2020

Detecting Rogue Manipulation of Smart Home Device Settings

David Zeichick

Detecting Rogue Manipulation of Smart Home Device Settings

by

David Zeichick

Dissertation Proposal submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Information Assurance

College of Computing and Engineering
Nova Southeastern University

2020

We hereby certify that this dissertation, submitted by David Zeichick conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

Wei Li

8/20/2020

Wei Li, Ph.D.
Chairperson of Dissertation Committee

Date

8/20/2020

Kurtis B. Kredo II, Ph.D.
Dissertation Committee Member

Date

8/20/2020

Yair Levy, Ph.D.
Dissertation Committee Member

Date

Approved:

Meline Kevorkian

8/20/2020

Meline Kevorkian, Ed.D.
Dean, College of Computing and Engineering

Date

College of Computing and Engineering
Nova Southeastern University

2020

An Abstract of a Dissertation Submitted to Nova Southeastern University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Detecting Rogue Manipulation of Smart Home Device Settings

by
David Zeichick
August 2020

Smart home devices control a home's environmental and security settings. This includes devices that control home thermostats, sprinkler systems, light bulbs, and home appliances. Malicious manipulation of the settings of these devices by an outside adversary has caused emotional distress and could even cause physical harm. For example, researchers have reported that there is a rise in domestic abuse perpetrated via smart home devices; victims have reported their thermostat settings being unwittingly manipulated and being locked out of their house due to their smart lock code being changed. Rapid adoption of smart home devices by consumers has led to an urgent need to research mitigation strategies to protect consumers from device takeover.

Currently there is not an easy way for home users to detect that a malicious actor is making unwanted changes to their smart home devices. Change requests to smart home devices travel across the network in the form of network packets. Most of time the payloads of the packets are encrypted using strong encryption methods, so it is not possible to simply read the contents of the packet to learn if the packet contains instructions for the smart device to change states. Previous research has successfully trained machine learning algorithms to identify unique network traffic patterns indicative of state change requests sent to smart home devices. This research extends previous research by identifying state change requests of smart home devices made by residents via a smart home device app on their smart phones or tablets. This research identified 13 key attributes of 3,178 encrypted network traffic connections. The attributes were used as features to train three machine learning algorithms to recognize state change requests. Four smart home devices were used chosen from the following categories: 1) devices with simple behaviors (turns on and off), 2) devices with complex behaviors (can be turned on for a set amount of time), and 3) devices that send a large amount of data (i.e. video camera).

The success of identifying state change requests over encrypted traffic from a mobile app, combined with previous research that identified state changes sent to the smart home device, allows for the development of a system that could block unwanted state changes that originate from a malicious user located outside of the house. Therefore, this research contributes to the body of knowledge of smart home device security and could be extended to the identification of other networking patterns based on encrypted traffic.

Acknowledgments

The completion of my Ph.D. marks almost 12 years of higher education spread over 30 years. The one constant, spanning most of those 30 years, has been the love and support of my wife. She encouraged me to pursue this path and has motivated me all along the way. The encouragement to finish and have a normal life has been amplified in the last few months and I appreciate that motivation. My daughters, Kaitlyn, Joy, and Audrey, while not part of the entire 30 years of this journey, have accepted that I am a perpetual student and may not recognize me without some school deadline hanging over my head. They have listened attentively to my dissertation ramblings and have even engaged in the conversation giving me that much needed encouragement. Thanks to my mother-in-law who took over most of the household chores, freeing up my time. To my dad, who let me know how proud he is of me. Plus, to my mom, who, when calling me in the last few months would ask if I was working on my research, to which I would almost always respond yes. She would then quickly say, “well then I am hanging up on you, bye”. Thank you, family!

I would also like to recognize the unbounded patience of Dr. Li, my dissertation Chair. He has been an invaluable leader and advisor this entire time. Thank you for all of the time provided by my committee, Dr. Levy and Dr. Kredo.

Table of Contents

List of Tables

Tables

List of Figures

Figures

Chapter 1

Introduction

Background

Domestic abuse hotlines have been receiving calls from women who have reported various issues with their smart home devices; one woman reported that she turned on her air-conditioner and a moment later it turned off all by itself, another said that the codes to her front door smart lock kept changing, and one reported that her smart doorbell would periodically ring with no one at the front door (Bowles, 2018). Bowles (2018) found that the changes were not occurring because of some bug in the software, but by men who were actively harassing their partners. This type of domestic abuse is on the rise thanks to the explosive adoption of Internet of Things devices(He et al., 2018).

The term Internet of Things (IoT) first appeared in 1999 and is attributed to the British technologist Ashton (Ashton, 1999). He described it as physical objects that connect to the Internet via sensors. The term has grown to include the data that is exchanged between devices, stored in the cloud, and analyzed (Weber, 2016). Smart home devices are a subset of IoT, referring to IoT devices used in a residence. This paper uses the term smart home devices instead of IoT since this research is focused on devices found in a home. Examples of smart home devices include: smart light bulbs that can turn on when we enter the room, smart refrigerators that remind us that we are almost out of

milk, smart doorbells that call our smart phone and allows us to talk to the person at the front door, to the more bizarre example of a soil sensors for house plants that tweets "water me please" when they are too dry (Hammill & Hendricks, 2013).

Two of the top vulnerabilities of smart home devices, weak password policies and a lack of account lockout by the device's Cloud server, make account takeover trivial for attackers (Alharbi & Aspinall, 2018). Once an attacker has commandeered a smart device, the type of damage inflicted is only limited by the attacker's imagination and the functionality of the smart home device. Theoretical attacks include: locking a resident's television until a ransom has been paid, targeting specific individuals for harassment, and even scaring someone out of their house so that the attacker can gain access for robbery or other purposes (Freed et al., 2018a; Ronen & Shamir, 2016a).

Problem Statement

One of the most risk-inducing features of smart home devices is that they can be accessed from anywhere in the world (Ali et al., 2017; Jia et al., 2017; Ronen & Shamir, 2016a). As a result, a malicious user can manipulate the devices with known user credentials (Freed et al., 2018a). More advanced attacks, such as a malicious actor gaining control of a Cloud server, is also possible (Alharbi & Aspinall, 2018).

Smart home devices, such as WiFi connected light bulbs and thermostats, are becoming more prevalent in residential homes (He et al., 2018). Currently there is not an easy way for home users to detect that a malicious actor is making unwanted changes to their smart home devices (Geeng & Roesner, 2019a; Matthews et al., 2017; Zeng et al., 2017). Change requests to smart home devices travel across the network in the form of network packets. Most of time the payloads of the packets are encrypted by using strong

encryption methods, so it is not possible to simply read the contents of the packet to learn if the packet contains instructions for the smart device to change states (Apthorpe, Reisman, Sundaresan, et al., 2017; Copos et al., 2016a). Despite the payload being encrypted, there are attributes of the packet that are not, such as the source Internet Protocol (IP) and Media Access Control (MAC) address, the destination address, any Domain Name System (DNS) queries, the protocol, and several other revealing pieces of the packet are unencrypted (Apthorpe, Reisman, & Feamster, 2017b; Vijay Sivaraman et al., 2015). These attributes, along with the size of the payload can be used to establish patterns indicative of a smart device state change request, versus an update, versus a status check (Meidan et al., 2017).

Researchers have successfully identified smart home devices and the state changes applied to the devices by implementing machine learning algorithms to categorize encrypted network traffic (Acar et al., 2018; Apthorpe, Reisman, Sundaresan, et al., 2017; Copos et al., 2016b; Marchal et al., 2019; Meidan et al., 2017; Miettinen et al., 2017a). Copos et al. (2016) and Acar et al. (2018) identified unique network traffic patterns indicative of state change requests with the assistance of two supervised learning algorithms, Random Forest (RF) and k-Nearest Neighbors (KNN). It was observed that when the Nest Thermostat transitions from Home to Away packets are sent from the Nest to a specific Nest Cloud server with payload sizes of 1375, 1391, and 2911 (Copos et al., 2016b).

Other researchers found that the Wemo Insight Switch receives large spikes of data when switched from off to on and vice versa (Acar et al., 2018). Several researchers were able to identify the specific smart home device, for example a smart smoke alarm

was connected to the network, through network traffic patterns (Marchal et al., 2019; Meidan et al., 2017; Miettinen et al., 2017a). Other researchers studied the flow of traffic, which they defined as the sequence of packets sent by a device over a particular protocol, such as Network Time Protocol (NTP), Address Resolution Protocol (ARP), Transmission Control Protocol (TCP), and others (Marchal et al., 2019). The researchers converted the flow into a binary time series which was segmented into one second intervals with each segment containing a one if there was at least one packet during that time and zero if there was not. They discovered that each device's flow of network traffic produced a distinct pattern. This research built upon the aforementioned research to identify state change requests across a variety of different types of popular smart home devices.

Dissertation Goal

This research extended previous research by identifying the best performing features and machine learning algorithm combination capable of identifying state change requests across a variety of different types of popular smart home devices. It was important to focus on popular smart home devices so that the outcome of this research was be applicable to the widest audience possible. The different types included smart home devices that have simple behavior (i.e. turn a switch on and off), complex behavior (i.e. turn water on for five minutes), and send large amounts of data (i.e. video cameras). Similar methods, implemented by previous researchers, leveraging machine learning algorithms were used to categorize encrypted network traffic patterns originating from the user's WiFi connected smart phone or tablet, indicative of state change requests of smart home devices.

Identifying state change requests across a variety of different types of smart home devices, must be done at the home network level, which is the common connection point for most smart home device communication (Zeichick, 2018). To accomplish this, home network traffic flow was sequenced into packet size over time intervals (Acar et al., 2018). To link user action to traffic patterns several features of the traffic flow was studied to identify patterns. Interesting features to study included the average packet size per sequence, standard deviation of packet sizes, average time series, protocols used in communication, and many other identifiable packet attributes.

To accurately train a machine learning algorithm it is important to identify the most meaningful features. Testing features for worthiness was accomplished by implementing a 4-fold cross validation was performed, which involved randomly splitting the training set into five distinct subsets, training and evaluating the model 4 times, picking a different fold for evaluation each time, and then training on the other 4 folds. The potential set of features to be studied can be represented as follows.

$$E_T = \{F, D, L\}$$

E, represents the extracted features and *T* represents the time interval. *F* are the features of the traffic sent to the smart home device to initiate a state change, which were mentioned earlier in the paper (e.g. the source IP and MAC address, the destination address, any DNS queries, and the protocol). *D* is the set of smartphones and tablets, and *L* is used to denote location of where a change request originated. *L* is a binary value representing whether the smartphone or tablet is connected to the home network, which means that it is home, or not connected to the home network, which identifies it as not home. *T* can be represented as follows: $T = \{t_1, t_2, \ldots, t_p\}$.

In the above definition, $F = \{f_1, f_2, \ldots, f_n\}$ in which $f_i (1 \leq i \leq n)$ represents a feature. The set *F* was e based on those commonly adopted by literature. One goal of this research was to identify a subset of *F* that can be used to identify state changes effectively and efficiently.

Based on the success of previous research in identifying state changes sent to the smart home device, this research tested the effectiveness of both the Random Forest classifier and KNN classifier to identify patterns of network originating from a smart phone and tablet. Additionally, the effectiveness of Naïve Bayes algorithms was also assessed.

The ability to detect a state change across multiple types of smart home devices is the missing piece to identify if a malicious actor is actively manipulating a resident's smart home device or devices. It was the intention that this research will assist in identifying where the state change request originated from; did the user request the change from inside their house, using their home WiFi connected smart phone or tablet, or did the change request originate from outside the home, from an individual communicating directly to the smart device's Cloud site? The key to differentiating between internal and external state change requests is to correlate the outbound request made by the user on their home WiFi connected smart phone/tablet to the inbound state change request from the smart device's Cloud site. If an outbound request exists and then a corresponding inbound request exists, then the change was made from inside the house. If there is only the inbound request, then the change request originated from outside of the house. This will address situations when a malicious actor has surreptitiously gained access to the smart device's Cloud site via compromised credentials giving them control

over the resident's smart home device. The specific scenario that was studied is when the user is at home and changes are being initiated on the Cloud site from outside the user's home by a malicious actor.

Other scenarios are also applicable. This research could help identify when a botnet has taken control of a smart home device and is actively controlling it. Additionally, it could identify when a manufacturer's Cloud server has been compromised and the attackers are actively controlling the smart home device.

Ideally, the smart device manufacturer would provide a solution to prevent rogue changes to smart home devices. One solution would be for manufacturers to alert users when they notice logins from unknown devices or devices located in previously unseen locations. Unfortunately, this feature does not appear to be provided by any manufacturer (He et al., 2018). Another approach to prevent unwanted changes from outside the home would be to prevent smart home devices from connecting to the Internet. This is not viable since researchers have determined that blocking smart home devices from connecting to the Internet causes many of the devices to stop working (Apthorpe, Reisman, Sundaresan, et al., 2017)

In summary, to identify change requests to smart home devices by smartphones or tablets, identifying features of network traffic were extracted and used to train several machine learning algorithms. The features extracted were tested to ensure that they did not mislead the machine learning algorithm. Next, several machine learning algorithms were trained and tested to identify which was suited to identify the state change requests.

Research Questions

The research questions focused on each aspect of the project, from choosing the correct smart home devices to include in the study, to identify smart home device changes that have been sent over an encrypted connection.

- What popular smart home devices receive their instructions from their Cloud server?
- What popular smart home devices connect to a home WiFi network?
- What popular smart home devices send unencrypted network traffic?
- What popular smart home devices send encrypted network traffic?
- Will publicly available network traffic captures of smart home devices be useful?
    - Will they contain traffic of change requests sent from a smart phone/tablet to a smart home device?
        - Will the traffic be identifiable since it is encrypted?
- Is it possible to learn the general goal of encrypted traffic sent by smart phone apps, by correlating the traffic to events on the smart home device?
    - Is it possible to differentiate commands from background traffic?
        - Updates to the device, time updates, other communication of this type?
- Which type of feature will be most useful in training a machine learning algorithm to recognize state change requests in encrypted payloads?
    - statistical features
    - aggregated features
    - synthesized
    - protocol specific

- Which machine learning algorithm will perform the most efficiently to identify patterns of encrypted network traffic indicative of state change requests of smart home devices?
    - RF
    - KNN
    - Naïve Bayes

Relevance and Significance

Initial research into the typical network architecture of smart home devices revealed some unique characteristics that may be used to alert a user that a malicious actor has made an unwanted change to their smart home device. The main concept is that most smart home devices are directly controlled by a manufacturer's Cloud server (Apthorpe, Reisman, & Feamster, 2017a). When a user is at home and makes a change on their smartphone via the smart device's app, the change request traverses their home WiFi network, is sent to the smart device's Cloud server, the change is noted on the Cloud server, sent to the home WiFi network, then, finally, applied to the smart home device (see Figure 1). This means that at the home network level, when a user is at home, changes originate from inside the home, travel outside the home, then back in again.

This is in contrast to when someone outside the home makes a change; the change is applied to the Cloud server, sent down to the home WiFi network, and applied to the smart device (see Figure 2). What is missing in this scenario, is the change request originating from inside the home. This missing piece can be used to establish if the change originated from inside the home or from outside the home. This can be used if the user is at home and wants to be alerted if someone outside the home has made a change.

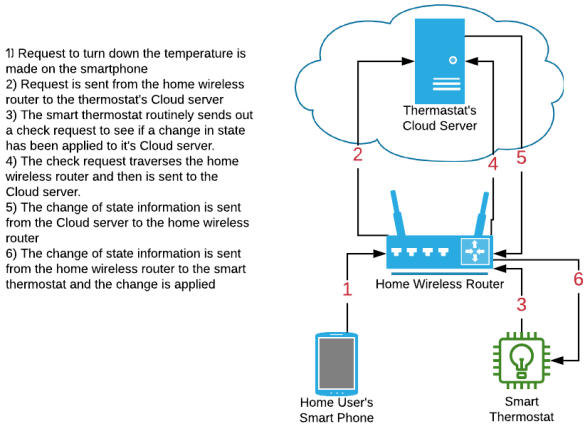

Figure 1. Change initiated by a home user (notice both the change request by the smart phone and the change pulled from the Cloud server traverse the home wireless router)

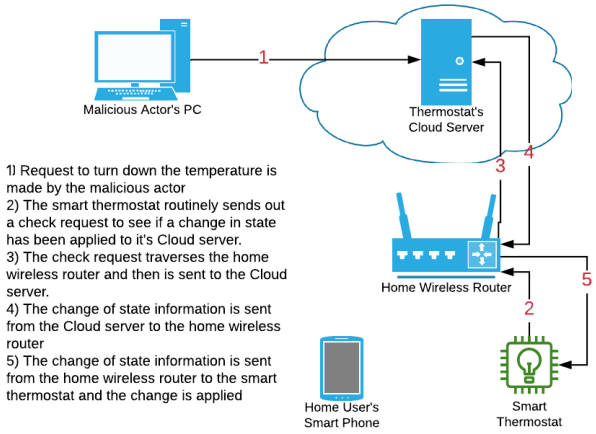

Figure 2. Change initiated by a rogue actor (notice that the change request made by the rogue actor does not traverse the home wireless

The purpose of this research was to identify “rogue changes”, which is defined here as changes made to a smart home device by an actor who is outside the home network. This addresses the situation of when a user is at home and wants to be alerted when unwanted changes are being made by an individual outside the home. Unfortunately, this situation is becoming more common in domestic abuse situations (Naughton, 2018). Naughton (2018) found that men are the ones that typically install a

smart home device, so they are the ones that also control the device. Naughton (2018) found men are using smart home devices to harass their partners. There are times when a home user will want to make changes to their smart home devices when they are away from their home, and, therefore, not on their home network. This research did not intend to create the full software solution to identify rogue changes. Instead, this research intended to fill a gap that would allow for the creation of such software. Previous research has been successful at using machine learning to identify state changes in network traffic for specific smart home devices, however, there doesn't appear to be research that has identified a solution across disparate device types. This research enables the identification of state change requests across several different device types. This provides the missing link to create a tool that is able to identify if a state change request originated from in the house or outside the house.

Another common scenario this addresses is compromised Cloud accounts. It has been shown that two of the top vulnerabilities of smart home devices are weak password policies and no account lockout by the smart device's Cloud server interface (Alharbi & Aspinall, 2018). This vulnerability introduces the risk of attacks being carried out against a compromised account. A Milwaukee couple's smart home devices were accessed by an attacker due to a compromised username and password (Sears, 2019). The attacker turned their thermostat up to 90 degrees, then started talking to them via their Nest Security camera, and finally started playing vulgar music over the security camera.

This research did not intend to address all aspects of smart home device security. Like computer security, it is a broad field covering topics such as data extraction, device manipulation, forming a botnet of smart home devices, and many others (Kolias et al.,

2017; Sikder et al., 2018). This research, which was focused on unwanted changes to smart device settings originating from outside of the house, is an area that does not appear to be covered.

One of the biggest challenges of this research was that most network traffic between a smart phone to the Cloud server and from the Cloud server to the smart device is encrypted (Copos et al., 2016b). Therefore, it was difficult to match the network traffic to the change being made. The action must be learned by correlating the action to type of network traffic. This is a black box problem (the actual work done to solve the problem is not known) which will rely on pattern recognition to solve. Acar et al. (2018) found a small discrepancy in traffic size when the Wemo Insight Switch was turned from on to off and from off to on. Other researchers analyzing network traffic have been able to positively identify the motion sensor of a Nest device being tripped and the wake word being spoken for Amazon's Echo (Apthorpe, Reisman, Sundaresan, et al., 2017).

Barriers and Issues

One of the main challenges for this research was to learn the purpose of encrypted network traffic based upon patterns identified through training a machine learning algorithm. Previous research has demonstrated that it is possible to determine, through network traffic patterns, when a smart light switch is turned on and off (Apthorpe, Reisman, & Feamster, 2017b). The same methodology implemented by Apthorpe et al. (2017b) was followed during this research.

Both datasets, one provided by Alrawi et al. (2019) and the other by Ren et al. (2019), are raw network traffic captures. Neither of the research groups provided labeling of the data. Therefore, another challenge faced by this research was interpreting the

encrypted payloads and then labeling the datasets. As mentioned, since the payloads are encrypted it was impossible to see if they contain state change requests. Therefore, state change requests were generated with the same smart home device in our lab environment and then were compared with the traffic from the publicly available network traffic. When the encrypted traffic that matched (e.g. payload size, response time, protocol, etc.) then the publicly available network traffic packet was labeled as a state change.

