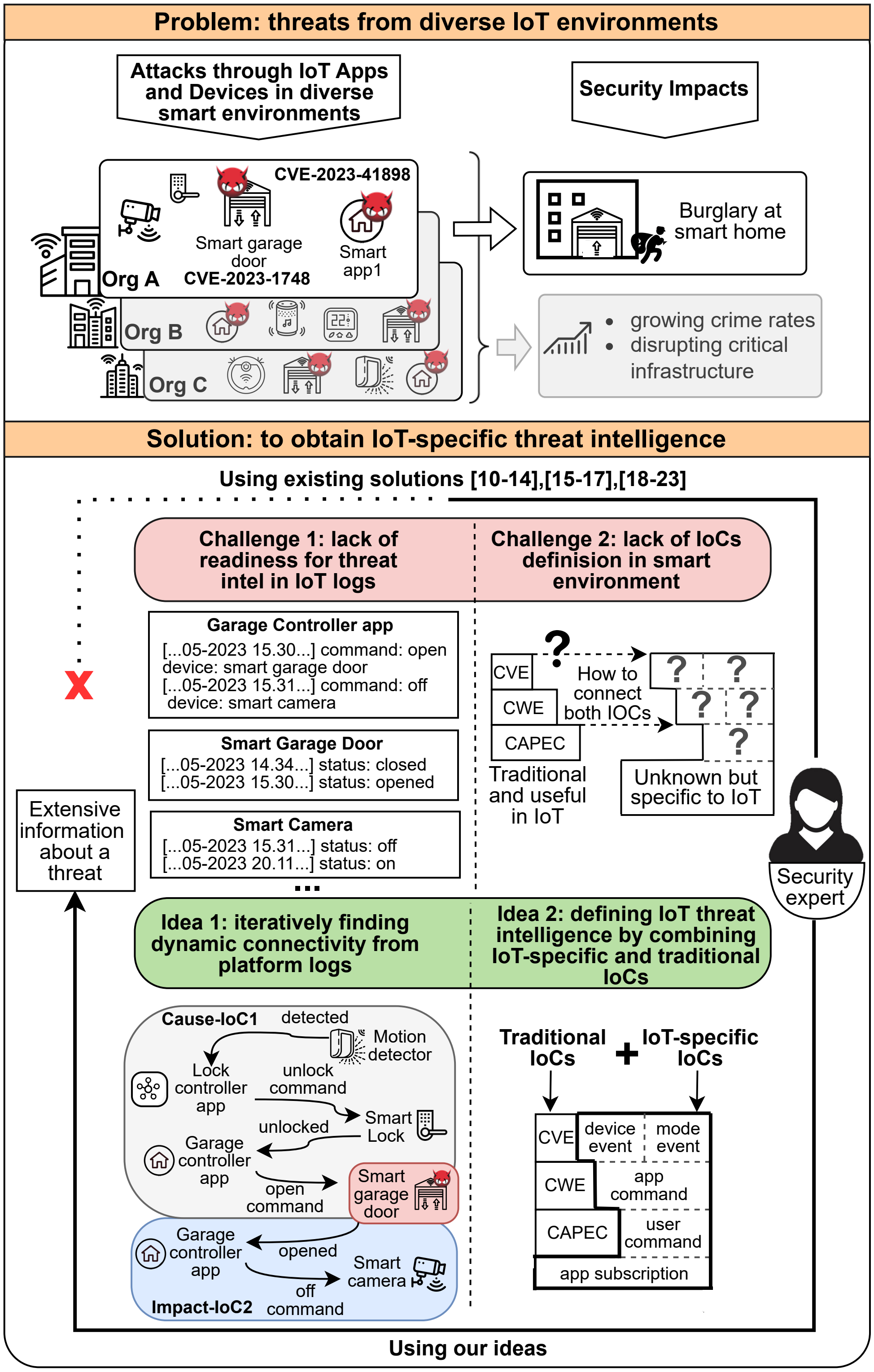
1.2.1 Problem

In a given scenario in an organization (`Org A`), there are a Smart garage door, Smart lock, Smart camera, and Smart app1 among others. While exploiting a vulnerability (CVE-2023-1748) in the `Smart garage door` and another vulnerability (CVE-2023-41898) in the smart application, `Smart app1`, an attacker unlocks the door without proper authorization and disables the exterior camera to remain stealthy [16]. Due to the popularity and cost-effectiveness of those products, other organizations (`Org B` and `Org C`) might use the same products and potentially face similar security and safety threats. Thus, while vulnerable IoT objects lead to small-scale threats such as burglary and intrusion within individual organizations, they have the potential to contribute to elevated crime rates in smart cities and, subsequently, impact the functionality of critical infrastructures. Therefore, it is essential to obtain intelligence about those emerging threats in `Org A, B, C`, etc. As the threat intelligence lifecycle includes many stages (e.g. direction, collection, processing, etc.), it is challenging to cover all these stages in one work. Thus, in this research, by IoT threat intelligence, we imply a combination of pieces of evidence related to the security incident in an IoT environment.

1.2.2 Current Challenges

The existing IoT threat intelligence solutions (e.g., [10–14]) focusing mainly on outbound traffic of the smart environment network, and hence the device and application level threats (as depicted above) remain undetected. In addition, the employment of traditional CTI approaches (e.g., [18–23]) in the IoT domain faces a set of challenges which hinder gathering threat information as follows.

- **Challenge 1:** The lack of readiness in IoT logs hinders their utilization in extracting IoT-specific threat intelligence. The implication of such a challenge is that a large volume of unprocessed entries requires considerable time and effort to identify incident-related records that could serve as IOCs manually. Specifically, Figure 1.1 presents log samples of the Garage Controller app, Smart garage door, and Smart camera that comprise such fields as timestamp, sent command, device, status, etc., indicating the device's status or the sent app command at a specific time. Thus, by observing these logs, it is impossible to identify which entries might be attack evidence and which device-app communications led to the incident. Furthermore, current methods are insufficient for addressing this challenge because of the lack of proper logging solutions. While some prior studies [15–17] propose using code instrumentation to capture the behavior of IoT


Problem: threats from diverse IoT environments
Attacks through IoT Apps and Devices in diverse smart environments
Security Impacts
CVE-2023-41898
Smart garage door
CVE-2023-1748
Smart app1
Org A
Org B
Org C
Burglary at smart home
growing crime rates
disrupting critical infrastructure
Solution: to obtain IoT-specific threat intelligence
Using existing solutions [10-14],[15-17],[18-23]
Challenge 1: lack of readiness for threat intel in IoT logs
Challenge 2: lack of IoCs definision in smart environment
Garage Controller app
Smart Garage Door
Smart Camera
...
CVE
CWE
CAPEC
How to connect both IOCs
?
Traditional and useful in IoT
Unknown but specific to IoT
Extensive information about a threat
X
Security expert
Idea 1: iteratively finding dynamic connectivity from platform logs
Idea 2: defining IoT threat intelligence by combining IoT-specific and traditional IoCs
Cause-IoC1
detected
Motion detector
Lock controller app
unlock command
Smart Lock
unlocked
Garage controller app
open command
Smart garage door
Garage controller app
opened
Smart camera
off command
Impact-IoC2
Traditional IoCs
+
IoT-specific IoCs
CVE
device event
mode event
CWE
app command
CAPEC
user command
app subscription
Using our ideas

Garage Controller app

```
[...05-2023 15.30...] command: open
device: smart garage door
[...05-2023 15.31...] command: off
 device: smart camera
```

Smart Garage Door

```
[...05-2023 14.34...] status: closed
[...05-2023 15.30...] status: opened
```

Smart Camera

```
[...05-2023 15.31...] status: off
[...05-2023 20.11...] status: on
```

Figure 1.1: Our motivating example demonstrates the necessity for IoT-specific threat intelligence due to the broad spectrum of threats in diverse IoT environments. Traditional solutions face challenges, prompting the need for a new approach, as hinted in our proposed ideas.

devices and applications, these approaches fall short due to smart apps running on third-party servers, which makes their codes inaccessible.

- **Challenge 2:** The lack of definition for IOCs in the IoT domain brings up questions such as:

  1) Considering that existing solutions rely on traditional IOCs for threat intelligence generation, which of these IOCs remain pertinent and effective for IoT threat intelligence?
  2) Given the scarcity of IOCs that characterize interactions among devices, applications, platforms, and users, what novel IoT-specific IOCs need to be introduced to provide insight into the smart environment behavior?
  3) How can traditional IOCs be combined with IoT-specific IOCs to provide holistic threat intelligence for the IoT domain?

  The implication of such a challenge is inaccurate detection of threats targeting IoT devices and applications behavior. Furthermore, the absence of an IOC definition for the IoT domain can lead to incomplete threat intelligence reports for security experts. Consequently, analyzing and understanding threats become challenging, as well as the ability to define effective countermeasures and mitigation strategies. The existing approaches [24–29] are insufficient to address this challenge mainly because they utilize traditional IOCs, as the source of the threat information is outbound network traffic. However, these works only cover artifacts collected about attacks, such as backdoors, DDos, injection, scanning, etc., and ignore the attacks emerging from device-app interactions in the IoT environment.

1.2.3 Our Ideas

To overcome those challenges, we propose:

- **Idea 1:** Tackling *Challenge 1*, where we first establish logging capabilities without relying on code instrumentation requirements. Additionally, we iteratively derive dynamic connectivity between devices and apps, transforming them into incident-related IOCs. The objective is to comprehend, for example, that the garage controller app initiated the garage door opening incident in response to the unlocked event, and the same app deactivated the camera as a result of the incident.

- **Idea 2:** To address *Challenge 2*, we formulate IoT threat intelligence. In this context, we define that traditional IOCs such as Common Vulnerabilities and Exposures (CVEs) [30], Common Weakness Enumerations (CWEs) [31], and Common Attack Pattern Enumeration and Classification (CAPECs) [32] offer valuable insights for threat intelligence in the IoT domain. Additionally, we introduce novel IoT-specific IOCs, encompassing elements like *device events*, *app commands*, *app subscriptions*, etc., to describe device-app behavior in the smart environment. Ultimately, we combine IoT-specific and traditional IOCs to better understand threats.

1.3 Thesis Contributions

This thesis complements existing IoT threat intelligence solutions as we propose a practical platform-centric approach for obtaining IoT-specific threat intelligence about threats implemented based on device-app interactions within a smart environment. First, our work introduces new, tailored, and specific to IoT IOCs. Second, we design our framework, namely, *IoTINT*, that enables logging in a smart environment, derives dynamic connectivity between IoT devices and apps from the logs, combines them with traditional IOCs (e.g., CVEs, CWEs) and generates various usable reports (e.g., machine-readable, human-readable). Third, we demonstrate the practicality of our approach to the different security contexts with two use cases. Finally, we evaluate our solution through extensive experiments based on realistic smart home scenarios and simulated attacks, comprising multiple smart applications and IoT devices.

The main contributions of this work are as follows:

- As per our knowledge, we are the first to design a practical framework for an IoT environment that obtains IoT-specific threat intelligence for various security incidents that allows threat intelligence extraction from IoT logs by deriving chronological connectivity between devices and apps from their interactions. Additionally, we combine derived IoT-specific IOCs with traditional IOCs to provide more insights into a threat.

- Our proposed approach is complementary to the existing IoT threat intelligence works by introducing new IoT-specific Indicators of Compromise (IOCs) that describe devices and apps' behavior in a smart environment (in contrast to the network-level IOCs from those other works).

- Using 10 different classes of real smart home attacks [16, 33–36], we show the ability of our tool to generate critical threat intelligence related to an incident in IoT, including information about both device and app vulnerabilities. In addition, we demonstrate the practicality of produced threat intelligence reports in two complementary use cases: (i) incident response for known threats and (ii) vulnerability assessment for unknown threats.

- We evaluate our solution's effectiveness on a dataset consisting of logs about smart home behavior generated using simulated IoT devices of SmartThings [37] (one of the most popular IoT platforms) where IoTINT shows 100% coverage in generating threat intelligence for smart homes. In addition, the efficiency and usability of IoTINT are evaluated. One of the results shows that the extraction of all IOCs relevant to the incident, whose number varies from 3 to 403 takes less than 20 seconds and spends not more than 110 MB of memory.

- The result of this thesis work has been accepted for publication at IEEE Internet of Things (IoT) Journal (with impact factor: 10.6).

1.4 Prior Publications

Journal Paper. Our work about obtaining IoT threat intelligence for various security incidents in the IoT environment has been accepted for publication in the IEEE Internet of Things Journal.

> **S. Smolyakova**, E. Khodayarseresht and S. Majumdar, ”Traditional IOCs Meet Dynamic App-Device Interactions for IoT-Specific Threat Intelligence,” in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2024.3413351. keywords: Internet of Things;Security;Smart homes;Faces;Cyber threat intelligence;Codes;Smart cameras;IoT security;threat intelligence;indicator of compromises,

Authors Contribution. Sofya Smolyakova contributed to the introduction, motivation, approach, methodology, implementation, experiments and case studies. Ehsan Khodayarseresh contributed to the defining mapping rules and obtaining ground truth to evaluate the coverage of IoTINT in gathering IOCs (i.e manual extraction of IOCs for attack scenarios A1-A10).

1.5 Organization of the Thesis

The rest of the thesis is organized as follows: Chapter 2 provides the necessary background knowledge to understand IoTINT and defines the threat model and its scopes. Chapter 3 describes the methodology and working principles of IoTINT. Chapter 4 describes the implementation details of IoTINT. Chapter 5 presents experimental results and performance evaluation of IoTINT. Chapter 6 presents the practicality of IoTINT through two case studies. Chapter 7 discusses various aspects of our approach and demonstrates the applicability, concerns and limitations. Chapter 8 reviews the existing works in the related domain, and compares them with our work. Chapter 9 concludes the thesis with the potential future research directions.

Chapter 2

Preliminaries

This section provides the necessary background, discusses existing challenges in IoT, and defines our threat model.

2.1 Background

This section provides the necessary backgrounds on indicators of compromise (IOCs) and smart home architecture.

2.1.1 Indicators of Compromise (IOCs)

The Indicator of Compromise (IOC) is a specific artifact, evidence or piece of forensic data that indicates that a system has faced or is potentially facing an attack or malicious activity [38] and is a crucial part of threat intelligence. Depending on the complexity and the level of detail in the data presentation, there are three types of IOCs [39]: *atomic*, *computed* and *behavioral*. The *atomic* IOCs are individual data fragments that point to an adversary activity and can not be broken into smaller parts; e.g., IP addresses, domains, URLs, and email addresses. The *computed* indicators are usually derived or calculated from the data involved in an incident; e.g., hash values of known malicious files. The *behavioral* indicators combine atomic and computed IOCs, offering a more comprehensive view of the various stages involved in an attack flow or malicious activity; e.g., MITRE tactics, techniques and procedures (TTPs) [40].

2.1.2 Smart Home Architecture

IoT platforms allow users to efficiently manage IoT devices, including sensors and actuators, by utilizing trigger-action rules (i.e., automation). SmartThings [37] is one of the most widely used and ubiquitous IoT platforms, offering support for a diverse array of hub-based and cloud-based IoT devices. Figure 2.1 shows how devices and apps interact in the SmartThings platform using an example. In this scenario where the app should turn on the light because of the motion detection, the following sequence of interactions ensues.

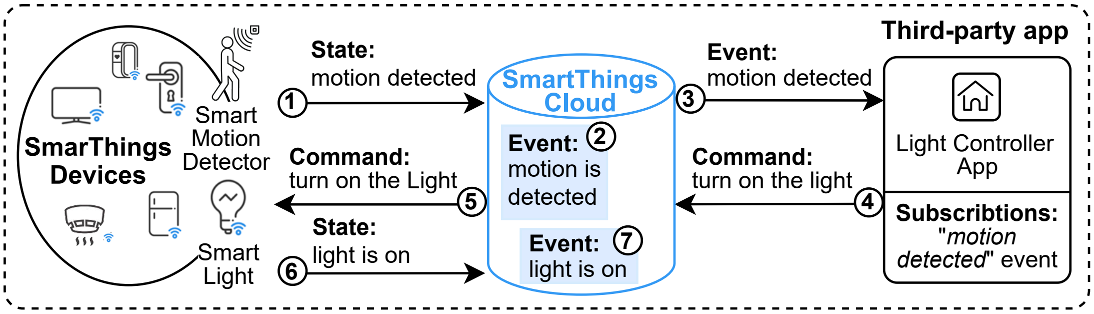


Figure 2.1: Device-app interactions in SmartThings [41].

1) The motion detector sends its state to the platform when the motion is detected.

2) The platform then posts a *motion detected* event.

3) Given that the smart application has subscribed to the *motion detected* event and is programmed to react upon its receipt, the platform transmits the event, signalling the occurrence of motion to the application.

4) After receiving the event, according to the app logic, it sends the command about turning the light *on* to the platform.

5) Subsequently, the platform transfers this command to the smart light device.

6) Further, when the light is *on*, it sends its state to the platform.

7) Finally, the *light is on* event is posted on the platform.

2.2 Challenges in Gathering Threat Intelligence from IoT Environments

Given that existing IoT threat intelligence and CTI solutions obtain traditional IOCs to produce threat reports, gathering threat intelligence within an IoT smart environment might face the following challenges.

- IoT devices often directly communicate with cloud services or gateways. Consequently, first, collecting the *atomic* traditional network-based IOCs, such as IP addresses, domains, URLs, etc., may be arduous or even infeasible. Second, the *computed* IOCs, which are typically derived or calculated through complex algorithms like anomaly detection or behavioral signatures, pose challenges for IoT devices due to their lightweight nature and limited processing power, storage, and memory. As a result, implementing complex IOC detection mechanisms may lead to potential performance issues or device failures. Lastly, the *behavioral* IOCs, which often present as stages of an attack flow, cannot be identified as there are no *atomic* or *computed* IOCs available that characterize the behavior of a smart environment. As a result, traditional IOCs mentioned above are not solely enough for gathering threat information about incidents in a smart environment.

Table 2.1: Smart environment terminologies

IOC type	**Type**	**Description**
Device Event	State	Shows the IoT device changed state.
App Command	Action	Aims to change the status of devices or locations.
App Subscription	Action	Allows a smart app to subscribe for different events.
User Command	Action	Aims to change the status of devices or locations as directed by users.
Mode Event	State	Shows a state change in a location mode (e.g., away or vacation).

- Device-app communication procedures in the IoT environment can be explained using specific terminologies presented in Table 2.1. If we can identify which device-app interactions led to the incident and occurred subsequently, these interactions become potential IOCs as they might constitute evidence related to the incident. Therefore, new IOCs representing *device events*, *app commands*, *app subscriptions*, etc., are associated with IoT-specific threats and offer deeper insights into security incidents. However, their extraction is non-trivial due to the absence of a direct mapping between device and app behavior in platform logs, and manual log analysis demands significant concentration and time. Consequently, retrieving IoT-specific IOCs relevant to the incident and identifying their chronological connectivity poses a significant challenge.

