5-2022

IOT Security for IOTMon Attacks Based on Devices’ app

Description

Raghad Jameel A. Alhazmi

Follow this and additional works at: https://repository.fit.edu/etd

Part of the Computer Sciences Commons
IOT Security for IOTMon
Attacks Based on Devices’ app Description

by
Raghad Jameel A Alhazmi

Bachelor of Science
Computer Sciences
University of Um Alqura
2017

A thesis submitted to the
College of Engineering and Science at
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Melbourne, Florida
May, 2022
The author grants permission to make single copies
We the undersigned committee hereby recommend that the attached document be accepted as fulfilling in part the requirements for the degree of Master of Science in Computer Information Systems.

“IOT Security for IOTMon Attacks Based on Devices’ app Description”
a thesis by Raghad Jameel A Alhazmi

Marius C. Silaghi, Ph.D.
Professor
Computer Science
Thesis Advisor

Siddhartha Bhattacharyya, Ph.D.
Associate Professor
Computer Engineering and Sciences
Committee Member

Veton Kepuska, Ph.D.
Associate Professor
Computer Engineering and Sciences
Outside Committee Member

Philip Bernhard, Ph.D.
Associate Professor and Head
Department of Computer Engineering and Sciences
Abstract

Title: IOT Security for IOTMon Attacks Based on Devices’ app Description

Author: Raghad Jameel A Alhazmi

Thesis Advisor: Marius C. Silaghi, Ph.D.

There are concerns associated with "inter-app" interactions, which occur when many independently developed home automation apps interact and affect one another, causing possibly dangerous unexpected app action. We extended a security tool named IoTMon, an IoT device management system capable of identifying all potential cross-app communication paths and analyzing their danger status. As part of our work, we keep an eye on the app description and safeguard IoTMon from being altered in any way that could obscure the real interaction related to another app action. We validate the IoTMon system’s integrity by applying the hash algorithm SHA512 with digital signature on the system files to ensure that it is dependable in controlling inter-app interactions. The extended IoTMon was able to identify and fix 22 out of 92 apps that their descriptions had been altered. Our evaluation indicates that the user will be notified of either purposefully or unintentionally modification in the IoTMon.
# Table of Contents

Abstract iii

List of Figures vi

List of Tables viii

Acknowledgments ix

Chapter 1 Introduction 1

Chapter 2 Background and Related Work 7

Chapter 3 Comparison between our extended IoTMon and other Iot apps 16

Chapter 4 IoTMON Architecture 22

4.1 Examination of Intra-app Processes ......................... 22

4.1.1 General Policy Structure ............................... 23

4.1.2 Intra-app Interaction Study .............................. 23
List of Figures

3.1 IoTGuard user console. . . . . . . . . . . . . . . . . . . . . . . . . . . 17
3.2 Illustration of events and action in the previous Figure (The light is switched on → change mode to home mode, and then one of the actions that occurs when it is home-mode is the door-unlock. . . . . . . . . . 18
3.3 extended IoTMon Vs. IoTGuard. . . . . . . . . . . . . . . . . . . . . 18
3.4 A visual representation of the specific policies of trigger-action platform. Unaccepted states are marked with X. . . . . . . . . . . . . . 19
3.5 ContexIoT System. [15] . . . . . . . . . . . . . . . . . . . . . . . . . 20
4.1 IoTMon System overview [10]. . . . . . . . . . . . . . . . . . . . . . 24
4.2 NLP algorithm for the description of application [10]. . . . . . . . . 25
4.3 An attacker change temperature control app description to make it look like lights-follows me app . . . . . . . . . . . . . . . . . . . . . . . . . . 29
5.1 Output of extended IoTMon that prompt the user regarding modifications applied to an app . . . . . . . . . . . . . . . . . . . . . . . . . . 32
5.2 The extended IoTMon cross-app interaction related to temperature channel .................................................. 32
5.3 The original IoTMon cross-app interaction after the modification attack becomes related to illumination channel ............... 32
5.4 Output of extended IoTMon that prompt the user regarding modifications applied to Curling Iron app .................................................. 35
5.5 The extended IoTMon Interaction chains after Curling-iron App ...... 36
5.6 The digital signature for the hash values. ................................. 39
5.7 Original Hash value of all IoTMon files. ....................................... 40
5.8 The new hash value that is computed every time IoTMon is used or updated. ................................................................. 41
5.9 The extended IoTMon code analysis ............................................... 43
A.1 The extended IoTMon output after discovering all modifications made to applications .................................................. 47
B.1 Full report on the extended IoTMon code ................................. 49
C.1 Digital signature of the hash file that has not been affected by any modification. ................................................................. 50
C.2 Digital signature of the modified hash file. ............................... 51
List of Tables

3.1 Comparison between extended IoTMon and other tools ........ 21

5.1 List of detected modified apps .............................. 31
Acknowledgements

I would like to express my special thanks of gratitude to my supervisor, Marius Silaghi, who gave me the opportunity, patience, and guidance to do this study.

I am forever thankful for the unconditional support of my husband throughout the process of researching and writing this thesis during difficult times. This accomplishment would not have been possible without him. Thank you.
Chapter 1

Introduction

Internet of Things (IoT) became an ubiquitous component of human habitat and their security is a challenging topic of research. Sensors, software, and other technologies are integrated into physical items, referred to as "things," on the Internet of Things (IoT). These "things" can communicate with one another, as well as with other devices and systems. There is no limit to what can belong to the Internet of Things, from everyday household items to high-tech industrial instruments are included. The addition of sensors and a network of interconnected objects gives previously dumb gadgets the ability to exchange real-time data without the involvement of a human being. More than one in five of the world’s IoT platforms are smart home platforms according to a research from 2017. Samsung SmartThings, Apple HomeKit, and Google Android Things are examples of these platforms. On the report of IHS Markit [33], there has been tremendous growth in the smart home consumer sector in the last several years to assist with the automatic control of home appliances (such
as, freezers, ovens, and washer/dryers), lights, air-conditioning systems, and different home security (such as, entrance sensors, alarms) and safety (such as, flood, freeze, and smoke monitors) [33].

