7-2-2020

A Comprehensive Security Framework for Securing Sensors in Smart Devices and Applications

Amit Kumar Sikder
asikd003@fiu.edu

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

A COMPREHENSIVE SECURITY FRAMEWORK FOR SECURING SENSORS IN SMART DEVICES AND APPLICATIONS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL AND COMPUTER ENGINEERING

by

Amit Kumar Sikder

2020

To: Dean John Volakis
College of Engineering and Computing

This dissertation, written by Amit Kumar Sikder, and entitled A Comprehensive Security Framework for Securing Sensors in Smart Devices and Applications, having been approved in respect to style and intellectual content, is referred to you for judgment.

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

Kemal Akkaya

Alexander Perez-Pons

Bogdan Carbunar

A. Selcuk Uluagac, Major Professor

Date of Defense: July 2, 2020

The dissertation of Amit Kumar Sikder is approved.

Dean John Volakis
College of Engineering and Computing

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

Florida International University, 2020

DEDICATION

To my parents, Ashoke Kumar Sikder and Shikha Sikder

and

my sister, Sharmistha Sikder

ACKNOWLEDGMENTS

First of all, I would like to thank my major professor and advisor, Prof. Dr. A. Selcuk Uluagac, for his invaluable guidance, mentorship, motivation, and constant support throughout this research and during all my doctoral studies. His positivity and advice always inspired to overcome all difficulties and achieve higher success in my professional and personal life. I would also like to express my gratitude to the members of my dissertation committee, Dr. Kemal Akkaya, Dr. Alexander Perez-Pons, and Dr. Bogdan Carbunar, for their insightful comments, guidance, encouragement, and generous support. They were great inspiration to me throughout my journey. Throughout my doctoral journey, I had several chances to work with great researchers around the world. I would like to thank my colleague and dearest friend Leonardo Babun for his constant help and encouragement. I would also like to thank our post doctoral researcher Dr. Hidayet Aksu, my labmate Abbas Acar, and all the members of Cyber-Physical Systems Security Lab (CSL) for their encouragement, accompaniment, and collaboration through all these years. I would like to thank Dr. Z. Berkay Celik from Purdue University, Dr. Trent Jaeger and Dr. Patrick McDaniel from Penn State University, and Dr. Engin Kirda from Northeastern University for their contribution in several research works in this dissertation. Last, but not the least, I like to thank my parents and my sister who always encouraged me throughout my doctoral studies. Finally, I would like to thank three of my closest friends, Sharraf Samin Mahin, Shammya Shananda Saha, and Farjana Siddiqua for their constant support and words of encouragement. I would like to acknowledge the support provided by the National Science Foundation, FIU University Graduate School, and the Department of Electrical and Computer Engineering at Florida International University. This dissertation is mostly based upon the work supported by the National Science Foundation under Award Number NSF-CAREER-CNS-1453647.

ABSTRACT OF THE DISSERTATION

A COMPREHENSIVE SECURITY FRAMEWORK FOR SECURING SENSORS IN SMART DEVICES AND APPLICATIONS

by

Amit Kumar Sikder

Florida International University, 2020

Miami, Florida

Professor A. Selcuk Uluagac, Major Professor

This doctoral dissertation introduces novel security frameworks to detect sensor-based threats on smart devices and applications in smart settings such as smart home, smart office, etc. First, we present a formal taxonomy and in-depth impact analysis of existing sensor-based threats to smart devices and applications based on attack characteristics, targeted components, and capabilities. Then, we design a novel context-aware intrusion detection system, 6thSense, to detect sensor-based threats in standalone smart devices (e.g., smartphone, smart watch, etc.). 6thSense considers user activity-sensor co-dependence in standalone smart devices to learn the ongoing user activity contexts and builds a context-aware model to distinguish malicious sensor activities from benign user behavior. Further, we develop a platform-independent context-aware security framework, AEGIS, to detect the behavior of malicious sensors and devices in a connected smart environment (e.g., smart home, offices, etc.). AEGIS observes the changing patterns of the states of smart sensors and devices for user activities in a smart environment and builds a contextual model to detect malicious activities considering sensor-device-user interactions and multi-platform correlation. Then, to limit unauthorized and malicious sensor and device access, we present, KRATOS, a multi-user multi-device-aware access control system for smart environment and devices. KRATOS introduces a formal policy language to understand diverse user demands in smart environment and implements a novel policy negotiation algorithm to automatically detect and resolve conflicting user demands and limit unauthorized access. For each contribution, this dissertation presents novel security mechanisms and techniques that can be implemented independently or collectively to secure sensors in real-life smart devices, systems, and applications. Moreover, each contribution is supported by several user and usability studies we performed to understand the needs of the users in terms of sensor security and access control in smart devices and improve the user experience in these real-time systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Smart devices such as smartphones, smart watches, smart locks, smart thermostats, smart plug-ins, and their applications have become omnipresent in our day to day lives. The role of smart devices and their applications has expanded from making phone calls and messaging to home security, health care, and even military applications. With the introduction of the Internet of Things (IoT) [LBAU17, CMT+19], smart devices can now interact with each other and with the physical world to perform different tasks and take autonomous decisions. As smart devices use different high-precision sensors to ensure seamless integration with the cyber and physical world, they provide more efficient and user-friendly applications [LML+10]. However, recent researches have proved that attackers can exploit the sensors of smart devices to perform different malicious activities [FJP16, JCW+17]. Attackers can leak sensitive information using sensors, trigger an existing malware on a device, or steal sensitive stored information using sensors [SAU17, SPA+18]. Attackers can even use the sensors as a communication channel to transfer a malware in a smart device [USB14]. These sensor-based threats have become more critical with the growing popularity of sensing-enabled applications and automation systems in smart environments (e.g., smart home, smart office, smart factory). Unfortunately, existing security solutions do not consider these threats and attackers can easily bypass the solutions to initiate sensor-based attacks in the smart devices. For instance, the Android sensor management system only enforces permission-based access control to the apps for specific sensors (e.g., GPS, camera, audio sensors). After granting initial permission, the user cannot control the usage of the sensors by the apps. More importantly, there are sensors that are by default activated by the sensor management systems. For instance, an installed app automatically gets the permission to access motion, light, and proximity sensors. Also, unauthorized sensor access in one device might lead to malicious activities in other connected devices [BSAU18]. For instance, in a smart home system, an attacker with unauthorized access to the smoke sensor can inject false data to trigger the fire alarm maliciously. In this context, a comprehensive security framework is needed to supervise atypical sensor access and detect these emerging sensor-based threats in both standalone and connected smart devices. Moreover, a detailed understanding of these threats as well as the needs of the users are also of paramount importance in this dissertation.

Investigation of Sensor-based Threats to Smart Devices & Applications: The use of sensors in smart devices inevitably increases the functionality of the devices; however, the sensors can also be used as vehicles to launch attacks on the devices or applications. Recently, there have been several attempts to exploit the security of smart devices via their sensors [SAU17]. Attackers can use the sensors to *transfer malicious code* or *a trigger message* to activate malware planted in a device [SUCB13, HSH$^+$13], *capture sensitive personal information* shared between devices (e.g., smartwatches, smart home devices, etc.) [WMJ19, SZZ$^+$11], or even *extract encrypted information* by capturing encryption and decryption keys [DPSKM15]. Moreover, attackers can use the sensors of one device as an attack platform to abuse or interrupt normal functionalities of connected devices [CA17]. For instance, a specific on/off pattern of a smart light can trick a smart camera to capture and leak pictures containing sensitive information in a smart home environment [SBAU19]. Recently, TrendMicro, a renowned security company, reported in 2019 three publicly available Android apps in Google Playstore used the motion sensor to evade malware scanners in the smartphone [Mic19]. These sensor-based threats pose a significant risk to the smart devices as manufacturers are not fully aware yet [USB14]. Indeed, sensor-based threats are becoming more prevalent because of the easy access to the sensors and the limited security measures that consider these threats. Hence, trivial execution, easy access to the sensors, and lack of knowledge about the sensor-based threats constitute significant risks for the smart devices. Thus, understanding these sensor-based threats and attacks in the literature is necessary for researchers and the community to design reliable solutions to detect and prevent these threats efficiently.

Sensor-based Threat Detection in Standalone Smart Devices and Applications: The use of sensors on smart devices (e.g., smartphone, smart watch, etc.) enables a seamless connection between the devices and the physical world. Indeed, modern smart devices come with a wide range of sensors (e.g., accelerometer, gyroscope, microphone, light sensor, etc.) that enable more efficient and user-friendly applications [LML$^+$10]. While the number of applications using different sensors is increasing and new devices offer more sensors, the presence of sensors have opened novel ways to exploit the smart devices [USB14]. Attackers can exploit the embedded sensors in standalone smart devices multiple ways [USB14]. In recent years, several sensor-based threats such as keystroke inference [HS12, MAJH16], task inference [NWX$^+$20], false data injection [Cof14], and eavesdropping [SZZ$^+$11] have been reported in standalone smart devices, especially smartphones and smart watches [SAU17]. Such *sensor-based* threats become more serious with the rapid growth of Apps utilizing many sensors [CA17]. In fact, these sensor-based threats highlight the flaws of existing sensor management systems used by smart devices. Albeit useful, existing security schemes overlook these critical threats which directly impact the security and privacy of the smart device ecosystem. Moreover, although embedded sensors on standalone smart devices seem to work independently from each other, a task or activity on a smart device may activate more than one sensor to accomplish the task. Hence, it is necessary to secure all the different sensors and consider the *context* of the sensors in building any solution against the sensor-based threats on standalone smart devices.

Sensor-based Threat Detection in a Connected Smart Environment: The capabilities of the smart devices in a connected smart environment (e.g., smart home, smart office, etc.) have evolved from simply controlling lights and opening garage doors to connecting our living spaces to the cyber world [ST17, LZCC14]. With the tremendous growth of Internet of Things (IoT), smart devices now have advanced capabilities to interact with other devices and create a *smart environment* to perform different user-defined tasks collectively. Such functionality provides more autonomous, efficient, and convenient daily operations [FRJP17]. Compared to early smart environment/systems with fixed device setup procedures and limited functionalities, modern smart environment have adopted a more user-centric, app-based model. Nowadays, users can download apps from the vendor's app market (or even develop their own) and easily set up and control the smart devices, which makes connected smart environment more popular and versatile than ever [Sta18].

The integration of programming platforms with smart devices and sensors surely enhances the functionalities, but also exposes the vulnerabilities of the smart devices to attackers [BAU19]. These attackers can release malicious apps in third-party markets and public repositories (e.g., GitHub) easily. Then, careless users may download and utilize them to control their devices. From here, the attackers can exploit smart devices in several ways. Recently, a repository of malicious apps in different smart platforms has been published exhibiting several vulnerabilities of the current smart home app development ecosystem [IoT17]. Nonetheless, a security solution that comprehensively detects these emerging threats associated with smart devices and sensors in a smart setting/environment does not currently exist and is direly needed. Recent studies have proposed the implementation of enhanced permission models for smart environment, which depend on specific user permissions [JCW^{+}17] or the analysis the source code of the apps to detect vulnerabilities, which is only effective against specific types of attacks [CBS+18, MBY+19]. In this circumstances, a context-aware and platform-independent security framework that considers user activity contexts, sensor-device-user interactions (e.g., movement directions, sensors activated, rooms involved), and multi-platform correlation is needed to ensure secure operation of smart devices and sensors in a connected environment.

Multi-user Multi-device-aware Access Control System for Smart Devices and Settings: A smart environment/platform (e.g., smart home, smart office, smart factory) allows multiple devices and sensors to be connected to automate daily activities and increase the overall efficiency of the homes and workplaces [Tea17, SAA+18]. Devices as simple as a light bulb to ones as complicated as an entire AC system can be connected and exposed to multiple users. The users then interact with the devices through different smart applications installed through a mobile host app provided by the smart device vendors. Traditional access control mechanisms proposed for personal devices such as computers and smartphones primarily target single-user scenarios. However, in a smart environment, multiple users access the same smart device, typically via a common controller app (e.g., SmartThings App), which can cause conflicting device settings. For instance, a homeowner may want to lock the smart door lock at midnight while a temporary guest may want to access the lock after midnight. Also, current smart device platforms do not allow the conflicting demands of the users to be expressed explicitly. Finally, the current access control mechanism in smart device platforms offer coarse-grained solutions that might cause safety and security issues [CMT18, JCW+17, BSAU18]. For instance, smart devices often give automatic full access to every user added to the system [Sam18a]. With full access, a new user can easily add new unauthorized users, remove existing users, or reconfigure the connected devices and sensors [TZL+17, SBAU19]. This benign, yet undesired action from the new user can lead to several safety issues [SPA+18, CMT+19, NSRU19].

In these real-life scenarios, current smart device platforms cannot fulfill such complex, asymmetric, and conflicting demands of the users as they can only handle primitive and broad controls with static configurations.

1.1 Research Problems

This dissertation has the following five major research components and problems investigated:

1. A detailed investigation and formal taxonomy of sensor-based threats to smart devices and applications to identify key characteristics and impacts of these threats.
2. The design of a novel threat detection system to protect the sensors against malicious attacks in a standalone smart devices (e.g., smartphone, smartwatch) and applications, called 6thSense.
3. The design of a novel centralized intrusion detection technique to protect the sensors in a multi-user multi-device smart environment, called Aegis.
4. The design of a novel fine-grained access control system to restrict unauthorized device access in a multi-user multi-device smart environment to limit and control sensor access and resolve conflicting demands of the users, called Kratos.
5. The usability studies of the proposed frameworks to understand and support the needs of the users and improve the users' experience in a smart system.

In general, to protect smart devices against sensor-based threats, a security framework must ensure high accuracy in detecting sensor-based threats in both stand-alone and connected smart settings. Additionally, the proposed framework must provide fine-grained access control to the users to protect the smart devices against sensor-based threats emerging from unauthorized sensor access. Also, the proposed framework should be scalable to ensure effectiveness against current and future sensor-based threats to smart devices. Finally, all the components of the proposed security framework must operate in real-life systems with minimal overhead to assure efficient performance.

1.2 Significance of the Studies

Cyberspace is expanding fast with the introduction of new smart device technologies dedicated to make our homes, offices, factories automated and smarter [Tea17, SAA$^+$18]. In this ecosystem, sensors and sensing-enabled applications on smart devices in smart settings have become very popular as they provide more user-friendly and efficient functions to the users. However, these sensors and applications, if compromised, can perform different malicious activities in the devices including sensitive information leakage, triggering existing malware, unauthorized device access, etc. For this reason, it is important to secure smart devices against these sensor-based threats. In this dissertation, we aim to secure the smart devices against sensor-based threats by proposing a comprehensive security framework which focuses on five specific needs: (1) A formal taxonomy and impact analysis of sensor-based threats to smart devices and applications based on common vulnerability metrics (2) A scalable and effective sensor-based threat detector for standalone smart devices and applications (3) A centralized sensor-based threat detection method for connected multi-device smart environment (4) A fine-grained access control mechanism to protect on-device sensors from unauthorized access in multi-user multi-device smart environment And, (5) A detailed usability study to understand the needs of the users in terms of security and access control in smart devices and improve the user experience in real-time systems for each security mechanism proposed in this dissertation.

1.3 Organization of the Dissertation

The rest of this dissertation is organized as follows. In Chapter 2, we present background information to support this dissertation. Then, in Chapter 3, we discuss the related work for each research problem stated in this dissertation. In Chapter 4, we provide a detailed taxonomy and provide an in-depth impact analysis of sensor-based threats to smart devices and applications. Chapter 5 introduces a novel context-aware security framework to detect sensor-based threats on standalone smart devices. In Chapter 6, we present a context-aware platform independent security framework for connected smart environment. Chapter 7 presents a multi-user multi-device-aware fine-grained access control system for the smart environment. Finally, we conclude the dissertation and propose future research directions in Chapter 8.

CHAPTER 2

BACKGROUND

In this chapter, we present the necessary background information regarding the architecture of smart devices, a connected smart environment, and security needs of sensors in smart devices. This chapter will give an overview of the fundamental building blocks of this dissertation. More detailed information can be found in the related chapters.

2.1 Overview of Smart Devices

In this section, we introduce the components of smart devices as they are relevant to understand the significance of sensor-based threats and attacks.

In general, a smart device is an electronic device which has the capability to connect, share, and interact with its user, peripheral, and other smart devices using their sensors and communication protocols [STTPLR13, Pos11, SFRS18]. These devices in general have the following salient features:

- <u>Sensing</u> - Sensors in smart devices help to sense the surrounding environment and perform different tasks based on measured events.
- <u>Automation</u> - Automation is the ability of a smart device to perform a task automatically based on specific events. For example, the user presence in a room can cause a sensor to turn a light on and off.
- <u>Accessibility</u> - Smart devices offer easy and remote accessibility to the users. Users can control and monitor a smart device from a remote location. For example, users can lock or unlock a smart lock from a remote location using a smartphone.
- <u>Context-Awareness</u> - Context-awareness refers to the ability of a device to understand and analyze its surroundings. A smart device can understand what is

happening around the device and perform different tasks accordingly. For example, a smart lock can automatically unlock as a specific person approaches the door. Here, the smart lock is aware of the user's presence in its proximity and unlocks the door.

- Self-Learning - Smart devices can learn usage patterns and change responses to perform different tasks without any manual instruction from the users. For example, a smart thermostat can learn the usage pattern of a user and adjust the temperature automatically to save power.

A smart device can have all or a subset of the aforementioned features. These features of smart devices are linked together with one common component- *sensors in smart devices*. For instance, an embedded temperature sensor can be used to trigger a smart thermostat at a pre-defined temperature which represents sensing, automation, and self-learning features of smart devices. Again, external sensors can be connected with smart devices using different communication protocols (e.g., ZigBee, Z-Wave, BLE, etc.) or via cloud [sam17]. An external presence sensor can be configured with a smart thermostat to turn on whenever a user enters a room which depicts context-awareness. Users can also control smart devices remotely that can be associated with the embedded sensors to automate on-going tasks. Hence, sensors in smart devices are the key components that enable aforementioned salient features in smart devices.

2.2 Smart Device Architecture

It is important to understand the smart device architecture to better visualize the sensor-based threats and attacks. The smart device architecture can be illustrated in four working layers (sensing, communication, data processing, and application layer) as shown in Figure 2.1. A smart device may have all the layers or a subset of these

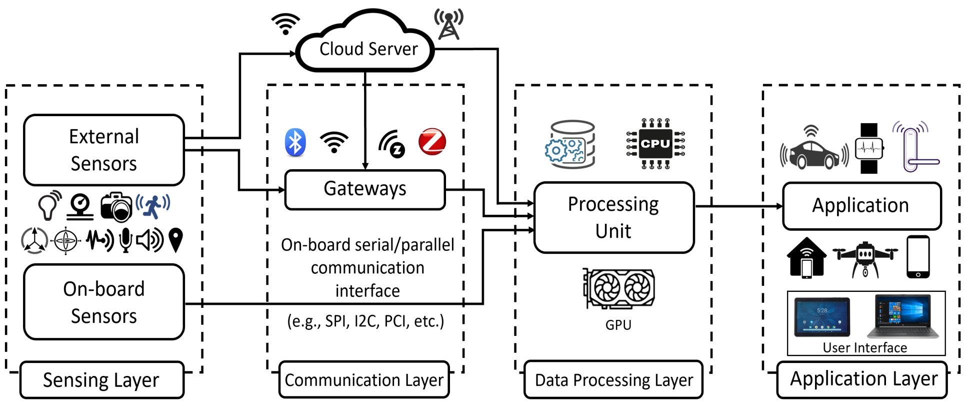


Figure 2.1: Smart device architecture layers and components. Some smart devices may have all the layers (e.g., smart thermostat) or a subset of these layers (e.g., smart sensors).

layers. For example, a smart sensor can have sensing, communication, and data processing layer [M+08] whereas a smart phone can have all four layers. Each of these layers are explained below.

2.2.1 Sensing Layer

The main purpose of the sensing layer is to identify any phenomena in the devices' peripheral and obtain data from the real world. This layer consists of several sensors, where multiple sensors are typically used together by applications to collect various data [KKZK12]. The sensing layer of smart device ecosystem can consist of both on-device sensors and external independent sensors. In both cases, sensors are usually integrated through sensor hubs [PJZ+13]. A sensor hub is a common connection point for multiple sensors that accumulate and forward sensor data to the processing unit of a device. A sensor hub may use several transport mechanisms (e.g., Inter-Integrated Circuit (I2C) or Serial Peripheral Interface (SPI)) for data flow between sensors and applications. For on-device sensors, the sensor hub uses Inter-Integrated Circuit (I2C) or Serial Peripheral Interface (SPI) to forward sensor data to the data processing layer. For external independent sensors, sensor data are forwarded to the cloud server from the sensor hub and smart devices can accumulate these data from the cloud server using the network layer. Sensors in smart devices can be classified into three broad categories:

Motion Sensors. Motion sensors measure the change in motion as well as the orientation of the devices. There are two types of motions one can observe in a device: *linear* and *angular* motions. The linear motion refers to the linear displacement of a smart device while the angular motion refers to the rotational displacement of the device.

Environmental Sensors Sensors such as light, pressure, etc. are used to sense the change in environmental parameters in the device's peripheral. The primary purpose of using environmental sensors is to help the devices to take autonomous decisions according to the changes in a device's peripheral. For instance, environment sensors are used in many applications to improve user experience (e.g., home automation systems, smart locks, smart lights, etc.).

Position sensors Position sensors deal with the physical position and location of the device. The most common position sensors used in smart devices are magnetic sensors and Global Positioning System (GPS). Magnetic sensors are usually used as a digital compass and help fix the orientation of a device's display. On the other hand, GPS is used for navigation purposes.

A detailed description of different sensors is given in Table 6.8.

2.2.2 Communication Layer

The communication layer acts as a channel to transfer data collected in the sensing layer to other connected devices. In addition, the communication layer also establishes a connection between the device and cloud server to accumulate data from the external

Sensor Type	Sensor Name	Description
Motion Sensors	Accelerometer	• An electro-mechanical device which can measure changes in acceleration forces along x, y, and z-axis. • Detects various types of motion like shake, tilt, etc. and adjusts the display of the device accordingly.
	Linear Acceleration Sensor	• An accelerometer which can detect acceleration along one axis without considering the effect of gravitational force. • Helps to adjust the display with motion.
	Gyroscope	• Measures the rate of change of angular momentum in all three axes. • Detects rotational movement of the device and adjusts display accordingly.
Environmental Sensors	Light Sensor	• A photodiode which changes characteristics with the change of light intensity. • adjusts brightness and contrast of the display of the device. • Controls automatic lighting system.
	Proximity Sensor	• IR-based sensor to detect the presence of nearby objects without any physical contact. • Reduces power consumption of the display by disabling the LCD backlight and avoids inadvertent touches.
	Temperature Sensor	• Measures temperature of the device as well as ambient temperature. • Controls and sets the temperature in a device.
	Audio Sensor	• Two types of audio sensor: microphone and speaker. • Microphone: Detects acoustic signal. • Speaker: Playbacks audio signal.
	Camera	• Deals with light intensity, device ambiance, etc. to capture pictures and videos of surroundings. • Provides live video feeds.
	Barometer	• Measures the pressure of the device peripheral.
	Heart rate	• Measures the heart rate of the user in beat per second.
	Fingerprint	• Optical or capacitive scanner to capture the fingerprint of the user. • Provides biometric authentication.
Position Sensors	GPS	• Captures signal from the satellite to infer the location of the device. • Helps in navigation systems.
	Magnetic Sensor	• Measures device's magnetic field with respect to earth's magnetic field. • It is also used to fix display position by considering the magnetic field.

Table 2.1: Sensors available in smart devices.

independent sensors [RDM16, STTPLR13]. In smart devices, the communication layer is realized by using diverse communication technologies (e.g., Wi-Fi, Bluetooth, Zigbee, Z-Wave, LoRa, cellular network, etc.) to allow data flow between other devices within the same network. The Communication layer also simplifies remote access to smart devices. For example, a user can control a smart light from different locations using an app on a smartphone. For on-board sensors, data communication from the sensor to the processing unit is performed by different serial and parallel communication protocols such as Serial Port Interface (SPI), Inter-Integrated Circuit (I2C) protocol, Peripheral Component Interconnect (PCI), etc.

2.2.3 Data Processing Layer

The data processing layer takes data collected in the sensing layer and analyses it to make data-driven decisions. This layer provides processed data to installed applications to perform different tasks. Also, in some smart devices (e.g., smartwatch, smart home hub, etc.), the data processing layer saves the results from previous analysis to improve the user experience. For instance, the data processing layer can learn the contexts and patterns during the user interactions to take autonomous decisions. This layer may share the result with other connected devices via the network layer.

2.2.4 Application Layer

The application layer presents and renders the results of the data processing to the user. In other words, the application layer is a user-centric layer which executes various tasks for the users. There exist diverse applications, which include smart transportation, smart home, personal care, healthcare, etc. [AIM10]. Application layer also provides user interface to the users where users can select, control, and monitor different applications of the smart devices.

2.3 Smart Environment

The term *smart environment* is commonly used to portray a residence comprising numerous connected entities (sensors and devices) that are capable of communicating

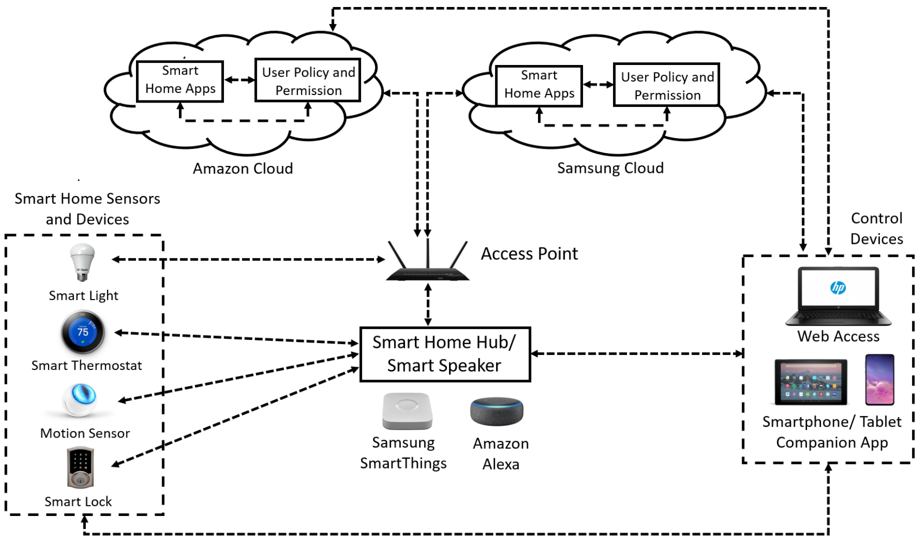


Figure 2.2: A smart home environment and its main components.

with each other and that can be controlled both centrally (via a hub) and remotely (via a smartphone). In Figure 2.2, a typical architecture of a smart environment is shown. Different smart systems such as Samsung SmartThings, Amazon Alexa, and Google Home use similar architecture. The only difference among these platforms resides in the communication protocols used to connect the different components [KPYK18]. A smart environment can incorporate a single-platform structure where all the smart devices and sensors are connected to a common access point (i.e., a hub). Smart environment/settings can also feature a multi-platform architecture where smart devices from different smart platforms share the same physical environment without any interconnection. Both single and multi-platform smart environment have four basic building blocks as shown in Figure 2.2. The first block of the smart environment comprises sensors and devices in the system. These smart devices and sensors are connected to each other via a smart hub. As there is no generic interoperability standard among smart devices, the hub provides a common access point for all the entities in the smart environment. In some cases, hubs also act as a controlling device and allow users to control smart devices using voice commands (e.g., Amazon Alexa, Google Home, etc.). Modern smart device platforms also allow devices to perform autonomous tasks as standalone devices. For example, *LIFX smart light* can directly connect to an access point such as Wi-Fi routers and perform pre-scheduled tasks. The installed smart devices are connected to the hub, which can be further connected to both a cloud backend service and a smartphone/tablet companion app. Users utilize the smartphone app to control and configure the smart devices and sensors or install different apps from the app stores. Indeed, we can group smart environment architectures in two main categories: a cloud-based architecture where the installed apps run in the cloud backend (e.g., SmartThings), and hub-based architecture where the installed apps run the hub locally (e.g., Apple HomeKit). Users may also develop their own apps using the web interface of the cloud backend part of the smart environment. For example, Samsung SmartThings allows its users to publish their own apps and share them with other users [SSDG]. Users can develop their own app or simply copy the source code available online to install the app in their smart environment.

2.4 Existing Sensor Management Systems and Security Needs of Smart Devices

Modern smart devices create a many-to-many relationship between apps and sensors that OSes manage. Managing this relation is a hard task and smart device OSes need effective and practical sensor management schemes to ensure secure data flow from the sensors to the apps. In addition, the sensor management in several smart devices (e.g., smart light, thermostat, etc.) also needs to assure a secure and seamless connection with external sensors to perform multiple tasks. Hence, an effective sensor management system is required to manage and ensure the security of all the sensors in the smart devices. In this section, we discuss different security requirements and goals of smart devices and how the existing sensor management systems address these requirements. Furthermore, we also articulate the shortcomings of existing sensor management systems.

To understand the security needs in smart devices, we consider the following smart device use cases. Assume a user, Bob, has several smart devices and sensors installed in his smart home system including smart lock, thermostat, motion sensor, temperature sensor, and presence sensor. Here, temperature and presence sensors are embedded in smart thermostat while the motion sensors are external sensors connected with smart devices using different communication protocols (e.g., ZigBee, Z-Wave, BLE, etc.) or via cloud [sam17]. We assume all the smart devices and sensors are in the same network. Bob installed several smart apps to automate and control tasks in smart devices. For instance, Bob installed an app in the thermostat to automate temperature control using the embedded temperature sensor. Also, Bob configured the external motion sensor with the smart lock to unlock the door with the users' motion. Based on the configurations and installed apps, the following scenarios can happen-

Case 1- An attacker having access to the same network installs a malicious motion sensor without alerting Bob. How can Bob identify the legitimate sensor while configuring the smart lock with the external motion sensor?

Case 2- Bob unknowingly installs a malicious app for the smart thermostat that is trying to access all the embedded sensors (both temperature and presence sensor). How can Bob limit the sensor access of the installed app?

Case 3- An attacker with the access of device peripheral captures the network packets between external sensors and the smart lock using a sniffing device. Additionally, the attacker is trying to change environment parameters (e.g., temperature) to change sensor reading and switch on the thermostat maliciously. How can Bob ensure that the attacker fails to extract any sensitive information from captured sensor-device communication and verify whether the sensor reading is legitimate or not?

Case 4- An attacker having access to the network sends malicious connection requests to the external motion sensor to make it unavailable for performing pre-defined tasks. How can Bob confirm sensor availability while configuring the smart lock with an external motion sensor?

To address these questions, current smart device ecosystem needs, (1) a sensor authentication system to identify fake or compromised sensors, (2) a sensor authorization framework to limit malicious sensor access, (3) Secure data sharing to confirm data confidentiality and integrity in sensors, (4) seamless connectivity to ensure sensor availability. In the following sub-sections, we briefly discuss existing sensor management systems and their shortcoming in addressing the aforementioned security needs.

2.4.1 Sensor Authentication

Smart devices can connect with each other to perform tasks collectively. Although this may increase the functionalities of smart devices, device authentication is needed to ensure secure communication among devices. The network layer of smart devices should have an authentication framework to connect with trusted devices and sensors. Similarly, the sensing layer of smart devices should also have an authentication framework to detect tampered sensors in the device ecosystem.

Although sensor authentication has not been a big concern for on-device sensors, an unauthenticated external sensor can perform malicious activities in connected smart devices [JCW$^+$17]. To authenticate an external sensor in a smart device ecosystem, device fingerprinting can be utilized at the time of pairing between a smart device and an external sensor. Here, we discuss how sensor fingerprinting is implemented in the Samsung SmartThings platform. Samsung SmartThings offers a capability-based sensor management system which can control sensors of several devices from one common platform (a hub or smartphone). When a new external sensor is installed in the system, a pre-defined device handler is used to pair the sensor which specifies the capabilities of the sensor. This device handler also contains the fingerprint of the sensor. A sample device handler snippet is given in Listing 1. Here, a Fibaro motion sensor device handler for the Samsung SmartThings platform is shown. From line 4 to 11 capabilities of the sensor are defined and after initial installation, the sensor can provide these pre-defined functions. From line 13 to line 17, different benign commands are defined which allowed for Fibaro motion sensor in Samsung SmartThings ecosystem. In lines 19 and 20, the fingerprint of the sensor is defined which allows the smart device ecosystem to understand the device type and authenticate the sensor at the time of installation. This fingerprint is hard-coded in the device handler and can be manually modified to create new handler.

Although capability-based sensor management provides automatic authentication of the connected external sensor at the installation time, the hard-coded capabilities and fingerprint in the device handler can be easily altered. The device handler can be changed manually and an attacker can easily create a fake device handler to trick smart device user to install a compromised sensor in the smart device ecosystem. Attackers can also exploit the sensors by mimicking the hard-coded fingerprint in a compromised or fake sensor [CBS+18].

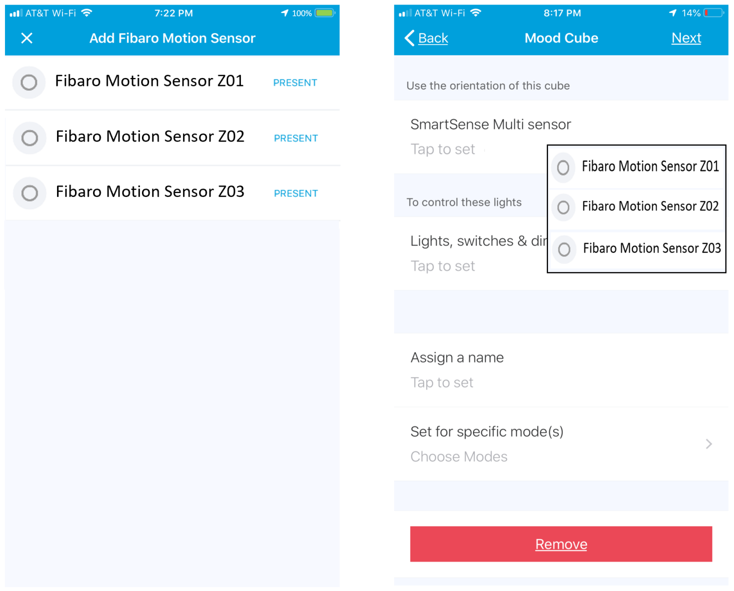
Listing 2.1: An example device handler of Fibaro Motion Sensor

```
 1 metadata {
 2 definition (name: "Fibaro Motion Sensor", namespace: "smartthings", author: "SmartThings", ocfDeviceType:
        "x.com.st.d.sensor.motion", runLocally: true, minHubCoreVersion: '000.021.00001',
        executeCommandsLocally: true)
 3 {
 4           capability  "Motion Sensor"
 5           capability  "Temperature Measurement"
 6           capability  "Acceleration Sensor"
 7           capability  "Configuration"
 8           capability  "Illuminance Measurement"
 9           capability  "Sensor"
10           capability  "Battery"
11           capability "Health Check"
```

```

        command "resetParams2StDefaults"
        command "listCurrentParams"
        command "updateZwaveParam"
        command "test"
        command "configure"

        fingerprint mfr:"010F", prod:"0800", model:"2001"
        fingerprint mfr:"010F", prod:"0800", model:"1001"
        }
```



(a) Adding sensor in the system

(b) Adding sensor in an app

Figure 2.3: Sensor authentication and automation in Samsung SmartThings.

Furthermore, after initial authentication, all the sensor from the same vendor is visible to any connected users to add automation rules [SBC+19]. Hence, an adversary with access to the smart environment can use any installed sensors to add malicious automation rules. Figure 2.3 shows the app installation process in the Samsung SmartThings platform. Here, three different Fibaro motion sensors are available to the users and users can choose any of these sensors to create new automation rules. The current ecosystem does not allow any security measure to restrict specific sensors after initial authentication. Again, if any of these three sensors is compromised, it can be used as a platform to attack connected devices sharing the smart environment.

2.4.2 Sensor Authorization

Modern smart devices use different apps to perform multiple tasks. These apps use multiple sensors to execute a task efficiently. At run-time, installed apps can ask for sensor access and it is necessary to check whether the requested access is legitimate as apps can use sensors for malicious purposes. For example, a simple flashlight app in the smartphone can access the motion sensor data which is irrelevant to the function of the app and can leak the information surreptitiously [SAU19a]. Smart devices should have a robust authorization framework to limit these unauthorized sensor accesses. Sensor authorization can be implemented in both the sensing and application layers. The sensing layer authorization can bind sensors with the apps while the application layer authorization can offer user control over sensors [PSJA15, FHE+12].

Current smart device OSes offer a permission-based sensor management system to control on-device sensor authorization at app installation time and run-time [iOSa, Andb]. Here, we briefly discuss the Android sensor management system as Android OS holds the highest market share in the smart device domain (approximately 37%) [mar]. Whenever an application wants to access a sensor in the OS, it has to communicate via a sensor manager framework (Figure 2.4). An application first sends a request to the sensor manager to register the desired sensor which invokes *ListenerService* service for the application. After receiving the request, the sensor manager creates a *ListenerService* for the application and maps the request with the designed sensor driver to acquire sensor data. If more than one App requests access for the same sensor, the sensor management system runs a multiplexing process to register one sensor to multiple Apps. This data acquisition path from the application to the sensor driver is initiated by the Hardware Abstraction Layer (HAL) which binds the sensor hardware with the device driver. The sensor driver then activates the requested sensor and creates a data flow path from the sensor to the app [andc].

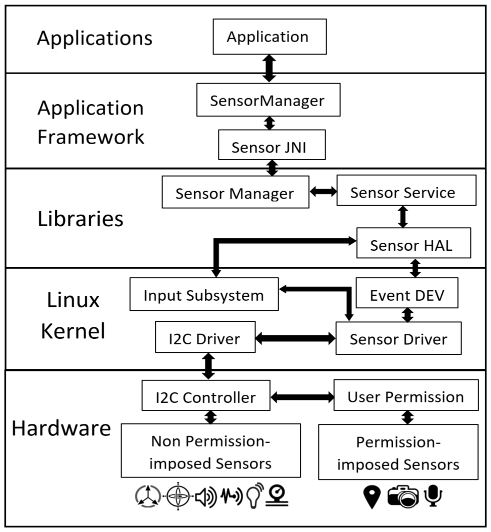


Figure 2.4: Example sensor management system for Android.

On the other hand, Windows and Blackberry OSes use Sensor Class Extension to connect sensor hardware with the device driver [win, bla]. Windows OS also uses the User Mode Driver Framework to detect sensor access request and create a data acquisition path between sensor API and the APP. In iOS, the sensor management system is divided into four core services: Core Motion, Core Audio, Core Location, and Core Video [iOSb]. The Core Motion service provides access to the motion sensors and some of the environmental sensors (e.g., barometer, light, proximity, etc.). The audio sensors (microphone and speakers), GPS, and the camera can be accessed via the Core Audio, the Core Location, and the Core Video services, respectively. These services provide data flow between the sensors and their apps according to the requests.

However, the main shortcoming of the permission-based sensor authorization is the dependence on the user's consent for sensor access. In most smart devices, permission-based sensor authorization is implemented for a subset of the supported sensors (e.g., GPS, camera, audio sensor). Whenever an application is installed in a smart device, it asks the users to grant permission to access various sensors. Thus, malicious applications may trick the user into allowing access to sensitive sensors to launch sensor-based attacks [SZZ+11, TRCK13, SAU17]. Users are typically unaware of what the malicious applications actually do with the sensed data [PAS+16, PRS+17]. Furthermore, permissions are imposed on selected on-device sensors only (e.g., camera, microphone, and GPS) and other sensors are automatically included without any explicit permission. Thus, applications can easily access other no-permission-imposed sensors such as accelerometer, gyroscope, light sensor, etc., as discussed in the following sections in further detail. These sensors can be exploited maliciously and various sensor-based threats (e.g., information leakage, denial-of-service, etc.) can be launched on smart devices [SPYG15, OHD+12, MVCT11]. Additionally, for external sensors, the smart device ecosystem offers one-time sensor authorization at the time of sensor installation. After the initial installation, any connected smart device in the same network can access the external sensor without any additional authorization step.

2.4.3 Data Confidentiality and Integrity

One major concern is to keep the collected sensor data secure in smart devices. Smart devices use multiple sensors to perform a task and recent studies have shown that user activities on a smart device can be inferred using the sensor data [SAU19a]. The current smart device ecosystem implements different encryption methods in the network layer to encrypt sensor data before sharing with the devices. For example, Azure IoT suite, Amazon AWS, and Weave use SSL/TLS protocol to ensure secure communication [ARC18]. Moreover, smart devices using ZigBee protocol use 128-bit AES encryption for secure communication [AFA+18]. However, most of the existing encryption schemes are available for communication between external sensors and smart devices or cloud communication. Some smart device platforms (e.g., Apple

HomeKit, Weave) allows disk encryption to secure saved sensor data. But any app running in the smart devices can access these encrypted data, even collect unencrypted data from the on-device sensors [LPMS12]. These sensor data can be further processed to gain sensitive user information such as PIN code for the devices, typed information, even on-going tasks on a device [SAU17].

2.4.4 Sensor Availability

To perform sensor-dependent tasks, smart devices should have uninterrupted sensor access. This requires sensor availability to the application layer of the devices from the sensing layer. Sensor availability is more important in external sensors than on-device sensors as attackers can target the network layer to perform a Denial-of-Service attack. The current smart device ecosystem offers firewall rules to filter unauthorized and malicious service requests to avoid unauthorized sensor access and avoid buffer overflow [KMP17]. One possible solution can be fine-grained access control systems in the application layer to ensure continuous data availability to legit *app requests*. However, the existing schemes cannot detect sensor unavailability caused by forced changes in the sensors (e.g., hacking gyroscope using acoustic signals [SSK$^+$15]).

2.4.5 Existing Sensor Management Systems and Their Shortcomings

Although existing sensor management systems in smart devices acknowledge the needs of securing sensors by addressing sensor authentication, authorization, and availability, there are several shortcomings that can be easily exploited by sensor-based threats.
(1) User Dependency. Existing sensor authorizations depend on user permission where users are asked to allow or deny sensor access permission to an app at instal-

lation time or run-time. However, no information about the nature of sensor usage is presented to the users. Hence, an app can easily trick the users to get desired sensor authorization and abuse sensors for malicious purposes [SA13, SZZ$^+$11].

(2) Selective sensor authorization. Existing sensor management systems impose permission-based sensor authorization for selective sensors such as microphone, camera, and GPS. However, any installed app can access other sensors such as motion, light, magnetic, and proximity sensors without any explicit user permission. Attackers can exploit this limitation to get access to sensors and perform malicious activities including keystroke inference [AHIN13], eavesdropping [MBN14], etc.

(3) Passive sensor sniffing. As smart devices allow external sensor integration to perform various tasks, it is possible to capture the network traffic between sensors and smart devices without interrupting normal operation. Also, both embedded and external sensors in a smart device are sensitive to environmental parameters which can be captured by a nearby smart device. For instance, typing in a keyboard creates a tap noise which can be captured by the microphone of a nearby smartphone [ZZT09, WLRC15]. Attackers can extract sensitive information from captured traffic and sensor data even if proper encryption schemes are used to protect confidentiality [AFA$^+$18, SBAU19]. Hence, current sensor management systems cannot protect sensor abuse from passive sniffing.

(4) Transitive access. Smart devices create a network of devices or smart environment where several devices are connected with each other to perform multiple tasks. Here, a newly installed smart device becomes automatically visible and can access other devices and sensors without any explicit privilege. As current sensor management systems use hard-coded capabilities and fingerprint to authenticate devices and sensors, attackers can introduce a compromised or fake device to capture sensitive sensor information and inject false data in the system to perform malicious activities [IoT17, JCW+17].

(5) Indirect sensor data injection. Current sensor management systems do not offer any verification method to check whether a sensor input is valid or not. As a result, an attacker can target to maliciously change or control environment parameters such as light intensity or magnetic field to spoof sensor data, trigger malicious activities, or interrupt normal device activities. For instance, an attacker can use inaudible acoustic signals to trigger a voice command in voice-assisted devices and interrupt drone operations [SSK+15, YZJ+19].

CHAPTER 3
LIERATURE REVIEW

In this chapter, we present the related work that are closely related to the research work presented in this dissertation.

3.1 Sensor-based Threats and Security Solutions in Standalone Smart Devices

Sensor-based threats [USB14] on smart devices have become more prevalent than before with the use of different sensors such as user's location, keystroke information, etc. Several works [SUCB13] have investigated the possibility of these threats and presented different potential threats in recent years. However, compared to reported threats, only few prior works presented comprehensive security solutions to protect the sensors from malicious attacks. In the following subsections, we discuss existing sensor-based threats and proposed security frameworks to mitigate the effect of these threats to smart devices.

3.1.1 Sensor-based Threats to Standalone Smart Devices

One of the most common threats is keystroke inference in smart devices. When a user types in the keyboard, motion sensor readings (i.e., accelerometer and gyroscope) change accordingly [CC12]. As different keystrokes yield different, but specific values in motion sensors, typing information on on-screen keyboard can be inferred from an unauthorized sensor such as motion sensor data or its patterns collected either in the device or from a nearby device can be used to extract users' input in smart devices [SPA+18, SPYG15, Ngu15, WLRC15]. The motion sensor data can be analyzed using different techniques (e.g., machine learning, frequency domain analysis, shared-memory access, etc.) to improve the accuracy of inference techniques such as [ASBS12, MVBC12, PSM15, MSS16, MAJH16]. Another form of keystroke inference threat can be performed by observing only gyroscope data. Smart devices have a feature of creating vibrations while a user types on the touchpad. The gyroscope is sensitive to this vibrational force and it can be used to distinguish different inputs given by the users on the touchpad [NSN14, CC11, MBN14]. Recently, ICS-CERT also issued an alert for accelerometer-based attacks that can deactivate any device by matching vibration frequency of the accelerometer [CA17, SSK$^+$15].

Light sensor readings also change while a user types on smart devices; hence, the user input in a smart device can be inferred by differentiating the light sensor data in normal and typing modes [Spr14]. The light sensor can also be used as a medium to transfer malicious code and trigger message to activate a malware [HSH$^+$13]. The audio sensor of a smart device can also be exploited to launch different malicious attacks (e.g., information leakage, eavesdropping, etc.) on the device. Attackers can infer keystrokes by recording tap noises on touchpad [FKK10], record conversation of users [SZZ$^+$11], transfer malicious code to the device [SUCB13], or even replicate voice commands used in voice-enabled different Apps like *Siri*, *Google Voice Search*, *Amazon echo*, *Google Smart Home* etc. [DLZZ14]. Cameras of different smart devices can also be used to covertly capture screenshot or video and to infer information about surroundings or user activities [SA13, MLMK15, SKSP14]. GPS of a smart device can be exploited to perform a false data injection attack on smart devices and infer the location of a specific device.

3.1.2 Solutions to Sensor-based Threats in Standalone Smart Devices

Although researchers identified different sensor-based threats in recent years, no complete security mechanism has been proposed that can secure sensors of a smart device. Most of the proposed security mechanisms for smart devices are related to anomaly detection at the application level which are not built with any protection against sensor-based threats [WH14]. On the other hand, different methods of intrusion detection have been proposed for wireless sensor networks (WSN) [PAL$^+$08], but they are not compatible with smart devices. Xu et al. proposed a privacy-aware sensor management framework for smartphones named *Semadroid* [XZ15], an extension to the existing Android sensor management system where users could monitor sensor usage of different Apps and invoke different policies to control sensor access by active Apps on a smartphone. Maiti et al. proposed a real-time activity detection framework to identify user activity on a smart device using motion sensor and allow motion sensor access based on the detected activity [MAJH16]. Petracca et al. introduced *AuDroid*, a SELinux-based policy framework for Android smartphones by performing behavior analysis of microphones and speakers [PSJA15]. AuDroid controls the flow of information in the audio channel and notifies users whenever an audio channel is requested for access. An extension of this work is *AWARE*, an authorization framework to secure privacy-sensitive sensors from malicious applications [PRS$^+$17]. *AWARE* considers both application requests and user interface to identify malicious user inputs in operation bindings for microphone and camera. Jana et al. proposed *DARKLY*, a trust management framework for smartphones which audits applications of different trust levels with different sensor access permissions [JNS13]. Darkly scans for vulnerability in the source code of an application and tries to modify the run-time environment of the device to ensure the privacy of sensor data.

3.2 Existing Security Threats and Detection Techniques in Connected Smart Environment

Smart environments such as smart home systems, smart offices, smart factories, etc. have become very popular with their user-centric customization options and third-party app development. Developers have offered different apps to increase the functionalities of smart devices in a connected smart environment. Nonetheless, the nature of the app-based models introduces several malicious threats to the smart environment. In the following subsections, we discuss existing security vulnerabilities that target sensors in smart environment to perform malicious activities. We also summarize existing security solutions proposed by the research communities and developers to mitigate these threats.

3.2.1 Security Vulnerabilities in Connected Smart Environment

In recent years, several works have outlined the security threats to connected smart environment (e.g., smart home systems) [NSG+14, Sch15, DKL13]. These threats mainly focus on three smart environment components: communication protocols, devices, and apps. As the concept of smart environment/system is still evolving, there are several implementation flaws in the communication protocols for smart systems. Attackers may exploit these flaws to leak sensitive user information from smart devices. Several prior works have reported multiple implementation flaws of smart device's communication protocol that can be abused to leak sensitive user information [FG13, Sea15, MV15]. For instance, an attacker can capture the network packets covertly using simple sniffing devices and extract shared information, even from the encrypted traffic [BJD16, HLM+16, AFA+18]. Fernandes et al. reported several design flaws in popular smart home platform (e.g., Samsung SmartThings) that includes system and app level vulnerabilities [FJP16]. Chi et al. showed that it is possible to exploit smart home platform by triggering malicious activities from legitimate user action [CZDY18]. As smart devices use multiple apps to automate different tasks, researchers showed that it is possible to execute a malicious task while performing legitimate user actions in connected devices. Moreover, current smart device platforms use smartphone as user interface and controlling device which can also be used to launch attacks to smart devices [SCEB16, FJP16]. Jia et al. reported the existence of several malicious apps that can be migrated from smartphone and IoT platforms to connected smart environment such as smart home [JCW+17]. Recently, a group of researchers published an online repository, *IoTBench* [IoT17, CBS+18], which revealed several malicious apps for existing smart platforms including Samsung SmartThings (19 malicious apps) and OpenHab (33 third-party rules).

3.2.2 Existing Security Solutions to Smart Environment

While researchers and developers reported these various threats to smart devices and apps in recent years, there is no comprehensive security solution that address these threats and secure the connected smart environment. Developers have introduced several policy-based security measures to limit unauthorized access to smart devices, sensors, and apps which depends completely on user decisions [TZL+17,SGV+15]. On the other hand, researchers proposed several countermeasures to bolster the security of smart environment by implementing encrypted data traffic in smart home communication protocols (e.g., ZigBee, Z-Wave, Wi-Fi, etc.) [ARS+17, DKJG17, MAbM14]. Additionally, researchers proposed several static and forensic analysis tools to detect threats in application level of smart devices and sensors [BSAU18, CBS+18]. In the followings, we discuss existing security mechanisms proposed by the researchers and their shortcomings:

- *Permission-based approach.* Most of the smart device platforms use a permission-based app management where users are asked to configure and allow permissions (device access, sensor access, etc.) at installation time. However, once user approves the permissions, smart platforms/systems does not provide any explicit information about how the app is using the granted accesses. This existing coarse-grained permission model could lead to unauthorized device access and sensitive information leakage in connected smart environment [FJP16]. Jia et al. introduced *ContextIoT*, a context-aware permission model to restrict unauthorized device access and detect malicious activities in smart home systems [JCW+17]. Their proposed model creates a run-time context of each app event, and asks for user permission before executing any unknown activity in the smart home system.

- *Policy and configuration analysis.* Several policy-based security measures were proposed to limit unauthorized access to smart devices and sensors in a connected smart environment [TZL+17, SGV+15, CWR13]. Similar to permission-based approaches, these solutions depend on user decision. Mohsin et al. presented *IoTSAT*, a framework to analyze threats on smart home/environment using device configurations and enforced user policies [MAH+16]. IoTSAT creates a behavioral model based on device configurations and network policies and compares it with enforced policies to identify unusual activities in the smart environment [MAH+16]. Wang et al. introduced *iRuler*, an automation rule analysis framework to detect inter-rule vulnerabilities in smart apps [WDY+19]. *iRuler* uses natural language processing techniques to detect trigger-action information flow and detect security risks in implemented applets in IFTTT platform. Similar to this work, Babun et al. proposed *IoTwatch*, a run-time analysis tool to identify privacy violations in smart apps [BCMU19]. *IoTwatch* collects the privacy settings of the users at installation time and collects run-time privacy sensitive data sharing in smart devices, sensors, and apps to detect privacy violations using natural language processing.