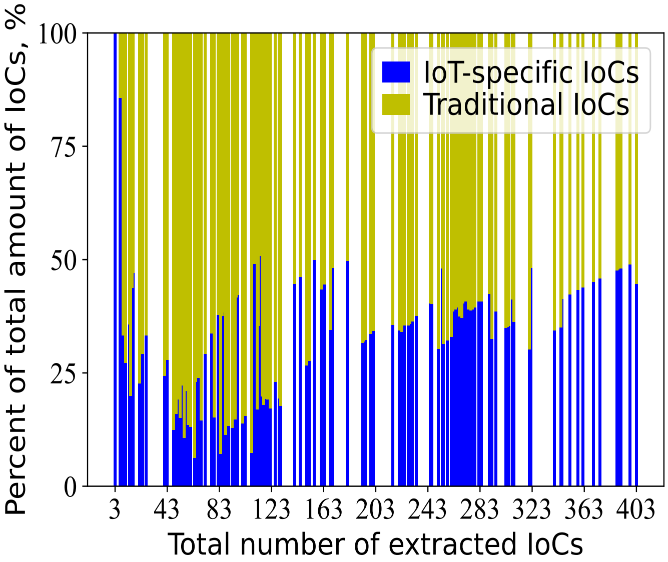


Figure 2.2: Distribution between IoT-specific and traditional IOCs in the obtained threat intelligence for various security incidents

- Furthermore, our preliminary study (as shown in Figure 2.2) identifies that most IoT security incidents involve both traditional and IoT-specific IOCs. Specifically, the graph depicts the percentage distribution between IoT-specific and traditional IOCs in the generated threat intelligence for various security incidents, with the total number of extracted IOCs ranging from 3 to 403, as indicated on the X-axis. Overall, IoT-specific IOCs contribute to approximately 50% of the total count when threat intelligence report size exceeds 150. In contrast, reports with fewer extracted indicators contain around 20-30% of IoT-specific IOCs. These statistics underscore the substantial role of IoT-specific IOCs in threat intelligence, highlighting that their exclusion may result in missing critical threat information.

We address these challenges in Chapter 3.

2.3 Classes of IOCs

Figure 2.3 demonstrates an example of incident-related threat intelligence in a graph format showcasing newly introduced IOCs and their interconnections. Within this scenario, Smart apps 1 and 2 subscribed and received the undetected smoke event. This triggered Smart app 1 to unlock the smart lock device. In contrast, Smart app 2 sent siren and strobe commands to the smart alarm. Moreover, Smart app 3 received the smart alarm siren event, which resulted in the opened window, which deviates from expected smart home behavior. *App command* IOCs highlighted with red color in Figure 2.3 represent malicious activity, as in a normal scenario, when no smoke is detected, the alarm devices should be in the off state. Thus, we classify individual nodes that point to an

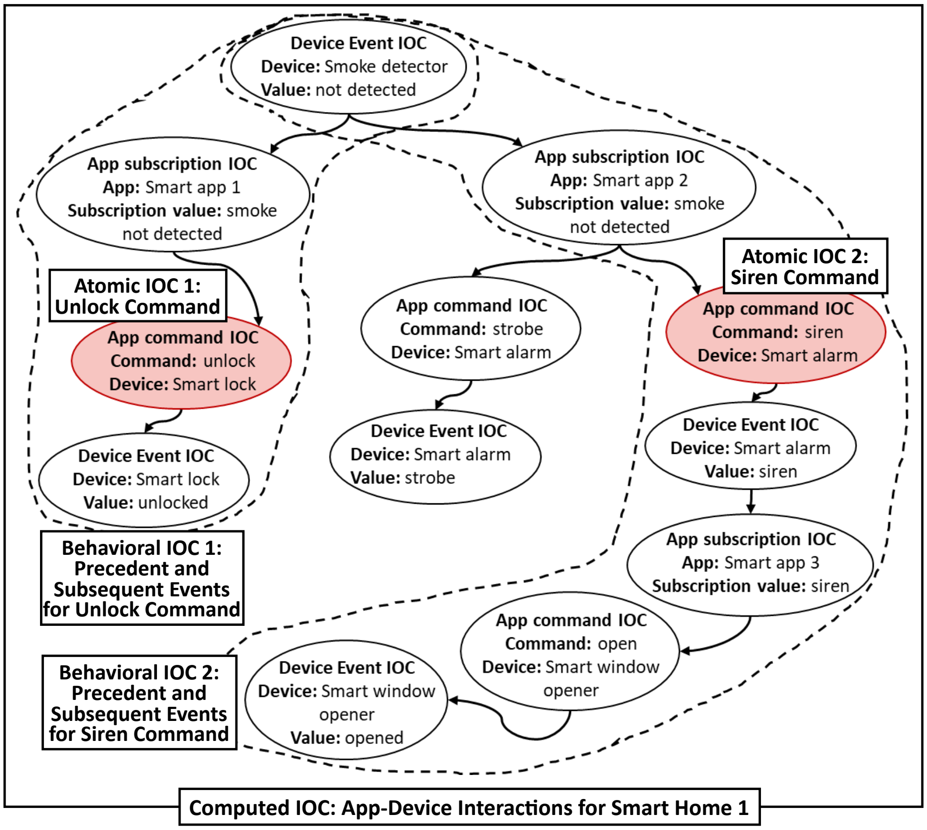


Figure 2.3: IoT-specific IOCs classification.

adversary activity as an *atomic* IOC. Specifically, in the example scenario, *atomic* IOC 1 is an unlock command, while *atomic* IOC 2 is a siren command. Furthermore, *behavioral* IOCs usually combine *atomic* and *computed* IOCs and define an attack pattern or various stages of malicious activity. Thus, we classify *atomic* IOC, together with its preceding and subsequent events as a *behavioral* IOC as it presents a smart home behavior pattern including nodes before and after the malicious activity. Figure 2.3 illustrates two branches outlined with dashed lines representing the *behavioral* IOCs 1 and 2. Finally, we categorize the complete graph outlining device-app interactions pertaining to a specific incident as a *computed* IOC. This IOC may encompass *atomic* and *behavioral* IOCs, necessitating multiple steps for its generation, as elaborated in Section 3.3.

2.4 Threat Model and Assumptions

In the context of IoT environments, security breaches can occur through various means, such as taking advantage of vulnerabilities within the IoT platform, smart applications, or the devices themselves. This research primarily relies on data recorded by the IoT platform for the purpose of threat intelligence generation. If certain platform logs are incomplete or not present, IoTINT is not focused to detect them and hence it might lead to inaccurate/incomplete threat intelligence generation. In this work, we consider, the main threats that lead to the security incidents inside the smart environment are as follows [42].

i) *Malformed/compromised smart apps:* Third-party apps can be installed on the IoT platforms, which can acquire unnecessary extra privileges during the installation time to perform undesired actions. Note that installation of smart apps does not necessarily involve downloading Android/iOS apps locally (e.g., accessible from a web browser). Instead, they run on third-party servers and can subscribe to platform events, which allows them to gain control over connected IoT devices.

ii) *Vulnerable/malformed devices:* We also include the threats from the devices with hardware or firmware vulnerabilities that attackers might exploit.

For our experiments in this thesis, due to the availability of a particular dataset, we mainly consider the scenarios with the malformed/vulnerable smart apps.

Chapter 3

Methodology

This chapter first gives an overview of how IoTINT works and then describes the major steps with a set of running examples.

3.1 Approach Overview

Figure 3.1 illustrates a high-level overview of the IoTINT approach in four major steps. First, to enable logging without code instrumentation (to overcome the limitations of existing works, e.g., [17]) and use them for the purpose of obtaining threat intelligence, IoTINT collects raw data from devices and apps through the platform, categorizes device/app-specific logs from raw data and constructs the database of logs and mapping rules (detailed in Section 3.2). Second, to identify interactions between devices and apps during the reported security incident, IoTINT initiates an iterative connectivity session that utilizes the database of logs and the mapping rules to extract all relevant IoT-specific IOCs, both preceding and subsequent to the incident (detailed in Section 3.3). Third, to provide more comprehensive insight into understanding the incident, IoTINT combines the IoT-related IOCs acquired in the earlier step with the traditional IOCs, including the CVEs, CWEs, and CAPECs related to the incident (detailed in Section 3.4). Finally, to enable various use cases, IoTINT produces threat intelligence reports in various formats (detailed in Section 3.5) and then applies graph-based reports suitable for manual inspection for incident response, and machine-readable reports for automated vulnerability assessment (Chapter 6).

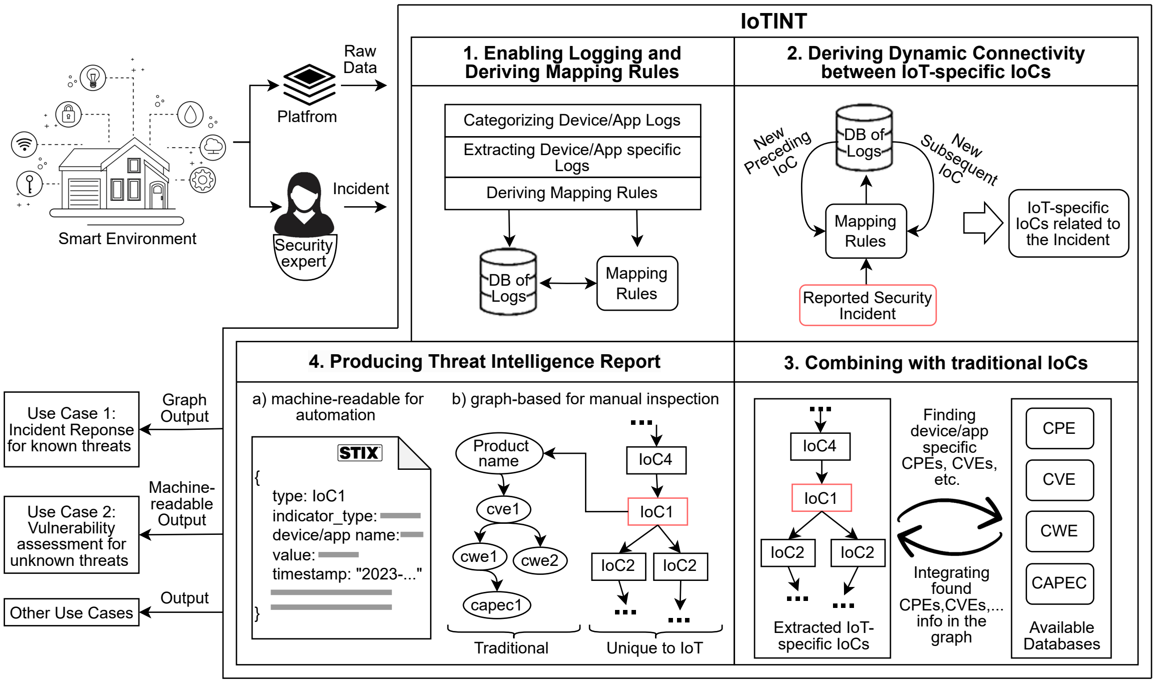

IoTINT
Smart Environment
Platfrom
Raw Data
Incident
Security expert
1. Enabling Logging and Deriving Mapping Rules
Categorizing Device/App Logs
Extracting Device/App specific Logs
Deriving Mapping Rules
DB of Logs
Mapping Rules
2. Deriving Dynamic Connectivity between IoT-specific IoCs
New Preceding IoC
DB of Logs
New Subsequent IoC
Mapping Rules
Reported Security Incident
IoT-specific IoCs related to the Incident
3. Combining with traditional IoCs
IoC4
IoC1
IoC2
IoC2
Extracted IoT-specific IoCs
Finding device/app specific CPEs, CVEs, etc.
Integrating found CPEs,CVEs,... info in the graph
CPE
CVE
CWE
CAPEC
Available Databases
4. Producing Threat Intelligence Report
a) machine-readable for automation
STIX
{
type: IoC1
indicator_type:
device/app name:
value:
timestamp: "2023-..."
}
b) graph-based for manual inspection
Product name
cve1
cwe1
cwe2
capec1
IoC4
IoC1
IoC2
IoC2
Traditional
Unique to IoT
Graph Output
Use Case 1: Incident Reponse for known threats
Machine-readable Output
Use Case 2: Vulnerability assessment for unknown threats
Output
Other Use Cases

Figure 3.1: An approach overview of IoTINT.

3.2 Enabling Logging and Defining Mapping Rules

This section discusses the logging of smart environment behavior by IoTINT along with defining mapping rules by authors.

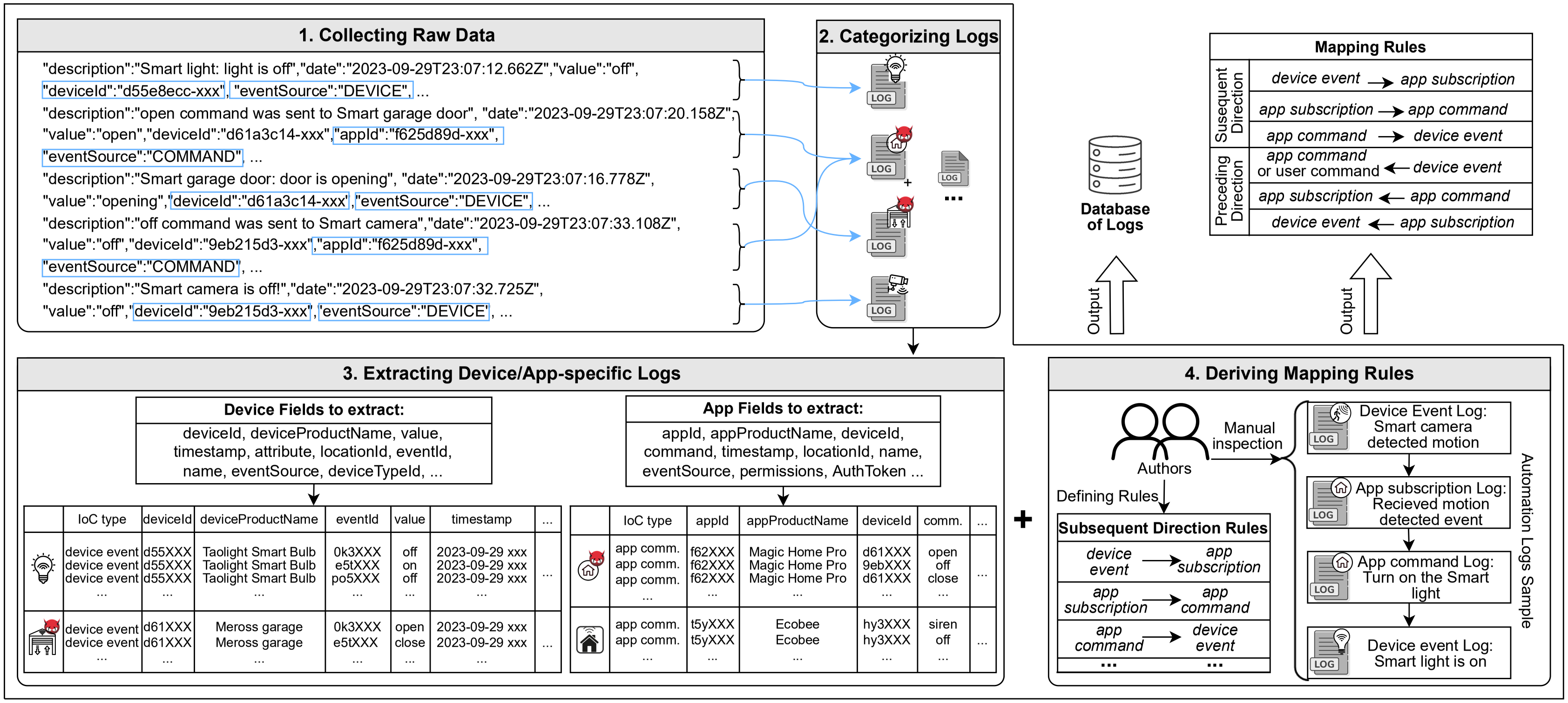
3.2.1 Enabling Logging

This step is to allow the logging of threat-related information, such as device-app interactions and to prepare raw data to become a future threat intelligence. Specifically, as a first step, we enable logging to source raw data from the IoT platform, encompassing interactions among devices, applications, and users without using code instrumentation. While code instrumentation could capture smart environment activities and their correlations (e.g., smart app activating light in response to a motion event), we target to log just raw data of events recorded by the IoT platform and interactions between third-party apps and the platform, as their connectivity is established in further steps of the methodology. In order to collect data, IoT platforms must employ monitoring and management solutions, such as Amazon CloudWatch, to capture and store JSON-formatted data for published device events and the transferred data between the IoT platform and the connected smart apps for each user's account. Within our implementation on the SmartThings platform, we establish a proxy channel to gather data between the SmartThings cloud platform and the server hosting the smart apps. This channel intercepts all network traffic between the components and monitors the exchanged data. Additionally, to capture event logs from the platform, we establish a WebSocket connection directly with the platform via the user account. Since the raw data is unstructured and comprises various fields, it necessitates preparation for utilization by other modules within IoTINT. Thus, as a second step, IoTINT classifies raw logs for each IoT device or smart app based on deviceIds, appIds, eventSources and other attributes. Subsequently, we reduce the size of stored logs by leveraging predefined lists of fields, as not all the log attributes might be needed to produce threat intelligence. Lastly, preprocessed potential attack-related evidences are saved in the database.

3.2.2 Defining Mapping Rules

To define mapping rules that are for identifying evidence pertinent to a security incident, we adopt a manual effort due to the variability of mapping specifics across different platforms. Thus, we examined log samples from a specific automation scenario within a smart home environment to define the mapping rules that construe the connectivity of app-device interactions in preceding and subsequent directions. Note that this manual step of deriving mapping rules only recur if IoTINT is ported to other IoT platforms (than the SmartThings platform, for instance).

Example 1. Figure 3.2 illustrates the enabling logging step for a smart home scenario, mirroring our motivating example, in four major steps. In Step 1, as a result of data collection from `Smart light`, `Smart garage door`, `Smart camera`, and `Smart app1`, the IoTINT receives the device and app events as raw inputs. In Step 2, leveraging key details highlighted in blue, such as `event source`, `device ID`, and `application ID`, IoTINT classifies logs for corresponding devices and apps. In Step 3, IoTINT extracts the `deviceID`, `eventID`, `value` and `timestamp` for the `Smart light` and `appID`, `deviceID` and `command attributes` for the malformed `Smart app1`, as predefined by the device and app filters to transform logs into a form of potential IOCs. In Step 4, we show mapping rules for a simple automation rule involving `Smart camera`, `Smart light`, and `Smart app1`. In this scenario, when the camera detects motion, the app receives this event and triggers a command to activate the light. We collect the



```
"description":"Smart light: light is off","date":"2023-09-29T23:07:12.662Z","value":"off",
"deviceId":"d55e8ecc-xxx","eventSource":"DEVICE", ...
"description":"open command was sent to Smart garage door", "date":"2023-09-29T23:07:20.158Z",
"value":"open","deviceId":"d61a3c14-xxx","appId":"f625d89d-xxx",
"eventSource":"COMMAND", ...
"description":"Smart garage door: door is opening", "date":"2023-09-29T23:07:16.778Z",
"value":"opening","deviceId":"d61a3c14-xxx","eventSource":"DEVICE", ...
"description":"off command was sent to Smart camera","date":"2023-09-29T23:07:33.108Z",
"value":"off","deviceId":"9eb215d3-xxx","appId":"f625d89d-xxx",
"eventSource":"COMMAND", ...
"description":"Smart camera is off!","date":"2023-09-29T23:07:32.725Z",
"value":"off","deviceId":"9eb215d3-xxx","eventSource":"DEVICE", ...
```

Device Fields to extract:
deviceId, deviceProductName, value, timestamp, attribute, locationId, eventId, name, eventSource, deviceTypeId, ...

	IoC type	deviceId	deviceProductName	eventId	value	timestamp	...
	device event device event device event ...	d55XXX d55XXX d55XXX ...	Taolight Smart Bulb Taolight Smart Bulb Taolight Smart Bulb ...	0k3XXX e5tXXX po5XXX ...	off on off ...	2023-09-29 xxx 2023-09-29 xxx 2023-09-29 xxx ...	...
	device event device event ...	d61XXX d61XXX ...	Meross garage Meross garage ...	0k3XXX e5tXXX ...	open close ...	2023-09-29 xxx 2023-09-29 xxx ...	...

