I, Austin J Marck, hereby submit this original work as part of the requirements for the degree of Master of Science in Information Technology-Distance Learning.

It is entitled:
Abusing Android TV Box for Fun and Profit

Student's name:  Austin J Marck

This work and its defense approved by:

Committee chair: Xue Tao Wei, Ph.D.

Committee member: Guo Freeman, Ph.D.

Committee member: Ki Jung Lee, Ph.D.
Abusing Android TV Box for Fun and Profit

A thesis submitted to the
Graduate School
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of

Master of Science

in the School of Information Technology
of the College of Education, Criminal Justice, and Human Services

by

Austin Marck

B.A. University of Cincinnati
July 2017

Committee Chair: Xuetao Wei, Ph.D.
Android devices can execute powerful attacks and leverage vulnerabilities within the Internet of Things (IoT) in ways similar to personal computers. Attacks on the Android platform against mobile devices is an area well explored by numerous researchers. Devices powered by Android that are not mobile in nature do not share many of the attack vectors found in prior studies, and the limitations of these devices had not been explored. We first examine the means with which one can exploit a TV box device that is connected within the IoT, and what gaining access to such a device means. We demonstrate the security severity of the Android TV box brought to home networks with case studies. We also show that these proposed attacks cannot be detected by current antivirus applications. Furthermore, we propose an effective multi-line defense approach to secure our home networks against these attacks.
# Contents

Abstract i

List of Figures iv

List of Tables v

1 Introduction 1

2 Background and Motivation 3

2.1 Background ................................ 3

2.1.1 Exploiting MXQ Pro Android TV Box .............. 5

2.1.2 Motivation ................................ 6

2.1.3 Threat Model ................................ 7

3 Busybox: A Double-edged Sword 8

3.1 The Sandbox Test Environment: An Experimental Home Network . . . . 9

3.2 Case Studies: Practical Exploitation on the Testbed Network .......... 11

3.2.1 Attacking the Security Camera .................... 11

3.2.2 Attacking a Vulnerable Host ...................... 13

3.2.3 The Screenshot Attack .......................... 14

3.2.4 Undetectable by Android Anti-Virus Apps .............. 17

3.2.5 Limitations of the Attacks ......................... 17

3.2.5.1 The Importance of Timing ....................... 17

3.2.5.2 The Constraint of Connectivity ................... 19

3.2.5.3 The Benefits of Power ......................... 19

4 Attack Mitigation 20

4.1 A Multi-line Defense Solution .......................... 20

4.2 Disabling Busybox .................................. 22

4.3 Supply Chain Concerns and Flashing Attacks .................. 22

5 Related Work 24

5.1 Embedded Devices .................................. 24

5.2 Android Security .................................. 25

5.3 Internet of Things ................................. 26

5.4 The Drammer Attack ................................ 27

5.5 Dirty Cow Exploit ................................ 28
Contents

6 Conclusion 29

Bibliography 30
# List of Figures

2.1 The MXQ Pro .................................................. 4  
2.2 Teensy ....................................................... 5  

3.1 Landscape of Our Test Environment .......................... 9  
3.2 Nmap Scan of Network ...................................... 11  
3.3 Attack model .................................................. 12  
3.4 Exploitation ................................................... 13  
3.5 Images taken from the Compromised Webcam ................. 13  
3.6 Exploitation ................................................... 13  
3.7 Enumeration of Users ........................................ 15  
3.8 Screenshot Attack ............................................. 16  
3.9 Data Upload .................................................... 16  
3.10 Kaspersky, Malwarebytes, Norton .......................... 18  
3.11 Usage Effect on Attack Susceptibility ....................... 18
List of Tables

2.1 Streaming Devices ........................................... 4
Chapter 1

Introduction

Android has been the leading operating system for mobile devices, e.g., smartphones and tablets, according to the International Data Corporation. As of November 2016, 86.8% of mobile phone devices are running on the Android platform[1]. In the blooming of Internet of Things, the Android operating system is becoming prevalent in the implementation of new IoT devices, which include devices such as DVD Players[2], Smart TVs[3], Car stereos[4], and Smart Refrigerators[5].