- *Network analysis.* Researchers have proposed several network analysis techniques to detect malicious activities in smart environment. Yamauchi et al. presented a network-based intrusion detection system (IDS) that uses benign network packets generated from user activities to detect malicious events in a connected smart home system [YOM+19]. Researchers implemented a network observer in the home gateway that learns user behaviors from the network traffic and detects malicious user commands from learned behavior. Anthi et al. proposed a supervised intrusion detection system to detect malicious and known network attacks in smart devices [AWS+19a]. In a recent work, Apthorpe et al. showed that it is possible to mitigate privacy leakage from smart home traffic by implementing traffic shaping [AHR+19].

- *Static analysis.* Recently, static analysis of smart apps have been proposed to detect information leakage and cross-app interference. Berkay and Babun et al. introduced a static analysis tool, *SaINT*, to track sensitive information in smart apps [CBS+18]. SaINT performs source code analysis to trace all the sensitive information from sources to sinks and identify potential information leakage. Chi et al. proposed a static analysis tool to extract app context from smart apps to detect cross-app interference in a connected smart environment [CZDY18]. Their proposed model considers data sources and sinks in an app to track the information flow and extracts the rules presented in an app. The extracted rules are saved and compared when a new app is installed to find possible conflicts of operations.

- *Forensic analysis.* Forensic analysis of smart platforms/settings has been proposed to identify malicious events in a smart environment. Wang et al. proposed a security tool, *ProvThings*, which logs run-time data from smart apps and perform provenance tracking to detect malicious activities [WHBG18]. ProvThings implements data collector in the source code of an app to identify data flows and create provenance graphs. These graphs are used to infer app functionalities and detect malicious activities in the smart systems. Babun et al. proposed *IoTDots*, a forensic analysis tools which can detect user behavior from logged data in a smart environment [BSAU18]. IoTDots collects data from smart apps and performs a context analysis to detect violation of security policies in a smart environment.

3.3 Smart Device and Sensor Access Control in Multi-user Smart Environment

In a smart environment, multiple users have access to multiple devices and sensors, typically through a dedicated app installed on a mobile device. Traditional access control mechanisms consider one unique trusted user that controls the access to the devices. However, multi-user multi-device smart settings pose fundamentally different challenges to traditional single-user systems. For instance, in a multi-user environment, users have conflicting, complex, and dynamically changing demands on multiple devices, which cannot be handled by traditional access control techniques. Moreover, the coarse-grained traditional access control results in over-privilege access in multi-user smart environment. Rather than providing fine-grained user access control, most of the prior works emphasize on limiting malicious activities via controlling app access [DZL+17, FPR+16, WHBG18, SBAU19, NSRU19] and studying device behaviors to detect malicious activities [BCMU19, LBAU17]. Moreover, several works focus on device access control and authentication on an IoT network for single-user scenarios [CPG+15, AHD+16, RSYS16, JCB16, RSS+17]. In a recent work, He et al. present a detailed smart home user study that portrays users' concerns of fine-grained access control in multi-user smart environments [HGP$^+$18]. Zeng et al. discuss their findings related to security and privacy concerns among smart home users [ZMR17]. In both works, smart home users clearly raise their concerns regarding the need of access control mechanism in smart environment such as smart home. In addition, these studies also summarize several design specification to reflect users' needs in an access control mechanism. Matthews et al., also points out relevant issues with smart home users that share the same devices and accounts [MLT$^+$16]. However, no explicit solution for multi-user access is proposed in any of these works.

In other works, researchers explore different access control strategies when multiple users share a single IoT device. Liu et al. suggested a user access framework for the mobile phone ecosystem called *xShare*, which provides policy enforcement on file level accesses [LRH$^+$09]. Ni et al. presented *DiffUser*, a user access control model for the Android environment based on access privileges [NYB$^+$09], which is only effective for a single device. Tyagi et al. discussed several design specification needed for multi-party access control in a shared environment [TSRG16]. Aside from these works, there are few prior proposing access control systems for multi-user multi-device smart environment. Gusmeroli et al. suggested a capability-based access control for users in a multi-device environment [GPR13]. However, this system is not flexible enough to express the real needs of the users. Jang et al. presented a set of design specification for access control mechanism based on different use scenarios of multi-user smart home system [JCP17]. Schuster et al. proposed a situation-based access control in the smart home system which considers different environmental parameters [SST18]. Here, the authors considered state of the device along with the location of the users to determine a valid access request. However, this work does not solve the conflicting demands of multiple users. Yahyazadeh et al. presented *Expat*, a policy language to define policies based on user demands [YPHC19]. In a recent work, Zeng et al.

built an access control prototype with different access control options for smart home users [ZR19]. Here, the authors considered four different access control mechanisms and assessed in a month-long user study among seven households to understand the users' needs and improve the design. However, they did not implement the framework in real-life systems and did not consider user conflicts while operating in a multi-user smart environment.

CHAPTER 4

TAXONOMY AND IMPACT ANALYSIS OF SENSOR-BASED THREATS TO SMART DEVICES

4.1 Introduction

In this chapter, we provide a detailed taxonomy of existing sensor-based threats to smart devices and applications. Sensor-based threats are emerging threat vectors to smart devices and it is important understand the characteristics and impacts of these threats to build secure smart devices and applications.

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

- We categorize sensor-based threats in smart devices based on different security requirements and present a comprehensive threat model
- We provide a detailed taxonomy of sensor-based threats and attacks to smart devices and discuss the mechanisms and effectiveness of the attacks in a detailed way. We also summarize the effectiveness of the threats and attacks based on known vulnerability metrics.
- We identify several open issues and discuss future research that could contribute to secure smart devices against emerging sensor-based threats.

4.2 Threat Model

In this section, we classify sensor-based threats in smart devices based on threat type, attacker capbilities and security requirements of smart devices.

4.2.1 Types of Sensor-based Threats

A sensor-based threat exploits on-device or external sensors in a smart device ecosystem to perform attacks such as false data injection, eavesdropping, information leakage, etc. to jeopardize the proper operation of the device. Based on the nature of the threats, sensor-based threats can be categorized in two categories.

- *Passive threats.* Passive sensor-based threats refer to the malicious sensor activities in smart devices without obstructing the normal operation of the device. For example, a malicious app installed in a smart device can run in the background and observe the sensor behavior to infer the ongoing task in the device [MJ18]. Passive sensor-based threats can accomplish its malicious intents by performing malicious activities within a smart device or by utilizing another near-by smart device.
- *Active threats.* Active sensor-based threats obstruct the normal operation of the smart device to perform malicious activities. An active sensor-based threats can directly abuse an on-board or external connected sensor by spoofing the sensor reading [IoT17] or obstructing sensor signals using external device [SSK+15].

4.2.2 Attacker's Capabilities

To perform sensor attacks, we consider adversaries have the following capabilities in terms of device access, security privilege, and processing capabilities.

- *Device access.* An adversary may need device access to perform malicious sensor activities in a smart device. Based on the type of access needed for an adversary, we categorized three different access types - direct access, transitive access, and peripheral access. In direct access, an adversary can directly access the sensors in a smart device to perform malicious activities. For example, a malicious app installed in a smartphone can directly access on-board sensors and collect data to infer sensitive information [SAU17]. For transitive access, an adversary uses access to a smart device or sensor to perform malicious sensor activities in a targeted smart device. For example, in a smart home environment, an adversary can get access and strobe a smart light to change the output of the light sensor and a targeted smart light [IoT17]. In peripheral access, an adversary implanted in a device (affected device) can perform malicious sensor activities in any smart device in its peripheral. Here, the affected device and the targeted device share the same environment, but are not connected with each other. For example, an adversary can use the audio sensor of a smartphone to eavesdrop to another smartphone in close proximity to infer keystrokes.

- *Security privilege.* An adversary needs different levels of security privileges to perform malicious sensor activities in a smart device. For instance, to perform eavesdropping, an adversary needs minimum (low) privileges in the targeted device while for false data injection in a sensor, an adversary needs maximum privileges to access the sensor. In this work, we consider an adversary can have both privileges to classify the sensor-based threats and attacks correctly.

- *Processing capability.* In smart devices, sensors mostly act as a triggering component to initiate automated applications. The sensed information in the smart device sensors often needs further processing to extract important information. Hence, an adversary needs processing capabilities to perform malicious sensor activities in smart devices. Based on the adversary's goal, the processing capacity may vary. For example, an adversary extracting keystrokes from motion sensors needs higher processing capabilities than an adversary recording phone conversation secretly off the device [SAU19a].

4.2.3 Threat Model

In this dissertation, we consider sensor-based threats and attacks in four working layers (sensing, communication, data processing, and application) of the smart devices. We consider adversaries that try to abuse the sensors to perform malicious tasks as a sensor-based threat. Additionally, this work considers passive threats to the sensors that do not disrupt normal functionalities of the smart devices. An adversary can be installed in a smart device to get access to the embedded sensors of the device or external sensors connected to the smart device. An adversary that has access to the peripheral of a targeted smart device to sniff the sensor data and network traffic is also within the scope of this work. Furthermore, we consider an adversary that can have direct or indirect access to the sensors of the smart devices to capture sensor data for further analysis. Note that a physical sensor abuse or sensor tampering that could lead to physical damage of the smart devices is not considered and outside the scope of this work. Specifically, we consider the following threats in our threat model.

- **Information Leakage.** An active or passive adversary may try to access the sensor data to steal sensitive information such as typing information, unlock code, PIN code, etc.
- **Transmitting Malicious Sensor Command.** An adversary may try to abuse sensors to transmit malicious sensor command to trigger malicious activities in a smart device.
- **False Sensor Data Injection.** An adversary may try to inject false sensor data to disrupt the normal functionalities of the smart devices.
- **Denial-of-Service.** An adversary may establish a sensory channel between on-device sensors and external entities (e.g., device, signal generator, etc.) to impede

normal sensor operation which eventually leads to obstructing an on-going task in the smart device.

4.3 Taxonomy of Threats, Attack Methods, and Their Impact

As existing sensor management systems and security schemes cannot provide adequate security to the sensors, attackers can exploit these sensors in various ways. In this section, we provide a detailed discussion about sensor-based threats and attacks to smart devices and survey the existing malicious attacks confirmed by the research community and developers [$\text{SZZ}^{+}11$, TRCK13, PSJA15, PMSJ16, $\text{PAS}^{+}16$, $\text{PRS}^{+}17$].

To understand the severity of sensor-based threats and attacks, we considered several common vulnerability scoring metrics for sensor-based threats in our discussion [Fir19]. These scoring metrics give insights of the characteristics and impact of the threats. Detailed of these metrics are given below.

- **Attack Method (AM).** Attack method reflects how the threats penetrate the smart device to perform malicious sensor events. For sensor-based threats, we consider three methods to assess the severity of the threat- active, passive, and combination.
- **Device Access (DA).** To initiate a malicious sensor activity in a smart device, sensor-based threats need to access the device directly or indirectly. Based on the nature of the threat, we categorize the device access of sensor-based threats in three categories - *direct access*, *transitive access*, and *peripheral access*. Direct access refers to the threats that need access to the targeted device. In transitive access, a sensor-based threat can preform malicious sensor activities by accessing a device that is connected with the targeted device. For example, a sensor-based threat

can perform malicious activities in a smart light by accessing a connected light sensor [SBAU19]. A sensor-based threat can also execute malicious sensor-activities by accessing the peripheral of the targeted device. For instance, keystrokes in a smartphone can be captured by a nearby smart speaker or smart watch [MJHB18].

- **Attack Complexity (AC).** Sensor-based threats and attacks can target one single sensor or multiple sensors to perform malicious tasks in smart devices. As abusing more than one sensor at a time may require immense effort from the attacker side, we consider two different levels (high and low) of complexity for sensor-based threats.
- **Required Privilege (RP).** To get access to the sensors for initiating malicious activities, sensor-based threats need to exploit existing security mechanisms of smart devices. As we explained in Section 2.4, sensors in a smart device can be categorized in two categories based on access permission: no-permission imposed sensor and permission imposed sensor. To access the no-permission based sensors, an adversary needs no excessive privilege while an adversary targeting permission imposed sensors needs high privilege. Hence, based on permission needed for accessing sensors in a smart device we consider two categories - high privilege threats (need excessive permission) and low privilege threats (need no permission).
- **User Interaction (UI).** This scoring metric portrays the need of user interaction other than the attacker to compromise the sensor functionalities in smart devices. Low user interaction indicates the higher impact of the sensor-based threats to smart devices.
- **Attack Impact (AI).** This scoring metric represents the impact of the sensor-based threats to various security requirements of the smart device. For sensor-based threats, we choose three important security features that might get affected - confidentiality, integrity, and availability.

- **Success Rate (SR).** Success rate of the sensor-based attacks is the fraction or percentage of success of an attack to perform malicious activities in a smart device among a number of attempts. We categorize this metric in three categories - high (success rate >90%), medium (success rate 70-90%), and low (success rate <70%).

In the following sub-sections, we summarize existing sensor-based threats and attacks in four broad categories based on the purpose and nature of the threats (presented in Section 4.2).

4.3.1 Information Leakage

Information leakage is the most common sensor-based threat for smart devices and their applications. Sensors on smart devices can reveal sensitive data like passwords, secret keys of a cryptographic system, credit card information, etc. This information can be used directly to violate user privacy or to build a database for future attacks. An adversary (e.g., malicious app) can get access to the sensor data by exploiting vulnerabilities of existing sensor managements systems such as selective sensor authorization and user dependency (Section 2.4: use case 2). Only one sensor can be enough for information leakage (e.g., eavesdropping using microphone [SZZ+11]) or multiple sensors can be exploited to create a more complex attack (e.g., keystroke inference using the gyroscope and audio sensors [NSN14]). Moreover, sensors of one smart device can be used to leak information from a nearby device (passive information leakage) (Section 2.4: use case 3). In general, information leakage can be accomplished for the purpose of *(1) keystroke inference, (2) task inference, (3) location inference,* or *(4) eavesdropping* as explained below.

Keystroke inference

Keystroke inference is a generic threat to smart devices. Most of the smart devices provide input medium such as the touchscreen, touchpad, keyboard (external or built-in virtual or real). Whenever a user types or gives input to a device, the device tilts and turns which creates deviations in data recorded by sensors (e.g., accelerometer, gyroscope, microphone, light sensor, etc.). These deviations in sensor data can be used to infer keystrokes in a smart device. Keystroke inference can be performed on the device itself or on a nearby device using sensors of the smart device. Keystroke inference can be performed actively (using on-board sensors) or passively (using external sensors). Here, we summarize different keystroke inferences based on the targeted sensors in the smart devices.

Keystroke Inference with Light Sensors - Light sensors in smart devices are usually associated with the display unit. In general, the display unit of the smart devices is touch-sensitive and provides a user interface to take inputs. For a constant state and unchangeable ambiance, the readings of the light sensor are constant. Each time a user touches and uses the touchscreen to interact with the device, he/she tilts and changes the orientation of the device, which causes changes in the readings of the light sensor. Each input may have a dissimilar light intensity recorded by the sensor. These changes in the readings of the light sensor of a device can be utilized to infer keystrokes of that particular device. An attacker can derive the various light intensities recorded by the light sensor by trying several keystrokes in a device and then construct a database. When users put their PINs or type something in the touchpad, attackers can capture the data maliciously from the device and collate these data with the database to decode keystroke information. As an example, some researchers developed a method named *PIN Skimming* to use the data from an ambient light sensor and RGBW (red, green, blue and white) sensor to extract PIN input of the smartphone [Spr14].

Keystroke Inference with Motion Sensors - The main purpose of using the embedded motion sensors (e.g., accelerometer, gyroscope, linear acceleration sensor) in smart devices is to detect changes in motion of the devices such as shake, tilt, etc.. Accelerometer and linear acceleration sensor measure acceleration force that is applied to a device while gyroscope measures the rate of rotation in the devices. In smart devices with user interface (e.g., smartphone, smart watch, tablet, etc.), the value given by the motion sensors depends on the orientation of the device and user interactions (striking force of the finger on the device display, resistance force of the hand, the location of the finger on the touchpad of the device, etc.). Thus, when a user gives inputs to a device, the motion sensors' data changes accordingly. Generally, smart devices use two types of user interface to take user input – on-screen user interface (e.g., touchpad) and external user interface (e.g., keyboard, keypad, etc.). For both user interfaces, input keys are in a fixed position and for a single keystroke, the motion sensors give a specific value [CC12]. As attackers do not need any user permission to access the motion sensors, it is easy to access the motion sensor data.

One common keystroke inference attack can be performed by exploiting accelerometer. As mentioned above, accelerometer gives a specific reading for each user input on a smart device, thus, attackers can build a database of pre-processed accelerometer readings with diverse input scenarios and make a matching vector of sensor data and keystrokes to extract users' input [AHIN13, HGC+19]. The data extracted from these attacks vary from text inputs to PINs and numbers typed in the touchpad which is much more serious as attackers can acquire the PIN or credit card information [SPYG15, ASBS12]. Owusu et al. developed an app named *ACCessory* which can identify the area of the touchscreen by analyzing accelerometer data of smart devices [OHD+12]. *ACCessory* can infer PIN input on smart devices based on the detected area from accelerometer data. Accelerometer data can also be used to infer keystroke from a nearby keyboard. Marquardt et al. presented an attack scenario where accelerometer data of a smart device can be used to guess input on a nearby keyboard [MVCT11]. Whenever a user types on the keyboard, a vibration occurs and the accelerometer of the smart devices can catch this vibration and keystrokes can be identified correctly by analyzing this data [AAUA18].

Another method of keystroke inference can be achieved by analyzing the gyroscope data of a smart device. Gyroscope measures the angles of rotation in all the three axes which vary based on the specific area of the touch on the screen. Many smart devices such as smartphones, tablets, etc. have a feature when users input something on the touchpad the device vibrates and gyroscope is also sensitive to this vibrational force. The orientation angle recorded in the gyroscope and the vibration caused by the input can be used to distinguish inputs given by the users. Moreover, the data of the gyroscope can be combined with the tap sound of each key recorded via the microphone which can increase the success rate of inferring keystrokes [NSN14, CC11, LS19]. The combination of accelerometer and gyroscope data can also be used for keystroke inference which yields more accurate results [XBZ12, MVBC12, Ngu15, HB18, LCY+18, LL19]. Additionally, the use of pattern recognition and deep learning algorithms can improve the success rate of keystroke inference attacks to smart devices [BFG+19].

In most wearables (smart bands, smartwatches, etc.), the motion sensors are utilized for monitoring the movement of the devices. A smartwatch, which is one of the most common wearables, maintains constant connectivity with smartphones via Bluetooth. While wearing a smartwatch, if a user moves his/her hands from an initial position, the motion sensor calculates the deviation and provides the data regarding the change of the position of the smartwatch [JLLA15]. Typing in the touchpad of a smart device while wearing a smartwatch will change the data recorded by the motion sensors of the smartwatch depending on user gestures. For a specific user input interface such as *QWERTY* keyboard of smartphones which has a specific distance between keys, the motion sensors' data of the smartwatch can be used to infer the keystrokes [LZD+15, WLRC15, MJHB15, SDN15]. Modern wearable devices (e.g., Apple Watch 5, Samsung Gear VR, etc.) also provide a user interface where users can provide inputs to the devices. Researchers showed that it is possible to infer the user input in wearables by observing hand movements [LLC+19]. In a recent work, researchers showed it is possible to infer the unlock code of a smart lock from the gyroscope data of a smart watch [MHSJ18].

Keystroke Inference with Audio Sensors - High precision microphones used in smart devices can sense the acoustic signals emanating from keyboards (built-in or nearby) which can be used to infer the keystrokes on a smart device. Asonov et al. proposed an experiment to record the sound of key tapping and infer the correct key from it [AA04]. In this experiment, the attacker is assumed to record the acoustic signal emanating from the device while the user types on the keyboard. Then, the attacker matches this signal with a training dataset recorded stealthily while the same user was typing in the training period.

Zhuang et al. showed that it would be possible to infer keystrokes by just analyzing the acoustic emanation without having a training data set [ZZT09]. In this attack scenario, a specific key is assigned to a pre-defined class according to the frequency of the acoustic signal it generates while being typed. The attacker then takes a ten-minute of recording of the acoustic signal of typing on a keyboard. This recorded signal is analyzed using machine learning and speech recognition feature named *Cepstrum* to match with the previously defined key classes and infer the input of a keyboard.

In another work, Halevi et al. introduced a new technique named *Time-Frequency Decoding* to improve the accuracy of keystroke inference from the acoustic signal [HS12]. In this technique, machine learning and the frequency-based calculations are combined to match the recorded acoustic signal data from a smart device with a training dataset and increase the success rate of the attack scenario. This technique also considers the typing style of users to minimize the error rate of keystroke inference.

Berger et al. divided a PC keyboard in regions based on tap sound generated by keys and modeled a dictionary attack [BWY06]. This attack utilizes signal processing and cross-correlation functions to process acoustic signal emanations from a nearby keyboard. Kune et al. proposed a timing attack on a number pad used in smartphone and ATMs using the audio feedback beeps generated while entering PIN [FKK10]. Inter-keystroke timing and distance between the numbers on the keypad are the main two features which are used to infer the input PIN in this attack. By analyzing the audio feedback recorded using the microphone of a nearby smart device, these two features are extracted and using Hidden Markov Model, the input numbers and PINs are inferred. Lu et al. proposed *KeyListener*, a context-aware inference method to predict the keystroke in QWERTY keyboard of smartphones and tablets using embedded microphones [LYC+19]. *KeyListener* uses a binary search tree algorithm to predict the typed information and achieves over 90% success rate. Similar to *KeyListener*, Shumailov et al. presented an acoustic side-channel attack which uses the tap noises of a virtual keyboard to infer the typed information in a smartphone [SSYA19]. Kim et al. further improved this work by capturing tap noises using multiple embedded microphones and combining the patterns of the acoustic signals [KJL20]. Here, researchers developed a tapstroke detection and localization algorithms which can infer the typed information with 85.4% accuracy. In a recent work, Zhou et al. presented *PatternListener+*, an inference attack to predict the unlock patterns on an Android device using acoustic signal [ZWY+19]. *PatternListener+* uses the speaker of a smartphone to play an inaudible sound and capture the reflected signal from users' fingertips using the embedded microphone. The reflected signal contains information of the hand movement which is further analyzed with a tree structure to infer the pattern of the lock. Backes et al. showed that acoustic signal emanated from a dot matrix printer which was collected by a nearby microphone of a smart device can be analyzed to predict the text printed on a paper [BDG+10]. In the training phase of this attack, words from a list are being printed, the acoustic signal is recorded and the data is stored. The audio signal processing and speech recognition techniques are used to extract the features of the acoustic signal to create a correlation between the number of needles used in the printer and the intensity of the audio signal. In the real attack scenario, the audio signal is captured by a nearby audio sensor and matched with the previous dataset to infer the printed text.

Zhu et al. showed a context-free attack scenario using the keyboard's acoustic emanation recorded in a smartphone to infer keystrokes [ZMZL14]. In this attack scenario, the acoustic signals emanated from the keyboards are recorded by two or more smartphones. For each pair of microphones of smartphones, the recorded acoustic signal strength will depend on the distance between the typed key and the smartphones. By calculating the time-difference of the arrival of the acoustic signal, the position of the key can be inferred.

In a similar attack, Chhetri et al. introduced a method to reconstruct the design source code sent to a 3-D printer [CCAF16]. In this attack scenario, the acoustic signal emanated the 3-D printer is being recorded by a recorder placed in close proximity of a 3-D printer and the recorded file is processed for extracting time and frequency domain

Attack name	Target device	Target sensor	Target layer	Vulnerability metrics[†]						Ref.
				AM	DA	AC	RP	UI	SR	
Pin Skimming	Smartphone	Light	Sensing	●	●	○	○	○	◑	[Spr14]
Text Inference	Smartphone	Accelerometer, Gyroscope	Application	●	●	●	○	○	○	[HB18]
Motion-based keystroke inference	Smartphone	Accelerometer, Gyroscope	Sensing	●	●	●	○	○	○	[CC12]
Keystroke inference on Android	Smartphone	Accelerometer, Gyroscope	Sensing	●	●	●	○	○	●	[AHIN13]
Input extraction via motion sensor	Smartphone	Accelerometer, Magnetometer	Sensing	●	●	●	○	○	◑	[SPYG15]
Accelerometer side channel attack	Smartphone	Accelerometer	Sensing	●	●	○	○	○	◑	[ASBS12]
ACCessory	Smartphone	Accelerometer	Sensing	●	●	○	○	●	○	[OHD+12]
(sp)iphone	Smartphone	Accelerometer	Sensing	○	○	○	○	●	◑	[MVCT11]
Single-stroke language-agnostic keylogging	Smartphone	Gyroscope, Microphone	Sensing	●	●	●	●	○	●	[NSN14]
Touchlogger	Smartphone	Accelerometer, Gyroscope	Sensing	●	●	●	○	●	◑	[CC11]
Taplogger	Smartphone	Accelerometer, Gyroscope	Application	●	●	●	○	●	●	[XBZ12]
I Know What You Type	Smartphone	Accelerometer, Gyroscope, Light	Sensing	●	●	●	○	●	○	[BFG+19]
Type and leak	smartphone	Accelerometer	Sensing	○	○	○	○	●	◑	[LLC+19]
Risk Assessment of motion sensor	smartphone	Accelerometer	Sensing	○	◑	○	○	●	○	[HGC+19]
Infer tapped and traced user input	Smartphone	Accelerometer, Gyroscope	Application	●	●	●	○	○	◑	[Ngu15]
Motion-based side-channel attack	Smartphone	Accelerometer, Gyroscope	Sensing	●	●	●	○	○	◑	[LS19]
When good becomes evil	Smart watch	Accelerometer	Sensing	○	○	○	○	●	○	[LZD+15]
Mole	Smart watch	Accelerometer	Application	○	○	○	○	●	○	[WLRC15]
(Smart) watch your taps	Smart watch	Accelerometer	Sensing	○	◑	○	○	○	●	[MJHB15, MJHB18]
Wristsnoop	Smart watch	Accelerometer	Sensing	○	◑	○	○	●	○	[SDN15]
Inferring Mechanical Lock Combinations	Smart lock	Gyroscope	Application	○	○	○	○	○	●	[MHSJ18]
Inference of private information	Smartphone	Accelerometer, gyroscope	Sensing	●	●	●	○	○	●	[LCY+18]
KeyListener	Smartphone	Microphone	Sensing	●	●	○	●	○	●	[LYC+19]
aLeak	Smart watch	Accelerometer, Gyroscope	Sensing	○	◑	●	○	○	●	[LL19]
Keyboard acoustic emanation	Smartphone	Microphone	Sensing	○	○	○	●	○	◑	[AA04]
Keyboard acoustic emanations revisited	Smartphone	Microphone	Sensing	○	○	○	●	○	●	[ZZT09]
A closer look at keyboard acoustic emanations	Smartphone	Microphone	Sensing	○	○	○	●	○	○	[HS12]
TapSnoop	Smartphone	Microphone	Sensing	●	○	○	●	○	◑	[KJL20]
Dictionary attacks using keyboard acoustic	Smartphone	Microphone	Sensing	○	○	○	●	○	●	[BWY06]
Timing attacks	Smartphone	Microphone	Sensing	●	○	○	●	○	-	[FKK10]
Acoustic Side-Channel Attacks on Printers	Smartphone	Microphone	Sensing	○	○	○	●	○	◑	[BDG+10]
Context-free keyboard acoustic emanations	Smartphone	Microphone	Sensing	○	○	○	●	○	◑	[ZMZL14]
PatternListener+	Smartphone	Microphone, speaker	Sensing	●	●	●	●	●	●	[ZWY+19]
Hearing your touch	Smartphone	Microphone, speaker	Sensing	●	●	○	●	○	○	[SSYA19]
PIN skimmer	Smartphone	Microphone, Camera	Sensing	●	●	●	●	●	○	[SA13]
Juice filming attack	Smartphone	Camera	Application	●	○	○	●	○	-	[MLMK15]
Beware, your hands reveal your secrets!	Smartphone	Camera	Sensing	○	○	○	●	○	◑	[SKSP14]
Smudge attack	Smartphone	Camera	Sensing	○	○	○	●	○	○	[Avi12]
iSpy	Smart security camera	Camera	Application	○	○	○	●	○	●	[RWG+11]
GazeRevealer	Smartphone	Camera	Application	●	●	○	●	●	◑	[WCGS19]
Compromising electromagnetic emanations	Smartphone	Magnetic	Sensing	○	○	○	○	○	●	[VP09]
My Smartphone Knows What You Print	Smart printer	Microphone, magnetic	Sensing	○	○	●	●	○	●	[SLB+16]

[†] Attack Method (AM): Active- ●, Passive- ○; Device Access (DA): Direct-●, Transitive-◑, Peripheral-○; Attack Complexity (AC): High-●, Low- ○; Required Privilege (RP): High privilege- ●, Low privilege- ○; User Interaction (UI): Needed - ●, not needed - ○; Success Rate (SR): High (>90%) - ●, medium (70-90%) - ◑, low (<70%) - ○.
[‡] Any type of keystroke inference impacts the confidentiality of the smart device.

Table 4.1: Summary of keystroke inference via sensors in smart devices.

features. These features are then cross-matched with a training dataset collected in a learning phase to infer the correct design. Song et al. improved this attack by adding magnetic sensor data to accurately reconstruct the physical prints and their G-code [SLB+16].

Keystroke Inference with Video Sensors - Modern smart devices come with powerful cameras which can both take still pictures and record high definition videos. By applying image processing techniques in captured images, keystroke inference can be done. Simon et al. developed a malware named *PIN skimmer* which uses the front camera of a smartphone and microphone to infer PIN input in a smartphone [SA13]. PIN skimmer records the tap sound on the touchpad of a smartphone and records video using the front camera of the phone. The movement recorded in the video is then analyzed to detect which part of the touchscreen is used. This information is then combined with the tap sound to infer the inputs correctly.

Another potential malware attack on the smart devices using the camera is *Juice Filming Attack* [MLMK15]. In this attack scenario, a malicious app uses the camera to take screenshots when any user-input is given in the touchpad and save the images on the storage unit (internal ROM or external memory card) of the device. Most of the smart devices use USB for heterogeneous applications (e.g., charging, data transfer, etc.) and when the compromised device is connected to the laptop or any other device with a storage unit, the app transfers the stored pictures to the storage device from which attackers can easily extract the information.

Shukla et al. showed a method to infer the PIN input by analyzing the hand position using the recorded video [SKSP14]. In this method, a background application gets access to the camera of the smartphone and records a video when a user starts typing in a touchpad. Then, analyzing the hand position and the position of the smartphone, an attacker can extract the inputs given in a touchpad. Another version of this attack is to record the typing scenario using an external camera. In this scenario, a camera of a smart device (e.g., smartphone, smart glass, smart surveillance system, etc.) is used to record the video of typing the PIN. In both cases, the input PIN can be inferred with high accuracy.

Adam J. Aviv introduced another type of attack named *Smudge Attack* using an external camera to infer pattern lock of a smart device [Avi12]. In this attack scenario, a smart device is placed in between two cameras of other smart devices (smartphone or smart glass) and high definition pictures are taken. Whenever the user gives the unlock pattern in the touchpad, some smudge marks are left on the screen, and captured by the cameras, which leak information about the unlock pattern to an attacker.

Raguram et al. developed a process named *iSpy* which can reconstruct the typed text by analyzing the reflection of the touchscreen in a reflective surface such as sunglass or smart glass [RWG+11]. The experimental setup of *iSpy* includes a high definition camera which can capture the video of the reflective surface while a user types in the touchpad of a phone. The reflection of the phone is being extracted from the video and consecutive frames are analyzed to extract stable pictures of the phone screen. Features (hand position, motion in the screen, etc.) are extracted from stable pictures extracted from the video and by using machine learning techniques, key press detection is done and typed text can be inferred successfully. In more recent work, Wang et al. proposed *GazeRevealer*, a novel side-channel attack to infer keystrokes in a smart device using the eye movement of the users [WCGS19]. *GazeRevealer* uses the front camera to capture video and analyzes to extract multiple features such as eye movement, head position, etc. These features are used to train a classifier which can predict the keystroke in real-time with high accuracy.

Keystroke Inference with Magnetic Sensors - Besides the aforementioned attack scenarios, electromagnetic emanations from the keyboard can be used to infer the input of a computer. As magnetic sensors of smart devices are sensitive to electromagnetic emanations, they can be used as the attack medium. Vuagnoux et al. showed that both wired and wireless keyboards emit electromagnetic signals when a user types and this signal can be further processed to infer keystroke [VP09]. In this method, electromagnetic radiation is measured by the magnetic sensor of a smart device when a key is pressed and using the *falling edge transition technique*, an attacker can infer the keystrokes.

Lessons learned for keystroke inference - We summarize the aforementioned threats and attacks in Table 4.1 with common vulnerability metrics. We can see smart devices with user input module (touchscreen, keypad, numeric keypad) are mostly the targeted device for keystroke inference. These threats and attacks affect the confidentiality of the sensor data. Another interesting fact we observe is the majority of the threats and attacks targets motion sensor (22 out of 42 reported threats and attacks) which does not require any permission to access in current smart device security schemes. Thus, these threats and attacks can easily access sensor data and extract keystroke information easily. For the targeted layer, we can notice the keystroke inference in smart devices only targets sensing (34 out of 42) and application (8 out of 42) layer. We can also observe a trade-off between attack complexity and required privilege in sensor-based threats targeting sensors in smart devices. For example, keystroke inference from the motion sensor (e.g., accelerometer) does not require any privilege to perform while keystroke inference from the audio sensor needs permission to access the microphone. However, accessing the motion sensor needs active vulnerability which may disrupt the on-going task in the smart device. On the other hand, capturing keystroke using the audio sensor can be both active and passive which increases the severity of the threat or attack. The outcomes of keystroke inference also have diverse effects on smart devices and users. As keystroke inference is directly related to user activities in smart devices, it impacts sensitive user information. Attackers can infer various typed information including device unlock code, password, banking information, typed and printed information, etc. These inferred information can be used to initiate another attack or directly used for malicious purposes such as ransom, data hijacking, identity theft, etc. In summary, passive keystroke inference with minimum required privilege (e.g., [MJHB15, SDN15, VP09]) can severely affect the confidentiality of the smart devices.

Task Inference

Task inference refers to a type of attack which reveals the information of an on-going task or an application in a smart device. Task inference reveals information about the state of the device and attackers can replicate this device state to launch an attack without alerting security policies implemented in the device. Sensors associated with smart devices show deviation in the reading for various tasks running on the devices. This deviation in the reading can be used to infer the running process inside a device and application of the device.

Task Inference with Light Sensor - Light sensor of a smart device can be used to infer an on-going task on a device. Smart devices with display emit lights with distinct intensity for different tasks. For example, playing separate videos in a smart TV will change the emitted light intensity based on the background and video quality. This change in light intensity can be used to infer an on-going task on the display. Chakraborty et al. showed that light intensity changed in a flat panel display (e.g., smart TV, smart monitor, etc.) can be used to infer what is written on the screen by a light sensor of a smartphone [COS17]. In this attack, an Android-powered smartphone is placed in front of the display to capture the light intensity emitted from the screen. These captured light signals can be sampled and deconvoluted to infer the task on the monitor such as on-going videos, specific web pages, etc. Berkay et al. used a smart light to passively leak the status of a smart home [CBS+18]. In this attack, if no user is present inside the home, a smart light will maliciously trigger an on-off pattern to notify the user. Maiti et al. proposed a new attack vector to infer the audio and video of a smart TV using the light emitted from a smart light [MJ18]. Here, researchers used the multimedia-visualization feature of smart light which creates a vibrant lighting effect in conjunction with audio and video playing nearby. Based on the light intensity emitted in audio frequencies, researchers successfully inferred an on-going audio or video.

Task Inference with Magnetic Sensors - Magnetic sensors in smart devices has the role to fix the orientation of the device with respect to Earth's magnetic field. Data recorded by a magnetic sensor change in the presence of an external magnetic field in the device's peripheral. This deviation in data can be used to identify the tasks running on a device. Many smart devices have a storage unit and whenever data is written or read from this storage unit, a change in the reading of the magnetic sensor can be observed. Magnetic sensors of a smart device can be used not only to infer information of the device itself, but can also be used as a medium to fetch information from a nearby device. Biedermann et al. showed that the magnetic sensor of a smartphone could be used to infer on-going tasks in a storage unit like the hard drives of the computers and servers [BKS15]. When an application is running on a computer, the hard drives generate a magnetic field which can be sensed by a magnetic sensor of a smartphone. Various actions cause distinct readings on the magnetic sensor which can be used to track the users' action. This can be considered as a serious threat to the device and attackers can fetch valuable information in this way. Ning et al. proposed *DeepMag+*, a side-channel attack to exploit on-board magnetic sensor for inferring smart apps installed in a smart device [NWX$^+$20]. *DeepMag+* captures the on-board magnetic sensor data while executing installed apps in a smart device and uses convolutional neural network to fingerprint the apps. Additionally, *DeepMag+* can combine motion sensor data with magnetic sensor to increase the inference accuracy up to 98%. Similar to this work, Matyunin et al. presented *MagneticSpy* , a novel website and application fingerprinting method exploiting magnetic sensors of a smart device [MWA$^+$19]. *MagneticSpy* analyzes the electromagnetic disturbances caused by the mobile processors which are proportional to the CPU workload. By

	Attack name	Target device	Target sensor	Target layer	Vulnerability metrics[†]						Ref.
					AM	DA	AC	RP	UI	SR	
Task Inference	LightSpy	Smartphone	Light	Application	○	○	○	○	○	◑	[COS17]
	IoTBench	Smart light	Light	Application	○	◑	○	●	●	-	[CBS+18]
	Hard drive side-channel attacks	Smartphone	Magnetic	Sensing	○	○	○	○	○	◑	[BKS15]
	Electro-magnetic Analysis of smart cards	Smart card	Magnetic	Sensing	○	○	○	○	○	-	[QS01]
	Power analysis attack	Smart devices	Any embedded sensors	Sensing	●	○	○	○	○	-	[OC15]
	Light Ear	Smart light	Light	Application	○	○	○	○	○	-	[MJ18]
	Peek-a-boo	Smart home devices	Motion, light, temperature	Communication	○	◑	●	○	○	●	[AFA+18]
	DeepMag+	smartphone	Magnetometer	Sensing	●	●	○	○	●	●	[NWX+20]
	MagAttack	smartphone	Magnetometer	Sensing	●	●	○	○	○	◑	[CJX+19]
	MagneticSpy	smartphone	Magnetometer	Sensing	●	●	○	○	●	●	[MWA+19]
Location inference	VoipLoc	Smartphone	Microphone, speaker	Sensing	●	●	●	●	○	◑	[Ano16]
	ACComplice	Smartphone	Accelerometer	Application	●	●	○	○	●	-	[HON+12]
	Inferring Your Secrets from Android	Smart Navigation device	Microphone, speaker	Application	●	●	●	●	○	◑	[HON+12]
	Permission less Location Attack	Smartphone	Magnetic	Application	●	●	○	○	○	◑	[BN18]
	Inferring User Routes and Locations	Smartphone	Accelerometer, gyroscope, magnetic	Application	●	●	●	○	○	○	[NVHBN16]
	MISSILE	Smartphone	Accelerometer, gyroscope	Sensing	●	●	●	○	●	◑	[ZH19]
	Prying into Private Spaces	Smartphone	Accelerometer, gyroscope	Sensing	●	●	●	○	●	◑	[FGSS19]
Eavesdropping	Soundcomber	Smartphone	Microphone	Application	●	●	○	●	●	◑	[SZZ+11]
	VoicEmployer	Smartphone	Microphone, Speaker	Application	●	●	●	●	●	●	[DLZZ14]
	CPVT	Smartphone	Microphone, Speaker	Application	●	●	●	●	●	-	[LWZ+13]
	Hidden voice commands	Smart car	Microphone	Application	○	○	○	○	●	●	[CMV+16]
	Gyrophone	Smartphone	Gyroscope	Sensing	●	●	○	○	○	◑	[MBN14]
	Spearphone	smartphone	Accelerometer	Sensing	○	○	○	○	●	●	[AWL+19]
	I Can Hear Your Alexa	smart speaker	Microphone	Communication	○	○	○	○	●	○	[KLW+19]

[†] Attack Method (AM): Active- ●, Passive- ○; Device Access (DA): Direct-●, Transitive-◑, Peripheral-○; Attack Complexity (AC): High-●, Low- ○; Required Privilege (RP): High privilege- ●, Low privilege- ○; User Interaction (UI): Needed - ●, not needed - ○; Success Rate (SR): High (>90%) - ●, medium (70-90%) - ◑, low (<70%) - ○.
[‡] Any type of information leakage impacts the confidentiality of the smart device.

Table 4.2: Summary of task inference, location inference, and eavesdropping via sensors in smart devices.

analyzing the deviance in different working conditions, *MagneticSpy* can infer the on-going CPU activity with high accuracy (up to 90%).

An electromagnetic (EM) emanation is a common phenomenon for smart devices. Electromagnetic emanations occur whenever current passes through a device and a task is running on a device. EM emanation attacks can also be observed in FPGA-based (Field-programmable gate array) smart devices [QS01,CCDP04,AARR03]. Attackers can record electromagnetic emission data generated from the FPGA-based smart devices to deduct which kind of application is running in the system and also the states of logic blocks of the devices. Such information leakages make the system vulnerable to the user. Smart cards also emit EM waves while performing various tasks which can be captured by a radio frequency (RF) antenna and the task can be inferred from the radiation [RW]. Cheng et al. proposed *MagAttack*, a side-channel attack to abuse the magnetic sensor of smart mobile devices [CJX$^+$19]. User activities such as application launching and operation has a slight but significant effect on CPU's power consumption, and hence in the EM emissions. An attacker can capture this EM emission using the magnetic sensors of a smart device and infer the on-going user activities in a laptop or workstation.

Task Inference with Power Analysis - Power analysis is a form of sensor-based threat where an attacker studies the power consumption and power traces of the sensors for extracting information from the devices [ÖOP03]. O'Flynn et al. introduced an attack scenario where the power analysis attack is launched against IEEE 802.15.4 nodes [OC15], which is a standard low power wireless protocol used in smart devices. Low power smart devices use this protocol standard for various communication purposes such as connecting to a network, communicating with other devices, etc.. In this attack scenario, an attacker uses differential power analysis in the sensors. As packets transmitted from the smart devices are encrypted, power analysis on the sensors can infer which encryption process is running in the device. Again, diverse encryption process leads to diverse power profiles which reveal associated information (e.g., key size, block size, etc.) about the encryption process. Encryption process also depends on the packet size which can be observed in the power profile and attackers can infer what type of information is being transmitted based on the packet size.

Task Inference from sniffing sensor data - In a connected environment such as smart home, several smart devices are connected with each other and with multiple sensors. These sensors communicate with the devices using various protocols (e.g., WiFi, ZigBee, Z-Wave, etc.) and work as triggering devices for several automated tasks. An attacker can sniff the communication traffics in the smart environment and infer user and device actions which can be considered as a privacy violation. Acar et al. showed that it is possible to infer user activities and devices states by capturing the communication packets and extracting sensor data in a smart home environment [AFA+18]. In this attack scenario, an adversary in close proximity of the smart environment can sniff the communication packets and infer the states of the devices (active/inactive). In addition, authors showed that the attacker could deduce the actions of the users (e.g., walking, opening doors, etc.) using machine learning techniques in captured traffics.

Lessons learned for task inference - Similar to keystroke inference, task inference in smart devices also affect the confidentiality of the devices. From Table 4.2, we can observe the majority of the task inference threats (6 out of 10 reported threats and attacks) are passive which indicates the high impact on the smart devices. Another interesting fact is the majority of these threats does not need any additional privilege (9 out of 10) to bypass existing security schemes. Also, task inference threats target sensing (6 reported threats), application (3 reported threats), and communication (1 reported threat) which indicates a broad attack surface of these threats. One limitation of reported task inference attacks is the lack of extensive evaluation of the attacks. To understand the effectiveness of a sensor-based attack, it is necessary to check the success rate of the attack on real-life smart devices. The majority of the task inference attacks are not appropriately evaluated with known evaluation metrics such as success rate, error rate, precision, etc.. Without proper evaluation metrics, especially without reported success rate, it is hard to understand the effectiveness and feasibility of task inference attacks on a smart device. Task inference directly impacts the confidentiality and privacy of the smart device users by leaking sensitive information such as user activity, installed security measures, installed apps on smart devices, etc. Attackers can profile a user based on task inference attacks to perform diverse types of malicious activities such as gaining access to the smart device and environment, bypassing security measures to leak data, manipulate or obstruct on-going tasks, etc. [AFA+18].

Location Inference

Researchers developed a novel location-privacy attack based on acoustic side-channels [Ano16]. The attack is based on acoustic information embedded within foreground-audio disseminated in a closed environment (i.e., conference room). The researchers studied how audio, generated by secure messaging clients in voice-call mode, can be abused to generate a location fingerprint. The attack leverages the pattern of acoustic reflections of the human voice at the user's location and does not depend on any characteristic background sounds. The attack can be used to compromise location privacy of participants of an anonymous VoIP session, or even to carry out confirmation attacks that verify if a pair of audio recordings originated from the same location regardless of the speakers. Other researchers have also shown that several heuristics can be used to identify sensitive locations (i.e., home and work locations) of a victim whose personal device is under an adversary control [PMSJ16]. Han et al. showed that it is possible to infer the location of a user using the accelerometer of a smartphone [HON$^+$12]. Here, researchers first derived an approximate motion trajectory from accelerometer reading and correlated the trajectory with the map to infer the exact location of the user. Zhou et al. showed that it is possible to infer the location of the user by analyzing verbal directions provided by navigation apps of a smart device [ZDH$^+$]. Researchers measured the on/off times of the speaker controlled by the navigation app to leak the driving instructions to the attacker. In a more recent work, Block et al. introduced a new location inference technique using the smartphone's magnetometer [BN18]. Here, researchers used small fluctuations originated by nearby magnetic fields while the smartphone is in motion to build a trajectory path of the user. Narain et al. proposed a combination of sensor data (accelerometer, gyroscope, and magnetometer) to further improve the accuracy of the inferred location [NVHBN16]. In a recent work, Zheng et al. proposed a location eavesdropping attack using the mobile inertia/motion sensors [ZH19]. Here, researchers showed that in the presence of specific indoor structures (e.g., elevators, fire stop doors, etc.), motion sensors display specific patterns which can be utilized to infer the location correctly. Similar to this work, Fyke et al. used the motion sensors data to recreate user's movement and plot maps and landmarks in private spaces (e.g., home, workplace, etc.) [FGSS19].

Lessons learned for location inference - Although location inference attacks impact the confidentiality of smart devices, all of the threats (7 reported threats and attacks) are active which limits the consequences (Table 4.2). Also, to execute malicious sensor activities, these threats need direct access to the devices which affect the easy deployability of these threats in real-life smart devices. One can also observe from Table 4.2 that the success rate of these attacks is low to medium range. Compared to keystroke and task inference attacks, location inference poses less effects on the security of the smart devices. However, leaking location information can violate user's privacy and propagate other attacks including a targeted physical attack on the user's vehicle [ZDH+].

Eavesdropping

Many smart devices such as voice-enabled speakers use audio sensors for making calls, recording audio messages, receiving voice commands, etc. Eavesdropping refers to a type of attack where a malicious app records a conversation stealthily by exploiting audio sensors and extract information from the conversation. An attacker can save the recorded conversation on a device or listen to the conversation in real-time. One of the recent examples of eavesdropping via the microphone of a smartphone is *Soundcomber* [SZZ+11]. In this example, a malicious app covertly records when a conversation is initiated from the device. As the recording is done in the background, a user does not have any idea about the recording. Several companies like banks, social security offices, credit card companies, etc. have automated voice messaging systems and users have to say their private information such as credit card numbers or social security numbers at the beginning of the call. Thus, *Soundcomber* does not have to record all the conversations to extract data. Only the beginning part of the conversations will be enough for extracting private information of the user. Moreover, a specific conversation can also be recorded by identifying the dialed number on a smartphone. The touchpad of the smartphone creates corresponding tones when any number is dialed. This tone can be recorded and processed to identify the dialed number. After that, when the desired number is dialed, the conversation can be recorded and then processed to extract information.

Another way to exploit microphones is to attack through voice assistant apps, e.g., Apple's Siri and Google Voice Search. Most of the smart devices nowadays have built-in voice search apps. Diao et al. developed a malware named *VoicEmployer* which can be installed on the device to record the voice command given in a smartphone [DLZZ14]. This malware can use the recorded command for various malicious activities such as replicate malicious voice command, transfer information to paired devices, etc. *Cyber Physical Voice privacy Theft Trojan horse (CPVT)* is another malware which uses the microphone of smartphones to record conversations [LWZ+13]. The recording of the conversation can be controlled by external control channels like SMS, Wi-Fi, or Sensory channels [USB14]. An attacker can trigger *CPVT* and create command about when to start recording and when to stop recording using SMS, Wi-FI, or even sensors. Recorded conversations are stored in the device and the attacker can gain the stored files using Email, SMS, or connecting via USB. Carlini et al. showed that it is possible to exploit voice assistant apps by inserting hidden voice commands [CMV+16]. In this attack, the attacker first records voice commands of the user and extracts features from the recorded audio clips. From the extracted features, a new command is generated which is not understandable by humans, but recognized by the voice assistant apps. In a recent work, Kennedy et al. showed that it is possible to infer the voice command given to a voice assistant device (e.g., Amazon Alexa) by capturing the network packet and using natural language processing [KLW+19].

The gyroscope on smart devices is also sensitive to an acoustic signal. The typical sampling rate of gyroscope covers some frequency of audible range which can be used to reconstruct the speech of a user. Michalevsky et al. proposed a new way of eavesdropping by analyzing vibrational noise in gyroscope caused by an acoustic signal [MBN14]. As the gyroscope does not cover the full audible range, this new process can distinguish speakers and one-syllable words by using signal processing and machine learning techniques. In a recent work, Anand et al. showed that the on-board accelerometer could be used to eavesdrop and reconstruct the speech of a user [AWL+19]. While a user talks on a smartphone, the loudspeaker of a smartphone shows some reverberations which impact the accelerometer reading. This deviation in accelerometer can be further analyzed to extract sensitive information such as speaker identification and gender classification.

Lessons learned for eavesdropping - Eavesdropping mostly affects smart devices with audio sensors and impacts the confidentiality of the devices. From Table 4.2, it is visible that the majority of the eavesdropping are active attacks (4 out of 5 reported threats and attacks) and require additional privileges (4 out of 5 threats and attacks) to bypass the existing security schemes. These threats also need users to interact with the system to perform malicious tasks that limit the impact of these threats. For performing eavesdropping, the majority of the threats and attacks also need direct access (4 reported threats) on a targeted smart device. Because of these dependencies, the impact of eavesdropping is lower than other types of information leakage attacks. Nevertheless, the information captured in the eavesdropping attack can be used to perform various malicious activities such as leaking private conversation, gaining physical access to a secured environment, etc. [ZQL+19].

4.3.2 Transmitting Malicious Sensor Commands

Sensors available in the smart devices can be used to transmit malicious sensor patterns or triggering commands to activate malware that may have been implanted in a victim's device [USB14]. Sensors may be employed to create unexpected communication channels between device peripherals. Such channels can be used to exchange

	Attack name	Target device	Target sensor	Target layer	Vulnerability metrics[†]							Ref.
					AI	AM	DA	AC	RP	UI	SR	
Transmitting malicious sensor commands	Out-of-band command	Smartphone	Light	Sensing	I	●	○	○	○	○	●	[HSH+13]
	Creating seizures using strobed light	Smart light	Light	Application	I	○	◑	○	●	●	●	[FJP16]
	IoTBench- Side channel attack	Smart light	Light	Application	I	○	◑	○	●	●	●	[CBS+18]
	Out-of-band command via magnetic sensor	Smartphone	Magnetic	Sensing	I	●	○	○	○	○	●	[HSH+13]
	Out-of-band command via audio sensor	Smartphone	Microphone	Sensing	I	●	○	○	●	○	●	[HSH+13]
	Inaudible sound as a covert channel	Smartphone	Microphone	Sensing	I	●	○	○	●	○	●	[Des14]
	Sensor side channels	Smartphone	Microphone	Sensing	I	●	○	○	●	○	-	[SUCB13]
	Skill squatting attack	Smart Speaker	Microphone	Sensing	C, I	●	○	●	●	●	●	[KPM+18]
	DolphinAttack	Smart Speaker	Microphone, Speaker	Application	C, I	●	○	●	●	●	-	[ZYJ+17]
	Injecting inaudible voice commands	Smart voice assistant	Microphone, Speaker	Application	C, I	●	○	●	●	●	-	[YZJ+19]
	Vaspy	smartphone	Microphone	Application	C, I	●	●	●	●	●	●	[ZCWZ19]
False sensor data injection	GPS spoofing attack	Smart navigation device	GPS	Application	I	●	●	○	●	○	-	[TPRC11]
	GPS jamming	Smart navigation device	GPS	Application	I	●	●	○	●	○	-	[Cof14]
	Spy-sense	Smart sensor network	-	Data processing	I	●	◑	○	○	○	-	[GD13]
	Injected and Delivered	Smart cars, drone	Accelerometer, gyroscope	Sensing	I	○	○	○	○	○	◑	[TLLH18]
	This ain't your dose	Smart medical device	Light	Sensing	I	○	○	○	○	○	●	[PSS+16]
	Illusion and dazzle	Smart car	Light	Application	I	○	○	○	○	○	-	[SKKK17]
	Remote attacks on automated vehicles	Smart car	Light, Camera	Application	I	○	○	●	●	○	●	[PSFK15]
	REEVE	Smart voice assistant	Microphone	Application	I	○	○	○	○	●	-	[YCW+18]
	Using AI to Hack IA	Smart voice assistant	Microphone, Speaker	Application	I	●	●	●	●	●	-	[ZCL+18]
	Light Commands	Smart voice assistant	Light	Application	I	○	○	○	○	○	-	[SCR+19]
Denial-of-Service	Rocking drones	Smart drone	Gyroscope	Communication	A	●	○	○	○	●	●	[SSK+15]
	Pairjam	Smart home device	Microphone	Communication	A	●	◑	○	●	○	◑	[MZL20]

[†] Attack Impact (AI): Confidentiality (C), Integrity (I), Availability (A); Attack Method (AM): Active- ●, Passive- ○; Device Access (DA): Direct-●, Transitive-◑, Peripheral-○; Attack Complexity (AC): High-●, Low- ○; Required Privilege (RP): High privilege- ●, Low privilege- ○; User Interaction (UI): Needed - ●, not needed - ○; Success Rate (SR): High (>90%) - ●, medium (70-90%) - ◑, low (<70%) - ○.