App Fields to extract:
appId, appProductName, deviceId, command, timestamp, locationId, name, eventSource, permissions, AuthToken ...

	IoC type	appId	appProductName	deviceId	comm.	...
	app comm. app comm. app comm. ...	f62XXX f62XXX f62XXX ...	Magic Home Pro Magic Home Pro Magic Home Pro ...	d61XXX 9ebXXX d61XXX ...	open off close ...	...
	app comm. app comm. ...	t5yXXX t5yXXX ...	Ecobee Ecobee ...	hy3XXX hy3XXX ...	siren off ...	...

Subsequent Direction Rules

device event → *app subscription*
app subscription → *app command*
app command → *device event*
...

Device Event Log: Smart camera detected motion
→ App subscription Log: Recieved motion detected event
→ App command Log: Turn on the Smart light
→ Device event Log: Smart light is on

Mapping Rules

Susequent Direction	*device event* → *app subscription*
	app subscription → *app command*
	app command → *device event*
Preceding Direction	*app command* or user command ← *device event*
	app subscription ← *app command*
	device event ← *app subscription*

Figure 3.2: An example showing the steps of enabling logging in a smart environment.

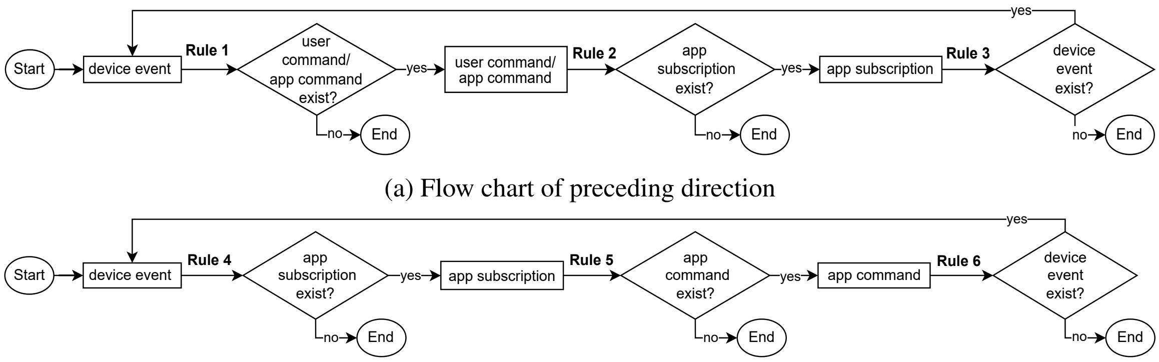
logs after the automation is executed and manually inspect each type of log, such as *device event* about motion being detected by `Smart camera`, *app subscription* about `Smart app1` receiving the event, *app command* about `Smart app1` sending a command to turn on the `Smart light`, and *device event* about `Smart light` being turned on, to define the rules that connect these logs chronologically. Specifically, for subsequent direction, we define how to map the `device event` to the `app subscription`, the `app subscription` to an `app command`, and the `app command` to the `device event`. Consequently, following the opposite chronological order, we define the mapping rules for a preceding direction that connects the `device event` to the `app command` or `user command`, the `app command` to the `app subscription` and the `app subscription` to the `device event`.

3.3 Deriving Dynamic Connectivity between IoT-specific IOCs

This step is to derive IoT-specific IOCs related to the incident and establish their connectivity to offer insights into the smart environment's behavior. This step addresses the challenge of laborious manual extraction of incident-related artifacts due to many concurrent irrelevant events that appeared at the time of the incident as well as the dynamic nature of those relevant connectivities that are mainly defined by the automation rules.

First, we receive the incident description (e.g., garage door was opened at night) as input from the user or the security expert and aim to pinpoint the log record in the database that closely aligns with the observed scenario. Thus, the found log record becomes the first IOC related to an incident and a starting point to find other related IOCs. Further, IoTINT proceeds to retrieve all IoT-specific IOCs connected to this input IOC in both preceding and subsequent directions. For this step, IoTINT utilizes the mapping rules (e.g., described below as Rules 1 and 4) and the database of logs that are identified in the previous step. To provide further elaboration, the preceding direction aims to identify all pertinent IOCs that occurred chronologically before an identified initial IOC. Conversely, the subsequent direction seeks to unearth all pertinent IOCs that transpired chronologically after the initial IOC.

Upon examining the architecture and behavior of the platform, we noted a consistent sequence of IOCs appearance in a smart environment: user command → device event → app subscription → app command → device event → repeat. This sequence serves as a subsequent direction for IOCs search. Conversely, a reversed sequence of IOCs would be followed if IoTINT is searching for IOCs in a preceding direction. These sequences are shown in Figure 3.3. Specifically, Figure 3.3a illustrates the flow chart that IoTINT follows while conducting a preceding direction search. In this process, the starting IOC is a *device event* provided as input, which may result from either *user command* or *app command* according to the observed sequence. Thus, by leveraging the initial IOC data to populate Rule 1, IoTINT constructs and executes a query to the database, retrieving a new IOC relevant to the incident. Subsequently, if the new IOC was found, IoTINT analogously uses Rule 2 to identify the relevant *app subscription* IOC and Rule 3 to identify the *device event* IOC. Note that if no IOC is found, IoTINT stops the search, indicating that no more relevant to the incident IOCs in this direction exist. Similarly, Figure 3.3b presents the flow chart followed by IoTINT while conducting a subsequent direction search. IoTINT utilizes Rule 4 to retrieve the *app subscription* IOC that happened chronologically after the *device event* IOC. Analogously, Rule 5 and Rule 6 are used to find relevant *app command* and *device event* IOCs.



(a) Flow chart of preceding direction

(b) Flow chart of subsequent direction

Figure 3.3: Flow charts that IoTINT follows while deriving connectivity between IoT-specific IOCs

In the following, we show how IoTINT represents mapping rules to be used for this step. Due to the space constraint, we only show two out of six rules. Specifically, Rule 1 corresponds to the rule that describes the mapping between *device event* IOC and *app command* or *user command* IOC. Rule 4 corresponds to the rule that maps the *app subscription* IOC to the *device event* IOC.

Within Rule 1, the variable IOC_1 represents the *device event*, the variable IOC_2 represents *app* *command* or *user command* IOCs and IOC_i represents a log entry from the log collection, $IOCs$.

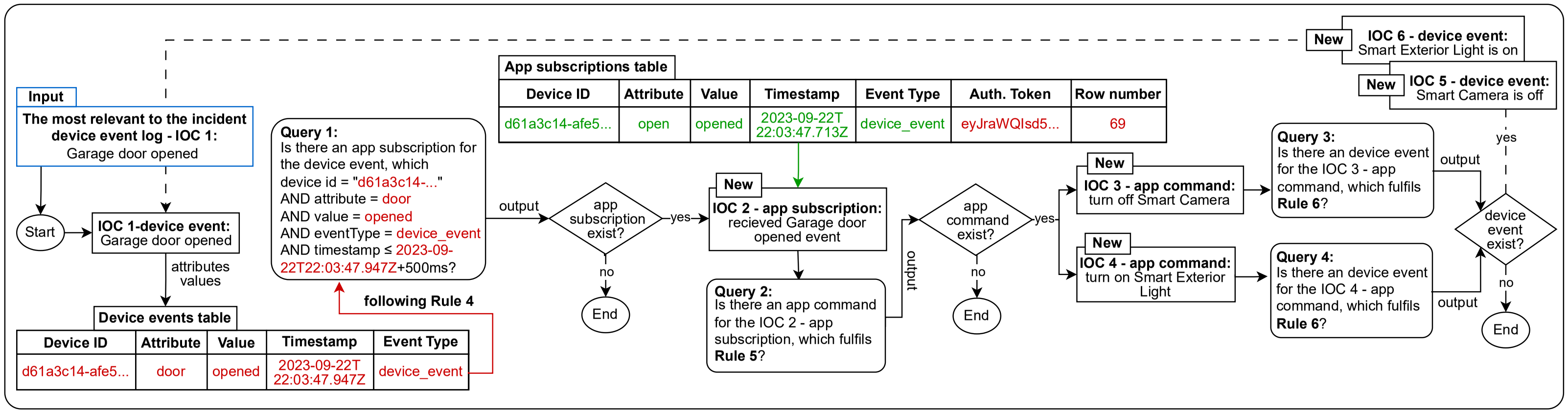
$$\begin{aligned} IOC_1 = IOC_2 \iff & IOC_1\{deviceId\} = IOC_2\{deviceId\} \wedge \\ & (IOC_1\{value\} \supseteq IOC_2\{command\} \vee \\ & IOC_1\{attribute\} = IOC_2\{command\}) \wedge \\ & |IOC_1\{timestamp\} - IOC_2\{timestamp\}| \leq 500ms, \\ & IOC_i \in Logs \end{aligned} \tag{1}$$

According to this rule, the IOC_2 can be mapped to the IOC_1 if their `deviceId` field values are identical. Simultaneously, the IOC_2 `command` field should be a subset of the IOC_1 `value` field or the IOC_2 `command` field should match the IOC_1 `attribute` field. In addition to the above conditions, these IOCs' `timestamp` fields must differ by 500 milliseconds or less. According to the rule, the IoTINT constructs a query utilizing the IOC_1 data. Specifically, given the IOCs are classified into the database tables according to their type, the query should target to search IOC_2 in the `app_command_table`, which consists of `app command` IOCs or the `user_command_table`, consisting of `user command` IOCs (e.g., `SELECT * FROM user_command_table...`). In addition, IOC_2 has to fulfill the conditions described by Rule 1, meaning that the specific column values in a database table should correspond to the IOC_1 field values as specified in the rule. Thus, the remaining query is structured as follows: "`WHERE deviceID=`$IOC_1\{$*deviceID*$\}$ `and (command =` $IOC_1\{$*value*$\}$ `or ...`". Consequently, query execution on the database of logs leads to the retrieval of the pertinent IOC in a preceding direction (e.g., *user command*). Subsequently, with newly acquired IOC IoTINT follows the steps from Figure 3.3a until no further relevant IOCs are discovered in the preceding direction. Following this, the IoTINT moves on the subsequent direction search, commencing from the same IOC utilized in the previous phase. As the starting IOC relates to a *device event* and we are looking for the following IOC, IoTINT utilizes Rule 4 that describes the connectivity between the *device event* IOC and *app subscription* IOC, according to Figure 3.3b. Here, IOC_1, IOC_3 represent the *device event* and *app subscription* IOCs, respectively and IOC_i represents a log entry from the log collection, $IOCs$. The rule defines that these IOCs can be mapped if they occur within a specific period of time (e.g., 500ms) and other attributes mentioned in the rule are the same

$$\begin{aligned} IOC_1 = IOC_3 \iff & IOC_1\{deviceId\} = IOC_3\{deviceId\} \land \\ & IOC_1\{attribute\} = IOC_3\{attribute\} \land \\ & IOC_1\{value\} = IOC_3\{value\} \land \\ & IOC_1\{eventType\} = IOC_3\{eventType\} \land \\ & |IOC_1\{timestamp\} - IOC_3\{timestamp\}| \leq 500ms, \\ & IOC_i \in IOCs \end{aligned} \tag{4}$$

Consequently, IoTINT populates the rule with the IOC data fields to build a query. Execution of the query retrieves the newly relevant IOC in the subsequent direction, and the mapping process is being continued. In essence, the IoTINT undertakes a comprehensive IOC extraction procedure by iteratively pursuing both backward and forward directions until all pertinent IOCs associated with the user's input have been revealed. The rest of the mapping rules (Rules 2,3,5,6) that IoTINT uses to derive IoT-specific IOCs are present in Appendix A.2 to preserve the better readability of the thesis.

Example 2. First, IoTINT receives the event log most pertinent to the incident, which becomes an IOC 1. Within this example, this log records the garage door opened device event that happened at night (`2023-09-22T22:03:47`), serving as input to IoTINT. Consequently, IoTINT initiates a search for the relevant IOCs in the preceding and subsequent direction, following the flow charts from Figure 3.3. Figure 3.4 illustrates an example of deriving dynamic connectivity between IoT-specific IOCs in subsequent direction leveraging the sequence depicted in Figure 3.3b. Initially, IoTINT retrieves all attributes associated with IOC 1, such as `Device ID`, `Attribute`, `Value`, etc., to facilitate further query construction. Then, utilizing Rule 4, which defines the linkage between *device event* IOCs and *app subscription* IOCs and incorporating the data from the garage door opened IOC, IoTINT constructs `Query 1`. This query aims to identify any *app subscription* IOC in the database which `device id = "d61a3c14-..." AND attribute = "door" AND value = "opened" AND eventType = "device_event" AND timestamp ≤ 2022-09-22T22:03:47.947Z+500ms`. IoTINT ceases the search in this direction if no matching IOC is found. Continuing with the example, IoTINT identifies a new *app subscription* IOC 2, along with its associated attributes, as the search progresses. Then, following


Input
The most relevant to the incident device event log - IOC 1: Garage door opened
Start
IOC 1-device event: Garage door opened
attributes values
Device events table
following Rule 4
output
app subscription exist?
yes
no
End
New
IOC 2 - app subscription: recieved Garage door opened event
output
app command exist?
yes
no
End
New
IOC 3 - app command: turn off Smart Camera
New
IOC 4 - app command: turn on Smart Exterior Light
output
device event exist?
yes
no
End
New
IOC 5 - device event: Smart Camera is off
New
IOC 6 - device event: Smart Exterior Light is on

Device events table

Device ID	Attribute	Value	Timestamp	Event Type
d61a3c14-afe5...	door	opened	2023-09-22T 22:03:47.947Z	device_event

Query 1: Is there an app subscription for the device event, which device id = "d61a3c14-..." AND attribute = door AND value = opened AND eventType = device_event AND timestamp ≤ 2023-09-22T22:03:47.947Z+500ms?

App subscriptions table

Device ID	Attribute	Value	Timestamp	Event Type	Auth. Token	Row number
d61a3c14-afe5...	open	opened	2023-09-22T 22:03:47.713Z	device_event	eyJraWQIsd5...	69

Query 2: Is there an app command for the IOC 2 - app subscription, which fulfils **Rule 5**?

Query 3: Is there an device event for the IOC 3 - app command, which fulfils **Rule 6**?

Query 4: Is there an device event for the IOC 4 - app command, which fulfils **Rule 6**?

Figure 3.4: An example of deriving dynamic connectivity between IoT-specific IOCs for the *device event* IOC (`garage door open`) through IoTINT in subsequent direction.

the subsequent flow chart, IoTINT generates Query 2 based on Rule 5 and the attributes of IOC 2, such as authentication token and row number. Execution of Query 2 against the log database retrieves new *app command* IOCs. Specifically, IOC 3 represents a command to turn off the smart camera, while IOC 4 directs the smart exterior light to turn on. Consequently, following Rule 6, IoTINT constructs Query 3 and Query 4 from the data associated with IOC 3 and IOC 4, respectively. Upon executing these queries, IoTINT retrieves two new *device event* IOCs, indicating the status of the smart camera and smart exterior light as being off and on, respectively. Thus, these are all new IOCs that can be extracted for subsequent direction within this example. The complete result of IoT-specific IOCs extraction for our motivating example is shown in Section 3.5.

3.4 Combining with Traditional Threat Intelligence IOCs

Those newer IOCs from the app-device dynamic connectivity (obtained in the previous step) need additional insights with the help of traditional IOCs (e.g., CVEs, CWEs, and CAPECs) to gain more detailed threat intelligence (as explained in Section 2.2). To that end, while finding corresponding CVEs, CWEs, and CAPECs for an IoT product (devices and apps), we utilize Common Platform Enumeration (CPE) [43] representations (as it includes both product details and their CVE numbers). However, in this process, we face the following challenges.

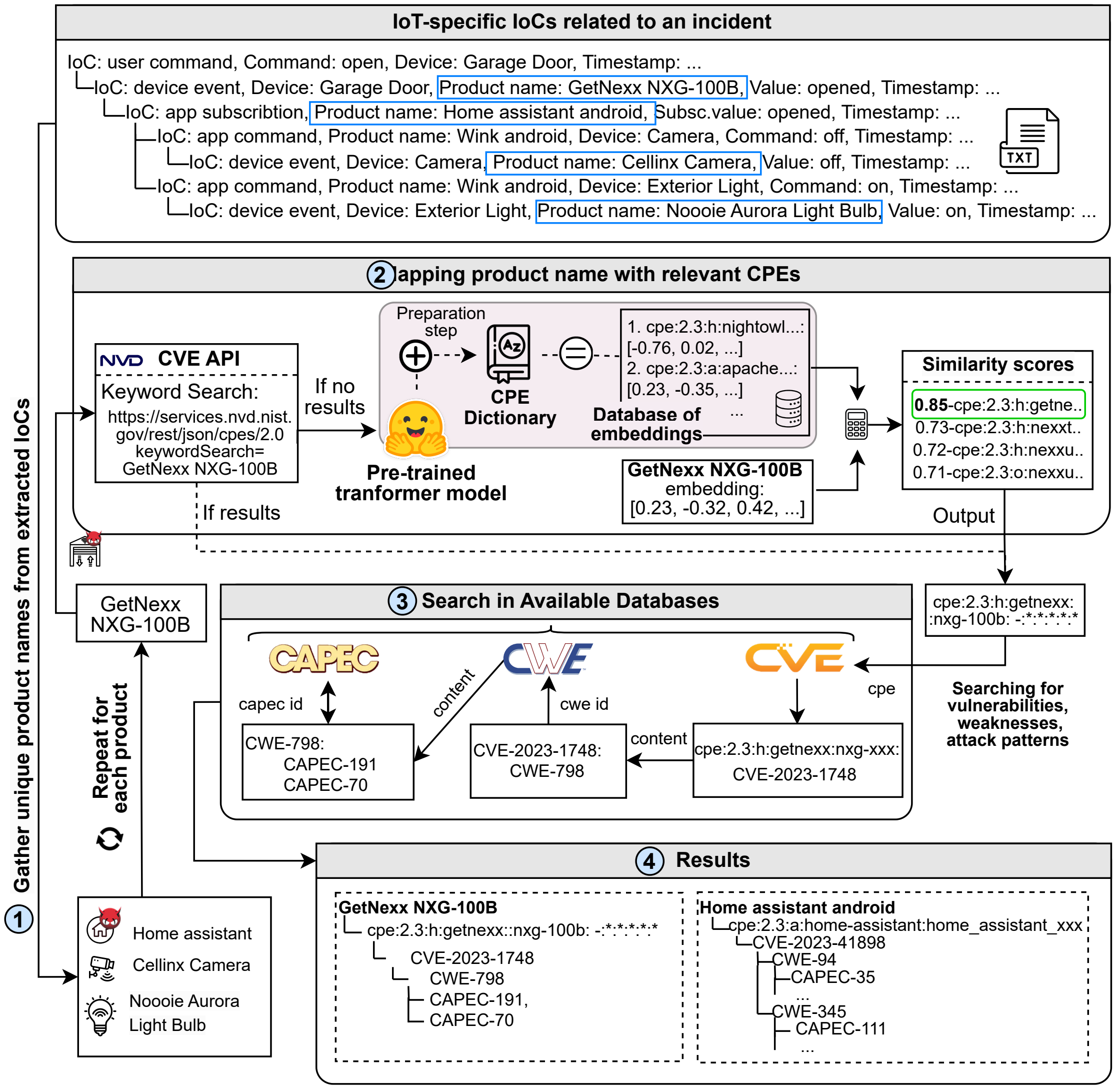
3.4.1 Challenges

CPE is a standardized naming format (e.g., *cpe:2.3:a: eaton:halo_home:1.11.4:*:*:*:*:android: :**) that comprises fields like vendor, product name, version, update, etc. to help differentiate various products. Initially, we attempted to map product names with their corresponding CPEs using the CPE API's keyword search functionality provided by the NVD. This approach retrieves all CPE entries in which all the provided keywords are present in their metadata title or reference links. However, this method often fails to yield results due to keyword discrepancies and the absence of specific keywords in the actual CPE metadata. For example, the API call with the Fibaro motion sensor fgms-001 product name does not find any related CPEs only because the keywords `motion` and `sensor` are not present in the relevant CPE metadata, and the users are not aware of these details while setting the names for their devices or apps.