In addition to serving individual needs, IoT also benefits the broader community. A wide range of smart devices are already servicing community-specific needs, such as monitoring surgical procedures in hospitals, detecting weather conditions, supplying monitoring and networking in transportation, and identifying animals using bio-chips [18]. IoT has many challenges. Particularly, security challenges have affected IoT ability to be included in all aspects of users lives. For instance, the Mirai malware, launched a major Distributed Denial of Service (DDoS) attack that brought the Internet to a complete halt by infecting over 600,000 IoT devices with known default credentials (usernames and passwords). It is challenging to prevent DoS and DDoS since the target is forced to slow down, crash, or shut down for a lengthy amount of time. It may potentially lead to long-term memory degradation in IoT systems due to the restricted resources of the relaying nodes. An adversary might undertake a multi-step attempt to get access to a local home network from the Internet by targeting a firmware issue and utilizing a malignant mobile application [27, 10]. In addition, it was discovered that a worm might exploit gaps in ZigBee’s communication protocols [10].

IoT devices can be used to threaten customers, but they are also used for national security purposes. Known as StuxNet, this worm attacked implanted controllers in nu-
clear centrifuges and forced them to spin excessively fast, resulting in the destruction of the centrifuges [20]. This study demonstrates that applications can produce new concerns such as Cross-App Interference risks, even when the authorization system rigorously adheres to the least privilege concept (as is the case with most permission systems). Because these threats are independent of the overprivilege problem, once we install a new app in the same smart environment to interact with one another, a variety of cross-app interference risks could be triggered [7].

Furthermore, IoT devices could be controlled in different ways if many third-party apps are downloaded into a smart house. Due to a mismatch between two apps’ interpretations of sensor values, it is possible for a smart door to open and close in the wrong direction, confusing users and posing major security risks [7]. Also, if one app is run on a smart device, it may start a chain of other apps, but not all of these chained apps are wanted by homeowners, and several could be dangerous or expose private information [7]. In addition, using physical abilities of IoT devices to compromise IoT environment have been looked into by some researchers, and they’ve shown that a hacked smart bulb might detect sensitive internal web data and leaking it using a sneaky flash [26]. People movement, for example, can alter the state of temp, humidity and motion sensors. Han et al. showed that to help consumers associate relevant devices it is essential to detect various physical effects generated by a particular event [13].

To overcome these challenges: researchers have proposed different frameworks
and tools. For example, Jia et al. developed a permission method that uses context data during the run time so that users can decide if a security-critical action will be executed. SmartAuth, identifies overprivilege in an app by comparing the app code analysis result with the code comments and app description, and lets users to establish the rules for access control [29]. Tyche creates a more flexible permissions system that categorizes commands based on risk, allowing users to choose a maximum degree of risk for each app they have installed. As a result, the program will not be able to execute if the stated permission level is exceeded [25].

There are features available in IoT applications that could put the individual’s safety and the ecosystems at danger such as unlocking doors while the user is away or shutting off the heater in the winter. Due to the unique properties of IoT systems, IoTMon inventor identifies a new sort of security issue that may arise. The capability of IoT devices to physically interact with their environment is one such property. Even though these IoT devices’ physical interactions may deliver good convenience to end users, they might perhaps be manipulated to threaten IoT settings. Because of the physical interaction qualifications, devices can communicate and interact with one other through common physical environments. It is possible that an IoT application that manages gadgets to control the physical environment could cause additional applications to be launched. An attacker might take advantage of unanticipated interactions if the program isn’t informed of all of its potential connections with other applications [10].
IoTMon proposes a framework that can ensure the safety of IoT systems by capturing all possible physical contacts between apps. As a first step in addressing issues that may arise as a result of unanticipated physical interactions, IoTMon uses static program analysis to extract application information such as triggers, devices, and actions that may be utilized to develop interactions within the app. In addition to the static analysis of applications, Natural Language Processing (NLP) approach is used in order to find physical channels in the IoT ecosystem by inspecting app descriptions and then constructing interaction chains between apps by connecting interactions inside the app via physical and system channels. After that, IoTMon implements a risk analysis approach to assess the dangers of all discovered cross-app interaction chains. This tool evaluation demonstrates that more than 160 SmartThings applications have been found to have concealed communication chains, and 37 of them are particularly hazardous since they might be taken advantage of by some adversary.

In this thesis, we extend IoTMon functionality to overcome aspects related to smart apps description. Specifically, we ensure that apps descriptions are monitored and prompt any users with changes made on these description. The goal is to make users aware of any suspicious behaviors or malicious attempts by an adversary. In summary, this thesis makes the following contributions:

- goal 1 Detect any modifications made to IoT apps descriptions.
- goal 2 Notify the user and correct the description with the original one.
- goal 3 Validating and protecting IoTMon integrity.
The reminder of this thesis is organized as follows: Chapter 2 explains background and related work. Chapter 3 introduces comparison between the extended IoTMon and other tools. Chapter 4 provides details about IoTMon architecture ended by the problem description. The evaluation and the case study are described in Chapter 5. In chapter 6, we concluded our thesis.
Chapter 2

Background and Related Work

As described in [9], many IoT architectures were easily attacked with denial of service (DOS) attacks. Internet of Things (IoT) technologies have advanced rapidly in the past few years, and many of them have already been used widely. Smart plugs, smart door locks, smart lamps, and a variety of other devices are examples. IoT ecosystems have piqued the interest of academics and of the information business, both of which are grappling with all kinds of security issues. IoT platforms’ physical interaction features permit IoT devices to interact with the physical world. Such characteristics, however, introduce new security risks, since attacks on IoT systems are based on physical interactions that attackers utilize. Traditional networks can now be secured using a variety of security techniques including cryptography mechanisms. However, due to the restricted capabilities of IoT devices, these precautions are frequently ineffective [14].