Table 4.3: Summary of other sensor-based attacks in smart devices.

critical sensor parameters (e.g., devices' motion, light intensity, magnetic field, etc.) or to transmit malicious commands (Section 2.4: use case 3).

Transmitting via Light Sensors - Light sensors can be used as a potential method of transmitting signals and malicious commands [JPPS11]. It is easier to transfer a bit stream via a light source by turning it on and off. Since the light sensor of a smart device can distinguish the intensity of the light source, the light intensity change can be decoded as a bit stream in the device. By controlling the voltage of a light source, an attacker can easily transfer trigger messages and can activate malware implanted in a device. Hasan et al. showed that TV screen or laptop monitor could also be used to transfer trigger messages to a compromised smart device by changing the light intensity of the monitor [HSH+13]. Fernandes et al. showed that a smart light could be maliciously programmed to strobe the light at a high rate and if the user has the health problem of seizure, this action will trigger the user's seizures which is really dangerous [FJP16]. Berkay et al. showed that a smart light could be programmed to operate in a specific pattern to trigger a smart camera and take pictures surreptitiously [CBS+18].

Transmitting via Magnetic Sensors - As mentioned earlier, magnetic sensors of a smart device are sensitive to the magnetic fields of the device's peripherals. By changing the magnetic field of the device ambiance, one can easily change the readings of the magnetic sensor which can be used as a triggering message of malware. Triggering messages encoded by an electromagnet can be sent to a smart device and there will be some deviations in the magnetic sensor's readings of the device due to this message. These deviations can be calculated and the triggering message can be extracted from this electro-magnetic signal. Moreover, the magnetic field deviations can be calculated in x, y, and z-axis and divergent values of the magnetic field deviations can be interpreted as disparate triggering messages [HSH+13].

Transmitting via Audio Sensors - Audio sensors can be used to transmit malicious commands to activate a malicious application in a smart device. Hasan et al. showed that a triggering message embedded in an audio song can be detected by the microphone and can trigger a malicious app in a smartphone [HSH$^+$13]. Moreover, microphones used in modern smart devices can detect audio signals with a frequency lower than the audible range. Malware can be transferred using this audio channel as a covert channel to bypass the security measures of the device. Deshotels et al. showed that the ultrasonic sound could be used to send information to smartphones without alerting the user or any security measurement implemented on the device [Des14]. Subramanian et al. showed that a trojan can be transferred by encoding it in an audio signal and transferring it using a buzzer [SUCB13]. In a recent work, Zhang et al. showed that it is possible to transmit an inaudible acoustic signal to smart speakers to trigger malicious activities [ZYJ$^+$17]. Yan et al. performed a feasibility study of previous work and concluded that it is possible to trigger several malicious events in smart devices including making a phone call, changing state of connected devices, etc. [YZJ$^+$19]. Kumar et al. showed that valid voice commands could be used to trick smart speakers (e.g., Amazon Alexa, Amazon Echo, etc.) to perform malicious activities in skill squatting attack [KPM$^+$18]. Here, researchers used misinterpretations of valid commands made by the smart speaker to trigger a malicious activity. For example, 'test your luck' can be misinterpreted by the smart speaker as 'test your lock' which can unlock the door. In a recent work, Zhang et al. proposed *Vaspy*, a malicious app installed in the smart device to exploit voice assistant devices [ZCWZ19]. *Vaspy* silently observes smartphone activities and captures the phone call conversations to extract the voice activation commands. Upon extracting the voice command, *Vaspy* uses a machine learning model to analyze user behavior and choose a specific time to launch an attack surreptitiously.

Lessons learned for transmitting malicious sensor commands - The threat of transmitting malicious sensor commands mostly affects the integrity of smart devices (Table 4.3). The majority of the threats and attacks (9 out of 11 reported) needs an additional privilege to bypass the existing security schemes. Also, upon successful attack, malicious sensor commands trigger malicious activities in the smart devices which obstruct normal operations. Thus, transmitting malicious sensor commands (all reported threats and attacks) act as active attacks to the smart devices. Another interesting fact we observe transmitting malicious sensor commands do not need any direct connection to the device. Only transitive (2 out of 11 threats) or peripheral access (9 out of 11) is enough to transmit malicious commands to the targeted smart devices. However, the success rate of most of the threats and attacks is high, indicating the high impact on smart devices. This trade-off between excessive privilege and success rate determines the effects of the threats and attacks.

4.3.3 False Sensor Data Injection

The applications of smart devices largely depend on data collected by sensors available on the devices. By altering the sensor data, one can control the applications of smart devices. False sensor data injection refers to an attack where the sensor data used in the smart applications is forged or intentionally changed to perform malicious activities. The false sensor data can be injected in the devices by accessing the device physically or by using various communication mediums (Bluetooth, ZigBee, Z-Wave, Wi-Fi, cellular network, etc.) covertly. An attacker can also introduce fake sensors in the IoT environment to inject false generated data and initiate malicious activities (Section 2.4: use cases - 1 and 3) [CBS+18,HAUA18]. Moreover, the sensors of smart devices can also be used to alter data typed or stored on the devices.

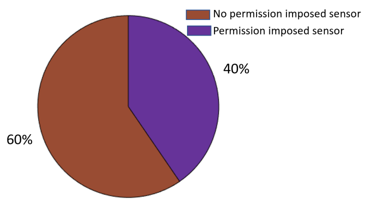
Tu et al. presented a *spoof attack*, where an out-of-band signal is inserted in smart devices via motion sensor [TLLH18]. This signal injection results in deviation in sensor output which disrupts the normal functionality of the smart devices. Park et al. used infrared light to disrupt normal operation of a smart medical device [PSS+16]. Here, researchers used a medical infusion pump to inject the spoof light signal and change the dose of the medicine in the device. In another recent work, Shin et al. exploited the light sensor of a smart car to change the output of the automatic obstruction detection system [SKKK17]. Petit et al. improved this attack by combining camera reading of a smart car to change the output of autonomous vehicle [PSFK15]. In a recent work, Zhou et al. proposed an attack to exploit the voice assistant of a smart car [ZQL+19]. In this attack, the adversary inserts malicious commands in an audio or video file which can inject malicious commands to the voice assistant apps upon playing.

The smart voice assistant is deployed in several smart devices such as smartphone, smart speaker, smart home hub, etc. These smart assistants usually triggered with a specific command such as "Hi Google", "Hey Siri", or "Alexa". Recent researches showed that it is possible to inject malicious commands to smart voice assistants by exploiting the microphone of the smart devices. As smart voice assistants constantly scan for desired a triggering command, an adversary needs no additional privilege to inject malicious audio signals to the device. Yuan et al. proposed *REEVE*, a stealthy voice manipulation attack to smart voice assistant [YCW+18]. *REEVE* uses benign audio signals such as TV or radio as a medium and insert malicious trigger commands which can be detected by a nearby voice assistant device. The researchers tested this attack on consumer voice assistant devices (Amazon Echo) and achieved high success rate. Zhang et al. improved this attack by eliminating the need of external audio signals [ZCL+18]. Here, researchers developed a spyware which can abuse the microphone of a smartphone to record phone conversations and detect the trigger messages. Later, the spyware replays the recorded command using the speaker of the same smartphone to inject false commands to the voice assistant service.

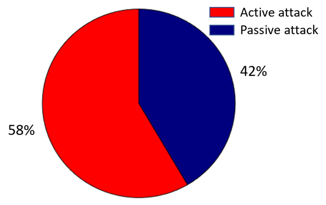
Tippenhauer et al. showed another attack scenario in GPS-enabled devices to change the real location of the device [TPRC11]. In this attack scenario, a vehicle with a GPS enabled device is used. The attacker transmits a forged GPS signal to the device to alter the location of the vehicle. In this way, the real location of the vehicle is disguised and the attacker can perform any physical attack on the disguised vehicle. The GPS data used in the smartwatches can expose the location of a user and this GPS data can then be forged and a new location can be given as a false input in the GPS [Cof14].

The power analysis attack on smart devices can also be used for injecting false data. The power analysis on smart devices running an encryption algorithm can reveal information about the encryption process including the block size, key size, even the actual encryption key [YN17]. This information can be used to encrypt a false data and replace the original data on the device. Thus, attackers can inject false encrypted data in the communication channel to change the action of a device for specific commands. Giannetsos et al. introduced a malicious app named *Spy-sense*, which monitors the behavior of the sensors in a device and can manipulate data by deleting or modifying it [GD13]. *Spy-sense* exploits the active memory region of a device and alters the data structure and reports back important data to a server covertly.

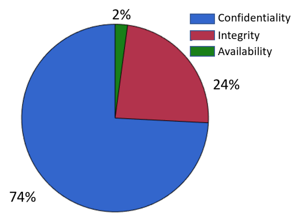
Lessons learned for false data injection - False data injection impacts the integrity of the smart devices as these threats and attacks disrupt the output of an on-going task. From Table 4.3, it is evident that the majority of the threats and attacks are simple and do not need any user interaction (8 out of 11 reported threats



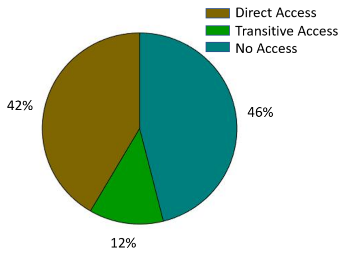
(a) Types of sensors



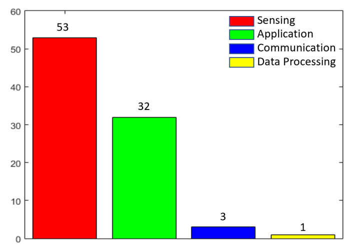
(b) Active vs. passive



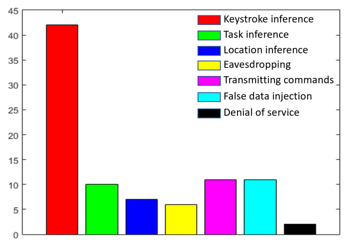
(c) Targeted security mechanisms



(d) Device access



(e) Targeted layers



(f) Types of threats

Figure 4.1: Overview of sensor-based threats and attacks to smart devices.

and attacks) to perform malicious tasks. Also, false data injection attacks are passive by nature (7 out of 11 threats) and do not need any excessive privilege (6 out of 11 threats) to perform the attack. Another interesting fact we observe is the effect of the successful attack directly impact the on-going activities of the smart devices. Hence, false data injection attacks are method-wise passive, but effect-wise active. However, the majority of the existing false data injection attacks did not report any success rate. Without proper evaluation, it is hard to understand the effectiveness of the reported attacks in real-life smart devices. Hence, further investigation is needed to properly evaluate the effectiveness of these attacks on real-life smart devices. The effects of false data injection are diverse as it can manipulate the targeted smart device to perform numerous malicious activities. For instance, false data injected in smart voice assistant can give the attacker access to any connected device in a smart environment which can cause device theft, undesired physical access to properties, unauthorized bank transactions and online shopping, etc. [ZCL+18, YCW+18].

4.3.4 Denial-of-Service

Denial-of-Service (DoS), by definition, is a type of attack where the normal operation of a device or application is denied maliciously. DoS attacks can be active attacks where an application or task is refused forcefully or passive attacks where attacking one application can stop another on-going task on the device. An adversary with access to smart device network and peripheral can send unauthorized access request or malicious signals to interrupt an ongoing task in smart devices (Section 2.4: use case 4). Indeed, recently, ICS-CERT published an active alert for a list of accelerometers used in smart devices which can be exploited using vibrational force [CA17]. Every accelerometer has a working frequency and if an external vibrational force can match this frequency, it is possible to turn off the devices forcefully. This reported threat is applicable for 20 different types of MEMS accelerometer which are used in multiple commercial and consumer smart devices. Hence, the impact of this threat is severe in real-life smart devices. Son et al. showed that it is possible to obstruct the flight control of a drone by exploiting gyroscope using a sound signal [SSK+15]. The MEMS Gyroscopes deployed in drones have a sensing mass inside of the sensor which is constantly vibrating. The gyroscope measures the rotational motion of the device with respect to the sensing mass. When the resonant frequency of the gyroscope is matched by an audio signal, an attacker can obstruct the normal performance of the gyroscope and change the course of the drone, or even turn it off. In a recent work, Mao et al. presented *Pairjam*, a DoS attack that uses inaudible noises to disrupt pairing between smart devices [MZL20]. In a smart environment, multiple smart devices are connected with each other to perform various tasks. The interconnection between the devices follows a device authentication/pairing method to ensure secure communication. *Pairjam* abuses the audio sensor of smart devices the inject inaudible noise signal in the smart environment which disrupts the normal pairing method and makes a targeted smart device unavailable for pairing.

Lessons learned for DoS - Denial-of-Service impacts the availability of the targeted sensor and disrupts an on-going task immediately in a smart device. There are only two reported DoS attack on smart devices which are passive and achieves high success rate (Table 4.3). However, one of the reported DoS attack [SSK+15] uses the MEMS accelerometer and gyroscope which is used by a vast number of smart devices [CA17]. This increases the impact of DoS attacks on smart devices. Moreover, both of these DoS attacks are applicable to both standalone and connected smart devices which widens the attack surface. Hence, DoS attacks have a high impact on real-life smart devices.

4.3.5 Summary of the Threats and Attacks

We categorized 89 reported sensor-based threats and attacks by the research community and industry in four categories. Additionally, we explained the attack methods and discuss the impacts of the sensor-based attacks based on common vulnerability metrics (attack impact, attack method, attack complexity, required privilege, user interaction, success rate). Some interesting findings of the aforementioned sensor-based threats and attacks are listed below:

- *Type of sensors targeted:* Existing threats and attacks target nine different sensors including both permission and no-permission imposed sensors discussed in Section 2.4. One interesting fact we observe that the majority of the threats and attacks target no-permission imposed sensors which make the existing permission-based sensor management system ineffective. From Figure 4.1(a), one can notice that 60% of the total threats and attacks target no-permission imposed sensors whereas only 40% of the reported threats and attacks target permission-imposed sensor which needs to bypass existing sensor management systems.

- *Active vs. passive:* As previously mentioned, sensor-based threats and attacks can be active or passive depending on the attack method. From Figure 4.1(b), one can notice the high percentage of active sensor-based threats and attacks on smart devices. While the majority of these are active, passive attacks and threats are also a point of interest to the attackers. As passive threats or attacks do not affect the normal functionalities of the devices, these may remain unnoticed to the implemented security mechanisms and perform malicious activities surreptitiously.
- *Targeted security mechanisms:* Sensor-based threats and attacks target various security mechanisms of the devices (e.g., confidentiality, integrity, availability) which make them hard to detect. Figure 4.1(c) shows different security mechanisms targeted by the sensor-based threats and attacks. One can notice that most of these threats and attacks aim to violate data confidentiality (74%) followed by integrity (24%) and availability (2%) of the sensors.
- *Device access needs:* To perform malicious sensor activities, sensor-based threats and attacks need device access (direct or transitive). Figure 4.1(d) illustrates the device access needs of sensor-based threats. While approximately 54% of the threats need direct or transitive access (42% direct and 12% transitive), 47% of the threats do not need any access to execute the malicious activity. As a sensor-based threat without any need of device access can easily bypass any security mechanism, the impact of the threats is high. Additionally, the exclusiveness of device access needs

of sensor-based threats manifests the shortcomings of the existing permission-based sensor management system.

- *Affected layers:* Smart devices typically have four architectural layers and attackers can target any of these layers to initiate a sensor-based attack. From Figure 4.1(e), it is evident that the most affected layer is the sensing layer followed by the application layer. As modern smart devices offer diverse sets of apps that use sensors for enhanced functionalities, attackers target these layers to modify and perform malicious activities in smart devices. The apps in smart devices directly bind the sensing and application layer which is the main reason for increasing threats to these layers.

- *Interest of the attackers:* From the discussion above, we can infer the most common sensor-based threats and attacks to smart devices is keystroke inferences followed by task inference and transmitting malicious sensor commands. As keystroke inference attacks typically target smart devices with user interfaces, we can observe higher number of sensor-based attacks in smartphones and smart watches. Figure 4.1(f) shows the common sensor-based threats and attacks to the smart devices.

- *Effect of the sensor-based threats and attacks:* In Table 7.4, we summarize the effects of sensor-based threats and attacks on smart devices. One can notice that keystroke inference attacks can leak diverse typing information such as passwords, PIN input, hand gestures, printed texts, etc. by exploiting a smart device directly or using a smart device to extract information from a nearby device. Task inference attacks reveal the nature of on-going tasks on smart devices either in the user interface of the device or in a connected smart environment. Sensor-based threats can also infer the geo-location of a smart device user as well as create a location map of users' route. By performing an eavesdropping attack using sensors, an adversary can capture users' conversations and smart device commands to extract

Attack type	Effect	Reference
Keystroke Inference	Information of lock code or PIN of a smart device.	[Spr14, CC12, AHIN13, SPYG15, OHD+12, NSN14, CC11, XBZ12, BFG+19, Ngu15, LS19, LCY+18, LYC+19, KJL20, ZWY+19, SSYA19, SA13, SKSP14, RWG+11]
	Typing information (typed text, notes, commands, etc.) in an embedded or virtual keyboard.	[HB18]
	Drawing patterns or unlock patterns on a smart device.	[ASBS12, MLMK15, Avi12]
	Typing information of a nearby keyboard or device.	[MVCT11, LLC+19, HGC+19, LZD+15, WLRC15, MJHB15, MJHB18, SDN15, MHSJ18, LL19, AA04, ZZT09, HS12, BWY06, FKK10, ZMZL14, WCGS19]
	Printed information in a printer.	[BDG+10]
Task Inference	on-going task reflected on user interface of a smart device.	[COS17, MJ18]
	On-going task by a user in a single smart device or connected smart environment.	[CBS+18, AFA+18]
	Information of an on-going task in a near by device.	[BKS15, QS01, OC15, NWX+20, CJX+19, MWA+19]
Location Inference	Geo-location of a user interacting with a smart device.	[Ano16, HON+12, BN18]
	Mapping smart device users' motion and routes to detect location.	[NVHBN16, ZH19, FGSS19]
Eavesdropping	Capturing conversation of the users using on-board sensors surreptitiously.	[SZZ+11, MBN14, AWL+19]
	Stealthily record a conversation or command and replaying it to get access to a smart device and perform malicious activities.	[DLZZ14, LWZ+13]
Transmitting malicious commands	Transmitting malicious sensor command to a nearby smart device and execute malicious activities.	[HSH+13, Des14, SUCB13, KPM+18, ZYJ+17, YZJ+19, ZCWZ19]
	Transmitting malicious commands using a smart device in a connected smart environment.	[FJP16, CBS+18]
False Sensor Data Injection	Injecting false sensor data to change the output of a smart device or a specific application running in a smart device application.	[TPRC11, Cof14, GD13, TLLH18, PSS+16, SKKK17, YCW+18, PSFK15, SCR+19]
Denial-of-Service	Using sensory channel to disrupt an on-going task in a smart device.	[SSK+15]
	Injecting inaccurate sensor signal to make a device unavailable in a smart environment.	[MZL20]

Table 4.4: Effect of sensor-based attacks on smart devices.

information and accessing a targeted device. An adversary transmitting malicious sensor commands can trigger malicious events on a smart device which can be propagated to nearby smart devices. Additionally, introducing false sensor data in a smart device can change the output of a smart device and make a device or sensor unavailable for performing a task (DoS).

4.4 Open Issues, Future Directions, and Recommendations

The concept of making devices 'smart' is no longer in the developing stage and new research ideas related to smart devices are emerging these days. In this section, we discuss open issues and future research directions in the context of sensor-based threats and attacks to smart devices.

4.4.1 Open Issues and Future Directions

Due to the lack of knowledge among users and research communities, sensor-based threats become compelling to the attackers to exploit the security of smart devices and perform malicious activities. There are several open issues that exist in smart devices that need attention from developers, researchers, and users. These open issues can be categorized in three major areas - (1) Smart device architectures and platforms, (2) Further investigation of existing threats, and (3) Solutions to detect sensor-based threats. In the following discussion, we briefly explain these open issues and summarize future research directions needed to counter sensor-based threats.

Smart device architectures and platforms. The smart device industry is growing rapidly and these smart devices are different from each other in terms of hardware, software, implementation, and functionalities. To understand the sensor-based threats, it is important to understand the smart device architecture and functionalities properly. Researchers and developers can investigate the following open issues in smart device architecture to understand the consequences of sensor-based threats properly.

Study of Smart Device Architectures and Sensor Operations - With the introduction of IoT, the number of smart devices in different domains is increasing rapidly. The smart devices have several internal architectures (i.e., software and hardware) with less knowledge available, which is an obstacle to secure sensors in these devices. For instance, there are several operating systems (e.g., Linux, Android, Contiki, TinyOS, etc.) available for smart devices which vary in terms of functionalities, operations, and integrated security features. Moreover, smart devices can connect with each other and create a network of smart devices to perform various tasks. The lack of knowledge of device architectures can affect the security of the devices as security flaws in one smart device can cause the compromise of other connected smart devices. Additionally, in a smart connected environment, multiple smart devices use one sensor to automate various tasks [SBAU19]. Hence, compromising one sensor can trigger malicious activities in several connected smart devices. Researchers and developers should study the smart device architectures (both standalone and connected smart devices) and functionalities to understand the sensor mechanism which will help to understand the consequences of emerging sensor-based threats.

Adoption of Standard Security Mechanisms - Currently, there exist several operating systems for smart devices that manage their on-board and external connected sensors in distinctive ways (Section 2.4). These dissimilarities make it hard to converge for a general security scheme to protect sensors of the smart devices [Liv16]. For example, in a smart environment, several smart devices from different vendors can share the same sensors and physical environment. Any sensor-based threats compromising normal functionalities of a sensor can propagate to several connected smart devices. In this scenario, installing vendor-specific sensor security schemes surely increase the security of smart devices from a specific vendor. However, sensor-based threats targeting smart devices from another vendor can compromise connected smart devices even with an installed vendor-specific security scheme [SBAU19]. Moreover, installing different security schemes in different smart devices can lead to high resource usage and introduce overhead in the smart environment. Hence, a comprehensive vendor-independent sensor security scheme is needed to secure sensors of smart devices in a connected smart environment. One of the future research efforts should be the standardization of development platforms for smart devices which will make it easier for researchers to come up with universal security measures to defend against sensor-based threats and attacks. Therefore, researchers should investigate the possibility of a common security mechanism for authentication of sensor data as well as authorization of legitimate sensor access.

Fine-grained Control of Sensors - Existing sensor management systems of smart devices offer permission-based sensor management which completely depends on user consent. Apps generally ask for permission to access specific sensors on installation time and once the permissions are granted, users have less control over the sensors' usage by the apps. Again, the user permission is enforced only to secure a limited number of the on-board sensors (e.g., microphone, camera, GPS). Granting permission to these sensors automatically grant permission for other sensors such as accelerometer, gyroscope, light sensor, etc. In recent years, researchers have also shown that both permission-enforced (microphone, camera, GPS) and no permission-enforced (accelerometer, gyroscope, light sensor, etc.) sensors are vulnerable to sensor-based threats and attacks. Therefore, a fine-grained sensor management system is needed to verify compliance between sensor access and user intent.

Further Investigation of Sensor-based Threats. Several prior works have reported many sensor-based threats to smart devices in recent years. However, these sensor-based threats are unique from one another in terms of attack methods, targeted devices, and attack consequences. To understand sensor-based threats, it is important to study the existing threats and use the knowledge to enhance the security of smart devices to tackle new sensor-based threats.

Study of Malicious Sensor Behavior and User Perspectives- Sensor-based threats are relatively new and there are only a few comprehensive studies available to understand the threats properly. This lack of knowledge is lucrative for attackers to target and trick smart device users to install malicious apps and perform malicious sensor activities [SWW15]. Users carelessly install any third-party apps with illegitimate sensor permissions which can compromise smart devices [FHE$^+$12, fel12]. Therefore, to secure sensors in smart devices, it is important to understand how users, smart devices, and apps are using sensors to perform and automate various tasks and what their views of sensor-based threats are. Researchers may perform additional usability studies to better understand how users can contribute to improving sensor access control via their inputs in smart devices.

Prevent Leakage of Sensor Data - Smart devices can autonomously sense their surrounding environment which can be used to prevent information leakage from the devices. Sensors in smart devices can anticipate an on-going task and detect the pattern of information accessed by the task. These sensor patterns vary for different activities and by observing these sensor behaviors, it is possible to prevent information leakage in smart devices [SAU17].

Control Sharing of Data among Sensors - Communication on smart devices become more sensor-to-sensor (i.e., machine-to-machine) compared to human-to-sensor or sensor-to-human (human-to-machine or machine-to-human) and the introduction of a huge number of sensors in smart devices is speeding up this shift. As smart devices deal with sensitive personal data, sensor-to-sensor communication channels should be secured, which helps in end-to-end security for the devices. Secure end-to-end communication from sensors to the devices and among devices is vital to avoid information leakage [HFH15, Web10]. Devices should share encrypted sensor data to avoid any information leakage via packet sniffing [AFA+18]. Sensor data should also be available to all the connected devices continuously to ensure unimpeded performance.

Security Measures for Sensor-based Threats. Currently, there is no comprehensive solutions to detect sensor-based threats in smart devices. The existing solutions focus on specific threats or sensors which are ineffective in addressing sensor-based threats extensively. Researchers and developers should focus on the following open issues to develop effective security measures to detect sensor-based threats properly.

Device Independent Security Measure - The majority of the existing solutions to secure sensors in smart devices focus on smartphone overlooking the security needs of other smart devices [STTPLR13]. However, the number of different smart devices are also increasing rapidly. Several prior works have verified that not only smartphones but all the smart devices (e.g., smart watch, smart home devices, etc.) are vulnerable to emerging sensor-based threats [SBAU19, NSRU19]. Additionally, smartphones can be used as a platform to launch sensor-based threats to other smart devices as smartphones act as controller device for several smart devices such as smart lock, smart camera, etc. [KS16]. Hence, researchers should consider sensor-based threat as a general threat to smart devices to develop device independent security measures.

Protect Sensor Data when at Rest - Smart device applications deal with multiple sensor data at a time and tampered data in the smart devices can impact the normal behavior of applications. To ensure the authenticity of sensor data, various end-to-end encryption mechanisms may be applied from the sensors to the program requesting it. Various security features of the hardware such as ARM TrustZone may be adopted to achieve secure data flow inside the devices [NPDS13]. Researchers may also invest their effort in studying the adoption of the blockchain technology as a way of designing highly distributed systems able to provide attestation and verification among multiparty and heterogeneous components part of a larger smart device ecosystem.

Protect Integrity of Sensor Operations - The research community has not invested enough effort in studying the design and development of tools for automated detection and analysis of sensors-based threats. For instance, no tool is available to automatically identify and analyze adversary-controlled sensors that would compromise the integrity of sensor operations, as well as the integrity of the data generated or modified by such operations. Also, no tool is available to automatically identify dangerous configurations in enforced access control policies, which may lead to risky operations by trusted programs that may compromise the integrity of the entire connected smart device environment.

Adoption of Intrusion Mechanisms to Detect Attacks - In recent years, multiple efficient techniques (e.g., machine learning (ML) and neural network (NN)) were applied to detect threats in various application domains. These detection techniques should be explored in detail to design novel intrusion detection mechanism, for smart devices and applications, able to identify when unsafe operations are authorized. Therefore, researchers should investigate NN and ML classification algorithms as viable solutions to identify and differentiate legitimate from illegal sensing activities.

4.4.2 Recommendations

Vendors - Vendors have to consider the emerging sensor-based threats and attacks and get the security requirements right for every embedded and connected sensors. With the introduction of IoT, sensors can be external devices and connected via different communication means. Hence, vendors need to consider sensors as embedded components as well as independent before implementing security measures. Smart device vendors also should have a strong research strategy to understand the sensor-based threats and attacks and its consequences to secure the devices.

End-users - The main victims of the sensor-based threats and attacks are end-users. Attackers mostly target end-users with less technical knowledge of sensor-based threats to perform malicious activities such as information leakage, task inference, etc. Although it is hard to understand the technical part of different sensor-based threats and attacks, end-users should know the consequences of these threats and attacks and be cautious before using any risky apps in the devices. Additionally, end-users can follow good security practices such as rejecting any suspicious sensor access, disabling automatic data sharing between apps, etc. to secure their devices and information. Users can also raise their concerns to the vendors regarding sensor-based threats and attacks.

Developers - Developers can play an important role in securing smart devices against sensor-based threats and attacks. Modern app-based platforms increase the popularity of smart devices rapidly and developers can build numerous apps and publish them in app markets. To secure the devices from the sensor-based threats and attacks, developers can follow the guidelines published by the vendors to minimize the sensor data abuse in the apps [anda]. Developers can also follow good app developing practices such as the use of encrypted sensor data in the app, trusted data flow path, use of only essential sensor permission, etc. Developers can also help the vendors to build specific security measures against the sensor-based threats and attacks.

Research community - Several on-going research efforts have already confirmed the necessity of securing sensors in smart devices [SAU17, KPM+18]. The research community can help the industry to address the sensor-based threats and attacks efficiently and propose various solutions. Researchers along with the industry experts should jointly propose a standard practice in app development to minimize the sensor abuses in smart devices. Furthermore, researchers should report newly found sensor-based threats to the vendors immediately to reduce the consequences.

Summary - In summary, there are several interesting research problems that may be tackled by the research community toward improving the security of sensors in smart devices and applications. While following the above directions toward better protection mechanisms against the sensor-based threats and attacks, researchers have to identify the key characteristics that differentiate IoT security from the commodity system security. Such unique characteristics will guide toward the design of innovative mechanisms that will be robust against the sensor attacks.

4.5 Conclusion

The growing popularity of topics like smart home, smart office, smart city is increasing attention towards security issues in smart devices and applications. In this chapter, we surveyed a lesser-known yet serious family of *sensor-based threats and attacks to smart devices.* We provided a detailed analysis of recent sensor-based threats and attacks and discussed how these threats and attacks can be used to exploits various sensors in smart devices. We also discussed some of the challenges for future research work in this area. In conclusion, we believe this chapter will have a positive impact in the research community by documenting recent sensor-based threats and attacks to smart devices and motivating researchers to develop further comprehensive security schemes to secure these devices against sensor-based threats and attacks.

CHAPTER 5

SENSOR-BASED THREAT DETECTION IN STANDALONE SMART DEVICES

5.1 Introduction

In this chapter, we present a novel intrusion detection (IDS) framework called *6thSense*, as a comprehensive security solution for sensor-based threats for standalone smart devices (e.g., smartphone, smart watch, etc.). The proposed framework is a *context-aware IDS* and is built upon the observation that for any user activity or task (e.g., texting, making calls, browsing, driving, etc.), a different, but a specific set of sensors becomes active. In a context-aware setting, the 6thSense framework is aware of the sensors activated by each activity or task. Here, context-aware refers to the ability of inferring user activities by tracking different sensors' data of a device. 6thSense observes sensors data in real time and determines the current use context of the device according to whether the current sensor use is malicious or not. 6thSense is context-aware and correlates the sensor data for different user activities (e.g., texting, making calls, browsing, etc.) on the smart devices and learns how sensors' data correlates with different activities. As a detection mechanism, 6thSense observes sensors' data and checks against the learned behavior of the sensors. In 6thSense, the framework utilizes several Machine Learning-based detection mechanisms to catch sensor-based threats including Markov Chain, Naive Bayes, and a set of other ML algorithms (e.g., PART, Logistic Function, J48, LMT, Hoeffding Tree, and Multilayer Perception). In this chapter, we present the design of 6thSense on different Android devices (smartphone and smart watch) because of its open-source nature, large market share [and16], and rich set of sensors. To evaluate the efficiency of the framework, we tested it with data collected from real users (100 different users, 16 different typical daily activities for smartphone and smart watch [act15a] including 153600 and 307200 different event-state information, respectively). We also evaluated the performance of 6thSense against three different sensor-based threats and finally analyzed its overhead. Our evaluation shows that 6thSense can detect sensor-based attacks with an accuracy and F-Score over 96%. Also, our evaluation shows a minimal overhead on the utilization of the system resources. Note that, this work is an extension of our previous work [SAU17]. We significantly improved the framework from our prior work and implemented 6thSense on smart watch and smart phone. We also evaluated the performance with new user data and analyzed the performance overhead in further detail.

Contributions: In summary, the main contributions of this chapter are threefold:

- *First*, the design of 6thSense, a context-aware IDS to detect sensor-based threats in standalone smart devices utilizing multiple machine learning based models from Markov Chain to Naive Bayes to LMT.
- *Second*, the extensive performance evaluation of 6thSense with real user experiments over 100 users for different standalone smart devices (smartphone and smart watch).
- *Third*, testing of 6thSense against three different sensor-based threats.

5.2 Differences from Prior Works

Though there is no direct comparable work to compare 6thSense with, differences between existing solutions and our framework can be noted as follows: The main limitation of Semadroid [XZ15] is that the proposed solution is only tested against a similar type of attack scenario (information leakage by a background application). Semadroid also does not provide any extensive performance evaluation for the proposed scheme. Finally, this work depends on user permissions to fully enforce an updated policy on the sensor usage which is vulnerable as users might unknowingly approve the sensor permissions for malicious Apps. Real-time activity detection proposed by Maiti et al. considers motion sensors to identify user activity on a smart device which is only effective against keystroke inference [MAJH16]. In Darkly [JNS13], the proposed framework is not tested against any sensor-based threats. Audroid presented a policy enforced framework to secure only the audio channels of a smart device. Albeit useful, similar to the others, this work does not consider other sensor-based threats, either. More recent work *AWARE* also considers selective sensors (e.g., camera and microphone) to identify malicious sensor accesses of the applications [PRS$^+$17]. *Compared to these prior works, 6thSense provides a comprehensive coverage to all the sensors in a smart device and ensures security against different types of sensor-based threats with high accuracy.*

5.3 Adversary Model and Design Assumptions

In this section, we discuss different threats that may abuse sensors to execute malicious activities on a smart device. Different design features and assumptions are also explained in this section.

5.3.1 Adversary Model

For this work, we consider the following sensor-based threats similar to [USB14]:

• *Threat 1-Triggering a malicious App via a sensor.* A malicious App can exist in the smart device which can be triggered by sending a specific sensory pattern or message via sensors.

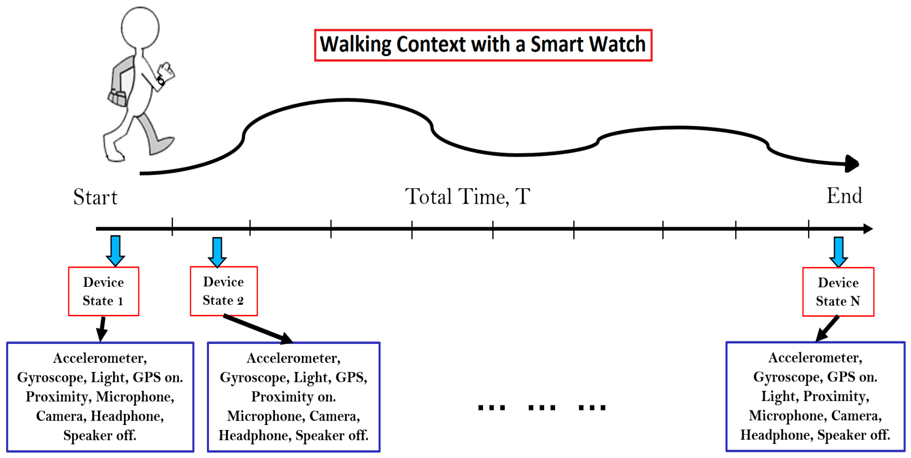
- *Threat 2-Information leakage via a sensor.* A malicious App can exist in the device which can leak information to any third party using sensors.

- *Threat 3-Stealing information via a sensor.* A malicious App can exist in the device which can exploit the sensors of a smart device and start stealing information after inferring a specific device mode (e.g., sleeping).

In this work, we cover these three types of malicious sensor-based threats. To build our adversary model, we consider any component on a smart device that interacts with the physical world as a sensor [PSJA15]. We designed specific malware to represent above-mentioned threats and test our proposed framework against these malware.

5.3.2 Design Assumptions and Features

In designing a comprehensive security scheme like 6thSense for sensor-based threats, we note the following design assumptions and features:



Walking Context with a Smart Watch
Start
Total Time, T
End
Device State 1
Device State 2
Device State N
Accelerometer, Gyroscope, Light, GPS on. Proximity, Microphone, Camera, Headphone, Speaker off.
Accelerometer, Gyroscope, Light, GPS, Proximity on. Microphone, Camera, Headphone, Speaker off.
... ...
Accelerometer, Gyroscope, GPS on. Light, Proximity, Microphone, Camera, Headphone, Speaker off.

Figure 5.1: Context-aware model for 6thSense.

- ***Context Awareness:*** The main feature of 6thSense is context awareness which refers to the ability to sense the physical environment and adapt its operations accordingly in realistic cases [TK12]. 6thSense builds a context-aware model by observing the sensors' behaviors on a smart device in different usage scenarios. When a user is performing a task on a smart device, several sensors (i.e., accelerometer, gyroscope, light sensor, etc.) may remain active. This active state of different sensors is not constant and can change over time. This shifting in sensor's state over time should be considered correctly to understand the context of an activity. 6thSense divides the total execution time of an activity into smaller times and observes the sensors' states (on/off) over a short time span. Thus, whenever a sensor state is changed, 6thSense can understand the context and take a decision according to the context. For example, while a user is walking with a smartphone on his hand, several sensors (i.e., accelerometer, gyroscope, light sensor, etc.) remain active. If we divide the time of the activity in smaller times, we can see different sets of sensors active for different sensor states (Figure 6.1). In this way, 6thSense considers all device states to understand the context of the activity and differentiate between benign and malicious activities.

• ***Sensor co-dependence:*** A sensor in a smart device is normally considered as an independent entity on the device. Thus, one sensor does not know what is happening in another sensor. However, in this work, given an activity, we consider sensors as co-dependent entities on a device instead of independent entities. The reason for this stems from the fact that for each user activity or task on a smart device, a specific set of sensors remains active. For example, if a user is walking with a phone in hand, motion sensors (i.e., gyroscope, accelerometer), the light sensor, GPS will be active. On the contrary, if the user is walking with the phone in the pocket or bag, instead of the light sensor, the proximity sensor will remain active. Thus, a co-dependent relationship exists between sensors while performing a specific task. Each activity uses different, but specific set of sensors to perform the task efficiently. Hence, one can distinguish the user activity by observing the *context of the sensors* for a specific task. 6thSense uses the context of all the sensors to distinguish between normal user activities and malicious activities. In summary, *sensors in a smart device are individually independent, but per activity-wise dependent* and 6thSense considers the context of the activities in its design.

- ***Adaptive sensor sampling:*** Different sensors have different sampling frequencies. To monitor all the sensor data for a specific time, a developed solution must consider and sample the sensor data correctly. 6thSense considers sampling the sensor data over a certain time period instead of individual sensor frequencies which mitigates any possible error in processing of data from different sensors. 6thSense collects each sensor data separately and samples the data according to their corresponding frequencies. These sample data are merged together to build contexts of different user activities in smart devices.

- ***Faster computation:*** Modern high precision sensors on smart devices have high resolution and sampling rate. As a result, sensors provide large volume of data even for a small time interval. A solution for sensor-based threats should quickly process any large data from different sensors in real time while ensuring a high detection rate. To address this, we use different machine learning algorithms as detection techniques of 6thSense which are proven simple and fast techniques.

- ***Real-time monitoring:*** 6thSense provides real-time monitoring to all the sensors which mitigates the possibility of data tempering or false data injection on the device.

- ***Configurability:*** 6thSense is configurable to provide different needs and flexible deployments. For example, 6thSense offers both online and offline training mode for different machine learning detection techniques to reduce power consumption.

5.4 Detection Techniques: Theoretical Foundation of 6thSense

In this section, we describe the theoretical foundation of the detection techniques used in 6thSense. For the context-aware IDS in 6thSense, we utilize several different ML-based techniques including Markov Chain, Naive Bayes and, a set of other ML algorithms (e.g., PART, Logistic Function, J48, LMT, Hoeffding Tree, and Multilayer Perception) to differentiate between normal and malicious behavior on a smart device.

As explained in Section 4, we consider the context awareness of user activities in a smart device which shows state transition and sensor co-dependence feature in a smart device. The Markov Chain model can illustrate these properties of the smart device's sensors accurately based on different user activities in the transition matrix. Another advantage of using Markov Chain model is that it is easy to build the model from a large dataset and computational requirements are modest which can be met by resource-limited devices. On the other hand, the Naive Bayes model can build multiple activity contexts from sensor data and identifies whether a test dataset belongs to a user activity or a malicious activity. The Naive Bayes model uses the sensor co-dependence feature to build the activity context and classifies data accordingly. In addition to this, the Naive Bayes technique is chosen for its fast computation rate, small training dataset requirement, and ability to modify it with new training data without rebuilding the model from scratch.

Apart from the Markov Chain and the Naive Bayes model, other ML techniques are also common in malware detection because of their high accuracy rate [YLAI17, SPA$^+$18]. We also investigate how other ML algorithms perform in building a context-aware model from sensor data and detecting sensor-based threats on a smart device. Our main purpose is to check whether popular ML algorithms can understand and build an effective context-aware model for sensor-based threats. A discussion of these approaches in the context of 6thSense is given below. The efficacy of these different approaches utilized in 6thSense is analyzed in Section 7.

5.4.1 Markov Chain-Based Detection

Markov Chain model can be described as a discrete-time stochastic process which denotes a set of random variables and defines how these variables change over time. There are two main assumptions for Markov Chain model: (1) Probability distribution of the state at time *t+1* depends on the state at time t only. Here, the state refers to the overall condition of the stochastic process. (2) A state transition from previous timestamp (t) to next timestamp (*t+1*) is independent of time. Markov Chain can be applied to illustrate a series of events where what state will occur next depends only on the previous state. In our study, a series of events represents user activity and state represents condition (i.e, values, on/off status) of the sensors in a smart device (Figure 5.2). We can represent the probabilistic condition of Markov Chain as in Equation 1 where X_t denotes the state at time t [Kei12].

$$\begin{gathered} P(X_{t+1} = x | X_1 = x_1, X_2 = x_2 ..., X_t = x_t) = P(X_{t+1} = x | X_t = X_t), \\ when,\ P(X_1 = x_1, X_2 = x_2 ..., X_t = x_t) > 0. \end{gathered} \tag{5.1}$$

In our study, we observe the changes of condition of a set of sensors as a variable which changes over time. The condition of a sensor indicates whether the sensor value is changing or not from a previous value in time. Let us assume S denotes a set which represents current conditions of n number of sensors. So, S can be represented with $S = \{S_1, S_2, S_3, ..., S_n\}$, where $S_1, S_2, S_3, ..., S_n = 0 \ or \ 1$. For a specific time, t, we consider the combination of all the sensors' conditions in the smart device as the state of our model. As we consider change in a sensor's condition as binary output (1 or 0, where 1 denotes that sensor value is changing from previous instance and 0 denotes that sensor value is not changing), the number of total states of our model

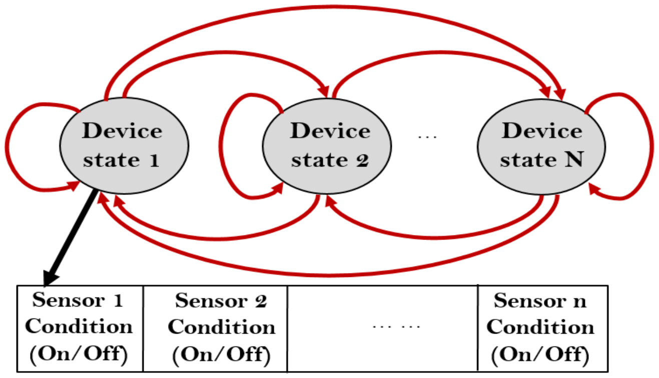

Device state 1
Device state 2
...
Device state N
Sensor 1 Condition (On/Off)
Sensor 2 Condition (On/Off)
... ...
Sensor n Condition (On/Off)

Figure 5.2: Markov Chain model for 6thSense

will be exponents of 2. For example, if we consider the total number of sensors in set S is 10, the number of states in our Markov Chain will be 2^{10} and the states can be represented as a 10 bit binary number where each bit will represent the state of a corresponding sensor. For this, p_{ij} denotes the probability that the system in a state j at time *t+1* given that system is in state i at time t. If we have n number of sensors and $m = 2^n$ states in our model, Markov Chain can be constructed by the following transition probability matrix:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & \cdots & p_{1m} \\ p_{21} & p_{22} & p_{23} & \cdots & \cdots & p_{2m} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{m1} & p_{m2} & p_{m3} & \cdots & \cdots & p_{mm} \end{bmatrix} \tag{5.2}$$

The transition probability matrix of this Markov Chain can be constructed by observing the transitions from one state to another state for a certain time. Assume that, system's states are $X_0, X_1, \ldots, X_T$ at a given time $t = 0, 1, \ldots, T$. We can represent the transition probability matrix as follows:

$$P_{ij} = \frac{N_{ij}}{N_i}, \tag{5.3}$$

where, N_{ij} = the number of transition from X_t to X_{t+1} where X_t in state i and X_{t+1} in state j; N_i = the number of transition from X_t to X_{t+1}, where X_t in state i and

X_{t+1} in any other state. The initial probability distribution of this Markov Chain can be represented as follows:

$$Q = \begin{bmatrix} q_1 & q_2 & q_3 & \dots & \dots & q_m \end{bmatrix} \tag{5.4}$$

where q_m is the probability that the model is in state m at time 0. Probability of observing a sequence of states $X_1, X_2, \dots, X_T$ at a given time $1, \dots, T$ can be computed using the following equation:

$$P(X_1, X_2, \dots, X_T) = q_{x1} \prod_{2}^{T} P_{X_{t-1}Xt} \tag{5.5}$$

For 6thSense, instead of predicting the next state, we determine the probability of occurring a transition between two states at a given time. We train our Markov Chain model with a training dataset collected from real users and build the transition matrix accordingly. Then, we determine sensor working condition for time *t* and *t+1*. Let us assume *a* and *b* are sensor's state in time *t* and *t+1*. We determine the probability of transition from state *a* to *b* which can be found by looking up in the transition matrix and calculating *P(a,b)*. As the training dataset consisted of sensor data from benign activities, we can assume that if transition from state *a* to *b* is malicious, the calculated probability from transition matrix will be zero.

5.4.2 Naive Bayes Based Detection

The Naive Bayes model is a simple probability estimation method which is based on Bayes' method. The main assumption of the Naive Bayes detection is that the presence of a particular sensor condition in a task/activity has no influence over the presence of any other feature on that particular event. The probability of each event can be calculated by observing the presence of a set of specific features.

Assume $p(x_1, x_2)$ is the general probability distribution of two events x_1, x_2. Using the Bayes rule, we can have the following equation:

$$p(x_1, x_2) = p(x_1|x_2)p(x_2), \tag{5.6}$$

where $p(x_1|x_2)$ = Probability of the event x_1 given that event x_2 will happen. Now, if we have another variable, c, we can rewrite Equation 7 as follows:

$$p(x_1, x_2|c) = p(x_1|x_2, c)p(x_2|c). \tag{5.7}$$

If knowledge of c is sufficient enough to determine the probability of event x_1, we can state that there is conditional independence between x_1 and x_2 [MS12]. So, we can rewrite the first part of Equation 8 as $p(x_1|x_2, c) = p(x_1|c)$, which modifies Equation 8 as follows:

$$p(x_1, x_2|c) = p(x_1|c)p(x_2|c). \tag{5.8}$$

In 6thSense, we consider users' activity as a combination of n number of sensors (Figure 5.3). Assume X is a set which represents current conditions of n number of sensors. We consider that conditions of sensors are conditionally independent (See Section 4.2), which means a change in one sensor's working condition has no effect over a change in another sensor's working condition. As we explained earlier, the probability of executing a task depends on the conditions of a specific set of sensors. So, in summary, although one sensors' condition does not control another sensor's condition, overall probability depends on all the sensors' conditions. For example, if a person is walking with his smartphone in his hand, the motion sensors (accelerometer and gyroscope) will change. However, this change will not force the light sensor or the proximity sensor to change its condition. Thus, sensors in a smart device change their conditions independently, but execute a task together. From Equation 9, we can have a generalized formula for this context-aware model [MS12]:

$$p(X|c) = \prod_{i=1}^{n} p(X_i|c). \tag{5.9}$$

In our contextual activity-oriented model, we have a set of training data for users' activities. Assume that B represents a set which denotes m numbers of user activities.

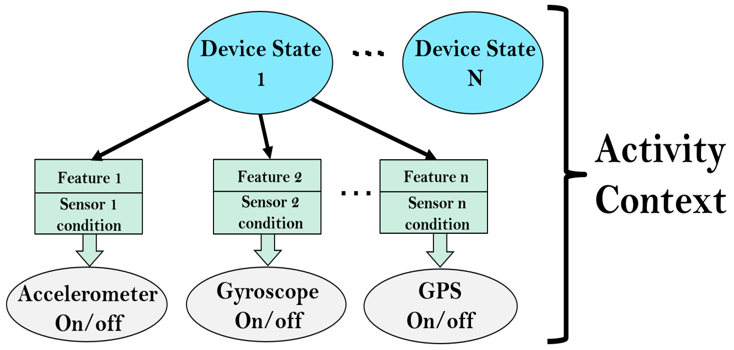


Figure 5.3: Naive Bayes model for 6thSense

We can determine the probability of a dataset X to be classified as a user activity using the following equation:

$$P(B_i|X) = \frac{P(X|B_i)P(B_i)}{P(X)}, \tag{5.10}$$

where i = 1, 2, . . . , m. As the sum of all the conditional probabilities for X will be 1, we can have the following equation which will lead to Equation 12—

$$\sum_{i=1}^{m} P(B_i|X) = 1. \tag{5.11}$$

$$P(B_i|X) = \frac{P(X|B_i)P(B_i)}{\sum_{i=1}^{m} P(X|B_i)P(B_i)}. \tag{5.12}$$

This calculated conditional probability then is used to determine the benign user activity or malicious attacks in 6thSense. In this way, we compute the probability of occurring an activity over a certain period of time.

We divide the sensor data into smaller time values (1 second) and calculate the probability of each instances to infer the user activity. The calculated probability per second data is then used in the expected value theorem to calculate total probability. If the probability of the first instance is p_1 with a value of a_1, probability of the second instance is p_2 with a value of a_2 and so on, up to value a_n, the expected value can be calculated by the following formula:

$$E[N] = \frac{a_1p_1 + a_2p_2 + a_3p_3 + \ldots\ldots + a_np_n}{a_1 + a_2 + \ldots\ \ldots + a_n}. \tag{5.13}$$

As all the values of a_1, a_2, ..., a_n are equally likely, this expected value becomes a simple average of cumulative probability of each instances. We infer the user activity by setting up a configurable threshold value in the 6thSense framework and checking whether calculated value is higher than the threshold or not. If it is lower than the threshold value, a malicious activity is occurring in the device.

5.4.3 Other ML-based Detection Techniques

In addition to the Markov Chain and the Naive Bayes models above, there are other machine learning algorithms (such as PART, Logistic Function, J48, LMT, Hoeffding Tree, and Multilayer Perception) that are very popular for anomaly detection schemes because of their fast computation and easy implementation.

In the alternative detection techniques, we used four types of ML-based classifier to build a context-aware analytical model for 6thSense. The following briefly discusses these classifiers and our rationale to include them:

Rule-based Learning. Rule-based ML works by identifying a set of relational rules between attributes of a given dataset and represents the model observed by the system [GPY+07]. The main advantage of the rule-based learning is that it identifies a single model which can be applied commonly to any instances of the dataset to make a prediction of outcome. As we train 6thSense with different user activities, the rule-based learning provides one model to predict data for all the user activities which simplifies the framework. For 6thSense, we chose, *PART* algorithm for the rule-based learning.