3.4.2 Solution

To address this issue, we adopt an Natural language processing (NLP)-based approach as follows. First, by leveraging a pre-trained transformer model [44], IoTINT computes the embedding of all existing CPEs and stores them in a database. Next, upon receiving the product name, it calculates the embedding using the same model and further the similarity scores between the product name and all existing CPEs and shortlists the CPEs with the highest similarity scores. Then a security analyst manually chooses the most relevant CPE(s), as the number of CPEs varies, and IoTINT cannot recognize the case when none of the CPEs in the shortlist are associated with a product name. After we map traditional IOCs to the product's CPEs, we obtain all CVEs associated with each CPE through a CPE API call to the CVE NVD database. Furthermore, the content of each CVE includes CWE IDs, specifying the weaknesses potentially leading to a particular vulnerability. Additionally, CWE content encompasses Related Attack Patterns, indicated by CAPEC IDs. These attack patterns serve as valuable insights for security analysts, aiding in understanding how adversaries exploit weaknesses. Consequently, IoTINT retrieves CVEs, CWEs, and CAPECs associated with the product names and structures into a hierarchical tree based on their interconnections, facilitating the expansion of the information about a threat. Finally, it combines the obtained traditional IOCs with the IoT-specific IOCs to construct a report of incident-related threat intelligence.

Example 3. Figure 3.5 outlines the steps involved in combining IoT-specific IOCs with traditional IOCs, following our example scenario. In Step 1, IoTINT extracts unique product names (marked in blue) from earlier retrieved IoT-specific IOCs related to the garage door being opened (`GetNexx garage door opener NXG-100B`, `Home assistant app`, `Cellinx Camera`, and `Noooie Aurora Light Bulb`). In Step 2, for the `GetNexx garage door opener NXG-100B`, the keyword-based API call gave no results, leading to the utilization of the NLP-based approach. Thus, the pre-trained transformer model calculates the embedding of this door opener and IoTINT calculates individual similarity scores between the door opener name and all CPEs. Then, the list of N CPEs exhibiting the highest similarity scores with the product name is



IoT-specific IoCs related to an incident
TXT
2 apping product name with relevant CPEs
NVD CVE API
Keyword Search:
https://services.nvd.nist.gov/rest/json/cpes/2.0 keywordSearch= GetNexx NXG-100B
If no results
If results
Preparation step
CPE Dictionary
1. cpe:2.3:h:nightowl...: [-0.76, 0.02, ...]
2. cpe:2.3:a:apache...: [0.23, -0.35, ...]
...
Database of embeddings
Pre-trained tranformer model
GetNexx NXG-100B embedding: [0.23, -0.32, 0.42, ...]
Similarity scores
0.85-cpe:2.3:h:getne..
0.73-cpe:2.3:h:nexxt..
0.72-cpe:2.3:h:nexxu..
0.71-cpe:2.3:o:nexxu..
Output
cpe:2.3:h:getnexx::nxg-100b: -:*:*:*:*:*
GetNexx NXG-100B
3 Search in Available Databases
CAPEC
CWE
CVE
capec id
content
cwe id
cpe
content
CWE-798: CAPEC-191 CAPEC-70
CVE-2023-1748: CWE-798
cpe:2.3:h:getnexx:nxg-xxx: CVE-2023-1748
Searching for vulnerabilities, weaknesses, attack patterns
1 Gather unique product names from extracted IoCs
Repeat for each product
Home assistant
Cellinx Camera
Noooie Aurora Light Bulb
4 Results

```
IoC: user command, Command: open, Device: Garage Door, Timestamp: ...
└IoC: device event, Device: Garage Door, Product name: GetNexx NXG-100B, Value: opened, Timestamp: ...
  └IoC: app subscribtion, Product name: Home assistant android, Subsc.value: opened, Timestamp: ...
    ├IoC: app command, Product name: Wink android, Device: Camera, Command: off, Timestamp: ...
    │ └IoC: device event, Device: Camera, Product name: Cellinx Camera, Value: off, Timestamp: ...
    └IoC: app command, Product name: Wink android, Device: Exterior Light, Command: on, Timestamp: ...
      └IoC: device event, Device: Exterior Light, Product name: Noooie Aurora Light Bulb, Value: on, Timestamp: ...
```

```
GetNexx NXG-100B
└ cpe:2.3:h:getnexx::nxg-100b: -:*:*:*:*:*
   └ CVE-2023-1748
      └ CWE-798
         ├ CAPEC-191,
         └ CAPEC-70
```

```
Home assistant android
└cpe:2.3:a:home-assistant:home_assistant_xxx
  └CVE-2023-41898
    ├CWE-94
    │ ├CAPEC-35
    │ ...
    └CWE-345
      ├ CAPEC-111
      ...
```

Figure 3.5: An example of combining IoT-specific IOCs related to the `garage door open` incident with the traditional threat intelligence IOCs

shown to the security expert, facilitating the manual selection of the most appropriate CPE(s). As a result, CPE entry: `cpe:2.3:h:getnexx:nxg-100b:-:*:*:*:*:*` was selected as the representation of this device. In Step 3, IoTINT searches for the vulnerabilities, weaknesses and attack patterns that are related to the identified CPE. Specifically, IoTINT obtained CVE-2023-1748, associated with the CPE, through the CVE API call to the NVD database. Within the content of this CVE, IoTINT identified CWE-798, a weakness potentially leading to the `CVE-2023-1748` vulnerability. Furthermore, the CWE content highlighted `CAPEC-191` and `CAPEC-70`, which are pertinent attack patterns. A procedure similar to the one above for fetching traditional IOCs is replicated for all other product names pertinent to this incident. As a result of Step 4, the traditional IOCs are identified for the garage door device and the smart application responsible for controlling the smart camera and exterior light devices. Each retrieved CVE, CWE, and CAPEC IDs are structured into a tree based on their relationships, as illustrated in Figure 3.5.

3.5 Producing Threat Intelligence Report

The final step of IoTINT focuses on generating threat intelligence reports in two distinct formats: machine-readable and human-readable (i.e., graphical). The snippet of the machine-readable report is included in Appendix A.1. Among the array of machine-readable formats available, we select STIX as the most optimal solution for storing and sharing threat intelligence [45]. To represent the threat intelligence report in the STIX format, we use two types of objects: *indicator*, which represents IoT-specific or traditional IOCs and *relationship*, which specifies the connectivity (edges) between indicator objects. The machine-readable format of the report can be utilized for an automated incident response when a previously observed threat is encountered and response procedures are already defined. In contrast, the human-readable format visually represents derived artifacts related to the incident, facilitating security professionals in a more efficient and comprehensive analysis. Use cases demonstrating the further utilization of these formats are detailed in Chapter 6 through case studies.

Example 4. Figure 3.6 presents the threat intelligence report in a human-readable format about the `garage door open` incident from our motivation example. In this graphical representation,

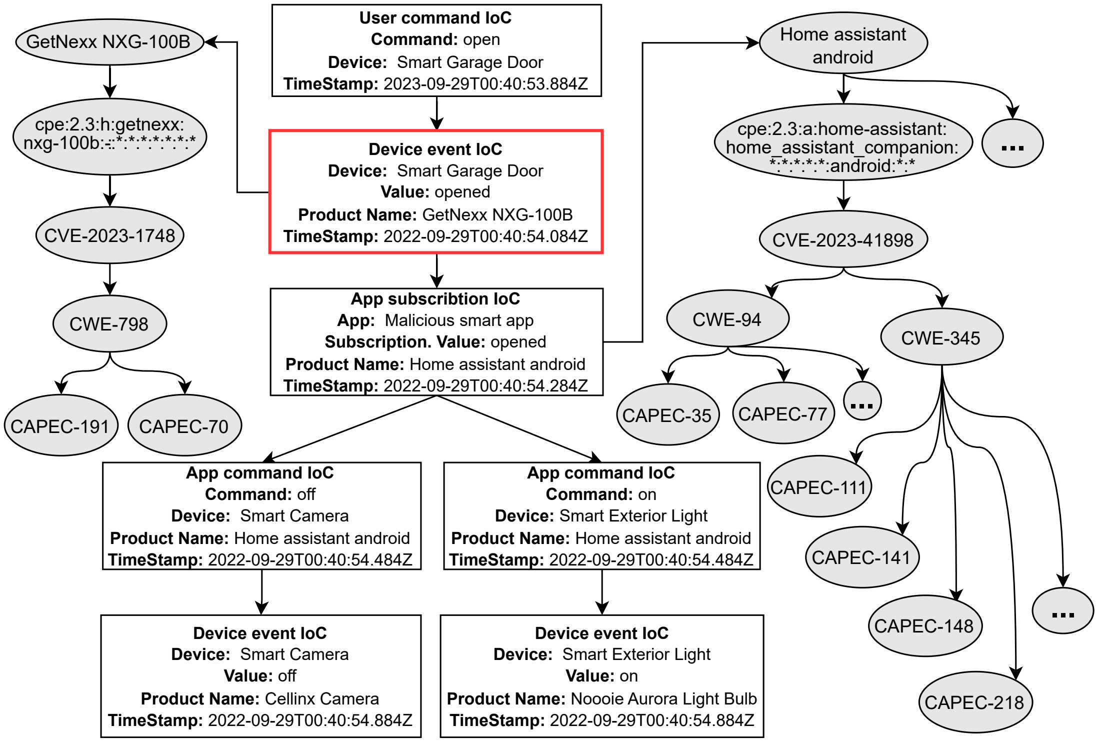

GetNexx NXG-100B
cpe:2.3:h:getnexx:
nxg-100b:-:*:*:*:*:*:*:*
CVE-2023-1748
CWE-798
CAPEC-191
CAPEC-70
User command IoC
Command: open
Device: Smart Garage Door
TimeStamp: 2023-09-29T00:40:53.884Z
Device event IoC
Device: Smart Garage Door
Value: opened
Product Name: GetNexx NXG-100B
TimeStamp: 2022-09-29T00:40:54.084Z
App subscribtion IoC
App: Malicious smart app
Subscription. Value: opened
Product Name: Home assistant android
TimeStamp: 2022-09-29T00:40:54.284Z
App command IoC
Command: off
Device: Smart Camera
Product Name: Home assistant android
TimeStamp: 2022-09-29T00:40:54.484Z
App command IoC
Command: on
Device: Smart Exterior Light
Product Name: Home assistant android
TimeStamp: 2022-09-29T00:40:54.484Z
Device event IoC
Device: Smart Camera
Value: off
Product Name: Cellinx Camera
TimeStamp: 2022-09-29T00:40:54.884Z
Device event IoC
Device: Smart Exterior Light
Value: on
Product Name: Noooie Aurora Light Bulb
TimeStamp: 2022-09-29T00:40:54.884Z
Home assistant
android
cpe:2.3:a:home-assistant:
home_assistant_companion:
::*:*:*:android:*:*
...
CVE-2023-41898
CWE-94
CWE-345
CAPEC-35
CAPEC-77
...
CAPEC-111
CAPEC-141
CAPEC-148
CAPEC-218
...

Figure 3.6: An example of a threat intelligence report in a visual graph representation for the `garage door open` incident

the rectangular box with a red border shows the initial log selected by the security analyst as the one most closely aligned with the observed incident, which becomes the starting IOC - specifically, the *device event* IOC, indicating that the garage door was opened. Furthermore, other rectangular boxes denote incident-related IoT-specific IOCs, while the oval-shaped boxes represent traditional IOCs, encompassing vulnerabilities, weaknesses, and attack pattern IDs. The middle part of the graph depicts that the garage door was opened because of the user command. Subsequently, the commands to turn off the camera and turn on the exterior light were issued from the smart app `Home assistant android`. This sequence of events raises suspicions from security analyst about the smart app's maliciousness, particularly as the camera should always be in *on* state for security reasons. Moreover, the vulnerabilities found in the smart app strengthen the security analyst's concerns regarding its potential maliciousness. However, a critical question remains: *How was the garage door opened if no authorized smart home users initiated the action?* The oval part of the graph reveals that the garage door device `GetNexx NXG-100B` has vulnerabilities, and consequently, they might be exploited to send the garage door opening command. To validate this hypothesis, security professionals must check the content of the relevant vulnerabilities. Notably, within this example, the comprehensive report itself is classified as a *computed* IOC, aligning with the taxonomy outlined in Section 2.3. Identified *atomic* IOCs encompass the *device event* IOC indicating the door being open and the smart *app command* IOC for disabling the camera, both of which are potentially indicative of malicious actions. Lastly, the branch extending from the top to the bottom left of the IoT-specific graph nodes is designated as a *behavioral* IOC, encapsulating smart home behavior intricately linked to the attack.

Chapter 4

Implementations

This section describes the system architecture of IoTINT and implementation details of some modules.

We implement IoTINT on the Samsung SmartThings platform due to its support for a wide range of IoT devices and open-source smart applications [33], along with extensive documentation for developers [46] as follows.

4.1 System Architecture

The high-level architecture of IoTINT is presented in Figure 4.1. IoTINT consists of three components: *IoTINT Dashboard*, *IoTINT Engine* and *Data Sources*, which is an input.

The *IoTINT Dashboard* allows interaction with IoTINT users and produces threat intelligence reports. It contains two sub-modules, namely *Interface* and *Report Generator*. The *Interface* module allows IoTINT users to select the log record from the database that closely aligns with the observed scenario and choose in which format they want the report to be generated. While, the *Report Generator* module is responsible for producing threat intelligence reports in graph and machine-readable formats (implementation details in Section 4.4).

The *Data Sources* consist of simulated IoT devices, IoT platform and smart apps running on third-party servers. We collect the communications (e.g., events, commands) between these entities in the form of raw data and transport it to the *IoTINT Engine* as an input. Further elaboration on data collection implementation from *Data Sources* is provided below in Section 4.2.

The *IoTINT Engine* serves as the central module within our solution, comprising four distinct sub-modules. The *Data Processor* is a module that takes raw data from the IoT platform as input, classifies logs for each device or app, applies predefined filters to transform logs into a form of potential IOCs (described in Section 3.2) and stores the essential logs inside an SQLite database. The *IoT-specific IOCs Extractor* module utilizes the database of logs and mapping rules implemented in the form of SQL-based queries to derive IoT-specific IOCs related to the incident and establish their connectivity (discussed in Section 3.3). This module is implemented in Python language using *datetime*, *calendar* and *sqlite3* libraries with the aid of SmartThings Documentation [4]. In addition, the *Traditional IOCs Collector* module is responsible for finding corresponding CVEs, CWEs and CAPECs for IoT products (devices, apps) related to the incident (discussed in Section 3.4). A detailed explanation of the implementation of Traditional IOCs Collector is provided in Section 4.3 below. Finally, gathered IoT-specific and traditional IOCs related to the incident feed the *Integration module*, which combines the IOCs and produces final threat intelligence information about the security incident. This information then transported to the *IoTINT Dashboard* for the report generation.

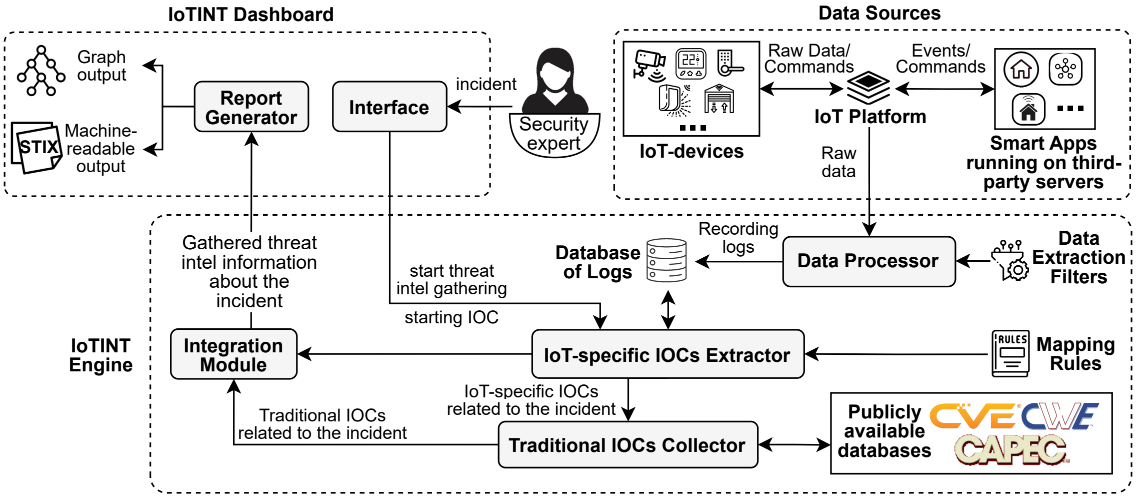


Figure 4.1: A high-level architecture of IoTINT.

4.2 Data Collection

In our setup, we establish a proxy channel to facilitate data collection between the SmartThings cloud platform and the server hosting the smart apps. This channel is responsible for intercepting all network traffic exchanged between these components and monitoring the data flow. To create this channel, we employ a web debugging proxy tool known as Fiddler Classic [47], which functions as both a forward and reverse proxy. To monitor the data flow between the smart apps and the SmartThings cloud platform comprehensively, we deploy a forward proxy to track commands and API calls initiated by the smart apps. Simultaneously, we utilize a reverse proxy to monitor webhook requests originating from the IoT platform. Our monitoring extended to network traffic over the application layer, specifically HTTPS, and for this, we use our server certificates with Fiddler to decrypt the traffic. In addition to this, to capture event logs from the IoT platform, we establish a WebSocket connection with the platform via the SmartThings Groovy IDE account [48]. Subsequently, we employ Fiddler Classic to monitor, gather, and decrypt all WebSocket data transmitted from the platform to our server. Following the data collection phase, IoTINT undergoes raw application and device logs pre-processing by categorizing and extracting specific fields from each log entry. Subsequently, various log types are stored in an SQLite database, enabling their further extraction as IOCs.

4.3 Traditional IOC Extraction

To derive traditional IOCs, we leverage accessible CPE Dictionary, CVE NVD, CWE, and CAPEC data sources. Our approach involves employing the CPE API with the *keywordSearch* parameter [49] to identify CPEs associated with product names. Furthermore, we utilize the CVE API's *cpeName* parameter [50] to find the relevant CVEs linked to each identified CPE. The other relationships between traditional IOCs are extracted from their content.

To implement the proposed alternative solution for mapping product names with CPEs discussed in Section 3.4, we utilize the *jinaai/jina-embeddings-v2-base-en* [51] pre-trained transformer model to generate embeddings for all existing CPE strings in the dictionary. The resulting database file containing CPEs and their embedding occupies 18 GB. Consequently, the data extraction takes 12 minutes while calculating cosine similarities between the product name and CPEs to identify the most relevant matches required an additional three minutes. This considerable time overhead for each product name → CPE mapping presents a significant optimization challenge that needs to be addressed.