Assumptions, Limitations and Delimitations

The main goal was to identify state change requests sent over a network from an app on a smart phone to the smart home device's Cloud site. Therefore, this research only focused on traffic sent from a smart phone or tablet to a smart device's Cloud site. Identifying these state changes filled the missing piece to create an application that could identify smart home device state change requests that originate from outside of the home. However, creating this application was outside the scope of this research.

This research includes smart home devices that have been designed to have change requests first flow through the device's Cloud site then to the smart home device. It is assumed that having change requests first go to the device's Cloud site and then to the device itself is a very common architecture. Therefore, this research is applicable to the most common smart home devices in use.

This research did not plan to address the other numerous security vulnerabilities of smart home devices; current security vulnerabilities include issues such as snooping on personal webcams, analyzing web traffic generated by smart home devices to determine if the homeowner is home or away, and maliciously gaining access to a smart home

device via a known vulnerability. These other issues have been well documented and there are several research efforts currently underway.

Definition of Terms

- Smart home device
    - An electronic device that connects to the Internet, can be controlled remotely by the user, and was purchased for use in the home.
- Malicious actor
    - A group or individual that wishes to cause harm, either physical or emotional, to its target group or individual.
- Cloud service
    - A server controlled and housed by the manufacturer of the smart home device
- Botnet
    - A collection of computers, which can include smart home devices, that have been commandeered by an attacker to cause harm to their target. The harm typically involves having all of the computers in the botnet send network traffic to one site in attempt to overwhelm the target preventing it from responding to legitimate traffic.
- Black box problem
    - A problem is presented and the answer is given without any explanation of how the answer was arrived at. The actual work done to solve the problem is not known. This is typical of machine learning algorithms that are trained with massive amounts of data, but not instructed on how to exactly

solve the given problem. The algorithm is fed the question and responds with an answer void of explanation of how the answer was reached.

List of Acronyms

- ACK
  - acknowledgement
- ANN
  - Artificial Neural Network
- ARP
  - Address Resolution Protocol
- CARA
  - Clairvoyant access right assignment
- CSV
  - Comma separated value
- DDOS
  - Distributed denial of service
- DOS
  - Denial of service
- DNS
  - Domain Name Service
- ESO
  - environmental situation oracles
- GPS
  - Global positioning service

- IAT
    - Inter-arrival time
- IDM
    - Intrusion Detection Mitigation
- IDS
    - Intrusion Detection System
- IoT
    - Internet of Things
- IP
    - Internet Protocol
- ISP
    - Internet Service Provider
- KNN
    - k-Nearest Neighbors
- LED
    - Light-emitting diode
- MAC (address)
    - Media access control (address)
- NIDS
    - Network-based Intrusion Detection System
- NTP
    - Network Time Protocol
- RF

    - Random Forest
- SDN
    - software defined networking
- SVM
    - Support Vector Machine
- SSL
    - Secure Sockets Layer
- SYN
    - synchronize
- TCP
    - Transmission Control Protocol
- TPR
    - true positive rate
- WiFi
    - Wireless networking technology

Summary

This research extended previous research by identifying the best performing features and machine learning algorithm combination capable of identifying state change requests across a variety of different types of popular smart home devices. This was accomplished by training a machine learning algorithm with home network traffic in order for it to learn the network pattern analogous of smart home device change requests from smart phones and tablets. Several categories of smart home devices were included in the study along with their corresponding smart phone apps.

Chapter 2

Review of the Literature

Introduction to Smart Home Device Security

Companies are rushing to meet consumers' growing need for smart home devices. This rush to market by manufacturers has produced serious deficiencies in privacy and security. This same mistake was made 20 years ago when consumers rushed to the Internet to shop and bank online (Shackelford et al., 2017). The Internet was designed to openly share data, which is the complete opposite of what is necessary for secure transactions. Malicious actors took advantage of the lack of security by creating malware and sniffing unencrypted data with the goal of stealing personal data (Shackelford et al., 2017). The industry responded by adding security layers and products. Secure Sockets Layer (SSL) was implemented to secure internet transactions, antivirus programs to rid computers of nasty malware, and passwords to authenticate users. This solution is not foolproof since it relies on consumers to implement many of the solutions. Unfortunately, most home users are not technical; they do not understand how to properly configure their systems or realize the importance of a strong password (Fu et al., 2017b). The same is true with smart home devices. Most home users can't perform basic security functions which leads to the question: should they be adding smart devices to their homes (Walker, 2014)?

Current Research of Smart Home Device Security

Research has uncovered major vulnerabilities in smart home devices. Alrawi et al. (2019) evaluated the security of 45 smart home devices by studying the security of the smart devices' services, its mobile applications, its Cloud endpoints, and its communications. They found that several of the devices' services had self-signed certificates, supported weak ciphers, used short Transmission Layer Security (TLS)/ Secure Sockets Layer (SSL) keys, permitted the use of vulnerable version of SSL, and had expired certificates (Alrawi et al., 2019a). For the mobile apps the researchers found that one or more issues related to permissions, sensitive data, or incorrect use of cryptography. They also found 24 over-privileged mobile applications that had permissions on the mobile device that were not used. On the smart devices' network, they found 18 devices that used outdated services, leaked sensitive information, lacked encryption for authentication, or ran a vulnerable service. They found that: 1) eight devices used cloud endpoints that are vulnerable and have public exploits, 2) seven devices authenticated with cloud endpoints in clear text, and 3) 26 devices used cloud endpoints that have TLS/SSL configuration issues, like self-signed certificates, domain name mismatch, and support for vulnerable versions of TLS/SSL protocol. One positive finding is that the majority of the devices used encryption when communicating over the Internet

Notra et al. (2014) experimented with several smart home devices, including the Phillips Hue light-bulb, the Belkin WeMo power switch, and the Nest smoke-alarm, and found that the devices lack encryption, appropriate authentication, and integrity checks. These vulnerabilities make IoT devices susceptible to a variety of attacks including:

denial of service (DOS), replay, man-in-the-middle, device tampering, information disclosure, side channel attack, and eavesdropping (Atamli & Martin, 2014; Kasinathan et al., 2013).

A French company, Eurecom, analyzed 123 smart devices and discovered 38 vulnerabilities that included bad encryption and deliberately set backdoors (Costin et al., 2014a). Out of the fifty smart home devices that Symantec studied none of them forced strong passwords or implemented authentication between the device and the cloud (Wueest, 2015). Hewlett Packard had similar findings and characterized the main smart home device security issues as: not encrypting network traffic, poor authentication, and vulnerable web interfaces (Enterprise, 2015). This problem is so bad that the FBI warned home users of smart home devices' vulnerabilities (FBI, 2015).

Attacks to smart home devices are external, the Mirai botnet, and internal, harassing residents by altering the thermostat (He et al., 2019a). These attacks can be categorized into five types of behavior: 1) ignoring the functionality, 2) reducing the functionality, 3) extending the functionality, 4) discerning residents' behavior based on smart device generated network activity, and 5) misusing the functionality (Apthorpe, Reisman, & Feamster, 2017b; Ronen & Shamir, 2016a). In the first type, the attacker ignores the designed feature of the smart home device (if it was a smart camera, they don't use any of the functionality of a camera) and instead treat the device as an embedded computer (Ronen & Shamir, 2016a). An example of this is installing malware on the smart home device to make it part of a botnet. Botnets comprised of smart home devices have been used to perform large scale distributed denial of service (DDOS)

attacks; an example is the Mirai attack which brought down major services such as Twitter, Netflix, Reddit, and GitHub (Kolias et al., 2017)

The second type of attack is reducing the functionality of the smart home device which involves disabling the device or features of the device. Examples include disabling a smart television so that it won't turn on and altering the functionality of a smart refrigerator so that it won't cool its contents (Ronen & Shamir, 2016a).

The third type of attack, extending the functionality, involves using the functionality in a different way than designed in order to achieve an unexpected or different physical effect. Ronen et al. (2016) demonstrated an attack in which they took control of a smart light bulb and strobed the lights in such a way as to trigger seizures in people suffering from photosensitive epilepsy. The same researchers also demonstrated how they could manipulate an light-emitting diode's (LED) light intensity to create a covert channel (Ronen & Shamir, 2016a). This was accomplished by quickly switching light intensities that mimic the sending of binary data. The light intensities used were so close in brightness that they could not be discerned by the human eye.

The fourth type of attack involves discerning residents' behavior based upon the network traffic generated by smart home devices (Apthorpe, Reisman, & Feamster, 2017b). Researchers studied the Sense Sleep Monitor, the Nest Cam Indoor security camera, the WeMo switch, and the Amazon Echo and found that the encrypted network traffic generated by these devices reveal sensitive information about the users (Apthorpe, Reisman, & Feamster, 2017b). For the Sense sleep monitor the network traffic peaked when the user interacted with it; in the smart home laboratory the researchers were able to deduce that the user went to bed at 12:30, briefly got up at 6:30am and then got up at

9:15am. These times correlated to spikes in network traffic generated by the Sense sleep monitor revealing the user's sleep pattern. The same correlations were made between device usage and traffic spikes for the Nest Cam Indoor security camera, the Wemo switch and the Amazon Echo (Apthorpe, Reisman, & Feamster, 2017b). Copos et al. (2016) was able to identify the network traffic patterns produced when the Nest smoke detector detects smoke and when the smoke alarm is triggered.

The fifth type of attack, misusing the functionality, is the main focus of this research project. This attack uses the functionality of the smart home device, but does so in an incorrect or unauthorized way (Ronen & Shamir, 2016a). These attacks are typically used to harass the resident. For example, an attacker my turn down the smart thermostat in the winter so that house if very cold or turn the lights on in the middle of the night to wake victim.

Domestic Abusers Use of Smart Home Devices

Smart home devices are becoming the weapon of choice for perpetrators of domestic abuse (Freed et al., 2018a). This is not surprising given their history of using technology against their victims. Examples of this abuse includes online harassment, cyberbullying, cyberstalking, and doxing (Douglas, 2016; Fraser et al., 2010; Vitak et al., 2017a; Wisniewski et al., 2016a). Domestic abuse is surprisingly common, with research indicating that one in three women and one in six men will experience intimate partner violence in their life (Freed et al., 2018a).

Examples of smart device domestic abuse include: switching the air-conditioner off right after the victim turns it on, changing the code for the smart front door lock, and triggering the doorbell to ring (Bowles, 2018). Abusers do this to either watch and listen,

or, more likely in domestic abuse cases, to show power (Bowles, 2018). These attacks are accomplished in the very low-tech method of signing into the device's Cloud account with the username and password. In some cases the abuser already knows the username and password because they were the one that setup the smart home device (Freed et al., 2018a). In other cases, they gain the password either by intimidating the victim to disclose it, guessing the password based upon intimate knowledge of the victim, or by answering the password reset security questions (Freed et al., 2018a).

Even though domestic abusers' attack methods are not technically sophisticated does not mean that they are easy to prevent. Freed et al (2018) analyzed current threat models and countermeasures and determined that they do not adequately address attacks in which the attacker possesses intimate knowledge of their victims. To solve this problem the researchers suggest focusing on attack methods of average computer users, like carrying out an attack with a compromised password. Freed et al. (2018) recommended analyzing the difference in legitimate user behavior versus the attacker's. They suggest the Cloud service use the learned difference in behavior during authentication to determine if it is the legitimate user logging in or the attacker.

Smart Home Device Architecture

The network architecture of smart home devices is typically configured in one of two ways: 1) Cloud-centric, which is mobile application to cloud or 2) direct access, which is mobile application to device (Wang et al., 2018). With Cloud-centric, the user issues changes via their smart phone which communicates directly with the smart home device's Cloud server, which relays the changes to the smart home device (Notra et al.,

2014). An example of a smart home device that uses this architecture is the Nest thermostat.

Direct access cuts out the Cloud server as the middle-man. Instead the user communicates directly to the smart home device via the app on their smart phone (Notra et al., 2014). Examples include the Philips Hue light-bulb and the WeMo switch.

The Cloud-centric architecture is currently the most popular (Intellectsoft, 2015). The focus of this research is primarily on the Cloud-centric architecture since it is focused on malicious state changes to smart home devices originating from outside of the home.

Blocking the Smart Home Device from Accessing the Cloud

On initial examination of how to block external attackers from making changes to smart devices located in the home it may seem like the best approach would be to block the smart home device from connecting to its Cloud site. After all, if the smart home device cannot connect to its Cloud site it will not get any of the changes requested by an external attacker. However, as outlined in the previous section, Cloud-centric is the most common configuration for smart home devices. Hence, if Cloud access is blocked, then the user who is at home will not be able to make any changes to their smart home device because all of their requests go through the smart device's Cloud site.

Additionally, completely blocking a smart home device from connecting to its Cloud server renders most smart home devices ineffective. Apthorpe et al. (2017) tested removing internet access to seven smart home devices and found that four of the devices lost most of their smart features while the remaining three devices completely lost functionality. It is worth noting that researchers found that they were able to block select

network traffic of some smart devices without losing any functionality (Copos et al., 2016b; Notra et al., 2014; Vijay Sivaraman et al., 2015). Sivaraman et al. (2014) and Notra et al. (2014) both were able to block the Nest's Smoke Alarm from sending logs to its Cloud logging server while still allowing a home user to be alerted when the smoke alarm detected smoke. Copos et al. (2016) blocked the Nest Smoke Alarm's access to all Cloud servers except for the Cloud servers responsible for authentication, notification, and token renewal. They set off the smoke alarm and successfully received a fire notification.

Access Control

An Access Control List (ACL) could be used to block unwanted changes to smart home devices. An ACL is used to specify if a subject or an object is approved or denied for a specific action (Schuster et al., 2018a). Traditional ACLs have been used for smart home devices; the problem is they are not specific enough to be effective with smart home devices. Decisions need to be made based upon the situation in which a change is being requested, the context of the change, or even the state of the environment (He et al., 2018; Jia et al., 2017; Tian et al., 2017a). Additionally, several different users interact with smart home devices, such as the family's Alexa device or their smart lock connected to their front door (He et al., 2018). This would be fine if all users in the house should have the same type of access. He et al. (2018) points out that households often have very complex social relationships; there may be parents who want to spy on their teenagers, mischievous children, or even abusive partners (Matthews et al., 2017; Ur et al., 2014a). It is extremely important to take these relationships into consideration when populating an ACL.

He et al. (2018) found that it was important to their research participants that users be physically present in the house whenever they change a smart device's behavior; 68% of the participants felt that the user must be home to control the lights, unless it was the owner or the spouse making the change. Other major factors in deciding access control was the age of the person making the change, the time the change is requested, the status of the induvial making the change, and the location of the smart device in the home, all of which are not supported by current smart home devices (He et al., 2018; Ravidas et al., 2019). Access controls based upon situational conditions in not a new thing, smartphone frameworks have been using this for many years (Schuster et al., 2018a). The main difference in ACLs between smartphones and smart home devices is that smartphones typically have one user and smart home devices have several users.

Determining if the user is at home has been implemented by many smart home devices including: SmartThings, Nest, Ecobee, Wink, Apple HomeKit, Sennse Mother, Abode, Netatmo, and Honeywell (Schuster et al., 2018a). The two main ways to determine if the user is at home is the global positioning system (GPS) coordinates of the user's smartphone and motion sensors on the smart home devices (Schuster et al., 2018a). The upside to the GPS location is that it is possible to uniquely identify the user since the user's phone is directly linked to the user. The downside is it that it not only tracks if the user is at home, but also everywhere they go outside of their home. This creates privacy concerns, especially if the user's location is shared with another smart home device that is simply attempting to determine if the user is home or not (Schuster et al., 2018a). The problem with the motion sensor is that it can only track if someone is home, not exactly who is at home.

Schuster et al. (2018) proposed environmental situation oracles (ESOs) which gather situational data from multiple smart home devices, such as the user's GPS location and if a particular motion sensor was tripped. The ESOs can be queried by an ACL to determine if a particular situation exists or not. For example, there may be a rule that a teenager must be home to control the lights. The ACL could query the ESO containing the teenager's GPS location. However, the ESO would not divulge the teenager's GPS coordinates, instead it would respond true if they are home or false if they are out. The ESO solution is currently theoretical and has not seen much (if any) industry adoption.

One of the most important features of an ACL is that it must be easy to use by a homeowner (Ravidas et al., 2019). Usability is particularly important since most home users have very little knowledge about security (Kim et al., 2011). Mahalle et al. (2013) developed a system modeled on a trust-based access control model designed to automatically set rules based on the trustworthiness of the user. Another system called Clairvoyant access right assignment (CARA), is designed to automatically give suggestions about the access rights a visitor to the home should have (Kim et al., 2011). One of the main constraints implemented by CARA is that the visitor must be in the house to access the device. Restricting the use of a device to only those physically present in the house is the main goal of this research project. Implementing an ACL is not a viable solution to the problem presented in this case since the attacker is masquerading as the home user; the attackers are using the victim's username and password to gain access. ACLs are not designed to block access to a system due to a compromised account.

Monitor Network Activity

Network attacks against smart devices can be passive or active; malicious actors can passively monitor network traffic to exfiltrate sensitive information or they can attack the devices, which creates network traffic. Several researchers have trained machine learning algorithms to passively learn network traffic patterns generated by smart home devices, to piece together clues from the devices' actions to infer the residents' in home behaviors (Acar et al., 2018; Apthorpe, Reisman, & Feamster, 2017a; Apthorpe, Reisman, Sundaresan, et al., 2017; Barrera et al., 2017b; Copos et al., 2016b; Junges et al., 2019; OConnor et al., 2019; Ren et al., 2019a; Subahi & Theodorakopoulos, 2019). Other researchers have used an Intrusion Detection system to monitor for attacks on smart home devices (Anthi et al., 2019; Hodo et al., 2016; Mehdi Nobakht et al., 2016; Ramapatruni et al., 2019; Vijay Sivaraman et al., 2015; Wang et al., 2018). The types of attacks to monitor for include: 1) Distributed Denial of Service (DDoS), 2) conventional attack, 3) routing attack, and 4) man-in-the-middle (Zarpelão et al., 2017).

Hodo et al. (2016) used a Network-Based Intrusion Detection System (NIDS) to identify and thwart DDoS attacks performed against smart home devices. The NIDS used an Artificial Neural Network (ANN) which was trained via a supervised learning procedure. This involved feeding the neural network with a labeled training set in order for it to learn the difference between normal and anomalous traffic (Hodo et al., 2016). Anthi et al. (2019) focused on detecting conventional attacks on smart home devices. The research involved establishing the normal behavior of each smart home device, identifying when an attack is occurring based on identified malicious packets, and determining the type of attack that is taking place against which smart home device. Ramapatruni et al. (2019) followed a similar approach leveraging machine learning

algorithms, such as Hidden Markov Models, to learn the normal traffic patterns of smart home devices. Using the normal traffic patterns as a baseline, Ramapatruni et al. (2019) identified any traffic outside the baseline as anomalous traffic. The researchers were successful 97% of the time in identifying malicious traffic.

Sivaraman et al. (2015) extended this concept by dynamically quarantining smart home devices that were producing traffic determined to be malicious. This solution would be implemented though the use of Software Defined Networking (SDN). SDN would allow for dynamic security rules, such as if someone is in the house or the time of day of an event, such as tuning on music at 2 a.m. Instead of implementing this solution in the house, Sivaraman et al. (2015) propose that a specialist, such as the Internet Service Provider (ISP), offer this service. The ISP would receive a feed of network traffic, learn the typical behavior of all of the smart devices and the residents' interactions with the devices, tweaking the rules as more data was fed into it (Vijay Sivaraman et al., 2015).

Routing attacks, the third type of attacks studied by researchers interested in Intrusion Detection System (IDS) for smart home devices, are designed to disrupt network traffic. One popular routing attack, the worm hole attack, disrupts network traffic by creating a network tunnel between two devices and then sending all of the network traffic through the tunnel (Pongle & Chavan, 2015). This attack is typically found in smart devices outside the home. Pongle et al. (2015) created an IDS specifically to detect wormhole attacks.