Regression Model. Regression model is widely used in data mining for its fast computation. This type of classifier observes the relations between dependent and independent variables to build a prediction model [SKE+12]. For 6thSense, we have a total 11 attributes where we have one dependent variable (device state: malicious/benign) and ten independent variables (sensor conditions). Regression model observes the change in the dependent variable by changing the values of the independent variables and build the prediction model. We use the logistic regression model in 6thSense, which also yields with high accuracy against conventional Android malware [SKE$^+$12].

Neural Network. Neural network is another common technique that is utilized by researchers for malware detection. In neural network techniques, the relation between attributes of dataset is compared with the biological neurons and a relation map is created to observe the changes for each attribute [LVM09]. We chose *Multilayer Perceptron* algorithm for training the 6thSense framework as it can distinguish relationships among non-linear dataset.

Decision Tree. Decision tree algorithms are predictive models where decision maps are created by observing the changes in one attribute in different instances [YWLY07]. These types of algorithms are mostly used in a prediction model where output can have a finite set of values. For 6thSense, we utilized and tested three different decision tree algorithms (*J48*, *LMT (Logistic Model Tree)*, and *Hoeffding tree*) to compare the outcome of our framework.

5.5 6thSense Framework

In this section, we provide a detailed overview of our proposed context-aware IDS framework, 6thSense, for detecting sensor-based threats on smart devices. As illustrated in Figure 5.4, 6thSense has three main phases: (1) data collection, (2) data processing, and (3) data analysis. In the Data Collection phase, we use a custom Android App to collect the sensor data for different user activities and the collected sensor data are then processed in the Data Processing phase. In Phase 3, the collected data is fed into detection models and the end result indicates whether the current state of the device is malicious or not. The following sub-sections briefly describe these three phases.

5.5.1 Data Collection Phase

In this phase, 6thSense collects data from different sensors of a smart device. There can be multiple sensors in a smart device. 6thSense considers nine sensors in total to identify different user activities using a sensor-rich Android device. The sensors selected are accelerometer, gyroscope, light sensor, proximity sensor, GPS, audio sensor (microphone and speaker), camers, and headphone. The chosen sensors are then categorized into two following categories.

No-permission-imposed sensors in 6thSense: For 6thSense, we chose four no-permission imposed sensors (i.e., accelerometer, gyroscope, light, proximity sensors). We can also refer these sensors as data-oriented sensors in the context of 6thSense because values provided by these sensors need to be observed to infer user activities. For example, accelerometer's and gyroscope's values change with motion and they give values on *X, Y,* and *Z* axes. To detect whether a sensor is activated or not for a specific activity, one needs to observe values of these sensors.

Permission-imposed sensors in 6thSense: We chose five permission-imposed sensors to build the context-aware model (microphone, GPS, speaker, camera, and headset) of 6thSense. The conditions of these sensors can be represented by their logical states (on/off status) for different user activities. Hence, we also referred to these sensors as logic-oriented sensors in the context of 6thSense. For example, microphone has only two values to identify users' activity: on and off. So, it can be represented with 0 or 1 to detect if the camera is on or off correspondingly.

To collect the data and logical values from sensors, we built a custom Android App and 6thSense used this in the data collection phase. In Android, this App uses *sensoreventlistener* API to log numerical values of the data-oriented sensors. On the other hand, the App determines the state of the sensor and logs 0 or 1 if the sensor is on or off, respectively. This App uses the user permission to use the microphone, GPS, and camera to record the working conditions of these sensors. For GPS, we consider two datasets - either GPS is turned on or not and either location is changing or not. In total, six different logic state information for five aforementioned permission-imposed sensors are collected by this App.

Note that 6thSense considers different typical daily human activities [act15b] that involve the smart devices (e.g., smart watch, smart phone, etc.) to build the contextual model. These activities include walking, talking, interacting (playing games, browsing), driving (as driver and passenger). Furthermore, the number of activities is configurable in 6thSense and is not limited to aforementioned examples. As also explained in the evaluation of 6thSense, a total of seven and nine typical daily activities are selected for smart watch and smart phone respectively as they are considered as common user activities [act15a]. 6thSense collects these data using the App for different users to train the framework which is then used to distinguish the normal sensor behavior from the malicious behavior. In summary, the aforementioned App collects data from eight different sensors for different typical user activities. 6thSense observes sensor state (combination of working conditions (i.e., values, on/off status) of different sensors) in a per second manner for each user activity. Each second of data for user activity corresponds to 512 state information from eight different sensors.

5.5.2 Data Processing Phase

In the second phase of the framework, 6thSense organizes the data to use. As different sensors have different frequencies on the smart device, the total number of readings of sensors for a specific time period is different. For example, the accelerometer and gyroscope of *LG Watch Sport* have a sampling frequency of approximately 418 Hz and 32 Hz, respectively while the light sensor has a sampling frequency of 5 Hz. Thus, the data collected in Phase 1 needs to be sampled and reorganized. 6thSense observes the change in the sensor condition in each second to determine the overall state of our device and from this per second change, 6thSense determines the activity of users. For this reason, 6thSense takes all the data given by a single sensor in a second and calculates the average value of the sensor reading. This process is only applicable for the data oriented sensors as mentioned earlier. Again, the data collected from the App is numerical value as given by the sensor. However, for the detection model, 6thSense only considers the condition of the sensors. 6thSense observes the data collected by the aforementioned App and determines whether the condition of sensors is changing or not. If the sensor value is changing from the previous value in time, 6thSense represents the sensor condition as 1 and 0 otherwise.

The logic state information collected from the sensors need to be reorganized, too as these data are merged with the data collected from the collected values from the other sensors to create an input matrix. We consider the condition of the sensors to be the same over time and organize the data accordingly. The reorganized data generated from the aforementioned App are then merged to create the training matrices.

5.5.3 Data Analysis Phase

In the third, 6thSense uses different ML-based detection techniques introduced in the previous section to analyze the data matrices generated in the previous phase.

For the Markov Chain-based detection, 6thSense uses 75% of the collected data to train 6thSense and generate the transition matrix. This transition matrix is used to determine whether the transition from one state to another is appropriate or not. Here, state refers to generic representation of all the sensors' conditions on a device.

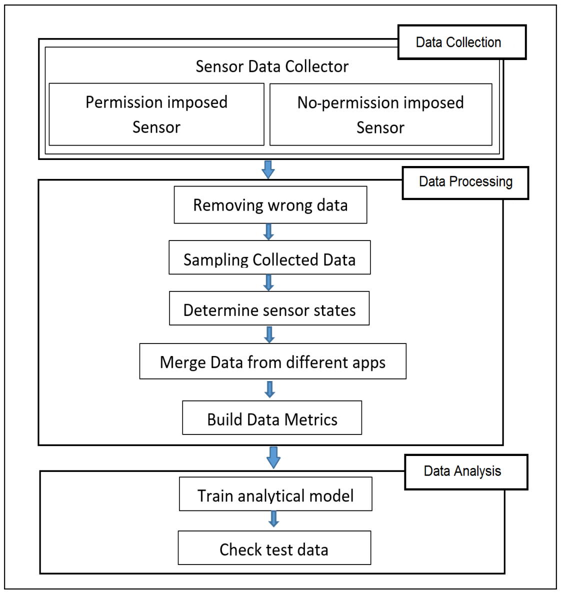


Figure 5.4: Overview of 6thSense.

For testing purposes, we have two different data set — benign activities or trusted model and malicious activities or threat model. The trusted model consists of 25% of the collected data for different user activities. We tested the trusted model to ensure the accuracy of the 6thSense framework in detecting benign activities. The malicious activities are built from performing the attack scenarios mentioned in Section 5.3. 6thSense calculates the probability of a transition occurring between two states at a given time and accumulates the total probability to distinguish between normal and malicious activities.

To implement the Naive Bayes-based detection technique, 6thSense uses the training sessions to define different user activities. In 6thSense, seven typical user activities are selected in total for smart watch as listed in Table 3. In addition to these user activities, we consider walking with smart device in pocket and making a video call as typical user activities to test 6thSense in smart phone. 6thSense uses ground truth user data to define these activities. Using the theoretical foundation explained in Section 5.4, 6thSense calculates the probability of a test session to belong to any of these defined activities. As 6thSense considers one second of data in each computational cycle, the total probability up to a predefined configurable time interval (in this case five minutes) is calculated. This calculated probability is used to detect malicious activities from normal activities. If the computed probability for all the known benign activities is not over a predefined threshold, then it is detected as a malicious activity.

For the other alternative machine-learning-based detection techniques, 6thSense uses WEKA, a data mining tool which offers data analysis using different machine learning approaches [HFH$^+$09].

5.6 Performance Evaluation of 6thSense

Sensor type	**Name**	**Model (Smart Watch | Smart Phone)**	**Specification (Smart Watch | Smart Phone)**
No-permission imposed sensors	Accelerometer	Bosch BMI160 Acceleration Sensor | MPU6500 Acceleration Sensor	78.4532 m/s^2, 417.67 Hz, 0.01 mA | 19.6133 m/s^2, 203.60 Hz, 0.25 mA
	Gyroscope	Bosch BMI160 Gyroscope Sensor | MPU6500 Gyroscope Sensor	17.453293 rad/s, 31.95 Hz, 0.01 mA | 8.726646 rad/s, 203.60 Hz, 6.1 mA
	Light Sensor	APDS-9306 Light Sensor | TMG399X RGB Sensor	30000 lux, 5 Hz, 0.11 mA | 600000 lux, 5.62 Hz, 0.75 mA
	Proximity Sensor	LG Wear Detection Sensor | TMG399X proximity sensor	1V, 0.15 mA | 8V, 0.75 mA
Permission-imposed sensors	Microphone	Qualcomm Snapdragon Wear 2100 built in microphone | Qualcomm Snapdragon 801 Processor built in microphone	120 dB, .12 mA | 86 dB, .75 mA
	Speaker	Qualcomm Snapdragon Wear 2100 built in speaker | Qualcomm Snapdragon 801 Processor built in speaker	90 dB, .18 mA | 110 dB, 1 mA
	Camera	N/A | Samsung S5K2P2XX	N/A | 12 megapixels, 30 fps, 4.7 mA

Table 5.1: Sensor list of LG Watch Sport and Samsung Galaxy S5 Duos smartphone used in experiment.

In this section, we evaluate the efficiency of the proposed context-aware IDS framework, 6thSense, in detecting the sensor-based threats on smart devices (smartphone and smart watch). We tested 6thSense with the data collected from different users for benign activities and adversary models described in Section 5.3. As discussed earlier, 6thSense considers three sensor-based threats: (1) a malicious App that can be triggered via a light or motion sensors, (2) a malicious App that can leak information via audio sensors, and (3) a malicious App that steals data via audio sensors. Furthermore, we measured the performance impact of 6thSense on the devices and present a detailed results for the efficiency of the 6thSense framework on both a smart watch and smart phone.

5.6.1 Training Environment and Dataset

In order to test the effectiveness of 6thSense, we implemented it on both a sensor-rich Android-based smart watch and smartphone. We used the *LG Watch Sport* as a reference Android smart watch with *Android Wear version 2.0* to collect sensor data for different typical user activities. We chose this Android device as the LG watch sport is a second generation, stand-alone Android wearable that provides a rich set of sensors. A list of sensors of *LG Watch Sport* is given in Table 5.1. As discussed earlier, we selected 7 different typical user activities or tasks to collect user data (Table 3). These are typical basic activities with the smart watches that people usually do in their daily lives [act15a]. The user activities/tasks are categorized in two categories as generic activities and user related activities.

Generic activities are the activities in which the sensor readings are not affected by the smart device users. Sleeping wearing smart watch, driving with the smart watch using GPS as a navigator, and driving with smart watch in hand are three generic activities that were considered in this work. Basically, in the generic activities, sensors' data are not affected by different users since users do not interact with the smart watch directly. For example, if a user is sleeping, sensors activity will be irregular depending on sleeping pattern. There will be less movement detected in the device and sensor data will be changed accordingly. For *user-related activities*, in which the sensor readings may be affected by the user, we identified four different activities including walking, playing games, browsing, and making voice calls via smart watch.

For implementing and evaluating the performance of 6thSense on smartphone, we chose *Samsung Galaxy S5 Duos* with Android OS version 7.1.2 (Android N) which provides a broad range of sensors. Samsung currently holds approximately 23.3% of total market share of smartphones [sam17] and is the largest Android operated smartphone manufacturer which motivates to implement 6thSense on Samsung smartphone. In addition to user activities used in the smart watch data collection, we considered two more user-related activities (walking with the device in pocket/bag and making video calls) for testing 6thSense on the smartphone.

6thSense was tested by 100 different individuals (50 smart watch users and 50 smartphone users) while the sensor data was collected from the smart watch and the smartphone. We note that our study with human subjects was approved by the appropriate Institutional Review Board (IRB) and we followed all the procedures strictly in our study. To train and test 6thSense on the smart watch, we collected 200 sets of data for four user-related activities for the smart watch where each dataset comprised of 300 seconds of data from the selected sensors mentioned in Section 5.5. We also collected three sets of data for each general activity. We asked the different users to perform the same activity to ensure the integrity for different tasks. We also asked the users to perform the tasks naturally without any influence of the lab environment. Users performed these tasks in a real-life workplace and outdoor in a natural environment. Additionally, users chose their preferred place, walking routes, and apps in the entire data collection process. For example, to collect data in walking scenario, users chose their preferred walking routes both inside their workplace and outside environment. Note that each five minutes of the data collected for user-related and generic activities corresponds to 300 events with 512 different states. So, a total of 153,600 different event-state information were analyzed by 6thSense for a user activity. For testing 6thSense on a smartphone, we collected data from 50 different individuals for nine different activities. We considered nine different sensors to build the context-aware model and each dataset depicted 300 events with 1024 different states and a total of 307,200 event-state information [SAU17].

For the malicious dataset, we created three different attack scenarios considering the adversary model mentioned in Section 4. For Threat 1, we developed two different Android Apps which could be triggered using the light sensor and motion sensors on the smart watch. We also created the same malicious Android app for the smart phone. To perform the attack described in Threat 2, we developed a malware that could record conversations as audio clips and playback after a specific time to leak the information. This attack scenario included both the microphone and speaker on the smart watch and smart phone. We developed another version of this app which could record conversations as audio clips in smartphone using a connected smart watch. Also, for Threat 3, we developed a malicious App that could scan all the sensors and if none of the sensors were changing their working conditions, the malicious App could open up the microphone and record audio clips surreptitiously. For Threat 3, we developed another version for smart devices with camera (e.g., smartphone) where a malicious App can scan all the sensors of a device and if device was inactive, the malicious App could activate camera and record videos covertly. We developed an updated version of this attack which could start recording via microphone in a smart watch if the connected smartphone was inactive. This version of the app could bypass the security feature introduced on Android P [and18b]. In summary, we created 10 different malware that could perform malicious activities in Android-powered smart phone and smart watch. We collected 18 different datasets (a total of 62,850 event-state information) from these three attack scenarios to test the efficacy of 6thSense against these adversaries in a smart watch.

In order to test 6thSense, we divided the collected real user data into two sections as it is a common practice [URC+13]. 75% of the collected benign dataset was used to train the 6thSense framework and 25% of the collected data along with malicious dataset were used for testing purposes. For the Markov Chain-based detection technique, the training dataset was used to compute the state transitions and to build the transition matrix. On the other hand, in the Naive Bayes-based detection technique, the training dataset was used to determine the frequency of sensor condition changes for a particular activity or task. As noted earlier, for the smart watch, there were seven activities for the Naive Bayes technique. We split the data according to their activities for this approach. For the analysis of the other ML-based approaches,

Task Category	**Task Name**
Generic Activities	1. Sleeping
	2. Driving as driver
	3. Driving as passenger
User-related Activities	1. Walking with smart watch in hand
	2. Playing games
	3. Browsing
	4. Making phone calls
	5. Walking with device in pocket/bag†
	6. Making video calls†

† Only considered for smart phone.

Table 5.2: Typical activities of users on a smart device [act15a].

the data in benign and malicious classes were used to train and test 6thSense using 10-fold cross validation for different ML algorithms.

	Smart Watch						**Smart Phone**					
Threshold[†]	**Recall**	**FN**	**Precision**	**FP**	**Accuracy**	**F-score**	**Recall**	**FN**	**Precision**	**FP**	**Accuracy**	**F-score**
0	0.66	0.33	1	0	0.77	0.79	0.62	0.38	1	0	0.68	0.76
1	0.77	0.22	1	0	0.85	0.87	0.86	0.14	1	0	0.88	0.92
2	0.88	0.11	1	0	0.92	0.94	0.96	0.04	1	0	0.96	0.97
3	0.97	0.02	0.98	0.01	0.97	0.98	0.98	0.02	1	0	0.98	0.98
5	0.99	0.001	0.89	0.11	0.96	0.94	1	0	0.9	0.1	0.98	0.94
6	1	0	0.84	0.16	0.95	0.92	1	0	0.8	0.2	0.96	0.89

[†] Number of consecutive malicious state is considered as threshold

Table 5.3: Performance evaluation of Markov Chain based model.

5.6.2 Performance Metrics

In the evaluation of 6thSense, we utilized the following six different performance metrics: Recall rate (sensitivity or True Positive rate), False Negative rate, Specificity (True Negative rate), False Positive rate, Accuracy, and F-score. True Positive (TP) indicates number of benign activities that are detected correctly while true negative (TN) refers to the number of correctly detected malicious activities. On the other hand, False Positive (FP) states malicious activities that are detected as benign activities and False Negative (FN) defines number of benign activities that are categorized as malicious activity. F-score is the performance metric of a framework that reflects the accuracy of the framework by considering the recall rate and specificity. These performance metrics are defined as follows:

$$Recall\ rate\ (TP\ Rate) = \frac{TP}{TP+FN}, \tag{5.14}$$

$$False\ negative\ rate = \frac{FN}{TP+FN}, \tag{5.15}$$

$$Precision\ rate\ (TN\ rate) = \frac{TN}{TN+FP}, \tag{5.16}$$

$$False\ positive\ rate = \frac{FP}{TN+FP}, \tag{5.17}$$

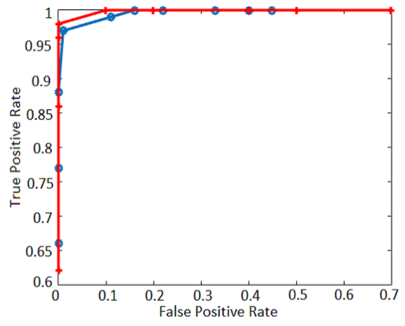
$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \tag{5.18}$$

$$F-score = \frac{2 * Recall\ rate * Precision\ rate}{Recall\ rate + Precision\ rate}. \tag{5.19}$$

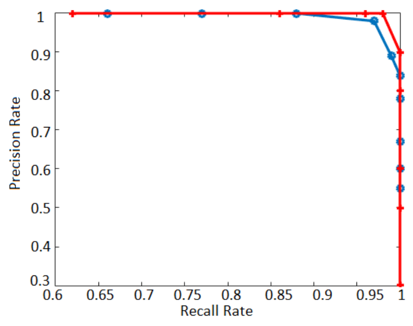
In addition to the aforementioned performance metrics, we considered Receiver Operating Characterstic (ROC) curve and Precision Recall Curve (PRC) as other performance metrics for 6thSense. As our collected dataset is imbalanced (number of benign events is higher than the malicious events), the accuracy of the framework can be influenced by the dataset. To address data imbalance problem in 6thSense, we used PRC as a performance metric which considers data imbalance and reflects the base-rate fallacy correctly [RDL+15].

5.6.3 Evaluation of Markov Chain-Based Detection

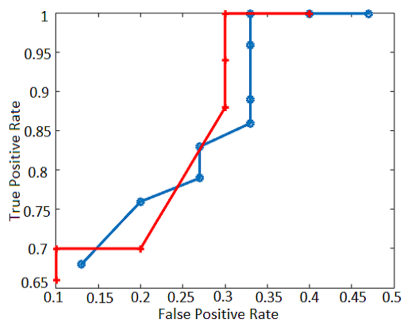
In the Markov Chain-based detection technique, we question whether the transition between two states (sensors' on/off condition in each second) is expected or not. In the evaluation, we used 66 testing sessions in total for the smart watch, among which 51 sessions were for the benign activities (both generic and user-related activities)



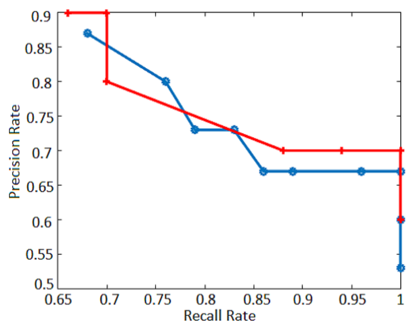
(a) ROC curve for Markov Chain



(b) PRC curve for Markov Chain



(c) ROC curve for Naive Bayes



(d) PRC curve for Naive Bayes

Figure 5.5: ROC curve and PRC curve of different detection techniques on smart watch (——) and smart phone (——).

and the rest of the sessions were for the malicious activities. For evaluation in the smartphone, we have 80 testing sessions in total (65 benign sessions). A session is composed of a series of sensory context conditions where a sensory context condition is the set of all available sensor conditions (on/off state) for different sensors. As discussed earlier in Section 6, a sensor condition is a value indicating whether the sensor data is changing or not. In this evaluation, the sensory context conditions were computed every one second. For Markov Chain-based detection, we referred each sensory context condition as state of the device of that particular moment. 6thSense provides both online and offline training method to reduce performance overhead of the resource-constrained devices. As the highest battery life is 430 mAh for LG watch sport, training with different user data will consume more power which will increase power-accuracy trade-off of our framework; hence, we chose offline training method [LG17]. For the test dataset, we used the transition matrix generated from the training period to determine whether transition from one state to another is malicious or not. We observed that in real devices, sometimes some sensor readings would be missed or real data would not be reflected due to hardware or software imperfections. And, real malicious Apps would cause consecutive malicious states on the device. Therefore, to overcome this, we also kept track of number of consecutive malicious states and used it as a threshold after which the session was considered as malicious. Table 5.3 displays the evaluation results associated with the Markov Chain-based detection technique. When the threshold for consecutive malicious states is 0, i.e., when no threshold is applied, the accuracy is just 77% and FNR is as high as 33%. With increasing the threshold value, the accuracy first increases up to 97% then starts decreasing.

The logical cut-off threshold should be three consecutive malicious occurrences which has both accuracy and F-score over 97%. In Table 5.3, different performance indicators for Markov Chain based detection are also presented. We can observe that FN and TN rates of Markov Chain-based detection decrease as the threshold of consecutive malicious states increases. Again, both accuracy and F-score reach to a peak value with the threshold of three consecutive malicious states on the device. From Figure 5.5, we can see that FP rate remains zero while TP rate increases at the beginning. The highest TP rate without introducing any FP case is over 88%. After 88%, it introduces some FP cases in the system. For the cut-off threshold of three consecutive malicious occurrences, TP rate of 6thSense increases over 97% with FP rates as low as 0.01%.

Table 5.3 also depicts evaluation of Markov chain model on the smartphone. Similar to the smart watch, TP rate and FP rate increase with consecutive malicious occurrences and FN and TN decrease with the threshold on a smartphone. The plausible cut-off threshold should be three consecutive malicious occurrences which is the same for the smart watch. The peak accuracy and F-score can be achieved for this threshold value which is over 98%. From Figure 5.5, we can also notice that the highest possible TP rate without introducing any FP cases is 98%. Figure 5.5(b) shows PRC curve for Markov Chain-based detection on both the smartwatch and the smart phone. We can see that for both the smart watch and the smartphone, area under PRC are approximately 1 which is ideal result for our imbalanced dataset. In summary, Markov Chain-based detection in 6thSense can acquire accuracy over 97% and auPRC approximately 1 with low FP rates (1.43%) for both the smart watch and the smartphone.

5.6.4 Evaluation of Naive Bayes-based Detection

In the Naive Bayes-based detection technique, 6thSense calculates the probability of a session to match it with each activity defined in Section 5.6.1. Here, 6thSense

	Smart Watch						Smart Phone					
Threshold[†]	Recall	FN	Precision	FP	Accuracy	F-score	Recall	FN	Precision	FP	Accuracy	F-score
55%	1	0	0.53	0.47	0.96	0.69	1	0	0.6	0.4	0.93	0.75
57%	1	0	0.6	0.4	0.96	0.75	1	0	0.7	0.3	0.95	0.82
60%	1	0	0.67	0.33	0.97	0.80	1	0	0.7	0.3	0.95	0.82
62%	0.96	0.04	0.67	0.33	0.94	0.79	1	0	0.7	0.3	0.95	0.82
65%	0.89	0.11	0.67	0.33	0.87	0.76	0.94	0.06	0.7	0.3	0.9	0.80
67%	0.86	0.14	0.67	0.33	0.85	0.75	0.88	0.12	0.7	0.3	0.85	0.78

[†] Calculated expected probability is considered as threshold.

Table 5.4: Performance evaluation of Naive Bayes model.

	Smart Watch						Smart Phone					
Algorithms	Recall	FN	Precision	FP	Accuracy	F-score	Recall	FN	Precision	FP	Accuracy	F-score
PART	0.98	0.012	0.69	0.30	0.98	0.80	0.99	0.01	0.65	0.35	0.99	0.79
Logistic Function	0.99	0.01	0.35	0.65	0.97	0.49	0.99	0.01	0.28	0.72	0.99	0.43
J48	0.99	0.01	0.71	0.29	0.99	0.81	0.99	0.01	0.65	0.35	0.99	0.79
LMT	0.99	0.01	0.95	0.05	0.99	0.97	0.99	0.01	0.93	0.07	0.99	0.96
Hoeffding Tree	1	0	0.07	0.93	0.99	0.12	1	0	0.06	0.94	0.99	0.11
Multi-layer Perceptron	0.99	0.01	0.65	0.35	0.98	0.81	0.99	0.01	0.69	0.31	0.99	0.82

Table 5.5: Performance of other different machine learning based-detection techniques tested in 6thSense.

checks the calculated probability of an activity from dataset against a threshold to determine the correct activity. If there is no match for a certain sensor condition with any of the activities, 6thSense detects the session as malicious. Table 5.4 shows the evaluation results.

For the smart watch, for a threshold value of 55%, FN rate is zero. However, FPR is too high (47%), which lowers F-score of the framework. For a threshold of 60%, FPR decreases while FNR is still zero. In this case, accuracy is 97% and F-score is 80%. If the threshold is increased over 65%, it reduces the recall rate which affects accuracy and F-score. The evaluation indicates that the threshold value of 60% provides an accuracy of 97% and F-score of 80%. Also, From Figure 5.5, one can observe the relation between FPR and TPR of Naive Bayes-based detection. For FPR larger than 0.33, TPR becomes 1.

For Naive Bayes-based detection on the smartphone, we considered nine activities in total (three general activities and six user-related activities) [SAU17]. From Table 5.4, we can observe that TP rate FP rates decrease with the threshold value while FN and TN increase. When the threshold is 60%, the peak accuracy (95%)

and F-score (82%) are achieved for the smartphone. Precision-Recall curve for Naive Bayes model is given in Figure 7.8(b). We can notice that PRC curve is more irregular compared to Markov Chain-based approach. Calculated auPRC for Naive Bayes-based approach is 0.7 for the smart watch and the smartphone, both of which indicate less efficient method for imbalanced dataset.

5.6.5 Evaluation of Alternative Detection Techniques

In alternative detection techniques, we used other supervised machine learning techniques to train the 6thSense framework for both the smart watch and the smart phone. For this, we utilized WEKA and it provides three types of analysis - split percentage analysis, cross-validation analysis, and supplied test set analysis. We chose 10 fold cross-validation analysis to ensure that all the data was used for both training and test. Thus, the error rate of the predictive model would be minimized in the cross validation. In Table 5.5, a detailed evaluation of different machine learning algorithms is given for 6thSense. For *Rule Based Learning*, 6thSense has the best result for *PART* algorithm, which has an accuracy of 0.98 and F-score of 0.80. On the other hand, for *Regression Analysis*, we use the logistic function which has high FPR (0.65) and lower F-score (0.49). *Multilayer Perceptron* algorithm gives an accuracy of 0.9878 and F-score of 0.80, which is higher than previously mentioned algorithms. However, FPR is much higher (0.35), which is actually a limitation for intrusion detection frameworks in general. Compared to these algorithms, *Linear Model Tree (LMT)* gives better results in detecting sensor-based attacks. This evaluation indicates that *LMT* provides an accuracy of 0.99 and F-score of 0.972 for the smart watch.

From Table 5.5, one can also see performance of different machine learning algorithms in 6thSense on a smartphone. Here, LMT achieves the highest accuracy and F-score of 0.99 and 0.96, respectively. Multilayer Perception algorithm also performs well with F-score of 0.82. However, false positive rate is high in this algorithm which decreases the performance. In summary, LMT works efficiently in both the smart watch and the smart phone.

5.6.6 Comparison of Detection Methods

In this subsection, we give a comparison among the different machine learning-based detection approaches tested in 6thSense for defending against sensor-based threats on the smart watch and the smartphone. For all the approaches, we select the best possible case and report their performance metrics.

Table 5.6 depicts comparison among different detection approaches on the smart watch. For Markov Chain-based detection, we chose three consecutive malicious states as valid device conditions. On the other hand, in Naive Bayes approach, the best performance is observed for the threshold of 60%. For other machine learning algorithms tested via WEKA, we chose LMT as it gives highest accuracy among other machine learning algorithms. These results indicate that both Markov Chain and LMT provide high accuracy and F-score compared to the Naive Bayes-based approach.

On the contrary, Naive Bayes model displays higher recall rate and less FNR than other approaches. However, the presence of FPR in IDS is an issue to the system since FPR refers to a malicious attack that is identified as a valid device state. Both Markov Chain and LMT has lower FPR. Again, as our dataset is imbalanced (number of benign activities is higher than malicious activity), we chose auPRC as one of the performance metric of 6thSense. From Table 5.6 we can see that Markov Chain-based detection has the highest auPRC (0.926) followed by LMT (0.892) and Naive Bayes (0.646). In summary, considering F-score, accuracy, and auPRC of all three approaches, we conclude that Markov Chain and LMT both performs effectively for 6thSense.

In Table 5.6, we present a comparison of different machine learning-based detection techniques used in 6thSense on the smartphone. Again, we chose the best possible (Markov Chain, Naive Bayes, and LMT) cases for all of the approaches and compare them in Table 5.6. Similar to results in the smart watch, threshold for Markov Chain-based detection is three consecutive malicious state. For Naive Bayes-based detection, best performance can be observed for 60% threshold probability. From Table 5.6, we can observe that Markov Chain and LMT performs with high accuracy and F-score compared to Naive Bayes-based approach. Naive Bayes model also introduces high FP rate (0.3) which indicates poor performance for IDS. On the contrary, Markov Chain and LMT shows lower FP rate (0 and 0.0694 respectively). Again, from Figure 7.8(b), we can observe that Naive Bayes model has low auPRC compared to Markov Chain-based detection in Figure 5.5(b). LMT also has high auPRC (0.91) which is suitable for our imbalanced dataset. In summary, both Markov Chain and LMT performs well for 6thSense on the smart phone with high accuracy, F-score, and auPRC.

Performance Metrics	**Markov Chain (Smart Watch| Smart Phone)**	**Naive Bayes (Smart Watch| Smart Phone)**	**LMT (Smart Watch| Smart Phone)**
Recall rate	0.9770 | 0.98	1 | 1	0.9998 | 0..998
False Negative Rate	0.0230 | 0.02	0 | 0	0.0002 | 0.0002
Precision rate	0.9857 | 1	0.67 | 0.7	0.9458 | 0.9306
False positive rate	0.0143 | 0	0.33 | 0.3	0.0694 |
Accuracy	0.9795 | 0.9833	0.9720 | 0.9492	0.998 | 0.9997
F-Score	0.9813 | 0.9899	0.80 | 0.8235	0.972 | 0.964
auPRC	0.926 | 0.947	0.646 | 0.686	0.892 | 0.91

Table 5.6: Comparison of different machine-learning-based approaches proposed for 6thSense on Smartwatch and Smartphone (i.e., Markov Chain, Naive Bayes, and LMT).

5.6.7 Performance Overhead

As previously mentioned, 6thSense collects data in an Android device from different sensors (permission and no-permission imposed sensors). In this sub-section, we measure the performance overhead introduced by 6thSense on the tested Android devices (smart watch and smartphone) in terms of CPU usage, RAM usage, file size, and power consumption. Table 5.7, Table 5.8, and Table 5.9 give the details of the performance overhead of 6thSense on the smart watch and the smartphone.

For no permission-imposed sensors, the data collection phase logs all the values within a time interval which causes an increased usage of RAM, CPU and Disc compared to permission-imposed sensors. For the power consumption, we observe that no permission-imposed sensors use higher power than permission-imposed sensors. This is mainly because permission-imposed sensors are logic-oriented and have lower sampling rate, which reduces its resource needs. The overall performance overhead is

Parameters	**Time**	**No-permission imposed sensors (Smart Watch| Smart Phone)**	**Permission-imposed sensors (Smart Watch| Smart Phone)**
CPU Usage	N/A	5.5% | 3.9%	2.5% | 0.3%
RAM Usage (MB)	N/A	17 | 23	11 | 14
Disc Usage (MB)	1 min	4 | 6.5	0.001 | 0.001
	5 min	7.5 | 9	0.001 | 0.002
	10 min	10 | 12	0.001 | 0.003
Power Consumption (mW)	1 min	10.5 | 13.5	2 | 3.12
	5 min	45.6 | 96.67	16.5 | 27.4
	10 min	78.4 | 133.33	27 | 45
Power Consumption (without datafile)	1 min	1.32 | 2.68	0.1 | 0.23
	5 min	8.7 | 23.4	2 | 9.63
	10 min	32.56 | 55.35	9 | 17

Table 5.7: Performance overhead of data collection.

as low as 5.5% of CPU, less then 17MB RAM space, and less than 10MB disc space for the smart watch. Compared to the smart watch, performance overhead for the smartphone is higher because of higher number of sensors. Nevertheless, smartphone offers more resources (CPU speed, RAM size, disc size) than the smart watch which minimizes the effect of performance overhead. Performance overhead for the smartphone is as low as 3.9%, less than 6.5MB RAM space, and less than 12MB disc space. Thus, 6thSense's overhead is minimal and acceptable for an IDS system on current smart devices. One of the main concerns of implementing 6thSense on smart device is the power consumption.

Table 5.7 also shows the power consumption of the Android app used in 6thsense. For one minute, 6thsense consumes 10.5 mW power which increases upto 78.4 mW for ten minutes on a smartwatch. For a smartphone, 6thSense consumes upto 133.33 mW power for ten minutes. The main reason of this high power consumption is that all the sensors are kept on for the data collection and all the data are saved on device for later analysis. To mitigate power-performance trade-off, in practical settings, the data are not saved on device rather a real-time analysis is done, which indeed decreases the power consumption. Without saving the data, the power consumption significantly becomes smaller. From Table 5.7, we can observe that the power consumption of 6thSense becomes 32.56 mW which is almost 2 times lower than otherwise on a smartwatch. For real-time analysis, power consumption of 6thSense decreases 2.4 times on the smartphone. As all the sensors do not have to remain on for the analysis part, data can be observed if the smart device is in unlocked status to lower the power consumption.

Moreover, for the data analysis phase of 6thSense, we also implemented Markov Chain, Naive Bayes, and LMT-based detection methods on the Android smartphone and smart watch. Table 5.8 shows the performance overhead of different detection techniques used in 6thSense on a smart watch. All three detection techniques yield less than 2% CPU usage and 10 MB of RAM usage. Note that we consider the disc usage as a performance overhead for the data analysis phase since results can be

Parameters	**Markov Chain**	**Naive Bayes**	**LMT**
CPU Usage	1%	1.5%	1%
RAM Usage	6 MB	10 MB	10 MB
Disc Usage (For 5 Min)	<1 MB	<400 kB	<400 kB
Power Consumption (For 5 min)	1 mW	2 mW	2.5 mW

Table 5.8: Performance overhead of the data analysis phase in 6thSense on smart watch.

stored for further performance evaluation of the framework. Our extensive evaluation shows that the disc usage for the data analysis of 6thSense is less then 1 MB in all the three detection methods for 5 minutes of analysis. Table 5.8 also provides the power consumption of different detection techniques of 6thSense. We can observe that the power consumption of the data analysis phase is comparatively lower (less than 5 mW) than the data collection phase of 6thSense. Finally, Table 5.9 shows performance evaluation of different detection techniques of 6thSense on an Android operated smartphone. 6thSense performs with minimum overhead with less than 3% CPU usage, 17 MB RAM usage, and 2 MB of disc usage. Power consumption in the smartphone is also as low as 6 mW for different detection techniques implemented on 6thSense. In summary, different detection methods used in 6thSense offer lower performance overhead in the system.

Parameters	**Markov Chain**	**Naive Bayes**	**LMT**
CPU Usage	1.2%	2.5%	1%
RAM Usage	12 MB	15 MB	17 MB
Disc Usage (For 5 Min)	<2 MB	<1 MB	<1 MB
Power Consumption (For 5 min)	4.5 mW	6 mW	3 mW

Table 5.9: Performance overhead of the data analysis phase in 6thSense on smartphone.

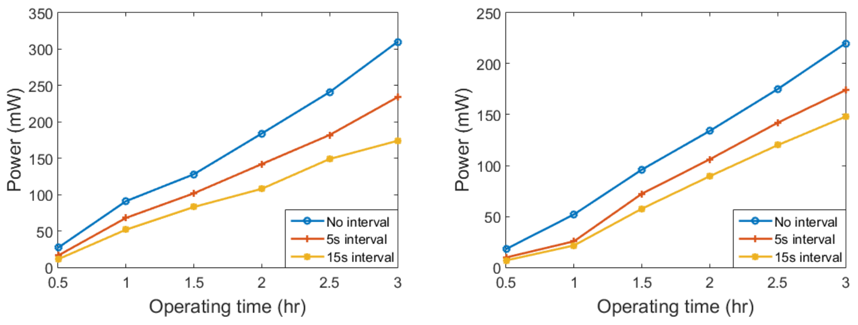
5.6.8 Power-efficiency Trade-off

One major concern of implementing a security framework in smart devices is power-efficiency trade-off. As smart devices such as smart watch and smart phone are resource constrained devices, an efficient security framework should work accurately with limited resources. 6thSense uses all the available sensors in a device to understand the state of the device and detects sensor-based threat based on state transition model. This can be a drawback in terms of power consumption of the device. To address this limitation, we performed a power-frequency trade-off study to determine the working condition of 6thSense in real-life settings. According to Nielsen, average American adult spends around 3 hours everyday on their smartphone [nie17].

We consider this as an average time that 6thSense has to run to detect any sensor-based threats in a smart device. In Figure 7.8(c) and 7.8(d), we illustrate the average power consumption graph for 6thSense with different scanning frequency in a smartphone and a smart watch, respectively. One can notice that 6thSense consumes 310 mW power for scanning continuously for 3 hours in a smart phone (Figure 7.8(c)). Average power consumption lowers to 234 and 174 mW with 5s and 15s time interval respectively. For smart watch, highest average power consumption for 6thSense is 220 mW for continuous scan. Average power consumption becomes as low as 174 mW and 148 mW for 5s and 15s time interval respectively.

5.7 Discussion

• *Sensor-based threats in real-life settings:* One limitation of 6thSense is the adversaries (sensor-based attacks) used in the evaluation were constructed in a lab-environment. Note that as of this writing there are no real sensor-based malware in the wild. However, many independent researchers have confirmed the feasibility of



(a) (b)

Figure 5.6: Power vs frequency graph for (a) smart watch (b) smartphone

sensor-based threats for smart devices [SPA+18]. Indeed, more recently, ICS-CERT also warned the vendors and the wider communities about the possibility of exploiting the sensors of a device to alter sensors' output in a controlled way to perform malicious behavior in the device [CA17]. Google also acknowledges the sensor-based threats by restricting sensor access for background apps in version of Android [and18b]. To understand the sensor-based threats and limitation of existing solutions, we built the proof-of-concept versions of the sensor-based threats discussed in Section 4. We also note that to ensure the reliability of the malware (i.e., specific malicious Apps) for the threats described in Section 4, we checked how they perform with respect to the real malicious software scanners. For this, we uploaded our malware on *VirusTotal* and tabulated the results of the performance of 60 different malware scanners available at the VirusTotal website in Table 5.10. As seen in this table, the sensor-based threats are not recognized by the different scanners. *In conclusion, current malware scanners are not aware of these threats yet and our malware can be reliably used to test the efficiency of 6thSense.*

- *Power monitoring app:* Different smartphone and smart watch vendors offer power monitoring apps which monitor running apps (both background and foreground apps) and minimize the power consumption of the device. For example, Samsung provides

Adversary Model	**Detection Ratio**
Threat-1	0/60
Threat-2	0/60
Threat-3	0/62

Table 5.10: *VirusTotal* scan result for the adversary models.

a power monitoring app that prevents background apps to drain power. Power monitoring apps activate sleep mode which disables the updates and notifications for the inactive apps. This conflicts with the malicious apps described in Section 7.1. However, the power monitoring app only works when the app is in the background. If a foreground app has malicious sensor logic, it can easily bypass the power monitoring app and initiate an attack. As power monitoring apps restrict important updates (e.g., messages from text apps, alarm apps, etc.), users can turn-off or modify this feature for convenience [pow17]. Moreover, the smart watches do not have any power monitoring option which makes them vulnerable to sensor-based threats. *In summary, power monitoring app can restrict sensor-based threats to some extent, but can not nullify them entirely.*

• *New OS feature:* Recently, Android introduced a new version of OS (Android P) which restricts camera and microphone usage if an app runs in the background. This feature certainly acknowledges the sensor-based threats and restricts sensor misuse in a smartphone. However, Android P only eliminates one threat model described in Section 4 and 7.1. Different malicious apps can still access other sensors in the background and perform multiple malicious activities. As explained in Section 7.1, Threat Model-1 uses motion and light sensors which does not have any conflict with Android P. Threat Model-2 uses the microphone of a connected smart watch which bypasses the security feature of Android P. Threat Model-3 triggers the camera of a smartphone if all the other sensors are inactive. Here, the malicious app opens the camera in the foreground which is allowed by Android P. We developed an updated version of this attack which could start recording via microphone in a smart watch if the connected smartphone was inactive and thus, bypass the security feature of Android P. Again, Android P only nullifies the threat if the app is installed in a smart phone. A malicious app installed in a smart watch can trigger the camera of a smart phone without any restriction. Also, only 1% of Android-powered devices support Android P currently which makes majority of the devices vulnerable to sensor-based threats using camera or microphone surreptitiously [and18a]. *In short, even with the introduction of the new OS, sensor-based threats can still affect normal operations of the smart devices.*

• *Optimum scanning frequency:* As smart devices are resource-constrained devices, an optimum scanning frequency is needed for 6thSense to lower the power consumption of the device. In Section 7.8, we illustrated that by scanning the sensors in fixed intervals (15s) and unlocked states, power consumption can be lowered by approximately 43%. However, some sensor-based threats can bypass the lock state and perform malicious activities in smart devices. To address this limitation, 6thSense can use the context-aware model to detect the lock state of the device and monitor limited sensors to minimize the power consumption. As Android P is restricting some sensors (microphone and camera), 6thSense can use this feature and select limited sensors to scan in the locked state. *In short, performance of 6thSense can be configured in terms of power consumption by selecting optimum scanning frequency and combining with existing permission model of OS.*

5.8 Conclusion

Wide utilization of sensor-rich smart devices created a new attack surface namely sensor-based attacks. Accelerometer, gyroscope, light, etc. sensors can be abused to steal and leak sensitive information or malicious Apps can be triggered via sensors. Security in current smart devices lacks appropriate defense mechanisms for such sensor-based threats. In this paper, we presented 6thSense, a novel context-aware task-oriented sensor-based attack detector for smart devices. We articulated problems in existing sensor management systems and different sensor-based threats for smart devices. Then, we presented the design of 6thSense to detect sensor-based attacks on sensor-rich smart devices (smartwatch and smartphone) with low-performance overhead. 6thSense utilized different machine learning (ML) techniques to distinguish malicious activities from benign activities on a device. To the best of our knowledge, 6thSense is the first comprehensive context-aware security solution against sensor-based threats. We evaluated 6thSense on real devices with 100 different individuals. 6thSense achieved over 97% of accuracy with different ML algorithms including Markov Chain, Naive Bayes, and LMT. We also evaluated 6thSense against three different sensor-based threats, i.e., information leakage, eavesdropping, and triggering a malware via sensors. The empirical evaluation revealed that 6thSense is highly effective and efficient at detecting sensor-based attacks while yielding minimal overhead.

CHAPTER 6

SENSOR-BASED THREAT DETECTION IN A CONNECTED SMART ENVIRONMENT

6.1 Introduction

In this chapter, we present Aegis, a novel platform-independent context-aware security framework to detect malicious sensor and app behavior in a connected smart environment (e.g., smart home, smart office, etc.). Aegis observes the changing patterns of the states (active/inactive) of smart sensors and devices for user activities in a smart environment and builds a contextual model to detect malicious activities. Here, context-awareness refers to the ability of Aegis to understand the changes in sensors' and devices' states due to ongoing user activities and determine if the behavior in the smart environment is benign or not. Smart devices are normally configured with different sensors to provide autonomous control and uninterrupted operations. Thus, different sensors in a connected smart environment can sense user activities (motion, opening doors, etc.) and trigger associated smart devices to perform pre-defined tasks. Aegis correlates these sensor-device relations with different user activities and builds a context-aware model to define benign user behavior. Aegis also uses smart app contexts to understand the trigger-action scenarios between installed smart devices and sensors and automatically upgrades the framework if new devices are added to the smart environment/platform. As a security framework, Aegis observes the current states (active or inactive) of sensors and devices and checks with the learned user behavior to detect any malicious behavior. Specifically, Aegis utilizes a Markov Chain-based machine learning technique to detect malicious behavior. Additionally, Aegis uses an action management system to alert the users in the event of malicious behavior and considers user responses to improve the context-aware model for better accuracy (adaptive training mode). We tested AEGIS in real-life smart home system (an example of connected smart environment) scenarios where 20 different users performed typical daily activities in three different home layouts generating over 85000 sensor-device correlated events. Furthermore, we considered different device settings (sensor-device relations), apps, smart device platforms (four different platforms including Samsung SmartThings, Philips HUE, LIFX Smart Light, and Amazon Alexa) and user policies to evaluate the performance of AEGIS against five different threats. Our extensive evaluation demonstrates that AEGIS can detect different threats to smart devices and sensors with high accuracy and F-score (over 95%). In addition, AEGIS yields minimum overhead in terms of latency and computing resource utilization, making this solution suitable for real-life deployment.

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

- **Aegis.** We present a novel context-aware security framework to detect malicious activities (both sensor and app activities) in connected smart environment. We capture sensor-device co-dependence in smart environment to understand the context of the user activity and detect malicious behavior. Additionally, we implemented an action management system to alert users about AEGIS's findings.

- **User-specific configurations.** We designed AEGIS to support different smart environment layouts (e.g., three different smart home layouts) and configurations. AEGIS allows easy integration of new devices and apps creating app contexts and reconfiguring the training data automatically. We also introduced an adaptive training model to automatically improve the detection mechanism from user responses.

- **Real-life implementation.** We implemented AEGIS in *Samsung SmartThings* platform and performed detailed evaluation analysis with real user data.

• **High accuracy and minimal overhead.** Through a detailed evaluation, we demonstrated how Aegis can detect different malicious activities in a smart environment. Our results show that Aegis can achieve high accuracy and F-score and impose minimum overhead in the system.

6.2 Differences from Existing Works

The main differences between Aegis and other existing solutions (although they are useful) can be articulated as follows. (1) While other solutions focus on securing shared data and improving current user permission system [JCW+17], Aegis detects malicious behaviors by considering user and device activity contexts in a connected smart environment. (2) Aegis considers both smart device settings, platform configurations, and installed apps to build a context-aware model and detect threats at run-time which outdo user-dependent; solutions [JCW+17]. (3) Additionally, no source code modification [BSAU18] is needed for Aegis to collect data from smart devices and detect malicious activities in a smart environment. (4) Unlike threat-specific existing solutions [CBS+18, WHBG18], Aegis can detect five different types of threat in a connected smart environment which makes it a more robust solution. (5) Aegis provides a platform-independent solution as it only considers user activity to build the context-aware model irrespective of smart systems/platforms, specific devices, and development platforms, (6) Finally, Aegis collects data from a common access point and performs behavior analysis at run-time which reduces cost in terms of processing and overhead from other prior works [BSAU18, CBS+18]. In addition, Aegis does not store usage data from smart devices which reduces the privacy risks and concerns from prior solutions [BSAU18]. Table 6.1 summarizes the differences of Aegis and other existing solutions.

In summary, Aegis *offers a context-aware security framework which uses behavior analysis, usage patterns, and app context to detect malicious activities at run-time and ensures security against five different threats to smart environment with high accuracy and minimal overhead.*

6.3 Design Features Considered in AEGIS

To design Aegis, we considered several features of connected smart environment to correctly capture sensor-device correlation based on on-going user activities and build a contextual model. In this section, we explain the design features considered in Aegis in details.

6.3.1 Context-awareness.

Context-awareness refers to the ability of a system to use situational and environmental information about the user, location, and devices to adapt its operation accordingly [SAU17, JCW+17]. In a connected smart environment (e.g., smart home system), all the sensors and devices follow different trigger-action scenarios to perform tasks. Here, sensors are used to provide input in the devices (trigger) and devices take autonomous decisions (actions) based on these inputs. When a user performs a task in a smart environment, several smart sensors and devices may become active in a sequential pattern. The pattern of active devices and sensors is different but specific for distinct user activities. Existing smart devices and platforms cannot observe these patterns in sensors' and devices' states over time and can not understand the context of the user activity. For example, while a user moves from one bedroom to a hallway, several devices and sensors become active in a sequential manner (Figure 6.1): moving towards bedroom door (sub-context 1: BL1, BLi1, BM1 are active), bedroom

Prior Work	Context Aware	Platform independent	Diverse generalized threats	No source code modification	User permission independent	Active notification	Run-time detection	Real-life implementation
ContextIoT [JCW+17]	●	○	●	○	○	●	●	●
IoTSAT [MAH+16]	○	●	○	●	●	○	○	○
iRuler [WDY+19]	○	○	○	○	●	●	●	●
IoTWatch [BCMU19]	○	○	●	○	●	●	●	●
Network IDS [YOM+19]	○	●	○	●	○	○	●	○
Supervised IDS [AWS+19a]	○	●	○	●	●	○	●	○
SaINT [CBS+18]	○	●	●	●	●	○	○	●
HomeGuard [CZDY18]	○	●	○	○	●	○	●	●
ProveThings [WHBG18]	○	○	●	○	●	○	○	●
IoTDots [CBS+18]	○	○	●	○	●	○	○	●
Aegis	●	●	●	●	●	●	●	●

Table 6.1: Comparison between Aegis and other existing security framework for smart environment.

door opens (sub-context 2: BL1, BLi1, BM1, BD1 are active), entering the hallway (sub-context 3: BL1, BLi1, BD1, HLi2, HL2, HM2 are active), bedroom door and light close and reaches the hallway (sub-context 4: HLi2, HL2, HM2 are active). To complete the activity (moving from the bedroom to the hallway), the user must follow the sub-contexts in the same sequential pattern. The user cannot skip one specific sub-context and move to the next one to complete the activity. For instance, the transition from sub-context 1 to sub-context 4 is not possible as a user cannot go to the hallway from the bedroom without opening the door. Motivated by this, Aegis is designed to understand this property of connected smart environment to build a context-aware model for different user activities and usage patterns and differentiates between benign and malicious activities of smart devices and sensors.

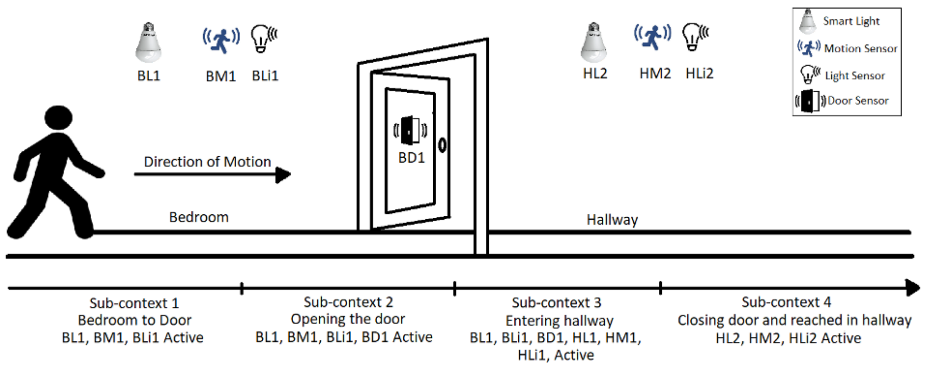


Figure 6.1: Context-awareness feature, which is not considered in existing solutions to protect smart environment against cyber attacks.