To address the aforementioned challenge, we turn to the Facebook AI Similarity Search (FAISS) library [52], leveraging its capabilities for efficient similarity searches. FAISS operates by constructing an index, a RAM-based data structure, from a collection of vectors x_i in a specified dimension, d. Once this structure is established, given a new vector x in dimension d, it efficiently performs the operation:

$$j = argmin_i ||x - x_i|| \tag{5}$$

where $||.||$ represents the Euclidean distance (L^2). Thus, FAISS operates by returning the k nearest neighbours—essentially, the most pertinent CPEs relevant to the given product name. After implementing this optimization technique, the index data structure size decreased notably to 6.72 GB, substantially reducing the time required to read the `.index` file to 7.5 seconds. Moreover, the extraction of k nearest neighbours is achieved in less than a second. As a result of these improvements, we keep this optimization strategy, as it significantly alleviates both storage constraints and processing time without compromising accuracy.

4.4 Report Generation

When IoTINT retrieves all the IoT-specific and traditional IOCs and their connectivity, we utilize *anytree* Python library [53] to store the gathered threat intelligence in a tree data structure format. Subsequently, IoTINT produces reports in visual and machine-readable formats. To visually represent IoT threat intelligence for a specific incident, we employ the *graphviz* Python library to create a graph. In addition, to generate a machine-readable report in STIX format, we leverage the *cti-python-stix2* library.

Chapter 5

Performance Evaluation

This chapter presents the experimental setup and our evaluation results to measure IoTINT's performance (in terms of accuracy, overhead, and feasibility) along with the validation of the obtained distribution of different IOC types during our experiments with the similar results from existing works [17, 54].

5.1 Experimental Setup

This section describes the IoTINT experimental setup and two types of datasets that we used to evaluate the performance of IoTINT.

5.1.1 Testbed Configuration

We deploy IoTINT on a desktop machine on Windows 10 Enterprise OS equipped with an Intel Core i7-10700 2.90 GHz processor and 32 GB of RAM. The implementation of IoTINT in Python leverages Graphviz [55], Node.js [56], FAISS [57] libraries and *jinaai/jina-embeddings-v2-base-en* [44] pre-trained transformers model. SQLite is used as a database, and real accumulated data is utilized in the threat intelligence generation process. To mimic the behavior of actual IoT devices in a smart home, we employ SmartThings IDE [48] that facilitates the simulation of device actuators and sensors as similar to prior studies [16, 17, 58, 59]. Note that all devices used in this work were simulated. We also install ten distinct smart applications, each offering various functionalities, and simulate malicious scenarios within our setup.

5.1.2 Dataset Description

In this work, we evaluate the performance of IoTINT using two types of datasets: a *normal behavior* dataset and an *attack* dataset.

The *normal behavior* dataset represents everyday normal interactions between smart home devices, apps, the IoT platform, and users. The purpose of this dataset is to assess the overhead (e.g., runtime, storage, and memory) on a relatively large volume of data. This dataset encompasses 50 IoT devices and sensors, totalling approximately 20 MB logs and containing over 3,400 events. To replicate a smart home environment, we employ simulated IoT devices, which are originally free from vulnerabilities or misconfiguration. Thus, to enable the extraction of traditional IOCs, we assign real product names to each device, with an emphasis on including a majority of vulnerable devices to enhance realism. Consequently, out of the 50 IoT devices in the typical dataset, 36 are labelled with the names of vulnerable products, while the remaining 14 IoT device names do not indicate any vulnerabilities. Furthermore, these devices and sensors are managed by four smart apps with varying automation logic, to which we also assigned actual names of vulnerable apps. While developing these smart apps with the Node.js language, we leverage the logic of 181 open-source SmartThings apps [37].

In contrast, the *attack* dataset incorporates both malicious behaviors and regular activities for 10 different attacks. The purpose of this dataset is to evaluate the capability of IoTINT to generate threat intelligence about a particular security incident. In contrast with the normal dataset, each attack dataset is smaller, featuring fewer than 23 IoT devices and sensors, with a size of less than 4 MB and containing around 400 events. We develop 10 smart apps capable of carrying out the attacks documented in recent attack papers [16, 33–36] and also assign real names of vulnerable smart apps to them. Each of these applications performs the specified attack concurrently with the regular smart home behavior, resulting in ten distinct attack datasets, which are summarized in Table 5.1.

Table 5.1: Description of the attack datasets

Attack Datasets	Incident	Number of Incident Occurrences	Average Size of Reports
A1: System Events [16]	Camera footage is black as exterior light was turned off	7	90
A2: Undesired Unlocking [34]	The door was unlocked during the night without any reason	3	46
A3: Side Channel [16]	Burglary. Neighbours noticed that the light was strobing for 1 min in the evening	6	142
A4: PinCode Injection [33]	The lock PINcode was changed without the user's knowledge	4	17
A5: Adware Notification [16]	Notification with suspicious advertisement to download malicious app	5	22
A6: App Update [16]	The lock device got uncharged but user didn't receive any notification.	3	6
A7: Remote Command [16]	Users accidentally found out about the fire in the kitchen. The fire alarm didn't work.	6	44
A8: Remote Control [16, 35]	The alarms are launching siren without any reason	4	50
A9: Spoof Mode Event [33, 36]	The location mode is randomly changed during the day	7	320
A10: Spoof Mode Event and System Events [33, 35]	Locks are unexpectedly unlocking during the day without user awareness	8	11

5.2 Evaluation Objectives

This section describes research questions whose answer is used to evaluate the performance of IoTINT in various aspects. The research questions (RQX) are as follows:

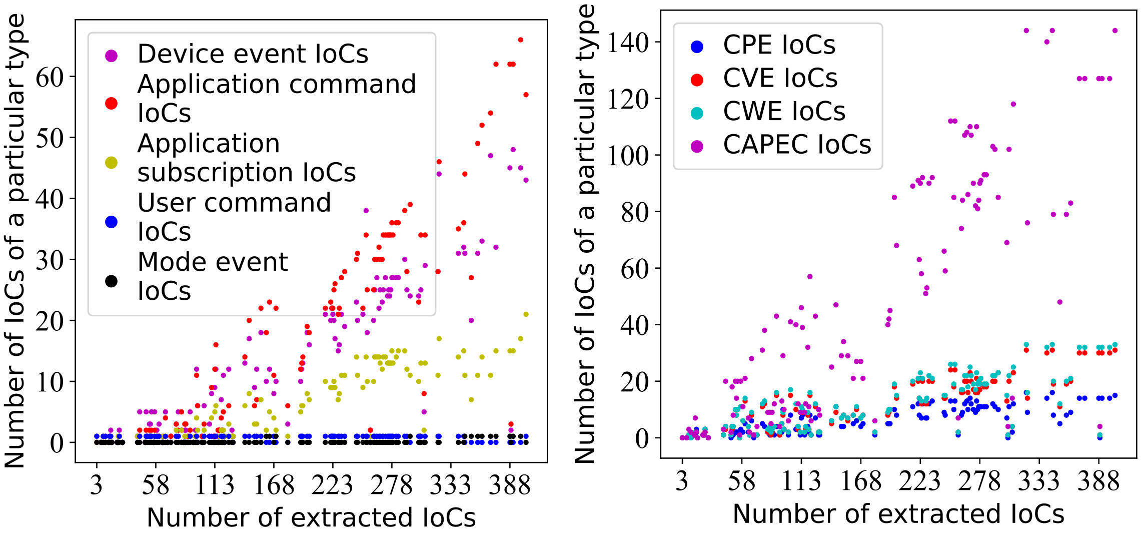
- RQ1: Does the distribution of various types of IoT-specific and traditional IOCs in the threat intelligence reports align with the trends from existing works?
- RQ2: What is the accuracy of IoTINT in the following scenarios: identifying IOCs, comparing with the ML-based approach and mapping CPEs to product names?
- RQ3: What is the performance overhead (in terms of time, storage and memory) of IoTINT in generating IoT-specific threat intelligence?
- RQ4: How feasible is human intervention required by IoTINT (i.e., mapping CPEs to the products from Section 3.4)?

5.3 Validation with Existing Works

In Figure 5.1, we provide the distribution of different types of IOCs within both IoT-specific and traditional categories. The graphs show the quantity of various IOC types relative to the overall number of IOCs in the threat intelligence reports produced by IoTINT from the typical dataset. We share these findings not only to contribute to our own research but also to offer insights for future works in this domain. Researchers in the field can leverage the distribution of various IOC types to customize their threat intelligence solutions in IoT. The results are further validated with state-of-the-art works. Here, by "validation" we mean confirming that the observed distribution trends in IOC types are correlating with the existing works.

5.3.1 IoT-specific IOC Validation

Figure 5.1a illustrates that the most prevalent IoT-specific IOC types are *application command* and *device event*, with slightly less popularity attributed to *application subscription* IOCs. On the other hand, the least common IOC types, each appearing either once or not at all in the output, are



(a) IoT-specific IOCs

(b) Traditional IOCs

Figure 5.1: Distribution of various IOCs

user command and *mode event* IOCs. To validate such distribution of IoT-specific IOCs, we compare the outputs of a state-of-the-art work, ProvThings [17], with the IoTINT outputs. To make a comparison, we extracted all five reported use cases by ProvThings and derived the only device/app behavior nodes from their graphs as they additionally detect smart app function calls. Consequently, we analyze the quantity of nodes describing smart environment behavior (e.g., app command, device event) produced by ProvThings in comparison to the number of various IOCs extracted by IoTINT. The validation results are presented in Table 5.2. The blue-coloured numbers denote the number of nodes extracted from ProvThings graphs, while the green-coloured numbers represent the IoTINT IOCs from similar use cases. Note that the numbers in the table indicate the numbers of similar IOC types that are extracted by both of these works. As a result, in most use cases, both solutions exhibit equal trends. However, there are instances where IoTINT extracts more IOCs, such as in UC5. While this could potentially be considered a false positive, it is noteworthy that ProvThings provides only preceding information to the input incident. Thus, if the open command for the window is considered an incident, the tool will show that it happened because of the detected smoke event. In contrast, IoTINT also reveals the smart home activity following the initiated command, such as an opened window event. In summary, our results align with the trends observed in ProvThings regarding the prevalence of IoT-specific IOCs.

Table 5.2: Validating IoT-specific IOCs extraction with ProvThings work [17]. Blue-coloured numbers denote the number of nodes of extracted from ProvThings graphs, while the green-coloured numbers represent the IoTINT IOCs from similar use cases

Incident (Use Cases)	**Number of IOCs of a particular type**									
	App command		Device event		App subscription		Mode event		User command	
Kitchen light was turned on by Apple HomeKit app (UC1)	1	1	2	2	1	1	0	0	0	0
Unintended unlock door event for a front door (UC2)	2	2	2	2	1	2	1	1	0	0
PIN code leakage (UC3)	3	3	6	6	2	2	0	0	0	0
Fake smoke event (UC4)	4	4	4	4	1	1	0	0	0	0
Window opened by SmokeMonitor (UC5)	1	1	1	2	1	1	0	0	0	0

5.3.2 Traditional IOC Validation

Figure 5.1b shows that CAPEC IOCs constitute the largest share of traditional IOC types, with individual reports containing as many as 150 CAPEC IOCs. Conversely, the number of CVE and CWE IOCs in the report exhibits a lower but more consistent trend, with counts reaching up to 38. In comparison, the report typically contains a maximum of 18 extracted CPE IOCs, particularly in those with a significant number of nodes.

Table 5.3: Validating traditional IOCs extraction with BRON's graph work [54]. The cells provide an average number of IOCs of one type (e.g., CVEs) connected to one IOC of another type (e.g., CPE).

	CPE → CVE	CVE → CWE	CWE → CAPEC
BRON [54]	1-2	0-1	8
IoTINT	1-2	1	4

To verify the distribution of traditional IOCs, we reference the work by Hemberg et al. [54], which produces the BRON's graph. Specifically, this work links MITRE TTPs, CWEs, CVEs, CPEs, and CAPECs, presenting all entities and relationships as a graph. To compare traditional

IOCs produced by IoTINT with the BRON's graph, we calculate the average number of IOCs connected to one IOC type (e.g., CPE, CVE) directionally, from CPE to CAPEC. The comparison results are outlined in Table 5.3. The column CPE→CVE displays the average number of vulnerabilities relevant to one CPE. The column CVE→CWE illustrates the average number of weaknesses associated with one CVE. Finally, the column CWE→CAPECs presents the average number of CAPECs connected to one weakness. In addition, some results are depicted as a range (e.g., from 0 to 1) since the number of connected IOCs cannot be fractional. The table indicates that the amount of CPEs, CVEs and CWEs should be similar, varying a maximum of two times, while the number of CAPECs exceeds several times. Note that the number of connected CAPECs to a single CWE differs between BRON's graph and IoTINT due to our utilization of a limited number of devices and apps, resulting in a slightly less accurate result. In summary, our findings corroborate the trends observed in BRON's graph regarding the prevalence of traditional IOCs.

5.4 Accuracy

In this section, we provide the results of experiments on the coverage and accuracy of IoTINT, along with ML-based approach comparison.

5.4.1 Coverage of IoTINT in Identifying IOCs

The experimental results of IoTINT coverage in identifying IOCs are present in Table 5.4. To evaluate the ability of IoTINT to produce accurate threat intelligence, the authors of this research collaborated to establish ground truth for 10 distinct attack scenarios. First, they independently examined each attack implementation, and one of them covered a coding part. Another person was tasked with manually constructing reports illustrating IOCs and their interconnections based on data collected from the IoT platform. Finally, the coder and report constructor met to validate the accuracy of the outputs generated by IoTINT using the manually constructed reports of IOCs. We evaluate IOC coverage by measuring the percentage of all nodes and edges in the IoTINT report in comparison with the ground truth (similarly, as other related works, e.g., [17]). The ground truth is divided into two parts: nodes representing IoT-specific IOCs and traditional IOCs.

Table 5.4: Coverage of IoTINT approach in comparison with LSTM approach in IoT-specific IOCs identification. Traditional IOCs coverage by IoTINT.

Attack Datasets	Ground Truth		IoT-specific IOCs			Traditional IOCs
	IoT-specific IOCs	Traditional IOCs	IoTINT Approach	LSTM-based approach		
				Attack-specific Training	Accumulated Training	
A1	40	1	40 (100%)	28 (70%)	5 (12.5%)	1 (100%)
A2	29	9	29 (100%)	10 (34.5%)	7 (24.1%)	9 (100%)
A3	141	1	141 (100%)	113 (80.1%)	54 (38.3%)	1 (100%)
A4	8	9	8 (100%)	6 (75%)	4 (50%)	9 (100%)
A5	6	9	6 (100%)	4 (66.7%)	3 (50%)	9 (100%)
A6	4	1	4 (100%)	3 (75%)	2 (50%)	1 (100%)
A7	3	0	3 (100%)	3 (100%)	2 (66.7%)	0 (100%)
A8	394	9	394 (100%)	157 (39.8%)	62 (15.7%)	9 (100%)
A9	35	42	35 (100%)	22 (62.9%)	11 (31.4%)	42 (100%)
A10	24	1	24 (100%)	11 (45.8%)	3 (12.5%)	1 (100%)

Table 5.4 shows the IOC coverage of our work based on different attack classes. The “IoTINT Approach” under the “IoT-specific IOCs” header and “Traditional IOCs” columns reflect the number of IOCs in the reports generated by IoTINT for various incidents and coverage percentage. When the number of traditional IOCs equals zero, it indicates the absence of vulnerable devices or apps relevant to the incident. The comprehensive ground truth metrics demonstrated that the IoTINT consistently recovered 100% of the relevant IOCs related to each incident. This robust performance underscores the effectiveness of the tool in accurately identifying and extracting IOCs in diverse attack scenarios we have implemented for this work.

5.4.2 Comparison with ML-based Approach

In addition to these experiments, we compare the IoTINT rule-based approach with an ML-based approach where an LSTM model is utilized in identifying incident-related IoT-specific IoCs. We use LSTM for this experiment, as it can capture long-term sequential dependencies that is what we need for our context in smart home. Smart home behavior consists of command and event sequences that trigger smart applications and change device states. In future work, we plan to explore other sequence-based approaches (e.g., Transformer). To conduct these experiments, we first converted the logs from ten attack datasets and the normal behaviour dataset into a "word" format. For instance, the device event log about the smart light being in an off state was converted to `de_sl1_off` word, where `de_` specifies the log type (e.g., device event, app command, app subscription, etc.), `sl1_` depicts the smart IoT device (e.g., smart camera, smart lock, etc.) and `off` represents the device state. Analogously, the word `ac_sa5_lock_slo2` describes the app command (`ac_`) log about smart app 5 (`sa5_`) sending a lock command to the smart lock (`slo2`). Similarly, other log types were converted into corresponding word formats. Subsequently, we sorted all words in each dataset based on their timestamps to generate sequences of smart home activities. Then, we trained individual LSTM models for each attack dataset (A1-A10). Additionally, we conducted accumulated LSTM training on the data that included the normal behavior logs sequence along with the sequences from the ten attack datasets. To compare the coverage of the IoTINT approach with the LSTM approach, we input a word representing the incident-related IOC (specific to each attack) into the model and request it to predict the next *N* words. From the predicted words, we count only those that match the IOCs detected as incident-related by the IoTINT.

Table 5.4 illustrates the outcomes of the experiments conducted using the LSTM-based approach. The "Attack-specific training" column under the "LSTM-based approach" header depicts the number of incident-related IoT-specific IOCs correctly predicted by the models separately trained on each attack dataset and the percentage according to the IOCs detected by IoTINT. In contrast, the "Accumulated training" column presents the number and percentage of incident-related IoT-specific IOCs correctly predicted by the model trained on the accumulated data, including normal behavior and attack dataset logs according to the IoTINT. Overall, the LSTM-based approach shows lower coverage compared to the IoTINT. Specifically, the coverage achieved through attack-specific training varies from 34.5% to 75%, with one instance reaching 100%, while the accumulated training coverage drops to a range of 12.5% - 66.7%. Such statistics can be explained by considering the density of attack behaviors. The model trained on a specific attack dataset can capture more attack-related evidence, while the model trained on the accumulated data is overwhelmed with the variety of different automation, hindering its ability to learn dependencies related to attack-specific IOCs. In summary, the LSTM-based approach shows lower coverage than the IoTINT rule-based approach in extracting related to the incident IOCs. Furthermore, the LSTM model struggles to learn complicated dependencies to predict parallel sequences (e.g., a tree format). Thus, the experiments underscore the significance and necessity of the IoTINT rule-based approach.

5.4.3 Measuring the Accuracy of Mapping CPEs to Products

Table 5.5 illustrates the accuracy of IoTINT in identifying CPEs representing specific product names using our approach in Section 3.4. To evaluate the accuracy of this approach, we conduct experiments on 19 real IoT products with varying threshold values from 0.01 to 0.034 with step 0.002 and calculate F1 scores. For each product name, we manually define the ground truth by checking each CPE option and the product it represents from the Internet. In evaluating the threshold, we aim to minimize FP and FN to mitigate the costs associated with additional or missing incident-related data. While including irrelevant CPEs is considered non-critical, the omission of CPEs representing the product name is deemed crucial, as it potentially results in the loss of vital information about devices or applications and incomplete threat intelligence. The F1 score is selected as the evaluation metric for its balanced consideration of both FPs and FNs, in contrast to accuracy metrics that treat the identification of one extra CPE and the loss of one CPE equivalently.

Table 5.5 presents the F1 scores corresponding to 13 threshold values ranging from 0.01 to 0.034 for 19 device names, where F1 scores reaching 1.00 are highlighted in green. Additionally, the device name rows for which F1 scores never reach 1.00 are highlighted in yellow, indicating that the threshold approach fails to map these device names with CPEs accurately. Unfortunately, no single threshold universally ensures the highest accuracy for all device names due to their variation and different amounts of relevant CPEs. However, we highlight the threshold 0.016 column with a darker green colour as it achieves 100% accuracy for the majority of devices (13). Hence, within these experiments, 0.016 emerges as the threshold that provides the most accurate results among these 19 device names.