There are three primary components to the SmartThings ecosystem, which include
smartphone application, the SmartThings storage and processing on the cloud, and hubs. In order to communicate with actual devices surrounding a user’s house, each hub bought by the user participates in a variety of radio protocols involving WiFi, ZigBee, and ZWave. The smartphone’s associated mobile application allows users to control their hubs, link devices to the hubs, and download SmartApps through an app store named SmartThings Mobile. It is SmartApps that run on the cloud backend. Smart devices are connected to software, and they represent actual gadgets in a user’s house, also run on the cloud backend.

Existing IoT security research has primarily focused on standard IoT security vulnerabilities such as device or protocol defects, malicious applications, side channels, and platform difficulties [10]. Cisco’s 7-layer model is thought to illustrate attacks. At the physical layer, substituting a smart device’s firmware with a malignant version, attackers may be able to read data in transit or saved on the device. The non-network side-channel attack is another technique of hardware exploitation. In that exploit, the attacker monitors the device’s electromagnetic signals in order to reveal the device’s state. Another hazard to smart devices is a denial-of-service (DoS) attack. DoS attacks might include resource exhaustion and battery draining [14].

Ronen et al. built a worm that might use ZigBee faults to propagate the worm across smart bulbs by taking advantage of weaknesses within communication protocols [26, 10]. When it comes to IoT platforms side channels, Ronen et al. demonstrated that enemies could convey sensitive data via strobed smart bulbs [10, 26].
The action made by a smart device as a result of one app’s execution may initiate a sequenced operation of another program. However, not all of these sequenced actions are wanted by house owners, and few can even raise safety or privacy concerns. For example, if one app turns off the light after a length of time when no motion is detected, and another app opens the blinds if the home is too gloomy throughout the day, the curtain opening considered a chained action as a result of the light-off action [7].

The widespread usage of the trigger-action programming model, which allows various devices and apps to be chained together, makes it difficult to determine the underlying reason behind the unexpected event. Because of this, fraudulent or susceptible IoT programs in a chain could have far-reaching ramifications for home security, such as gaining access to sensitive data or performing privilege-based functions [31].

Interactions in an IoT environment where applications and gadgets work together to create a smart home environment could be divided into cyberspace interactions and physical interactions [11]. First, with cyberspace interaction, Apps communicate in cyberspace via a shared device or system event (for example, on/off or home-mode). For instance, after sunset, one app switch on the light, and when the same light is switched on, another app opens the door. On the same device, these two apps connect through the switch. On event and share a device attribute.

Second, with physical interactions, one of the most interesting aspects of the Internet of Things is that apps and devices can have an impact on the physical
world. Apps that operate on separate devices can externally connect with one another via common physical environment channels such as temperature, brightness, and humidity because of such physical impacts [11]. For instance, the heater is turned on through its application, when thermal sensors determine that the temperature is above a predetermined level, the temperature-control app will open the window. The heater and temperature sensor are linked by the temperature channel in this situation, resulting in a physical connection [11].

Regarding conducting an extensive review of a smart home platform’s security, Fernandes et al. utilized Samsung SmartThings as an instance and uncovered security problems such as excessive privileges. Through the use of a malicious monitoring battery SmartApp, the door lock’s Pincode can be exposed, posing major security risks. Then, utilizing the technology that monitors information flow in IoT, Fernandes et al. proposed FlowFence, a technique for controlling information flow and isolating data, with the goal of preventing sensitive information leaks on the SmartThings platform, which allows users who get permissions to utilize data safely and legally [12, 30].

Recent reports have revealed design defects in current IoT platform permission models, putting consumers at risk of break-ins and theft. A new access control mechanism for present and future IoT platforms is required to address these issues. ContextIoT is an IoT platform permission system that recognizes the context at a finer level for critical operations, as well as runtime alerts with a variety of context-based
data to assist users in permission management, therefore, guaranteeing contextual integrity [15].

IoT systems with several applications may have unsafe and harmful circumstances due to cyberspace interactions. Soteria [5] and IoTSan [22] provide a collection of reference policies in terms of IoT safety, security, and functionality that can be used to determine whether an IoT ecosystem is safe and secure and runs appropriately. Then, assuming several apps are installed to control the same devices, they implement model checking approaches to verify the appropriate security or safety attributes. Soteria and IoTSan are primarily concerned with finding potentially dangerous or insecure cyberspace interactions, such as if various programs alter the same device attributes in unexpected or inconsistent ways [11].

Physical interactions can also put consumers in dangerous and unforeseen conditions, which could be used by attackers to execute IoT attacks. Suppose a criminal has access to a temperature-triggered window control program, for example. In that case, he could intentionally causing a window to open by changing the temperature, perhaps leading to a break-in [11]. SmartAuth [29] provides a semantic-based intelligent authorization system that is user-centric. It automatically gathers data about security from the descriptions, code, and comments of IoT Apps, and generates user-authentication interfaces to improve the creation of platform security policies [29], [30].
**Home-Guard** [7] In smart home apps, concerns regarding cross-app interference can be found through inspecting policies and configurations using SMT (satisfiability modulo theory) models. Their Threat Detector detects CAI threats whenever a new app is installed, or an existing app setting is altered by examining the interaction relations between the installed or updated app’s rules and those of other programs that are already installed in the smart home. IoTGuard is a solution for enforcing dynamic policies in IoT contexts. It monitors the runtime behaviors of IoT programs and inhibits harmful and unwanted states. IoTGuard, on the other hand, like Soteria and IoTSan, focuses on cyberspace interactions in multi-app IoT systems [11, 6]. Furthermore, IoTGuard mainly considers app actions and ignores the communication behavior of IoT devices.