6.3.2 Sensor-device Co-dependence

In a smart environment, sensors, and devices can be configured as independent entities. However, they work in a co-dependent manner to provide autonomous functionalities. For instance, smart lights can be configured with motion sensors to light up when motion is sensed. Here, the smart light depends on the input from the motion sensor while the motion sensor alone cannot provide any significant function in a connected smart environment. The functions of devices and sensors create a co-dependent relationship with each other. In this way, sensors and devices in the smart environment can build many-to-many co-dependent relationships. However, existing smart environment/platforms do not consider this co-dependent relationship and can not visualize the context of a user activity by observing the usage pattern of smart devices and sensors. In short, sensors and devices in a connected smart environment are configured as independent components, but in reality they are function-wise co-dependent. Aegis considers these relations to build the context of the user activities in a smart environment.

User Activity-device Correlation. In a connected smart environment, different users utilize and control smart devices in multiple ways. For instance, a user can set a security camera to take pictures whenever a motion is detected in the associated motion sensors. On the other hand, users control devices in multiple ways. For example, a user can unlock a door by using the smartphone app or entering the code manually. Here, the state of the lock can be determined by user activity on the smartphone or by using a presence sensor to detect the user near the smart lock. In short, by observing the user activities in a smart environment, it is possible to determine the normal operation of smart devices. One can define normal or malicious user behavior with the user activity-device correlation. Current smart platforms cannot correlate user activity and device actions correctly, which is considered as a feature in Aegis to differentiate benign and malicious activities.

6.3.3 Multi-platform Correlation.

Modern smart systems (e.g., smart home system) allow users to install smart devices from different vendors within the same physical environment. These installed smart devices can perform autonomous tasks collectively via a common hub or as standalone devices connected to the Internet via an access point (e.g., Wi-Fi router). However, any on-going task in a smart device can be perceived by other standalone smart devices, even if they are not interconnected via a hub. For instance, if a user gives a voice command to smart speaker to open the main door, a presence sensor connected to a smart light can verify the presence of the user inside the smart environment and confirm the user command as a valid input in the smart devices and sensors. Here, the presence sensor can verify a benign user activity even if it is not connected to the smart speaker. Hence, smart devices from different smart platforms are correlated and can capture user activity context properly. In short, sensors and devices from multiple smart platforms that are configured as standalone devices can be correlated via user activity context. Existing smart systems do not consider these correlations between standalone devices from multiple platforms, which is considered as a feature in Aegis to build user activity context in a multi-platform smart environment.

6.4 Problem Scope and Threat Model

In this section, we introduce the problem scope and articulate the threat model considered in Aegis.

6.4.1 Problem Scope

This work assumes a fully automated smart home (an example of connected smart environment), S, with several smart devices and sensors, which is illustrated in Figure 2.2. The smart home includes smart light, smart smoke detector, smart locks, smart thermostats, smart speakers, motion sensors, smoke sensors, light sensors, presence sensors, and temperature sensors. Here, the following sensor-device triggering rules are configured - the smart lights are configured with motion sensors, the smart smoke detector is configured with smoke sensor. Additionally, The smart home system allows manual device control by the users (e.g., unlocking smart lock with PIN). We also assume that the user utilizes customized third-party apps to control the devices. Furthermore, the smart home system has more than one user authorized to control the devices in the system. Authorized users in the smart home systems can control the smart devices either via using controller devices such as smartphone or by using voice commands via the installed smart speaker in the system. We assume the following trigger-action incidents are happening throughout the day in the smart home - (1) one user is walking inside the bedroom but the lights are not triggered by the motion sensors, (2) one user is trying to unlock the smart lock using PIN code, (3) a fire alarm installed in the kitchen is being triggered in the system, (4) a smart light in the guest bedroom executes a blinking pattern while no user is present, (5) a user is trying to change the temperature of the smart thermostat but the temperature is not changing, (6) the smart speaker installed in living room executes a voice command to turn on the smart TV even though no user is present in the room?

We propose Aegis as a novel security framework that builds a context-aware model based on user activities to determine benign and malicious incidents in the smart environment (in this case, smart home system). Aegis answers several questions that may arise from the above-mentioned incidents - (1) What is the reason for no activity in the smart light installed in the bedroom?, (2) Is a legitimate user or an attacker is trying to unlock the door using PIN code?, (3) Is the fire alarm being triggered by a malicious app?, (4) What caused the smart light to blink and what is the intent of this activity?, (5) Why the user is not able to change the temperature of the smart thermostat?, (6) Why the smart speaker is executing a voice command without the presence of any user in the room? Aegis differentiates between normal and malicious activities happening in a connected smart environment. Furthermore, Aegis detects

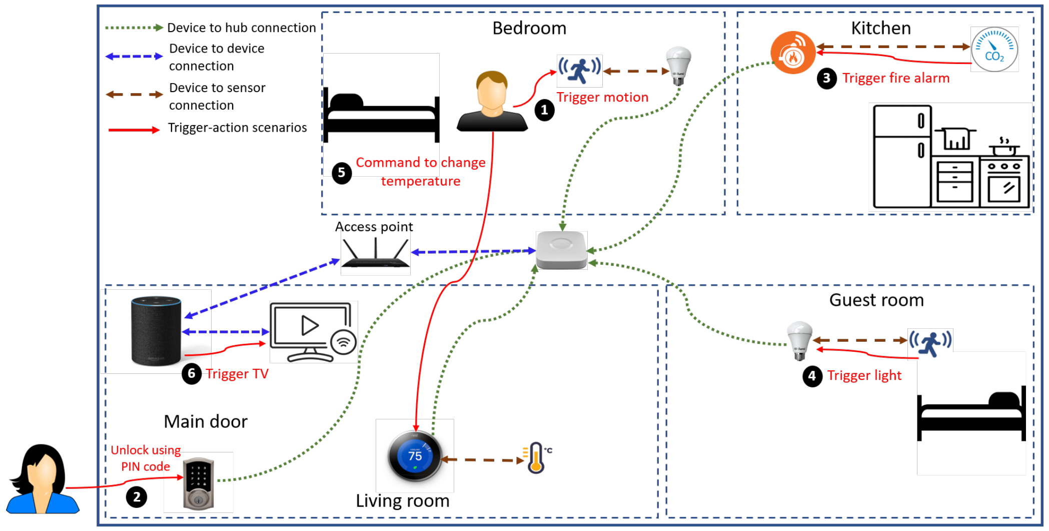


Figure 6.2: Sample smart home/environment with multiple users generating multiple trigger-action scenarios.

malicious activities occurring in a device or sensor by observing the ongoing activities of all the connected devices in the smart environment.

6.4.2 Threat Model

Aegis considers *anomalous user behaviors* (e.g, unauthorized users changing the device states) that may disrupt the normal functionalities of the smart environment. Also, device vulnerabilities that may cause device malfunction or open doors to threats like impersonation attacks and false data injection attacks are considered by Aegis. Additionally, this work assumes carelessly-designed and malicious smart applications that may cause unauthorized or malicious activities in the smart environment. These malicious activities may facilitate side channel and denial-of-service (DoS) attacks. Moreover, Aegis considers threats arising from malicious user commands on controller device such as smart speaker. To better capture the threat model, we classify it in the following five categories:

- *Threat-1: Impersonation Attack.* An unauthorized user can try to get access to smart devices or applications by stealing valid user credentials or recording legitimate voice commands using a malicious app.

- *Threat-2: False Data Injection.* A malicious smart app can exist in the smart system and inject forged data (e.g., sensor data, voice commands, etc.) to perform malicious activities. This threat represents false data injection in smart devices and sensors.

- *Threat-3: Side channel attack.* A carelessly developed smart app with design imperfections installed in the smart environment/setting can perform legitimate, yet vulnerable side-channel activities which can be harnessed by other apps (considered malicious) in the system or the attacker himself. This threat represents a side channel attack on smart devices and sensors.

- *Threat-4: Denial-of-service.* A malicious smart app installed in the system can impede normal behavior of other smart devices, sensors, and applications. This threat represents a denial-of-service attack in a connected smart environment.

- *Threat-5: Triggering a malicious app.* A malicious smart app can exist in the system which can be triggered by a specific activity pattern or device action (e.g., switching a smart light in a specific on/off pattern) in a smart environment.

In Section 6.6, we present specific examples of attack scenarios that are used later to evaluate the effectiveness of Aegis. The information leakage caused by a compromised device or untrusted communication channel in the smart environment/setting are considered out of scope of Aegis. We also assume that the data collected from the devices and central management system (e.g., Hub, cloud, etc.) is not compromised.

6.5 AEGIS Framework

Aegis has four main modules: (1) data collector, (2) context generator, (3) data analysis, and (4) action management, as depicted in Figure 6.3. First, Aegis collects data from smart sensors and devices installed in the environment from day-to-day user activities. The *data collector module* uses smart apps to collect all the device and sensor states (active/inactive). Additionally, this module is fed from smart app rules extracted from different smart apps using the *app rule extractor* and that are stored in the multi-platform rule repository. The device state data is used to understand the context of the user activities and feeds the *context generation module.* This module creates context arrays generated from usage patterns and the predetermined user policies in the smart apps. Each context array contains overall information of the user activities and device states in the connected smart environment.

The context arrays generated in the context generation module are used by the *anomaly detector module* to implement machine learning-based analysis and build the context-aware model of the smart environment. Additionally, *anomaly detector module* decides whether or not malicious activities occur in the smart environmen-t/setting. Finally, the malicious activities detected by the anomaly detector module are forwarded to the *action management module.* This module notifies the users regarding the unauthorized device and sensor activities. Also, it offers adaptive training mode where users can validate any false positive or false negative occurrence and re-train the detection model to improve the performance of Aegis.

6.5.1 Data Collector Module

The data collector module has two sub-modules: *device data collector* and *app data collector.*

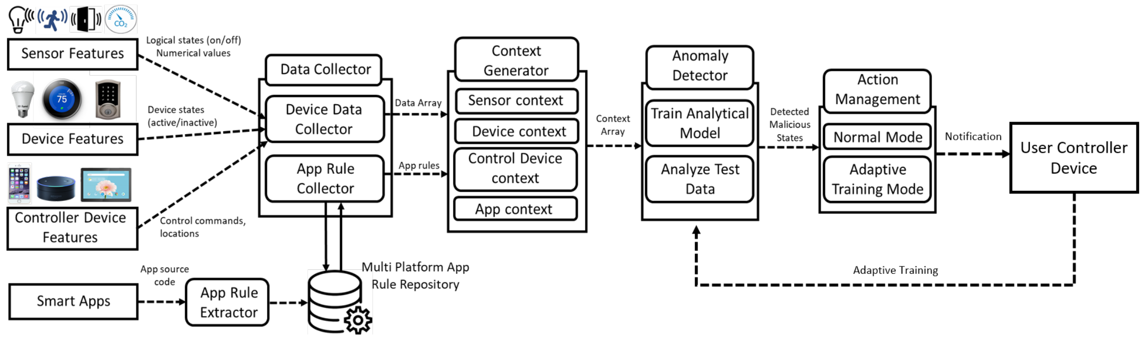


Figure 6.3: Aegis framework for multi-platform smart environment.

Device Data Collector

Aegis collects data from smart devices and sensors using the data collector module. In a smart environment, there can be multiple devices and sensors connected through a hub and operating in a co-dependent manner. Additionally, devices and sensors can act as standalone smart device and perform different automated tasks individually. The data collector module of Aegis collects the state (active or inactive) of these devices autonomously for both connected and standalone smart devices and forwards these data to the context generation module. Based on the type of data, the collected data is governed by:

$$\textit{Data array},\ E = \{S, D, M\}, \tag{6.1}$$

where S is the set of features extracted from the sensors, D is the set of features extracted from the devices, and M is the features extracted from the associated controller devices (e.g., smartphone, smart tablet, smart speakers) in a smart environment. We describe the characteristics of these features below.

- *Features extracted from sensors (S):* A smart environment such as smart home can comprise several sensors such as motion and light sensors. They sense changes in the vicinity of the devices and work as input to multiple devices. Sensor data can be both logical states (e.g., motion sensor) and numerical values (light sensor). For Aegis, we consider both logical states and numerical values of sensors to create the context of user activities.

- *Features extracted from devices (D):* In a smart environment, several devices can be connected with each other and with different sensors or act as standalone device. These devices remain active based on user activities in a smart environment. Aegis observes the daily activities of users and collects the device state data (active/inactive state) to build the context of the associated activity.

- *Features extracted from controller devices (M):* In a smart environment, Smartphone or tablet works as a control device to the smart system and users can control any installed device using the associated smart app. Additionally, smart speakers such Amazon Alexa, etc. can be used as a controller device by allowing users to control installed smart devices using voice commands. Aegis considers any control command given from the controller device as a feature to understand the context of user activity. The location of the connected controller device can also work as an input to control multiple devices. For instance, a thermostat can be configured to the desired temperature whenever the smartphone of the user is connected to the smart home network. Aegis considers the location of the controller device as a feature to build the context of user activities.

As user activities on a connected smart environment can vary based on the number of users, Aegis considers multi-user settings to understand the user activity contexts correctly. Moreover, user activities also change based on the daily routine of users. For this, in the data collection process, Aegis also offers time-based activity settings (weekday and weekend settings).

App Rule Extractor

Modern smart device platforms offer an app-centric model where users install different apps to automate the functionality of smart devices. These apps mostly define trigger-action scenarios for specific devices. For instance, an app can automate a smart light by configuring it with a motion sensor or a light sensor. Here, the sensors work as a trigger and the state of the smart light (on/off) refers to the action. These trigger-action scenarios can represent the app context which can be used to validate the user activity context in a smart environment. Additionally, the app context is also used to train the analytical model for new devices in smart environment. Aegis utilizes source code analysis to identify and extract relevant app information related to these trigger-action scenarios. In this work, we assume that the source code of the smart apps is freely available for analysis. We consider this assumption realistic as some of the most popular smart device platforms implement their apps as open-source (e.g., Samsung SmartThings, openHAB, Windows IoT, and AWS IoT) [Sam18b, Mic20, Ope]. Aegis implements a logic extractor to collect the smart app logic and infer the app context. The logic extractor takes the source of the apps and generates its Asymmetric Syntax Tree (AST). With the AST, the app rule extractor implements the Inter-Procedural Control (ICFG). The ICFG contains the nodes that define the different trigger-action events that are of interest of Aegis [CBS+18]. Further, the rule extractor visits every single node of the ICFG and collects the trigger-action scenarios that are defined in every app. For example, if a smart light is configured with a contact sensor, Aegis extracts the following logic from the app. In Section 6.6, we provide the implementation details of the app rule extractor in Aegis. Finally, as we found most smart apps that feature the same type of devices define similar trigger-action scenarios, we define an app rule extractor module that is able to analyze apps in parallel to Aegis's analysis. With this, we can collect and analyze apps from different platforms and create a multi-platform app rule repository that further feeds the data collector module previously described. There are several advantages to this design approach. First, it permits building a corpus of trigger-action scenarios that include information from several apps from multiple platforms simultaneously. Second, it prevents the need of updating Aegis's framework every time new apps and devices are included in the smart environment. Finally, it reduces the overhead introduces by Aegis as smart apps are mostly analyzed prior to Aegis's execution.

Sample Smart App for App Context One of the features of Aegis is using app context to verify the device states in the smart environment. To build the app context, we used similar static analysis approaches used in prior works [CBS+18, CZDY18]. We performed a source-to-sink taint analysis similar to [CBS+18] to extract the app context. Additionally, we consider the sources for smart apps proposed in [CZDY18, JCW+17]. We then built the abstract syntax tree (AST) and model a trigger-action scenario of an app. We tracked the *Subscribe* method to represent the trigger and follow the conditional statement (e.g., if and switch) to reach the sink. This flow from entry point (source) to a sink is used to construct the condition of an app which is then represented into app context. We collected 485 official *Samsung SmartThings* apps (available in their website) and created the app context database using this method.

Listing 6.1: A code snippet of a sample smart app

```
/* This is a sample smart light app for Samsung SmartThings */
definition(
    name: "Smart Light App",
    namespace: "smartthings",
    author: "anonymous",
    description: "Turn lights on when door is open.",
    category: "Convenience",
)
preferences {
        section("When the door opens/closes...") {
                input "contact1", "capability.contactSensor", title: "Where?"
        }
        section("Turn on/off a light...") {
                input "light1", "capability.light"
        }
}
def installed() {
        subscribe(contact1, "contact", contactHandler)
}
```

```
def updated() {
        unsubscribe()
        subscribe(contact1, "contact", contactHandler)
}
def contactHandler(event) {
        if (event.value == "open") {
                light1.on()
        } else if (event.value == "closed") {
                light1.off()
        }
}
```

Listing 6.2: Trigger-action scenarios extracted from a sample app.

```
Trigger: Contact1
Action: Switch1
Logic 1: contact1 = on, light1 = on
Logic 2: contact1 = off, light1 = off
```

6.5.2 Context Generation Module

The data collector module forwards the collected data to the context generation module to build the context of different user activities. Then, the context generation module maps and aggregates the data to build context arrays. Each context array consists of information on the usage patterns in the smart environment for different activities, which can be used for further analysis and to determine malicious activities in the system. The context array modeling process has the following steps:

- *Context of sensors:* Sensor features collected in the data collector consists of both logic state (on/off) and numerical values. Aegis observes the sensor data and generates the conditions of the sensors. These conditions represent the changing pattern of the sensor. If the current sensor value is different than the previous one, Aegis considers this as an active condition that is represented as 1. Similarly, conditions labeled as inactive are represented as 0.

- *Context of devices:* Data collector of Aegis collects device state (active/inactive) data for every connected device. These device state data are converted to logical states (1 represents active and 0 represents inactive) to build the context of user activities in a smart environment.

- *Context of controller devices:* There are two features of the controller device (e.g., smartphone, tablet, speakers, etc.) that are collected by AEGIS: Control command for the devices and the location of the controller device. For any command from the smartphone, tablet, or smart speakers, AEGIS considers the active condition of controller device which is represented as 1 in context array or 0 otherwise. A smart environment/setting allows two different states to represent the location of the controller device - *home* and *away.* Home location indicates that the controller device is connected to the home network and away represents otherwise. AEGIS represents the "home" location of the smartphone as 1 and the "away" location as 0 in the context array.

The final context array can be represented as follows:

$$Context\ Array,\ C = [\{S_1, S_2, ..., S_X\}, \{D_1, D_2, ..., D_Y\}, \{M_1, M_2\}], \tag{6.2}$$

where $S_1, S_2, ..., S_X$ captures the conditions of X number of sensors in the system, $D_1, D_2, ..., D_Y$ the conditions of Y number of sensors in the system, and M_1, M_2 the conditions of smartphone/tablet in the system.

Context generation module also generates the app's context. As most of the app's logic represents a trigger-action scenario, the context generation module converts the logic in a binary representation. For example, the logic extracted from the app presented in Listing 1 is given below:

Listing 6.3: Generated app context of a sample app

```
App Context 1: contact1 = 1 , Light1 = 1
App context 2: contact1 = 0, Light1 = 0
```

Here, for the contact sensor, 1 and 0 represent the contact state from "open" or "close" respectively. Similarly, for the smart light, 1 and 0 represent the light state from "on" or "off", respectively. These app contexts are used to validate the sensor-device co-dependence captured in the context array. Additionally, these app contexts are used to update the training dataset whenever a new device is added to the smart environment.

Finally, as noted before, we found that the app context is highly-depended on the specific type of devices and sensors the smart app controls. For instance, an app controlling a smart thermostat is likely to contain trigger-actions related to `Temperature = high, Thermostat = on`. Similarly, if the smart app controls a smart light and a motion sensor, the trigger-action `Motion = on, Light = on`. We use this finding to group together apps with similar expected trigger-action scenarios and improve the practicality of Aegis. More specifically, we consider the devices that every specific app controls to infer expected trigger-action scenarios. With this, we found that Aegis does not need to consider and analyze every new app in the market to maintain the context generation module up-to-date. This finding considerably increases the usability of Aegis as incorporating new apps to control the smart devices and sensors would not limit the effectiveness of the context generation module.

6.5.3 Anomaly Detector Module

In the third module, Aegis takes context arrays generated in the context generation module as input and trains a Markov Chain-based machine learning model which is used to detect malicious activities in connected smart environment.

The Markov Chain model is based on two main assumptions: (1) the probability of occurring a state at time $t + 1$ only depends on the state at time t only and (2) the transition between two consecutive states is independent of time. Aegis uses this Markov Chain model to illustrate a series of events in a smart environment. Here, a series of events denotes user activity and usage pattern and the state represents the context array at a specific time t generated in the context generation module. The probabilistic condition of Markov Chain model is shown in Equation 6.3, where X_t

denotes the state at time t for a user activity in the smart environment [Kei12].

$$\begin{aligned} P(X_{t+1} = x | X_1 = x_1, X_2 = x_2 ..., X_t = x_t) = P(X_{t+1} = x | X_t = x_t), \\ when,\ P(X_1 = x_1, X_2 = x_2 ..., X_t = x_t) > 0 \end{aligned} \tag{6.3}$$

Aegis considers the context array given in Equation 6.2 as an array of variables and observes its changes over time. For every user activity on a connected smart environment, several context arrays are created. These arrays follow a different but specific pattern for different user activities. Each element of the context array represents the condition of a smart entity (active/inactive states of sensor, device, or smartphone). For a distinct time, t, we consider the combination of all the smart devices' and sensors' condition as binary output (1 for the active state of an entity and 0 for the inactive state). Thus, the number of total states (A) will be the exponent of 2 and can be represented as a n-bit binary number, where n is the total number of entities in the smart environment. Let assume P_{ij} denotes the transition probability of the system from state i at time t to state j at time $t + 1$. If the smart system/setting has n number of entities (sensors, devices, and controllers) and $m = 2^n$ states in the system, the Markov Chain model of Aegis can be illustrated as in Figure 6.4. Here, each transition probability from one state to another state represents an element of the transition matrix. The transition probability matrix of the Markov Chain model constructed from context arrays can be represented by the following equation:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & \cdots & p_{1m} \\ p_{21} & p_{22} & p_{23} & \cdots & \cdots & p_{2m} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ p_{m1} & p_{m2} & p_{m3} & \cdots & \cdots & p_{mm} \end{bmatrix} \tag{6.4}$$

If the smart environment/system has $X_0, X_1, ..., X_T$ states at a given time $t = [0, T]$, the elements of the transition matrix can be shown as $P_{ij} = \frac{N_{ij}}{N_i}$, where N_{ij} denotes the number of transition from X_t to X_{t+1}, X_t is the state at time t, and X_{t+1} is the state at time $t+1$ [Y$^+$00]. If we represent Q as the initial state at time t = 0, the initial probability distribution of the system can be represented by the following equation:

$$Q = \begin{bmatrix} q_1 & q_2 & q_3 & \cdots & \cdots & q_m, \end{bmatrix} \tag{6.5}$$

where q_m is the probability that the model is in state m at time 0. Probability of observing a sequence of states $X_1, X_2, \ldots, X_T$ at a given time $1, \ldots, T$ in the smart environment can be computed using the following equation:

$$P(X_1, X_2, \ldots, X_T) = q_{x1} \prod_{2}^{T} P_{X_{t-1}Xt}. \tag{6.6}$$

As the number of states in the smart environment depends on the total number of installed devices and sensors, predicting the probability of the next state can introduce overhead in terms of computing time and resource usage. Instead of predicting the next state using the Markov Chain model, Aegis determines the probability of transition between two states in the smart environment at a given time. We train the Markov Chain model with the generated context arrays from the context generation module and construct the transition matrix. Using this transition matrix, Aegis determines the probability of transition from one state (i.e., context array) to another state over time. For example, in Figure 6.1, the transition between sub-context 1 and sub-context 2 is valid as the user can perform this activity. However, the transition from sub-context 1 to sub-context 4 is invalid as the user cannot go from the bedroom to the hallway without going through sub-context 2 and 3. Hence, the transition probability matrix can manifest the state transition based on user activity contexts. Let us assume a and b are system's overall state in time t and *t+1*. We determine the probability of transition from state a to b which can be found by looking up in the transition matrix and calculating *P(a,b)*. As users cannot skip any sub-context

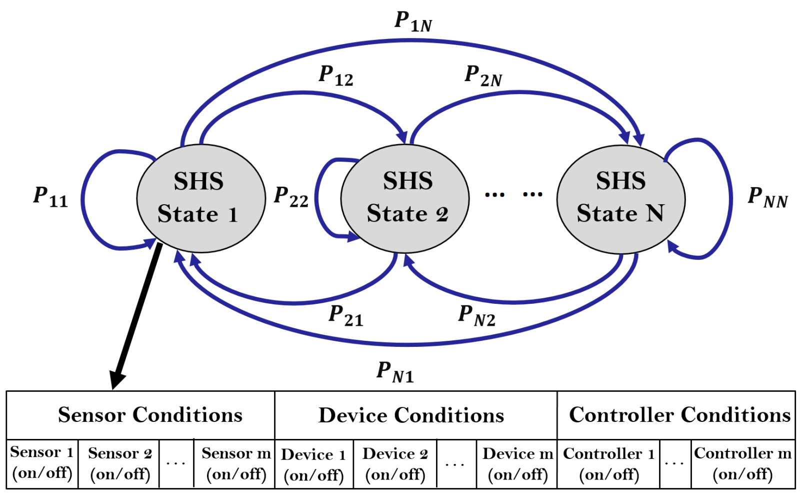


Figure 6.4: Markov Chain model for Aegis.

of an activity, *P(a,b)* will result in zero if transition from state *a* to *b* is malicious. Thus, Aegis defines benign or malicious device behavior based on user activities.

6.5.4 Action Management Module

Finally, the action management module notifies the users in the event of malicious activity in the smart environment. Action management module has two operation modes - detection mode and adaptive training mode.

• *Detection mode:* In the detection mode, Aegis pushes a notification in the controller device's user interface (smartphone, smart tablet) to notify the users if malicious activity is detected. Aegis provides the device ID and the installed app names to the user for further action.

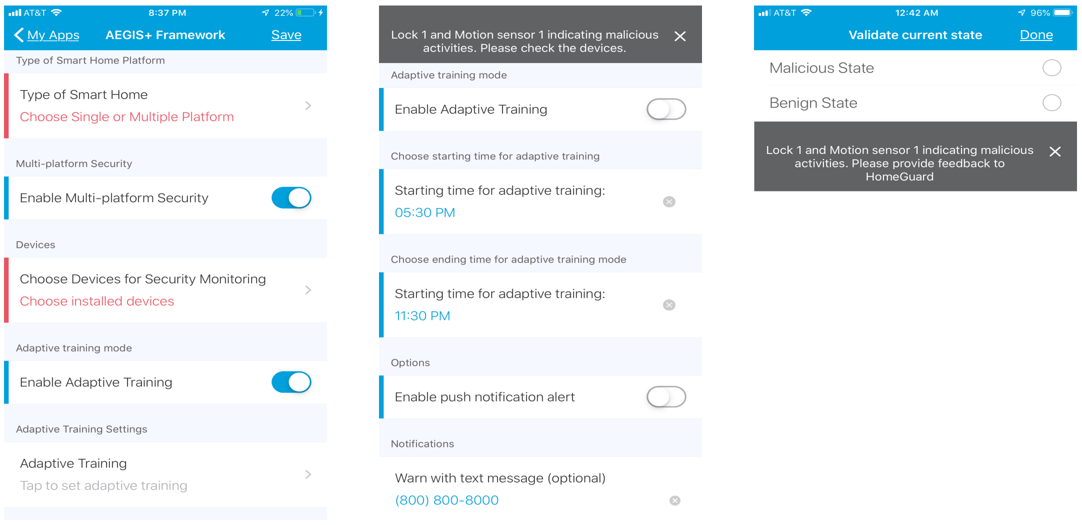
• *Adaptive training mode:* As Aegis builds a contextual model from user activities, it is important to verify the correct context of an ongoing user activity [PZCG14]. In a smart environment, users can perform different activities in an irregular pattern. For example, a guest may come to the house which will introduce some new activity patterns in the system. These abrupt data patterns may cause a higher false positive rate in the contextual model. To address this issue, Aegis offers the adaptive training mode, a user feedback process to improve the performance. In the adaptive training mode, whenever a malicious activity is detected, Aegis sends a notification to the controller device's user interface (smartphone, smart tablet) for user confirmation. Users can either confirm the malicious activity from the controller device or mark the activity as benign. If the user confirms the activity as benign, Aegis label that activity context and train the framework automatically. Hence, Aegis can correctly and automatically improve the training dataset by adding irregular or new user activity pattern.

6.6 AEGIS Implementation

We developed Aegis as a multi-platform security framework for smart environment. To implement and test the effectiveness of Aegis in real-life smart environment, we chose several smart home platforms and devices including Samsung SmartThings, Amazon Alexa, Google Home, etc. We built a real-life smart home environment which represents the multi-platform connected smart environment. In this section, we provide details of Aegis's design considerations and implementation steps in the following subsections.

6.6.1 Design Features and Goals

Existing smart platforms offer diverse sets of devices and sensors to automate day-to-day activities. However, configurations of smart devices and sensors depend on users' demands, device choices, home layout, home occupancy, preferences, and social relationships. As Aegis builds the context-aware model based on data collected for user activities, we consider following design features and goals. **Layouts.** Smart


AT&T 8:37 PM 22%

My Apps | AEGIS+ Framework | Save

Type of Smart Home Platform

Type of Smart Home
Choose Single or Multiple Platform

Multi-platform Security

Enable Multi-platform Security

Devices

Choose Devices for Security Monitoring
Choose installed devices

Adaptive training mode

Enable Adaptive Training

Adaptive Training Settings

Adaptive Training
Tap to set adaptive training

Lock 1 and Motion sensor 1 indicating malicious activities. Please check the devices.

Adaptive training mode

Enable Adaptive Training

Choose starting time for adaptive training

Starting time for adaptive training:
05:30 PM

Choose ending time for adaptive training mode

Starting time for adaptive training:
11:30 PM

Options

Enable push notification alert

Notifications

Warn with text message (optional)
(800) 800-8000

AT&T 12:42 AM 96%

Validate current state | Done

Malicious State

Benign State

Lock 1 and Motion sensor 1 indicating malicious activities. Please provide feedback to HomeGuard

(a) AEGIS interface (b) Normal Mode (c) Adaptive Training Mode

Figure 6.5: User interfaces of AEGIS in different operation modes.

environment such as smart home, smart office, etc. vary based on the layout as number of devices, sensors, and apps differ in different layouts. To test the effectiveness of AEGIS, we selected three different smart home layouts to represent connected smart environment: single bedroom apartment, double bedroom home, and duplex home. We selected these three layouts as these are the most common rental units in USA [fHSoHU20].

Occupancy. Smart environment is typically a multi-user environment where users share the installed smart devices and sensors. As smart environment usually have more than one occupants, we considered several multi-user environments (two, three, and four users in same layout) to implement AEGIS in real-life.

Types of devices and sensors. As smart device and sensor configuration depends on users' personal needs and preferences, the types of devices and sensors installed in the smart environment vary in different layouts. Since, AEGIS offers a platform-independent security framework, we considered 14 different types of off-the-shelf devices to build a real-life smart environment.

Date- and Time-dependent Activity. User activity in a smart setting depends on the user's daily routine which may change for different days of the week. For example, a working adult may spend more time at home on the weekends than weekdays which increases user interaction in smart devices and sensors. We considered this while designing Aegis and implementing in real-life environment.

User-specific rules. Current smart device platforms allow users to build and define multiple policies to control smart devices. The context of user activities may change based on user-defined policies in a smart environment. For example, a smart light can be controlled via the motion, door, or presence sensor. To understand the event associated with the light sensor and build the context of user activity, one must understand the user-defined policy enforced in the smart light. We addressed this property in the implementation of Aegis by allowing users to define their own policies in the smart devices and sensors in the data collection process.

Platform Independence. Smart environment allow users to install devices from different smart device platforms in same physical environment. These devices can be connected with each other via hub or perform tasks as standalone devices. Aegis offers platform independence by allowing users to install smart devices from different platforms in the same layout. Also, we considered both hub-connected and standalone smart devices in the data collection and contextual model of Aegis.

6.6.2 Implementation

To test the efficacy of Aegis in real-life smart environment, we built a smart home testbed to represent the smart environment. We implemented Aegis as a platform-independent security framework in the smart home system. Also, as the majority of the smart environments support their functionality with services running in the

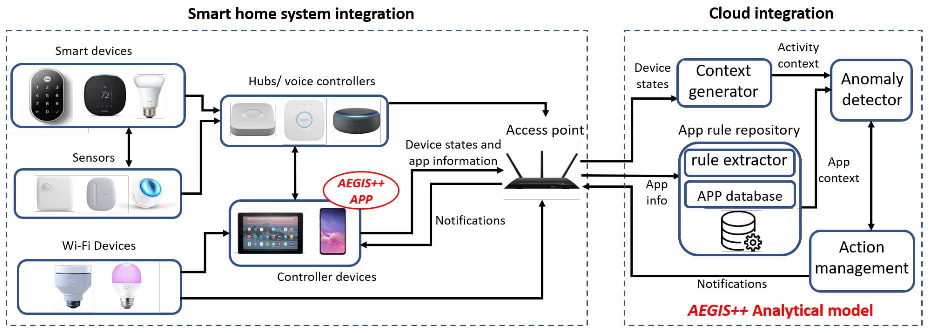


Figure 6.6: Aegis implementation in a multi-platform smart environment.

cloud, we implemented Aegis as a cloud-based solution. The integration of Aegis in real-life smart home platforms was divided into the following integration phases.

- **Smart home integration.** In the smart home integration phase, we built a customized smart app (Aegis app) to represent the data collector module described in Section 6.5. Figure 6.5 shows the sample user interfaces of Aegis app implemented in Samsung SmartThings platform. This app has two modules: data collection and action management. In the data collection module, Aegis app lists the device states as log files in smart home system and forwards the logs to Aegis's context generation module. Aegis app also logs the information of the installed apps to generate app contexts in context generation module. The action management module of Aegis app captures any notification generated in the anomaly detector (implemented in the cloud) in the event of any malicious activity in the smart home system and notifies the users. Additionally, users can control operational modes (detection and adaptive training) of Aegis using the action management module.

- **Cloud integration.** We implemented the analytical model of Aegis (context generation and anomaly detector) as a cloud-integrated security framework. The context generation module collects devices states forwarded from the Aegis app and creates user activity contexts by considering device states, sensor states, and timestamp. As users utilize different devices to implement their smart home environments, we also designed an app context extraction process capable of analyzing apps from multiple smart home platforms. We implemented a tool similar to available static analysis tool to automatically analyze and extract the trigger-action scenarios that define the app context [BSAU18,BCMU19,CBS+18]. For trigger-actions defined for other platforms, we found that the manual analysis and extraction was feasible as these platforms mostly define specific binary combinations of trigger-action interactions to control the devices. For the SmartThings apps, as they are written in Groovy [Sam18b], we used specific compile-time tools offered by the Groovy Metaprogramming capabilities [Met]. We first used the `ASTTransformation` class to extract the app's AST and build its ICFG. From there, we take advantage of pre-defined class visitors to explore the different nodes in the ICFG and extract the different trigger-action scenarios defined within the app's event handler methods [CBS+18]. In total, we used the app context extraction tool to automatically analyze 485 Smarthings apps, extracting a total of 587 different trigger-action scenarios. These trigger-actions scenarios were further stored in the multi-platform app rule repository that supports Aegis analysis. The app information forwarded from the Aegis app are cross-referenced with App rule repository to add app contexts into the training dataset for data validation purpose. The app contexts are only used in the adaptive training mode when the user installs a new device or new apps (details in Section 6.5.4). Aegis uses the generated app contexts to validate user activity context in a new smart home configuration to minimize the effect of new device and app addition. The device and app contexts are used to build the user activity context in a smart home system by the anomaly detection module of Aegis. The anomaly detection module uses this user activity contexts to train and detect any malicious events in the smart home (smart environment). In the event of any malicious activity in smart home system, the anomaly detector sends a notification to the Aegis app which uses push notification to alert users in real-time. Figure 6.6 shows the implementation of Aegis in a multi-platform smart environment (smart home).

6.6.3 Data Collection and Real-life Testbed

To test the efficacy of Aegis, we implemented a smart home testbed as a representation of connected smart environment where the users had the freedom to design their own smart home configuration and perform regular activities in a timely order. While collecting the user activity data, we considered five features that satisfy aforementioned design goals: *Anonymous User ID*, *User Role*, *Smart Home Layout*, *Activity Day-time*, and *User Policy*. For each user, we assigned an anonymous ID to ensure privacy. We also assigned a specific user role to each participant to understand the context of the user activities in a multi-user scenario. As mentioned earlier, we considered three different home layouts (single bedroom apartment, two bedroom home, and duplex home) and let the users choose their layout and smart devices. Moreover, we considered users' daily routine and collected the user activities performed in both weekdays and weekends, separately. Finally, we allowed the users to define their own policies in the smart home system during the data collection process.

We obtained necessary Institutional Review Board (IRB) approval to collect daily usage data of a smart home with multiple users. We invited current smart home users to participate in our study by circulating university-wide open calls and community outreach via emails and flyers. We selected 20 smart home users to collect daily usage data and provided monetary compensation to the users. While selecting participants for our study, we considered several features: (1) owns more than one smart device, (2) uses customized or vendor-specific apps/automation rules to control devices, (3) diverse user roles (working adults, housewives, young adults, student, etc.), (4) beginner level knowledge on using smart devices, (5) shares smart home environment with multiple users. To create the smart home environment, we considered the most common off-the-shelf devices. We chose 8 types of devices and 6 types of sensors to build the smart home testbed (total 14 types of devices). The detailed device lists and apps selected in our smart home environment are given in Table 6.2. We collected data from 20 different individuals with different user roles, user policies, and smart home layouts. Our dataset consisted of over 85000 events collected in a 15-day period. We considered two implementation scenarios based on smart home platforms while collecting user activity data.

- **Single platform smart environment.** Single platform smart environment (e.g., smart home system) offers a single access point or hub where all the installed devices and sensors in a smart environment are connected to perform various tasks collectively. We considered single platform smart home system in our testbed to collect daily user activity data and implement Aegis. We chose Samsung SmartThings platform to create the single platform smart home environment because of its large app market and compatibility with other smart devices [Gun17]. We implemented Aegis app as a third-party app in Samsung SmartThings hub that uses the *ListEvent* command from SmartThings API to collect the device logs. These device logs are forwarded to the Aegis's context generation module. We also created an app context database which consists of 485 official *Samsung SmartThings* apps to cross-reference and create app contexts. Whenever users install an app in smart home system, Aegis searches for existing app context in the database and adds the context into the training dataset for data validation purpose. For any third party app, users can manually use the source code of the app in Aegis and generate the app context which is later added in the database.

- **Multi-platform smart environment.** For multi-platform smart environment (smart home), we considered four different smart home platforms and devices in

Device Type	**Model**	**Description**	**Selected Apps**
Smart Home Hub	Samsung SamrtThings Hub	• Works as a central access point for smart home entities. • Supports Wi-Fi, ZigBee,and Z-Wave.	• Samsung SmartThings App.
Smart Speaker	Amazon Alexa Google Home	• Works as a central access point for smart home entities. • Supports Wi-Fi, ZigBee,and Z-Wave.	• Amazon Alexa app. • Samsung SmartThings App integration.
Smart Light	Philips Hue Light Bulb	• Uses a separate communication bridge to connect with smart home hub. • Uses ZigBee to communicate with other components in smart environment. • Supports up to 12 different sensors.	• Philips Hue app. • Brighten Dark Places. • Brighten my path. • Bright when dark and bright after sunset.
Smart Lock	Yale B1L Lock with Z-Wave Push Button Deadbolt	• Uses Z-Wave to connect with other devices. • Offers different pin code for different users. • Provides both manual and remote access.	• Lock it at a specific time. • Unlock it when I arrive.
Fire Alarm	First Alert 2-in-1 Z-Wave Smoke Detector and Carbon Monoxide Alarm	• Uses Z-Wave to connect with the hub • Provides built-in smoke and CO sensors.	• CO detector. • Smart alarm.
Smart Monitoring System	Arlo by NETGEAR Security System	• Uses Wi-Fi to connect with smart home hub. • Offers both live monitoring and still pictures.	• cameras on when I'm away
Smart Thermostat	Ecobee 4 Smart Thermostat	• Uses Wi-Fi to communicate with smart hub. • Can be configured with sensors.	• Ecobee connect. • Its too cold. • Its too hot. • Keep Me Cozy.
Smart TV	Samsung 6 Series UN49MU6290F LED Smart TV	• Connects with smart home hub using Wi-Fi.	• Power allowance. • Make it so.
Motion, Light, & temperature sensor	Fibaro FGMS-001 Motion Sensor	• Uses Z-Wave to connect with the hub. • Can be configured with different devices simultaneously.	• Any apps associated with smart devices.
Door Sensor	Samsung Multipurpose Sensor	• Uses ZigBee protocol to connect with smart home hub.	• Any app associated with smart devices.

Table 6.2: List of devices used in the data collection.

single physical environment. We chose Samsung SmartThings, Philips Hue, LIFX smart bulb, and Amazon alexa as different smart home platforms. Here, the smart home platforms and devices are installed as independent entitiy and no common access point is considered in the installation. However, they share the same network access point and perform different tasks independently. We customized and implemented Aegis app in each of these smart home platforms. Aegis app collects device logs using following APIs: *ListEvent* command from SmartThings API, *LIFX HTTP API* for LIFX smart lights [LIF18], *HUE API* for Philips Hue [Hue18], and *Smart home skill API* for Amazon Alexa [Ale18]. All the device logs collected by Aegis app in different smart home platforms are forwarded to cloud integrated context generation module of Aegis. We also used existing static analysis tools [BSAU18, CBS+18, BCMU19] to create app contexts from official apps and automation rules (485 SmartThings app, 20 Alexa skills, 10 LIFX rules) that are used to add app contexts in training dataset. These device logs are used to create the context-aware model and detect malicious activities in smart home environment.

For collecting the malicious dataset, we considered five different threat models (Section 6.4). We built five malicious apps to represent these threats. Our malicious apps cover several known threats presented by other researchers in [JCW+17, IoT17]. Our handcrafted malicious apps also included smart home attacks using smart speakers such as Amazon Alexa, Google Home, etc. [CMV+16, ZYJ+17]. To perform the attack described in Threat 1, we built a battery monitor App for smart locks that leaks the unlock code via SMS to the attacker. We realized the impersonation attack by unlocking the smart lock as an outsider using the leaked unlock code. For Threat 2, we built an app that injects false smoke sensor data to trigger the fire alarm in the smart home system. For Threat 3, we created an app that flickered a smart light in a specific pattern while nobody was in the home. To perform the denial-of-service attack described in Threat 4, we developed an app that stopped the smart thermostat for a pre-defined value. For Threat 5, we created an app that could generate

Aegis Threat Model	App Description	ContextIoT [JCW+17]	IoTBench [IoT17]	Voice command attacks
Threat-1: Impersonation attack	A battery monitoring app leaks unlock code via SMS and an attacker uses the code to unlock the smart lock by impersonating as valid user	• Backdoor pin code injection. • Lock access revocation. • LockManager. • App Update – PowersOutAlert. • Lock access revocation.	• Permissions- Implicit 2	
	An app that captures and replay a voice command to impersonate valid users.	—	—	• Stealing voice attack [AVEF12]. • Voice spoofing attack [MSS15].
Threat-2: False data injection	An app injecting false sensor data to a device.	• Fake alarm. • Remote control – FireAlarm. • Remote command – SmokeDetector.	—	—
	A voice command that maliciously or mistakenly inject false data to a connected smart device	—	—	• Hidden voice commands [CMV+16]. • Dolphin attack [ZYJ+17].
Threat-3: Side channel attack	An app flickers a smart light in a specific pattern while no user is present.	• Leaking information. • creating seizures using strobed light. • IPC – MaliciousCameraIPC & PresenceSensor. • MidnightCamera.	• Side Channel - Side Channel 1. • Side Channel - Side Channel 1.	
	A malicious signal (audio or light) causing a smart speaker to execute a legit yet vulnerable command	—	—	• Light commands [SCR+19].
Threat-4: Denial-of-Service	A malicious app that cancel any ongoing tasks in smart devices at a predefined value	• Disabling vacation mode. • Abusing permission.	—	
Threat-5: Triggering a malicious app	An app changing state of devices in a specific pattern to trigger an malicious app in a connected device	• Surreptitious surveillance. • Undesired unlocking. • IPC – MaliciousCameraIPC & PresenceSensor.	—	

Table 6.3: Malicious Apps mapping of Aegis, ContextIoT [JCW+17], IoTBench [IoT17], and existing voice command attacks.

morse code using a smart light while no person was in the room and triggered a smart camera to take stealthy pictures. Our malicious apps cover several existing attacks on smart home devices presented by the researchers [JCW+17, IoT17]. In Table 6.3, we mapped our threat models with existing malicious apps presented by the researchers. Additionally, we added some malfunctioning devices (e.g., a smart lock without power, fused smart light, etc.) in the smart home system to test Aegis in cases that include device malfunction. We collected 24 different datasets (4 dataset for each attack scenario) for a total of over 15000 events. We used 75% of the benign user data to train the Markov Chain model. Then the remaining 25% of data along with the malicious dataset was used in the testing phase which is a common practice [SAU19a, Goo19, AWS19b].

6.7 Performance Evaluation

In this section, we evaluate the effectiveness and feasibility of Aegis in detecting malicious activities in smart environment with real user data. We train the anomaly detector module of Aegis with data collected from multiple smart home users (representation of connected environment) for benign daily activities. For testing purposes, we use the user data as well as the malicious data collected from the adversary model described in Section 6.4.

In the evaluation of Aegis, we consider several research questions:

RQ1 What is the performance of Aegis in different smart home environment layouts and devices? (Sec 6.7.2)

RQ2 What is the impact of different apps, policies, and configurations on the performance of Aegis? (Sec. 6.7.3)

RQ3 What is the impact of different user behavior on the performance of Aegis? (Sec. 6.7.4)

RQ4 What are the performance overhead introduced by Aegis in a smart home system? (Sec. 6.7.5)

6.7.1 Performance Metrics

In the evaluation of Aegis, we used six different performance metrics: True Positive rate (TP), False Negative rate (FN), True Negative rate (TN), False Positive rate (FP), Accuracy, and F-score. TP rate indicates the percentage of correctly identified benign activities while TN rate refers to the percentage of correctly identified malicious activities. On the other hand, FP and FN illustrates the number of malicious activities identified as benign and the number of benign activities detected as malicious activities respectively. F-score is a indicator of accuracy of a framework which considers TP and TN as computational vector. The performance metrics are defined by the following equations:

$$TP\ rate = \frac{TP}{TP + FN}, \tag{6.7}$$

$$FN\ rate = \frac{FN}{TP + FN}, \tag{6.8}$$

$$TN\ rate = \frac{TN}{TN + FP}, \tag{6.9}$$

$$FP\ rate = \frac{FP}{TN + FP}, \tag{6.10}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{6.11}$$

$$F - score = \frac{2 * TP * TN}{TP + TN}. \tag{6.12}$$

In addition to the aforementioned metrics, we also considered several parameters such as power usage, memory usage, CPU usage, latency, etc. to measure performance overhead of Aegis in real-life smart environment.

6.7.2 Evaluation with Different Layouts

To evaluate Aegis in different smart environment layouts, we consider two important criteria (1) different smart home layouts, (2) multiple numbers of users. A smart home system can have different layouts and different number of devices. We tested the efficiency of Aegis in a multi-user environment and different smart home layouts.

Different smart home layouts: User activities in a smart home can vary depending on the home layout as different layouts can lead to different usage patterns. To evaluate Aegis, we considered three different layouts: single bedroom home, double bedroom home, and duplex home. Here, we considered a single authorized smart home user in different layouts. We collected data from 15 different users in these layouts.

		Normal Training						Adaptive Training					
Smart Home Layout	**Smart platforms**	**Recall**	**FN**	**TN**	**FP**	**Accuracy**	**F-score**	**Recall**	**FN**	**TN**	**FP**	**Accuracy**	**F-score**
Single Bedroom	Single platform	0.95	0.05	1	0	0.9547	0.9604	0.97	0.03	1	0	0.9712	0.9847
Home	Multi-platform	0.94	0.06	0.99	0.01	0.965	0.9643	0.97	0.03	0.98	0.02	0.975	0.9749
Double Bedroom	Single platform	0.93	0.07	1	0	0.9340	0.9655	0.96	0.04	1	0	0.964	0.9795
Home	Multi-platform	0.92	0.08	0.97	0.03	0.945	0.9443	0.945	0.055	0.99	0.01	0.9675	0.9669
Duplex Home	Single platform	0.91	0.09	1	0	0.9119	0.9529	0.96	0.04	1	0	0.9614	0.9688
	Multi-platform	0.90	0.1	0.96	0.04	0.93	0.9290	0.93	0.07	0.98	0.02	0.955	0.9543

Table 6.4: Performance evaluation of Aegis in different smart home layouts.

Our dataset also include single platform and multi-platform smart systems/settings. Table 6.4 presents the evaluation results associated with different smart home layouts. We can observe that accuracy and F-score for different layouts in single platform smart home vary from 96-91% and 97-95%, respectively. For multi-platform smart home, we can observe that the accuracy and F-score vary from 96-93% and 96-92% in different smart home layout, respectively. Aegis also achieves high TP (96-91%) and TN rate (100-96%) irrespective of layouts and number of platforms (single and multi-platform). One can safely confirm that variation in different layouts has a minimal impact on the performance of Aegis. Table 6.4 also shows how the performance of Aegis improves in adaptive training mode. Here, whenever the controller device (e.g., smartphone, tablet, etc.) is connected in the smart home network, we infer the user is in home location and use adaptive training mode. One can notice that the accuracy of Aegis increases from 95% to 97% in adaptive training mode for single bedroom layout for both single and multi-platform smart systems. For double bedroom and duplex home, Aegis achieves 96.4% and 96.1% accuracy in single platform and 96.6% and 95.4% in multi-platform smart system, respectively. As adaptive training mode uses user validation to reduce FP and FN events, F-Score increases to approximately 95-97% for all three layouts. In summary, Aegis can achieve accuracy and F-score over 95% for all three smart home layouts.

Different number of authorized users: Smart environment allow users to add more than one authorized user for the same smart system. Hence, a smart environment can have multi-user scenarios with different user activities happening at the same time. To evaluate this setting of the smart environment in Aegis, we collected data from several multi-user settings with different users performing their daily activities at once. We used different smart home layouts (example of with several multi-user scenarios (two, three, and four authorized controllers/conflicting users) in our data collection process. Additionally, we performed the aforementioned attack scenarios to collect malicious dataset and tested the efficiency of Aegis in different multi-user environments. Table 6.5 illustrates the detailed evaluation of Aegis in different smart home settings. For single bedroom layout, we can observe that accuracy and F-score reach the peak (0.9477 and 0.9729, respectively) for the two users setup. If we increase the number of authorized users in the smart home system, the accuracy gradually decreases with an increasing FN rate. Similarly, for double bedrooms and duplex home layout, Aegis achieves the highest accuracy and F-score for two authorized users' setup. Both accuracy and F-score decreases while the FN rate increases as the number of authorized users increases. The highest accuracy achieved in two bedrooms and duplex home layouts are 92.29% and 90.38%, respectively. As different users interact with smart home devices in varied ways, the FN rate increases with the number of users in the system. To minimize the number of FN events, we implement the adaptive training mode in Aegis. In a multi-user scenario, a notification is pushed in all the controller devices if Aegis detects a malicious event in adaptive training mode. All the authorized users can confirm the event based on their activities and Aegis trains the analytical model with validated data. One can notice from Table 6.5 that Aegis achieves the highest accuracy and F-Score (97% and 98%, respectively) for two users setup in single bedroom layout. Adaptive training mode also decreases FN rate approximately by 38.6% and increases the accuracy to 96% and 95.25% for three and four authorized user scenarios respectively. For two bedroom and duplex

		Normal Training						**Adaptive Training**					
Smart Home Layout	**No of Controllers**	**Recall**	**FN**	**Precision**	**FP**	**Accuracy**	**F-score**	**Recall**	**FN**	**Precision**	**FP**	**Accuracy**	**F-score**
Single Bedroom Home	2	0.9472	0.0528	1	0	0.9477	0.9729	0.9685	0.0315	1	0	0.9711	0.9839
	3	0.9399	0.0601	1	0	0.9405	0.9690	0.9564	0.0436	1	0	0.96	0.9777
	4	0.9041	0.0959	0.96	0.04	0.9352	0.9312	0.9482	0.0588	1	0	0.9525	0.9734
Double Bedroom Home	2	0.9222	0.0778	1	0	0.9229	0.9595	0.9654	0.0346	1	0	0.9682	0.9823
	3	0.9058	0.0942	0.9529	0.0471	0.9062	0.9288	0.9523	0.0477	0.9785	0.0215	0.9545	0.9652
	4	0.8806	0.1194	0.8941	0.1059	0.8807	0.8873	0.9476	0.0524	0.96	0.04	0.9486	0.9537
Duplex Home	2	0.9017	0.0983	1	0	0.9038	0.9483	0.958	0.042	1	0	0.9615	0.9785
	3	0.8901	0.1099	0.9238	0.0762	0.8909	0.9067	0.9512	0.0488	0.975	0.025	0.9531	0.9629
	4	0.8694	0.1306	0.8857	0.1143	0.8698	0.8775	0.9388	0.0612	0.953	0.047	0.94	0.9458

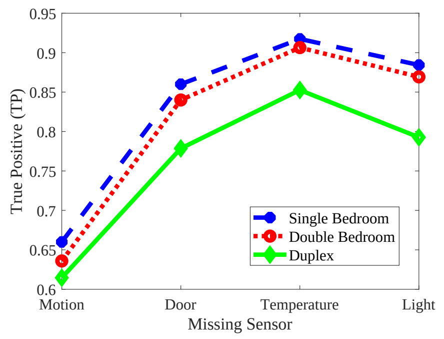
Table 6.5: Performance evaluation of Aegis in different multi-user scenarios.

home layout, adaptive training mode also increases the efficiency of Aegis. Adaptive training mode reduces FN and FP rate approximately by 60% while accuracy and F-Score increases to approximately 96% and 98% respectively in a double bedroom and duplex home layout. In summary, Aegis can minimize the effect of conflicting user activities in a multi-user scenario in adaptive training mode while increasing efficiency.