In man-in-the-middle attacks, the attacker is able to intercept their adversary's traffic in order to monitor the traffic, modify it, or stop it completely (Tertytchny et al.,

2019a). Researchers were able to gain access to a LightwaveRF smart home device via a man-in-the-middle attack in which they intercepted the device's firmware update, modified the update so that they could easily access the device, then sent the update to the device (Barcena & Wueest, 2015). Tertytchny et al. (2019) created an IDS which was successful in identifying these attacks about 90% of the time. Nobakht et al. (2016) created a host-based intrusion detection system called IoT- Intrusion Detection Mitigation (IDM) designed to differentiate between suspicious and normal network activity and block identified suspicious activity. The researchers tested IoT-IDM with a Hue Smart Light Bulb system in which they were able to sniff the secret key, known as a whitelist token, which is used to authenticate a known user. The whitelist token was used by the simulated attacker, who was connected to the home network, to log into the Hue Smart Light Bulbs. IoT-IDM, using a learning model that leverages SVMs to classify the data, was able to identify the attack with an accuracy of 100%.

Passive attacks involve capturing network traffic generated by smart home devices. Once captured, researchers have demonstrated that patterns identified by machine learning algorithms can show what the smart home device is doing, even if the network traffic is encrypted (Junges et al., 2019; Subahi & Theodorakopoulos, 2019). To train the machine learning algorithms researchers have used a variety of characteristics of the network traffic including: the throughput, burstiness, direction, size of payload, the proportion of synchronize (SYN) and acknowledgement (ACK) packets (which are involved in establishing a TCP connection), plus various statistics calculated about the network traffic (Acar et al., 2018; Apthorpe, Reisman, & Feamster, 2017a; Apthorpe, Reisman, Sundaresan, et al., 2017; Barrera et al., 2017b; Copos et al., 2016b; Junges et

al., 2019; OConnor et al., 2019; Ren et al., 2019a; Subahi & Theodorakopoulos, 2019). Apthorpe et al. (2017) were able to link device state changes to its network traffic for a variety of devices including the Amazon Echo, Nest Security Camera, and the Belkin WeMo Switch (see Figure 3 below). Acar et al. (2018) found a small discrepancy in traffic size between the Wemo Insight Switch being turned from on to off and from off to on (see Figure 4 below).

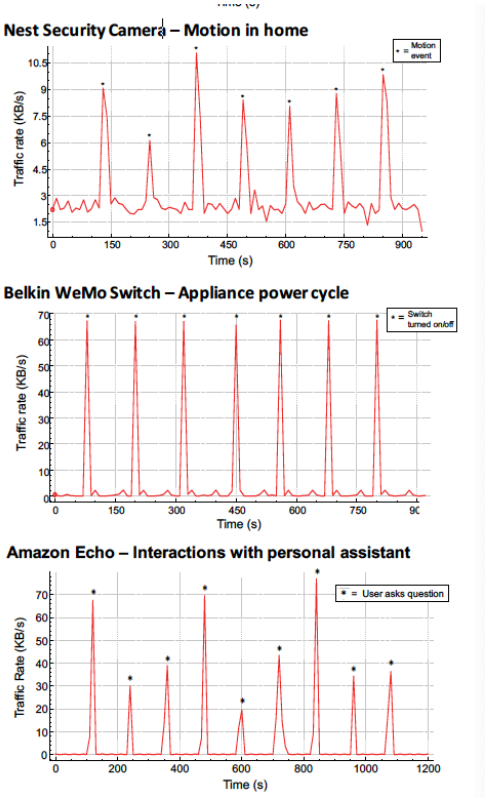

Figure 3. Network traffic send and receive rates corresponding to user activities (Apthorpe, Reisman, Sundaresan, et al., 2017)

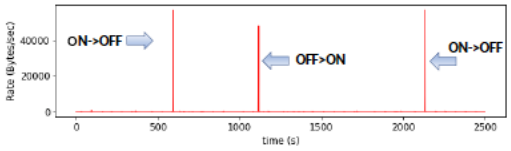

Figure 4. Wemo Insight Switch's network traffic volume when switched from on to off and then from off to on (Acar et al., 2018)

This type of attack is similar to the research in this paper, but differs in that the attack in this research is accomplished through an account takeover by someone who is not connected to the home network and is performing the attack from somewhere on the Internet. Therefore, this research tested the worthiness of each of these characteristics. An in-depth explanation of each characteristic listed above can be found in the Methodologies section of this paper.

The attack outlined in this research is more in-line with typical account takeover attacks. This involves the malicious actor leveraging a user's credentials. In this attack the malicious actor is not normally connected to the user's home wireless network to perform the attack. Extensive research of current journal and conference papers on intrusion detection systems for smart home devices did not identify any research of how to prevent remote attackers who have gained compromised credentials, to take over smart home devices. This research intended to fill that gap.

Summary

Smart home devices are rapidly being added to houses around the world. Researchers have discovered many concerning vulnerabilities with smart home devices and have demonstrated several successful attacks on the devices. Domestic abusers, who have been using technology to harass their victims, have started to adopt smart home devices as a new attack vector.

Domestic abusers have followed the low-tech approach of commandeering a smart home device via a compromised username and password. This method is effective due to the architecture of most smart home devices, which involves state change requests of smart home devices going through the device's Cloud server, then passed to the smart home device. Blocking a smart home device's connection to the Cloud service is ineffective since researchers have determined that taking this approach renders most smart home devices useless.

Intrusion detection systems have proved useful to most traditional types of active attacks against smart home devices. Passive attacks can be successful in learning about a households' activities by learning the traffic patterns generated by smart home devices. A compromised cannot be defended against with the methods researchers developed for passive or active attacks. This is also true for access control lists. ACLs are effective at restricting what type of access a user has to a device based on a number of factors. A compromised account is outside the scope of an ACL.

Chapter 3

Methodology

Overview

This research was carried out following an experimental design through a lab experiment. The main goal was to identify state change requests sent over a network from an app on a smart phone to the smart home device's Cloud site. Researchers have found that the payload of this network traffic is typically encrypted (Acar et al., 2018; Alrawi et al., 2019a; Apthorpe, Reisman, Sundaresan, et al., 2017; Bezawada et al., 2018; Junges et al., 2019; Miettinen et al., 2017a; OConnor et al., 2019; Ren et al., 2019a; Sivanathan et al., 2017; Subahi & Theodorakopoulos, 2019). Therefore, network traffic patterns were studied to see if state change requests generate identifiable patterns.

This was accomplished by selecting specific attributes, also known as features, which can be represented as follows.

$$E_T = \{F, D, L\}$$

E, represents the extracted features and *T* represents the time interval. *F* are the features of the traffic sent to the smart home device to initiate a state change, $F = \{f_1, f_2, \dots, f_n\}$ in which $f_i (1 \leq i \leq n)$ represents a feature. *D* is the set of smartphones and tablets, and *L* is used to denote location of where a change request originated. The end goal was to identify the combination of features, *F*, and machine learning classifiers

which are the most successful in identifying state changes hidden in the encrypted payloads across several types of smart home devices.

The Scikit-Learn platform was used for machine learning (*Scikit-Learn 0.22.2*, 2020). This included the Juypter notebook to store and run the machine learning tasks, Python as the primary programming language, and the numpy and pandas libraries (*NumPy — NumPy*, 2020; *Pandas 1.0.3*, 2020).

Four popular smart home devices were included in this research. The criteria for selecting these devices is that they must connect to the home WiFi network and must be controlled by its corresponding Cloud server. Additionally, at least one device was included from the following rough categories: 1) a device that has a simple behavior (turns on and off), 2) a device with a complex behavior (can be turned on for a set amount of time), and 3) sends a large amount of data (i.e. video camera). The selection of these smart home devices depended on the availability of publicly available datasets from previous research. The publicly available datasets did not prove adequate, meaning it did not having enough network traffic captures of smartphone to smart device interactions, so simulation data was created. This was accomplished by setting up our own lab with the smart home devices on a home WiFi network and capturing the network traffic while generating numerous state change behaviors of each smart home device.

This research followed the typical steps involved in training a machine learning algorithm. An overview of the steps is listed below with a detailed explanation of each step following this list.

1. Get the data
    a. List the data needed and how much is needed

   b. Capture or acquire the network traffic
   c. Sample a test set
2. Explore the data
   a. Create a copy of the data for exploration
   b. Study each attribute and its characteristics
   c. Identify the target attribute
   d. Label the data
   e. Convert the data for Scikit-Learn
   f. Visualize the data
   g. Study the correlations between attributes
   h. Identify the promising transformations
3. Prepare the data
   a. Data cleaning
   b. Feature selection, feature engineering, and feature scaling
      i. $F = \{f_1, f_2, \dots, f_n\}$ in which $f_i (1 \leq i \leq n)$ represents a feature
      ii. One goal of this research is to identify subsets of *F,* represented by *R*, that can be used to identify state changes effectively and efficiently. $R = \{r_1, r_2, \dots, r_m\}$ in which $r_i (1 \leq i \leq m)$ represents a subset of *F*
4. Identify promising models
   a. Train many models using standard parameters, represented as the set $C = \{c_1, c_2, \dots, c_k\}$ in which $c (1 \leq i \leq k)$ represents a

classifier. The goal is to identify which subset *S* ($S \subseteq C$), when trained with *R* performs the best

b. Measure and compare their performance

c. Analyze the most significant variables for each algorithm

d. Analyze the types of errors the models make

e. Repeat the five previous steps for each smart home device's network traffic and then for all of the smart home devices' network traffic combined

f. Select the top three to five most promising models

5. Fine-Tune the System

a. Fine-tune the hyperparameters using cross-validation

b. Try Ensemble methods

c. Estimate the generalization error

(Géron, 2019)

Step 1: Get the data

There were two main datasets to evaluate. One dataset was provided by Alrawi et al. (2019). The researchers evaluated 45 devices from several disparate categories including appliances, cameras, home assistants, media and network devices. They collected network traffic over a period of 13 days which resulted in 150 GB of data.

The second dataset came from the work of Ren et al. (2019). The full dataset includes network traffic captured from 81 different smart home devices located in labs in the United States and the United Kingdom. Over the period of a month they conducted 34,586 automated and manual experiments on the smart home devices.

As mentioned previously, the publicly available datasets did not provide enough instances of encrypted network traffic between the smart home device's app and the smart home device's Cloud site so the data was supplemented by generating and capturing our own network traffic. The network traffic generated by the smart home device apps were captured using tcpdump on the home router (in this case a router running the OpenWRT operating system) (*OpenWrt Project*, 2020; *Tcpdump*, 2017). It was necessary to generate our own supplemental network captures, so the same method was used as Apthorpe et al (2017). Each device was isolated on its own network, and all possible means of triggering the device were explored.

Once all the device's states were triggered, the network traffic generated by the device was analyzed for uniqueness. The same behavior was triggered and again compared to the previous network traffic. A similar method was used by Copos et al. (2016), in which they found that packets of a certain size were sent when the Nest motion sensor was tripped which allowed them to determine with 88% accuracy that the sensor was tripped.

Smart home devices can be roughly grouped into three, possibly overlapping, categories: 1) those with simple behaviors (i.e. smart plugs which can be turned on or off), 2) more complex behaviors (i.e. smart watering systems that be set to turn on for a set amount of time), and 3) those that send large amount of data (i.e. Alexa sending a voice recording and Ring sending a video clip). Experiments with smart home devices from each of these categories were conducted. The list of devices includes: 1) TP-Link smart plug (simple behavior), 2) the Belkin WeMo switch (simple behavior), 3) the Chamberlain myQ Garage Door opener (simple behavior), 4) Nest camera (large amount

of data), 5) Rachio Smart Sprinkler Controller (complex behavior). These devices were selected because they are popular smart home devices and several of them were used in the publicly available dataset from Alrawi et al.(2019) and Ren et al.(2019). Some of the devices listed proved to be less than ideal candidates.

Step 2: Explore the data

Researchers have used the tool Zeek (formerly known as Bro) to assist in interpreting the network traffic (Copos et al., 2016b; Paxson, 1999). Zeek can be used to read in a pcap file and then produce a list of all connections including information about the source, destination, protocol used for the connection, duration, and number or bytes sent. Therefore, Zeek was used in this research to examine connection patterns in the publicly provided network capture files from Alrawi et al.(2019) and Ren et al.(2019). This helped identify which packet captures, and specifically which parts of those packet captures, involved an interaction between a smart device and the smart home device's Cloud server. Once these interactions were identified, Wireshark was be used to study each attribute and its characteristics; the IP address of the Cloud site was identified, the domain that the IP belonged to, the protocol used for communication, if the payload was encrypted or not, the size of the payload, and how many transactions occurred during each session (*Wireshark*, 2020).

Once all of the possible smart device state changes were correlated to specific network traffic, the traffic was labeled to start training a machine learning algorithm. This step was particularly challenging since the publicly available data is not currently labeled. It was necessary to identify patterns in the encrypted network traffic indicative of a state change request to a smart home device. This involved making state change requests on

smart home devices in our test environment and comparing the traffic generated for each state change request to the traffic in the publicly available network traffic. The traffic patterns in the publicly available network traffic that match the traffic patterns in our test environment would have been labeled as state changes. Unfortunately, no state changes made by a smartphone were identified in the publicly available dataset. Therefore, a lab environment was set up and traffic was generated on the smart home devices listed above.

The steps to identify the state changes included exporting the network capture out of Wireshark as a comma separated value (CSV) file and then importing it into Microsoft Excel (*Microsoft Excel*, 2019). In Excel, a filter was applied to the data allowing us to separate network packets that include state change requests sent from the smart device to the Cloud site and unrelated network packets. A column was created in Excel to for the label. The value "1" was inserted into the label column for packets that involve a state change and a "0" for the rest.

Once this task was completed, the CSV was ready to be loaded into the Jupyter environment (*Jupyter Notebook 6.0.3*, 2020). This was done with a Pandas function which reads the csv into a Pandas dataframe. The Pandas dataframe is a two dimensional data structure that is comparable to the structure of a spreadsheet; the dataframe is comprised of rows and columns. The Pandas dataframe is the main container type used for all phases of machine learning (i.e. the cleaning, training, and analyzing steps) in the Scikit-Learn platform (Géron, 2019).

Step 3: Prepare the data – Feature Selection

One of crucial steps with machine learning, and therefore this research, was feature selection, formally represented as the set $F = \{f_1, f_2, \ldots, f_n\}$ in which $f_i (1 \leq i \leq n)$ represents a feature. Feature selection involved identifying all of the relevant characteristics of the home network traffic that were indicative of a state change request sent from a smart home device's app running on a smart device to a smart home device's Cloud site. State changes could be found in the payload of the packet. Therefore, network traffic attributes were selected as features to effectively train machine learning algorithms to identify patterns indicative of smart device state change requests. Table 1, below, summarizes the promising features, by category, including a reference to the research that implemented said feature.

Table 1

N*etwork Data Flow Features*

Category	**Feature**	**Reference**
Statistical	Average bytes per session from the client and from the server	OConnor et al. (2019); Ren et al. (2019)
Statistical	Maximum bytes per session from the client, and from the server	OConnor et al. (2019); Ren et al. (2019)
Statistical	Standard deviation of bytes between server sequences	OConnor et al. (2019)
Statistical	Standard deviation of bytes between IoT device sequences	OConnor et al. (2019)
Statistical	Median absolute deviation of packet size	Acar et al. (2018)
Statistical	Mean and standard deviation of the amount of traffic (bytes) sent or received by the device in consecutive s-second samples	Apthorpe, Reisman, & Feamster (2017)
Statistical	Burstiness the proximity of arrival instances within each other plus the variance between each arrival. It is measured by examining the variance in terms of both payload size and inter-arrival times	OConnor et al. (2019)

Aggregated	Aggregate bytes per session from the client and the server	OConnor et al. (2019)
Aggregated	Kurtosis with respect to packet sizes and inter-arrival times	Ren et al. (2019)
Aggregated	The distribution of inter-packet intervals	Apthorpe, Reisman, Sundaresan, et al. (2017)
Aggregated	Skewness with respect to packet sizes and inter-arrival times	Ren et al. (2019)
Synthesized	deciles of the distribution with respect to packet sizes and inter-arrival times	Ren et al. (2019)
Synthesized	IAT bin (a representation of traffic rate) bin index of packet inter-arrival time (IAT) using three bins: < 0:001 ms, 0:001 ms to 0:05 ms, and > 0:05 ms	Nguyen et al. (2019); Subahi & Theodorakopoulos (2019)
Synthesized	Mean inter-arrival time	Acar et al. (2018)
Synthesized	The total number of packets in a flow	Apthorpe et al. (2017); Bezawada et al. (2018)
Synthesized	total time of connection	OConnor et al. (2019)
Protocol specific	The proportion of SYN and ACK packets per flow Flow is a set of packets associated with a 5-tuple of sender_ip, recipient_ip, sender_port, recipient_port, and protocol within some time window	Apthorpe et al. (2017)
Protocol specific	Packet sequence information	Subahi & Theodorakopoulos (2019)
Protocol specific	Synchronicity, the observed measurements that describe how a client and server take turns sending data	OConnor et al. (2019)
Protocol specific	Synchronicity within the context of a session	OConnor et al. (2019)
Protocol specific	Synchronicity of server sequences per session	OConnor et al. (2019)

It is important to know if the same features, trained with the same machine learning classifiers, provide the best results across all types of smart home devices. Or is possible that the results will vary based upon the type of smart home device? The goal

was to identify which features, *F*, combined with which machine learning algorithm, provide the most optimal identification of state changes, across all types of smart home device. In other words, identify the subsets of *F*, denoted as *R*, that performs best with a machine learning algorithm:

$R = \{r_1, r_2, \ldots, r_m\}$ in which $r_i (1 \leq i \leq m)$ represents a subset of *F*

Statistical features have proven to work well with devices that have simple behaviors. As mentioned above, Acar et al. (2018), found that traffic size could be used to identify when a smart plug was turned on and off. Junges et al. (2019), were successful in determining state change requests of smart plugs and smart lamps by using the encrypted payload size as their main feature.

To the best of our knowledge, there is a lack of published research that has focused primarily on identifying the behavior of smart home devices with complex behaviors. Complex behavior of a smart home device is defined here as being able to choose to switch the device between more than two states (i.e. more than turning a light from on to off). An example is the Rachio smart sprinkler system which allows the user to turn the sprinklers on for five minutes. The Rachio accepts the request, turns the water on, then responds back when the five minutes of watering has completed. In order train a machine learning algorithm to identify this pattern, more complex features must be used than the statistical features described above. In this case, it was believed that synthesized features, such as total number of packets in a flow, would be most useful. Therefore, network traffic flow was a big focus. This helped identify time intervals between TLS sessions, which is important when identifying when a smart device state change request involves several interactions between the smart home device and the user.

Ren et al. (2019) found that aggregated features are effective in inferring interactions from devices that that send a large amount of data such as cameras, televisions, and audio devices. Although their goal was not to identify the optimal classifier, but to simply understand if the devices' activities are inferable.

Most fine-grained features did not help with identifying device state changes. However, several assisted in identifying the smart device requesting the change and the smart home device's Cloud site that the change request is sent to. Therefore, some fine-grained features were selected such as destination IP address and source MAC address. Source and destination ports were also used as features to identify when an encrypted payload is being sent.

The combinations of all the aforementioned features were tested in order to identify the ideal combination that performs best across all types of smart home devices. This involved testing several combinations of the feature sets as highlighted in Table 2 below.

Table 2

Possible Combinations of Feature Types

Feature type A	Feature type B	Feature type C
Statistical	Synthesized	Aggregated
Statistical	Aggregated	Protocol specific
Statistical	Protocol specific	Synthesized

Step 4: Identify promising models

The sets of features were used to train machine learning algorithms, also known as classifiers. Tests were conducted to see which machine learning algorithm, combined

with statistical, aggregated, synthesized or protocol specific features, performed best with traffic from smart home devices that have simple behavior, complex behavior, and send a lot of data. The classifiers can be formally represented as the set $C = \{c_1, c_2, \ldots, c_k\}$ in which $c(1 \leq i \leq k)$ represents a classifier. The goal was to identify which *c*, when trained with *f* performs the best:

$$S \subseteq C$$

S represents a subset of classifiers that produces the most accurate predictor out of the tested classifiers combined with the *R*, the selected features used to train the classifiers.