It can be difficult for developers of embedded systems to secure devices from tampering[6]. Debugging and gaining root access to systems running inside the device can be done from USB, UART, JTAG, EMMC, etc.[7] The practice of leaving these ports available makes these IoT devices more vulnerable, which leaves the operating system unsecured and grants access to attackers. Such compromised devices could disrupt the operations of networks and systems that they are connected to.

The Android TV box is different from Android phones as it has limited access to sensitive information when compared to cell phone devices using the same operating system. The attacks performed against cell phones are to gain access to the information within the device. However, the attacks performed against the TV box are attacks on the network, the local devices, and the individual who owns them. The TV box leaves tools available to the attacker in the form of Busybox and root access from the factory.

The TV box is susceptible to the same malware but it lacks the connectivity of a mobile device. Connectivity such as bluetooth, GSM/CDMA, camera, microphone, and the data they receive are not a part of the platform. Android devices come in a variety of form factors and specifications. The computational resources of both devices can be relatively equal, however the TV box is not constrained by battery power which is a concern in some attacks on Android.
In order to demonstrate the impact of attacks from the Android TV box in the smart home environment, we use an Android TV box as a precedent to explore the ways in which compromised devices can damage and carry out attacks remotely against a victim’s network. We conduct thorough experiments to evaluate the possible impact of these attacks, as well as the consumer usage. The exploration of attacks involves both real world and proof-of-concept attacks, and further correlates with usage patterns shown within the usage survey conducted. The results of the study show that typical users are very prone to data leakage through their Android TV box.

**Contribution:** This thesis makes the following contributions to the field:

- We demonstrate that a simple script could leverage the Busybox utilities to launch attacks from the system-on-chip, non-mobile, Android-powered TV device, which indicates Busybox utilities could be a double-edged sword to IoT devices. We also build the sandbox test environment (an experimental home network) to examine the attack capability of the Android TV box.

- We design and implement three examples to attack the network based on the compromised Android TV box, including attacking a vulnerable host, attacking an IoT security camera and a local screenshot attack against the Android TV Box. We demonstrate that controlling the Android TV box could significantly harm the integrity of the network by capturing the sensitive information in other devices. For example, the attacks from the Android TV box could capture the user’s sensitive information including passwords for users’ accounts and images taken without the users’ notice. Android TV can be just as effective as any other platform for attacking network devices despite its limitations.

- We propose a multi-line approach to defend against these attacks and the evaluation results show that our approach is effective in securing our home networks.
Chapter 2

Background and Motivation

2.1 Background

Android TVs are media devices that perform numerous video and audio consumption needs for a broad array of users. These tasks may include listening to music, watching videos, or other media that is loaded internally on the device or streamed from another hosting service. The appeal of these devices is the low cost and ease of use. Media can be loaded onto a flash-drive device and directly plugged into these Android TV devices with a USB. These devices can stream or load media of various formats at no cost to the user. Some of these devices have features that draw attention to them from their costly competitors, in compatibility or form-factor.

Amlogic is a hardware platform developer for system-on-chip devices. The chips they produce are found in cheap Android TV devices. The hardware produced by Amlogic has an ARM CPU, and uses Android as its operating system. Amlogic is a company that fully involves itself in open source, providing its Android SDK freely to business partners, and its Linux kernel free to any interested party. The manufacturer maintains a website which hosts information on debugging, accessing, and the alteration of Amlogic powered devices. These devices are designed with the purpose of being open and well supported, which is similar to the Android OS.

The Amlogic s905 powered MXQ Pro is an Android platform based smart TV capable device. The Amlogic processor is developed by a manufacturer of non-mobile system-on-chip integrated circuits powered by Android. The popularity of these processors spans across numerous offerings by vendors re-purposing the devices for various form factors and preferences among consumers. While the chipset can be found in numerous devices (including TV tuners), the research within this thesis is specifically focused on
Table 2.1: Streaming Devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Rank</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Fire TV</td>
<td>1</td>
<td>89.99</td>
</tr>
<tr>
<td>Roku Streaming Stick</td>
<td>2</td>
<td>49.00</td>
</tr>
<tr>
<td>Steam link</td>
<td>4</td>
<td>59.99</td>
</tr>
<tr>
<td>XBMCMART Android TV Box</td>
<td>8</td>
<td>99.95</td>
</tr>
<tr>
<td>PlayStation 4</td>
<td>12</td>
<td>324.90</td>
</tr>
<tr>
<td>MXQ TV Box</td>
<td>13</td>
<td>34.99</td>
</tr>
<tr>
<td>Slingbox M2</td>
<td>33</td>
<td>119.20</td>
</tr>
</tbody>
</table>

the Android TV application functionality. This particular chipset is 4K compatible and popular with online resellers.