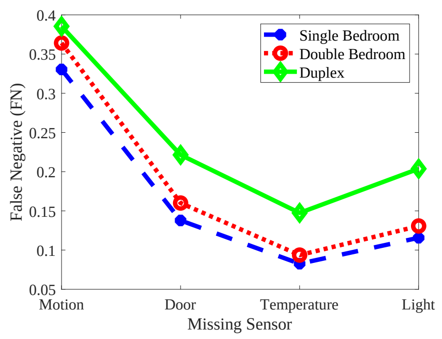
6.7.3 Evaluation with Different Smart Environment Configurations

In this sub-section, we evaluate Aegis based on different smart Environment configurations including (1) different sensor configurations, (2) different user policies, (3) different controller device configurations, and (4) number of installed apps.

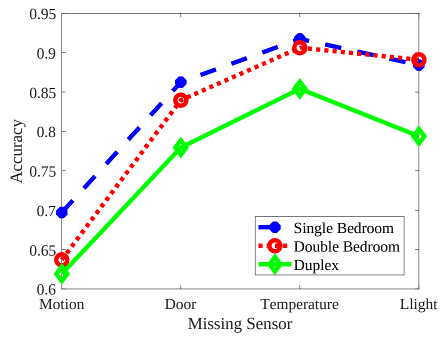
Different sensor configurations: To evaluate the efficiency of Aegis based on deployed sensors, we use several combinations of sensors to build the context-aware model of user activities and report accuracy in Figure 6.7. Since Aegis considers different smart sensors and devices as co-dependent components, we try to understand to what extent changing the combinations of sensors in a smart environment affects Aegis's performance. For this, we tested the efficacy of Aegis with four different combinations of sensors: without motion sensor, without the door sensor, without the



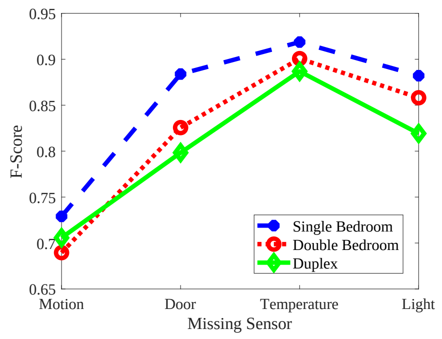
(a) True Positive (TP)



(b) False Negative (FN)



(c) Accuracy



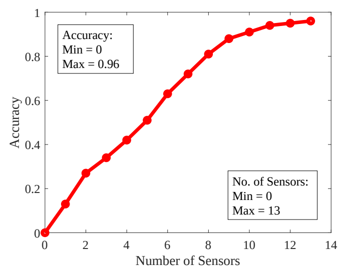
(d) F-Score

Figure 6.7: Performance evaluation of Aegis with different sensors.

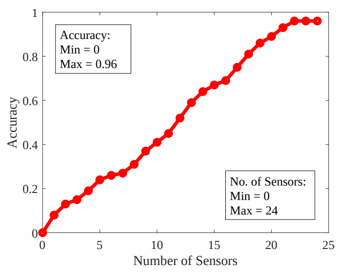
temperature sensor, and without the light sensor. As seen in Figure 6.7(c) and 6.7(d), decreasing the number of sensors from the context-aware model in Aegis declines the accuracy and F-score of the framework. Removing the motion sensor resulted in the lowest accuracy and F-score (61% and 68% in duplex home layout, respectively). As motion sensors are configured with the majority of the devices (smart light, smart lock, etc.) and used in most of the user activity context, it affects the performance of Aegis significantly. We can also observe that removing sensors from the smart home system introduces high FN rate as our proposed framework cannot build the context of the user activities correctly (Figure 6.7(b)). Again, Figure 6.7(c) illustrates that removing the temperature sensor from the smart home system does not influence the performance significantly (85-91% accuracy and 88-91% F-score in different layouts).

The main reason is that the temperature sensor can be configured with a limited number of devices; hence, it is affected by user activities less than other sensors. Without the door sensor and light sensor, Aegis can achieve moderate accuracy ranges from 77%-86% and 79%-88%, respectively. Figure 6.8 illustrates the change in accuracy of Aegis for changing the number of sensors in different smart home layouts. For all three smart home layouts (single bedroom, double bedroom, and duplex home), limiting the number of sensors in the system decreases the accuracy of Aegis. In conclusion, limiting the number of sensors in a smart environment can reduce the efficiency of Aegis by introducing FN cases in the system.

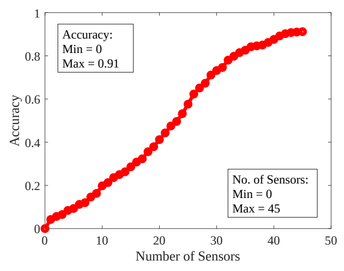
Evaluation Based on Installed Apps: Users in a smart environment can install multiple smart apps to configure and control the same devices or different devices at the same time. For example, users can install two different apps to control a smart light at a time with motion and door sensor respectively. To test the effectiveness of Aegis based on the installed apps, we installed 12 benign apps in total in the system to build the context-aware model of user activities. Figure 6.8(d) shows the accuracy of Aegis in detecting malicious apps in a smart home based on installed apps. Here, we installed different malicious apps (Section 6.4) in the system with multiple benign apps to determine the effectiveness of Aegis. From 6.8(d), one can notice that Aegis achieves the highest accuracy of 98.15% for Threat-2 and the lowest accuracy of 94.34% for Threat-3 for only one benign smart app installed in the system. With the increment of benign apps in the smart platform (highest 12 benign apps), accuracy ranges between 98% to 95% and 94% to 92.5% for Threat-2 and Threat-3, respectively. The accuracy of Aegis in detecting Threat-1, Threat-2, and Threat-5 varies between 96% to 93%. We also tested different malicious apps installed at once in the smart platform with a fixed number of benign apps (12 benign apps) to understand the effectiveness of Aegis completely. Figure 6.8(e) depicts the accuracy



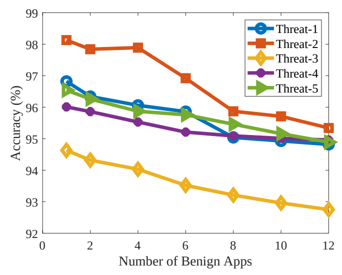
(a) Single Bedroom



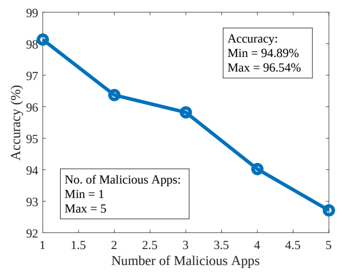
(b) Double Bedroom



(c) Duplex Home



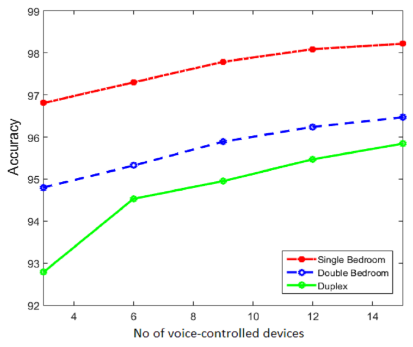
(d) Accuracy vs. benign



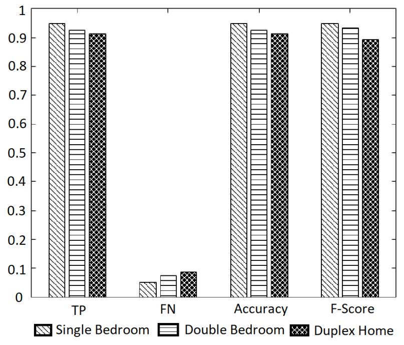
(e) Accuracy vs. malicious

Figure 6.8: Accuracy of Aegis with different number of sensors (a), (b), (c) and with different number of benign and malicious apps (d), (e).

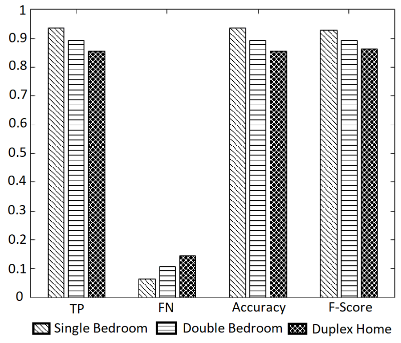
of Aegis based on the number of malicious apps installed in the system. One can notice that Aegis can achieve an accuracy of 98% for one malicious app installed in the smart home system which decreases very little with higher number of malicious apps (92.57% with five malicious apps). In conclusion, the performance of Aegis changed very little with the change in the number of benign apps and malicious apps installed in the smart environment.



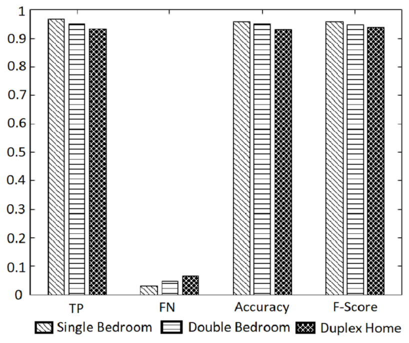
(a) Voice-controlled smart devices



(b) User Policy 1



(c) User Policy 2



(d) User Policy 3

Figure 6.9: Performance evaluation of Aegis in a voice-controlled smart environment (a) and performance of Aegis in a policy-enforced smart environment (c), (d), (e).

Evaluation Based on Controller Device Configurations: Modern smart systems allow users to configure smart devices with different controller devices. Smart device users control devices usually via vendor-specific controller apps installed in smartphone and tablet. Additionally, smart devices can also be controlled via voice commands if configured with a smart speaker (e.g., Amazon Alexa, Google Home, etc.). As explained in Section 6.4 and 6.6, attackers can manipulate voice-controlled configuration of smart devices by injecting false commands and impersonation (details in 6.3). In this subsection, we test the efficacy of Aegis in detecting malicious activities targeting voice controlled smart devices. From Figure 7.7(d), one can notice that the accuracy of Aegis increases with the number of voice-configured smart devices as the increment in number of voice-controlled devices allows Aegis to understand the user activity contexts properly. For single bedroom layout, the accuracy of Aegis varies from 98-96% in detecting malicious activities targeting voice controlled devices. Aegis also achieves high accuracy (96-94% and 95-92%, respectively) in double bedroom and duplex home layout. In summary, Aegis achieves high accuracy in detecting malicious activities targeting voice-controlled configurations regardless of environment layouts.

Evaluation Based on User Policies: In smart environment, users can define customized policies to control the smart devices. For example, users can impose a time window to activate a smart light in a smart home system. In this sub-section, we test the efficiency of Aegis with different policies enforced in smart environment. We consider the following user policies to evaluate Aegis:

User Policy 1: Users can apply time-specific operations for different smart devices and sensors. In policy 1, users configure a smart light with the motion sensor which will be enforced only from sunset to sunrise.

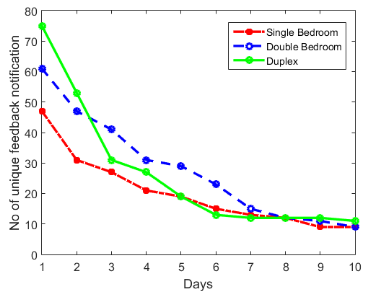
User Policy 2: Users can apply sensor specific operations for different smart devices. In policy 2, users configure smart lights with light, motion, and door sensors.

User Policy 3: Users can configure smart speakers to control smart devices using voice commands. In policy 3, users configure smart lights with smart speakers to control via voice commands (e.g., bedroom light on).

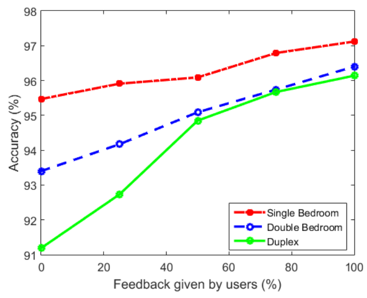
Figures 6.9(b), 6.9(c), and 7.7(c) present the performance of Aegis in these policies enforced in smart environment. One can observe that Aegis achieved accuracy as high as 95% while enforcing time-specific operations in smart environment (smart home) (Figure 6.9(b)). The F-score also ranges from 89% to 94% for different layouts with time-specific operations with low FN rate (5%-8%). For User Policy 2, one can observe a slight fall in the accuracy and F-score as changing sensor-device configuration introduces FN cases in the system. From Figure 6.9(b), we can see that Aegis can perform with an accuracy ranging from 85% to 93% for different layouts while changing the sensor-device configurations. Aegis also achieves F-score ranging from 86.5-92% for different configurations. For controller-specific operation (voice-controlled operation), the accuracy of detecting malicious events ranges from 96-93% with low FN rate (3-6%) in different smart environment layouts. Also, Aegis achieves high F-scores (96-94%) in detecting malicious activities in voice-controlled device configuration. In summary, Aegis can detect malicious activities in policy-enforced connected smart environment with high accuracy and F-score.

6.7.4 Evaluation with Different User Behavior

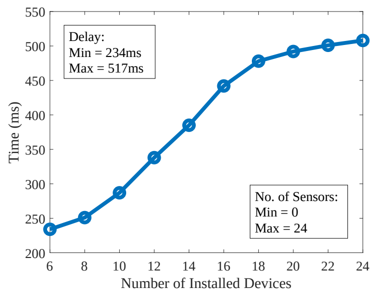
In this sub-section, we test Aegis in terms of user interactions and behavior in the smart environment. Aegis uses an adaptive training method which requires users' feedback to detect FP and FN cases. This adaptive training method may cause user fatigue with excessive feedback notifications [AF13]. To determine how the user fatigue may affect the performance of Aegis, we performed accuracy vs user feedback study which is showed in Figure 6.10(a) and 6.10(b). Figure 6.10(a) shows the number of notifications generated in adaptive training mode by Aegis in different smart system settings over a 10 day period. One can notice that in all three layouts the number of generated notification decreases significantly. For the single home layout, the number of notifications decreases by 59% in 5 days. For double bedroom and duplex layout, the number of notifications also decreases by 52.45% and 74.67%, respectively. This indicates that users only have to deal with higher feedback requests for a short period of time. Note that Aegis pushes a notification for both FP and TN events as our test dataset includes both normal user activity and malicious events. Hence, the number of notifications generated for only FP events is lower than it seems



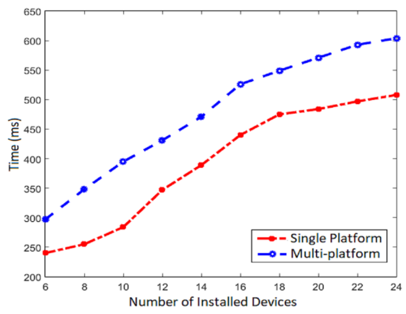
(a) User notification vs time



(b) Accuracy vs. Feedback



(c) Training time vs. Device



(d) Training time vs. Platforms

Figure 6.10: Performance of Aegis in terms of user feedback (a), (b) and performance overhead of Aegis (c), (d).

in Figure 6.10(a). For example, in day 10, the total number of notification is 10 among which 6 notifications are from FP. Figure 6.10(b) shows how user feedback affects the accuracy of Aegis in detecting different threats. One can notice that the accuracy of Aegis increases very little from 50% to 100% user feedback. This indicates that if the users actively train Aegis in the initial period (1-5 days) in adaptive training mode, the performance improves significantly. Again, Aegis always provides the option to choose a specific time for adaptive training mode to the users. In conclusion, Aegis can negate the consequence of user fatigue by terminating adaptive training mode after an initial period, which are configurable by the users.

6.7.5 Performance Overhead

We illustrate the performance overhead of Aegis, including resource overhead and latency. As Aegis runs in the *Samsung SmartThings* hub (Data Collection and Action Management Module) and local machine (context generator and anomaly detector), the performance overhead is minimal in terms of resources. We identify three major features that could introduce a time delay in real-time operation.

Delay in adaptive training model: Aegis offers adaptive training mode where any malicious event detected by Aegis is forwarded to the user for validation. Aegis uses this validated data to retrain the analytical model which introduces a time delay in the operation. In Figure 6.10(c), we illustrate the time needed for re-training the framework with respect to the number of devices installed in the device. One can notice that Aegis takes approximately 230 ms to train when the system has 6 different devices installed in the smart environment. The training time increases to 519 ms for 24 installed devices in the smart environment. In short, Aegis introduces negligible overhead in terms of time delay in adaptive training mode.

Delay in multi-platform smart environment: Aegis allows multi-platform system configuration where smart devices form different platforms can share the same physical home environment. Aegis uses multiple customized apps to collect device's and sensor's state information to build a context-aware model from user activities. As smart devices from different platforms vary in resource (memory, CPU, command execution frequency, etc.), collecting data and building context-aware model from multi-platform smart environment introduces a time delay in the operation of Aegis. Figure 6.10(d) shows a comparison between time delay introduced in single and multi-platform (4 different smart home platforms) smart environment with respect to number of installed devices. One can notice that for a minimum of 6 devices Aegis takes approximately 220 ms and 300 ms in single and multi-platform smart environment, respectively. The highest time delay introduced in multi-platform smart environment is approximately 600 ms for 24 installed device compared to 508 ms in single platform smart environment. In summary, AEGIS introduces minimal time delay in multi-platform smart environment compared to single platform environment. Hence, AEGIS is effective in detecting malicious activities irrespective of number of smart devices and platforms installed in the smart environment.

Delay in action management module: Action management module of AEGIS alerts users in the event of malicious activity in smart environment. The alert message is sent to the controller device (smartphone, tablet, etc.) in a form of notification which introduces a time delay in the action management module. We use a *SmartThings* app to send notifications to controller devices of authorized users. This app communicates with the cloud server via *http* protocol which is connected with the action management module (Section 6.5.4). On average, action management takes 210 ms time to send a notification to the controller device from the moment of malicious activity detection which is low for real-world deployment. In short, we conclude that AEGIS meets the efficient demands in the action management module.

6.8 User Scenarios and Benefits

In this section, we illustrate how deploying AEGIS in a smart environment can help different groups of consumers using several use scenarios and discuss different benefits of AEGIS.

6.8.1 User Scenarios

We illustrate three different user scenarios to understand the benefits of AEGIS among vendors, end-users, and developers.

Vendors- Smart device vendors can use Aegis to detect an abnormal behavior in a customer's implemented smart system. Here, a customer, *Alice*, installs several smart security devices (smart lock, smart camera, smart fire alarm, etc.) and the corresponding smart apps to control them. However, one of the installed app has malicious code that injects false sensor data when no one is at home to trigger the fire alarm (Table 6.3- Threat- 2). As Alice does not have any idea of this malicious event, she calls the security service provider/vendor for support. In this situation, the security service provider can identify that the alarm is generated from a false data from the state model generated by Aegis and support the customer with appropriate suggestions such as deleting the malicious app, reinstalling the correct app, etc.

End-users- End-users constitute the most common victims of malicious events in a smart environment. Attackers can perform several malicious activities including gaining physical access to the smart environment. For instance, a smart device user, *Bob*, installs a new smart lock and the corresponding app in the smart home system. However, the installed app has a malicious snippet to forward the unlock code to the attacker so he can unlock the smart lock by impersonating Bob (Table 6.3- Threat 1). Aegis can identify this event and notify the user in real-time. Moreover, Bob can change the state of the lock to unlock and prevent any physical access of the smart home. Smart device users also tend to install devices from different vendors in the same physical environment. Aegis provides a platform-independent security framework to the end-users which can detect malicious activities irrespective of the platform of the devices. For instance, a malicious app can inject a voice command to the smart speaker to open the front door (Table 6.3- Threat 2). The contextual model of Aegis can confirm the presence of the user by checking the state of the installed presence sensor even if it is not connected to the smart speaker. Hence, Aegis can detect malicious activities in smart devices using the context-aware model in a multi-platform smart environment.

Developers- Developers or tech enthusiastic users can deploy Aegis in their smart system and specify different rules to enhance the security of the their system. For instance, *Kyle*, a smart home user, installs multiple smart lights and motion sensors in his system. Kyle also builds a new smart app to control the lights with motion. By using the logic extractor of Aegis, Kyle can understand whether his app logic is correct or not. Moreover, Kyle can use the adaptive training mode to see how the overall state of the smart home changes with new devices and apps. If the action of the new smart lights contradicts the existing system, or any malicious event occurs (e.g., Table 6.3- Threat 3), Kyle can understand the cause of the event and take necessary steps. Moreover, Kyle can understand the working conditions of smart devices and improve his technological knowledge using Aegis.

6.8.2 Discussion

Deployability in Real-life System- One of the prime features of Aegis is easy deployability in real-life systems. Aegis uses simple smart apps to collect device states from multiple smart platforms installed in same smart environment and build the context-aware model. The detection mechanism runs in the cloud which does not hamper the normal operation of the smart environment. Users can install Aegis similarly to any other smart app.

Applicability and Real-life Threats- Security risks may arise from smart apps performing side-channel attacks. For instance, a smart app can flash the light in a specific pattern to leak information or trigger another connected device which can be considered as a threat. While most of the existing solutions consider this threat as out of scope [CBS$^+$18,MAH$^+$16], Aegis successfully detects such malicious behaviors.

In addition, Aegis can detect device malfunctions inside a smart environment. For instance, if a smart light makes light patterns while no one is around, there will be activity in the light sensor and the light bulb, but no activity in the motion and presence sensors. Additionally, device malfunctions inside smart environment can be detected. For instance, if a smart light is configured with the motion sensor, one should expect that the light turns on due to the active motion. Other outcomes from this specific context may be categorized by Aegis as a malfunction.

Multi-user activity in smart environment- In smart environment, more than one user may perform different activities simultaneously. As Aegis utilizes user activity contexts to detect malicious actions, correctly distinguishing between different user activities is key. Instead of single-context analysis, Aegis uses a pattern of contexts to understand the user activities. Hence, Aegis can detect simultaneous activities performed by different users and devices in a smart environment. For instance, if two users are walking towards the same point from opposite directions, Aegis observes the related contexts to identify two different motion activities.

Trigger-action effect time- Smart devices use sensors to automate tasks. For instance, a smart light can be triggered by a motion sensor or a door sensor. Each trigger-action scenario has an effect time (time duration of a device being active). This effect time has to be correctly considered to build the context of the user activity. Aegis mitigates this time dependency by considering the pattern of device utilization. For instance, the user sets a smart light to remain on for two minutes if a motion is detected. This case is detected by Aegis by checking consecutive states of the overall smart home and is used to detect malicious apps or malfunctioning devices (if the motion is sensed by the sensor and it holds the state for 20s, the smart light should be also on for, at least, 20s otherwise broken or malicious). Aegis uses these trigger-action scenarios to mitigate the effect of the time interval and builds the contextual model from device state patterns.

Detecting rare events- In a smart environment, different autonomous events occur based on device configuration and user activities. These events may include rare events such as triggering fire alarms. As Aegis uses daily user activities to train its analytical model, these rare events might be unaddressed and flagged as threat. To solve this, we use the app context to verify unrecognized events in Aegis. Any alert triggered in Aegis is verified with app context generated from the installed app (Section 4.2). If the app context is matched with the rare event, Aegis considers the event as benign and retrain the model automatically. Users can also check and verify rare natural events through action management module (Section 6.5.4).

Multi-platform support- Aegis supports multi-platform smart environment which allows users to install smart devices from different platforms in the same physical environment. For instance, users can install both Samsung SmartThings devices and Wi-Fi devices such as LIFX smart bulbs in the same smart home. While existing solutions cannot ensure security in this multi-platform environment, Aegis observes devices state changes and builds a contextual model which can envision user activity contexts and multi-platform correlation properly in a smart environment. Also, Aegis only considers device states to build the contextual model which does not need any platform-specific and app-specific modification. Hence, Aegis can detect malicious activities in a multi-platform smart environment.

6.9 Conclusion

Modern app-based connected smart environment expose the smart ecosystem to novel threats. Attackers can easily manipulate the smart devices and sensors to perform different attacks or deceive users into installing malicious apps. However, current detection solutions only consider threats to devices and apps in the smart environment, skipping the very-complex and rich context-based relationships among smart platforms, devices and apps. In this chapter, we presented Aegis, a novel context-aware platform-independent security framework for connected smart environment that detects malicious device and sensor activities by (1) observing the change in device behavior based on user activities in multi-platform smart environment, (2) correlating sensor-device trigger-action scenarios with user activities in multi-platform environment, and (3) building a contextual model to differentiate benign and malicious behavior. We evaluated Aegis in multiple smart settings, with real-life users, with real smart devices (i.e., Samsung SmartThings, LIFX smart bulbs, Amazon Alexa, Google Home platform), and with different day-to-day activities. Our detailed evaluation shows that Aegis can achieve over 95% of accuracy and F-score in different smart settings. We also tested Aegis against several malicious behaviors. Aegis is highly effective in detecting threats to smart home systems regardless of the layouts, smart platforms, the number of users, and enforced user policies. Finally, Aegis can detect different malicious behavior and threats in smart environment with minimum overhead.

CHAPTER 7

MULTI-USER MULTI-DEVICE-AWARE ACCESS CONTROL SYSTEM FOR SMART DEVICES AND ENVIRONMENTS

7.1 Introduction

This chapter presents KRATOS, a multi-user multi-device-aware access control system for smart environment (e.g., smart home, smart factory, smart office, etc.). We designed KRATOS based on a user study conducted among 72 smart home users to reflect the real-life needs of access control in smart environment. KRATOS introduces a formal policy language for smart device users to create different device usage policies by specifying their needs in a structured manner. KRATOS implements a novel policy negotiation algorithm that automatically detects and solves conflicting demands from multiple users sharing same physical environment by leveraging user roles and priorities. KRATOS generates optimized policies for multiple users in a smart environment, reviewing the outcomes of the policy negotiation and implementing the negotiation results over the smart devices, sensors, and installed apps. We implemented KRATOS in a real-life a multi-user multi-device smart environment (smart home system) that include 17 different sensors, actuators, and devices. We further evaluated performance of KRATOS on 219 different policies including 146 demand conflicts and 33 restriction policies collected from 72 smart device users. We also tested KRATOS against five different threats and achieved 100% accuracy in detecting the threats in real-time. Finally, we performed a usability study among 43 real-life smart device users. Our extensive evaluation shows that, KRATOS can resolve demand conflicts and detect different threats with 100% success rate with an average rating of 4.6 out of 5 in usability in smart environment.

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

- We conducted an access control user study with 72 different smart device users to understand the real-life needs for multi-user multi-device access control system in smart environment.
- We introduced Kratos, a novel multi-user multi-device-aware access control mechanism for smart environment. Kratos considers conflicting user demands in multi-user smart environment and implements a flexible user policy-based access control to define user roles and understand users' demands. Kratos introduces a formal policy language to express users' desires and a novel policy negotiation algorithm to automatically identify and resolve conflicting user demands and restrictions in a multi-user smart environment.
- We implemented Kratos on a real-life smart environment (smart home system) using 17 different smart devices and sensors. Further, we evaluated its performance with 289 different policies collected from 72 different smart device users and five different threats. Our extensive evaluation shows that Kratos effectively identifies and resolves conflicting user demands and detects different threats in real-time.
- Finally, we performed a usability study with 43 different smart device users to understand the effectiveness of Kratos. Our results showed that Kratos achieves an average of 4.6 ratings out of 5 from the users based on common usability metrics including user friendliness, acceptability of use, availability, real-time response, and effectiveness.

Prior Work	Domain	Multi-user Multi-device environment	A.C. Threat Model	User Interface	Conflict resolution	Overhead analysis	A. C. Language
xShare [LRH+09]	Smartphone	○	○	●	○	●	○
DiffUser [NYB+09]	Smartphone	○	○	●	○	●	○
Capability-based A. C. [GPR13]	IoT network	●	○	●	○	○	○
Situation-based A. C. [SST18]	Smart home	●	○	○	○	○	●
Expat [YPHC19]	Smart home	●	○	○	○	●	●
Zeng et al. [ZR19]	Smart home	●	○	●	○	○	○
KRATOS	Smart home	●	●	●	●	●	●

Table 7.1: Comparison between KRATOS and other access control mechanisms (A.C. stands for Access Control).

7.2 Differences from Existing Works

Table 7.1 shows the major differences between Kratos and existing access control mechanisms in smart environment. In general, previous works consider access control needs in either single-device single-user or single-device multiple-user environment. Additionally, existing works acknowledge the conflicting user demands in multi-user multi-device environments that reflects users' relationships, social norms, and personal preferences. Compared to previous works, Kratos presents a usable fine-grained access control system that considers multi-user multi-device smart environment. Kratos reflects the user feedback in its design collected from a user study. Kratos offers easy new-user addition with user priority level that considers device-specific usage, user-specific restrictions, location-based access, and novel policy negotiation algorithm for resolving conflicting demands. In addition, Kratos provides easy policy assignment and management capabilities for multiple users in a smart environment by implementing easy-to-use user interfaces. Kratos also detects different threats arising from over-privileged users and coarse-grained access control systems with minimum overhead in real-life deployment.

7.3 Access Control Needs in Smart Environment

To understand the real needs for multi-device multi-user access control systems in smart environment, we conducted a user study among 72 real-life smart device users. We designed Kratos to address these needs in a fine-grained access control system. As user demands in a smart environment such as smart home vary based on diverse relationships, preferences, and social norms, it is important to understand uses' expectation in an access control system. We obtained Institutional Review Board (IRB) approvals to conduct the user study and provided monetary compensation to the participants. We divided the survey into three main blocks of questions to efficiently characterize users' needs in a multi-user multi-device smart environment.

- *Block 1 – User Characterization*: This group of questions focused on characterizing the users based on age, technical experience, household characteristics, etc. With this, we aim to understand the needs and interest of a diverse group of users. We asked questions about the user's preferences and experiences in a smart environment including user experience, interests on particular smart device and platform, household characteristics, social relationships and norms in multi-user settings, etc.
- *Block 2 – Smart Home Access Control*: The second set of questions aimed to characterize the user's smart home setting preferences in different multi-user settings. We asked questions to understand (1) users' experience in a multi-user smart environment, (2) real-life needs for access control systems, and (3) users' expectations about the outcome of the access control system.
- *Block 3 – Multi-user/Multi-device Scenarios*: Finally, in the third block of questions, we asked the users about general multi-user and multi-device settings. The goal of these questions is to understand users' views regarding how different conflicting demands, policy negotiations, user restrictions, and users with different priorities should be handled.

In the following, we present the results of the user study and explain how KRATOS reflects the needs of users in its design features.

7.3.1 User Characterization

We surveyed a total of 72 smart device users. We fully characterize the group of users and their households based on age, smart device usage and interests, and technical experiences.

Age: Out of the total 72 users, 20 (27.79%) users reported ages in the range of 25-34 years, 41 (56.94%) users were in the range of 25-34 years, and 11 (15.27%) in the range of 35-44 years.

Smart Device Usage: 60 out of 72 users (85.7%) mentioned that they either have previously used or currently have some smart devices in their households.

Smart Device Types: We also asked users regarding the smart device types they had experienced with. The most popular devices among the surveyed users were: Smart TV 49 (68.06%), smart light 31 (43%), smart thermostat 20 (27.78%), smart camera 15 (20.83%), smart lock 12 (16.67%), and smart switch 10 (13.89%).

Smart Home Platforms: We included 10 different smart device platforms and asked users regarding their experience with these platforms. The users stated that they were familiar with four smart device platforms: Google Home (67 users - 93.05%), Samsung SmartThings (55 users - 76.38%), Apple HomeKit (39 - 54.16%), and Open-HAB (12 users - 16.67%). Furthermore, we questioned user regarding their likelihood of using these smart device platforms. Similar to the previous results, Google Home, Samsung SmartThings, and Apple HomeKit were the most preferred smart device platforms among the users, 42 (58.33%), 20 (27.78%), and 10 (13.89%), respectively.

Technical Experience: We also characterized the users based on their technical experience with smart devices, platforms, and Apps. Out of the total, 55 (76.38%) of the users reported that they knew how to set up smart devices, 38 (52.78%) stated that they knew how to install Apps, and 35 (48.61%) said that they felt comfortable integrating different smart devices using a hub or cloud-based platforms.

Household Characteristics: Finally, we concluded the first block of the survey by asking questions to visualize the household characteristics of the users. The users reported that they lived with different family/roommate sizes. For instance, 26 (36.1%) users stated that they lived in a size of 4, 17 (23.61%) reported living in a size of 3, and 21 (29.16%) users shared their spaces with at least another person. The remaining eight users lived in a family size of 5 or higher. Furthermore, we asked the users regarding their experience in shared smart home devices and platforms. Interestingly, out of 36 users, only 4 (5.5%) reported that they did not share smart devices with other users. Then, 63 out of the 68 remaining users disclosed that they shared multiple smart devices with at least two more household members and up to 7.

We use the answers obtained in this block of survey questions to characterize the targeted users of Kratos. In most cases, users of smart devices, platforms, and apps know how to configure devices and install apps (vendor provided and third-party). Additionally, multi-user smart households represent a positive potential smart environment for fine-grained multi-user access control systems like Kratos. Finally, we note that most of the users reported that they share a smart device with at least two other household members.

7.3.2 Smart Home Preferences and Characterization

In this block of questions, we also asked smart device users questions regarding expected access control features in a smart environment.

Multi-member Settings: We asked users if they had ever considered or felt need to define various control policies on the other users in multi-user environment while installing or using smart apps. In total, 53 (73.61%) users answered “Yes” to this question which reflects user needs of access control system in multi-user smart environment.

Multi-member Access: Further, we investigated if the users were ever had given device accesses to other members. In this case, a higher majority of users (61 users, 84.7%) answered "Yes"

Conflicts Among Settings: In multi-user scenarios, 43 (59.72%) surveyed users disclosed to update or re-evaluate smart device and platform settings after discovering that previous configured settings had been modified by other members of the household that had authorized access to the devices.

Multi-member Admin Interface: Regarding the multi-member scenarios, 64 out of 72 users agreed that smart devices, platforms, and apps should have a user-friendly interface that can be regularly checked by the device owner to control and monitor the access rights of installed smart devices.

Guest Access: Lastly, we asked about giving device access to guest/temporary members (visitors, tenants, etc.). A vast majority of users (61 users - 84.72%) responded "Yes" to the option of smart Apps having an automated mechanism to revoke temporary access requests from guest users.

Overall, these set of questions shows the need for access control mechanisms in smart home systems. We found that the device owners frequently had to deal with user conflicts as a result of settings changed by other members of the household. Additionally, the device owner wants to have control regarding who accesses the devices and desires to enforce limited access controls for the users that are not fully trusted.

7.3.3 Multi-user/Multi-device Access Control

Finally, we assess the need for the fine-grained access control mechanisms, and feedback of users on the design and implementation of such mechanisms. Below, we group the responses of the users regarding the features of access control mechanisms in 10 different topics.

Integrated Access Control: We asked the users whether they felt the need of an access control mechanism in smart environment while configuring and using smart devices. A majority of 64 (88.8%) users answered that it would be an essential feature and smart platforms should provide an integrated access control system reflecting user needs.

Separated Access Control: We further asked whether there is a need for a separate access control app/system to manage the smart devices and platforms. Surprisingly, 58 (80.56%) users positively answered. Out of 58 users, 41 (70.68%) users desired to use an access control application if it were free and secure, while 17 users (29.32%) stated that they are willing to pay for the access control service if available. Additionally, the users agreed to share some specific personal information (PI) with the access control app if required for the app design. Out of 6 different options provided, the users stated that they would allow the app to use their email address (39 users - 67.24%), smart home user ID (34 users - 58.6%), smart home account credentials (31 users - 53.44%), and smart device ID (21 users - 36.2%).

User Priorities: In a multi-user environment, access control can be provided by assigning priorities to different users. We asked the survey participants about who in the household should have the highest (the most trusted member) and the lowest (the least trusted member) priority. The users assigned priority levels in between these boundaries. The "spouse/partner", "father", and "Mother" are among the members with the highest priorities. On the contrary, "babysitter ", "temporary guest", "frequent visitor", and "cleaning personnel" were among the members that had the lowest priority. In summary, temporary users are considered as least trusted members by the survey participants and should be given lowest priority.

Device Priorities: We evaluated what type of devices should be included in the access control mechanism. Out of 18 device options, the devices related to security and safety were selected to be the most important devices for access control. This list includes devices such as smart lock, smart thermostat, smart fire alarm, smart monitoring system, presence sensor, and smoke sensor. On the other hand, devices with the least importance to the user were the smart coffee machine, doorbell, and smart Light.

Automated System: Out of 72 smart device users, 45 (62.5%) users answered positively to the possibility of having an automated negotiation system to solve access control conflicts among users with the same level of priority.

Update Policy Feature: Out of 72 survey participants, 53 (73.61%) participants expressed their interest in having a feature to update/change current access control policies.

Negotiation Process: We presented four different options to the users about how the policy negotiation process should work among users of the same priority (superusers or admin level). The answer of users shows that users desire to automate this process with minimal interaction (51 users - 70.83%). The users' answers also suggested that the access control system should notify the members affected by the policy conflict.

Multiple Policies: The users reported that the access control system should allow multiple policies when a conflict occurs. 56 (77.78%) users suggested that a simple notification and an automated approval of the non-conflicting policies is sufficient to notify users regarding conflicting demands and policies.

Conflicting Policies: If two policies conflicts and one of them is defined from a member with lower priority, 33 (45.83%) users suggested that the lower priority policy should be rejected. However, 25 (34.72%) other users indicated that the member with the higher priority should be asked to assess the possibility of changing its policy to allow the lower priority member to add its settings without conflict. In both cases, they again suggested that a notification system is a critical feature to resolve policy conflicts. Furthermore, we presented a use-case scenario where a member having the same priority level as the owner introduces a conflicting policy on behalf of a low-priority member. In this scenario, 39 (54.17%) users suggested that both the owner and the member with similar priority should be notified so that they can negotiate together how to solve the conflict. However, 23 (31.94%) users proposed that the high-priority member should be notified about the conflict with the owner and the new policy should be automatically rejected.

Low-Priority Members: The last two questions were about managing the low-priorities members. The majority of the users (55 - 76.38%) confirmed that the access control system should have a feature to monitor the actions and commands from low-priority members while other 15 (20.83%) suggested that this may be an additional feature to have. We obtained similar results when we specifically asked about access control for guest/temporary members. In this case, 47 (65.27%) survey participants replied that guests/temporary users needed to have restrictions while other 20 (27.78%) users said that this would be an additional feature to include in the access control mechanism.

Clearly, the users had expressed their interests in having access control mechanisms for multi-user multi-device smart environment. Also, users suggested that despite a necessary notification system, the access control mechanism should work effectively with minimal user interaction. Finally, users stated that the access control system should be able to differentiate between users with different priority levels and negotiate conflicting demands accordingly. Conflicts between users with same priorities should be resolved with the intervention when necessary while conflicts between members with different priority levels should be resolved by rejecting the requests from the later. Additional features were suggested to monitor and restrict the actions from low-level priority members. *In summary, our design for multi-device multi-user aware access control system,* KRATOS, *was heavily influenced by the needs of the users and* KRATOS *aims to achieve all of these desires from the users into design.*

7.4 Problem Scope

In this section, we first define several important terms that we use in this work. Then, we introduce the challenges of an access control system in the smart environment through an example scenario in a smart home system. Finally, we articulate the threat model considered in this work.

7.4.1 Terminology

We define several important terms that we use in this work.

Policy. In this work, we consider *Policy* as the group of requests made by the users to control device usage and parameters in a multi-user smart environment. Based on the nature of request, there are three types of policies considered in KRATOS.

1. *Demand Policy.* We consider *Demand Policy* as the group of requests made by a user or that define the control rules/configuration for a specific device or group of devices in the smart environment. Demand policies can be general (i.e., created by the admin and applied to all the authorized users in the system) or specific to

a certain user. If a demand policy is general to all users, we define that policy as *General Policy.*

2. *Restriction Policy.* We consider *Restriction Policy* as the set of rules that govern the accessibility and level of control of a user or group of users to a certain device or group or devices in the smart environment. Restriction policies regulate (1) what devices the user has access to, (2) the time frame in which the user is authorized to use/control the devices, and (3) the control parameter limits.

3. *Location-based Policy* We consider *Location-based Policy* as the set of automation rules enforced to an authorized user that are only applicable if the user is connected in the system network. Location-specific policies regulate (1) what devices the user has remote access to and (2) the control setting limit if a specific user is not present in the network.

Priority. We define *Priority* as the importance level of a user that may be used to provide controls for users of higher priority over users with lower priority during different smart system functionalities such as new user addition, policy enforcement, restriction, and demand negotiation processes. In Section 7.5, we detail the different priority levels considered in this work.

Conflict. For the purpose of this work, *Conflict* is defined as the dispute process that is generated from two or more demand policies assigned by the users that interfere or contradict based on the specific requests of the policies. Based on the nature of the demand and restriction policies, three types of conflict can occur.

1. *Hard conflict.* A hard conflict occurs when demand policies of a specific device enforced by two different users do not have any overlapping device condition.

2. *Soft conflict.* When demand policies enforced by the users for a specific device have overlapping device conditions, a soft conflict occurs.

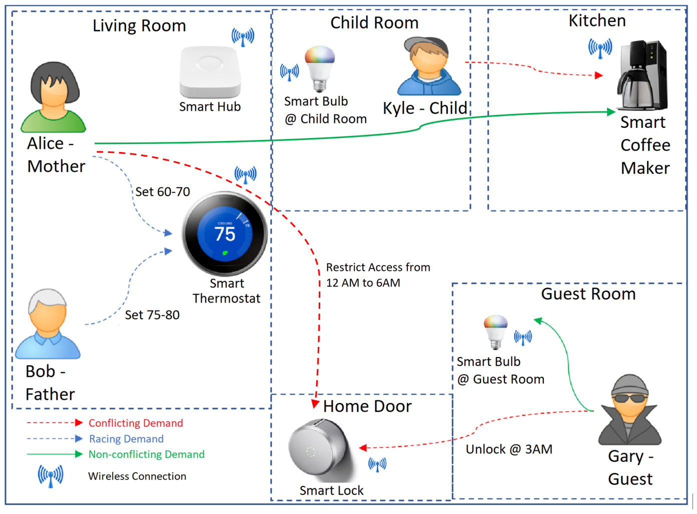


Figure 7.1: Sample smart home with multiple users attempting to control multiple devices with conflicting demands.

3. *Restriction conflict.* If the restriction policy for a specific device is being disputed by the restricted user, KRATOS identifies that as restriction conflict.

7.4.2 Problem Definition

To explain the problem of coarse-grained access control in smart settings, we consider a use-case scenario in smart home system. We assume a smart home setting (S) similar to the one depicted in the Figure 7.1. The smart home has several installed devices to create an automated smart environment. In S, four different users – Bob (father), Alice (mother), Kyle (child), and Gary (guest) interact with the devices. We assume Bob and Alice are the owners of the smart devices and all four users have access to the smart home system through their controller app (installed on their smartphone or tablet). Here, the term *access to the smart home system* refers to the ability to control the devices, configure the system (add/delete devices), and add new users to the system. We assume that the users are performing the following activities which result in conflicting demands- (1) Bob and Alice are trying to configure the smart thermostat to different temperature values at the same time which results in conflicting demand, (2) Alice tries to restrict Kyle from using the smart coffee machine but smart home system does not allow her, (3) Alice wants to set the smart lock after midnight, but (4) Gary wants to enter the home after midnight which results in conflicting demand, and finally, (5) Gary wants to add his friend Steve in the system who is unknown to Bob and Bob can not restrict the activity. Hence, a new access control system is needed and should be designed to answer the following questions: (1) How Bob and Alice can solve their conflicting demand and use the thermostat simultaneously? (2) How Alice can restrict a specific device for a specific user? (3) How Alice can give exclusive permission to use the smart lock to Gary after midnight? (4) How Bob can limit the access of Gary to add a new user? To address these, we propose Kratos, a fine-grained access control system for the smart home that allows users to resolve the conflicting access control demands automatically, add new users, select specific devices to share, limit the access to specific users, and prevent undesired user access in the system.

7.4.3 Threat Model

Kratos considers undesired access control decisions that may arise from existing coarse-grain solutions. For instance, a new user automatically gets full access to the system (i.e., over privileged control) which may lead to *undesired device access*. Also, Kratos considers legitimate smart home users *trying to change the system settings without authorization* (e.g., overriding existing system by installing new apps) that may result in undesired device actions such as installing unknown apps and overriding device conditions (i.e., privilege abuse), even deleting device owners from the system (i.e., privilege escalation). Furthermore, Kratos considers threats that arise from *inadequate*, *inaccurate*, or *careless access control* to multi-user multi-device smart

Threat	Attack Method	Attack Example
Threat-1	Over privileged controls	A newly added smart home user gets the access to use all the connected devices which can lead to undesired activities in the smart home system.
Threat-2	Privilege abuse	A newly added smart home user can abuse the granted privilege to perform *malicious activities* in the smart home system.
Threat-3	Privilege escalation	A newly added smart home user can use the legitimate permissions to remove devices and apps, change device settings, or make a device unavailable to the owner.
Threat-4	Unauthorized access	A temporarily added smart home user can have an access to the smart home system if the owner forgets to delete the access manually.
Threat-5	Transitive privilege	A newly added user adds a new user in the system who automatically gets the same privilege level as the owner and may utilize this transitive privilege to perpetrate his/her exploits.

Table 7.2: Summary of the threat model considered for Kratos.

home (i.e., transitive privilege). In fact, access to a smart environment/platform granted to unknown parties by an authorized user other than the owner may escalate to additional threats (i.e., unauthorized device access), that Kratos also considers as malicious activity. Also, if a temporary guest is not timely removed from the system by the authorized user, it may lead to malicious activities such as sensitive information leakage. Summary of these threats are given in Table 7.2

We do not consider any unauthorized user access due to malicious apps installed in the system. We also assume that the smart environment is not compromised, which means no malicious user is added automatically at the time of system installation as they are different problems from the contributions of Kratos.

7.5 KRATOS Design

In this section, we present the architecture of the Kratos and its main components. Kratos is a fine-grained access control system for multi-user multi-device smart environment where users can express their conflicting demands, desires, and restrictions through policies. Kratos allows an authorized user to add new users in the system and enforce different usage policies to smart devices based on the needs of users and the environment. Kratos considers all the enforced policies from authorized users and offers a policy negotiation algorithm to optimize and solve conflicts among users. In designing the Kratos framework, we consider the following design features and goals.

User-friendly Interface. An access control system should have a user-friendly interface to add or remove users and assign policies in the smart environment. We integrate Kratos into the mobile app provided by smart platform vendors to provide a single user interface to manage users and assign device policies in the installed devices.

Diverse User Roles/Complex Relations. In a smart environment, users have different roles that an access control system needs to define. For example, a user having a parent role should be able to express controls on a user with a child role, while adults in the same priority class should be able to negotiate the access control rules automatically. To address this design feature, Kratos introduces user priority in the system to define user roles.

Conflict Resolution. As discussed earlier, diverse needs in device usage result in conflicts among users in a shared smart environment. The main challenge of an access control system in a smart environment is to resolve these conflicts in a justified way. In addition, users in a multi-user multi-device smart environ should agree with the outcome of conflict resolution provided by the access control system. Kratos uses a novel policy negotiation system to automatically optimize and resolve the conflicting demands among users and institute a generalized usage policy reflecting the needs of all the users. Additionally, Kratos notifies the users the results of the policy negotiation system.

Policy Expiration. In a multi-user multi-device smart environment, temporary access for different smart devices might be needed for guests, occasional, or temporary users. To automate the temporary access to the users, KRATOS offers policy expiration time in assigned devices. KRATOS automatically deletes the device access and the users from the system after the expiration period to avoid undesired access in the system.

Location-based Access. Smart devices and platforms offer remote access and users can control the devices within the smart environment. KRATOS uses the location of the users to trigger a user-specific device policy to limit device usage and meet diverse needs of the users. For example, the parents may want to restrict remote access of the kids for specific devices. KRATOS only allows users to access location-restricted devices if he/she is connected to the home network.

Expressive Control. In a smart environment, a user should be able to express the desired device settings easily. An access control system should provide a simple method to the users to express their diverse needs efficiently and correctly. KRATOS introduces a unified policy language that covers different control parameters (e.g., role, environmental, time, device, location-specific expressions) of smart devices and platforms to understand the users' needs and control the devices accordingly.

Unified Policy Enforcement. All user commands [CBS+18] to the smart devices should go through an *unified access control enforcement* layer to provide fine-grained access control in smart devices and platforms. KRATOS uses an execution module that checks all enforced policies generated from user-assigned policies before executing a command in the smart environment.

Figure 7.2 shows the architecture of the KRATOS system. KRATOS includes four main modules: (1) user interaction module, (2) back-end module, (3) policy manager, and (4) policy execution module. First, the *user interaction module* provides a simple user interface to add new users and assign priorities based on the user's

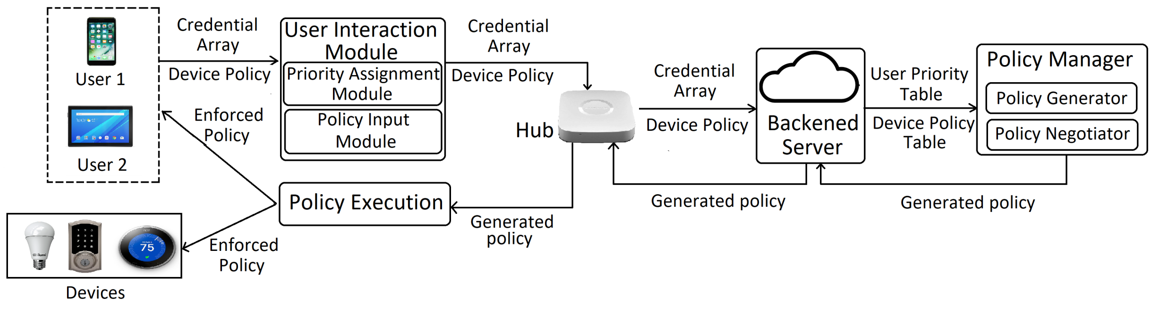


Figure 7.2: Architecture of KRATOS system.

role and preferences. This module also collects user-defined device policies for smart devices. These device policies and priority assignment data are forwarded to the *back-end module* via the hub or edge devices. Back-end module captures these data and creates user priority and device policy list for the users by analyzing the user and device information. The *policy manager* module gathers user priorities and device policies from the generated lists and triggers policy generation and negotiation process to resolve any identified conflicts. After successful negotiation, KRATOS generates final policies and forwards the policies to policy execution module which enforces the policies to associated smart devices and apps. The following subsections details each module in KRATOS and explains how policy generation and negotiation processes are initiated by KRATOS.

7.5.1 User Interaction Module

The user interaction module collects priority assignment data and device policies from the users using a simple user interface. It includes two sub-modules: priority assignment and policy input .

Priority Assignment Module.

The priority assignment module operates as a user interface to add new users and assign priorities. KRATOS introduces a formal format to specify new users, illustrated as follows: $U_a = [A_{id},\ N_{id},\ P,\ D,\ T]$, where, A_{id} is the unique ID of the commanding user, N_{id} is the new user ID that is added in the system, P is the priority level of the new user, D is the permission to add or remove devices from the system, and T is the validity time of the new user in the system. The user priority level is used in the policy generation module to initiate policy negotiation process and resolve conflicting demands. For adding a new user and assigning priorities, we consider the following rules to avoid conflicts in the priority assignment.

- Each user has an authority to add new users and assign a priority.
- The Owner of the device/system will have the highest priority in the system.
- Priority in the system is depicted with a numerical value. The lower the priority of a user, the higher is the level of priority. For example, the owner of the device has the priority of "0".
- Each user can only assign the same or higher value of the priority to a new user, e.g., a user with a priority of "1" can only assign priority of "1" or higher to a new user.
- If two existing users add the same new user with a different priority level, the user with a higher priority level gets the privilege to add the new user.
- If two existing users with the same priority level assigned different priority levels to a new user, the system notifies the existing users to fix a priority level of the new user.
- Each user can assign permissions for adding or removing devices to a new user if the commanding user has the same permission.