Acar et al. (2018) obtained an 88% accuracy of correctly detecting activities using Random Forest (RF) and 91% using KNN. Ren et al. (2019) trained a RF machine learning classifier. Junges et al. (2019) were able to train a KNN classifier to identify actions with a high accuracy of up to 98.4%. Therefore, this research project used the KNN algorithm to correlate smart device activity to network traffic, and explored other supervised learning algorithms such as RF and naïve Bayes.

Step 5: Fine-Tune the System

Several evaluations were performed to test the accuracy of the predictor. A root mean square error will be computed using numpy's built in mean_squared_error function (Géron, 2019). The function is:

$$RMSE(X, h) = \sqrt{\frac{1}{g}\sum_{i=1}^{g}(h(x^{(i)}) - y^{(i)})^2}$$

g is the number of instances in the dataset that are measuring the RMSE on. $x^{(i)}$ is a vector of all of the feature values of the i^{th} instance in the dataset, with $y^{(i)}$ representing its label. X is a matrix containing all of the feature values of all instances in the dataset. h, also called the hypotheses, is the system's prediction function.

This ran against all of the predictors; several predictors were created by combining features and running them through various machine learning classifiers.

Also, Scikit-Learn's K-fold cross validation feature was implemented. This feature randomly splits the training set into a set number of folds, which are distinct subsets of the training set. It then trains a classifier a set number of times, choosing a different fold for evaluation and using the other folds for training. The output is an array containing the evaluation score for all of the runs.

To identify correlations between features the standard correlation coefficient between every pair of features was computed. When the results were close to one, there was a strong positive correlation and when the result was close to negative one, there was a strong negative correlation. Scikit-Learn's corr() function was used to perform this calculation.

One very quick and effective test for accuracy is called the precision of the classifier. Precision is the accuracy of the positive predictors. It is represented by the equation:

$$precision = \frac{TP}{TP + FP}$$

TP is the number of true positives and FP is the number of false positives. Precision is usually used in conjunction with recall, also known as sensitivity or the true positive rate (TPR). It is calculated with the formula:

$$recall = \frac{TP}{TP + FN}$$

FN is the number of false negatives.

The harmonic mean of precision and recall, which is the F_1 score, was computed. It is different than precision in that it gives much more weight to low values. This results in only getting a high F_1 score if both recall and precision are high. The formula is:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

Finally, a confusion matrix was used to count the number of incorrect classifications. For example, it showed the number of times network traffic was incorrectly identified as containing a state change request for a smart home device. Scikit-Learn's cross_val_predict function will be used for the confusion matrix.

Preliminary Experiment

The dataset used for this preliminary experiment came from the work of Ren et al. (2019). A test case focused on network traffic generated by a Wemo plug over two separate dates. The researchers generated several GetBinaryState events in which the Wemo plug responded with its current state which is either on or off. The traffic and the payload were not encrypted, making it easy to identify the exact packets responsible for the GetBinaryState events. This was used to create an accurate label for the data.

The basis of this research was to learn the state change hidden in an encrypted payload. Since an encrypted payload cannot be read, it would not make sense to include an unencrypted payload in this test data. Therefore, the field containing the payload information was removed from the dataset prior to any training or testing.

The packet captures from April 24, 2019 was used to train the Random Forest Regressor and the packet capture from April 25, 2019 was used to test the resulting predictor. This was all performed in the Jupyter notebook environment using the Pandas and Numpy Python libraries to load and prepare the dataset. Once trained, the accuracy of the predictor will be tested using the mean square error formula. The result was zero, which indicates a perfect prediction. To double check this, the predicted output was compared to the test case and they were both identical.

The next step was to learn which features were the most important in training the predictor. In other words, which features had the highest correlations. The results, listed in Table 3 below, indicate that the Source and Destination ports are more correlated (.15) to GetBinaryState than Length is (.08). This is surprising since previous research placed a high emphasis on payload size to indicate a state change. This was be re-tested and analyzed during the research portion of this project.

Table 3

Correlation Matrix

	No.	**Time**	**Length**	**Src port**	**Dest port**	**GetBinary State**
No.	1.000000	0.999861	0.009366	0.019650	0.017530	0.001433
Time	0.999861	1.000000	0.009378	0.019745	0.017695	0.001206
Length	0.009366	0.009378	1.000000	-0.374397	-0.405949	0.086608
Src port	0.019650	0.019745	-0.374397	1.000000	0.841381	0.158049

	No.	Time	Length	Src port	Dest port	GetBinary State
Dest port	0.017530	0.017695	-0.405949	0.841381	1.000000	0.157310
GetBinaryState	0.001433	0.001206	0.086608	0.158049	0.157310	1.000000

This preliminary experiment showed promise that the research methods listed above were valid. It was important during the experiment portion of this research, to test several different features, with several different machine learning classifiers, to discover the optimal combination, to identify state changes hidden in the encrypted payload of network traffic. This research successfully discovered this combination.

Resources

Two main datasets were evaluated; one dataset provided by Alrawi et al. (2019) and the other by Ren et al. (2019). Evaluation of the datasets were done using Wireshark, Zeek, and Microsoft Excel. Since more data was necessary, network traffic was captured in a lab environment using tcpdump running on an OpenWRT router. The lab environment includeed the following smart home devices: Belkin WeMo switch, TP-Link WiFi Smart Plug, Rachio Smart WiFi Sprinkler Controller, the Chamberlain myQ garage door opener, and Amazon's Ring video camera.

Scikit-Learn was used to prepare the data, train the classifiers, and fine tune the system. Scikit-Learn contains several Python libraries that enable us to do this.

Summary

In summary, the research for the detection of rogue manipulation of smart home devices, involved the following steps:

1. Identify potential network traffic features that will enable the identification of smart home device state changes hidden in the encrypted payloads
2. Explore the publicly available network traffic datasets provided by Ren et al. (2019) and Alrawi et al. (2019)
3. Generate our own network traffic in our lab environment with a set of smart home devices
4. Train several machine learning algorithms with the identified feature sets
5. Evaluate the results of the trainings
6. Identify the machine learning algorithm, trained with one of the feature sets, that performs the best across all of the selected smart home devices

Chapter 4

Results

Overview

The main goal of this research was to categorize encrypted network traffic patterns originating from the user's WiFi connected smart phone or tablet, indicative of state change requests of smart home devices. To accomplish this goal, the research was conducted in three major phases:

1) focused on analyzing the publicly available network data captures provided by Alrawi et al. (2019), called YourThings, and the dataset published by Ren et al. (2019) called MonIoTrPublic,
2) involved capturing the network traffic of several smart home devices' apps and identifying the traffic patterns indicative of a state change request,
3) included identifying network traffic features, using those features to train machine learning algorithms, evaluating the results of the trainings, and identifying the machine learning algorithm that performs the best across all of the selected smart home devices. The results of each phase will be addressed in this chapter.

Phase 1 – Analysis of Publicly Available Datasets

Publicly available network traffic captures of smart home devices were rigorously searched for during the literature review portion of this project. This included contacting

researchers who had published papers related to smart home devices. This resulted in the successful identification and obtainment of two datasets.

One dataset was provided by Alrawi et al. (2019) as part of their YourThings initiative. The researchers evaluated 45 devices from several disparate categories including appliances, cameras, home assistants, media and network devices. They collected network traffic over a period of 13 days which resulted in over one thousand separate pcap files totaling 150 GB of data. The second dataset came from the work of Ren et al. (2019). The full dataset, referred to as MonIoTrPublic, includes network traffic captured from 81 different smart home devices located in labs in the United States and the United Kingdom. In-depth analysis of both datasets was performed.

The YourThings dataset presented some significant challenges due to both the quantity of pcap files and total data collected. Therefore, it was determined that a database should be used since this is the best method to store and query large quantities of data. A fork of the MySQL database, called MariaDB was selected as the database. All of the pcap files were loaded into MariaDB. This involved converting the pcap file to csv format, which results in the loss of a significant amount of detail. However, all of the relevant data to identify state changes was maintained; the csv file contained information about the source and destination IP addresses, ports, protocol used, bytes sent, bytes received, and the information field that includes a summary of data sent in the packet.

The main goal in evaluating the YourThings data in the MySQL database was to identify state change requests made from a smartphone or tablet. The SmartThings researchers included a mapping of device to IP address which helped with the initial identification of network traffic from the researcher's iPad and iPhone. Based on the

mapping, SQL queries were created to determine on which of the capture days' both the iPad and iPhone were used. Results showed that the iPad was used all four of the capture days (3/20/18, 3/21/18, 3/28/18, and 4/15/18) and the iPhone was used two of the days (3/21/18 and 3/28/18).

The next step was to identify network traffic from the iPhone and iPad to either one of the smart home devices or the smart home devices' Cloud sites. Once again, a SQL query was used which, not surprisingly, showed that both the iPhone and iPad generated a significant amount of outbound traffic to a multitude of various external IP addresses. It was not immediately apparent who owned the external IP addresses. To determine this, the Cloud sites for several of the smart home devices were identified, then a whois search was performed on each to determine who the IP range belonged to. This did not provide much useful information; most of the selected smart home devices host their Cloud server with Amazon's AWS service. Therefore, most of the whois queries resulted in AWS as the owner of the IP address. This presented the challenge of differentiating traffic bound for the disparate smart device Cloud sites. Fortunately, when the smart device's app on the iPad or iPhone is used, it generates a DNS query for the smart device's Cloud site. Using the database to track down DNS queries from the iPhone and iPad and Wireshark to open the corresponding pcap file, the Cloud site for each of the smart devices was identified.

Once the smart devices' Cloud sites were identified, it was possible to tell each time that the smart device's app was used on the iPhone or iPad. This only showed that the app sent traffic to the smart device's Cloud site, not exactly what was sent (i.e. was the app used to turn the smart switch on and off?). To determine if a state change was

requested, it was necessary to look for incoming traffic to the smart home devices and then see if that correlated to outbound traffic from the iPhone and iPad to the smart device's Cloud site. Using the previously stated method of discovering each IP address and which smart device it belonged to, it was possible to identify each time a smart device talked to its Cloud server.

Unfortunately, for all of the smart home devices listed above, there was not one instance when both the smart device's app and the smart device were sending or receiving network traffic within the same timeframe (i.e. with five minutes of each other). In other words, it did not appear that a state change was requested on the app since there was a lack of traffic during the same timeframe between the smart device and its Cloud site. To reiterate, for the traffic that spanned the four capture days, there were hundreds of instances of outbound traffic from several smart devices' apps to the Cloud sites and hundreds of instances of traffic to and from the smart devices to their Cloud sites. Yet, none of traffic overlapped in common timeframes.

One possible reason for this lack of correlation could be that the smart hubs and motion sensors used in the project generated state changes to the smart home devices, instead of the state changes being requested by the app on the iPhone. The smart hubs and motion sensors include: the Samsung SmartThings Hub, Phillips HUE Hub, Insteon Hub, Belkin WeMo Motion Sensor, Wink Hub, Caseta Wireless Hub, Google Home, and Apple HomePod. These hubs and sensors are designed to directly control the smart home devices without the end user controlling the device via an app.

It is possible to view the smart hub setup as delegitimizing this research since the intention of this project was to focus on state changes initiated from a smartphone.

However, it could also be argued that the setup used in the YourThings research project is advanced and not typical of most smart home device users today. The hub and sensor setup would require home users to purchase additional equipment, for several hundred dollars, plus the home user would need to know how to program the hubs and sensors to perform certain tasks for each corresponding event. This may be out of the technical know-how of most home users, since Fu et al., (2017) found that most home users are not tech savvy.

Despite the limited use of hubs, it is possible that the techniques used in this research could be applied to identify state changes sent from a hub to a smart device's Cloud site. Just like smart phones, hubs send state change requests over the home WiFi network to the device's Cloud site. Therefore, machine learning algorithms could be trained to recognize state change patterns sent via encrypted network traffic from hubs similarly to the success this research had with state changes sent from smart phones.

Additionally, there is a growing number of smart assistants which allow users to control their smart home devices via voice commands. For example, Amazon Alexa can be configured to allow a user to turn a smart switch on simply by speaking the command to the Alexa. The Alexa, just like a smart phone and a hub, then sends the state change request to the smart home device's Cloud site over encrypted WiFi network traffic. Once again, it is possible that this research is applicable to discovering state changes sent from a smart assistant.

The evaluation of the MonIoTrPublic dataset was accomplished in just a few steps. The MonIoTrPublic dataset contained network traffic for over twenty smart home devices, but it did not include any network traffic from a smartphone or tablet.

Phase 2 – Capturing and Identifying Network Traffic of Smart Home Devices

Since the two publicly available datasets did not contain any identifiable state changes made from a smartphone or tablet to a smart home device, it was necessary to set up a lab to capture this traffic. The focus of this research was to identify state changes across three types of categories of smart home devices: 1) has a simple behavior (turns on and off), 2) has a complex behavior (can be turned on for a set amount of time), and 3) sends a large amount of data (i.e. video camera). Therefore, smart home devices were selected across those three categories.

Selecting the Smart Home Devices

For the simple behavior, the TP-Link WiFi Plug and the Chamberlain myQ Garage Opener were selected. The TP-Link WiFi Plug is only capable of turning a switch on and off, while the Chamberlain myQ Garage Opener's sole purpose is to open and close a garage door. The Rachio Smart Sprinkler system is capable of turning the sprinklers on and off for varying amounts of time, which is a more complex behavior than simply turning something on and off or opening and closing something, so it was placed in the complex behavior category. Finally, Amazon's Ring video camera constitutes a device that sends a large amount of data which fits the final category.

Two other devices were initially selected but later excluded due to not being able to reliably identify their state change requests. The Belkin WeMo Switch, which was used in the preliminary tests for this research, previously sent state changes in unencrypted traffic but has since improved their security and now encrypts their traffic. All of the other smart home devices in this research encrypt their traffic as well, but identifiable patterns enabled the identification of state changes in their traffic. The Belkin

WeMo Switch did not exhibit identifiable patterns that would allow for the labeling of state changes necessary to a train machine learning algorithm. This is also true for the Fujitsu Mini-Split heating and air conditioner. This device was initially included since it allows the user to change several different settings of the system, putting it in the complex behavior category. To include these two devices in future research it would be necessary to either decrypt the network traffic or work with the manufacturer to understand how their systems function.

Table 4

Smart Home Devices with Specifications Used in this Research

Device	**Firmware version**	**Functionality**	**App version**	**Utilized in research**
TP-Link WiFi Plug	1.5.6	Turn a plug on and off	2.23	Yes, state change traffic identified
Chamgerlain myQ Garage Opener	A.0.4.12	Open and close a garage door	5.158.19859	Yes, state change traffic identified
Rachio Smart Sprinkler system	5-115	Turn on a sprinkler system to various amount of time	4.1.12	Yes, state change traffic identified
Amazon Ring video camera	Cam-1.4.1.5200	Switch to live mode to stream a live feed	5.28.0	Yes, state change traffic identified
Belkin WeMo Switch	2.00.11420	Turn a plug on and off	1.25.1	No, unable to reliably identify the state changes
Fujitsu Mini-Split system	2.4.5.1	Control a heating and air system by setting the temperature, fan speed, and more	3.1.0	No, unable to reliably identify the state changes

Capturing the Network Traffic

The environment and method used to collect network traffic were based on the same methods used by Ren et al. (2019) and Alrawi et al. (2019). The environment, based on the lab setup of Alrawi et al. (2019), consisted of a GL.iNet Mini Travel Router running OpenWRT, an iPhone, and smart home device. During testing, the iPhone and the smart home device were configured to connect to the mini travel router (see Figure 5 below).

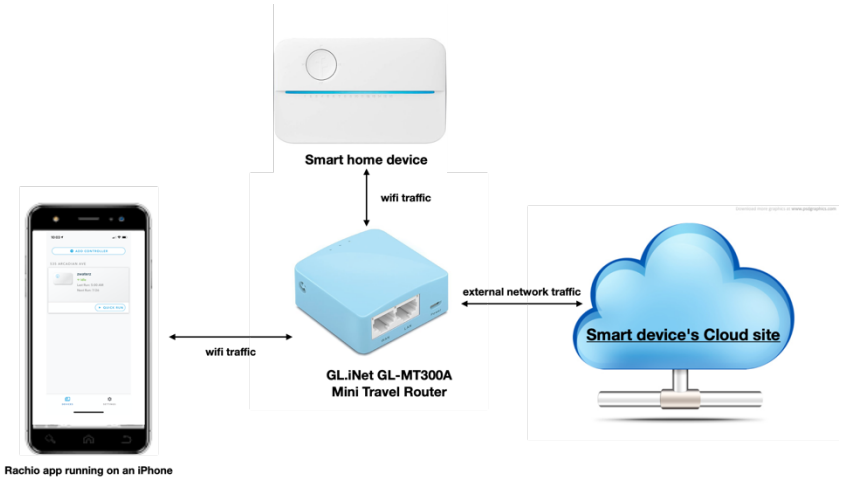

Figure 5. Lab configuration with an iPhone running the smart home device's app, the smart home device connected to a Mini Travel Router, and the Mini Travel Router connected to smart device's Cloud site(s)

The method used to capture the traffic followed what Ren et al. (2019) termed interaction experiments. This involved interacting with the IoT device via the app on the iPhone. For each interaction, the device would be turned on, and then two minutes later the network capture would be initiated. The capture would continue during the entire time the state change, or changes, took to complete, then after an additional 5-15 seconds, the capture would be stopped (Ren et al., 2019b).

The following are the exact steps taken for each network capture:

1. Change the iPhone's WiFi settings to connect to the mini router

2. Configure the smart home device to connect to the mini router
3. SSH into the mini router from a MacBook Pro
4. Launch the tcpdump utility on the mini router
5. Disconnect the MacBook Pro from the mini router
6. Launch the smart device's app on the iPhone
7. In the app, make a state change to the smart home device (i.e. turn the smart switch off)
    a. In some cases, repeatedly perform this action
    b. In some cases, don't perform any state changes in the app which will be used to differentiate between state change traffic and all other traffic
    c. Record the time the state change was made
8. SSH back into the mini router from the MacBook Pro and copy the pcap file to the MacBook Pro for analysis

Identifying State Changes

The next step was to analyze the network traffic in order to identify the state changes that were made. To accomplish this, the network analysis tools Zeek and Wireshark were leveraged. Zeek was used to summarize each connection. The utility creates several files detailing information about the network traffic such as the source and destination of each connection, how much data was sent, the protocol that was used, and several other informative fields (see Appendix A).

All of the outbound connections from the iPhone were analyzed using Zeek, which includes a feature to parse pcap files creating several log files (Copos et al., 2016a; Dai et al., 2019; Flosbach et al., 2019; Paxson, 1999). This research followed the example

of Flosbach et al. (2019), who successfully parsed pcap files using Zeek. One of the log files that was created, the conn.log files, records connection information of the network and transport layer including information such as when a connection occurred, for how long, the protocol used, and several other details (see example data in Table 5 below). This information was used to identify traffic patterns for each device. A detailed account of how each pattern was identified for each device is detailed in the sections below. Once the state change patterns were identified, they would be validated by matching the state change patterns in the network traffic with the recorded times a state change was made on the app on the iPhone. If the times matched up, then it was more likely that the identified state change was a true positive and not a false positive.

Table 5

Example of fields and values produced by Zeek on network traffic sent between an iPhone a smart device's Cloud site

Field	**Value**
time stamp	2020-06-26T14:23:21-0700
source IP	192.168.8.248
source port	55451
destination IP	13.83.97.206
destination port	443
protocol	tcp
service	ssl
duration	0.372338
orig_bytes	1152
resp_bytes	7246
conn_state	S1 (Connection established, not terminated)
orig_pkts	24
orig_ip_bytes	3552
resp_pkts	26
resp_ip_bytes	15836
server name	api.myqdevice.com

To add further evidence that the traffic pattern was indicative of a state change, network traffic sent between the smart home device and its Cloud site was analyzed. Since the mini router captured all of the network traffic, not just the traffic between the smartphone and the Cloud site, the traffic between the smart home device and the Cloud site was contained in the same conn.log file as was used in the previous step. The timestamp of the identified state change event was correlated with the timestamp of a connection, or connections, made from the smart home device to its Cloud site (see Table 6 below). If the identified state change traffic was sent to the Cloud site within a given timeframe of traffic being sent between the smart home device and the Cloud site, then this further legitimizes that it is indeed a state change. Junges et al. found that sessions are usually within 2.5 seconds of each other (Junges et al., 2019). Therefore, if the state change request sent from the smartphone and the state change sent from the Cloud to the smart home device are within 2.5 seconds of each other, then they will be considered linked.