The MXQ Pro was the device chosen to test the various tailored attacks constructed for Android TV devices. It is noteworthy that this device provides root access with no modification; it was not designed as a forward security device. Anyone with access to the box at any point can create any changes to the operation of the device quickly.

Amlogic does not publish their sales information. Searching for vendors sales through websites such as Amazon or Alibaba can give insight into the popularity of these devices. Searching for “Android TV” or “Android media” results in the most popular or “recommended” devices which are again, powered by Amlogic. In regards to media device popularity, visiting Amazon with Google’s “Incognito” mode shows what devices are “Featured” in the “Streaming Media Players” section of Amazon[8].

Table 2.1 shows the rank by the popularity of these devices according to Amazon’s “Best Sellers Electronics Streaming Media Players”. Two of the top 15 items are Android-powered media devices. These devices are Amlogic, USB, and SD card enabled media devices. The XBMCMART Android TV Box and the research device we used in this paper, the MXQ TV Box, are two well-selling devices, which are built from the same hardware, available within the Amazon Marketplace. Both of these devices are identical in hardware and the XBMCMART is already being leveraged by a vendor to do remote support. The device has received reviews from more than a thousand users for only a
In general, there are two types of exploits in regards to the TV box: local exploits and remote exploits. A constraint with local exploits is in the means with which an attacker can compromise a device; they must be physically near it. An example of two local exploits are USB and UART exploitation. Remote exploitations are carried out without local access to the attacker. This attack may involve coaxing a user into running software, installing firmware or launching a remote exploit on a service running on the device.

**USB Exploitation:** The MXQ Pro Android TV box fully supports keyboard and mouse connectivity to any of its USB ports. These USB ports can be used to load malware. Specialized hardware can be used to install and execute the bash script that has been created. With access to the TV device and a "Teensy"[9] micro-controller, an attacker could load and run the attack on a victims Android TV box automatically. The Micro-controller is small enough to masquerade as a standard USB flashdrive or another everyday device. The Teensy controller will act as a USB capable keyboard and mouse able to input a series of commands on behalf of the attacker, then write the script to a bash file and place it in the initialization of the Android devices “init”[10].

**Exploitation over UART:** It was mentioned that this attack could be used on other embedded systems without USB connectivity by leveraging the UART connection to
gain access to the system. [6]. Many devices maintain the UART port connectivity to load firmware and debug devices. The ports may include password protection but many provide a root shell on connection.

**Remote Exploits:** Conventional remote exploits can also be used to compromise the device and launch attacks. Attacks on Android users using phishing are very successful [11]. Android smartphone attacks are growing in complexity, some leveraging SMS exploitation to gain access to banks, propagating through SMS attacks on the contacts of the victim, and intercepting these communications [12]. While these complicated attacks are not necessary for Android TV devices, similar principles of social engineering techniques can also be applied to lure a user’s interest and ultimately the control of their device.

### 2.1.2 Motivation

The security of Android-based smartphones has been a well explored topic since their inception. Numerous defense mechanisms are proposed, which are also supported by advancements of both smartphone software and hardware. However, Android is also widely used in non-mobile devices, as part of the Internet of Things, due to its open-source nature and easy customization to each vendor’s specific requirements. These hardware platforms can be seen powering home appliances, routers, and DVD players. The low-cost and competitive features make the MXQ Pro Android TV box applicable for many scenarios including home entertainment and business activities.

Take for example John, who has no technical background. He searches online for the smart TV box to enhance his TV with smart and advanced features. With only around $10 dollars, John could buy a device and plug it into his home networks without any scrutiny. He might use his new device to view video streaming services. He could sign into his favorite online shopping service Amazon, to purchase items directly from his TV. Meanwhile, John would also open his laptop to finish some work for his employer using his home network. Low-cost devices enrich our lives, but bring more risks to the integrity of the networks.