@U_1: Alice	
U_2–	restrict smart thermostat between 60-70 degrees.
U_3–	restrict access to smart coffee maker.
U_3–	allow access to smart bulb at the child room if U_4 is home.
U_4–	allow access to smart bulb at the guest room.
U_4–	restrict access to smart lock at the home door from 12:00 AM to 6:00 AM.
@U_2: Bob	
U_1–	restrict smart thermostat between 75-80 degrees.
U_3–	allow access to smart bulb in the child's room between 7:00 PM and 7:00 AM
U_4–	allow access to smart lock at the guest room and the home door
@U_3: Kyle	
U_1–	desires to access smart bulb at the child room
U_1–	desires to access coffee maker
@U_4: Gary	
U_1–	desires to access smart lock at the guest room and the home door at 3 AM
U_1–	desires to access the smart bulb in guest room

Figure 7.3: An example demand and restriction requirements of users in Figure 7.1.

The priority assignment of Kratos can also be configured to define the roles of the users. For example, the smart home system in Figure 7.1, Alice and Bob (parents) can be assigned to priority 0, Gary (guest) can be assigned to priority 2, and Kyle (child) can be assigned to priority 3. We use this priority list to explain the functions of Kratos throughout the paper. In Kratos, administrator or homeowner obtains the privilege to define the priority-role mappings in the system. Kratos also allows the users to add temporary users by specifying validity time (T) of a user in the system. After the specified validity time, Kratos removes the user from the system automatically preventing any unauthorized access from a temporary user or guest.

Policy Input Module and Access Policy Language.

Policy input module provides an interface to the users for assigning policies in smart devices. Authorized users can choose any installed smart device and create device policies using policy input module. To define the device policies, Kratos introduces a formal access control policy language for the smart environment to express complex and diverse user preferences (e.g., users' demands, desires, and restrictions) by utilizing an existing open-source smart environment ecosystem (e.g., Samsung Smart-

Things). Each user defines a policy about their preferences for installed smart devices and any restriction over others' accesses in the system. For instance, sample policies for the smart home of four users shown in Figure 7.1, where each user defines her requirements for other users in a smart home with the thermostat, bulbs, lock, and coffee maker, are shown in Figure 7.3. This criteria defined by the users are used throughout this sub-section to construct their policies.

Policy Structure.

Kratos represents the policies as collections of clauses. The clauses allow each user to declare an independent policy for their demands and other users. The clauses have the following structure: $\langle$users$\rangle$: $\langle$devices$\rangle$: $\langle$conditions$\rangle$: $\langle$actions$\rangle$. The first part of the policy is *users*, which includes the information of both policy assigner and assignee. The second part, *devices*, specifies the device or a list of devices included in this statement. Kratos uses device ID assigned by the smart platform to distinguish device-specific policies in a multi-device environment. The third part, *conditions*, is a list of device conditions defining different control parameters (time-based operation, values, etc.) based on the capabilities of the smart devices. For instance, a user may define a condition where only a pre-defined range of commands or only a certain time-window is matched. The final part of the policy is $\langle$action$\rangle$ which states the clause type, *demand*, *restrict*, or *location.* We note that the Kratos's policy language allows users to define multiple clauses. For instance, a user may restrict a distinct subset of smart devices for different conditions and different users. A sample policy scenario is illustrated in Figure 7.4.

```
@U₁  restrict ::    : thermostat₁ : temperature ∉ [60 − 70] ;
     restrict :: U₃ : coffeemaker :    ;
     location :: U₃ : bulb₃ : location ∈ [Home];
     demand :: U₄ : bulb₄ :    ;
     restrict :: U₄ : lock₁ : time ∉ [6 : 00am − 9 : 00pm];
@U₂  restrict ::    : thermostat₁ : temperature ∉ [75 − 80] ;
     demand :: U₃ : bulb₃ : time ∈ [7 : 00pm − 7 : 00am];
     demand :: U₄ : lock₁, lock₄ :   ;
  ...                          ...
```

Figure 7.4: Sample policy clauses to partially implement demands and restrictions shown in Figure 7.3.

7.5.2 Back-end Module

The user interaction module collects the user credentials and device policies generated using the access policy language. It then forwards them to the back-end module where these data are stored and formatted for policy generation and negotiation. The back-end module has two functionalities: (1) generating user priority list, and (2) generating device policy list.

User Priority List.

The back-end module collects the credential arrays and creates a database for authorized users and their assigned priorities. All the credential arrays are checked with the priority assignment rules (explained in Section 7.5.1) and sorted as valid and invalid priority assignments. For each invalid priority assignment, the back-end module notifies the users who initiated the priority assignment. The back-end module also checks the validity of the users added in the user priority list based on the specified time in the credential arrays. The back-end module automatically removes user with expired validity and updates the user priority table. A sample priority list is given in Figure 7.5.

Device Policy List.

The back-end module accumulates all the policies assigned by the users and creates a database based on the device ID. As explained in Section 7.5.1, the access policy language assigns a device ID to determine the intended policy for each device. This list is updated each time a user generates a new policy.

7.5.3 Policy Manager Module

The policy manager module collects the user priority list and device policy list from the back-end module and compares different user policies. This module consists of two sub-modules (policy negotiation module and policy generation module) to initiate the policy negotiation and generation processes.

Policy Negotiation Module

The policy negotiation module compares all the user-defined policies and detects different types of conflicts based on user priorities and demands. Similar to traditional RBAC, Kratos uses assigned user roles and priorities to understand the user needs

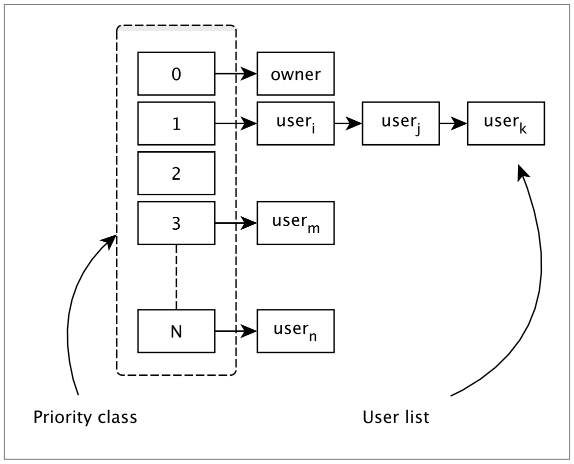


Figure 7.5: A sample user priority list generated by Kratos.

in a smart environment hierarchy. However, a smart environment needs a more fine-grained approach than RBAC to address the conflicting scenarios based on users' relationship, social norms, and personal preferences. To address these diverse needs, KRATOS uses an automatic policy negotiation module to resolve conflicts in multi-user smart environment. The policy negotiation module identifies types of conflicts based on user roles and priorities, categorizes the conflicts based on implemented policies, automatically decides whether a policy should be executed or not, starts a negotiation method between conflicting users using notification methods, and chooses an optimum operating point for both users upon mutual agreement. For policy negotiation, KRATOS considers two essential research questions: (1) How does KRATOS handle the policy conflicts between users with the same and different priority levels?, and (2) How does KRATOS handle restriction policies without affecting smart devices operations?, In the following, we address these questions.

The policy negotiation algorithm of KRATOS processes all the policies and computes the negotiated results by modeling the users' authorities (classes, roles) in a multi-layer list. User authorities are split into ordered classes. Class 0 has the highest priority, and a higher class number means a lower priority. Each class may include a list of users (or roles as roles are just a set of users). Users at the same priority class shares the same priority. KRATOS considers three types of conflicts (hard, soft, and restriction conflicts) between user policies after users are classified into authorities.

When two different policies include clauses of the same user's access for the same device, there can be an *interference* between those clauses. Any such possible interference is further checked to disclose the potential conflicts. In this, hard conflicts can happen when two interfering clauses dictate different actions for some overlapping cases or dictate the same action for never overlapping cased. In other words, when policies have no possible way of cooperation or compromising, KRATOS detects a hard conflict. However, if the same action exists with some common overlap while opposite actions never occur together, such interference is a soft conflict. Moreover, hard and soft conflicts are further categorized as *Priority Conflicts* or *Competition Conflicts* based on the priority of policy owners. When the conflict happens between users' policies who have different priority classes, KRATOS defines a *hard or soft priority conflict*. However, if the users have the same priority, *hard or soft competition conflicts* happens. For hard priority conflicts, KRATOS enforces the policy defined by the user with higher priority. In hard competition conflicts, KRATOS initiates a negotiation process between the users with an average operational condition calculated from both policies. If the users mutually agree with an average operational condition, KRATOS creates a new policy for the targeted device. In case of no mutual operation condition, KRATOS notifies the higher priority user/admin to resolve the dispute with a common policy. In the case of both soft priority and soft competition conflicts, the result of the negotiation process of KRATOS is a new clause with common set of conditions. If any interference is caused by the nature of action requested in two different policies, KRATOS detects a restriction conflict in the system. By incorporating these with hard, soft, and restriction conflicts, KRATOS overall implements five distinct conflict types. (details in following subsections 7.5.3 and 7.5.3).

Policy Negotiation Algorithm

In the policy negotiation algorithm of KRATOS, each policy clause is compiled into a quintuple, $\Psi = \{P, U, D, \mathcal{C}, A\}$, where P is the policy assigner (that shows who states this clause), U is the assignee (about whom this statement is), D is the targeted smart device, $\mathcal{C}$ is a set of conditions over D and U, and configurable environmental attributes, and finally $\mathcal{A} \in \{demand, restrict\}$ is the action requested by this statement when the set of conditions are satisfied. KRATOS implements an algorithm to solve the policy conflicts through a set of equations as follows:

$$interfere(\Psi_i, \Psi_j) \leftarrow U_i = U_j \wedge D_i = D_j \tag{7.1}$$

$$\begin{aligned} hard\_conflict(\Psi_i, \Psi_j) \leftarrow interfere(\Psi_i, \Psi_j) \wedge (\\ (A_i \neq A_j \ \wedge \forall c \in \mathcal{C}_i \cap \mathcal{C}_j : \Theta(\mathcal{V}(c, \mathcal{C}_i), \mathcal{V}(c, \mathcal{C}_j))) \\ \vee(A_i = A_j \ \wedge \exists c \in \mathcal{C}_i \cap \mathcal{C}_j : \neg\Theta(\mathcal{V}(c, \mathcal{C}_i), \mathcal{V}(c, \mathcal{C}_j)))) \end{aligned} \tag{7.2}$$

$$\begin{aligned} soft\_conflict(\Psi_i, \Psi_j) \leftarrow interfere(\Psi_i, \Psi_j) \wedge (\\ (A_i = A_j \ \wedge \forall c \in \mathcal{C}_i \cap \mathcal{C}_j : \Theta(\mathcal{V}(c, \mathcal{C}_i), \mathcal{V}(c, \mathcal{C}_j))) \\ \vee(A_i \neq A_j \ \wedge \exists c \in \mathcal{C}_i \cap \mathcal{C}_j : \mathcal{V}(c, \mathcal{C}_i) \neq \mathcal{V}(c, \mathcal{C}_j))) \end{aligned} \tag{7.3}$$

$$HPC(\Psi_i, \Psi_j) \leftarrow hard\_conflict(\Psi_i, \Psi_j) \wedge \Xi(P_i) \neq \Xi(P_j) \tag{7.4}$$

$$SPC(\Psi_i, \Psi_j) \leftarrow soft\_conflict(\Psi_i, \Psi_j) \wedge \Xi(P_i) \neq \Xi(P_j) \tag{7.5}$$

$$HCC(\Psi_i, \Psi_j) \leftarrow hard\_conflict(\Psi_i, \Psi_j) \wedge \Xi(P_i) = \Xi(P_j) \tag{7.6}$$

$$SCC(\Psi_i, \Psi_j) \leftarrow soft\_conflict(\Psi_i, \Psi_j) \wedge \Xi(P_i) = \Xi(P_j) \tag{7.7}$$

$$\begin{aligned} RC(\Psi_i, \psi_j) \leftarrow Restriction\_conflict(\Psi_i, \psi_j) \wedge \Xi(P_i) > \Xi(P_j) \\ \wedge A_i = restrict \end{aligned} \tag{7.8}$$

where Ψ_i, Ψ_j is the evaluated pair of policies, and $\mathcal{V}(c, C)$ is the value function that returns the value of conditional c in the set C, $\Theta(x, y)$ checks the overlap between the provided (x, y) tuple and $\Xi(u)$ returns the priority of user u as the value of user's assigned priority class.

Policy Negotiation Process.

The negotiation $\mathcal{N}$ between two given policy clauses (Ψ_i, Ψ_j) can be formally expressed and computed by a sample function as in Equation 7.9.

$$\mathrm{N}(\Psi_i,\Psi_j)=\begin{cases}\begin{cases}\Psi_i & \text{if } \Xi(P_i)>\Xi(P_j)\\ \Psi_j & \text{otherwise}\end{cases}, & \text{if } HPC(\Psi_i,\Psi_j)\\ \begin{cases}\{P_i\cup P_j,U_i,D_i,\mathcal{C}_i\cup\mathcal{C}_j,A_i\} & \text{if } A_i=A_j\\ \{P_i\cup P_j,U_i,D_i,\mathcal{C}_i\cup\neg\mathcal{C}_j,A_i\} & \text{otherwise}\end{cases}, & \text{if } SPC(\Psi_i,\Psi_j)\\ \begin{cases}majority\_vote(\Psi_i,\Psi_j) & \text{if } binary(D_i)\\ arbitrate(\Psi_i,\Psi_j) & \text{otherwise}\end{cases}, & \text{if } HCC(\Psi_i,\Psi_j)\\ \begin{cases}\{P_i\cup P_j,U_i,D_i,\mathcal{C}_i\cup\mathcal{C}_j,A_i\} & \text{if } A_i=A_j\\ \{P_i\cup P_j,U_i,D_i,\mathcal{C}_i\cup\neg\mathcal{C}_j,A_i\} & \text{otherwise}\end{cases}, & \text{if } SCC(\Psi_i,\Psi_j)\end{cases} \tag{7.9}$$

Figure 7.6: Negotiation algorithms to resolve conflicts where HPC is hard priority conflict, SPC is soft priority conflict, HCC is hard competition conflict, and SCC is soft competition conflict.

Here, as an example, in the case of a hard priority conflict, (e.g., mother vs. child with contradicting clauses) result is the clause of the user with the higher priority (e.g., mother). For hard competition conflict, both the users are notified with overlapping conditions assigned in the policies and Kratos offers a common operating condition to the users. This common condition is enforced as a policy to the device upon users' agreement. On the other hand, in the case of both soft priority and soft competition conflicts, the result of the negotiation is a new clause with common set of conditions. For restriction conflict, both restricted user and policy assigner are notified and if the policy satisfies conditions in Equation 8, restriction policy is enforced in the device.

Policy Generation Module

The goal of the policy generation module is to construct valid policies that reflect the demands and restrictions of all authorized users based on the device policies generated in the user interaction module. The generated policies are passed to the back-end module and stored in a database. Thereafter, these policies are enforced in smart devices. The negotiated policies computed by the policy negotiation algorithm are converted into enforceable access control rules. The negotiated policy clause, $\Psi = \{P, U, D, \mathcal{C}, A\}$, has a 5-tuple format and is indeed well suited for existing attribute-based access control (ABAC) systems. Thus, Kratos uses an ABAC-like enforcement for the final generated rules. Here, the policy, B, is the set of {action, subject, resource, constraints} tuples for a negotiated device policy. An example of mapping a sample policy to ABAC rule through a transformation function can be illustrated as follows:

$$\begin{aligned} ABAC(\Psi_i)\ \{B \mid \text{action } (B) = \mathrm{A_i} \wedge \text{subject } (B) = \mathrm{U_i} \\ \wedge \text{ resources } (B) = \mathrm{D_i} \\ \wedge \text{ constraints}(B) = \mathrm{T}_{\mathcal{C}_\mathrm{i}}\} \end{aligned} \tag{7.10}$$

where T_C $\{c$ | c satisfies the same conditions of C in mapped attributes into ABAC policy. Here, $ABAC(\Psi_i)$ holds a direct translation of actions, subjects, resources, and constraints. We develop an ABAC-like rule generator that enforces the rules in a control device. The generator is integrated into the hub device as a unified enforcement point.

7.5.4 Policy Execution Module

Policy execution module enforces the final policies generated from the policy negotiation process. Smart devices can be controlled through a controller app (installed in smartphone/tablet) or by installing different device-specific apps in the smart platform (e.g., Samsung SmartThings). Policy execution process appends the generated policies in the smartphone controller app or the installed smart apps. To append the policies, Kratos adds conditional statements to the app source code to enforce the policies. When a user tries to change the state of the device, the app asks the policy execution module to check in the policy table generated by the policy generator. If an acceptable condition is matched, the policy execution engine returns the policy to the app and creates a binary decision (true for the accepted policy and false for the restricted policy) in the conditional branches. Based on the decision enforced by the policy execution engine, the user command in a smart app is executed.

7.6 KRATOS Implementation

We implemented Kratos in Samsung SmartThings platform which has the largest market share in consumer IoT, supports highest number of off-the-shelf smart devices, and open-source apps [Gun17].

7.6.1 Implementation and Data Collection

We setup a smart home system to represent a smart environment and test the effectiveness of Kratos in a real-life setting. We used Samsung SmartThings hub and connected multiple smart devices and sensors to the hub to create a functional smart environment. The complete list of devices in our smart home system is provided in Table 7.3. The setup included four different types of devices: smart light, smart lock, smart thermostat, and smart camera, which are some of the most common smart devices used in smart environment (e.g., smart home system) [Sta17]. We also used three different types of sensors: motion, temperature, and contact sensors to provide autonomous control. Further, we collected data from 43 different smart device users. We selected the participants by conducting an institution-wide open call for participation and flyers for community outreach. We obtained the necessary Institutional Review Board approval for collecting data from real-life smart device users. While selecting participants for our study, we considered several features: (1) owns more than one smart devices, (2) shares smart environment with multiple users (e.g., parents, partners, friends, or housemates), (3) diverse user roles (working adults, housewives, young adults, student, etc.), and (4) beginner level knowledge on using smart devices. The participants were grouped into 14 different groups and asked to choose their roles in a smart environment. First, we recorded different conflicting scenarios experienced by the users and asked them to use Kratos to assign device policies. We investigated several multi-user scenarios for the policy generation and negotiation processes as detailed below:

Device Type	**Model**	**Quantity**
Smart Home Hub	Samsung SamrtThings Hub	1
Smart Light	Philips Hue Light Bulb	4
Smart Lock	Yale B1L Lock with Z-Wave Push Button Deadbolt	1
Smart Camera	Arlo by NETGEAR Security System	1
Smart Thermostat	Ecobee 4 Smart Thermostat	1
Motion Sensor	Fibaro FGMS-001 ZW5 Motion Sensor with Z-Wave Plus Multisensor	6
Temperature Sensor	Fibaro FGMS-001 ZW5 Motion Sensor with Z-Wave Plus Multisensor	1
Door Sensor	Samsung Multipurpose Sensor	2

Table 7.3: Devices and sensors used in our smart home setup to evaluate Kratos.

Scenario 1: Multiple policies for the same device. We selected common devices (e.g., smart thermostat) and enforced different policies set by multiple users. Users assigned demand and restriction policies in the system for the same device. We collected 44 sets of policies (a set of policy includes at least two policies from multiple users) which included 13 hard, 17 soft, and 8 restriction conflicts.

Scenario 2: Multiple policies for different devices. We used multiple devices from the same device category (e.g., smart light, smart lock, smart thermostat) to enforce different policies over the same type of devices. Here, we collected 48 sets of policies from 43 users which resulted in 15 hard, 22 soft, and five restriction conflicts.

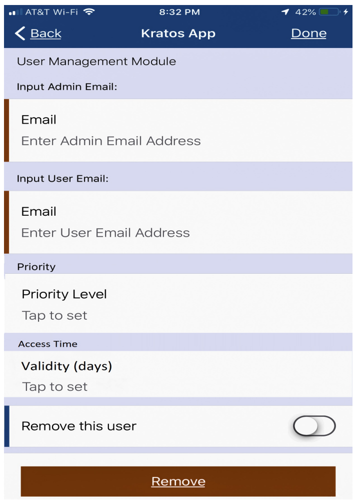
Scenario 3: Multiple apps for the same device. In the smart environment, we allowed users to install different apps to control the same device (e.g., smart light). For example, multiple users can configure a smart light with both motion and door sensors using different apps. We chose three different smart light apps available in SmartThings marketplace (light control with motion sensor, door sensor, and luminance level, respectively) and asked the users to install preferable apps and assign device policies accordingly. Here, we collected 35 sets of policies including 8 hard, 18 soft, and five restriction conflicts.

Scenario 4: Single app for multiple devices. We considered an individual app controlling multiple same types of devices in the smart environment/system. We chose a single light controlling app to control four different lights and asked users to enforce device policies in different devices using one single app. We collected 32 sets of policies in this scenario which includes 12 hard, 15 soft, and 3 restriction conflicts.

Scenario 5: Temporary users in the system. We considered a temporary user/guest is added in the system and trying to access a smart light and smart lock after the access is expired for that specific user. We collected 30 sets of policies in this scenario.

Scenario 6: Location-based access in the system. In the location-based access control, we allowed multiple users to set location-based policies for a smart thermostat. Here, users are allowed to define both location-based restriction and demand policies. We collected 30 sets of policies in this scenario.

Malicious scenarios. We also implemented five real-life threats in our smart environment testbed to generate malicious data and further evaluate the effectiveness of KRATOS (more details in Section 7.7) . For Threat-1 (Over privileged controls), we asked the users to add restriction clauses to the smart thermostat and asked the restricted users to change the temperature. For Threat-2 (Privilege abuse), we asked a newly added user with lower priority to install a new app in the smart system and trigger a smart camera. Threat-3 (Privilege escalation) is presented by a scenario where a new user changed the lock code of a smart lock and removed the smart lock from the environment. For Threat-4 (Unauthorized access), we added a temporary authorized user with limited priority and asked the users to control a smart thermostat outside their accepted time range. For Threat-5 (Transitive privilege), we asked the user with lower priority to add a new user with higher priority in the system.


AT&T Wi-Fi 8:32 PM 42%

Back Kratos App Done

User Management Module

Input Admin Email:

Email
Enter Admin Email Address

Input User Email:

Email
Enter User Email Address

Priority

Priority Level
Tap to set

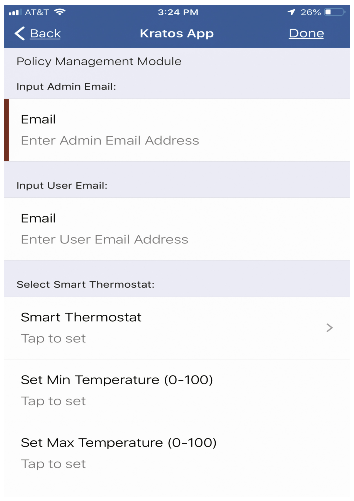
Access Time

Validity (days)
Tap to set

Remove this user

Remove

(a) User Management


AT&T 3:24 PM 26%

Back Kratos App Done

Policy Management Module

Input Admin Email:

Email
Enter Admin Email Address

Input User Email:

Email
Enter User Email Address

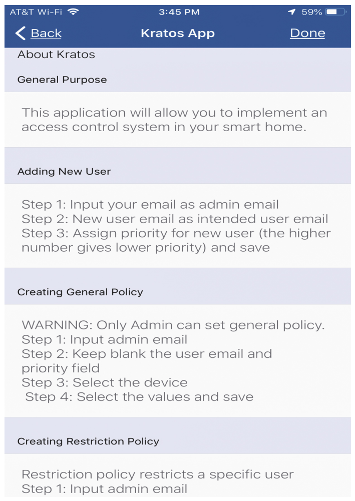
Select Smart Thermostat:

Smart Thermostat
Tap to set

Set Min Temperature (0-100)
Tap to set

Set Max Temperature (0-100)
Tap to set

(b) Policy Management


AT&T Wi-Fi 3:45 PM 59%

Back Kratos App Done

About Kratos

General Purpose

This application will allow you to implement an access control system in your smart home.

Adding New User

Step 1: Input your email as admin email
Step 2: New user email as intended user email
Step 3: Assign priority for new user (the higher number gives lower priority) and save

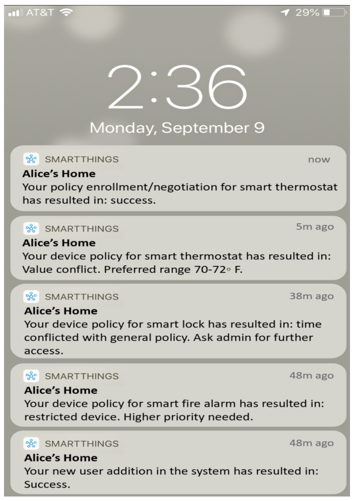
Creating General Policy

WARNING: Only Admin can set general policy.
Step 1: Input admin email
Step 2: Keep blank the user email and priority field
Step 3: Select the device
Step 4: Select the values and save

Creating Restriction Policy

Restriction policy restricts a specific user
Step 1: Input admin email

(c) Instruction Set


AT&T 29%

2:36
Monday, September 9

SMARTTHINGS now
Alice's Home
Your policy enrollment/negotiation for smart thermostat has resulted in: success.

SMARTTHINGS 5m ago
Alice's Home
Your device policy for smart thermostat has resulted in: Value conflict. Preferred range 70-72° F.

SMARTTHINGS 38m ago
Alice's Home
Your device policy for smart lock has resulted in: time conflicted with general policy. Ask admin for further access.

SMARTTHINGS 48m ago
Alice's Home
Your device policy for smart fire alarm has resulted in: restricted device. Higher priority needed.

SMARTTHINGS 48m ago
Alice's Home
Your new user addition in the system has resulted in: Success.

(d) Notification System

Figure 7.7: User interfaces of Kratos

7.6.2 User Interface

We built a SmartThings app that represents the user interaction module described in Section 7.5. This app has two modules: user management and policy management. The user management module allows users to add new users and assign priorities. In addition to adding a new user and assigning priorities, our implemented system allows the owner/admin to define role-based priority levels to the different users. We define five different roles and priority levels in Kratos (i.e., father/owner - priority

0, mother/owner - priority 0, adult - priority 1, guest - priority 2, child - priority 3). These roles and priorities can be assigned by the smart home owner or by authorized users with the same or higher priority to the one being assigned. Upon created a new role/priority, the information is sent and stored in the backend server. In the policy management module, users select devices and create new policies. KRATOS provides options to add either general device policies (intended for all existing users) or policies that apply only to specific users. KRATOS allows users to use different device conditions (operation-based, time-based, value-based, etc.) to define the policies. As our implementation environment had devices that only allows time-based and value-based conditions, we classified the policies in three different possible categories: (1) time-based device policy, (2) value-based device policy, and (3) time-value-based policy. The policies for different devices in our implementation can be represented by a device policy array: Device Policy, P = $\{U,\ D,\ C_1,\ C_2,\ R\}$.

• *User ID (U):* The first element of the policy array is to identify the policy assignee. We utilized the user email as a personal identifier in our implementation.

• *Device ID (D):* SmartThings assigns a unique device ID for each installed device which was used for the devices and policies.

• *Time conditions (C_1):* Users could assign a start time and an end time for any device action in the policies. For example, a smart light can be accessed from sunset to sunrise only.

• *Value conditions (C_2):* Users could assign a maximum and minimum value to specify an acceptable range to control a device functionality. For example, a user can set the operational range of a thermostat from 68°F to 70°F.

• *Restricted User (R):* High-priority Users could define the restriction policy for a specific lower-priority user by adding the user ID to the restricted user's list. Users could also assign general policies (Section 7.4.1) for the devices by assigning '0' in this field.

Figure 7.7 shows the user interface of KRATOS. We implemented KRATOS as a customized smart app in Samsung SmartThings platform. We built the KRATOS app in *Groovy* platform and installed the app using SmartThings web interface. As Samsung allows each users to install customized apps in same smart environment using the web interface, KRATOS app can be easily installed in each user's controller device in multi-user smart environment. Each user can use official SmartThings app in the controller device to use KRATOS app to assign new users and device policies. The information of new users and device policies are forwarded to the policy generator via the backend server for generating final device policies.

7.6.3 Policy Enforcement

The final step during implementation is to enforce the generated policies by KRATOS. We utilized 10 different official SmartThings apps that control 17 different devices and installed them in the smart home/environment. We installed all the apps and observed the user-specific policies generated in the policy generation module. We modified these apps to connect with the backend server and capture the generated policies from the policy generator. These policies were appended to the conditional statements inside the app to execute the policies. A sample modified app is given below to illustrate the steps to enforce policies in a SmartThings app.

Listing 7.1: Policy enforced at install-time

```
definition(
    name: "Big Turn ON modified",
    namespace: "smartthings",
    author: "Anonymous",
    description: "Turn your lights on when the SmartApp is tapped.",
    category: "Convenience",
    iconUrl: "https://s3.amazonaws.com/smartapp-icons/Meta/light_outlet.png",
    iconX2Url: "https://s3.amazonaws.com/smartappicons/Meta/light@2x.png"
)
import groovy.time.*
preferences {
        section("When I touch the app, turn on...") {
```

```
        input "switches", "capability.switch", multiple: false
        input name: "email", type: "email", title: "Email", description: "Enter Email Address", required: true,
            displayDuringSetup: true}}
def installed()
{ atomicState.SmartLightTimes = [:]
        atomicState.SmartLightAdmins = [:]
        atomicState.SmartLightUsers = [:]
        atomicState.SmartLightDevID = [:]
        atomicState.SmartLightTimeStart = [:]
        atomicState.SmartLightTimeEnd = [:]

    log.debug "${new Date()}"
        getSmartLightJsonData()

    def item = atomicState.SmartLightUsers.indexOf(email)
    if (item>=0){
        int index = atomicState.SmartLightUsers.indexOf(email)
        def between = timeBetween (atomicState.SmartLightTimeStart[index], atomicState.SmartLightTimeEnd[
            index])
        if (between == true){
            subscribe(location, changedLocationMode)
            subscribe(app, appTouch)
            log.info app.getAccountId()}}
}
def updated()
{ atomicState.SmartLightTimes = [:]
        atomicState.SmartLightAdmins = [:]
        atomicState.SmartLightUsers = [:]
        atomicState.SmartLightDevID = [:]
        atomicState.SmartLightTimeStart = [:]
        atomicState.SmartLightTimeEnd = [:]
    getSmartLightJsonData()

    def item = atomicState.SmartLightUsers.indexOf(email)
    if (item>=0){
        int index = atomicState.SmartLightUsers.indexOf(email)
        def between = timeBetween (atomicState.SmartLightTimeStart[index], atomicState.SmartLightTimeEnd[
            index])
        if (between == true){
            unsubscribe()
            subscribe(location, changedLocationMode)
            subscribe(app, appTouch)}}
}
def changedLocationMode(evt) {
        log.debug "changedLocationMode: $evt"
        switches?.on()}
def appTouch(evt) {
        log.debug "appTouch: $evt"
        switches?.on()}
def getSmartLightJsonData(){
    def listTimes = []
    def listAdmins = []
    def listUsers = []
    def listIDs = []
    def listTimeStarts = []
    def listTimeEnds = []
    def params = [uri: "https://mywebserver/xxxyyyzzz/2/public/values?alt=json",]
    try {
        httpGet(params) { resp ->
            for (object in resp.data.feed.entry){
                                listTimes.add (object.gsx$time.$t)
                listAdmins.add (object.gsx$adminemail.$t)
                listUsers.add (object.gsx$restricteduseremail.$t)
                listIDs.add (object.gsx$deviceid.$t)
                listTimeStarts.add (object.gsx$timerangestart.$t)
                listTimeEnds.add (object.gsx$timerangeend.$t)
            }
            atomicState.SmartLightTimes = (listTimes)
            atomicState.SmartLightAdmins = (listAdmins)
            atomicState.SmartLightUsers = (listUsers)
            atomicState.SmartLightDevID = (listIDs)
            atomicState.SmartLightTimeStart = (listTimeStarts)
            atomicState.SmartLightTimeEnd = (listTimeEnds)}
    } catch (e) {
        log.error "something went wrong: $e"}
}

def timeBetween(String start, String end){
    long timeDiff
    def now = new Date()
    def timeStart = Date.parse("yyy-MM-dd'T'HH:mm:ss","${start}".replace(".000-0400",""))
    def timeEnd = Date.parse("yyy-MM-dd'T'HH:mm:ss","${end}".replace(".000-0400",""))
    long unxNow = now.getTime()
    long unxEnd = timeEnd.getTime()
    long unxStart = timeStart.getTime()
    if (unxNow >= unxStart && unxNow <= unxEnd)
        return true
    else
        return false
}
```

7.7 Performance Evaluation

We evaluate Kratos by focusing on the following research questions:

RQ1 How effective is Kratos in enforcing access control in multi-user scenarios while handling different threat models? (Sec 7.7.1)

RQ2 What is the overhead introduced by Kratos on the normal operations of the smart environment/setting? (Sec. 7.7.2)

7.7.1 Effectiveness

In this sub-section, we present the experimental results of Kratos while enforcing access control in different multi-user smart environment scenarios and threat models. We first considered a use case scenario to explain the results of Kratos in different smart device operations. Then, we considered six different utilization scenarios (explained in Section 7.6) to evaluate the effectiveness of Kratos.

To understand the performance of Kratos, we assume two users Alice and Bob using the same smart thermostat and assigning different policies according to their needs. This usage scenario may lead to conflicts in which case Kratos uses policy negotiation module to solve the conflicts. For instance, let us assume Alice and Bob has the same priority level which is 2 and assign temperature range 60-70 and 75-80 respectively. Kratos considers this as a hard competition conflict and starts the negotiation process with average range 67-75. If Alice and Bob both agree with

Conflict type	Policy example	Kratos outcome
Hard priority conflict	Alice (priority-1) and Bob (priority-2) set up the temperature range 60-70 and 75-80 respectively in the smart thermostat.	As Alice has higher priority, KRATOS sets the thermostat to 60-70 and notifies the users with the decision
Soft priority conflict	Alice (priority-1) and Bob (priority-2) set up the temperature range 60-70 and 65-75 respectively in the smart thermostat.	• As Alice has the higher priority, KRATOS sets the thermostat to 60-70 and notifies Alice with common range (65-70). • If Alice agrees with common range, KRATOS sets the temperature range 65-70.
Hard competition conflict	Alice (priority-2) and Bob (priority-2) set up the temperature range 60-70 and 75-80 respectively in the smart thermostat.	• KRATOS starts the negotiation with average range (67-75) and upon mutual agreement from the users set the range. • If the users fail to agree, KRATOS notifies higher level user/admin to decide the policies.
Soft competition conflict	Alice (priority-2) and Bob (priority-2) set up the temperature range 60-70 and 65-75 respectively in the smart thermostat.	KRATOS sets the temperature range 65-70 and notifies the users with updated policy.
Restriction conflict	Alice (priority-1) set the temperature range 60-70 and restrict Bob (priority-2) to change the thermostat. Bob sets the temperature range 75-80.	KRATOS sets the temperature range 60-70 and notifies Bob regarding restriction.
Temporary access	Alice (priority-1) added Gary (priority- 4) as a temporary user for 2 days. After 2 days, Gary tries to unlock the smart lock.	KRATOS automatically detects the expired validity for smart home access and deletes Gary from authorized user list to prevent any undesired access.
Location-based access	Alice (priority-1) set up the temperature range 70-72 and restrict Kyle (priority-3) from using the smart thermostat remotely. Kyle sets the temperature range 74-76.	• If Kyle is not in the home network, KRATOS disregard Kyle's access policy. • KRATOS checks the location of both Kyle and Alice. If only Kyle is home, KRATOS sets the temperature range 74-76. If both Kyle and Alice are home, KRATOS sets the temperature range 70-72.

Table 7.4: Different usage scenarios and outcomes of KRATOS.

the range, KRATOS generates a new policy for the thermostat with the temperature range 67-75 and enforces this in the device. On the other hand, if Alice and Bob cannot agree, KRATOS notifies a higher level user/admin to resolve this conflict by assigning a new policy for the device. We also consider a temporary user scenario in evaluating KRATOS where Alice (priority-1) adds a temporary user Gary (priority-4) in the system for 2 days. After the validity period (2 days), Gary tries to access the smart devices. However, KRATOS automatically detects any expired validity of the users in the system and restricts the temporary users to access the system. Table 7.4

summarizes the outcome of KRATOS in different usage scenarios. Table 7.5 also shows the summary of policy conflicts and negotiations between smart device users in different multi-user scenarios explained in Section 7.6. In Scenario-1, KRATOS successfully negotiated 44 sets of policies collected from 43 users and executed the generated policies in the smart environment. Average policy generation time including the policy negotiation was 0.68 seconds. In Scenario-2, KRATOS evaluated 48 sets of policies in total with an average policy generation time of 1.2 seconds. In Scenario-3 and 4, KRATOS manages 35 and 32 sets of policies with an average generation time of 0.86 and 0.48 seconds respectively. In Scenario-5, KRATOS successfully manages 20 sets of policies and automatically detects unauthorized access for expired temporary access. For location-based access in Scenario-6, KRATOS successfully manages 30 sets of policies and provides location-based acess to multiple users. KRATOS also successfully resolves all the conflicts generated in different scenarios. In summary, KRATOS successfully resolved the policy conflicts and created optimized final policies that could be executed within different smart apps.

We also evaluated the effectiveness of KRATOS in preventing different threats in the smart environment. We considered five different threats presented in Section 7.6. We collected data from fifty malicious occurrences in total to evaluate KRATOS against these threats. Table 7.6 summarizes the performance of KRATOS in identifying different threats. In each of these scenarios, KRATOS detected the policy violation with 100% accuracy and effectively notified the smart homeowner/policy assigner via push

Usage Scenario	No. of policies	No. of hard conflicts	No. of soft conflicts	Restriction policies	No conflicts	Average time (s)	Success rate (s)
Scenario-1	44	13	17	8	6	0.68	100%
Scenario-2	48	15	22	5	6	1.2	100%
Scenario-3	35	8	18	5	4	0.86	100%
Scenario-4	32	12	15	3	2	0.48	100%
Scenario-5	30	-	12	6	12	0.2	100%
Scenario-6	30	10	8	8	4	0.32	100%

Table 7.5: KRATOS's performance in different scenarios.

notifications. For Threat-1, Kratos achieves the lowest average detection and notification time 0.25 and 0.4 seconds respectively. To identify Threat-2 and 3, Kratos takes 0.4 and 0.47 seconds on average with average notification time 0.6 seconds. For Threat-4 and 5, the average detection time is 0.35 and 0.28 seconds, respectively. In summary, Kratos can detect different threats with 100% accuracy and notify users with minimum delay.

Threat model	No. of occurances	Success rate	Average Detection time (s)	Average Notification time (s)
Threat-1	10	100%	0.25	0.4
Threat-2	10	100%	0.4	0.6
Threat-3	10	100%	0.47	0.6
Threat-4	10	100%	0.35	0.52
Threat-5	10	100%	0.28	0.45

Table 7.6: Performance of Kratos against different threats.

7.7.2 Performance Overhead

We considered the following research questions to measure the performance overhead of Kratos:

RQ3 What is the impact of Kratos in normal operations of the smart environment/settings? (Table 7.7)

RQ4 What is the impact of Kratos in executing a user command in the smart environment via the smart apps? (Table 7.8)

RQ5 How does the impact of Kratos change with different parameters in the smart environment/setting? (Figure 7.8)

For different multi-user scenarios, we considered four different scenarios as explained in Section 7.6.

Latency Introduced by Kratos. Kratos considers three different types of conflicts (hard conflicts, soft conflicts, and restriction policy) during policy generation and

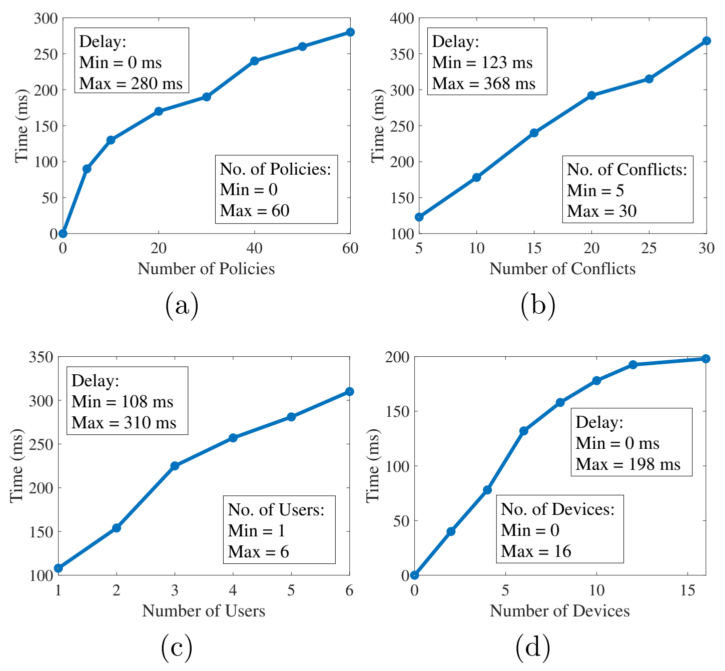


Figure 7.8: Impact of different evaluation parameters on KRATOS's performance: (a) number of policies, (b) number of conflicts, (c) number of users, and (d) number of devices.

negotiation based on user priorities and policy types. These policy generation and negotiation processes normally introduce latency in the normal operations of smart devices and the smart apps to analyze given policies and solving conflicts. Table 7.7 illustrates the delay introduced by KRATOS while handling the policy conflicts and negotiations. We note that the average negotiation time increases with the number of policies for all types of policy conflicts. For hard conflicts, the average negotiation time is 0.403 seconds for ten policies, which increases to 1.21 seconds for 30 policies. Because the hard conflicts require all the conflicted users to interact with the system to resolve the conflicts, it takes more time than soft conflict and restriction policies. For soft conflicts, the average negotiation time is 0.27 seconds for ten policies which increases to 0.73 seconds for 30 policies. For the restriction policies, the latency is introduced only when a low-priority user tries to assign policies to high-priority users.

In this case, average negotiation times vary from 0.102 seconds to 0.25 seconds from 10 to 30 policies.

Conflict types	No. of Policies	Average negotiation time (s)
Hard conflict	10	0.403
	20	0.715
	30	1.21
Soft conflict	10	0.27
	20	0.53
	30	0.73
Restriction Policy	10	0.102
	20	0.117
	30	0.25

Table 7.7: Overhead of KRATOS in handling policy negotiations.

Impact of Kratos on Executing User Commands. As the policies in KRATOS are enforced in the smart apps installed via the controller device (e.g., smartphone and smart tablet), it introduces overhead in the controller devices while installing the apps and executing users' command. Table 7.8 depicts the impact of KRATOS on executing user commands based on generated policy. Here, we used eight different apps to measure the performance overhead of KRATOS. We also considered three types of constraints on the policies: time constraint, value constraint, and both time and value constraints. Time constraint refers to the specific time range for the desired action of a smart device (e.g., turning on lights at sunset) while value constraint refers to the specific range of inputs to a smart device (e.g., the temperature of the smart thermostat). With no policy enforced on a device, the average time to install an app and execute user command is 1.3 seconds with 1.75% and 1.6% of CPU and RAM utilization, respectively. For time constraints and value constraints, the average time is 1.72 and 1.46 seconds, respectively. Average CPU and RAM utilization are almost similar for both time and value constraints (2.1-2.2% and 2.25-2.6%, respectively). For both time and value constraints, the average execution time increases to 1.92 seconds. The CPU and RAM utilization also increases to 2.5% and

2.82%, respectively. Considering the CPU and RAM available in modern smartphones and tablets, the overhead introduced by Kratos can be considered negligible [SAU17, SAU19a, SAU19b].

Type of policy	Avg. time (s)	Avg. CPU usage	Avg. RAM usage
No policy	1.3	1.75%	1.6%
Time constraint	1.72	2.2%	2.6%
Value constraint	1.46	2.1%	2.25%
Time and Value constraint	1.92	2.5%	2.82%

Table 7.8: Overhead of Kratos in policy executions.

Impact of Different Parameters on Performance Overhead. Kratos considers different parameters in smart environment to define and execute device policies reflecting diverse user demands. Here, we observed the performance overhead of Kratos by changing various parameters. As policy generation and negotiation are executed at the backend server, Kratos does not pose any performance overhead to computational parameters (CPU and RAM utilization). The only noticeable change is observed in delay imposed by Kratos in the normal operation of the smart devices and apps. In Figure 7.8, the delay introduced by Kratos is shown based on the number of policies, conflicts, users, and devices. One can notice from Figure 7.8(a), the delay introduced by Kratos increases with the number of policies generated by the users. Kratos introduces 90 ms delay in the operation for five policies to execute a user command which increases to 280 ms delay for 60 policies. The delay increases linearly with the number of conflicts and users in the system (Figure 7.8(b) and Figure 7.8(c)). The highest delay to execute a user command is 368 ms, which occurs when the system includes 30 different policy conflicts. Kratos also takes 310 ms to execute a command with six different users presents in the system. This delay is the result of the overhead introduced by notifying different users about executing the command. For the number of devices, the delay introduced by Kratos becomes steady after adding 12 different devices in the smart environment (Figure 7.8(d)).

7.8 Usability Study

To understand the usability of Kratos among users, we also performed a second usability study with 43 smart home users. Again, although it is not the primary goal of this work, it is important to understand the users' perspectives on the usability effectiveness of Kratos.

Again, we obtained Institutional Review Board (IRB) approval and we gave monetary compensation to the users to test our proposed access control system. In this study, users experienced the proposed access control system in a real-life smart environment (smart home system) supported by Samsung SmartThings. We created SmartThings app for Kratos and made it available to the users to install and use it to add new users, add demand policies, add restriction policies for specific users, and experience policy conflict resolution provided. The questions included in the usability study were divided into three different categories:

- *Installation and tutorial:* In this part, users were asked to install the Kratos app in the system and learn how to use Kratos in the smart environment.
- *Policy enforcement and notification system:* In the second part of the usability test, users were asked to create different types of policies (demand and restrict policy) using Kratos and experience the notification system implemented in Kratos.
- *Policy conflict and implementation:* In the last part, users experience the conflict resolution of Kratos and observe the implemented policies in the system.

In the following, we summarize the findings of the usability study and discuss how users took Kratos in a smart environment. A summary of the study is given in Table 7.9.

Installation and tutorial. 95.3% of the users installed the app successfully using the instructions provided in the app and 97.7% of the users thought the provided tutorial was adequate to operate the app and perform different functions successfully.

Category	Rating	Category	Rating
User interface	★★★★★	Processing time	★★★★☆
Tutorial	★★★★★	Policy generation	★★★★★
Installation process	★★★★☆	Conflict resolution	★★★★★
Notification system	★★★★★	User restriction	★★★★★
Availability	★★★★★	User-friendly	★★★★⯪
Ease	★★★★⯪	Effectiveness	★★★★★

Table 7.9: Summary of the usability study of Kratos.

In terms of device availability for policy enforcement, Kratos scored 5 on a scale of 5.

Policy enforcement and notification system. In terms of priority assignment, 93% users understood and correctly added new users in the system. For assigning demand policies, 100% of the users successfully enforced and understood the notifications correctly. 97.7% users understood the notification messages clearly.

Policy conflict and implementation. In this part, users experience how Kratos implemented the generated policies in the system and resolve conflicts between different user demands. Finally, the 97.7% of users were satisfied with the demand policy decisions generated by Kratos while 100% of the users were satisfied with the restriction policy decisions.

7.9 Benefits of KRATOS

In this section, we explain the benefits of Kratos in smart environment using a use-case scenario of smart home. Consider a user, Bob, who defines himself as a technology savvy person and owns a smart home (an example of smart environment). The home is set with devices such as smart lock, thermostat, fire alarm, and smart coffeemaker. Bob's is the head of a family of three members, including his wife Alice, and his teenage son Matt. Finally, Bob is an enthusiastic entrepreneur that offers high-quality vacation rentals to Airbnb users.

Efficient Conflict Resolution. With several devices shared among all household members (including the Airbnb tenant), Bob feels that there is an immediate need for some control mechanism that defines how all the smart devices are being set up and managed among the different users. However, despite trying devices and smart apps from different platforms (e.g., Samsung SmartThings, Google Home, etc.), Bob cannot find a feasible and user-friendly solution that consider the needs of the different users (e.g., Bob and Alice's priority is to keep the thermostat temperature as high as possible while Matt's idea is to have cooler temperature). Kratos offers an access control mechanism for the smart environment that allows Bob to provide access control based on the users' needs and priorities.

Multi-users/Multi-devices. As mentioned before, Bob's setup comprises several different devices with different levels of usability based on their impact on the quality of life of users and their contribution to the general protection and security of the household. Additionally, different users may have different levels of access based on Bob's and the household's best interests. Based on these scenarios, Bob expects an smart home access control system capable of managing multi-user and multi-device environments. Kratos realizes and offers an access control system where the administrator (i.e., Bob) can assign priority levels to the different devices and users. This allows control mechanisms that consider the importance of the various devices, but also the needs of the users based on admin's pre-defined priorities.

Suitability for Complex User Demands. Users' demands can be very complex at times. For instance, in addition to the demands and interests of Bob, Alice, and Matt, new access control policies can be generated in case Bob decides to give some control to his Airbnb tenant Ed. Adding new users and devices to an already configured system increase the complexity due to new conflicts between users and policies. To solve these issues, Kratos can actively analyze and solve policy conflicts through negotiations in an optimized fashion based on the different user and device priorities.

Inherent Security. Bob has certain rules to protect his ecosystem. First, security-related devices (e.g., smart lock) have the highest priority. Second, he would like to have strict and unique control over these devices, so no other user can change their settings or expected behavior. Finally, users with the lowest priority (e.g., Ed) should not be able to add new devices, change device settings, etc. Our framework was designed to provide inherent security based on the specific user's needs. Specifically, Kratos offers the means to provide complex control and demands through comprehensive policy negotiation and conflict resolution.

Intuitive and Easy User Interaction. Finally, Bob desires a user-friendly tool, especially because some users with little technical knowledge may need to interact with the new access control system. Kratos addresses all the steps from gathering users' declared demands and policies to access control enforcement with minimal user interaction. For this, our framework learns from the different priorities from users and devices to create efficient and fully automated policy conflict resolution mechanisms.

Encrypted Sensitive Data. Kratos accumulates user preferences, device usage, and connected users' credentials which can be considered as sensitive data. These data should be encrypted to ensure privacy of the users. Kratos is implemented in Samsung SmartThings which uses encrypted communication channels between smart home devices and the controller devices (smartphone, tablets, etc.). Furthermore, we used encrypted cloud space (Google Cloud) to store the user priority list and generated device policies to ensure user privacy in Kratos.

7.10 Conclusion

In a smart environment (e.g., smart home, smart office, smart factory, etc.), multiple users have access to multiple devices simultaneously. In these settings, multiple users may want to control and configure the devices with different preferences which give rise to complex and conflicting demands. In this chapter, we explored the need of fine-grained access control mechanism in smart environment and developed KRATOS, a fine-grained access control system that addresses the diverse and conflicting demands of authorized users in a shared multi-user multi-device smart environment. KRATOS implements a priority-based policy negotiation technique to resolve conflicting user demands in a shared smart environment. We implemented KRATOS on real-life settings and evaluated its performance through real smart devices in a multi-user setting. KRATOS successfully covers the users' needs, and our extensive evaluations showed that KRATOS is effective in resolving the conflicting requests and enforcing the policies without significant overhead. Also, we tested KRATOS against five different threats and found that KRATOS effectively identifies the threats with high accuracy.

CHAPTER 8

CONCLUDING REMARKS AND FUTURE WORK

In this dissertation, we introduced a comprehensive security framework for smart devices and applications to detect emerging sensor-based threats. Our developed frameworks tackled five main security concerns of sensor-enriched smart devices, applications, and settings. First, we presented a detailed study of sensor-based threats on smart devices and provided taxonomy of existing threats considering attack characteristics, targeted components, attack mechanisms, and impact on smart device operation. Based on this analysis, we developed three security mechanisms to secure sensors in smart devices and applications. We designed a context-aware security framework to detect sensor-based threats in standalone smart devices such as smart phone, smart watch, etc. Further, we developed a platform-independent context-aware security solutions to detect threats in connected smart devices in a smart setting (i.e., smart home). Finally, we presented a fine-grained access control system for multi-user multi-device smart environment to prevent unauthorized and malicious sensor access.

To detect sensor-based threats in standalone smart devices, we introduced 6thSense, a novel context-aware intrusion detection framework. 6thSense observes and learns the changing patterns of the sensors states and correlates with ongoing user activities to detect malicious sensor activities in smart devices. For each user activity, 6thSense learns the sensors patterns and builds a context-aware model. Then, in the detection phase, 6thSense uses different machine-learning techniques to match activity contexts with sensor patterns and detects malicious sensor activities. To test the effectiveness of 6thSense, We implemented 6thSense in an Android-powered smartphone and a smart watch and collected user activity data from 100 different real-life users. Furthermore, we tested 6thSense against three different sensor-based threats. Our extensive evaluation showed that 6thSense is effective in detecting different sensor-based threats with high accuracy and minimal overhead.