HoMonit compares the SmartApps activities concluded from enciphered wireless communication with their predicted behaviors defined in their UI interfaces or source code to monitor SmartApps. However, in order to predict the transitional state of IoT device behavior, Homonit requires certain wireless sniffers based on hardware to keep track of enciphered wireless data [33, 30].

ProvThings automates the monitoring of IoT apps and device APIs. It provides a comprehensive description of system actions, including harmful activities through the information it produces [31].

As smart gadgets continue to proliferate, Denning et al. have identified several emerging dangers to smart homes. There are, for example, concerns about
eavesdropping and direct device compromise. Attacks incorporating data deletion, illegal physical entrance, and privacy violations are also explored by Denning and colleagues [8, 12].

Several smart home hubs, such as SmartThings, were subjected to a security audit by Veracode. It was determined whether SSL/TLS was utilized, if there was replay attack protection, and if strong passwords were used as part of the security assessment. An open telnet debugging interface on the SmartThings hub was found to have been incorrectly implemented by Veracode, although this issue has subsequently been resolved [12, 4].

In terms of assessing risks of Android apps, WHYPER researchers concentrate on a single app’s permissions and see if the app’s description explains why it demands a certain permission. This model utilizes Natural Language Processing approaches to find statements in an application description that reflect the necessity for a specific authorization. As a result, they were able to determine that an Android app’s description was coherent to its actions. A corpus of 581 common Android app descriptions including approximately 10,000 natural language lines is used to test their architecture. Their evaluation shows a significant progress above basic searches based on keywords. [24].

Malicious Android apps can be evaluated for their risk using machine learning techniques. For instance, in RiskMon SVM was utilized to calculate an app’s risk score based on the trusted apps. They create a risk assessment baseline by pairing
the expectations of users with the trusted programs’ runtime behaviour. Every time an attempt is made to get access to private information, RiskMon calculates a risk score and uses that to assign a ranking order to the various applications [16].

In addition to harmful programs, it is necessary to configure a smart app with sensors and actuators when it is installed. Poor IoT system setups can lead to dangerous physical conditions. Such errors are frequently caused by a variety of factors, including, but not limited to, the description of an application is not clear.

When it comes to safety and security in a network, IoT devices’ types play an important role in protecting these devices’ communication with their controlling smartphone apps. Device firmware analysis is generally more complicated than one might expect. A set of 32 associated apps were needed to analyze 96 common IoT devices. Using mobile apps as a starting point, researchers may identify possible flaws in the Internet of Things equipment. There, 31 percent of associated apps do not encrypt their communication with the device. Furthermore, local or broadcast communications are utilized by a substantial number of applications. Therefore, the inadequacy of cryptography or the usage of hard-coded enciphering keys creates an attack route [17].

A survey on the security of smart homes was provided by Komninos et al., classifying threats within the network domain, for instance, eavesdropping, traffic analysis, and replay attack. Moreover, they highlight distinct usage situations while also establishing the idea of home automation, including device imitation, updates, and illegal
Cryptographic misuse and memory corruption have all been found in Android apps under examination for possible security flaws. For instance, Egele et al. evaluated the infraction of six conditions including the utilize of ECB mode and constant keys/IVs/seeds. Apps that employ the ECB method of encryption can be detected using Wei and his colleagues’ static analysis tool for Android apps, which is limited in scope [32, 17, 19, 21].

Employing natural language processing approaches, some researchers have shown that it is possible to derive policy flows from the descriptions of apps [10, 23, 29].

Our purpose is to make IoTMon more secure through our research. Since it deals with devices’ physical interactions, it would be dangerous if some attacker tries to manipulate these devices actions. Specifically, changing the app description that would affect how the physical channel is identified. Thus, if IoTMon is un-capable of identifying those channels, it would not function properly and therefore the attacker can control the modified-description device freely. In other words, the attacker blinds IoTMon by changing the description of the app before controlling the IoT device for malicious purposes.
Chapter 3

Comparison between our extended IoTMon and other Iot apps

**IoT Guard**  When an app receives an event and tries to perform an action, IoT-Guard evaluates the app’s events and actions versus a set of policies. The rules serve as models for various aspects of safety and security. However, the IoTGuard rules are initiated and labeled manually. When the user isn’t home, a policy like ”user-not-present—appliances-off and doors-locked” mandates that the door be locked and the appliances turned off. Moreover, the focus is mainly about apps actions, not devices interactions [6].

IoTGuard can either be configured to automatically prohibit an action that breaches a policy, or it can be configured to prompt the user to approve or refuse the action at runtime. This latter approach is less secure for people who do not
pay attention to the warnings before installing applications. Real-time automation could be hindered by run-time prompts, as consumers must be awake to authorize an activity. According to researchers, the IoT console may be made more user-friendly by including data from IOTGUARD’s data collector, such as app descriptions and device locations. So, it is considered a violation only if the interacting events cause an action that breaches a predefined policy (see Figure 3.1).

They designate an IoT platform’s events and actions with trusted label, and the data it collects as secret. Depending on the trigger-action platform rules, they label the platform’s events and actions. For example, a send-email event will be tagged with a trusted label if there is a rule which activates a smart switch as soon as the
user sends an email. In IoTMon, we notify the user about any changes made to app description as well as correcting it with the original description (see Figure 3.3).