Furthermore, the impression of low attack capacity on these embedded systems may not be true anymore. These devices are equipped with more customized tools to mimic their full-featured counterparts. The advancements on hardware enables them to launch attacks that are far beyond the perceived computational and software capabilities by the public. For example, the tiny version of the package of many common UNIX utilities, Busybox, is implemented in many embedded systems. Busybox enables the attacker to have many of the utilities of a Unix system within an ARM based operating system.
To make matters worse, Busybox can be exploited by an adversary, who can create simple scripts that use Busybox to attack local and remote devices without uploading additional malicious code into the device.

The above observations motivate our study that is spurred by the new trends of Internet of Things, the goal of which is to connect every object into the Internet, as well as propose potential defenses. In this thesis, we abuse the Android TV box to measure the security severity of bringing the unvetted IoT device into our networks and answer the following high-level questions: Can an Android TV box be as effective as any other platform for attacking network users and devices despite its limitations? What is the degree to which these devices can harm the integrity of our networks? In the next section, we demonstrate that we could easily turn the device into an attack platform with one script taking advantage of Busybox. Even though we implemented the attacks using specific devices, the threats that the TV box raise apply in general to many of smart TV devices.

In the following, we first present the threat model and then show how a simple script could leverage the Busybox to craft attacks from the Android TV box.

### 2.1.3 Threat Model

In this thesis, we assume the role of an adversary, who can exploit Android TV box via USB/UART or remote exploits. The Android TV box was packaged with Busybox installed by default from the manufacturer. We infect the device with our crafted malicious code that takes advantage of Busybox, which silently runs within the Android TVs normal operation and the device is totally in our control. After the device is compromised, we place it in our testbed network, which is a sandbox environment for us to understand the characteristics and limits of attacks launched by the Android TV box. We connect to the Android TV device, which is also in use by the victim and is sharing a network with all other devices, including laptops, cameras, printers, etc. The Android TV box will only be aware of devices within the testbed via a reconnaissance tool, and is unaware of which devices can be attacked without the use of this tool.
Chapter 3

Busybox: A Double-edged Sword

Busybox has been downloaded from the Google Play store more than 15 million times [13]. The set of tools is also available for direct download from Busybox [14] and is included in various embedded system firmware packages [15].

Busybox is a suite of common Unix based binaries that have been made available for embedded systems. The multicall binary creates a symbolic link to each command added to the PATH directory within the system in order to emulate the multiple binaries of Unix systems. This single binary contains the commands CHMOD, CHOWN, FTPGET, FTPPUT, GZIP, NC, Telnet, and many more. These tools grant permissions to download, zip and unzip and connect to the attacker’s remote service. All of these tools are included within the Android firmware delivered with the Android TV box. Busybox is popular with individuals who need a command-line interface with their Unix based systems, including Android. The utilities are not as robust as the Unix binaries they are based on, but they are sufficient for crafting attacks. Busybox provides us with native utilities able to run in the background invisible to users.

Busybox is a double-edged sword for the users of Android TV boxes. The Busybox utilities are required for various applications which require root to run on Android devices. For users of Android TV devices not using the rooted features of an Android TV device this binary poses a threat without benefit. We craft our malicious code, which compromises the Android TV box, with a combination of available penetration testing software for UNIX systems and a bash script. The purpose of this script is to perform a proof of concept for tailored attacks against the Android operating system. The Android TV device does have limitations based on its operating system and architecture. The hardware limitations include a wireless card without promiscuous mode, Bluetooth connectivity, or an RF transceiver. The device can only support software compiled for
an ARM system. The device itself has only 8GB of storage available meaning attacks are limited in the amount of space they can consume.

3.1 The Sandbox Test Environment: An Experimental Home Network

In order to characterize the attack territory from the Android TV box and its limitations, we assemble an experimental home network which is a sandbox test environment for the compromised Android TV box. The landscape of our test environment is shown in Figure 3.1. We emulate the real environment by including various devices that typically appear in the home network, including a 50inch Visio Smart TV, the MXQ Android TV box, one Xbox One console for media usage, two D-link brand security cameras, two Surface Pro 4 devices, a 17 inch Mac Book Pro, and two iPhone 7s. These devices are connected over WiFi. As well as two custom gaming Desktop PCs and printer that are connected via Ethernet.