The second security mechanisms proposes Aegis, a platform-independent context-aware security framework to detect malicious activities in connected smart environment such as smart home, office, etc. In a smart environment, multiple smart devices and sensors can connect with each other to perform different user-defined tasks collectively. As different user activities in a smart environment triggers a different but specific sensor-device pattern, Aegis correlates these sensor-device relation with app contexts and builds a contextual model to characterize benign user activities. In the detection phase, Aegis checks current states of the smart devices and sensors with learned user behavior and utilizes a Markov Chain-based machine learning technique to detect ongoing malicious activities. We implemented Aegis in different real-life smart home systems and collected data from 20 real-life users. We also considered different smart device settings, platforms, and user-defined policies to test the efficacy of Aegis against five different threats. Our evaluation shows that Aegis can detect different threats in smart environment with high accuracy and F-Score. Furthermore, Aegis yields minimum overhead which makes this solution suitable for real-life deployment.

Finally, to limit unauthorized and malicious sensor access in smart environment, we introduced Kratos, a fine-grained multi-user multi-device-aware access control system. Kratos introduces a formal access control language that allows users to define access policies for smart devices in a shared smart environment. Kratos also offers a novel policy negotiation algorithm that automatically detects conflicting user demands and initiates automatic negotiation by leveraging user roles and priorities. Finally, Kratos monitors assigned user policies and enforces the negotiation results on installed smart devices, sensors, and apps to limit unauthorized access. We implemented Kratos in multi-user smart environment and collected device policies from 72 real-life smart device users. We also tested Kratos against five different threats and achieved 100% accuracy. Furthermore, we performed a detailed usability study with 43 smart device users to understand the users' perceptions on access control systems in multi-user smart environment. Our usability study shows that Kratos resolves diverse demands and achieves an average of 4.6 out of 5 in usability based on user friendliness, ease of use, and effectiveness.

For every security mechanisms presented in this dissertation, we collected data from real-life smart users and improved our design features. We collected user defined configurations and smart device settings in the data collection process of Aegis and tested the effectiveness based on collected user data. Furthermore, as access control systems closely depends on users' demands, we performed both user study and usability study for Kratos. The user and usability study indicates that our designed security frameworks can effectively detect sensor-based threat in smart devices while satisfying user demands as a security framework.

We present several key directions for future research.

- In this dissertation, we presented three unique security mechanisms to secure sensors in smart devices, applications, and settings. However, most of the smart devices are resource-constrained devices and implementing external security mechanisms can affect the normal operation of the devices. Although we implemented all the developed security framework in real-life smart devices and systems, real-life deployment of these frameworks may introduce new implementation challenges such as power-frequency trade-off in resource-limited smart devices, latency in real-time systems, etc. To overcome these challenges, proposed security frameworks should be implemented in an adaptive manner and in-depth investigation should be conducted to minimize the overhead.

- Sensor-based threats are relatively new and very few comprehensive studies are available to understand the characteristics of these threats. This dissertation presents a comprehensive study of existing sensor-based threats and presents different security mechanisms to secure sensors in smart devices and environment. However, more detailed study should be conducted to understand users views on sensor-based threats.
- Smart devices vary in operating system, programming language, and implemented protocols which make it hard for the researchers to develop standard security mechanisms to secure sensors in smart devices. This dissertation presents platform-independent security mechanisms to effectively detect sensor-based threats in different smart devices and systems. However, several smart device platforms are closed source and do not allow any third-party program integration. One future research direction should be standardization of smart devices platforms to implement and test the proposed security framework in different smart devices and systems.
- Finally, although we considered comprehensive threat models for each security mechanisms proposed in this dissertation, we believe that new sensor-based threats can be available in the wild. In future, proposed security mechanisms should be tested against new sensor-based threats available in real-life.

BIBLIOGRAPHY

[AA04] D. Asonov and R. Agrawal. Keyboard acoustic emanations. In *IEEE Symposium on Security and Privacy*, pages 3–11, 2004.

[AARR03] Dakshi Agrawal, Bruce Archambeault, JosyulaR. Rao, and Pankaj Rohatgi. The em side—channel(s). In *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer Berlin Heidelberg, 2003.

[AAUA18] A. Acar, H. Aksu, A. S. Uluagac, and K. Akkaya. Waca: Wearable-assisted continuous authentication. In *IEEE Security and Privacy Workshops (SPW)*, pages 264–269, May 2018.

[act15a] This is the main reason why people want smartwatches. `http://www.businessinsider.com/why-people-want-smartwatches-2015-4`, April 2015.

[act15b] A week in the life analysis of smartphone users. `http://www.pewinternet.org/2015/04/01/`, April 2015.

[AF13] Devdatta Akhawe and Adrienne Porter Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 257–272, Washington, D.C., 2013. USENIX.

[AFA$^+$18] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and A Selcuk Uluagac. Peek-a-boo: I see your smart home activities, even encrypted! *arXiv preprint arXiv:1808.02741*, 2018.

[AHD$^+$16] Ioannis Agadakos, Per Hallgren, Dimitrios Damopoulos, Andrei Sabelfeld, and Georgios Portokalidis. Location-enhanced authentication using the iot: Because you cannot be in two places at once. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications*. ACM, 2016.

[AHIN13] Ahmed Al-Haiqi, Mahamod Ismail, and Rosdiadee Nordin. Keystrokes inference attack on android: A comparative evaluation of sensors and their fusion. *Journal of ICT Research and Applications*, 7(2):117–136, 2013.

[AHR+19] Noah Apthorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. Keeping the smart home private with smart (er) iot traffic shaping. *Proceedings on Privacy Enhancing Technologies*, 2019(3):128–148, 2019.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[Ale18] Amazon Alexa. Understand the smart home skill api, 2018.

[anda] Security for android developers. `https://developer.android.com/topic/security/`. Accessed: 2018-10-23.

[Andb] Sensor overview. `https://developer.android.com/guide/topics/sensors/sensors_overview.html`. Accessed: 2017-10-23.

[andc] Sensor stack. `https://source.android.com/devices/sensors/sensor-stack.html`. Accessed: 2017-03-10.

[and16] Smartphone os market share, 2016 q2. `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`, August 2016.

[and18a] Android 8.0 oreo is now on almost 15 percent of active devices. `https://www.digitaltrends.com/mobile/android-distribution-news/`, August 2018.

[and18b] Android p won't let apps secretly use your mic or camera in the background. `https://www.theverge.com/2018/3/7/17091104/android-p-prevents-apps-using-mic-camera-idle-background`, March 2018.

[Ano16] Anonymous. Voiploc: Compromising location-privacy via acoustic side-channel attacks. *`https://www.semanticscholar.org/paper/VoipLoc-%3A-Compromising-location-privacy-via-attacks/b3a04badcab8e68491277735ceb4dcd12c3e3f71`*, 2016.

[ARC18] Mahmoud Ammar, Giovanni Russello, and Bruno Crispo. Internet of things: A survey on the security of iot frameworks. *Journal of Information Security and Applications*, 38:8–27, 2018.

[ARS+17] Noah Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic. *arXiv preprint arXiv:1708.05044*, 2017.

[ASBS12] Adam J Aviv, Benjamin Sapp, Matt Blaze, and Jonathan M Smith. Practicality of accelerometer side channels on smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 41–50. ACM, 2012.

[AVEF12] Federico Alegre, Ravichander Vipperla, Nicholas Evans, and Benoït Fauve. On the vulnerability of automatic speaker recognition to spoofing attacks with artificial signals. In *2012 Proceedings of the 20th european signal processing conference (EUSIPCO)*, pages 36–40. IEEE, 2012.

[Avi12] Adam J Aviv. *Side channels enabled by smartphone interaction.* PhD thesis, Pennsylvania State University, 2012.

[AWL$^+$19] S Abhishek Anand, Chen Wang, Jian Liu, Nitesh Saxena, and Yingying Chen. Spearphone: A speech privacy exploit via accelerometer-sensed reverberations from smartphone loudspeakers. *arXiv preprint arXiv:1907.05972*, 2019.

[AWS$^+$19a] Eirini Anthi, Lowri Williams, Małgorzata Słowińska, George Theodorakopoulos, and Pete Burnap. A supervised intrusion detection system for smart home iot devices. *IEEE Internet of Things Journal*, 6(5):9042–9053, 2019.

[AWS19b] Amazon AWS. Splitting the data into training and evaluation data, May 2019.

[BAU19] Leonardo Babun, Hidayet Aksu, and A. Selcuk Uluagac. A system-level behavioral detection framework for compromised cps devices: Smart-grid case. *ACM Trans. Cyber-Phys. Syst.*, 4(2), nov 2019.

[BCMU19] 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.

[BDG$^+$10] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic side-channel attacks on printers. In *USENIX Security Symposium*, pages 307–322, 2010.

[BFG$^+$19] Luo Bo, Li Fengjun, Wang Guanghui, et al. *I Know What You Type on Your Phone: Keystroke Inference on Android Device Using Deep Learning.* PhD thesis, University of Kansas, 2019.

[BJD16] Joseph Bugeja, Andreas Jacobsson, and Paul Davidsson. On privacy and security challenges in smart connected homes. In *Intelligence and Security Informatics Conference (EISIC), 2016 European*, pages 172–175. IEEE, 2016.

[BKS15] Sebastian Biedermann, Stefan Katzenbeisser, and Jakub Szefer. Hard drive side-channel attacks using smartphone magnetic field sensors. In *Financial Cryptography and Data Security*, volume 8975 of *Lecture Notes in Computer Science*, pages 489–496. Springer Berlin Heidelberg, 2015.

[bla] Sensors. `https://developer.blackberry.com/native/documentation/device_comm/sensors/`. Accessed: 2017-03-10.

[BN18] Kenneth Block and Guevara Noubir. My magnetometer is telling you where i've been?: A mobile device permission less location attack. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 260–270, 2018.

[BSAU18] Leonardo Babun, Amit Kumar Sikder, Abbas Acar, and A Selcuk Uluagac. Iotdots: A digital forensics framework for smart environments. *arXiv preprint arXiv:1809.00745*, 2018.

[BWY06] Yigael Berger, Avishai Wool, and Arie Yeredor. Dictionary attacks using keyboard acoustic emanations. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 245–254, 2006.

[CA17] Cybersecurity and Infrastructure Security Agency. Mems accelerometer hardware design flaws (update a), 2017.

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

[CC11] Liang Cai and Hao Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. *HotSec*, 11:9–9, 2011.

[CC12] Liang Cai and Hao Chen. On the practicality of motion based keystroke inference attack. In *International Conference on Trust and Trustworthy Computing*, pages 273–290. Springer, 2012.

[CCAF16] Sujit Rokka Chhetri, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. Poster: Exploiting acoustic side-channel for attack on additive manufacturing systems. 2016.

[CCDP04] Vincent Carlier, Hervé Chabanne, Emmanuelle Dottax, and Hervé Pelletier. Electromagnetic side channels of an fpga implementation of aes. In *CRYPTOLOGY EPRINT ARCHIVE, REPORT 2004/145*. Citeseer, 2004.

[CJX+19] Yushi Cheng, Xiaoyu Ji, Wenyuan Xu, Hao Pan, Zhuangdi Zhu, Chuang-Wen You, Yi-Chao Chen, and Lili Qiu. Magattack: Guessing application launching and operation via smartphone. In *Proceedings of the ACM Asia Conference on Computer and Communications Security*, pages 283–294, 2019.

[CMT18] Z. Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated iot safety and security analysis. In *USENIX Annual Technical Conference (USENIX ATC)*, 2018.

[CMT+19] Z. B. Celik, P. McDaniel, G. Tan, L. Babun, and A. S. Uluagac. Verifying internet of things safety and security in physical spaces. *IEEE Security Privacy*, 17(5):30–37, 2019.

[CMV+16] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden voice commands. In *25th USENIX Security Symposium*, pages 513–530, 2016.

[Cof14] Jeff Coffed. The threat of gps jamming: The risk to an information utility. *Report of EXELIS, Jan. Chicago*, 2014.

[COS17] Supriyo Chakraborty, Wentao Ouyang, and Mani Srivastava. Lightspy: Optical eavesdropping on displays using light sensors on mobile devices. In *IEEE International Conference on Big Data*, pages 2980–2989. IEEE, 2017.

[CPG$^+$15] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, and G. Ferrari. Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios. *IEEE Sensors Journal*, 15(2):1224–1234, Feb 2015.

[CWR13] Antorweep Chakravorty, Tomasz Wlodarczyk, and Chunming Rong. Privacy preserving data analytics for smart homes. In *Security and Privacy Workshops (SPW), 2013 IEEE*, pages 23–27. IEEE, 2013.

[CZDY18] Haotian Chi, Qiang Zeng, Xiaojiang Du, and Jiaping Yu. Cross-app threats in smart homes: Categorization, detection and handling. *arXiv preprint arXiv:1808.02125*, 2018.

[Des14] Luke Deshotels. Inaudible sound as a covert channel in mobile devices. In *8th USENIX Workshop on Offensive Technologies*, San Diego, CA, 2014.

[DKJG17] Ali Dorri, Salil S Kanhere, Raja Jurdak, and Praveen Gauravaram. Blockchain for iot security and privacy: The case study of a smart home. In *Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on*, pages 618–623. IEEE, 2017.

[DKL13] Tamara Denning, Tadayoshi Kohno, and Henry M. Levy. Computer security and the modern home. *Commun. ACM*, pages 94–103, January 2013.

[DLZZ14] Wenrui Diao, Xiangyu Liu, Zhe Zhou, and Kehuan Zhang. Your voice assistant is mine: How to abuse speakers to steal information and control your phone. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pages 63–74, 2014.

[DPSKM15] Santos Merino Del Pozo, François-Xavier Standaert, Dina Kamel, and Amir Moradi. Side-channel attacks from static power: when should we care? In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 145–150. EDA Consortium, 2015.

[DZL$^+$17] Soteris Demetriou, Nan Zhang, Yeonjoon Lee, XiaoFeng Wang, Carl A Gunter, Xiaoyong Zhou, and Michael Grace. Hanguard: Sdn-driven protection of smart home wifi devices from malicious mobile apps. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2017.

[fel12] How to ask for permission. In *7th USENIX Workshop on Hot Topics in Security*, Bellevue, WA, 2012. USENIX.

[FG13] Behrang Fouladi and Sahand Ghanoun. Honey, i'm home!!-hacking z-wave home automation systems. *Black Hat*, 2013.

[FGSS19] Zakery Fyke, Isaac Griswold-Steiner, and Abdul Serwadda. Prying into private spaces using mobile device motion sensors. In *17th International Conference on Privacy, Security and Trust (PST)*, pages 1–10. IEEE, 2019.

[FHE$^+$12] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the eighth symposium on usable privacy and security*, page 3. ACM, 2012.

[fHSoHU20] Joint Center for Housing Studies of Harvard University. America's rental housing, 2020.

[Fir19] First.org. Common vulnerability scoring system version 3.1: Specification document, 2019.

[FJP16] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging smart home applications. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 636–654. IEEE, 2016.

[FKK10] Denis Foo Kune and Yongdae Kim. Timing attacks on pin input devices. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 678–680, 2010.

[FPR$^+$16] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. Flowfence: Practical data protection for emerging iot application frameworks. In *USENIX Security Symposium*, pages 531–548, 2016.

[FRJP17] Earlence Fernandes, Amir Rahmati, Jaeyeon Jung, and Atul Prakash. Security implications of permission models in smart-home application frameworks. *IEEE Security & Privacy*, pages 24–30, 2017.

[GD13] Thanassis Giannetsos and Tassos Dimitriou. Spy-sense: Spyware tool for executing stealthy exploits against sensor networks. In *Proceedings* *of the 2Nd ACM Workshop on Hot Topics on Wireless Network Security and Privacy*, pages 7–12, 2013.

[Goo19] Google. Cloud automl, May 2019.

[GPR13] Sergio Gusmeroli, Salvatore Piccione, and Domenico Rotondi. A capability-based security approach to manage access control in the internet of things. *Mathematical and Computer Modelling*, 58(5):1189 – 1205, 2013.

[GPY$^+$07] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Usenix Security*, volume 7, pages 1–16, 2007.

[Gun17] Rachel Gunter. Making sense of samsung's smartthings initiative, 2017.

[HAUA18] Muhammad A Hakim, Hidayet Aksu, A Selcuk Uluagac, and Kemal Akkaya. U-pot: A honeypot framework for upnp-based iot devices. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2018.

[HB18] Duncan Hodges and Oliver Buckley. Reconstructing what you said: Text inference using smartphone motion. *IEEE Transactions on Mobile Computing*, 18(4):947–959, 2018.

[HFH$^+$09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[HFH15] Md Mahmud Hossain, Maziar Fotouhi, and Ragib Hasan. Towards an analysis of security issues, challenges, and open problems in the internet of things. In *IEEE World Congress on Services (SERVICES)*, pages 21–28, 2015.

[HGC$^+$19] Yan Huang, Xin Guan, Hongyang Chen, Yi Liang, Shanshan Yuan, and Tomoaki Ohtsuki. Risk assessment of private information inference for motion sensor embedded iot devices. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2019.

[HGP$^+$18] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. Rethinking access control and authentication for the home internet of things (iot). In *27th USENIX Security Symposium*, Baltimore, MD, 2018.

[HLM$^+$16] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *Proceedings of the 11th ACM on Asia conference on computer and communications security*, pages 461–472. ACM, 2016.

[HON$^+$12] Jun Han, Emmanuel Owusu, Le T Nguyen, Adrian Perrig, and Joy Zhang. Accomplice: Location inference using accelerometers on smartphones. In *Fourth International Conference on Communication Systems and Networks*, pages 1–9. IEEE, 2012.

[HS12] Tzipora Halevi and Nitesh Saxena. A closer look at keyboard acoustic emanations: random passwords, typing styles and decoding techniques. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 89–90. ACM, 2012.

[HSH$^+$13] Ragib Hasan, Nitesh Saxena, Tzipora Haleviz, Shams Zawoad, and Dustin Rinehart. Sensing-enabled channels for hard-to-detect command and control of mobile devices. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, pages 469–480, 2013.

[Hue18] Philips Hue. How to develop for hue: Hue api, 2018.

[iOSa] Apple developer documentation. `https://developer.apple.com/documentation`. Accessed: 2015-12-1.

[iOSb] Core motion. `https://developer.apple.com/documentation/coremotion`. Accessed: 2017-10-23.

[IoT17] IoTBench Repository, L. Babun, Z. Berkay Celik and A. Kumar Sikder. `https://github.com/IoTBench`, 2017. [Online; accessed January-2020].

[JCB16] Maia Jacobs, Henriette Cramer, and Louise Barkhuus. Caring about sharing: Couples' practices in single user device access. In *Proceedings of the 19th International Conference on Supporting Group Work*. ACM, 2016.

[JCP17] William Jang, Adil Chhabra, and Aarathi Prasad. Enabling multi-user controls in smart home devices. In *Proceedings of the Workshop on Internet of Things Security and Privacy*. ACM, 2017.

[JCW$^+$17] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z Morley Mao, Atul Prakash, and Shanghai JiaoTong Unviersity. Contexiot: Towards providing contextual integrity to appified iot platforms. In *Proceedings of The Network and Distributed System Security Symposium*, 2017.

[JLLA15] Zhe Ji, Zhi-Yi Li, Peng Li, and MaoBo An. A new effective wearable hand gesture recognition algorithm with 3-axis accelerometer. In *12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 1243–1247, 2015.

[JNS13] Suman Jana, Arvind Narayanan, and Vitaly Shmatikov. A scanner darkly: Protecting user privacy from perceptual applications. In *IEEE Symposium on Security and Privacy (SP)*, pages 349–363, 2013.

[JPPS11] G Joy Persial, M Prabhu, and R Shanmugalakshmi. Side channel attack-survey. *Int J Adva Sci Res Rev*, 1(4):54–57, 2011.

[Kei12] Julian Keilson. *Markov chain models—rarity and exponentiality*, volume 28. Springer Science & Business Media, 2012.

[KJL20] Hyosu Kim, Byunggill Joe, and Yunxin Liu. Tapsnoop: Leveraging tap sounds to infer tapstrokes on touchscreen devices. *IEEE Access*, 8:14737–14748, 2020.

[KKZK12] Raees Khan, Samee U Khan, Rifaqat Zaheer, and Sharifullah Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *10th International Conference on Frontiers of Information Technology (FIT)*, pages 257–260. IEEE, 2012.

[KLW$^+$19] Sean Kennedy, Haipeng Li, Chenggang Wang, Hao Liu, Boyang Wang, and Wenhai Sun. I can hear your alexa: Voice command fingerprinting on smart home speakers. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 232–240. IEEE, 2019.

[KMP17] Won Min Kang, Seo Yeon Moon, and Jong Hyuk Park. An enhanced security framework for home appliances in smart home. *Human-centric Computing and Information Sciences*, 7(1):6, 2017.

[KPM+18] Deepak Kumar, Riccardo Paccagnella, Paul Murley, Eric Hennenfent, Joshua Mason, Adam Bates, and Michael Bailey. Skill squatting attacks on amazon alexa. In *27th USENIX Security Symposium*, pages 33–47, 2018.

[KPYK18] Ah-Lian Kor, Colin Pattinson, Max Yanovsky, and Vyacheslav Kharchenko. Iot-enabled smart living. In *Technology for Smart Futures*, pages 3–28. Springer, 2018.

[KS16] Muhammad Hassam Khan and Munam Ali Shah. Survey on security threats of smartphones in internet of things. In *2016 22nd International Conference on Automation and Computing (ICAC)*, pages 560–566. IEEE, 2016.

[LBAU17] Juan Lopez, Leonardo Babun, Hidayet Aksu, and A Selcuk Uluagac. A survey on function and system call hooking approaches. *Journal of Hardware and Systems Security*, 1(2):114–136, 2017. Accessed: 04-17-2020.

[LCY+18] Yi Liang, Zhipeng Cai, Jiguo Yu, Qilong Han, and Yingshu Li. Deep learning based inference of private information using embedded sensors in smart devices. *IEEE Network*, 32(4):8–14, 2018.

[LG17] Lg watch sport. `http://www.lg.com/us/smart-watches/lg-W280A-sport`, May 2017.

[LIF18] LIFX. Lifx http api, 2018.

[Liv16] Mobile World Live. Analysis: Mobile world congress 2016 wrap-up, March 2016.

[LL19] Yang Liu and Zhenjiang Li. aleak: Context-free side-channel from your smart watch leaks your typing privacy. *IEEE Transactions on Mobile Computing*, 2019.

[LLC+19] Zhen Ling, Zupei Li, Chen Chen, Junzhou Luo, Wei Yu, and Xinwen Fu. I know what you enter on gear vr. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 241–249. IEEE, 2019.

[LML+10] Nicholas D Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T Campbell. A survey of mobile phone sensing. *IEEE Communications magazine*, 48(9), 2010.

[LPMS12] Mariantonietta La Polla, Fabio Martinelli, and Daniele Sgandurra. A survey on security for mobile devices. *IEEE communications surveys & tutorials*, 15(1):446–471, 2012.

[LRH+09] Yunxin Liu, Ahmad Rahmati, Yuanhe Huang, Hyukjae Jang, Lin Zhong, Yongguang Zhang, and Shensheng Zhang. xshare: supporting impromptu sharing of mobile phones. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*. ACM, 2009.

[LS19] Jessy Lin and Jason Seibel. Motion-based side-channel attack on mobile keystrokes. *https://pdfs.semanticscholar.org/95cb/6a266e7a7319334775d8c89e353adf9b514e.pdf*, 2019.

[LVM09] Ondrej Linda, Todd Vollmer, and Milos Manic. Neural network based intrusion detection system for critical infrastructures. In *International Joint Conference on Neural Networks*, pages 1827–1834. IEEE, 2009.

[LWZ+13] Lingguang Lei, Yuewu Wang, Jian Zhou, Daren Zha, and Zhongwen Zhang. A threat to mobile cyber-physical systems: Sensor-based privacy theft attacks on android smartphones. In *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 126–133, 2013.

[LYC+19] Li Lu, Jiadi Yu, Yingying Chen, Yanmin Zhu, Xiangyu Xu, Guangtao Xue, and Minglu Li. Keylistener: Inferring keystrokes on qwerty keyboard of touch screen through acoustic signals. In *IEEE International Conference on Computer Communications*, pages 775–783, 2019.

[LZCC14] Changmin Lee, Luca Zappaterra, Kwanghee Choi, and Hyeong-Ah Choi. Securing smart home: Technologies, security challenges, and security requirements. In *Communications and Network Security (CNS), 2014 IEEE Conference on*, pages 67–72. IEEE, 2014.

[LZD+15] Xiangyu Liu, Zhe Zhou, Wenrui Diao, Zhou Li, and Kehuan Zhang. When good becomes evil: Keystroke inference with smartwatch. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1273–1285. ACM, 2015.

[M$^+$08] Gerard CM Meijer et al. *Smart sensor systems*, volume 7. Wiley Online Library, 2008.

[MAbM14] Teddy Mantoro, Media A Ayu, and Siti Munawwarah binti Mahmod. Securing the authentication and message integrity for smart home using smart phone. In *Multimedia Computing and Systems (ICMCS), 2014 International Conference on*, pages 985–989. IEEE, 2014.

[MAH$^+$16] Mujahid Mohsin, Zahid Anwar, Ghaith Husari, Ehab Al-Shaer, and Mohammad Ashiqur Rahman. Iotsat: A formal framework for security analysis of the internet of things (iot). In *Communications and Network Security (CNS), 2016 IEEE Conference on*, pages 180–188. IEEE, 2016.

[MAJH16] Anindya Maiti, Oscar Armbruster, Murtuza Jadliwala, and Jibo He. Smartwatch-based keystroke inference attacks and context-aware protection mechanisms. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 795–806, 2016.

[mar] Who leads os share in internet of things era? `https://spectrummattersindeed.blogspot.com/2017/04/who-leads-os-share-in-internet-of.html`. Accessed: 2017-10-23.

[MBN14] Yan Michalevsky, Dan Boneh, and Gabi Nakibly. Gyrophone: Recognizing speech from gyroscope signals. In *USENIX Security Symposium*, pages 1053–1067, 2014.

[MBY$^+$19] J. Myers, L. Babun, E. Yao, S. Helble, and P. Allen. Mad-iot: Memory anomaly detection for the internet of things. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, 2019.

[Met] Metaprogramming. `http://docs.groovy-lang.org/docs/next/html/documentation/core-metaprogramming.html`. [Online; accessed January-2020].

[MHSJ18] Anindya Maiti, Ryan Heard, Mohd Sabra, and Murtuza Jadliwala. Towards inferring mechanical lock combinations using wrist-wearables as a side-channel. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 111–122. ACM, 2018.

[Mic19] Trend Micro. Google play apps drop anubis banking malware, use motion-based evasion tactics, 2019.

[Mic20] Microsoft. Windows IoT core documentation, 2020.

[MJ18] Anindya Maiti and Murtuza Jadliwala. Light ears: Information leakage via smart lights. *arXiv preprint arXiv:1808.07814*, 2018.

[MJHB15] Anindya Maiti, Murtuza Jadliwala, Jibo He, and Igor Bilogrevic. (smart) watch your taps: side-channel keystroke inference attacks using smartwatches. In *Proceedings of the ACM International Symposium on Wearable Computers*, pages 27–30. ACM, 2015.

[MJHB18] Anindya Maiti, Murtuza Jadliwala, Jibo He, and Igor Bilogrevic. Side-channel inference attacks on mobile keypads using smartwatches. *IEEE Transactions on Mobile Computing*, 17(9):2180–2194, 2018.

[MLMK15] Weizhi Meng, Wang Hao Lee, SR Murali, and SPT Krishnan. Charging me and i know your secrets!: towards juice filming attacks on smartphones. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, pages 89–98, 2015.

[MLT$^+$16] Tara Matthews, Kerwell Liao, Anna Turner, Marianne Berkovich, Robert Reeder, and Sunny Consolvo. "she'll just grab any device that's closer": A study of everyday device & account sharing in households. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM, 2016.

[MS12] Saurabh Mukherjee and Neelam Sharma. Intrusion detection using naive bayes classifier with feature reduction. *Procedia Technology*, 4:119–128, 2012.

[MSS15] Dibya Mukhopadhyay, Maliheh Shirvanian, and Nitesh Saxena. All your voices are belong to us: Stealing voices to fool humans and machines. In *European Symposium on Research in Computer Security*, pages 599–621. Springer, 2015.

[MSS16] M. Mohamed, B. Shrestha, and N. Saxena. Smashed: Sniffing and manipulating android sensor data for offensive purposes. *IEEE Transactions on Information Forensics and Security*, pages 1–1, 2016.

[MV15] Byungho Min and Vijay Varadharajan. Design and evaluation of feature distributed malware attacks against the internet of things (iot). In *Engineering of Complex Computer Systems (ICECCS), 2015 20th International Conference on*, pages 80–89. IEEE, 2015.

[MVBC12] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. Tapprints: your finger taps have fingerprints. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 323–336. ACM, 2012.

[MVCT11] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. (sp)iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 551–562, 2011.

[MWA$^+$19] Nikolay Matyunin, Yujue Wang, Tolga Arul, Kristian Kullmann, Jakub Szefer, and Stefan Katzenbeisser. Magneticspy: Exploiting magnetometer in mobile devices for website and application fingerprinting. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, pages 135–149, 2019.

[MZL20] Jian Mao, Shishi Zhu, and Jianwei Liu. An inaudible voice attack to context-based device authentication in smart iot systems. *Journal of Systems Architecture*, 104:101696, 2020.

[Ngu15] Trang Nguyen. Using unrestricted mobile sensors to infer tapped and traced user inputs. In *12th International Conference on Information Technology-New Generations (ITNG)*, pages 151–156. IEEE, 2015.

[nie17] How much time do people spend on their mobile phones in 2017? `https://hackernoon.com/how-much-time-do-people-spend-on-their-mobile-phones-in-2017-e5f90a0b10a6`, May 2017.

[NPDS13] Cornelius Namiluko, Andrew J Paverd, and Tulio De Souza. Towards enhancing web application security using trusted execution. In *WASH*, 2013.

[NSG$^+$14] Sukhvir Notra, Muhammad Siddiqi, Hassan Habibi Gharakheili, Vijay Sivaraman, and Roksana Boreli. An Experimental Study of Security and Privacy Risks with Emerging Household Appliances. In *Communications and Network Security (CNS), 2014 IEEE Conference on*, pages 79–84. IEEE, 2014.

[NSN14] Sashank Narain, Amirali Sanatinia, and Guevara Noubir. Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning. In *Proceedings of the ACM Conference* *on Security and Privacy in Wireless & Mobile Networks*, pages 201–212, 2014.

[NSRU19] AKM Iqtidar Newaz, Amit Kumar Sikder, Mohammad Ashiqur Rahman, and A Selcuk Uluagac. Healthguard: A machine learning-based security framework for smart healthcare systems. In *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 389–396. IEEE, 2019.

[NVHBN16] Sashank Narain, Triet D Vo-Huu, Kenneth Block, and Guevara Noubir. Inferring user routes and locations using zero-permission mobile sensors. In *IEEE Symposium on Security and Privacy (SP)*, pages 397–413, 2016.

[NWX$^+$20] Rui Ning, Cong Wang, ChunSheng Xin, Jiang Li, and Hongyi Wu. Deepmag+: Sniffing mobile apps in magnetic field through deep learning. *Elsevier Pervasive and Mobile Computing*, 61:101106, 2020.

[NYB$^+$09] Xudong Ni, Zhimin Yang, Xiaole Bai, Adam C Champion, and Dong Xuan. Diffuser: Differentiated user access control on smartphones. In *6th International Conference on Mobile Adhoc and Sensor Systems*. IEEE, 2009.

[OC15] Colin O'Flynn and Zhizhang Chen. Power analysis attacks against ieee 802.15.4 nodes. *IACR Cryptology ePrint Archive*, 2015:529, 2015.

[OHD$^+$12] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. Accessory: Keystroke inference using accelerometers on smartphones. *Proc. of HotMobile*, 2012.

[ÖOP03] Sıddıka Berna Örs, Elisabeth Oswald, and Bart Preneel. Power-analysis attacks on an fpga–first experimental results. In *Cryptographic Hardware and Embedded Systems-CHES*, pages 35–50. Springer, 2003.

[Ope] OpenHAB: Open Source Automation Software for Home. `https://www.openhab.org/`. [Online; accessed January-2020].

[PAL$^+$08] Kanthakumar Pongaliur, Zubin Abraham, Alex X Liu, Li Xiao, and Leo Kempel. Securing sensor nodes against side channel attacks. In *11th IEEE High Assurance Systems Engineering Symposium, 2008.*, pages 353–361. IEEE, 2008.

[PAS$^+$16] Giuseppe Petracca, Ahmad Atamli, Yuqiong Sun, Jens Grossklags, and Trent Jaeger. Aware: controlling app access to i/o devices on mobile platforms. *arXiv preprint arXiv:1604.02171*, 2016.

[PJZ$^+$13] Charith Perera, Prem Jayaraman, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Dynamic configuration of sensors using mobile sensor hub in internet of things paradigm. In *IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 473–478, 2013.

[PMSJ16] Giuseppe Petracca, Lisa M Marvel, Ananthram Swami, and Trent Jaeger. Agility maneuvers to mitigate inference attacks on sensed location data. In *IEEE Military Communications Conference*, pages 259–264, 2016.

[Pos11] Stefan Poslad. *Ubiquitous computing: smart devices, environments and interactions*. John Wiley & Sons, 2011.

[pow17] Stop the galaxy s8 from auto 'sleeping' your apps. `https://androidforums.com/threads/stop-the-galaxy-s8-from-auto-sleeping-your-apps.1141445/`, April 2017.

[PRS$^+$17] Giuseppe Petracca, Ahmad-Atamli Reineh, Yuqiong Sun, Jens Grossklags, and Trent Jaeger. Aware: Preventing abuse of privacy-sensitive sensors via operation bindings. In *26th USENIX Security Symposium*, pages 379–396, Vancouver, BC, 2017.

[PSFK15] Jonathan Petit, Bas Stottelaar, Michael Feiri, and Frank Kargl. Remote attacks on automated vehicles sensors: Experiments on camera and lidar. *Black Hat Europe*, 11:2015, 2015.

[PSJA15] Giuseppe Petracca, Yuqiong Sun, Trent Jaeger, and Ahmad Atamli. Audroid: Preventing attacks on audio channels in mobile devices. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 181–190. ACM, 2015.

[PSM15] Dan Ping, Xin Sun, and Bing Mao. Textlogger: inferring longer inputs on touch screen using motion sensors. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, page 24. ACM, 2015.

[PSS$^+$16] Youngseok Park, Yunmok Son, Hocheol Shin, Dohyun Kim, and Yongdae Kim. This ain't your dose: Sensor spoofing attack on medical infusion pump. In *10th {USENIX} Workshop on Offensive Technologies*, 2016.

[PZCG14] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials*, 16(1):414–454, 2014.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *Smart Card Programming and Security*, pages 200–210. Springer, 2001.

[RDL$^+$15] Sankardas Roy, Jordan DeLoach, Yuping Li, Nic Herndon, Doina Caragea, Xinming Ou, Venkatesh Prasad Ranganath, Hongmin Li, and Nicolais Guevara. Experimental study with real-world data for android app security analysis using machine learning. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 81–90, 2015.

[RDM16] Abdul Fuad Abdul Rahman, Maslina Daud, and Madihah Zulfa Mohamad. Securing sensor to cloud ecosystem using internet of things (iot) security framework. In *Proceedings of the International Conference on Internet of things and Cloud Computing*, pages 1–5, 2016.

[RSS$^+$17] Sarah Rajtmajer, Anna Squicciarini, Jose M Such, Justin Semonsen, and Andrew Belmonte. An ultimatum game model for the evolution of privacy in jointly managed content. In *International Conference on Decision and Game Theory for Security*, pages 112–130. Springer, 2017.

[RSYS16] H. Ren, Y. Song, S. Yang, and F. Situ. Secure smart home: A voiceprint and internet based authentication system for remote accessing. In *2016 11th International Conference on Computer Science Education (ICCSE)*, pages 247–251, 2016.

[RW] Yanting Ren and Liji Wu. Power analysis attacks on wireless sensor nodes using cpu smart card. In *22nd Wireless and Optical Communication Conference (WOCC), 2013*, pages 665–670, May.

[RWG$^+$11] Rahul Raguram, Andrew M White, Dibyendusekhar Goswami, Fabian Monrose, and Jan-Michael Frahm. ispy: automatic reconstruction of typed input from compromising reflections. In *Proceedings of the*

18th ACM conference on Computer and communications security, pages 527–536, 2011.

[SA13] Laurent Simon and Ross Anderson. Pin skimmer: Inferring pins through the camera and microphone. In *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*, pages 67–78, 2013.

[SAA$^+$18] Amit Kumar Sikder, Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, Kemal Akkaya, and Mauro Conti. Iot-enabled smart lighting systems for smart cities. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 639–645. IEEE, 2018.

[sam17] Samsung SmartThings Architecture. `http://docs.smartthings.com/en/latest/architecture/`, 2017. [Online; accessed 9-November-2017].

[Sam18a] Samsung. How do i share my location and manage users in SmartThings classic?, 2018.

[Sam18b] Samsung. Marketplace in the smartthings classic app, 2018.

[SAU17] 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*, pages 397–414, Vancouver, BC, 2017.

[SAU19a] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. A context-aware framework for detecting sensor-based threats on smart devices. *IEEE Transactions on Mobile Computing*, 2019.

[SAU19b] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. Context-aware intrusion detection method for smart devices with sensors, September 17 2019. US Patent 10,417,413.

[SBAU19] Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A Selcuk Uluagac. Aegis: a context-aware security framework for smart home systems. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 28–41, 2019.

[SBC$^+$19] Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A Selcuk Uluagac.

Multi-user multi-device-aware access control system for smart home. *arXiv preprint arXiv:1911.10186*, 2019.

[SCEB16] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. Smart-phones attacking smart-homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 195–200. ACM, 2016.

[Sch15] Michael Schiefer. Smart home definition and security threats. In *IT Security Incident Management & IT Forensics (IMF), 2015 Ninth International Conference on*, pages 114–118. IEEE, 2015.

[SCR$^+$19] Takeshi Sugawara, Benjamin Cyr, Sara Rampazzi, Daniel Genkin, and Kevin Fu. Light commands: Laser-based audio injection attacks on voice-controllable systems. 2019.

[SDN15] Allen Sarkisyan, Ryan Debbiny, and Ani Nahapetian. Wristsnoop: Smartphone pins prediction using smartwatch motion sensors. In *IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, 2015.

[Sea15] Tara Seals. Blackhat: Critical zigbee flaw compromises smart homes, 2015.

[SFRS18] Manuel Silverio-Fernández, Suresh Renukappa, and Subashini Suresh. What is a smart device?-a conceptualisation within the paradigm of the internet of things. *Visualization in Engineering*, 6(1):3, 2018.

[SGV$^+$15] Vijay Sivaraman, Hassan Habibi Gharakheili, Arun Vishwanath, Roksana Boreli, and Olivier Mehani. Network-level security and privacy control for smart-home iot devices. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on*, pages 163–167. IEEE, 2015.

[SKE$^+$12] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, pages 161–190, 2012.

[SKKK17] Hocheol Shin, Dohyun Kim, Yujin Kwon, and Yongdae Kim. Illusion and dazzle: Adversarial optical channel exploits against lidars for automotive applications. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 445–467. Springer, 2017.

[SKSP14] Diksha Shukla, Rajesh Kumar, Abdul Serwadda, and Vir V. Phoha. Beware, your hands reveal your secrets! In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 904–917, 2014.

[SLB$^+$16] Chen Song, Feng Lin, Zhongjie Ba, Kui Ren, Chi Zhou, and Wenyao Xu. My smartphone knows what you print: Exploring smartphone-based side-channel attacks against 3d printers. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 895–907, 2016.

[SPA$^+$18] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A Selcuk Uluagac. A survey on sensor-based threats to internet-of-things (iot) devices and applications. *arXiv preprint arXiv:1802.02041*, 2018.

[Spr14] Raphael Spreitzer. Pin skimming: Exploiting the ambient-light sensor in mobile devices. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pages 51–62, 2014.

[SPYG15] Chao Shen, Shichao Pei, Zhenyu Yang, and Xiaohong Guan. Input extraction via motion-sensor behavior analysis on smartphones. *Computers & Security*, 53:143–155, 2015.

[SSDG] Samsung Samsung SmartTings Development Guide. `https://developers.smartthings.com/`. [Online; accessed April-2018].

[SSK$^+$15] Yunmok Son, Hocheol Shin, Dongkwan Kim, Young-Seok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, Yongdae Kim, et al. Rocking drones with intentional sound noise on gyroscopic sensors. In *USENIX Security*, pages 881–896, 2015.

[SST18] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Situational access control in the internet of things. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 1056–1073, 2018.

[SSYA19] Ilia Shumailov, Laurent Simon, Jeff Yan, and Ross Anderson. Hearing your touch: A new acoustic side channel on smartphones. *arXiv preprint arXiv:1903.11137*, 2019.

[ST17] Biljana L Risteska Stojkoska and Kire V Trivodaliev. A Review of Internet of Things for Smart Home: Challenges and Solutions. *Journal of Cleaner Production*, 140:1454–1464, 2017.

[Sta17] Statista. Ownership of smart home technology products in the united states in 2017 (in million households/units in use), by category, 2017.

[Sta18] Statista. Installed base of home automation/smart home systems in the united states from 2012 to 2017 (in millions), 2018.

[STTPLR13] Guillermo Suarez-Tangil, Juan E Tapiador, Pedro Peris-Lopez, and Arturo Ribagorda. Evolution, detection and analysis of malware for smart devices. *IEEE Communications Surveys & Tutorials*, 16(2):961–987, 2013.

[SUCB13] V. Subramanian, S. Uluagac, H. Cam, and R. Beyah. Examining the characteristics and implications of sensor side channels. In *IEEE International Conference on Communications (ICC)*, pages 2205–2210, 2013.

[SWW15] Ahmad-Reza Sadeghi, Christian Wachsmann, and Michael Waidner. Security and privacy challenges in industrial internet of things. In *Proceedings of the 52nd Annual Design Automation Conference*, page 54. ACM, 2015.

[SZZ$^+$11] Roman Schlegel, Kehuan Zhang, Xiao-yong Zhou, Mehool Intwala, Apu Kapadia, and XiaoFeng Wang. Soundcomber: A stealthy and context-aware sound trojan for smartphones. *NDSS*, 11:17–33, 2011.

[Tea17] Trefis Team. Why smart home devices are a strong growth opportunity for best buy, 2017.

[TK12] GS Thyagaraju and Umakanth P Kulkarni. Design and implementation of user context aware recommendation engine for mobile using bayesian network, fuzzy logic and rule base. *International Journal of Pervasive Computing and Communications*, pages 133–157, 2012.

[TLLH18] Yazhou Tu, Zhiqiang Lin, Insup Lee, and Xiali Hei. Injected and delivered: Fabricating implicit control over actuation systems by spoofing inertial sensors. In *27th USENIX Security Symposium*, pages 1545–1562, 2018.

[TPRC11] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the requirements for successful gps spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 75–86, 2011.

[TRCK13] Robert Templeman, Zahid Rahman, David Crandall, and Apu Kapadia. PlaceRaider: Virtual theft in physical spaces with smartphones. In *The 20th Annual Network and Distributed System Security Symposium (NDSS), To appear*, Feb 2013.

[TSRG16] Alpana Tyagi, Anna Squicciarini, Sarah Rajtmajer, and Christopher Griffin. An in-depth study of peer influence on collective decision making for multi-party access control. In *17th International Conference on Information Reuse and Integration (IRI)*, pages 305–314. IEEE, 2016.

[TZL+17] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague. Smartauth: User-centered authorization for the internet of things. In *26th USENIX Security Symposium*, Vancouver, BC, 2017.

[URC+13] Selcuk Uluagac, Sakthi Vignesh Radhakrishnan, Cherita L Corbett, Antony Baca, and Raheem Beyah. A passive technique for fingerprinting wireless devices with wired-side observations. In *IEEE Conference on Communications and Network Security (CNS)*, pages 471–479, Washington, USA, 2013.

[USB14] A.S. Uluagac, V. Subramanian, and R. Beyah. Sensory channel threats to cyber physical systems: A wake-up call. In *IEEE Conference on Communications and Network Security (CNS)*, pages 301–309, 2014.

[VP09] Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In *USENIX security symposium*, pages 1–16, 2009.

[WCGS19] Yao Wang, Wandong Cai, Tao Gu, and Wei Shao. Your eyes reveal your secrets: An eye movement based password inference on smartphone. *IEEE Transactions on Mobile Computing*, 2019.

[WDY$^+$19] Qi Wang, Pubali Datta, Wei Yang, Si Liu, Adam Bates, and Carl A Gunter. Charting the attack surface of trigger-action iot platforms. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1439–1453, 2019.

[Web10] Rolf H Weber. Internet of things–new security and privacy challenges. *Computer law & security review*, 26(1):23–30, 2010.

[WH14] Wen-Chieh Wu and Shih-Hao Hung. Droiddolphin: A dynamic android malware detection framework using big data and machine learning. In *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, pages 247–252, 2014.

[WHBG18] Qi Wang, Wajih Ul Hassan, Adam Bates, and Carl Gunter. Fear and logging in the internet of things. In *NDSS*, 2018.

[win] Introduction to the sensor and location platform in windows. `https://msdn.microsoft.com/en-us/library/windows/desktop/dd318936(v=vs.85).aspx`. Accessed: 2017-03-10.

[WLRC15] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. Mole: Motion leaks through smartwatch sensors. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 155–166. ACM, 2015.

[WMJ19] Raveen Wijewickrama, Anindya Maiti, and Murtuza Jadliwala. dewristified: handwriting inference using wrist-based motion sensors revisited. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, pages 49–59, 2019.

[XBZ12] Zhi Xu, Kun Bai, and Sencun Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 113–124, 2012.

[XZ15] Zhi Xu and Sencun Zhu. Semadroid: A privacy-aware sensor management framework for smartphones. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 61–72. ACM, 2015.

[Y$^+$00] Nong Ye et al. A markov chain model of temporal behavior for anomaly detection. In *Proceedings of the IEEE Systems, Man, and Cybernetics*

Information Assurance and Security Workshop, volume 166, page 169. West Point, NY, 2000.

[YCW$^+$18] Xuejing Yuan, Yuxuan Chen, Aohui Wang, Kai Chen, Shengzhi Zhang, Heqing Huang, and Ian M Molloy. All your alexa are belong to us: A remote voice control attack against echo. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2018.

[YLAI17] Yanfang Ye, Tao Li, Donald Adjeroh, and S Sitharama Iyengar. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, 50(3):41, 2017.

[YN17] Masaya Yoshikawa and Yusuke Nozaki. Hierarchical power analysis attack for falsification detection cipher. In *IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1–6, 2017.

[YOM$^+$19] Masaaki Yamauchi, Yuichi Ohsita, Masayuki Murata, Kensuke Ueda, and Yoshiaki Kato. Anomaly detection for smart home based on user behavior. In *2019 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6. IEEE, 2019.

[YPHC19] Moosa Yahyazadeh, Proyash Podder, Endadul Hoque, and Omar Chowdhury. Expat: Expectation-based policy analysis and enforcement for appified smart-home platforms. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, pages 61–72, 2019.

[YWLY07] Yanfang Ye, Dingding Wang, Tao Li, and Dongyi Ye. Imds: Intelligent malware detection system. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1043–1047, 2007.

[YZJ$^+$19] Chen Yan, Guoming Zhang, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. The feasibility of injecting inaudible voice commands to voice assistants. *IEEE Transactions on Dependable and Secure Computing*, 2019.

[ZCL$^+$18] Rongjunchen Zhang, Xiao Chen, Jianchao Lu, Sheng Wen, Surya Nepal, and Yang Xiang. Using ai to hack ia: A new stealthy spyware against voice assistance functions in smart phones. *arXiv preprint arXiv:1805.06187*, 2018.

[ZCWZ19] Rongjunchen Zhang, Xiao Chen, Sheng Wen, and James Zheng. Who activated my voice assistant? a stealthy attack on android phones without users' awareness. In *International Conference on Machine Learning for Cyber Security*, pages 378–396. Springer, 2019.

[ZDH$^+$] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A. Gunter, and Klara Nahrstedt. Identity, location, disease and more: inferring your secrets from android public resources. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security*, pages 1017–1028.

[ZH19] Huadi Zheng and Haibo Hu. Missile: A system of mobile inertial sensor-based sensitive indoor location eavesdropping. *IEEE Transactions on Information Forensics and Security*, 2019.

[ZMR17] Eric Zeng, Shrirang Mare, and Franziska Roesner. End user security and privacy concerns with smart homes. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, Santa Clara, CA, 2017.

[ZMZL14] Tong Zhu, Qiang Ma, Shanfeng Zhang, and Yunhao Liu. Context-free attacks using keyboard acoustic emanations. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 453–464, 2014.

[ZQL$^+$19] Man Zhou, Zhan Qin, Xiu Lin, Shengshan Hu, Qian Wang, and Kui Ren. Hidden voice commands: Attacks and defenses on the vcs of autonomous driving cars. *IEEE Wireless Communications*, 26(5):128–133, 2019.

[ZR19] Eric Zeng and Franziska Roesner. Understanding and improving security and privacy in multi-user smart homes: A design exploration and in-home user study. In *28th {USENIX} Security Symposium*, 2019.

[ZWY$^+$19] Man Zhou, Qian Wang, Jingxiao Yang, Qi Li, Peipei Jiang, Yanjiao Chen, and Zhibo Wang. Stealing your android patterns via acoustic signals. *IEEE Transactions on Mobile Computing*, 2019.

[ZYJ$^+$17] Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. Dolphinattack: Inaudible voice commands. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 103–117, 2017.

[ZZT09] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM Trans. Inf. Syst. Secur.*, 13(1):3:1–3:26, 2009.

VITA

AMIT KUMAR SIKDER

2014	B.S., Electrical and Electronics Engineering Bangladesh University of Engineering and Technology Dhaka, Bangladesh
2014 - 2015	Solution Manager HUAWEI Technologies Limited Dhaka, Bangladesh
2015 - 2019	M.S., Electrical and Computer Engineering Florida International University Miami, Florida
2015 - 2020	Doctoral Degree Electrical and Computer Engineering Florida International University Miami, Florida

SELECTED PUBLICATIONS, PATENTS, AND INVENTION DISCLOSURES

Amit Kumar Sikder, Leonardo Babun, Z. Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, A. Selcuk Uluagac, *KRATOS: Multi-User Multi-Device-Aware Access Control System for Smart Home*, accepted to be appeared in 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (ACM WiSec 2020).

Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, A Selcuk Uluagac, *Peek-a-Boo: I see your smart home activities, even encrypted!*, accepted to be appeared in 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (ACM WiSec 2020).

AKM Iqtidar Newaz, Amit Kumar Sikder, Leonardo Babun,and A. Selcuk Uluagac, *HEKA: A Novel Intrusion Detection System for Attacks to Personal Medical Devices.* Accepted to appear in the 8th IEEE Conference on Communications and Network Security (IEEE CNS 2020).

Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A. Selcuk Uluagac, *AEGIS: A Context-aware Security Framework for Smart Home Systems*, In proceedings of the 35th Annual Computer Security Applications Conference (ACSAC 2019).

Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac, *A Context-aware Frame-*

work for Detecting Sensor-based Threats on Smart Devices, IEEE Transactions on Mobile Computing , February, 2019.

AKM Iqtidar Newaz, Amit Kumar Sikder, A. Selcuk Uluagac, and Mohammad Ashiqur Rahman, *HealthGuard: A Machine Learning-Based Security Framework for Smart Healthcare System*, In proceedings of the International Symposium on Health and Medical informatics, Management and Security (HMiMS 2019).

Z. Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A. Selcuk Uluagac, *Sensitive Information Tracking in Commodity IoT*, In proceedings of the 27th USENIX security symposium, 2018.

Amit Kumar Sikder, Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, Kemal Akkaya, and Mauro Conti, *IoT-enabled Smart Lighting Systems for Smart Cities*, In proceedings of the 8th IEEE Annual Computing and Communication Workshop and Conference (CCWC), 2018.

Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac, *6thSense: A Context-aware Sensor-based Attack Detector for Smart Devices*, In proceedings of the 26th USENIX Security Symposium, 2017.

Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac, *A context-aware Intrusion Detection method for smart devices with sensors*, granted by U.S. Patent Office, September 2019 (US10417413B2).

Amit Kumar Sikder, Hidayet Aksu, and A. Selcuk Uluagac, *A context-aware intrusion detection method for a smart home, office, and building systems with smart devices*, submitted to Florida International University, September 2018.

Amit Kumar Sikder, Leonardo Babun , Abbas Acar, Hidayet Aksu, and A. Selcuk Uluagac, *A multi-user multi-device access control system with policy management for an IoT-enabled smart environment*, submitted to Florida International University, September 2018.

L. Babun, A. K. Sikder, Abbas Acar, and A. S. Uluagac, *A Method for Automatic Evaluation and Modification of Smart Applications to Enable Comprehensive Forensic Analysis for Smart Settings*, invention disclosure submitted to Florida International University, 2018.