Table 6

Example of fields and values produced by Zeek on network traffic between a smart device and its Cloud site

Field	**Value**
time stamp	2020-06-26T14:23:21-0700
source IP	20.42.27.108
source port	8883
destination IP	192.168.8.206
destination port	50854
protocol	tcp
service (application protocol)	-
duration	24.947513
orig_bytes	85
resp_bytes	425
conn_state	OTH (No SYN seen, just midstream traffic)

orig_pkts	14
orig_ip_bytes	730
resp_pkts	14
resp_ip_bytes	1410
server name	Microsoft Corporation (MSFT)

Once all of the state changes were confirmed a label was added to mark the state changes. For each device a separate number was used to represent a state change; a one was used for TP-Link, a two was used for the Ring, a three for the Rachio, and a four was used for the myQ device. Next, all of the labeled data per device was combined into one csv file. These files contained the network traffic captures for each device combined with the captures that did not have state changes.

iPhone versus Android app state change patterns

This research chose to run the smart device's app on an iPhone and not an Android device. While it is possible that the Android app is programmed differently, it is highly unlikely that an entirely separate Cloud infrastructure is implemented since smart home device manufacturers often leverage software development kits (SDKs) provided by smart home platform providers, such as Amazon's AWS IoT service (Zhou et al., 2019). This would indicate that the Cloud infrastructure remains the same between smartphone apps, which lends credence to the notion that the mobile app would behave similarly across platforms.

Identifying State Changes of TP-Link State Changes

The state change requests for the TP-Link WiFi Plug were the easiest out of all the smart home devices to identify. To establish a pattern, the plug was turned on or off 24 times, spread relatively evenly, over a period of three days (6/20/20, 6/22/20, and 6/26/20). The first pattern that emerged was that whenever the plug was turned from on

to off or from off to on, a new connection was created between the iPhone and the TP-Link WiFi Plug directly. The traffic sent was encrypted, however the values of the attributes id.resp_p, orig_bytes, and resp_bytes were identical every time the plug was switched from off to on and from on to off (see Table 7 below). It is worth noting that there is no guarantee that these values will always remain the same. If the vendor changes the encryption algorithm that they use or makes alterations to their code controlling the state changes, it is very likely that different values will be used. However, it is possible that if the aforementioned changes are made, new patterns will emerge which could then be used to identify state changes.

Table 7

iPhone to TP-Link WiFi Plug captured traffic of a state change, made on the iPhone (192.168.8.248), to the TP-Link device(192.168.8.247)

Orig host IP	Orig port	Resp host IP	Resp port	Proto	Duration	Orig bytes	Resp bytes	Conn state
192.168.8.248	52791	192.168.8.247	9999	tcp	0.188	106	49	SF
192.168.8.248	52792	192.168.8.247	9999	tcp	0.250	106	49	SF
192.168.8.248	52794	192.168.8.247	9999	tcp	0.190	106	49	SF
192.168.8.248	52796	192.168.8.247	9999	tcp	0.208	106	49	SF

Second, the app would communicate with the following TP-Link Cloud sites: n-use1-wap.tplinkcloud.com, n-wap.tplinkcloud.com, and api.tplinkra.com. If the app was simply opened and no state change was made, there would be no connection made to the TP-Link plug. However, the app would connect to the same three TP-Link Cloud sites listed above.

Identifying State Changes of Chamberlain myQ Garage Opener

The Chamberlin myQ Garage Opener app behaved more similarly to a typical smart home device app since it did not communicate directly with the myQ device directly, instead it communicated state changes to the myQ Cloud site. Notra et al. (2014) made this observation several years ago, and based on our experiments, this is still true today. In this experiment ten separate state changes, in the form of opening or closing the garage door, were captured over four days (6/12/20, 6/13/20, 6/23/20, and 7/20/20). A pattern emerged in which the app on the iPhone communicated with the account-devices-gdo.myq-cloud.com Cloud site whenever the door was either opened or closed. The Cloud site consistently had the IP address 13.83.240.23 in our experiments (see Table 8 below). This is unusual behavior since in the network captures for the other smart home devices, the IP address associated with a Cloud site would change for each capture and sometimes several different IP addresses were used during the same capture.

Table 8.

Connections involving state changes sent from the iPhone to the myQ Cloud site (this table is abbreviated, the full table can be found in Appendix A)

Resp host IP	Duration	Orig bytes	Resp port	Conn state	History	Orig packets	Orig IP bytes	Resp packets	Resp bytes
13.83.240.23	0.3476	1121	5746	S1	ShADTadt	24	3490	24	12732
13.83.240.23	0.3241	1248	5746	S1	ShADTadt	24	3744	24	12732
13.83.240.23	0.4003	1248	5746	S1	ShADTadt	22	3640	24	12732
13.83.240.23	1.0862	1174	434	S1	ShADTadt	20	4422	20	1900
13.83.240.23	0.4760	1249	5746	S1	ShADTadt	24	3746	24	12732
13.83.240.23	0.3590	1249	5746	S1	ShADTadt	24	3746	24	12732

Another pattern can be observed in the state changes listed in Table 8. Most of state changes have very similar values across all of the fields, with the outlier being the

fourth row: the duration, origination bytes sent, response bytes, connection state, history, origination packets, origination IP bytes, response bytes, and response IP bytes are all in the same range.

To validate that the pattern was indicative of a state change request, the recorded time that the state change was made on the iPhone matched up with the time listed in Table 8 above. Additionally, the time the change request was seen in the traffic from the iPhone to the Cloud site correlates with traffic identified between the myQ device and the myQ device's Cloud site (see Table 9 below). Looking at the actual beginning and ending times of each connection, Wireshark shows that the app on the iPhone started its connection at 10:38:06.189218000 PDT and ended it at 10:38:06.430220000 PDT. This was almost immediately followed by the connection between the myQ device and the myQ Cloud site which lasted from 10:38:06.4895 PDT until 10:38:31.4370 PDT.

Table 9.

Sample state change traffic sent from the iPhone (192.168.8.248) to the myQ Cloud site (13.83.240.23) and then from the myQ Cloud site (20.42.27.108) to the myQ device (192.168.8.206)

Timestamp	Orig host IP	Orig port	Resp host IP	Resp port	Proto	Duration	Orig bytes	Resp bytes	Conn state	Orig pkts
2020-06-23T10:38:06-0700	192.168.8.248	55447	13.83.240.23	443	tcp	0.359	1249	5746	S1	24
2020-06-23T10:38:06-0700	20.42.27.108	8883	192.168.8.206	50854	tcp	24.947	85	425	OTH	14

Identifying State Changes of Ring

The state change for the Ring video camera is the act of putting the Ring into Live mode allowing the end user to view the live feed from the camera. Analysis of the network traffic led to the discovery that Ring uses the Real-time Transport Protocol (RTP) to stream live video. The precursor to RTP traffic is Session Initiation Protocol

(SIP) traffic, since it is responsible for setting up the connection between the Ring Cloud site and the iPhone. Table 10, below, shows the SIP traffic which is indicative of putting the Ring camera into Live mode captured over four different days (6/24/20, 7/5/20, 7/12/20, and 7/13/20). This traffic was labeled as state changes and used to train the classifiers which is discussed later in this chapter.

Table 10

Four instances of network traffic containing Ring Live mode requests sent between the iPhone and the Ring Cloud site

Resp host IP	34.223.30.139	44.226.215.196	34.223.30.114
Resp port	15064	15064	15064
Proto	tcp	tcp	tcp
Duration	4.544156	14.338902	5.820299
Orig bytes	4392	4437	4088
Resp bytes	7451	7471	6753
Conn state	S1	S1	S1
History	ShADTadttT	ShADTadttT	ShADTadtTt
Orig pkts	48	60	48
Orig ip bytes	11304	11922	10648
Resp pkts	40	48	36
Resp ip bytes	16998	17454	15394

Identifying State Changes of the Rachio Smart Sprinkler System

The Rachio app was capable of performing a more complex behavior than simply turning a device on and off or opening or closing a door; the Rachio app allows a user to choose a sprinkler system zone to run and for how long it is to run. To start a “quick run” of the sprinklers, the mobile app first communicates with a server that resolves to api-service-prod.us-west-2.elasticbeanstalk.com. Over SSL it sends out about 1,000 bytes and then 407 bytes. Next the mobile app sends the request to start the “quick run” by

connecting to a second Cloud site identified as rach.io and sends 10,000 bytes of data and receives several tens of thousands of bytes from the Cloud server. One unique finding is that in some cases both the mobile app and the Rachio device both connected to the same Cloud site.

This exact behavior was seen for 10 other "quick runs" performed over several different days (6/12/20, 6/23/20, 6/24/20, 7/5/20, and 7/13/20). To further validate this state change, a network traffic capture was performed in which the Rachio app on the mobile app was used to view the watering schedule, but not make any changes (like a "quick run"). This time the Cloud site identified as rach.io was not accessed, confirming that the Rachio app performs this identifiable pattern only when performing a quick run. To summarize, the only time the Cloud site identified as rach.io was accessed was when a quick run was requested. Therefore, network traffic connections between the mobile app and rach.io were labeled as state changes for the Rachio (see Table 11 below).

Table 11

Three instances of network traffic containing quick change requests sent between the iPhone and the Rachio Cloud site

Resp host IP	34.213.56.42	34.213.56.42	52.32.41.198
Resp port	443	443	443
Proto	tcp	tcp	tcp
Duration	23.285352	28.606642	15.913739
Orig bytes	9217	9239	9230
Resp bytes	65852	66284	66837
Conn state	S1	S1	S1
History	ShADTadttT	ShADTadttT	ShADTadttT
Orig pkts	208	202	190
Orig IP bytes	29178	28934	28316
Resp pkts	212	200	190
Resp IP bytes	142978	144060	144646

Phase 3 – Train and Evaluate Machine Learning Algorithms to Identify State Changes

The main goal of this phase was to identify the combination of features, F, and machine learning classifiers, C, which are the most successful in identifying state changes hidden in the encrypted payloads across several types of smart home devices. This involved identifying the features where $F = \{f_1, f_2, \ldots, f_n\}$ in which $f_i(1 \leq i \leq n)$ F is comprised of the connection summaries produced by the Zeek utility when run on the network traffic captures of the interaction experiments. The next step was to identify a subset of F. This was represented by R, where $R = \{r_1, r_2, \ldots, r_m\}$ in which $r_i(1 \leq i \leq m)$, includes the most promising features that were identified to train the classifier, C. This culminated in identifying which subset S ($S \subseteq C$) when trained with R performs the best.

The following steps were used to train the machine learning algorithms: 1) get the data, 2) explore the data, 3) prepare the data, 4) identify promising models, and 5) fine tune the system. Step 1 and 2 were covered above in Phase 2 – Capturing and Identifying Network Traffic of Smart Home Devices. The rest of the steps will be covered here.

Prepare the Data

One of crucial steps of this research is feature selection, F. The goal of this step was to identify $F = \{f_1, f_2, \ldots, f_n\}$ in which $f_i(1 \leq i \leq n)$. To ensure a robust feature set, features were selected from one of the three categories: 1) aggregated, 2) synthesized, and 3) protocol specific. This resulted in several features being selected from each category. The list of features selected by category can be found in Table 12 below.

Table 12

Features selected by category with a description of each feature

Category	Abbreviation	Description
Aggregated	orig_pkts	Number of packets that the originator sent
Aggregated	orig_ip_bytes	Number of IP level bytes that the originator sent (as seen on the wire, taken from the IP total_length header field).
Aggregated	resp_pkts	Number of packets that the responder sent.
Aggregated	resp_ip_bytes	Number of IP level bytes that the responder sent (as seen on the wire, taken from the IP total_length header field).
Synthesized	history	Records the state history of connections as a string of letters (such as SO, Connection attempt seen, no reply, and SF, normal establishment and termination).
Synthesized	conn_state	The state of the TCP connection which involves a combination of one, several, or none of the following packet types: SYN, ACK, FIN and RST
Synthesized	duration	How long the connection lasted. For 3-way or 4-way connection tear-downs, this will not include the final ACK.
Protocol specific	id.orig_p	The originator's 4-tuple of endpoint port.
Protocol specific	id.resp_h	The responder's 4-tuple of endpoint address.
Protocol specific	id.resp_p	The responder's 4-tuple of endpoint port.
Protocol specific	proto	The transport layer protocol of the connection.
Protocol specific	orig_bytes	The number of payload bytes the originator sent.
Protocol specific	resp_bytes	The number of payload bytes the responder sent. See orig_bytes.

Identify Promising Models

The outcome of this step was to identify which c, when trained with f, produces the most accurate predictor out of the tested classifiers, $S \subseteq C$. Acar et al. (2018) obtained an 88% accuracy of correctly detecting activities using Random Forest (RF). Therefore, the RF algorithm was selected as a starting point. To evaluate the effectiveness of RF, a 4-fold cross validation was performed. This involved randomly

splitting the training set into five distinct subsets, training and evaluating the model 4 times, picking a different fold for evaluation each time, and then training on the other 4 folds.

This step involved two phases: 1) training each smart home device individually with the selected features and RF, and 2) training with the combined set of smart home devices with all state changes labeled the same. The results all both phases by device are presented in the following sections.

TP-Link RF Training

The first training of the RF classifier on the TP-Link device involved all 13 of selected features listed above. RF aggregates and then produces a mean of several Decision Trees all trained from different random subsets of the network traffic (James et al., 2013). The training data, the collection of which was described in phase 2 above, resulted in 430 instances of no state change requests and 24 instances of state change requests. It is important to reiterate that the training data, collected from the network traffic of all the smart home devices, contained instances of state changes, smart device app use but no state changes, and traffic from non-smart device apps. The TP-Link data was collected over a period of three days (6/20/20, 6/22/20, and 6/26/20). It is believed that the traffic patterns identified as state changes will stay the same until a major software change is applied. Therefore, three days of traffic was deemed sufficient.

A 4-fold cross validation was performed resulting in cross validation scores of 100% across all folds. The root mean square score was also 0 across all folds, with a mean of zero and a standard deviation of zero. The recall, precision, and F_1 scores were calculated. These also resulted in perfect scores of 100%.

The next step was to reduce the number of features by identifying the most important features used to train the RF classifier. The Jupyter notebook environment using the Pandas and Numpy Python libraries was used during the entire machine learning process to identify the most important RF features. The most popular method used in this environment is an RF grid search (Géron, 2019). An RF grid search asks the RF classifier to rank each feature by order of importance. The code to perform this task is contained in the sklearn library GridSearchCV. The parameters for GridSearchCV were leaf_size set to 30, n_jobs of none, n_neighbors equal to two and p = 2. The results are listed in Table 13 and Figure 6 below.

Table 13

RF feature importance scores for TP-Link traffic

Feature	**Value**
Resp bytes	0.28819812
Orig bytes	0.11335768
Orig IP bytes	0.09588693
Resp IP bytes	0.08113802
Resp port	0.06382875
Resp pkts	0.03410074
Orig pkts	0.02077022
Orig port	0.01948455
Duration	0.01168756

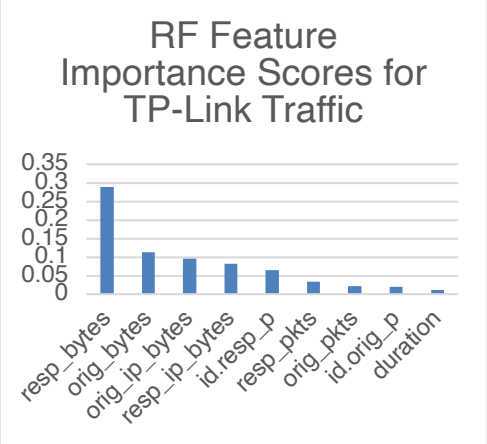

Figure 6. Feature importance scores for TP-Link traffic determined by the Random Forest classifier

The lead field, with over a quarter of the overall importance is resp_bytes, represents the number of payload bytes the responder sent. This is followed by the number of payload bytes the originator sent (orig_bytes) and the number of IP level bytes that the responder sent (orig_ip_bytes). With the feature set reduced down to three features, $F = \{\text{resp\_bytes}, \text{original\_bytes}, \text{orig\_ip\_bytes}\}$ a 4-fold cross validation with the RF classifier was run. Once again, this resulted in perfect scores for the cross validation across all folds, the root mean square score across all folds, and for the precision, recall, and F_1 scores. With perfect scores, it was obvious that the three identified features combined with the RF classifier could not be beat.

myQ RF Training

Once again, the RF classifier was used with all 13 features. Also, the training data which included 1,216 connections without a state change and eight with a state change, was collected as outlined in phase 2 above. The identical 4-fold cross validation, used with the TP-Link device was used and resulted in cross validation scores of 100%, 99.673%, 100%, and 99.346%. The root mean squared errors were 0, .228, 0, and .323. The mean was .138 with a standard deviation of .142. The precision, recall, and F_1 scores were also all 100%. An RF grid search produced the results in Table 14 and Figure 7 below. This indicated that the most important feature is the responding server's IP address (id.resp_h), with over a quarter of the overall importance. The next two most important features are orig_bytes and duration.

Table 14

RF feature importance scores for myQ

Feature	Value
Resp host IP	0.273051817
Orig bytes	0.178656021
Duration	0.12654089
Orig IP bytes	0.066071173
Resp bytes	0.041034603
Orig pkts	0.035720589
Orig port	0.034761846
Resp IP bytes	0.026948269

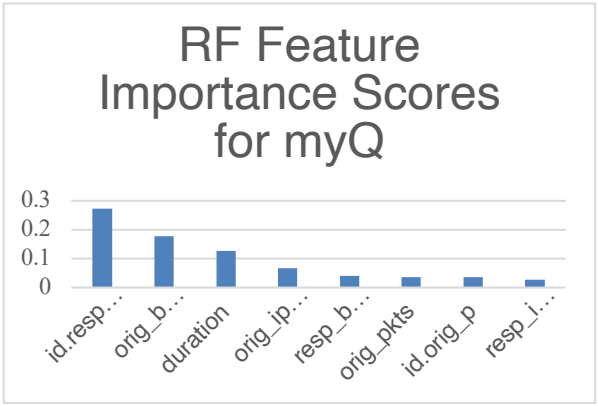

Figure 7. Feature importance scores for myQ traffic determined by the Random Forest classifier

The RF classifier was run again with the three most important features $F = \{\text{id. resp\_h}, \text{orig\_ip\_bytes}, \text{duration}\}$. This resulted in cross value scores of 99.673%, 100%, 100%, and 99.673%. Root mean square errors of 0, 0.228, 0, and 0.228, a mean of 0.0571, and a standard deviation of 0.099. The precision, recall, and F_1 scores were also all 100%. This validated the most important features and resulted in a strong RF predictor. Attempting to reduce the number of features to less than the three listed above, resulted in lower precision, recall, and F_1 scores.

Ring RF Training

Data collected from phase 2 was used to train the RF classifier. This included all 13 features, with training data that included 463 connections without a state change and eight with a state change. The same 4-fold cross validation was repeated, resulting in cross validation scores of 100%, 99.152%, 100%, and 100 %. The root mean squared errors were 0, .184, 0, and .184. The mean was .054 with a standard deviation of .092. The precision, recall, and F_1 scores were also all 100%. An RF grid search produced the results in Table 15 and Figure 8 below. This indicated that the most important feature is the responding server's IP address (id.resp_p), followed by orig_ip_bytes and resp_ip_bytes.