We assume the testing environment is the victim’s network with the Android TV box compromised. We launch the attacks, mentioned in the following, against these devices via the Android TV box. In our experimental home network, these devices are not all exploitable. However, they are available to be scanned by the compromised Android TV box, and provide the attacker with additional information about the users within the network. In this test, MXQ Android TV box collects information from surrounding
devices and transmits its collection via the Internet and stores this on the attackers remote host. In the following, we will introduce the detail specifications of these devices used in the testbed.

- The Android TV box is the MXQ Pro that has 1GB of system memory and an 8GB NAND flash for storage. The Amlogic s905 processor which includes a quad core ARM Cortex-A53 CPU and Penta-core ARM Mali-450MP. For peripherals, it has 4 USB 2.0 ports and support for an SD card. For connectivity, it hosts Ethernet and WiFi. For video connectivity, it has an HDMI 2.0 port, AV and SPDIF.

- The two security cameras are D-Link DCS-930L IP cameras with the “A1” hardware configuration. These cameras run a custom 2.6.x Linux distribution from the D-Link factory. The devices have 32MB of RAM, a RaLink RT3050 CPU, and WiFi/ Ethernet connectivity.

- The Xbox one is a standard Xbox connected via WiFi. It houses a 500GB hard drive.

- The PC devices are two Surface Pro 4 devices, i5 Intel processors, with 8GB RAM and 128GB of disk, connected over WiFi. Two built gaming desktops both with i7 Intel processors, Nvidia GTX 980 graphics cards, 32GB of RAM, and an HP printer connective by Ethernet.

- The two phones are iPhone 7 devices, both in 128GB storage configurations, connected via wireless. A 50-inch Vizio smart TV connected via Ethernet to the network.

- The router is a Netgear r6300 flashed with DD-WRT firmware. The router has AC1750 support, a dual core 800MHz processor, 256MB of RAM, USB 2.0 and 3.0 ports. The device provides the WiFi and Ethernet connectivity to the rest of the network.

- The final device is the Surfboard SB6190 which provides the Netgear r6300 with Internet connectivity as the cable modem. The modem is DOCSIS 3.0 compliant and supports download speeds up to 1.4 Gbps and upload speeds of up to 262 Mbps.

**Reconnaissance from the Android TV Box:** Nmap [16] is a security scanning software used by the attackers to carry out network reconnaissance. The resulting scans show the connected devices within the network. The scans detect various open ports and available services within the network that can be leveraged to execute attacks on
the devices connected to it. The scans also provide information about the number of Hosts, version, and OS. The output is concatenated into a file and sent to the attacker for review. Leveraging nmap [16] and the framework of the FTP scripts we can scan the network for all connected devices. The script searches and records possible attacks on open ports and available services, which are then zipped and sent to the attacker.

![Nmap Scan of Network](image)

**Figure 3.2: Nmap Scan of Network**

### 3.2 Case Studies: Practical Exploitation on the Testbed Network

In this section, we will present three case studies detailing trials using the Android TV box to attack the testbed network.

#### 3.2.1 Attacking the Security Camera

IoT cameras are sensitive devices within the IoT landscape for anyone who has them installed. The explored IoT camera is the D-Link DCS-930L, which is an IP camera that sends video and audio stream to the user from the device across the local WiFi connection. The device can be viewed locally or remotely using the D-Link app on the smartphone.

To exploit this IP camera, we first scan the network with the network mapping software nmap, which is accessible via the wget function within the Busybox utilities. Nmap needs to be extracted but not installed on the device; this is handled by the same script that downloads nmap. We can now evaluate the network and make a decision that one or many devices or services can be exploited from the nmap scanner’s response. One
of the devices shown within the testbed by nmap is the D-Link DCS-930L device. The D-Link DCS-930L has a vulnerability in that it can be backdoored with assistance from the metasploit tool.

To perform an attack on this Android TV device the Metasploit framework is required. Metasploit is not supported for Android without installing a more robust version of linux onto the device, which can not be done remotely. To perform this attack, we decided to access the network from a laptop PC. Using this x86 device will alleviate the constraints of the Android TV box. The Android TV box will be used as a VPN service for the network. This is done by installing a VPN server on the Android TV device which can be connected to by the attackers other devices. This provides VPN pivoting and layer-2 access to the network for any number of devices an attacker may want to connect, as shown in Figure 3.3. We connect to the network with a laptop device with standard capabilities, including a web browser and Metasploit. The attacker inputs the IP address of the device within the network which was obtained using nmap. Metasploit attacks the DCS-930L and provides a root shell to the attacker. The exploit used is an authenticated remote command execution Metasploit module [17]. This attack is able to compromise and launch a shell on the linux powered IP camera. This shell gives root access to the attacker and allows for changes to the file-system. An HTTP webcrawl of the device is taken to show any directories available via a web-browser. The scan provides us with the directory “mvideo.htm” which when visited shows the view of the IP camera in real-time. The video here can now be recorded or screenshot remotely by the attacker.