Integrity and confidentiality was considered in terms of policy violation rather than in the integrity of app description. Furthermore, whenever an untrusted event alters a trusted attribute, an integrity policy violation occurs. Integrity violations include things like an app switching on the light when the person is tagged in a Facebook image, according to S.1 in the Figure 3.4 (the light is turned on by untrusted user-
Figure 3.4: A visual representation of the specific policies of trigger-action platform. Unaccepted states are marked with X.

tag event). However, this violation will not occur if the trusted labeled event was done by an attacker to cause another app action such as door-unlock, and then, he could break into the house. This can be called event spoofing, accidentally triggering certain actions in SmartApps when an unreal event is operated \[30\], see Figure 3.4.

Policy definitions can be set up through IOTGUARD’s GPL. An erroneous policy formulation may restrict valid states, fail to block hazardous and insecure states, or collide with another policy in extremely complex IoT contexts. If, for example, one policy allows "a" to be performed when a particular event happens, a second policy may disallow all other actions that are part of the set that "a" is part of \[6\].

Moreover, an applet’s event or action may not be compatible with the functionalities of an IoT platform, making it difficult to extract and label IFTTT events and actions. A massive number of IFTTT applets cannot be supported by this method \[6\].

ContexIoT is similar to IoTGuard in that they accommodate their design to SmartApps’ trigger-action-based programming platform and assume that the platform is reliable and free of flaws \[15\] (see Figure 3.5).

For example, in Figure 3.5 a permission-context mapping table is stored for every
user by the backend permission service. A permission request with context information is delivered to the backend every time a ContexIoT patched app tries to conduct a security critical activity. In order to determine if the context has earlier been authorized or rejected, the cloud-based permission service performs a verification. If it was not in the cloud, the user is prompted with a popup that displays the permission request and the accompanying context, and an additional entry is added to the mapping table to record the user’s choice as security preferences [15]. This could be a drawback if the attacker triggers a previously approved action, the user will not be informed.

ContexIoT provides an IoT platform context-based permission system that enables users to make access control decisions based on fine-grained context information. To divide the SmartApp’s execution into the collecting of context and the granting of permissions phases. To counter this, ContexIoT requires users to actively help make decisions at runtime, which detracts from the convenience of home automation while also putting it at risk of exceeding SmartThings’ 20-second execution time constraints [7].
Table 3.1: Comparison between extended IoTMon and other tools

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>cross-app analysis</th>
<th>No run-time intervention</th>
<th>App description Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoTGuard</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>ContexIoT</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>SmartAuth</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Extended IoTMon</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

One major difference between IoT settings and traditional networks is that IoT devices may communicate with the physical world around them, such that even in the absence of a network, IoT devices can communicate with one another via physical channels. Nonetheless, It is not possible to see these physical interactions from individual apps. Therefore, contexIoT versus IoTMon have no inter-app interactions discovery or analysis. It is possible for malicious apps to pass a traditional code review but interact with other apps in a detrimental way because of Cross-App Interference threats, which we detail in this research. Moreover, the interaction of even the most benign apps can result in unwanted or even hazardous results [7].

This Table 3.1 summarizes the differences between the extended IoTMon and other systems. As we stated, IoTGuard does stop a state if it violates a predefined policy, and it is not discovering all hidden/possible interactions among apps that could put the user in unsafe environment. The extended IoTMon focuses on all possible cross-app interactions and it’s effects on the user safety, it has no run-time intervention, and it verifies the app description and system Integrity.
Chapter 4

IoTMON Architecture

We begin with precise detail on IoTMon’s architecture and implementation in this section. Application analysis, finding connection chains, risk assessment and management are the three main phases of the IoTMon system. We first discuss the two parts of application analysis, intra-app analysis methods and physical channel determining. An important part of this component’s goal is establishing inter-app interaction chains by connecting several intra-app interactions after identifying the physical channels. In the following section, we explain how to find the inter-app interaction chain. Lastly, we illustrate their tool approach to risk assessment and risk reduction.

4.1 Examination of Intra-app Processes

This is one of the two components in the application analysis phase in Figure 4.1.
4.1.1 General Policy Structure

The "If-This-then-that" (IFTTT) programming paradigm is commonly used in IoT applications. They introduce a general policy structure for inspecting the app code. The trigger-action connection in IoT applications is defined in IoTMon by three key characteristics. The general policy framework is constructed by gathering the following data from applications:

- An explanation of the software: Each application’s creator normally provides this section at the beginning of the app.

- Triggering occurrence and related gadget: The application’s source code specifies the criteria that cause an event to occur. An instance of this is "if the measured temperature exceeds a predefined limit” as a trigger condition.

- Action and related gadget: In the application’s source code we can find the initiated actions. For instance, we could describe the initiated action as switching on/off some gadget.

4.1.2 Intra-app Interaction Study

There are three stages to this tool’s analysis of an application. The application’s Abstract Syntax Tree (AST) is first created. For the second step, the program examines the code’s preference part, which lists all of the system’s abilities and applications inputs. Third, this tool is designed to obtain the app’s triggers and actions.
4.2 Physical Channel Identification

Physical channels in an IoT setting like illumination, humidity, motion, and temperature are directly tied to IoT device physical interaction abilities, such as adjusting illuminance or rising temperature. To illustrate the process of physical channel identification, they use three steps to identify these channels. To begin, channel entity keywords are extracted from application descriptions using NLP algorithms see Figure 4.2. After that, they use Word2Vec to compare the extracted keywords to each other. Finally, they group entity keywords together relying on their similarity and use entity keyword clusters to determine physical channels [10]. For instance, it is possible in this system to cluster "lights" and "bulbs" in an application description based on the similarity of their respective keywords. Further details will be discussed in the following section, Section 4.5.
4.3 Finding Interaction Chain

Using the physical channels and intra-app connections that were discovered in the preceding steps, the tool also discovers inter-app interaction chains. Their findings reveal that Samsung’s SmartThings platform has a number of system channels that may be utilized to connect different apps and services to each other. On the same platform, multiple programs can use these system channels at the same time. Their finding of the inter-app connection chain includes these shared system channels as well. If the capability of non-physical channel is utilized as a trigger in one interaction within the app and as an operation in another, researchers treat this ability as a common system variable throughout the interaction modeling within the app.
4.4 Examining and Reducing Risk

The goal of this section is to present an inter-app connection chain with a technique for risk assessment and analysis. Communication chains between apps face two challenges in determining their safety. To begin, they require an approach for calculating the physical effects of various interactions within the app and between several apps. It is also necessary to have something to measure possibly dangerous interactions against such as benign interactions.