Table 15

RF feature importance scores for Ring

Feature	**Value**
Resp port	0.203392693
Orig IP bytes	0.10158929
Resp IP bytes	0.09997153
Duration	0.042720683
Orig bytes	0.035100149
Resp bytes	0.034483921
Orig port	0.033622461

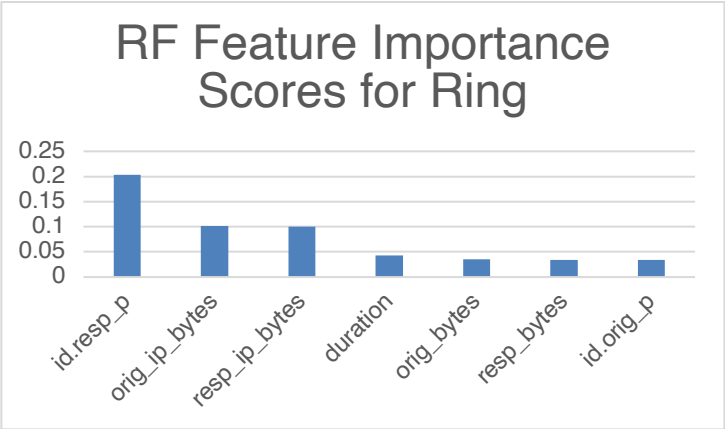

Figure 8. Feature importance scores for Ring traffic determined by the Random Forest classifier

The three most important features $F = \{id.\,resp\_p, orig\_ip\_bytes, resp\_ip\_bytes\}$ were used to train the RF classifier resulting in cross value scores of 100%, 100%, 99.236% and 100 %. Root mean square errors of 0, 0. 0.174, 0, and 0, a mean of 0.043, and a standard deviation of 0.075. The precision, recall, and F_1 scores were also all 100%. This once again validated the most important features and resulted in a strong RF predictor. Attempting to reduce the number of features to less than the three listed above, resulted in lower precision, recall, and F_1 scores.

Rachio RF Training

The final device was trained in the exact same way as the previous devices; the RF classifier was used with all 13 features, with training data that included 1,019 connections without a state change and eleven with a state change from the data collected during phase 2. The 4-fold cross validation resulted in cross validation scores of 98.449%, 99.224%, 99.221%, and 99.221%. The root mean squared errors were all .264. The mean was 0.264 with a standard deviation of .0003. The precision, recall, and F_1 scores were also all 100%. An RF grid search produced the results in Table 16 and Figure

9 below. This indicated that the most important feature is orig_pkts, followed by resp_ip_bytes and orig_ip_bytes.

Table 16

RF feature importance scores for Rachio

Feature	Value
Orig pkts	0.15045
Resp IP bytes	0.12303946
Orig IP bytes	0.11599227
Resp pkts	0.09410367
Orig bytes	0.06746237

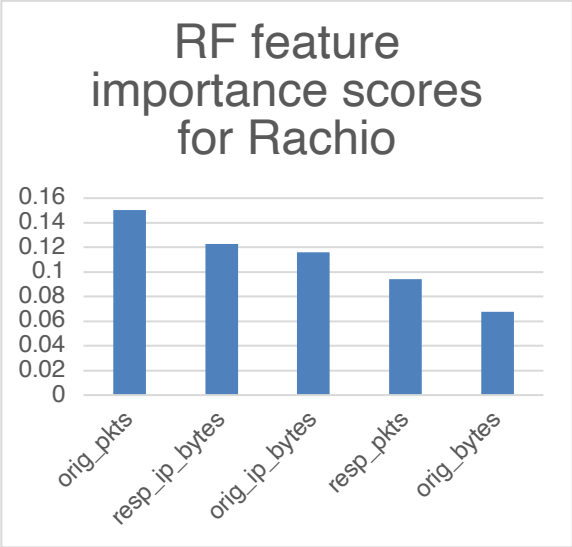

Figure 9. Feature importance scores for Rachio traffic determined by the Random Forest classifier

The three most important features $F = \{orig\_pkts, resp\_ip\_bytes, orig\_ip\_bytes\}$ were used to train the RF classifier resulting in cross value scores of 98.44961%, 99.225%, 99.222% and 99.222%. Root mean square errors of .374, .323, .265, and .265, a mean of 0.307, and a standard deviation of 0.046. The precision, recall, and F_1 scores were also all 100%. This validated the most important features and resulted in a strong

RF predictor. Once again, attempting to reduce the number of features to less than the three listed above, resulted in lower precision, recall, and F_1 scores.

Combination of all the Devices' Traffic

The culminating effort is to identify any state change, regardless of what device it was produced for, across all three types of smart home devices. All of the files from all of the device trainings listed above were combined. All of the state changes were labeled with a one. This resulted in a test set consisting of 3,128 connections without a state change and 51 with a state change. Once again, the same process was followed for training the classifiers on the state changes for each smart home device; the 4-fold cross validation was run to calculate the cross-validation scores, root mean squared errors and mean. Again, a confusion matrix was calculated, plus the precision, recall and F_1 score. This time three different classifiers, Random Forest, Naïve Bayes, and K-Nearest Neighbors, were used across multiple combinations of features to identify which subset S ($S \subseteq C$) when trained with R performs the best.

Random Forest on the Combination of all the Devices' Traffic

The 4-fold cross validation using RF resulted in cross validation scores of 98.365%, 98.365%, 99.119%, and 99.244%. The root mean squared errors were .128, .1279, .094, and .094. The mean was .11087019 with a standard deviation of .0170055. The confusion matrix was also about the same $\begin{bmatrix} 3128 & 0 \\ 20 & 31 \end{bmatrix}$, resulting in a precision score of 99.682%, recall score of 80.392%, and F_1 score of 87.645%. The next goal was to drastically improve these scores. To do so, the RF feature importance scores were used to choose which features to include in the next training round (see Table 17 below). The

features all had about the same importance rating, with the majority of features' importance ranging between 7-9%.

Table 17

RF feature importance scores for all of the devices combined

Feature	Value
Resp bytes	0.09419
Orig IP bytes	0.0800405
Resp IP bytes	0.07896994
Orig bytes	0.0725435
Orig port	0.07226243
13.83.240.23	0.07168504
Resp pkts	0.04634177
Orig pkts	0.04617784

The top three features were chosen to rerun through the system (resp_bytes, orig_ip_bytes, and resp_ip_bytes). This resulted in cross value scores were 98.36478%, 98.365%, 99.748%, and 99.496%. Root mean square errors of .128, .128, .050, and .071, a mean of .094, and a standard deviation of 0.034. The precision score was 99.651%, recall was 78.431%, and F_1 score of 86.07%. This was not much of an improvement.

In an attempt to improve the training of the system, a fourth feature was added, orig_bytes. This slightly improved most of the scores, but more importantly, had big effects on recall and on the F_1 score. The recall score improved from 78.431% to 87.254% and the F_1 score went from 86.07% to 92.59%. Both of these are very encouraging improvements.

To see if these scores could be improved any further, the fifth feature, id.orig_p, was added. This resulted in almost perfect cross value scores of 99.057%, 99.214%, 99.843%, and 99.213%. Root mean square errors of .089, 0.089, 0.039, and 0.069, a mean of .071, and a standard deviation of 0.020. The most encouraging results were the

precision, recall and F_1 score. The precision was 99.900%, which is hard to improve. The recall was 93.902 % and the F_1 score was 96.703%. A summary of the features used and their scores can be found below in Table 18 and Figure 10. These results indicate that it is possible to identify state changes from disparate types of devices in the encrypted traffic sent from an iPhone to the smart devices' Cloud sites. One interesting finding in the Table 18 below, is that the F_1 score is lower when all of the features were used to train the RF algorithm as opposed to when it is trained with a select subset of features. The reason for this could be that some of the features are misleading in terms of identifying state change patterns. When those misleading features were removed, then the F_1 score improved.

Table 18

Features used for RF training and their resulting scores

Features	**Precision**	**Recall**	**F_1**
id.orig_p, id.resp_h, orig_bytes, resp_bytes, orig_ip_bytes, resp_ip_bytes	0.99524941	0.70731707	0.79071681
resp_bytes, orig_ip_bytes, resp_ip	0.99650794	0.78431373	0.86074785
All features	0.99698125	0.81372549	0.88402822
All features except duration, conn_state, history, resp_pkts	0.9139747	0.92993393	0.92179774
orig_bytes, resp_bytes, orig_ip_bytes, resp_ip_bytes	0.9979306	0.87254902	0.92592944
id.orig_p, orig_bytes, resp_bytes, orig_ip_bytes, resp_ip_bytes	0.99840663	0.90196078	0.94485422

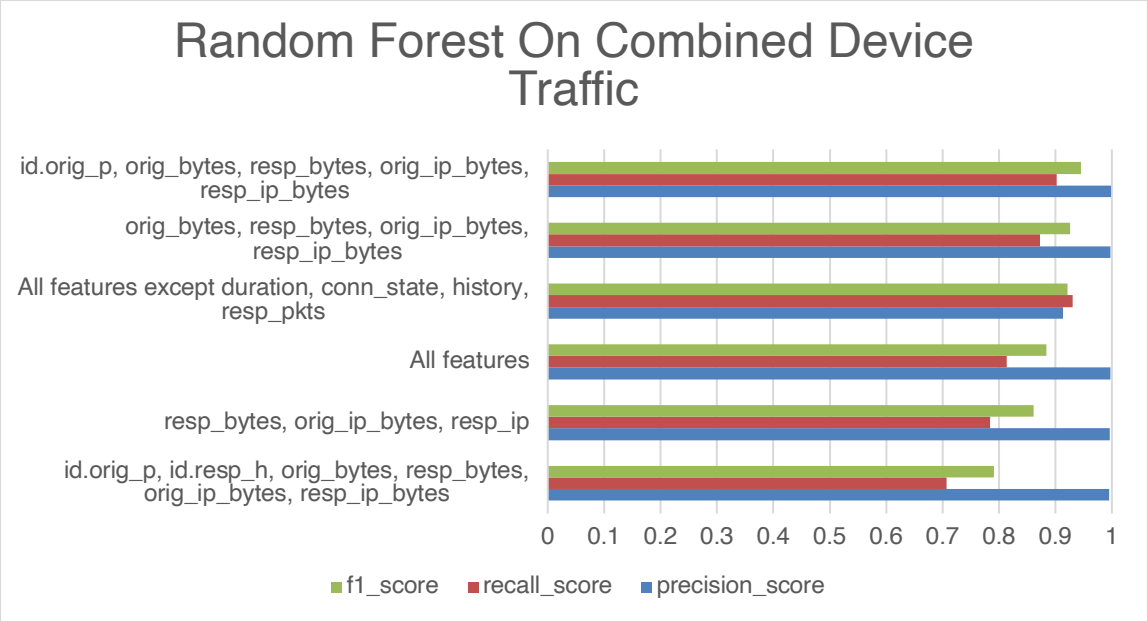

Figure 10. The Random Forest classifier trained on all device network traffic

Naïve Bayes

The same exact training as discussed above was repeated, but this time with the Naïve Bayes classifier (NB), which uses conditional probability to assign the most likely class to an observation (James et al., 2013). To start the training, all of the features were used. This resulted in a very high F_1 score of .942. To see if it was possible to improve the scores and to reduce the number of features, several iterations of removing different features was performed (see Table 19 and Figure 11 below). The end result was that the F_1 score of .942 could not be beat. However, it was possible to achieve the same scores by removing three of the features leaving the following ten features: id.orig_p, id.resp_h, id.resp_p, proto, orig_bytes, resp_bytes, conn_state, history, orig_pkts, ip_bytes.

Table 19

Features used for Naïve Bayes training and their resulting scores

Set	Features	Precision	Recall	F_1
1	All	0.8984375	0.99792199	0.94243709
2	All except duration, conn_state, history, resp_pkts	0.84459459	0.99632353	0.90615498
3	All except duration, conn_state, history	0.84459459	0.99632353	0.90615498
4	All except resp_pkts	0.8984375	0.99792199	0.94243709
5	All except duration, resp_pkts	0.8984375	0.99792199	0.94243709
6	All except duration, resp_pkts, resp_ip_bytes	0.8984375	0.99792199	0.94243709
7	All except duration, resp_conn_state, history, resp_pkts, resp_ip_bytes	0.84459459	0.99632353	0.90615498
8	id.orig_p, orig_bytes, resp_bytes, orig_ip_bytes, resp_ip_bytes	0.50890693	0.60411338	0.2182256

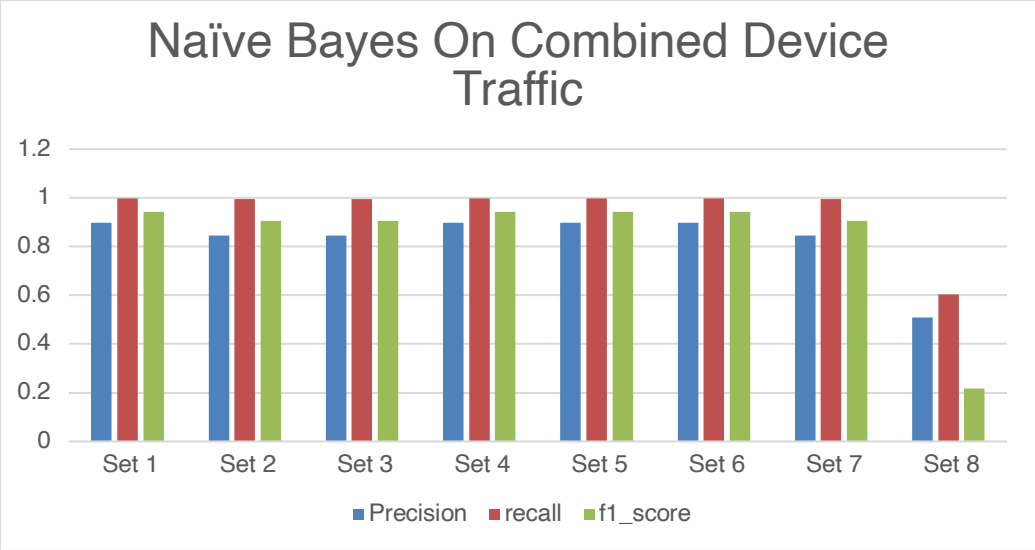

Figure 11. The Naïve Bayes classifier trained on all device network traffic

K-Nearest Neighbors

The final classifier to train with was k-nearest neighbors (KNN). KNN uses the majority class of the K nearest observations to classify new observations (James et al., 2013). To optimize performance, we set the leaf size to 30 and the number of neighbors

to five. A small leaf size slows query times, while a larger leaf size turns the algorithm into a brute force attempt. For this research several leaf sizes and neighbor combinations were tested, with 30 providing the optimal result. Once again, to begin the training, all features were used in the exact same manner as outlined in the Random Forest and Naïve Bayes training sections. The results of all the features was a promising F_1 score of .903. In an attempt to improve this score and reduce the number of features used, features were added and removed over several training iterations (see Table 20 and Figure 12 below). The winning combination was the use of nine features: id.orig_p, id.resp_h, id.resp_p, proto, orig_bytes, resp_bytes, orig_pkts, orig_ip_bytes, resp_ip_bytes (all except duration, conn_state, history, resp_pkts). This produced a much improved F_1 score of .944.

Table 20

Features used for K-Nearest Neighbors training and their resulting scores

Set	Features	Precision	Recall	F_1
1	All except duration, conn_state, history, resp_pkts	0.95721548	0.93073316	0.94356571
2	All except resp_pkts, duration	0.91900245	0.95934569	0.9382125
3	All except duration, history, resp_pkts	0.91900245	0.95934569	0.9382125
4	All except resp_pkts, duration, resp_ip_bytes	0.89887964	0.92961424	0.9136548
5	All except resp_pkts	0.9167699	0.90068201	0.90856184
6	All	0.91491004	0.89087809	0.90252259
7	orig_bytes, resp_bytes, orig_ip_bytes, resp_ip_bytes	0.90217997	0.8711104	0.88599791
8	id.orig_p, orig_bytes	0.87112676	0.89972293	0.88487307
9	id.orig_p, orig_bytes, resp_bytes, orig_ip_bytes, resp_ip_bytes	0.87112676	0.89972293	0.88487307
10	id.orig_p, orig_ip_byes, resp_ip_bytes	0.89375732	0.87095055	0.88200103

11	id.orig_p, orig_pkts, resp_pkts, resp_ip_bytes	0.82861226	0.81968728	0.81961994

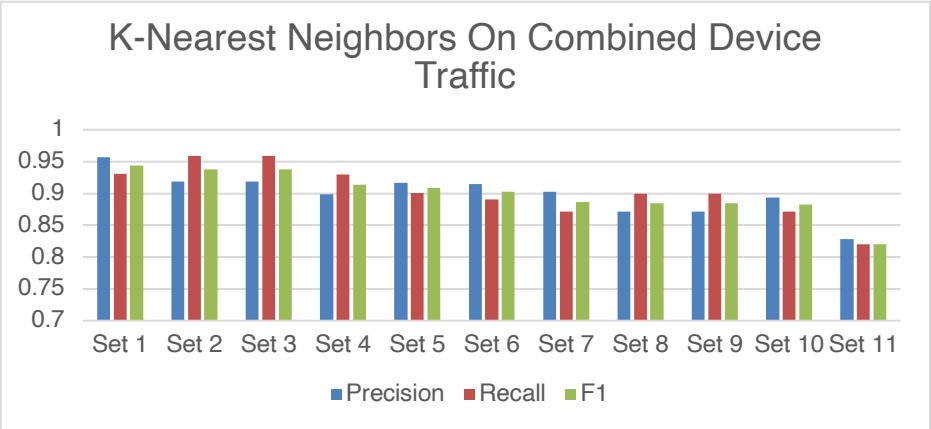

Figure 12. The Naïve Bayes classifier trained on all device network traffic

Best Classifier and Features

Overall, all three classifiers, K-Nearest Neighbors, Naïve Bayes, and Random Forest performed almost equally well with F_1 scores of .944, .942, and .945 respectively (see Table 21, Figure 13, and Figure 14 below). The main difference was the precision versus recall scores, plus the number of classifiers required to train the classifiers. The most balanced scores were achieved by K-Nearest Neighbors with a precision score of .957 and a recall score of .931. If it is important to have very few false positives then the Random Forest classifier, with a precision of .998, would be the best choice. However, this would result in failing to identify about 10% of the state changes since the recall for Random Forest is .902. Alternatively, if the goal is to identify almost all of the state changes, then Naïve Bayes is the best classifier to choose since it had a recall score of .998. This again comes with a downside: Naïve Bayes had a precision score of .898, meaning that about 10% of its predictions would be false positives.

Table 21

Scores comparisons of all three classifiers

Classifier	Precision	Recall	F_1
KNN	0.95721548	0.93073316	0.94356571
Naïve Bayes	0.8984375	0.99792199	0.94243709
Random Forest	0.99840663	0.90196078	0.94485422

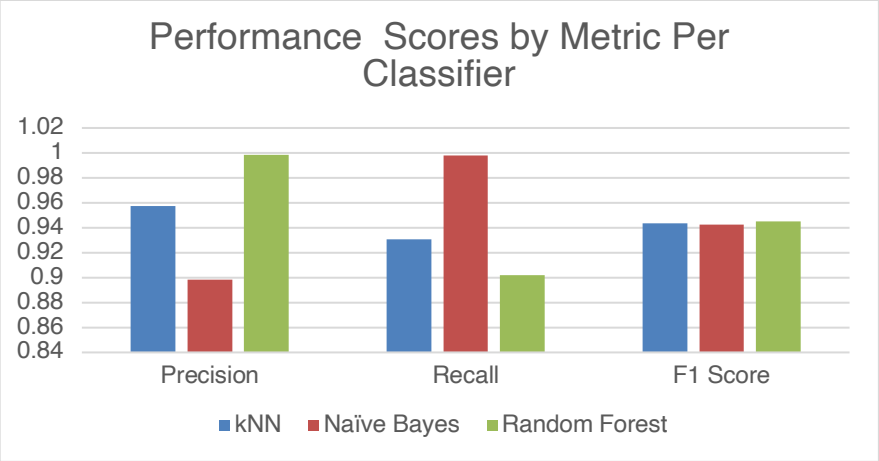

Figure 13. Precision, recall, and F_1 scores for each of the three classifiers

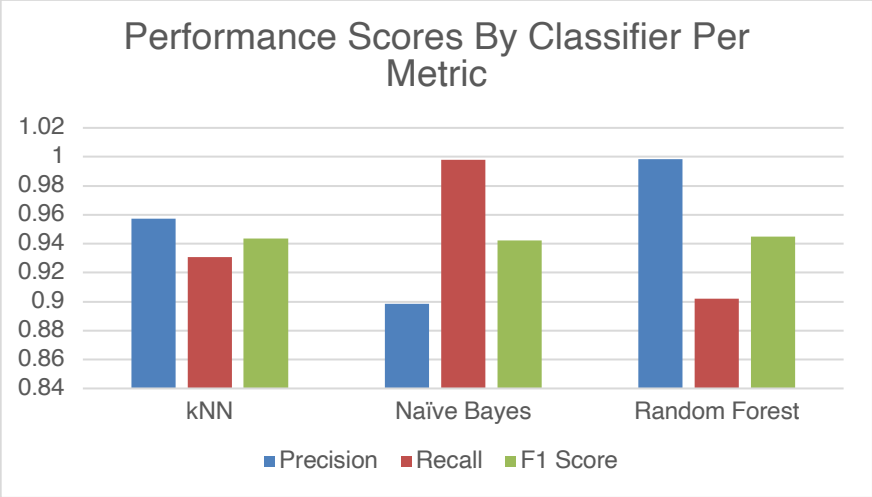

Figure 14. Each classifier and their correspond precision, recall, and F_1 scores

Random Forest was by far the best classifier in terms of requiring the fewest number of features to optimally train the classifier, requiring just five features (see Table

22 below). Both K-Nearest Neighbors and Naïve Bayes required almost twice as many, needing nine and ten features respectively. Requiring fewer features lowers the overhead in future training and can simplify the implementation of the classifier.