Figure 3.3: Attack model
These credentials can now be used to monitor the house of a victim via the installed IP cameras. Figure 3.4 shows the exploitation of the network IP camera via Metasploit; the image is the view from the compromised camera via a web browser. The compromised webcam can now provide video back to the attacker. This study shows this device can act as a “drop-box” for attackers who wish to exploit network devices without concerns for the limitations of the TV box operating system architecture.

3.2.2 Attacking a Vulnerable Host

The purpose of this section is to demonstrate that server, laptop, and desktop devices can be attacked. These attacks can be performed directly from the Android TV box without the need for VPN pivoting.

We launched a vulnerable virtual machine from a desktop machine using “Metasploitable” [18] and VMware Workspace then bridged its connection to the host network. Metasploitable is a virtual machine that is intentionally insecure, it has a series of exploitable
services to demonstrate and practice various penetration testing techniques. Hosting this VM creates a network device which can be exploited.

Firstly, the TV box is used to scan the network to find the vulnerable host. The host is then scanned to discover ports available for exploitation. The two open ports which will be attacked are the “VSFTP” FTP service running on port 21, and the backdoor Trojan disguised as the ingreslock port on port 1524. The Android TV box was able to perform attacks against the host and use backdoors available within Metasploitable. The first attack is on the “vsftpd” FTP server. The version running within Metasploitable has a backdoor that opens a listening shell from port 6200, which can be Telnet into. To access this shell Telnet into the standard FTP port “21” and provide a username that ends with the characters “:)” in that order, the smiley face will open port 6200. We can now Telnet into the 6200 port to be granted a root shell. Another backdoor accessible from the Android TV box is the ingreslock port backdoor. For this attack, we Telnet into the backdoor available within the Metasploitable host provided by the ingreslock port at 1524. The port gives us root access on connection to the host. Another attack from nmap on the users on the Metasploitable VM can be through the nmap script “smb-enum-users.nse”, which is included with the nmap repository. The script uses the Samba smbd ports 139 and 445 to find all of the users within the remote system. For the attack, we run “nmap -script smb-enum-users.nse -p 445 192.x.x.x” on the target and concatenate it to a file for later viewing. The script works by scanning users RID values.

The information would give insight into the users of the system, as shown in Figure 3.7. The username provides information about who is currently logged into a system. The Metasploit administrator account was found for this example, but in a home environment, usernames may be people’s first and last name. This also gives a username which can be used to bruteforce other services from the TV device.

### 3.2.3 The Screenshot Attack

There are special considerations surrounding data leakage on a media specific device such as the Android TV device. Data leakage is privacy sensitive information being leaked through insecure channels. While HTTPS can mitigate this leakage, a substantial portion of the information transmitted today is over HTTP and subject to cookie information leakage[19]. The information sent from these devices can endanger the entire security landscape[20]. Studies have shown that the most detrimental information leakage is commonly in the form of emails[21]. However, this data is not typically accessed by Android TV devices, and certainly less so than mobile devices, which commonly
store sensitive information. Rather the information provided by the Android TV device is more organic to the behavioral patterns of the individuals using the devices [22]. The shows, websites, and media consumption tasks can be just as personal and sensitive as their mobile relatives, but the information found may be from multiple individuals rather than a single user.

Screenshot attacks, also called snapshot attacks, circumvent the need to capture HTTPS traffic by capturing photographs of user activity encrypted or otherwise. The attack uses the android command-line utility ADB, or Android Debug Bridge. The script behaves much in the same way as the sniffing attacks. Instead of capturing “PCAP” sessions, photos are captured in predefined intervals and uploaded to the attacker’s server. The frequency of snapshot can be scaled up or down depending on the needs.

We can capture a screenshot every hour to better understand the usage over a week. The attack can also be performed constantly as we demonstrate below.