Table 5.5: The F1 score values with different threshold values for 19 IoT products.

Threshold	0.01	0.012	0.014	0.016	0.018	0.02	0.022	0.024	0.026	0.028	0.03	0.032	0.034
Agshome smart alarm	0.67	0.67	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Atomtech smart life android app	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Cellinx IP camera	0.50	0.50	0.50	0.50	0.50	1.00	1.00	1.00	0.75	0.75	0.75	0.75	0.75
Eaton halo home app	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Fibaro Motion Sensor FGMS-001	0.67	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Glue Smart door Lock	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	1.00	0.57	0.57	0.57
Ismartgate garage door opener	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.57	0.57	0.57	0.57	0.57
Magic Home Pro app	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.33	0.33	0.33	0.33	0.33	0.33
Meross garage door opener MSG100	0.80	0.80	0.80	1.00	1.00	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
Mi xiaomi LED Desk Lamp	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.57
Nightowl smart doorbell	0.67	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
OpenHAB 2.5.11 app	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33
Sengled e1e-g7f Light switch	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33	0.33
Syska Smart Bulb	0.67	0.67	0.67	0.67	0.67	0.67	0.67	1.00	1.00	1.00	1.00	1.00	1.00
TaoLight Smart light Bulb	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Vivint SkyControl panel	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
WAFU Smart Lock	0.67	0.67	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Wemo Insight Smart Plug	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Wemo switch 28b	1.00	1.00	1.00	1.00	0.57	0.57	0.57	0.57	0.57	0.57	0.57	0.57	0.57

5.4.4 Statistics for IOC Classes in the Attack Datasets

Figure 5.2 illustrates the number of IOCs detected across attack datasets A1-A10, categorized into classes, such as *atomic*, *computed*, and *behavioral*. As previously discussed, we defined *atomic* IoT-specific IOCs as single graph nodes that present malicious activity. *Behavioral* IOC is one or more branches of the graph that includes the *atomic* IOC and preceding and subsequent IOCs to the incident. Lastly, *computed* IOC constitutes a full threat intelligence report produced by IoTINT about the incident. These statistics were collected only from the incident reports, the quantity of which varies depending on the nature of the attack, and are presented in Table 5.1.

Thus, the number of *computed* IOCs for each attack equals the number of processed incident occurrences by IoTINT and stays in the range from three to eight. The number of *behavioral* IOCs ranges from one to eight, while the count of *atomic* IOCs may vary from modest numbers under 10 to higher figures like 45 or 70. Such fluctuations are tightly related to the specifics of each attack

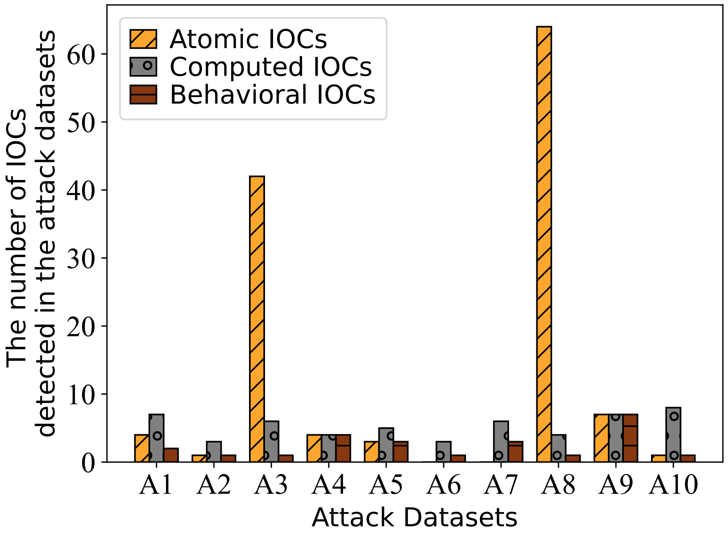


Figure 5.2: Number of different IOC types detected while generating threat intelligence for the attack datasets

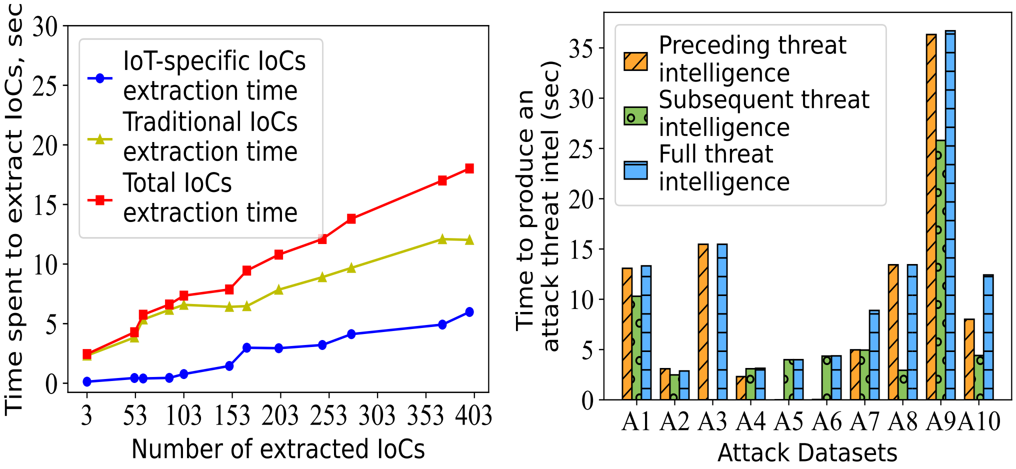
scenario. For instance, while some attacks may involve only a single malicious command, others may feature a multitude of malicious commands executed by a compromised application, such as altering the brightness of a light to convey a message via strobe patterns.

5.5 Performance Overhead

In this section, we present the results of experiments on measuring the performance overhead of IoTINT, such as time, storage and memory overhead.

5.5.1 Time Overhead

For this experiment, we consider that each device event in a typical dataset with more than 3400 events is potentially a security incident, and we intend to generate threat intelligence for each of these incidents. IoTINT requires around four hours to complete the threat generation for 3,400 incidents. While this may seem lengthy, it's important to note that in practical situations, the percentage of incidents amongst all observed events is much less, as we took the most extreme case. For instance, if incidents constitute only 1% of all events in a typical dataset, IoTINT would be required to generate threat intelligence for 34 incidents, which will take around 2.5 minutes.



(a) Time overhead

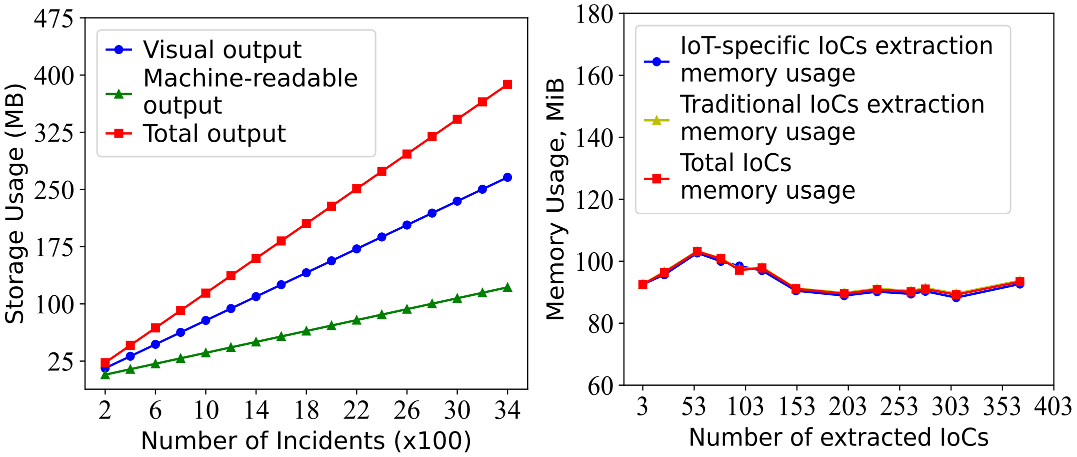
(b) Time for malicious datasets

Figure 5.3: Time overhead of IoTINT in threat intelligence generation

Figure 5.3a illustrates the time required for extracting IOCs for the reports consisting of 3 to 403 IOCs. The figure demonstrates that IoT-specific IOCs can be extracted more quickly than traditional IOCs. Specifically, the extraction time for IoT-specific IOCs varies from 1 to 6 seconds, while traditional IOCs take from 3 to 13 seconds, depending on the report size. Overall, the total time to generate threat intelligence for a single smart home incident ranges from 3 to 18 seconds.

In Figure 5.3b, we provide a breakdown of the time IoTINT requires to produce attack-related threat intelligence for ten different attack datasets labelled as A1-A10. Each attack dataset contains an observed incident, whose description and a number of occurrences are provided in Table 5.1. To ensure comprehensive threat intelligence generation for each attack, IoTINT must generate reports for every instance of an incident within a dataset.

In addition, security professionals may be interested in understanding only the cause of the incident, meaning attack evidence that appeared before the incident. Alternatively, they might be concerned with only the consequences of the attack, meaning the attack-related behavior happened after the incident. Thus, Figure 5.3b displays the time IoTINT takes to produce the preceding and subsequent to the incident threat intelligence with yellow and green colours, respectively. The blue bar illustrates the overall time spent producing complete threat intelligence for attack datasets ranging from A1 to A10. Note that for certain attack datasets, IoTINT exclusively generated either preceding or subsequent threat intelligence, indicating the absence of attack evidence in the opposite



(a) Storage overhead

(b) Memory overhead

Figure 5.4: Resource overhead of IoTINT in threat intelligence generation

direction. Additionally, the time spent to produce preceding, subsequent and full threat intelligence varies from 2 to 32 seconds, 2 to 26 seconds and 3 to 37 seconds, respectively. Notably, the summary of time spent to produce preceding and subsequent threat intelligence does not equal the time of full threat intelligence due to the separate extraction of traditional IOCs, which requires more time, as depicted in Figure 5.3a. Attack datasets A1, A3, and A9 show longer processing times for extracting threat information compared to other datasets due to the larger report sizes detailed in Table 5.1. Specifically, for the datasets A1, A3, and A9, IoTINT generated reports with an average of 90-142, and 320 recovered nodes, while other datasets' report sizes are smaller.

5.5.2 Storage Overhead

Figure 5.4a provides an overview of the storage cost incurred by IoTINT's report based on the number of processed incidents. When the tool generates a machine-readable report as a .txt file, it consumes approximately 120 MB of storage for 3,400 incidents. On the other hand, the graph report presented as a .pdf file requires 260 MB of storage for the same number of incidents. Combining both report types, namely the machine-readable and visual formats, results in a total storage usage of 360 MB for 3,400 incidents.

5.5.3 Memory Overhead

Figure 5.4b illustrates the typical runtime memory consumption of IoTINT depending on the produced report size. The memory overhead of IoTINT is displayed for incidents with varying final report sizes. Notably, our tool demonstrates memory efficiency, utilizing less than 105 MiB of memory for various reports consisting of 3 to 403 nodes. Specifically, the algorithms that separately extract only IoT-specific IOCs, traditional IOCs or both use a very similar amount of memory, maintaining a consistent memory footprint.

5.6 Feasibility of Manual Efforts Required by IoTINT

To assess the practicality of the manual effort needed for our approach in Section 3.4, a study is conducted involving nine participants comprising Master's and PhD students from diverse cybersecurity research backgrounds within our lab. The study involves four sets of questions, each containing 20 questions. Each question presents a unique product name (e.g., Cellinx IP camera) and requires participants to select the most relevant CPEs corresponding to that product. In addition, we assume that the product name consists of a vendor (e.g., Cellinx), name (e.g., IP camera), and any additional information such as version, update, edition, etc. Moreover, each question presents five CPE options and a *None of them* choice, indicating that none of the given CPEs might match the product name. We manually establish ground truth by individually verifying and cross-referencing each CPE option against online sources to ensure its alignment with the represented product. In addition, before completing the tasks, participants receive brief instruction on CPE functionality and structure due to their limited prior knowledge in this domain. Note that all questions vary from each other by the unique product name, and accordingly, the CPE options that are provided by the NLP approach are different from question to question.

Figure 5.5 illustrates the results of this feasibility study. For each question in a set, we measure the correctness of the answer and the time spent, from the moment the participant saw the question for the first time to the switching to the next question. The green line depicts the first attempt, where participants encountered unfamiliar questions. Mistakes were made in half of the questions, and completion times ranged between 25 to 50 seconds per question. The purple line reflects the

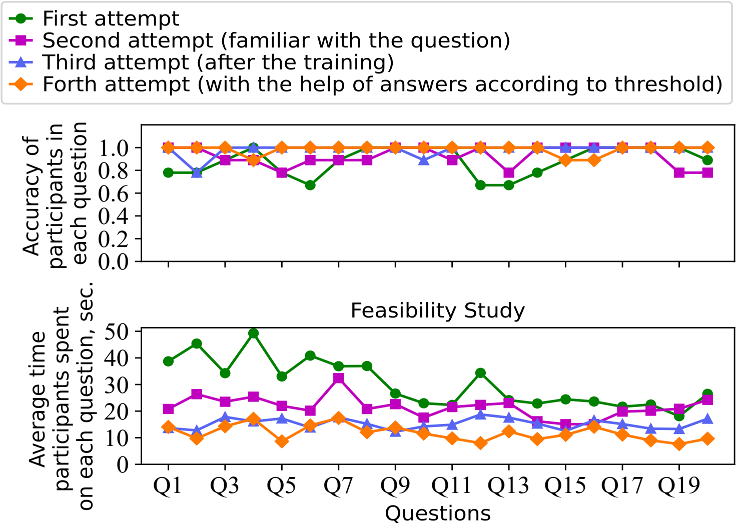


Figure 5.5: Average time and accuracy achieved by the participants across four attempts in mapping the CPEs to product names while the tasks are varied among those attempts

second attempt, indicating reduced completion times (15-30 seconds) as participants became more acquainted with the question patterns. However, the average accuracy remained at 10 out of 20 questions with errors. After the second attempt, we trained participants to identify relevant CPEs correctly and pointed out details to pay attention to. Following training on CPE identification, the blue graphs represent the third attempt, showing substantial accuracy improvement with errors in only two questions. Participants also became more efficient, spending 10 to 20 seconds per question. Finally, in the fourth set of questions, we used the product names from previous threshold experiments and correct CPE options were highlighted based on a threshold of 0.016. Additionally, participants were advised about the limitations of threshold results, emphasizing that they should rely on them only to a certain extent. Thus, the orange graphs show that accuracy remained consistent with the third attempt, and completion time decreased to around 10 seconds. As a result, participants noted that the highlighted options aided them in identifying correct answers efficiently. Further, students rated the task complexity at an average of 2 on a 1 to 5 scale. Totally the accuracy in completing such tasks is expected to be higher, but we are showing extreme cases where participants are not familiar with the topic. Overall, iterative training significantly improved accuracy and reduced effort, demonstrating the feasibility of mapping CPEs to product names for security professionals.

Chapter 6

Case Studies

This chapter presents the usability of IoTINT through two case studies.

Our proposed solution might be useful for various contexts, including vulnerability assessment, incident response and mitigation, security risk management, etc. To illustrate its practicality, we present two case studies involving distinct smart home scenarios and their corresponding incidents.

6.1 Case Study 1

In this scenario, the smart home was robbed while the homeowners were on vacation. During the video check from camera footage, security experts discovered that the backyard smart camera had been switched off. Consequently, they reported the incident about the camera's unexpected power-off state to our tool. Our solution generated output in the STIX format, a snippet of which is shown in Figure 6.1. This output includes IoT-specific IOCs detailing the smart home's behavior and traditional IOCs that outline known vulnerabilities, weaknesses and attack patterns of the IoT products. According to information provided by our solution, the homeowners had set the house to vacation mode after leaving. Notably, the "blink for home" smart app received this event and maliciously powered down the smart camera precisely 15 minutes after the users left. Our solution also pinpointed a vulnerability in this smart app, identified as CVE-2018-20161.

Given the wealth of threat intelligence platforms and feeds containing previously observed threat information and response procedures, it is prudent to check if a similar scenario has already been documented. Our solution facilitates this process by providing output in STIX, the most used threat intelligence feed format. In this particular case study, such suspicious smart home behavior has been observed repeatedly as a Threat A in the figure. Specifically, the malicious command to turn off the camera always occurs 15 minutes after switching the home to vacation mode. Furthermore, the app “blink for home” responsible for this command is identified as misconfigured because of the exploited vulnerability. Thus, if all the indicators extracted by our tool match the documented indicators about threat A, we assume that this is a known attack. It is important to note that organizations have the flexibility to establish their own criteria for matching the STIX output of our tool with previously observed attack data. For instance, they may employ a similarity score with a specific threshold to determine which threats are considered matches. Considering Threat A, it has associated Response Procedures A, including 1) manually turn on the cameras, 2) stop using the misconfigured app “blink for home” they serve as a fixing option. Thus, these response procedures are sent to the homeowner as soon as the threat match is found to assist with getting the smart environment back into the secured state. Thus, this scenario exemplifies the real-world application of our solution in the context of incident response and mitigation.