Table 22

Number of features by category required per classifier

Classifier	Synthesized	Protocol specific	Aggregated	Total # of Features
Random Forest	0	3	2	5
K-Nearest Neighbor	0	6	3	9
Naïve Bayes	2	6	2	10

All three categories of features, synthesized, protocol specific, and aggregated, were used to optimally train the classifiers (see Table 22, above). However, synthesized features were only used by Naïve Bayes. The most heavily relied upon category of features for training was protocol specific, accounting for 15 out of the total of 24 features used.

Four features were required to optimally train all three of the classifiers: id.orig_p, orig_bytes, resp_bytes, and orig_ip_bytes (see Table 23, Figure 15, and Figure 16 below). All of these features, except for id.orig_p, deal with the number of bytes sent in the connection. This indicates that the number of bytes sent is vital information for all three classifiers when identifying state changes.

Table 23

Features by category required per classifier

Feature Category	Random Forest	K-Nearest Neighbor	Naïve Bayes
Protocol specific	id.orig_p	id.orig_p	id.orig_p
Protocol specific		id.resp_h	id.resp_h
Protocol specific		id.resp_p	id.resp_p
Protocol specific		proto	proto

Protocol specific	orig_bytes	orig_bytes	orig_bytes
Protocol specific	resp_bytes	resp_bytes	resp_bytes
Synthesized			conn_state
Synthesized			history
Aggregated		orig_pkts	orig_pkts
Aggregated	orig_ip_bytes	orig_ip_bytes	orig_ip_bytes
Aggregated	resp_ip_bytes	resp_ip_bytes	

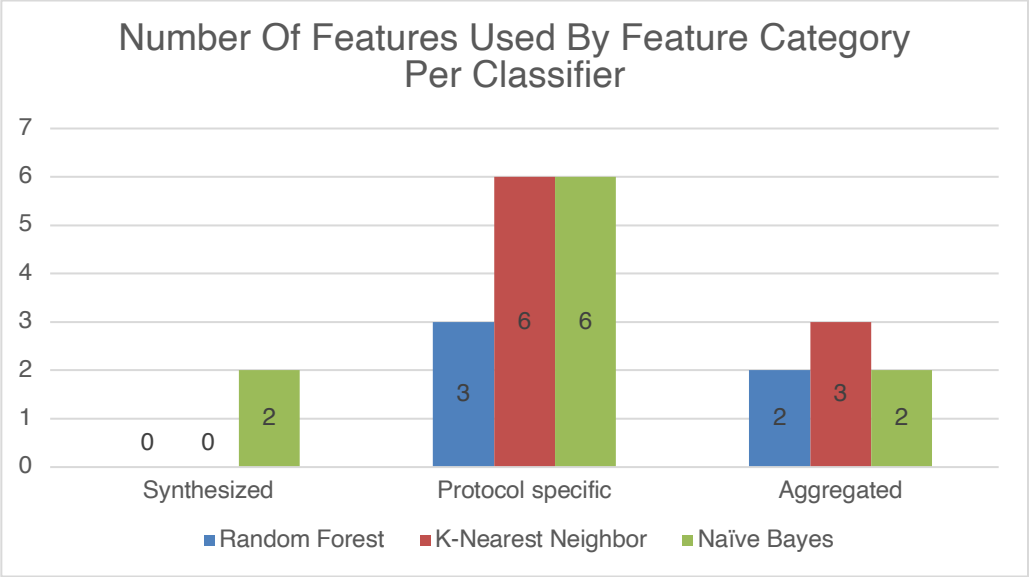

Figure 15. The number of features used in each category for each of the optimally trained classifiers

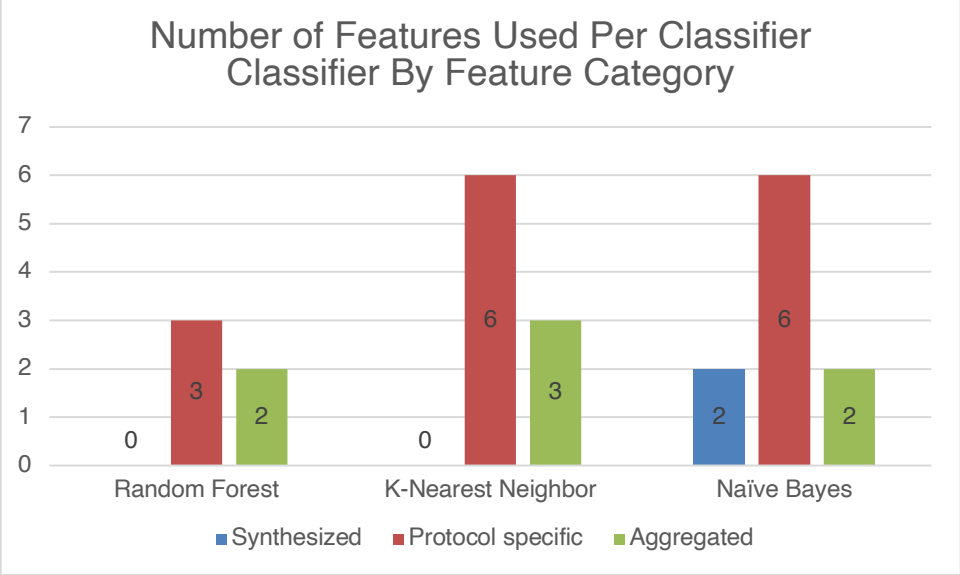

Figure 16. The number of features used by each of the optimally trained classifiers per feature category

One important aspect to note is overhead issues. When training all three of the different machine learning algorithms, all three completed their runs within just a few seconds. Given that the dataset only contained hundreds of lines of data, as opposed to thousands of lines, a quick run was not surprising. Therefore, it was not possible to judge which algorithm could require more overhead due to a longer runtime.

Summary

It is possible to identify state changes in encrypted traffic sent from an iPhone to a smart device's Cloud site. Training a classifier to spot these change requests is highly successful when done per device. In other words, near perfect identification of turning a light switch from on to off from a mobile app can be achieved by identifying patterns in the encrypted traffic.

It is also possible to train a classifier to identify all state changes hidden in encrypted network traffic across several types of smart home devices. Three different classifiers were tested: Random Forest, Naïve Bayes, and K-Nearest Neighbors. All three had similar success varying only in their recall and precision scores. In this research experiment, there was success in identifying state changes sent from three different categories of smart home devices: those that have simple state changes, those that have more complex state changes, and those that send a lot of data when a state change is made. The simple devices included the TP-Link WiFi plug and the Chamberlain myQ

Garage Opener. The more complex device was the Rachio Smart Sprinkler system. The device that sent a lot of traffic was Amazon's Ring camera.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions

Smart home device account takeover is trivial due to two of the top vulnerabilities of smart home devices: weak password policies and a lack of account lockout by the device's Cloud server (Alharbi & Aspinall, 2018). Once an attacker has commandeered access to a smart home device, they could harass the resident, lock the user out of their device, scare the user out of their house (to gain access to rob it), and several other malicious attacks (Freed et al., 2018a; Ronen & Shamir, 2016a). Currently, there is not an easy way for home users to detect that a malicious actor is making unwanted changes to their smart home devices (Geeng & Roesner, 2019a; Matthews et al., 2017; Zeng et al., 2017). Previous research has leveraged patterns discovered in encrypted network traffic to identify a smart home device and if it has received a state change request (Acar et al., 2018; Apthorpe, Reisman, & Feamster, 2017a; Copos et al., 2016a; Marchal et al., 2019; Meidan et al., 2017; Miettinen et al., 2017a; V. Sivaraman et al., 2015). The research detailed in this paper successfully extended previous research by identifying state changes sent via network traffic between a smart device's app running on a smartphone and the smart device's Cloud site.

This research also succeeded in identifying state change requests sent from a smartphone to smart home devices' Cloud sites, across a variety of different types of

popular smart home devices: 1) devices with simple behaviors, 2) devices with complex behaviors, and 3) devices that send a lot of data. This included four smart home devices: the TP-Link smart plug and the myQ garage door opener representing device's with simple behaviors, the Rachio smart sprinkler system, which is a device capable of more complex behaviors, and Amazon's Ring video camera which sends a large amount of data. The network traffic patterns indicative of a state change were identified for each of the smart home devices and then used to label the captured network traffic. The Random Forest algorithm was successfully trained using this labeled network traffic data for all four of the devices individually.

The main goal of this research was also achieved. This entailed identifying the best combination of network traffic features and machine learning algorithms capable of identifying state changes in encrypted traffic across all four of the devices. The most balanced results was achieved with the k-Nearest Neighbors algorithm, the Random Forest algorithm had the highest precision score, and Naïve Bayes had the best recall score. The most efficient algorithm was determined to be Random Forest and the most important features were id.orig_p, orig_bytes, resp_bytes, and orig_ip_bytes since they were used to optimally train each of the three algorithms.

Implications

The purpose of this research was to identify rogue changes made to a smart home device by an actor who is outside the home network. This addresses the situation when a user is at home and does not want state changes to be made to their same home devices by an individual outside the home. Identifying these rogue changes is possible due to the commonly designed infrastructure of smart home devices; most smart home devices are

directly controlled by the smart home device's Cloud server (Apthorpe, Reisman, & Feamster, 2017a). Therefore, the Cloud server is usually the middleman between the smart device's app running on a smartphone and the smart home device. When a user requests a change on their smartphone via the smart device's app, the change request is sent to the smart device's Cloud server, the change is noted on the Cloud server, and then sent to the smart home device.

When a user initiates a state change request at home, the change request originates from inside the home across the home WiFi network, travels outside the home to the Cloud server, then back in again. This is in contrast to when someone outside the home makes a change; the change is applied to the Cloud server, sent down to the home WiFi network, and applied to the smart device. What is missing when the change request originates from outside the home is the traversal of the state change request across the home WiFi network. This missing piece can be used to establish if the change originated from inside the home or from outside the home; if the state change is seen in the home WiFi traffic, then it originated from inside the home, if it is not seen, then the change originated from outside the home.

This research, using machine learning algorithms, successfully identified state change requests sent from a smartphone, across the home WiFi network, to the smart device's Cloud site. This was accomplished for three main types of smart home devices: those are capable of simple behaviors, those with more complex behaviors, and those that send a lot of data.

A limitation of this research was that the Zeek was the main tool used to analyze network traffic. There are several other means to explore network traffic which should be

explored in future research. Additionally, most of the identification tasks included in this research are binary. The implication is that if multiple categories are added, then the performance will more than likely degrade.

Recommendations

This research extends previous work which explored the use of machine learning to identify state changes applied to smart home devices. This research can also be extended. Network traffic generated from other smart home device apps can be added to the traffic generated and labeled from this research. This would create a list of devices that could or could not have their state changes identified in encrypted traffic by machine learning algorithms. Further testing could be done by adding more smart home devices to each category of smart home devices and then testing the effectiveness of machine learning within each category or combinations between categories.

Additionally, smart hubs and smart assistants could also be studied. As stated previously, both devices act similarly to smart phones in that they send state changes via encrypted traffic over a home WiFi network. Therefore, it is possible that using a similar approach taken in this research, state changes sent by smart hubs and smart assistants could be identified.

This research project used 13 different network traffic features, in various combinations to train machine learning algorithms. This omitted eight features that were identified by this research as potential candidates. These could be tested along with others that this research did not identify. It is possible that the features that were not tested prove easier to derive or may even improve the training.

This research did not cover an exhaustive combination of features tested with machine learning algorithms. Many more could be tested. Additionally, new features could be combined with new machine learning algorithms that were not identified in this research. Combinations of these could also be included in future research.

Once again, the purpose of this research was to identify and possibly prevent unwanted external state changes. Therefore, future research could build such a system that would be capable of doing just that: either alerting the home user of a rogue change or blocking the change from being applied to the smart home device. This would involve tying together this research with previous research that demonstrated the feasibility of identifying state changes sent from the Cloud site to the smart home device. The missing piece of the proposed system would need to be capable of linking state changes sent out to the Cloud site via a smartphone on the home WiFi network to those that were coming in from the Cloud site to the smart home device. Preventing the state change from taking place has previously been studied in the form of intrusion prevention systems that are capable of dropping unwanted packets.

Summary

Smart home devices are becoming more popular with consumers with the number of devices in peoples' homes increasing (He et al., 2018). Malicious users have taken notice of this trend and have begun to manipulate these devices through known vulnerabilities such as poor authentication practices (Freed et al., 2018a). One way to monitor these malicious changes is by monitoring home network traffic.

Change requests to smart home devices travel across the network in the form of encrypted network packets. Since the traffic is typically encrypted, it is not possible to

simply read the contents of the packet to learn if the packet contains instructions for the smart device to change states (Apthorpe, Reisman, Sundaresan, et al., 2017; Copos et al., 2016a). However, there are attributes of the packet that are not that are not encrypted, which, along with the size of the payload can be used to establish patterns indicative of a smart device state change request, versus an update, versus a status check (Meidan et al., 2017). This research has extended previous research by identifying the best combination of features and machine learning algorithms combination capable of identifying state change requests across a variety of different types of popular smart home devices. This was accomplished in three major phases: phase 1) focused on analyzing the publicly available network data captures, phase 2) involved capturing the network traffic and identifying the traffic patterns indicative of a state change request of several smart home devices, and phase 3) included identifying network traffic features, using those features to train machine learning algorithms, and identify the machine learning algorithm that performed the best across all of the selected smart home devices.

Phase one involved the datasets provided by Alrawi et al. (2019) and Ren et al. (2019). Alrawi et al. (2019) collected network traffic over a period of 13 days resulting in over 150 GB of data. The analysis of this dataset found that there was not one instance when both the smart device's app and the smart device were sending or receiving network traffic within the same timeframe (i.e. within five minutes of each other). The second dataset from Ren et al. (2019), included network traffic captured from 81 different smart home devices located in labs located in the United States and the United Kingdom. In-depth analysis of both datasets was performed. Unfortunately, it was discovered that the network traffic did not contain any network traffic from a smartphone or tablet.

Phase two involved capturing network traffic in a lab environment and then identifying the state changes of four smart home devices from three different categories of devices: 1) simple behavior, where the TP-Link WiFi Plug and the Chamberlain myQ Garage Opener were selected, 2) complex behavior, which included the Rachio Smart Sprinkler, and 3) sending large amounts of data, in which the Amazon's Ring video camera was selected. The TP-Link's state change was identified by a new connection that would be created between the iPhone and the TP-Link WiFi Plug directly whenever the switch was turned on or off. Additionally, identical values for three network attributes were seen every time the plug's state was changed. For the myQ device, a pattern emerged in which the app on the iPhone would communicate with the account-devices-gdo.myq-cloud.com Cloud site whenever the door was either opened or closed. Next, the Ring device's state change was identified by SIP traffic that was seen every time the camera was put into Live mode. Finally, the Rachio "quick run" state change was identified by the connection to the Cloud site rach.io.

The training of the machine learning algorithms involved selected features from three categories: 1) aggregated, 2) synthesized, and 3) protocol specific. This included 13 features across the three categories. The first step involved using the Random Forest algorithm to identify state changes of each of the devices. The TP-Link plug was successfully trained with the just three features: resp_bytes, original_bytes, and orig_ip_bytes. Perfect scores for cross validation across all folds was achieved, with perfect scores for the root mean square score across all folds, and for the precision, recall, and F_1 scores. Only three features was necessary to optimally train the myQ device: id.resp_h,orig_ip_bytes, and duration. The precision, recall, and F_1 scores were all 100%.

Again, only three features were needed to successfully train the Ring device: id.resp_p, orig_ip_bytes, and resp_ip_bytes. The precision, recall, and F_1 scores were all 100%. Finally, the Rachio device also required three features to optimally train it: orig_pkts, resp_ip_bytes, and orig_ip_bytes. The precision, recall, and F_1 scores were, once again, all 100%.

The culminating task was to combine all of the network traffic from all four of the devices and then train three different machine learning algorithms with different combinations of features. The three different algorithms were Random Forest, Naïve Bayes, and K-Nearest Neighbors. Starting with the Random Forest algorithm, the best results were obtained with five features: resp_bytes, orig_ip_bytes, resp_ip_bytes, orig_bytes, and id.orig_p. This produced a precision score of 99.9002%, recall of 93.902 %, and an F_1 score of 96.7033%. The Naïve Bayes algorithm performed similarity, with a precision score of 99.9002%, recall of 93.902 % and an F_1 score of 96.7033%. Finally the K-Nearest Neighbors algorithm produced the best results when trained with ten features: id.orig_p, id.resp_h, id.resp_p, proto, orig_bytes, resp_bytes, orig_pkts, ip_bytes, resp_ip_bytes (all except duration, conn_state, history, resp_pkts). This produced an impressive F_1 score of .9435.

Overall, all three classifiers, K-Nearest Neighbors, Naïve Bayes, and Random Forest performed almost equally well with F_1 scores of .94356, .9423, and .9449 respectively. They varied in their precision and recall scores, plus the number of classifiers required for training. K-Nearest Neighbors achieved the most balanced results with a precision score of .9572 and a recall score of .9307. The Random Forest algorithm had the highest precision score of .9984 and Naïve Bayes had the best recall score of

.9979. In terms of number of features required to train an algorithm, Random Forest was by far the best requiring just five features. Finally, it was evident that the most important features, which were used for all three classifiers, came down to just four features: id.orig_p, orig_bytes, resp_bytes, and orig_ip_bytes.

In summary, our research goal – the identification of state changes by using encrypted IoT network traffic – was achieved empirically. The success of identifying state change requests sent from a mobile app, combined with previous research that identified state changes sent to the smart home device, allows for the development of a system that could block unwanted state changes that originate from a malicious user located outside of the house. Therefore, this research adds to the body of knowledge to IoT security and could be extended to the identification of other networking patterns based on encrypted traffic.

Appendix A

Zeek Information

Zeek's conn.log fields

Field	Description
ts	This is the time of the first packet.
uid	A unique identifier of the connection.
id.orig_h	The originator's 4-tuple of endpoint address.
id.orig_p	The originator's 4-tuple of endpoint port.
id.resp_h	The responder's 4-tuple of endpoint address.
id.resp_p	The responder's 4-tuple of endpoint port.
proto	The transport layer protocol of the connection.
service	An identification of an application protocol being sent over the connection.
duration	How long the connection lasted. For 3-way or 4-way connection tear-downs, this will not include the final ACK.
orig_bytes	The number of payload bytes the originator sent.
resp_bytes	The number of payload bytes the responder sent. See orig_bytes.
conn_state	There are several possible conn_state values (see table below).
local_orig	If the connection is originated locally, this value will be T. If it was originated remotely it will be F.
local_resp	If the connection is responded to locally, this value will be T. If it was responded to remotely it will be F.
missed_bytes	Indicates the number of bytes missed in content gaps, which is representative of packet loss.
history	Records the state history of connections as a string of letters (see table below).
orig_pkts	Number of packets that the originator sent
orig_ip_bytes	Number of IP level bytes that the originator sent (as seen on the wire, taken from the IP total_length header field).
resp_pkts	Number of packets that the responder sent.
resp_ip_bytes	Number of IP level bytes that the responder sent (as seen on the wire, taken from the IP total_length header field).
conn_state value	conn_state value descriptions

Zeek's possible conn_state values

Value	Description
S0	Connection attempt seen, no reply.