These issues necessitate the development of an approach to measuring the distance between channels by giving them the appropriate values this is called behavioural analysis approach. When looking for examples of innocuous interactions, they search no farther than platforms’ official applications or third-party apps that were reviewed and certified by those systems. As a result, when developing the system, they look at the dangers presented by inter-app interactions in the context of trusted programs’ intra-app connections baseline. It is their belief that if a certain communication is not included in the baseline, it is considered to pose a threat to the user.

After that, they classify all of the interactions within the app utilizing K-means grouping to get a baseline. With this baseline, they can lastly compute risk scores for cross-app communication paths that look dubious. To limit the amount of potentially dangerous cross-app communication paths, they suggest a strategy for reducing risk. They utilize a channel tuple to describe channel associated data such as trigger and action for an interaction within the app. Cross-app interaction chains have many
different interactions within the app and associated channels since they are interconnected. Vectors are used to express cross-app and within-app interaction activities. Every vector is made up of all the physical and system channels that are accessible. When a channel is used, its status is displayed by the value of its corresponding element in the vector.

It is possible to map all interactions into a high-dimensional vector depending on the connected channels and their values. In each channel, the channel’s behavior is reflected by the channel’s associated value. As an example, in their Samsung Smart Things platform based prototype, they determine seven physical and four system channels: temperature and humidity, water, smoke and illumination, motion and presence are the physical channels. Location-Mode, time, switch, and lock are the system channels. The distance between comparable vectors is used to calculate the similarity across connections. For each inter-app connection risk measurement, the distance to the nearest baseline cluster is calculated. Risk mitigation strategies in IoTMon are adaptable enough to handle a wide range of circumstances. Incorporating new trigger criteria, such as verifying an additional relevant state, can help lessen the likelihood of unanticipated interaction chains between apps. Thus, it is possible that malignant programs will be unable to trigger vulnerable programs.
4.5 Problem Description

IoTMon concentrates on application-level threats against app-powered IoT platforms. Physical channels are being abused by adversaries in an attempt to set off unanticipated activities which could harm the physical environment [10]. They assume that app description is trustworthy although this cannot be true in case some attacker tries to modify it, as a result, this will compromise the whole IoTMon system since its mainly depends on app description in order to identify the physical channel, as in Figure 4.1.

Suppose that an adversary tries to change temperature-control app description to make it look like lights-related app Figure 4.3. When it goes to physical channel identification, it will be considered as illuminace channel instead of temperature channel. Thus, it is important for the physical channel to be accurate in order to find all possible cross-app interaction chains. Otherwise, all the discovered cross-app interactions and the risk score of the physical channel will be incorrect. To illustrate, in the description example we mentioned in Figure 4.3, if the attacker copied another app description such as lights related app into temperature-control app description, the channel will change from temperature to illumination. Every channel has its own potential inter-app interference and risk score measurement. In summary, we help ensure safe manage of the interactions related to IoT devices by monitoring all applications descriptions against modification attacks and protecting the physical channels from change.
Figure 4.3: An attacker change temperature control app description to make it look like lights-follows me app
Chapter 5

Evaluation and case study

Here I describe a set of cases studies for comparing behavior with and without the security features added by the proposal. After modifying the app description there are two possible situations here:

1. If the device was shared between two applications such as lighting and curtains apps this could lead to unexpected app action and inter-app interactions. For example, instead of opening curtains using curtains control app, IoTMon processes the lights on action. Then, another app could close curtains since the room is bright.

2. The other scenario is that if the device is only working with one app such as thermostat app, and the modified description made IoTMon unable to identify if it is for temperature control. Therefore, IoTMon without the extended version will not be able to discover inter-app interactions or doing risk analysis for it.
The extended IoTMon was able to detect all changes made to app descriptions in Figure 5.1.

Since the original IoTMon uses NLP to determine the channel and therefore, discovering inter-app interactions. It is important for the app description to be checked for any modification attacks. These two Figures 5.2 and 5.3 illustrate the risks in the original IoTMon that is fixed in our extended version. In Figure 5.2 our version was able to discover the real inter-app interactions related to temperature channel since it protects the physical channels from any changes. It detects the window opening action which has a high risk score. However, in Figure 5.3 the malicious description makes IoTMon process the temperature app as an illumination channel app, and discovering other apps interactions related to illumination such as closing curtains when the room is too bright which is not considered risky compared to window opening action.