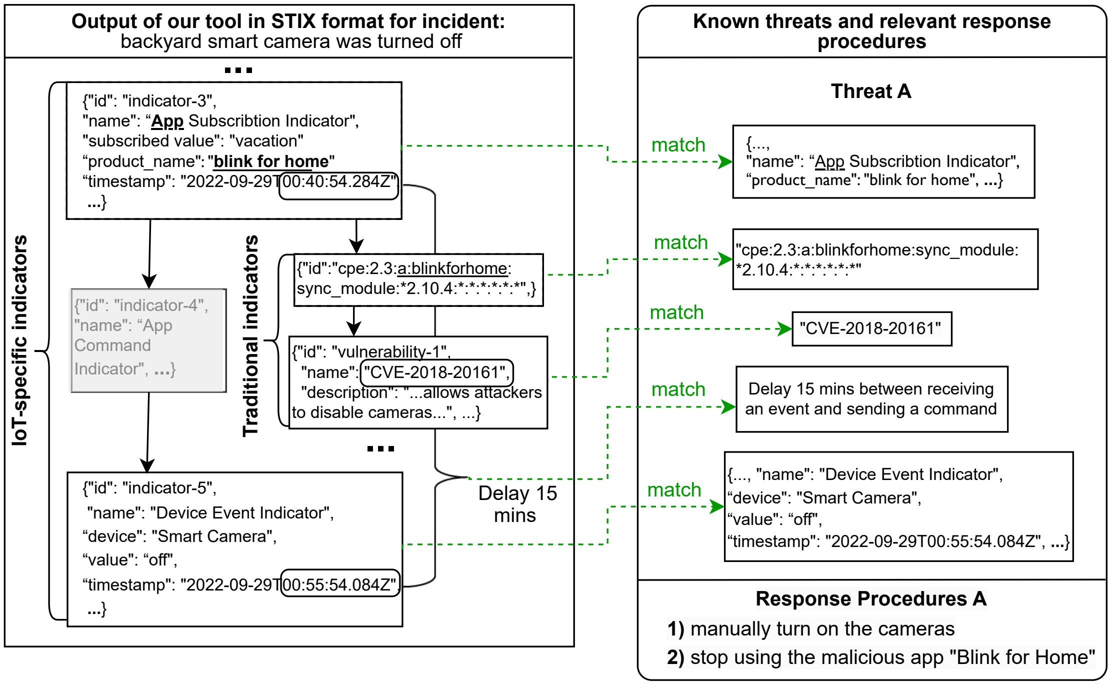
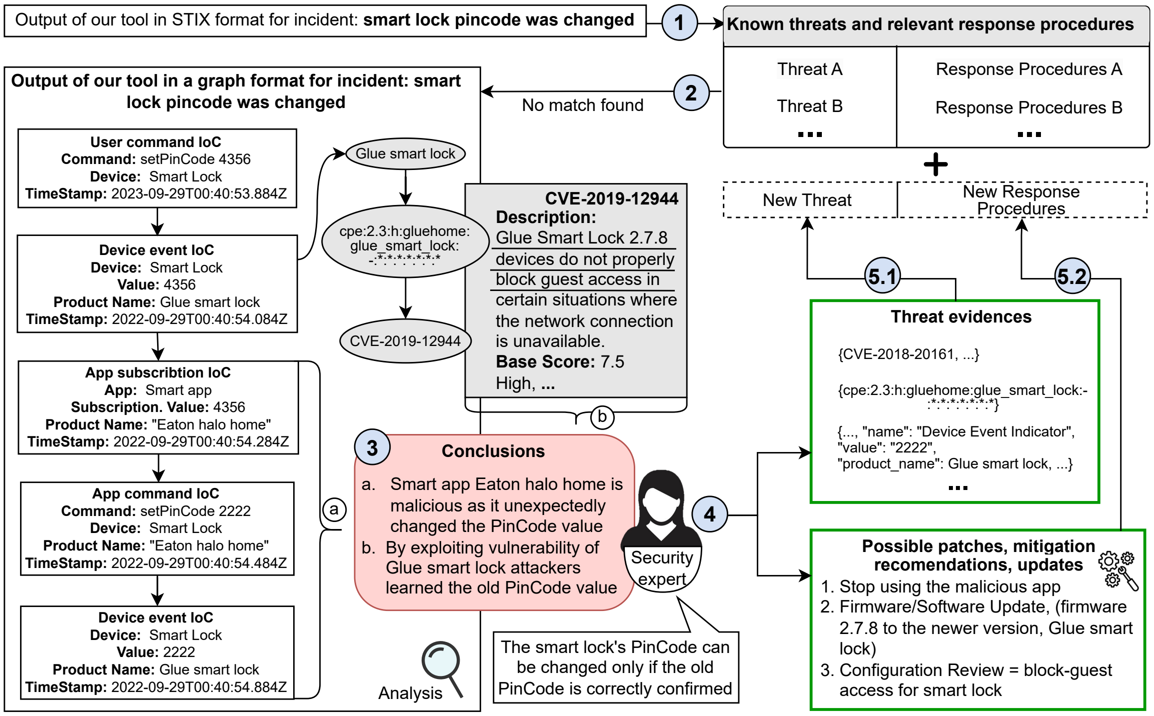

Output of our tool in STIX format for incident:
backyard smart camera was turned off
...
IoT-specific indicators
{"id": "indicator-3",
"name": “App Subscribtion Indicator",
"subscribed value": "vacation"
“product_name": "blink for home"
“timestamp": "2022-09-29T00:40:54.284Z"
...}
{"id": "indicator-4",
"name": “App Command Indicator", ...}
Traditional indicators
{"id":"cpe:2.3:a:blinkforhome:sync_module:*2.10.4:*:*:*:*:*:*",}
{"id": "vulnerability-1",
"name": "CVE-2018-20161",
"description": "...allows attackers to disable cameras...", ...}
...
{"id": "indicator-5",
"name": "Device Event Indicator",
“device": "Smart Camera",
“value": “off",
“timestamp": "2022-09-29T00:55:54.084Z"
...}
Delay 15 mins
Known threats and relevant response procedures
Threat A
match
{...,
"name": “App Subscribtion Indicator",
“product_name": "blink for home", ...}
match
"cpe:2.3:a:blinkforhome:sync_module:*2.10.4:*:*:*:*:*:*"
match
"CVE-2018-20161"
match
Delay 15 mins between receiving an event and sending a command
match
{..., "name": "Device Event Indicator",
“device": "Smart Camera",
“value": “off",
“timestamp": "2022-09-29T00:55:54.084Z", ...}
Response Procedures A
1) manually turn on the cameras
2) stop using the malicious app "Blink for Home"

Figure 6.1: Incident response for known threats

6.2 Case Study 2

In the second scenario, a smart home user sets a PIN for the smart lock to “4356.” However, s/he encounters an issue when attempting to access the house because the smart lock device gives an error due to the incorrect PIN. Such an incident prevents the owner from accessing the house and gives the potential access to the attackers. The initial suspicion is that the PIN may have been maliciously altered. Consequently, a security expert reports the incident of the PIN being changed to an unexpected value to our tool. Similar to the previous scenario, our tool generates an output in STIX format, providing information on relevant smart home events tied to the incident, along with potentially involved device/app vulnerabilities. Subsequently, a check is conducted to determine whether a similar attack scenario has been documented in the available threat intelligence platforms/feeds. In this use case, no matching threat records are found, indicating the possibility of a new threat that requires manual investigation by the security expert. Our tool offers a visual graph representation of the IoT-specific and traditional IOCs related to the incident to support the security professional.



Output of our tool in STIX format for incident: smart lock pincode was changed
1
Known threats and relevant response procedures
Threat A
Threat B
...
Response Procedures A
Response Procedures B
...
2
No match found
+
New Threat
New Response Procedures
Output of our tool in a graph format for incident: smart lock pincode was changed
User command IoC
Command: setPinCode 4356
Device: Smart Lock
TimeStamp: 2023-09-29T00:40:53.884Z
Device event IoC
Device: Smart Lock
Value: 4356
Product Name: Glue smart lock
TimeStamp: 2022-09-29T00:40:54.084Z
App subscribtion IoC
App: Smart app
Subscription. Value: 4356
Product Name: "Eaton halo home"
TimeStamp: 2022-09-29T00:40:54.284Z
App command IoC
Command: setPinCode 2222
Device: Smart Lock
Product Name: "Eaton halo home"
TimeStamp: 2022-09-29T00:40:54.484Z
Device event IoC
Device: Smart Lock
Value: 2222
Product Name: Glue smart lock
TimeStamp: 2022-09-29T00:40:54.884Z
Glue smart lock
cpe:2.3:h:gluehome:glue_smart_lock:-:*:*:*:*:*:*:*
CVE-2019-12944
CVE-2019-12944
Description:
Glue Smart Lock 2.7.8 devices do not properly block guest access in certain situations where the network connection is unavailable.
Base Score: 7.5
High, ...
a
b
Analysis
3
Conclusions
a. Smart app Eaton halo home is malicious as it unexpectedly changed the PinCode value
b. By exploiting vulnerability of Glue smart lock attackers learned the old PinCode value
Security expert
The smart lock's PinCode can be changed only if the old PinCode is correctly confirmed
4
5.1
5.2
Threat evidences
{CVE-2018-20161, ...}
{cpe:2.3:h:gluehome:glue_smart_lock:-:*:*:*:*:*:*:*}
{..., "name": "Device Event Indicator", "value": "2222", "product_name": Glue smart lock, ...}
...
Possible patches, mitigation recomendations, updates
1. Stop using the malicious app
2. Firmware/Software Update, (firmware 2.7.8 to the newer version, Glue smart lock)
3. Configuration Review = block-guest access for smart lock

Figure 6.2: Vulnerability assessment for unknown threats

Figure 6.2 illustrates the graph, showing that after the user initially sets the PIN to “4356,” the smart app subscribed to this event subsequently sends a “setPinCode” command with a value of “2222”. Note that the security expert is aware that to define a new PIN, the attackers must correctly insert the value of the old one. Thus, analyzing the part of the graph with the traditional indicators, the specialist learns that the Glue smart lock device is vulnerable through guest access, indicated by the grey oval.

In the outcome of the graph inspection, the security professional concludes that *(a)*: smart app Eaton Halo Home is misconfigured as it unexpectedly changed the PinCode value and *(b)*: the attackers learned the old PinCode value by exploiting the vulnerability of the Glue smart lock device. Using this detailed knowledge about the attack, the security expert can now define the crucial evidence that identifies the presence of the attack and add it to the observed threats database. Likewise, s/he produces and publishes possible patches, mitigation recommendations or updates as response procedures for this attack. Consequently, the second scenario underscores our solution’s practicality in the Vulnerability Assessment context. Overall, by introducing these use cases, we illustrate the applicability of our solution across various contexts. Moreover, we underscore the value of graphical representations and machine-readable formats as essential outputs for further dealing with security threats in IoT environments.

Chapter 7

Discussion

This chapter discusses some of IoTINTs' considerations.

Applicability of IoTINT to Other IoT ecosystems. IoTINT needs some adaptation to work in practice with other IoT platforms, such as openHAB, Home Assistant, and AWS IoT Core. IoTINT operates on the uniform database of potential attack-related evidence using the mapping rules to identify connectivity between IOCs. Furthermore, IoTINT utilizes predefined device and app filters to transform raw logs into potential IOCs. Thus, IOC mapping rules and predefined filters are specific to each platform and require manual analysis based on corresponding documentation and log formats. Nevertheless, this is a one-time task, and afterwards, IoTINT will produce IoT-specific threat intelligence like the current SmartThings prototype.

Integration of Traditional and IoT-specific IOCs. Combining IoT-specific and traditional IOCs allows to provide the smart environment behavior related to the incident along with the knowledge about devices/apps vulnerabilities participating in it. Such a combination gives critical information to security analysts. When IoT-specific IOCs indicate malicious activity, but traditional IOCs reveal no vulnerabilities, it suggests the potential presence of a zero-day vulnerability. Conversely, if no malicious behavior is evident in IoT-specific IOCs but vulnerabilities exist in devices or apps, it prompts security analysts to consider device or app updates or implement additional protective measures. Furthermore, the coexistence of both malicious activity and vulnerabilities in collected IOCs indicates that attacks could have been achieved by exploiting vulnerabilities. In summary, by combining IoT-specific and traditional IOCs, threat intelligence reports become more comprehensive and insightful, providing a deeper understanding and aiding in effective decision-making. However, the gathered IoT-specific and traditional IOCs information is not leveraged for any further intelligence generation. We consider it as a limitation and plan to address it in future work. In addition, as of now, IoTINT gathers traditional IOCs, such as CPEs, CVEs, CWEs, and CAPECs. This list can potentially be extended to TTPs as future work to provide more information for the security experts.

Chapter 8

Related Work

This chapter reviews both IoT-specific and traditional threat intelligence generation solutions and compares them with ours.

8.1 IoT-specific Solutions

Most existing works focus on collecting IoT-related attack artifacts and threat intelligence on a large scale. For instance, Pour et al. [10] and Shaikh et al. [24] leverage internet-scale network telescope data to capture traffic from compromised IoT devices engaged in malicious activities. Furthermore, the authors extract threat intelligence about malformed devices such as geolocation, associated domain names, organizational affiliations, and hosting environments, together with unique attack patterns and traffic captured from compromised devices. On the other hand, Khoury et al. [25] propose an innovative solution that combines telescope and honeypot methodologies to collect a diverse range of malware artifacts from IoT devices. These artifacts encompass system commands, evidence of file-less attacks, URLs housing payloads, executable and linkable format (ELF) binaries, and harvested login credentials. Works that implement honeypot systems [14, 26, 28, 60] focus on the analysis of communication exchanges between the honeypots and compromised IoT devices. This approach enables identifying and retrieving significant malware artifacts, such as unique attack vectors, command and control (C&C) methods, issued commands, login attempts, and downloading malware binaries utilized by compromised IoT devices. Meanwhile, Koloveas et al. [12] employ machine learning techniques to systematically gather IoT-centric Cyber Threat Intelligence (CTI) from various web domains, including the clear, social, and dark web. Similarly, Al-Hawawrch et al. [13] and Kumar et al. [27] leverage deep learning algorithms to extract IoT threat patterns and their types, ranging from backdoors, Distributed Denial-of-Service (DDoS) attacks, ransomware, scanning activities, injection attempts, etc. Moreover, MUDscope [11] introduces an approach tailored to monitoring malicious network activities impacting IoT systems within real-world consumer settings. This work identifies attack signatures associated with different devices and derives insights into emerging attack patterns by analyzing these signatures.

8.2 Traditional Solutions

In contrast, CTI is the traditional solution for collecting, processing and analyzing information about threat actors' motives, targets, and attack behaviors, which is not specializing in IoT. Many studies delve into CTI mining, targeting diverse threat information types with specific objectives. Some research focuses on cybersecurity-related entities and events sourced from hacker forums, Twitter, or cybersecurity articles, encapsulating impacted organizations, locations, vulnerabilities, and attack categories like phishing, DDoS, and hijacking [61–66]. Others leverage machine learning to extract tactics, techniques, and procedures (TTPs) and hacker profiles from forums and CTI reports [67–73]. Many studies involve extracting Indicators of Compromise (IOCs), such as IPs, signatures, hashes, or malware evidence, aiming to detect malicious or potentially harmful activities [21–23, 74]. Some of these studies delve further, employing Natural Language Processing (NLP) to establish relationships between extracted IOCs. Another category of works concentrates on discovering product/service vulnerabilities, predicting exploits, and unearthing malware details [75–78]. Further studies emphasize CTI for threat hunting, aiding in identifying unknown or ongoing threats within organizational networks [79–82]. These investigations source CTI data from various outlets like hacker forums, the dark web, CTI reports, audit logs, and intrusion alerts. Despite the efficacy of CTI in traditional cybersecurity, its application to IoT environments still remains limited.

8.3 Comparison between Related Works

Table 8.1 summarizes the comparison between existing works and IoTINT. The first and second columns enlist existing works and methodologies employed to harvest IoT threat intelligence. Note that presented approaches (e.g., telescope, honeypot) highlight the gathering of the network traffic-related IOCs instead of the data collection technique. The next two columns compare these works according to the threat scope. The scope of malformed IoT devices is checked when a solution collects threat intelligence artifacts about malware-infected IoT devices. When a solution gathers threat evidence about malicious smart apps, we check the scope of malformed smart apps. In addition, we use a dash symbol on the threat scope if the work is not applicable to that category. The next three columns compare the presented works according to the IOC coverage. Depending on the types of IOCs present in the extracted threat information (e.g., atomic, computed, behavioral), we mark those types in the table. In the last three columns of the table, we compare the works based on the sources of threat intelligence (sources of TI). Specifically, the presented works extract attack-related artifacts from such sources as network traffic, smart device/app interactions or OS.

In summary, IoTINT mainly differs from the state-of-the-art works as follows. Firstly, it stands out as one of the few approaches that gather threat intelligence about both malformed IoT devices and smart apps, along with two other works that used deep learning methods. Secondly, IoTINT is the only solution that incorporates all three classes of IOCs within the produced threat intelligence. Finally, unlike alternative solutions, IoTINT is the only solution to extract threat-related information from the device-app interactions inside the smart environment; which provides a new perspective to this literature. Thus, we consider IoTINT complementary to the other state-of-the-art solutions (that gather threat intelligence from the outbound to the smart environment network traffic). However, IoTINT still has some limitations. It needs one-time adaptation effort when applied to other IoT platforms (as discussed in Chapter 7). IoTINT mainly addresses attacks arising from device-app communication, and relying on existing works for the network-related threats like scanning, flooding, DDoS, and botnet activities.

Table 8.1: Comparing existing solutions with IoTINT. The symbols (•), (○) and (-) mean supported, not supported and not applicable, respectively.

Proposals	**Approach**	**Threat Scope**		**IOC Coverage**			**Sources of TI**		
		Malformed IoT Devices	Malformed Smart Apps	Atomic	Computed	Behavioral	Network Traffic	Smart Device/App Interactions	OS
Shaikh et al. [24]	Telescope	●	○	●	●	○	●	○	○
eX-IoT [10]	Telescope	●	○	●	○	○	●	○	○
Khoury et al. [25] HoneyComb [26]	Telescope	●	○	●	●	○	●	○	○
MUDscope [11]	Telescope + MUD profiles	●	○	○	○	●	●	○	○
Koloveas et al. [12]	ML/DL	-	-	○	○	●	●	○	○
All-Hawawareh et al. [13] DLTIF [27]	ML/DL	●	●	○	○	●	●	○	○
ATLAS [28] Tambe et all. [14]	Honeypot	●	○	●	●	○	●	○	○
Tabari et all. [29]	Honeypot	●	○	●	○	○	●	○	○
Honware [60]	Honeypot	●	○	●	○	●	●	○	●
IoTINT	Custom provenance-based	●	●	●	●	●	○	●	○

Chapter 9

Conclusion

In this dissertation, we proposed IoTINT, a practical framework designed to obtain IoT-specific threat intelligence about threats arising from device-app interactions within a smart environment. It iteratively extracted IoT-specific IOCs and their chronological connectivity before and after the security incident. Furthermore, it combined traditional IOCs with IoT-specific to provide enhanced insights into threats. We designed a prototype of IoTINT for the Samsung SmartThings platform and evaluated its performance on 10 realistic IoT attack scenarios and typical smart environment behaviors. The findings demonstrate that IoTINT provides comprehensive coverage in extracting IoT threat intelligence with minimal overhead.

However, IoTINT has some limitations. First, IoTINT needs some adaptation while applying it to another IoT platform. In future work, we potentially target to automate this process while minimizing manual efforts by utilizing machine learning and other modern approaches. Second, IoTINT exclusively addresses attacks arising from device-app communication, overlooking network-related threats like scanning, flooding, DDoS, and botnet activities. However, existing state-of-the-art works cover network-related threats, and we may combine their approaches with our work to develop a more universal solution in terms of threat coverage. Finally, this work mainly focuses on a rule-based approach to retrieve and connect incident-related IoT-specific IOCs. As shown in Table 5.4, the rule-based approach is needed, and only an ML-based approach might not be sufficient. At the same time, we acknowledge that an additional ML-based approach might complement the current version of IoTINT in terms of automation and coverage; which will be explored in future.

Appendix

A.1 Machine-readable Output

```
{
  type: bundle,
  id: bundle--12345678-XXX,
  spec_version: 2.1,
  objects: [
    {type: indicator,
     id: indicator--8e74baaa-XXX,
     created: 2023-08-15T14:30:00Z,
     modified: 2023-08-15T14:30:00Z,
     name: Device Event Indicator,
     description: Smart Garage Door is
     opened,
     device: Smart Garage Door,
     value: opened,
     timestamp: 2022-09-29T00:40:54.084Z,
     product_name: GetNexx NXG-100B,
     labels: [iot, device-event],
    },
    {
```

```
  type: indicator,
  id: indicator--8e744bbb-XXX,
  created: 2023-08-15T14:30:00Z,
  modified: 2023-08-15T14:30:00Z,
  name: App Subscribtion Indicator,
  description: Smart App received device
  event Garage Door is opened,
   app: Malicious smart app,
   subscribtion_value: opened,
   timestamp: 2022-09-29T00:40:54.284Z,
  product_name: Home assistant android,
  labels: [iot, app-subscribtion],
},
{
  type: indicator,
  id: indicator--8e744ccc-XXX,
  created: 2023-08-15T14:30:00Z,
  modified: 2023-08-15T14:30:00Z,
  name: App Command Indicator,
  description: Smart App sent turn off
  command to the Smart Camera,
   app: Malicious smart app,
   device: Smart Camera,
   command: off,
   timestamp: 2022-09-29T00:40:54.484Z,
  product_name: Home assistant android,
  labels: [iot, app-command],
},
      {
        type: relationship,
        id: relationship--98765411-XXX,
        created: 2023-08-02T10:00:00Z,
        relationship_type: was_received_by,
        source_ref: indicator--8e74baaa-XXX,
        target_ref: indicator--8e744bbb-XXX
      },
      {
        type: relationship,
        id: relationship--98765422-XXX,
        created: 2023-08-02T10:00:00Z,
        relationship_type: causing,
        source_ref: indicator--8e744bbb-XXX,
        target_ref: indicator--8e744ccc-XXX
      }
    ]
  }
```

Listing 1: An example of IoT-specific IOCs in STIX format

A.2 Mapping Rules

In this section, we provide the rest of the four mapping rules (Rules 2,3,5,6) used in Step 2 of our methodology (in Section 3.3) to derive the connectivity between apps and devices related to a security incident.