S1	Connection established, not terminated.
SF	Normal establishment and termination.
REJ	Connection attempt rejected.
S2	Connection established and close attempt by originator seen (but no reply from responder).
S3	Connection established and close attempt by responder seen (but no reply from originator).
RSTO	Connection established, originator aborted (sent a RST).
RSTR	Responder sent a RST.
RSTOS0	Originator sent a SYN followed by a RST, we never saw a SYN-ACK from the responder.
RSTRH	Responder sent a SYN ACK followed by a RST, we never saw a SYN from the (purported) originator.
SH	Originator sent a SYN followed by a FIN, we never saw a SYN ACK from the responder (hence the connection was "half" open).
SHR	Responder sent a SYN ACK followed by a FIN, we never saw a SYN from the originator.
OTH	No SYN seen, just midstream traffic (a "partial connection" that was not later closed).

Zeek's conn.log history field

Letter	Meaning
s	a SYN w/o the ACK bit set
h	a SYN+ACK ("handshake")
a	a pure ACK
d	packet with payload ("data")
f	packet with FIN bit set
r	packet with RST bit set
c	packet with a bad checksum (applies to UDP too)
g	a content gap
t	packet with retransmitted payload
w	packet with a zero window advertisement
i	inconsistent packet (e.g. FIN+RST bits set)
q	multi-flag packet (SYN+FIN or SYN+RST bits set)

Letter	Meaning
^	connection direction was flipped by Zeek's heuristic

Note: If the event comes from the originator, the letter is in upper-case

Connections involving state changes sent from the iPhone to the myQ Cloud site

id.orig_h	**id.orig_p**	**id.resp_h**	**id.resp_p**	**proto**	**service**	**duration**	**orig_bytes**	**resp_bytes**	**conn_state**	**history**	**orig_pkts**	**orig_ip_bytes**	**resp_pkts**	**resp_ip_bytes**
192.168.248	55417	13.32.0.23	443	tcp	ssl	0.347665	1121	5746	S1	ShADTdt	24	3490	24	12732
192.168.248	55434	13.32.0.23	443	tcp	ssl	0.241997	1248	5746	S1	ShADTdt	24	3744	24	12732
192.168.248	55446	13.32.0.23	443	tcp	ssl	0.400331	1248	5746	S1	ShADTdt	22	3640	24	12732
192.168.248	55423	13.32.0.23	443	tcp	ssl	1.086221	1174	434	S1	ShADTdt	20	4422	20	1900
192.168.248	55435	13.32.0.23	443	tcp	ssl	0.476044	1249	5746	S1	ShADTdt	24	3746	24	12732
192.168.248	55447	13.32.0.23	443	tcp	ssl	0.359015	1249	5746	S1	ShADTdt	24	3746	24	12732

References

Acar, A., Fereidooni, H., Abera, T., Sikder, A. K., Miettinen, M., Aksu, H., Conti, M., Sadeghi, A.-R., & Uluagac, A. S. (2018). Peek-a-Boo: I see your smart home activities, even encrypted! *ArXiv Preprint*, *arXiv:1808.02741*. https://doi.org/10.1145/3395351.3399421

Alharbi, R., & Aspinall, D. (2018). An IoT analysis framework: An investigation of IoT smart cameras' vulnerabilities. *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, 47–57. https://doi.org/10.1049/cp.2018.0047

Ali, W., Dustgeer, G., Awais, M., & Shah, M. A. (2017). IoT based smart home: Security challenges, security requirements and solutions. *2017 23rd International Conference on Automation and Computing (ICAC)*, 1–6. https://doi.org/10.23919/IConAC.2017.8082057

Alrawi, O., Lever, C., Antonakakis, M., & Monrose, F. (2019b). Sok: Security evaluation of home-based IoT deployments. *2019 IEEE Symposium on Security and Privacy (SP)*, 1362–1380. https://doi.org/10.1109/SP.2019.00013

Anthi, E., Williams, L., Słowińska, M., Theodorakopoulos, G., & Burnap, P. (2019). A supervised intrusion detection system for smart home IoT devices. *IEEE Internet of Things Journal*, *6*(5), 9042–9053. https://doi.org/10.1109/JIOT.2019.2926365

Apthorpe, N., Reisman, D., & Feamster, N. (2017a). Closing the blinds: Four strategies for protecting smart home privacy from network observers. *ArXiv Preprint ArXiv:1705.06809*.

Apthorpe, N., Reisman, D., & Feamster, N. (2017b). A smart home is no castle: Privacy vulnerabilities of encrypted IoT traffic. *ArXiv:1705.06805 [Cs]*. http://arxiv.org/abs/1705.06805

Apthorpe, N., Reisman, D., Sundaresan, S., Narayanan, A., & Feamster, N. (2017). Spying on the smart home: Privacy attacks and defenses on encrypted IoT traffic. *ArXiv Preprint ArXiv:1708.05044*. https://arxiv.org/abs/1708.05044

Ashton, K. (1999). That 'internet of things' thing. *RFID Journal*, *22*(7), 97–114.

Atamli, A. W., & Martin, A. (2014). Threat-based security analysis for the internet of things. *Secure Internet of Things (SIoT), 2014 International Workshop On*, 35–43. http://ieeexplore.ieee.org/abstract/document/7058906/

Barcena, M. B., & Wueest, C. (2015). Insecurity in the internet of things. *Security Response, Symantec*. https://pdfs.semanticscholar.org/6d7f/60b16adead96aafa9e975207980eb32671b5.pdf

Barrera, D., Molloy, I., & Huang, H. (2017a). IDIoT: Securing the internet of things like it's 1994. *ArXiv Preprint ArXiv:1712.03623*.

Bastos, D., Shackleton, M., & El-Moussa, F. (2018). Internet of things: A survey of technologies and security risks in smart home and city environments. *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, 1–7.

Bezawada, B., Bachani, M., Peterson, J., Shirazi, H., Ray, I., & Ray, I. (2018). IoTSense: Behavioral fingerprinting of IoT devices. *ArXiv Preprint ArXiv:1804.03852*.

Bowles, N. (2018, June 23). Thermostats, locks and lights: Digital tools of domestic abuse. *The New York Times*. https://www.nytimes.com/2018/06/23/technology/smart-home-devices-domestic-abuse.html

Copos, B., Levitt, K., Bishop, M., & Rowe, J. (2016a). Is anybody home? Inferring activity from smart home network traffic. *2016 IEEE Security and Privacy Workshops (SPW)*, 245–251. https://doi.org/10.1109/SPW.2016.48

Costin, A., Zaddach, J., Francillon, A., & Balzarotti, D. (2014b). A large-scale analysis of the security of embedded firmwares. *23rd USENIX Security Symposium (USENIX Security 14)*, 95–110. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin

Dai, R., Gao, C., Lang, B., Yang, L., Liu, H., & Chen, S. (2019). SSL malicious traffic detection based on multi-view features. *Proceedings of the 2019 the 9th International Conference on Communication and Network Security*, 40–46. https://doi.org/10.1145/3371676.3371697

Douglas, D. M. (2016). Doxing: A conceptual analysis. *Ethics and Information Technology*, *18*(3), 199–210. https://doi.org/10.1007/s10676-016-9406-0

Flosbach, R., Chromik, J. J., & Remke, A. (2019). Architecture and Prototype Implementation for Process-Aware Intrusion Detection in Electrical Grids. *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, 42–4209. https://doi.org/10.1109/SRDS47363.2019.00015

Fraser, C., Olsen, E., Lee, K., Southworth, C., & Tucker, S. (2010). The new age of stalking: Technological implications for stalking. *Juvenile and Family Court Journal*, *61*(4), 39–55. https://doi.org/10.1111/j.1755-6988.2010.01051.x

Freed, D., Palmer, J., Minchala, D., Levy, K., Ristenpart, T., & Dell, N. (2018b). "A stalker's paradise": How intimate partner abusers exploit technology. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–13. https://doi.org/10.1145/3173574.3174241

Fu, K., Kohno, T., Lopresti, D., Mynatt, E. D., Nahrstedt, K., Patel, S. N., Richardson, D., & Zorn, B. (2017a). Safety, security, and privacy threats posed by accelerating

trends in the internet of things. *Computing Community Consortium (CCC) Technical Report*, *29*(3).

Geeng, C., & Roesner, F. (2019b). Who's in control? Interactions in multi-user smart homes. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–13. https://doi.org/10.1145/3290605.3300498

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and Tensorflow: Concepts, tools, and techniques to build intelligent systems* (2nd ed.). O'Reilly Media.

Hammill, R., & Hendricks, M. (2013, April 24). Gadgets to help tend a garden. *The New York Times*. https://www.nytimes.com/2013/04/25/technology/personaltech/calling-on-gadgetry-to-keep-the-garden-growing.html

He, W., Golla, M., Padhi, R., Ofek, J., Dürmuth, M., Fernandes, E., & Ur, B. (2018). Rethinking access control and authentication for the home internet of things (IoT). In *27th USENIX Security Symposium (Security 18)* (pp. 255–272). USENIX Association.

He, W., Martinez, J., Padhi, R., Zhang, L., & Ur, B. (2019b). When smart devices are stupid: Negative experiences using home smart devices. *2019 IEEE Security and Privacy Workshops (SPW)*, 150–155. https://doi.org/10.1109/SPW.2019.00036

Hodo, E., Bellekens, X., Hamilton, A., Dubouilh, P. L., Iorkyase, E., Tachtatzis, C., & Atkinson, R. (2016). Threat analysis of IoT networks using artificial neural network intrusion detection system. *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, 1–6. https://doi.org/10.1109/ISNCC.2016.7746067

Intellectsoft. (2015, September 1). 3 Types of software architecture for internet of things devices. *Intellectsoft*. https://www.intellectsoft.net/blog/3-types-of-software-architecture-for-connected-devices/

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). Tree-Based Methods. In G. James, D. Witten, T. Hastie, & R. Tibshirani (Eds.), *An Introduction to Statistical Learning: With Applications in R* (Vol. 103, p. 319). Springer. https://doi.org/10.1007/978-1-4614-7138-7_8

Jia, Y. J., Chen, Q. A., Wang, S., Rahmati, A., Fernandes, E., Mao, Z. M., Prakash, A., & Unviersity, S. J. (2017, February). ContexIoT: Towards providing contextual integrity to appified IoT platforms. *NDSS*.

Junges, P.-M., François, J., & Festor, O. (2019). Passive inference of user actions through IoT gateway encrypted traffic analysis. *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 7–12.

Jupyter Notebook 6.0.3. (2020). https://jupyter-notebook.readthedocs.io/en/stable/

Kasinathan, P., Pastrone, C., Spirito, M. A., & Vinkovits, M. (2013). Denial-of-service detection in 6LoWPAN based internet of things. *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 600–607. https://doi.org/10.1109/WiMOB.2013.6673419

Kim, T. H., Bauer, L., Newsome, J., Perrig, A., & Walker, J. (2011). Access right assignment mechanisms for secure home networks. *Journal of Communications and Networks*, *13*(2), 175–186. https://doi.org/10.1109/JCN.2011.6157417

Kolias, C., Kambourakis, G., Stavrou, A., & Voas, J. (2017). DDoS in the IoT: Mirai and other botnets. *Computer*, *50*(7), 80–84. https://doi.org/10.1109/MC.2017.201

Marchal, S., Miettinen, M., Nguyen, T. D., Sadeghi, A.-R., & Asokan, N. (2019). AuDI: Toward autonomous IoT device-type identification using periodic communication. *IEEE Journal on Selected Areas in Communications*, *37*(6), 1402–1412. https://doi.org/10.1109/JSAC.2019.2904364

Matthews, T., O'Leary, K., Turner, A., Sleeper, M., Woelfer, J. P., Shelton, M., Manthorne, C., Churchill, E. F., & Consolvo, S. (2017). Stories from survivors: Privacy & security practices when coping with intimate partner abuse. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (pp. 2189–2201). ACM.

Meidan, Y., Bohadana, M., Shabtai, A., Ochoa, M., Tippenhauer, N. O., Guarnizo, J. D., & Elovici, Y. (2017). Detection of unauthorized IoT devices using machine learning techniques. *ArXiv Preprint ArXiv:1709.04647*.

Microsoft Excel (16.16.19). (2019). [Computer software]. Microsoft. https://products.office.com/en-us/excel

Miettinen, M., Marchal, S., Hafeez, I., Asokan, N., Sadeghi, A., & Tarkoma, S. (2017b). IoT SENTINEL: Automated sevice-type identification for security enforcement in IoT. *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2177–2184. https://doi.org/10.1109/ICDCS.2017.283

Naughton, J. (2018, July 1). The internet of things has opened up a new frontier of domestic abuse. *The Guardian*. https://www.theguardian.com/commentisfree/2018/jul/01/smart-home-devices-internet-of-things-domestic-abuse

Nobakht, M., Sivaraman, V., & Boreli, R. (2016). A host-based intrusion detection and mitigation framework for smart home IoT using OpenFlow. *2016 11th International Conference on Availability, Reliability and Security (ARES)*, 147–156. https://doi.org/10.1109/ARES.2016.64

Notra, S., Siddiqi, M., Gharakheili, H. H., Sivaraman, V., & Boreli, R. (2014). An experimental study of security and privacy risks with emerging household

appliances. In *2014 IEEE Conference on Communications and Network Security* (pp. 79–84). IEEE.

NumPy—NumPy. (2020). https://numpy.org/

OConnor, T., Mohamed, R., Miettinen, M., Enck, W., Reaves, B., & Sadeghi, A.-R. (2019). HomeSnitch: Behavior transparency and control for smart home IoT devices. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks* (pp. 128–138). Association for Computing Machinery.

OpenWrt Project (18.06.8). (2020). [Linux]. https://openwrt.org/

Pandas 1.0.3. (2020). https://pandas.pydata.org/docs/whatsnew/v1.0.0.html

Paxson, V. (1999). Bro: A system for detecting network intruders in real-time. *Computer Networks*, *31*(23), 2435–2463. https://doi.org/10.1016/S1389-1286(99)00112-7

Pongle, P., & Chavan, G. (2015). Real time intrusion and wormhole attack detection in internet of things. *International Journal of Computer Applications*, *121*(9), 1–9.

Ramapatruni, S., Narayanan, S. N., Mittal, S., Joshi, A., & Joshi, K. P. (2019). Anomaly detection models for smart home security. In *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)* (pp. 19–24). IEEE.

Ravidas, S., Lekidis, A., Paci, F., & Zannone, N. (2019). Access control in internet-of-things: A survey. *Journal of Network and Computer Applications*, *144*, 79–101.

Ren, J., Dubois, D. J., Choffnes, D., Mandalari, A. M., Kolcun, R., & Haddadi, H. (2019b). Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach. *Proceedings of the Internet Measurement Conference*, 267–279. https://doi.org/10.1145/3355369.3355577

RND_Musings. (2015, May 24). r/ringdoorbell—Well, I captured some traffic from my Ring Pro. *Reddit*. https://www.reddit.com/r/ringdoorbell/comments/7as4a9/well_i_captured_some_traffic_from_my_ring_pro/

Ronen, E., & Shamir, A. (2016b). Extended functionality attacks on IoT devices: The case of smart lights. *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, 3–12. https://doi.org/10.1109/EuroSP.2016.13

Schuster, R., Shmatikov, V., & Tromer, E. (2018b). Situational access control in the internet of things. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 1056–1073. https://doi.org/10.1145/3243734.3243817

Scikit-learn 0.22.2. (2020). https://scikit-learn.org/stable/

Sears, A. (2019, September 23). 'Felt so violated:' Milwaukee couple warns hackers are outsmarting smart homes. *FOX6Now.Com*. https://fox6now.com/2019/09/22/felt-so-violated-milwaukee-couple-warns-hackers-are-outsmarting-smart-homes/

Shackelford, S., Raymond, A., Balakrishnan, R., Dixit, P., Gjonaj, J., & Kavi, R. (2017). When toasters attack: A polycentric approach to enhancing the 'security of things.' *University of Illinois Law Review*, *16*(6), 415–469.

Sikder, A. K., Petracca, G., Aksu, H., Jaeger, T., & Uluagac, A. S. (2018). A survey on sensor-based threats to internet-of-things (IoT) devices and applications. *ArXiv Preprint ArXiv:1802.02041*.

Sivanathan, A., Sherratt, D., Gharakheili, H. H., Radford, A., Wijenayake, C., Vishwanath, A., & Sivaraman, V. (2017). Characterizing and classifying IoT traffic in smart cities and campuses. *Proc. IEEE INFOCOM Workshop SmartCity, Smart Cities Urban Comput.*, 1–6.

Sivaraman, V., Gharakheili, H. H., Vishwanath, A., Boreli, R., & Mehani, O. (2015). Network-level security and privacy control for smart-home IoT devices. *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, 163–167. https://doi.org/10.1109/WiMOB.2015.7347956

Sivaraman, Vijay, Gharakheili, H. H., Vishwanath, A., Boreli, R., & Mehani, O. (2015). 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* (pp. 163–167). IEEE.

Subahi, A., & Theodorakopoulos, G. (2019). Detecting IoT user behavior and sensitive information in encrypted IoT-app traffic. *Sensors*, *19*(21), 4777–4805. https://doi.org/10.3390/s19214777

Tcpdump (4.9.2). (2017). [Linux]. http://www.tcpdump.org

Tertytchny, G., Nicolaou, N., & Michael, M. K. (2019b). Differentiating attacks and faults in energy aware smart home system using supervised machine learning. *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, 122–127. https://doi.org/10.1145/3312614.3312642

Tian, Y., Zhang, N., Lin, Y.-H., Wang, X., Ur, B., Guo, X., & Tague, P. (2017b). SmartAuth: User-centered authorization for the internet of things. *26th USENIX Security Symposium (USENIX Security 17)*, 361–378. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tian

Ur, B., Jung, J., & Schechter, S. (2014b). Intruders versus intrusiveness: Teens' and parents' perspectives on home-entryway surveillance. *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 129–139. https://doi.org/10.1145/2632048.2632107

Vitak, J., Chadha, K., Steiner, L., & Ashktorab, Z. (2017b). Identifying women's experiences with and strategies for mitigating negative effects of online harassment. *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, 1231–1245. https://doi.org/10.1145/2998181.2998337

Walker, K. (2014, July). *The legal considerations of the internet of things*. ComputerWeekly.Com. https://www.computerweekly.com/opinion/The-legal-considerations-of-the-internet-of-things

Wang, Q., Hassan, W. U., Bates, A., & Gunter, C. (2018, February). Fear and logging in the internet of things. *Network and Distributed Systems Symposium*. Network and Distributed Systems Security (NDSS) Symposium 2018, San Diego, CA, USA. http://dx.doi.org/10.14722/ndss.2018.23282

Weber, R. H. (2016). Governance of the internet of things—From infancy to first attempts of implementation? *Laws*, *5*(3), 28. https://doi.org/10.3390/laws5030028

Wireshark (3.2.2). (2020). [MacOS]. https://www.wireshark.org

Wisniewski, P., Xu, H., Rosson, M. B., Perkins, D. F., & Carroll, J. M. (2016b). Dear diary: Teens reflect on their weekly online risk experiences. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 3919–3930. https://doi.org/10.1145/2858036.2858317

Wueest, C. (2015, March 12). *Is IoT in the smart home giving away the keys to your kingdom?* Symantec Security Response. http://www.symantec.com/connect/blogs/iot-smart-home-giving-away-keys-your-kingdom

Zarpelão, B. B., Miani, R. S., Kawakani, C. T., & de Alvarenga, S. C. (2017). A survey of intrusion detection in internet of things. *Journal of Network and Computer Applications*, *84*, 25–37. https://doi.org/10.1016/j.jnca.2017.02.009

Zeichick, D. (2018). Have my smart lightbulbs been weaponized: Introducing computer security issues related to IoT devices. *Journal of Computing Sciences in Colleges*, *33*(4), 123–129.

Zeng, E., Mare, S., & Roesner, F. (2017). End user security & privacy concerns with smart homes. In *Thirteenth Symposium on Usable Privacy and Security* (pp. 65–80). USENIX Association.

Zhou, W., Jia, Y., Yao, Y., Zhu, L., Guan, L., Mao, Y., Liu, P., & Zhang, Y. (2019). Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms. *Proceedings of the 28th USENIX Conference on Security Symposium*, 1133–1150.

Zhou, W., Jia, Y., Yao, Y., Zhu, L., Guan, L., Mao, Y., Liu, P., & Zhang, Y. (2019). Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms. *Proceedings of the 28th USENIX Conference on Security Symposium*, 1133–1150.