<table>
<thead>
<tr>
<th>App</th>
<th>Original Description</th>
<th>Modified Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flood Alert</td>
<td>&quot;Get a push notification or text message when water is detected where it doesn’t belong.&quot;</td>
<td>&quot;Locks a deadbolt or lever lock when a SmartSense Presence tag or smartphone leaves a location.&quot;</td>
</tr>
<tr>
<td>It’s Too Cold</td>
<td>&quot;Monitor the temperature and when it drops below your setting get a text and/or turn on a heater or additional appliance.&quot;</td>
<td>&quot;Turn your lights on when motion is detected and then off again once the motion stops for a set period of time.&quot;</td>
</tr>
<tr>
<td>Smart Home Monitor</td>
<td>&quot;Monitor your home for intrusion, fire, carbon monoxide, leaks, and more.&quot;</td>
<td>&quot;Send a text message when there is motion while you are away.&quot;</td>
</tr>
<tr>
<td>Unlock it when i arrive</td>
<td>&quot;Unlocks the door when you arrive at your location.&quot;</td>
<td>&quot;When the humidity Vent reaches a specified level, activate one or more vent fans.&quot;</td>
</tr>
<tr>
<td>Lock It When I Leave</td>
<td>&quot;Locks a deadbolt or lever lock when a SmartSense Presence tag or smartphone leaves a location.&quot;</td>
<td>&quot;Get a push notification or text message when water is detected where it doesn’t belong.&quot;</td>
</tr>
</tbody>
</table>

As a case study, we applied the extended version of IoTMon on the data set that is provided in the official GitHub page of IoTMon. The data set contains a 92 number of apps that are designed in Samsung’s Smart Things platform. The goal of this case
The app name is itsTooCold

description: "Monitor the temperature and when it drops below your setting get a text and/or turn on a heater or additional appliance."

description: "Turn your lights on when motion is detected and then off again once the motion stops for a set period of time."

Warning: The app description has been modified

The Changed Words: Turn your lights on when motion is detected and then off again once the motion stops for a set period of time."

It was successfully reverted back to it’s original description.

Figure 5.1: Output of extended IoTMon that prompt the user regarding modifications applied to an app

Figure 5.2: The extended IoTMon cross-app interaction related to temperature channel

Figure 5.3: The original IoTMon cross-app interaction after the modification attack becomes related to illumination channel
study was to analyze whether our extension was capable of identifying all modified apps. Thus the extension would be able to prompt the users and allow them to take necessary actions, such as deleting an app that is maliciously modified by an attacker. As we mentioned before, IoTMon relies on the description of apps in the Smart Things platform to identify physical channels that will be then used by IoTMon to discover interactions between these apps in the system.

We manually edited 22 of the apps that are included in the data set to mimic the malicious actions that will be taken by an attacker. As a result, the number of modified apps in our case study is 22. The rest of the apps were left unmodified. Our goal behind this step was to see whether our extension would work properly and avoid falsely prompting the users with apps that have not been modified. As a result, our extension was successfully able to capture and fix all modified applications and prompt the users accordingly. Figure 5.1 shows an example of an app that has been maliciously modified. In this example, the attacker changed the description from "Monitor the temperature and when it drops below your setting get a text and/or turn on a heater or additional appliance" to "Turn your lights on when motion is detected and then off again once the motion stops for a set period of time". The attacker successfully modified the description to prevent IoTMon from identifying temperature as a physical channel for the "temperature control" app. Based on the new description, the new channel for the modified app would be illumination.

Table 5.1 demonstrates other modified apps that were captured successfully by
our extended IoTMon. Analyzing the modified descriptions of these apps show how
the attacker would be successfully able to manipulate the physical channels that will
be used by the original IoTMon if these changes went undetected. For instance,
the description of “Unlock it when I arrive” was changed from “Unlocks the door
when you arrive at your location” to “When the humidity vent reaches a specified
level, activate one or more vent fans”. These changes in the description modified
the original physical channel, which was Location, to a new physical channel, which
is humidity. Thus, all subsequent interactions that will be used and discovered by
the original IoTMon will be manipulated as a result of these malicious modifications.
This would result in inaccurate risk analysis results, which may reflect the attacker’s
goals to manipulate the IoTMon and create opportunities for malicious actions, such
as burglary. Hence, the original IoTMon will be turned from a tool that can help
provide accurate risk analysis and identify any risky interactions between apps within
the smart system to a tool that can be used to meet attackers’ goals and needs. The
list of all apps that were examined in our case study is presented to the reader in the
Appendix A.1.

Another case study example is Curling-iron, SwitchChangesMode and MakeItSo
can be chained together to construct a hidden connection after CurlingIron app
switches on a set of ports when there is motion. As a consequence, a door is un-
locked by this secret condition. If an attacker tries to mislead the user by modifying
curlingIron app description to, for example, AC control related description. Then,
The app name is Curling Iron

description: ”Turns on an outlet when the user is present and off after a period of time”,

description: ”Monitor the temperature and when it rises above your setting get a notification and/or turn on an A/C unit or fan.”

Warning: The app description has been modified.

The Changed Words: ”Monitor the temperature and when it rises above your setting get a notification and/or turn on an A/C unit or fan.”

It was successfully reverted to the original description: ”Turns on an outlet when the user is present and off after a period of time”

Figure 5.4: Output of extended IoTMon that prompt the user regarding modifications applied to Curling Iron app through the motion channel the attacker can use a robotic vacuum cleaner to break a door. Without the extension we will not discover this hidden interaction that leads to door unlocking because the channel would change from motion to temperature. Thus, we ensure that IoTMon is beneficial and reliable in terms of protecting users safety and security by discovering the real inter-app interaction chains with their risk scores in order to mitigate threat, see Figure 5.5. We ran IoTMon against curling iron, and as expected, our version was able to capture the changes made and notify the user. In addition, the modified description was reverted to the original one. Therefore, all modifications made to app descriptions were discarded by the extended IoTMon, see Figure 5.4.

Our evaluation in this case study showed how the extension that we implemented
Figure 5.5: The extended IoTMon Interaction chains after Curling-iron App
is essential for IoTMon to function properly. We demonstrated how our extension was successful in capturing all modified apps and prompted the users to act and protect their IoT systems. We showed how our extension didn’t provide any false positives. In other words, if an app has not been modified, our extension will successfully skip it and not prompt the users with anything related to that app. We demonstrated how changing a physical channel can compromise the whole functionality of IoTMon.