Rule 5: app subscription →app command (subsequent direction):

$$\begin{aligned} IOC_3 = IOC_4 \iff & IOC_3\{authToken\} = IOC_4\{authToken\} \land \\ & IOC_3\{rowNumber\} < IOC_4\{rowNumber\} \land \\ & IOC_i \in IOCs \end{aligned} \tag{5}$$

where IOC_3, and IOC_4 represent the `app subscription` IOCs, `app command` IOCs, respectively, and IOC_i represents a log entry from the log collection, $IOCs$. IOC_4 illustrates a set of sent commands by the smart app after receiving a device event. According to this rule, the IOC_4 can be mapped to the IOC_3 if their `authToken` fields are equal and the `rowNumber` field value of IOC_4 is more than IOC_3.

Rule 6: app command →device event (subsequent direction):

$$\begin{aligned} IOC_4 = IOC_5 \iff & IOC_4\{deviceId\} = IOC_5\{deviceId\} \land \\ & IOC_4\{capability\} = IOC_5\{attribute\} \land \\ & IOC_4\{command\} \subseteq IOC_5\{value\} \land \\ & |IOC_4\{timestamp\} - IOC_5\{timestamp\}| \leq 500ms, \\ & IOC_i \in logs \end{aligned} \tag{6}$$

where IOC_4, and IOC_5 represent the `app command`, and `device event` IOCs, respectively, and IOC_i represents a log entry from the log collection, $IOCs$. Specifically, IOC_5 shows the device event(s) that were published by the IoT platform as a result of the app command. According to this rule, the IOC_5 can be mapped to the IOC_4, if their `deviceId` field values are identical. Simultaneously, the IOC_4 `capability` field should match IOC_5 `attribute` field and IOC_4 `command` field should be a subset of the IOC_5 `value` field. In addition to the above conditions, these IOCs' timestamps (e.g., `timestamp` field) must differ by 500ms or less.

Rule 2: app command →app subscription (preceding direction):

$$
\begin{aligned}
IOC_4 = IOC_3 \iff & IOC_4\{appId\} = IOC_3\{appId\} \land \\
& IOC_4\{authToken\} = IOC_3\{authToken\} \land \\
& |IOC_4\{timestamp\} - IOC_3\{timestamp\}| \leq 5000ms, \\
& IOC_i \in logs
\end{aligned}
\tag{2}
$$

where IOC_4, and IOC_3 represent the `app command`, `app subscription` IOCs, respectively, and IOC_i represents a log entry from the log collection, $IOCs$. Specifically, IOC_3 shows a relevant app subscription because of which the command was sent to the platform. According to this rule, the IOC_3 can be mapped to the IOC_4 if their `appId` and `authToken` fields match and they occur within a specific period of time (e.g., 500ms).

Rule 3: app subscription →device event (preceding direction):

$$
\begin{aligned}
IOC_3 = IOC_1 \iff & IOC_3\{order\} = IOC_1\{order\} \land \\
& IOC_3\{deviceId\} = IOC_1\{deviceId\} \land \\
& IOC_3\{capability\} = IOC_1\{capability\} \land \\
& IOC_3\{value\} = IOC_1\{value\} \land \\
& IOC_3\{timestamp\} - IOC_1\{timestamp\} \leq 5000ms, \\
& IOC_i \in logs
\end{aligned}
\tag{3}
$$

where IOC_3, and IOC_1 represent the `app subscription`, `device event` IOCs, respectively, and IOC_i represents a log entry from the log collection, $IOCs$. Specifically, IOC_1 shows an original device event published by the IoT platform that the app received because of its subscription. The rule defines that IOC_1 can be mapped to IOC_3 if they occur within a specific period of time (e.g., 500ms) and other attributes mentioned in the rule are the same.

Bibliography

[1] Iot Devices Trend, 2023. URL `https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/`.

[2] attacksstatistics, 2023. URL `https://www.statista.com/statistics/1377569/worldwide-annual-internet-of-things-attacks/`.

[3] attacksstatistics, 2024. URL `https://www.veridify.com/77-percent-increase-in-malware-attacks-for-iot-connected-devices-in-1h22/`.

[4] Smartthings, 2023. URL `https://www.smartthings.com/`.

[5] AWS IoT, 2023. URL `https://aws.amazon.com/iot/`.

[6] Google IoT core, 2023. URL `https://cloud.google.com/iot-core/`.

[7] Openhab – an open-source platform for empowering home automation, 2023. URL `https://www.openhab.org/`.

[8] Jayasree Sengupta, Sushmita Ruj, and Sipra Das Bit. A comprehensive survey on attacks, security issues and blockchain solutions for IoT and IIoT. *Journal of Network and Computer Applications*, 149:102481, 2020.

[9] Prabhat Kumar, Randhir Kumar, Govind P Gupta, Rakesh Tripathi, and Gautam Srivastava. P2TIF: A blockchain and deep learning framework for privacy-preserved threat intelligence in industrial IoT. *IEEE transactions on industrial informatics*, 18(9):6358–6367, 2022.

[10] Morteza Safaei Pour, Dylan Watson, and Elias Bou-Harb. Sanitizing the IoT cyber security posture: An operational cti feed backed up by internet measurements. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 497–506. IEEE, 2021.

[11] Luca Morgese Zangrandi, Thijs Van Ede, Tim Booij, Savio Sciancalepore, Luca Allodi, and Andrea Continella. Stepping out of the MUD: Contextual threat information for iot devices with manufacturer-provided behavior profiles. In *Proceedings of the 38th Annual Computer Security Applications Conference*, pages 467–480, 2022.

[12] Paris Koloveas, Thanasis Chantzios, Christos Tryfonopoulos, and Spiros Skiadopoulos. A crawler architecture for harvesting the clear, social, and dark web for IoT-related cyber-threat intelligence. In *2019 IEEE World Congress on Services (SERVICES)*, volume 2642, pages 3–8. IEEE, 2019.

[13] Muna Al-Hawawreh, Nour Moustafa, Sahil Garg, and M Shamim Hossain. Deep learning-enabled threat intelligence scheme in the internet of things networks. *IEEE Transactions on Network Science and Engineering*, 8(4):2968–2981, 2020.

[14] Amit Tambe, Yan Lin Aung, Ragav Sridharan, Martín Ochoa, Nils Ole Tippenhauer, Asaf Shabtai, and Yuval Elovici. Detection of threats to IoT devices using scalable VPN-forwarded honeypots. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pages 85–96, 2019.

[15] Muslum Ozgur Ozmen, Xuansong Li, Andrew Chu, Z Berkay Celik, Bardh Hoxha, and Xiangyu Zhang. Discovering IoT physical channel vulnerabilities. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2415–2428, 2022.

[16] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, Atul Prakash, and SJ Unviersity. ContexIoT: Towards providing contextual integrity to appified IoT platforms. In *ndss*, volume 2, pages 2–2. San Diego, 2017.

[17] Qi Wang, Wajih Ul Hassan, Adam Bates, and Carl Gunter. Fear and logging in the internet of things. In *Network and Distributed Systems Symposium*, 2018.

[18] Daegeon Kim and Huy Kang Kim. Automated dataset generation system for collaborative research of cyber threat analysis. *Security and communication networks*, 2019:1–10, 2019.

[19] Jun Zhao, Qiben Yan, Jianxin Li, Minglai Shao, Zuti He, and Bo Li. TIMiner: Automatically extracting and analyzing categorized cyber threat intelligence from social data. *Computers & Security*, 95:101867, 2020.

[20] Hyeonseong Jo, Yongjae Lee, and Seungwon Shin. Vulcan: Automatic extraction and analysis of cyber threat intelligence from unstructured text. *Computers & Security*, 120:102763, 2022.

[21] Xiaojing Liao, Kan Yuan, XiaoFeng Wang, Zhou Li, Luyi Xing, and Raheem Beyah. Acing the IOC game: Toward automatic discovery and analysis of open-source cyber threat intelligence. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 755–766, 2016.

[22] Jian Liu, Junjie Yan, Jun Jiang, Yitong He, Xuren Wang, Zhengwei Jiang, Peian Yang, and Ning Li. TriCTI: an actionable cyber threat intelligence discovery system via trigger-enhanced neural network. *Cybersecurity*, 5(1):8, 2022.

[23] Zhenyuan Li, Jun Zeng, Yan Chen, and Zhenkai Liang. AttacKG: Constructing technique knowledge graph from cyber threat intelligence reports. In *European Symposium on Research in Computer Security*, pages 589–609. Springer, 2022.

[24] Farooq Shaikh, Elias Bou-Harb, Nataliia Neshenko, Andrea P Wright, and Nasir Ghani. Internet of malicious things: Correlating active and passive measurements for inferring and characterizing internet-scale unsolicited IoT devices. *IEEE Communications Magazine*, 56(9): 170–177, 2018.

[25] Joseph Khoury, Morteza Safaei Pour, and Elias Bou-Harb. A near real-time scheme for collecting and analyzing IoT malware artifacts at scale. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, pages 1–11, 2022.

[26] Morteza Safaei Pour, Joseph Khoury, and Elias Bou-Harb. HoneyComb: A darknet-centric proactive deception technique for curating IoT malware forensic artifacts. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2022.

[27] Prabhat Kumar, Govind P Gupta, Rakesh Tripathi, Sahil Garg, and Mohammad Mehedi Hassan. DLTIF: Deep learning-driven cyber threat intelligence modeling and identification framework in IoT-enabled maritime transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[28] Yan Lin Aung, Martín Ochoa, and Jianying Zhou. ATLAS: A practical attack detection and live malware analysis system for IoT threat intelligence. In *International Conference on Information Security*, pages 319–338. Springer, 2022.

[29] Armin Ziaie Tabari, Guojun Liu, Xinming Ou, and Anoop Singhal. Revealing human attacker behaviors using an adaptive internet of things honeypot ecosystem. In *IFIP International Conference on Digital Forensics*, pages 73–90. Springer, 2023.

[30] Cve: National vulnerability database, 2023. URL `https://nvd.nist.gov/vuln`.

[31] Cwe: Common weaknesses enumeration, 2023. URL `https://cwe.mitre.org/`.

[32] Capec: Common attack pattern enumeration and classification, 2023. URL `https://capec.mitre.org/`.

[33] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging smart home applications. In *2016 IEEE symposium on security and privacy (SP)*, pages 636–654. IEEE, 2016.

[34] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *Proceedings of the 11th ACM on Asia conference on computer and communications security*, pages 461–472, 2016.

[35] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. {6thSense}: A context-aware

sensor-based attack detector for smart devices. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 397–414, 2017.

[36] Bin Yuan, Yuhan Wu, Maogen Yang, Luyi Xing, Xuchang Wang, Deqing Zou, and Hai Jin. Smartpatch: Verifying the authenticity of the trigger-event in the IoT platform. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[37] SmartThings Open-Source Smart Apps, 2023. URL `https://github.com/SmartThingsCommunity/SmartThingsPublic`.

[38] IoC definition, 2023. URL `https://www.crowdstrike.com/cybersecurity-101/indicators-of-compromise/`.

[39] Antonio Villalón-Huerta, Ismael Ripoll-Ripoll, and Hector Marco-Gisbert. Key requirements for the detection and sharing of behavioral indicators of compromise. *Electronics*, 11(3):416, 2022.

[40] Mittreattack, 2024. URL `https://attack.mitre.org/`.

[41] Platfrom architecture, 2023. URL `https://developer.smartthings.com/docs/getting-started/architecture-of-smartthings`.

[42] Eryk Schiller, Andy Aidoo, Jara Fuhrer, Jonathan Stahl, Michael Ziörjen, and Burkhard Stiller. Landscape of iot security. *Computer Science Review*, 44:100467, 2022.

[43] CPE dictionary, 2024. URL `https://nvd.nist.gov/products/cpe#:~:text=CPE%20is%20a%20structured%20naming,and%20tests%20to%20a%20name`.

[44] jina-embeddings-v2-base-en pre-trained model, 2024. URL `https://huggingface.co/jinaai/jina-embeddings-v2-base-en`.

[45] Alfonso Iacovazzi, Han Wang, Ismail Butun, and Shahid Raza. Towards cyber threat intelligence for the IoT. In *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pages 483–490. IEEE, 2023.

[46] Smartthings developers, 2022. URL https://developer.smartthings.com/docs/getting-started/welcome/.

[47] Fiddler classic, 2022. URL https://www.telerik.com/fiddler/fiddler-classic.

[48] Smartthings groovy ide, 2022. URL https://graph.api.smartthings.com/.

[49] CPE API NVD, 2023. URL https://nvd.nist.gov/developers/products.

[50] CVE API NVD, 2023. URL https://nvd.nist.gov/developers/vulnerabilities.

[51] Jina pre-trained transformer model documentation, 2023. URL https://huggingface.co/jinaai/jina-embeddings-v2-base-en.

[52] FAISS library documentation, 2023. URL https://faiss.ai/index.html.

[53] anytree python library, 2024. URL https://anytree.readthedocs.io/en/latest/.

[54] Erik Hemberg, Jonathan Kelly, Michal Shlapentokh-Rothman, Bryn Reinstadler, Katherine Xu, Nick Rutar, and Una-May O'Reilly. Linking threat tactics, techniques, and patterns with defensive weaknesses, vulnerabilities and affected platform configurations for cyber hunting. *arXiv preprint arXiv:2010.00533*, 2020.

[55] Graphviz Python library, 2024. URL https://pypi.org/project/graphviz/.

[56] Node.js library, 2024. URL https://nodejs.org/docs/latest-v12.x/api/.

[57] Faiss library, 2024. URL https://github.com/facebookresearch/faiss.

[58] Leonardo Babun, Z Berkay Celik, Patrick McDaniel, and A Selcuk Uluagac. Real-time analysis of privacy-(un) aware IoT applications. *arXiv preprint arXiv:1911.10461*, 2019.

[59] Z Berkay Celik, Gang Tan, and Patrick D McDaniel. IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT. In *NDSS*, 2019.

[60] Alexander Vetterl and Richard Clayton. Honware: A virtual honeypot framework for capturing cpe and IoT zero days. In *2019 APWG symposium on electronic crime research (eCrime)*, pages 1–13. IEEE, 2019.

[61] Isuf Deliu, Carl Leichter, and Katrin Franke. Collecting cyber threat intelligence from hacker forums via a two-stage, hybrid process using support vector machines and latent dirichlet allocation. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5008–5013. IEEE, 2018.

[62] Rupinder Paul Khandpur, Taoran Ji, Steve Jan, Gang Wang, Chang-Tien Lu, and Naren Ramakrishnan. Crowdsourcing cybersecurity: Cyber attack detection using social media. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1049–1057, 2017.

[63] Nuno Dionísio, Fernando Alves, Pedro M Ferreira, and Alysson Bessani. Cyberthreat detection from twitter using deep neural networks. In *2019 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2019.

[64] Taneeya Satyapanich, Francis Ferraro, and Tim Finin. Casie: Extracting cybersecurity event information from text. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8749–8757, 2020.

[65] Yong Fang, Yuchi Zhang, and Cheng Huang. CyberEyes: cybersecurity entity recognition model based on graph convolutional network. *The Computer Journal*, 64(8):1215–1225, 2021.

[66] Hieu Man Duc Trong, Duc-Trong Le, Amir Pouran Ben Veyseh, Thut Nguyn, and Thien Huu Nguyen. Introducing a new dataset for event detection in cybersecurity texts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5381–5390, 2020.

[67] Ghaith Husari, Ehab Al-Shaer, Mohiuddin Ahmed, Bill Chu, and Xi Niu. Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of CTI sources. In *Proceedings of the 33rd annual computer security applications conference*, pages 103–115, 2017.

[68] Yiming Wu, Qianjun Liu, Xiaojing Liao, Shouling Ji, Peng Wang, Xiaofeng Wang, Chunming Wu, and Zhao Li. Price tag: towards semi-automatically discovery tactics, techniques and procedures of e-commerce cyber threat intelligence. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[69] Wenhan Ge and Junfeng Wang. SeqMask: Behavior extraction over cyber threat intelligence via multi-instance learning. *The Computer Journal*, page bxac172, 2022.

[70] Yizhe You, Jun Jiang, Zhengwei Jiang, Peian Yang, Baoxu Liu, Huamin Feng, Xuren Wang, and Ning Li. Tim: threat context-enhanced TTP intelligence mining on unstructured threat data. *Cybersecurity*, 5(1):3, 2022.

[71] Sagar Samtani, Ryan Chinn, Hsinchun Chen, and Jay F Nunamaker Jr. Exploring emerging hacker assets and key hackers for proactive cyber threat intelligence. *Journal of Management Information Systems*, 34(4):1023–1053, 2017.

[72] Umara Noor, Zahid Anwar, Tehmina Amjad, and Kim-Kwang Raymond Choo. A machine learning-based fintech cyber threat attribution framework using high-level indicators of compromise. *Future Generation Computer Systems*, 96:227–242, 2019.

[73] John Grisham, Sagar Samtani, Mark Patton, and Hsinchun Chen. Identifying mobile malware and key threat actors in online hacker forums for proactive cyber threat intelligence. In *2017 IEEE international conference on intelligence and security informatics (ISI)*, pages 13–18. IEEE, 2017.

[74] Ziyun Zhu and Tudor Dumitras. ChainSmith: Automatically learning the semantics of malicious campaigns by mining threat intelligence reports. In *2018 IEEE European symposium on security and privacy (EuroS&P)*, pages 458–472. IEEE, 2018.

[75] Arpita Roy, Youngja Park, and Shimei Pan. Predicting malware attributes from cybersecurity texts. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2857–2861, 2019.

[76] Yi Chen, Luyi Xing, Yue Qin, Xiaojing Liao, XiaoFeng Wang, Kai Chen, and Wei Zou. Devils in the guidance: predicting logic vulnerabilities in payment syndication services through automated documentation analysis. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 747–764, 2019.

[77] Yi Chen, Yepeng Yao, XiaoFeng Wang, Dandan Xu, Chang Yue, Xiaozhong Liu, Kai Chen, Haixu Tang, and Baoxu Liu. Bookworm game: Automatic discovery of LTE vulnerabilities through documentation analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1197–1214. IEEE, 2021.

[78] Aritran Piplai, Sudip Mittal, Anupam Joshi, Tim Finin, James Holt, and Richard Zak. Creating cybersecurity knowledge graphs from malware after action reports. *IEEE Access*, 8:211691–211703, 2020.

[79] Yali Gao, Xiaoyong Li, Hao Peng, Binxing Fang, and S Yu Philip. HinCTI: A cyber threat intelligence modeling and identification system based on heterogeneous information network. *IEEE Transactions on Knowledge and Data Engineering*, 34(2):708–722, 2020.

[80] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, Ramachandran Sekar, and VN Venkatakrishnan. Holmes: real-time APT detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1137–1152. IEEE, 2019.

[81] Sagar Samtani, Hongyi Zhu, and Hsinchun Chen. Proactively identifying emerging hacker threats from the dark web: A diachronic graph embedding framework D-GEF. *ACM Transactions on Privacy and Security (TOPS)*, 23(4):1–33, 2020.

[82] Azqa Nadeem, Sicco Verwer, Stephen Moskal, and Shanchieh Jay Yang. Alert-driven attack graph generation using S-PDFA. *IEEE Transactions on Dependable and Secure Computing*, 19(2):731–746, 2021.