5.1 Validating IoTMon Integrity

As we ensure the integrity of physical channels that affects discovering the real cross-app interactions. We also implement a hash function to store hash values of the whole extended IoTMon system in a file and then, every time we use IoTMon it computes the hash value in order to compare it with the first hash file made. The goal of this step is to protect the whole system from any modification attacks made to the code that could compromise the results and therefore, the safety of IoT environment, see Figures 5.7 and 5.8. If an attacker introduces a malicious script into the system or presents a malignant application to the user that could put him in a dangerous or insecure condition by modifying the IoTMon code, we consider the hashing step as a defense method against malicious code changing. To ensure that the hashing function results are not also compromised or tampered with, we applied a digital signature method. Also, we verified the digital signature using GNU Privacy Guard tool (GPG), it can be utilized for files or email encryption. PGP Pretty Good Privacy
is implemented by the GPG software toolkit. It is an enciphering technique that allows two individuals to securely transmit data using both symmetric and asymmetric encryption. Additionally, PGP can be used for signing files, it allows recipients to verify the sender’s identity and ensure that the message has not been tampered with while in transmission. In other words, the ability of attackers or malicious programs to resist protective measures such as cryptography and faked inputs is something we take into account Figure 5.6. Digital signatures used in combination with PGP increase their security and lower the likelihood of security problems while transferring public keys.

GnuPG does not save your private key in the form of a raw file on your computer [2]. The private key of the signer is used to generate a signature, and it is encrypted using a passphrase. In order to obtain your private key, an adversary that is not in control of the run time memory must overcome two obstacles: first, he should obtain the key file itself, and second, he should overcome the password encryption [2]. The public key related to the signer is used to verify that the signed file is not modified. We used SHA-512 as a hash function in our digital signature process and generating hashes for our extended IoTMon’s files. Unlike vulnerable hash functions such as SHA-1 and MD5, SHA-512 is one of the few standards that are recommended by the National Institute of Standards and Technology (NIST). [28].
Figure 5.6: The digital signature for the hash values.
Figure 5.7: Original Hash value of all IoTMon files.
Figure 5.8: The new hash value that is computed every time IoTMon is used or updated.
5.2 Extension Impact on IoTMon

We did a code static analysis on the extended version of IoTMon to examine whether our extension introduces any defects, code inconsistencies or other issues that may negatively affect IoTMon. To do so, we used a groovy extension [3] that has code-narc embedded in it. Code-narc [1] has been used for code static analysis on groovy-based language tools to discover and pinpoint any issues that are similar in nature to the aforementioned issues that we mentioned. As a result, running the groovy-based extension did not reveal any issues related to the part that we integrated into IoTMon to extend its functionality. In other words, our extension of IoTMon did not introduce any issues that need to be addressed. To generate a Code-Narc-based report, we leveraged the official docker image provided by the Code-narc organization [1]. Code-Narc took approximately 3 seconds to generate its full report on the extended IoTMon. We provide a snapshot of the generated report in Figure 5.9. The full report is presented in Appendix B.1.
Figure 5.9: The extended IoTMon code analysis
Chapter 6

Conclusion

Inter-app interaction dangers arise when numerous differently built apps interact and affect one another in the environment of home automation, producing unwanted and possibly hazardous collisions. Especially, when we do not know about them due to an adversary performing a modification attack that blinds the system from reporting the real cross-app interaction. Therefore, the main goal of controlling and analyzing physical interactions risks by IoTMon will be unfulfilled, and IoT users will not be informed about the dangerous consequences of such an interference.

In this thesis we extended a security tool named IoTMon, which is a management system for physical communications of IoT devices capable of identifying all possible cross-app communication chains and assessing their threat status. For cross-app risk reduction the current IoTMon can give guidelines for apps’ programmers to include other trigger rules in the app code to minimize the likelihood of triggering defenseless application while prompting the users with risk warnings. In our extension, we
monitor and protect IoTMon from any changes made to apps description to help identifying the real inter-app interactions. In addition, we monitor the integrity of the whole IoTMon system to make sure it is reliable in terms of controlling inter-app interactions by generating hash function of the system, and whenever the IoTMon is used a new hash is generated to be compared with the original hash. We protect these hash files from any possible modifications using digital signature. Our evaluation shows that the extended IoTMon was able to discover 22 apps that their description was modified. As well as prompting the user whether intentionally or unintentionally changes were made to the IoTMon tool.

We have different plans for future work such as creating a tool that can be platform independant and focuses on analyzing apps descriptions. In addition, conducting a study to examine the user perception regarding whether they pay attention to the app description or they view it as easy to follow. Our new tool would have more features than the current IoTMon such as summarizing apps descriptions. Although we encrypted and stored the private key using the pgp tool, we will look into other ways to store the key off the local machine either offline such as on a USB drive or on the cloud.
Appendix A

Case Study of Samsung

SmartThings IoT Apps

Figure A.1 shows the output of the extended IoTMon with the list of apps that have been modified. The modified apps have a warning message that indicates what the original description and the changed one are.
Figure A.1: The extended IoTMon output after discovering all modifications made to applications
Appendix B

Code-narc Full report on IoTMon

Static analysis of groovy-based language code has been performed using Code-narc. A groovy extension with code-narc incorporated in it was used to accomplish this task. We provide a full list of warnings that are not related to our extended groovy class Figure B.1. There are no signaled issues that need to be resolved as a result of our work with IoTMon extension.
Figure B.1: Full report on the extended IoTMon code
Appendix C

Verifying Digital Signature

We verify digital signature to examine the integrity of hash files we created for IoT-Mon. Figure C.1 shows Good signature which means that the hash file is free from any malicious modification. Whereas in Figure C.2 it shows Bad signature as we tried to mimic an adversary or virus changes to the hash file.

![Digital signature output]

Figure C.1: Digital signature of the hash file that has not been affected by any modification.
Figure C.2: Digital signature of the modified hash file.
References