In Figure 3.8, a screenshot attack demonstrating recording a user entering their password via an Android TV remote is displayed. The TV box highlights the characters typed by the user. The script for this use-case was set to record images at a rapid
enough rate to capture the input of the victim’s password. The example is of an Amazon web login, but this attack applies to all account information inputted using the MXQ Android TV remote control or similar device.

Snapshot attacks show information that is not always available to key-loggers, such as sensitive videos which are loaded by the victims of the attack. A key-logger, while useful to the attackers for stealing account information, does not collect the data leakage the snapshot attacks will.

Key-logging attacks are well documented across the Android platform[23]. The data from these tools does not provide information for applications that use adaptive searches such as Youtube. Youtube will provide searches based on the browsing habits and history of its users; logging keystrokes is not enough for an attacker to appropriately understand
what the usage patterns of an individual are. The attacker can use a screenshot attack to monitor the usage of the device over time, analyzing what videos are being viewed, some of which may be privacy sensitive. Videos on treating illness, videos on traveling, or even family videos from an unlisted channel on Youtube, or USB thumb drive are very confidential.

Figure 3.8 shows the results of running a snapshot attack for a short duration uploading every second the device is in use. For practical purposes, we recommend using both a key-logger and a snapshot attack for a more holistic view of the attack vector available to the Android TV device. Using standard keylogging for passwords, and screenshots to bolster the information gathered, adequate information is provided to the attacker.

3.2.4 Undetectable by Android Anti-Virus Apps

Android Anti-virus is unable to detect malware using the Busybox utilities. After installing a series of popular Anti-virus software available for Android, none of them could detect or make recommendations to remove the script.

We performed tests on three popular Android Anti-virus applications (Kaspersky, Malwarebytes, and Norton). All three were unable to find the malicious script running on the device, Malwarebytes specifically scanning 1845 files on the small 8GB of storage the device holds. The VPN server or open Telnet services were in no way mentioned, and the AV software stated the device is clear of threats, as shown in Figure 3.10. The scans, attacks, and other uses were performed in the background without any notifications from these security applications.

3.2.5 Limitations of the Attacks

Special considerations are necessary when analyzing attacks on Android TV devices. These devices are non-mobile which adds complexities to methods both aiding and harming the success of attacks using the Android platform. Figure 3.11 shows a model of the attack rating of a device on the network, based on the usage patterns within its environment.

3.2.5.1 The Importance of Timing

The first limitation on attacking networks via the Android TV platform is its operational nature. These devices are regularly turned off when not in use which creates a smaller attack window. The attacks can not take place when the device is powered off. The
Figure 3.10: Kaspersky, Malwarebytes, Norton

Figure 3.11: Usage Effect on Attack Susceptibility
attacker can instead seek to create persistence on the network by attacking a device that is regularly powered, e.g., a desktop device. Invasion through USB propagation or other means to locally exploit a persistently powered PC. If these attacks fail, the methods can be updated remotely by the attacker to best leverage the time given by the victim. This adjustment is advised by the usage patterns shown through data leakage of the device. If a victim visits sensitive data that is transmitted from the device, only between the hours of 4PM and 5 PM, exhaustive monitoring and surveillance attack can take place at those times.

3.2.5.2 The Constraint of Connectivity

Mobile Android devices such as cellphones leverage their GPS, sound, camera, and GSM/CDMA network capabilities, which provide always-on connectivity to the malware. The devices can transmit all data provided by their sensors to the attacker for data-leakage purposes. Android TV devices do not share these connected features. The device can be expected to be attached to the same network at all times after purchase, so long as the owner and user of the device are not changed. While this does limit the amount of information gathered by the attacker over time, it allows the attacker to profile the network in ways a mobile device that is switching WiFi connections throughout a day. Changing connections does not provide the attacker with a consistent network to perform attacks on.

3.2.5.3 The Benefits of Power

Android TV devices do not rely on batteries which give an attacker the means to carry out attacks on the network without concern to power consumption. A concern noted by other researchers [24] is the constraint of battery longevity when performing attacks. As long as the device has been left powered on by the user, the device can act on the local network and the Internet. Many of these devices are multi-core, network connected, and substantial devices an attacker can leverage to carry out both local and remote attacks. This includes, but is not limited to, denial of service attacks which have been noted to be very battery intensive on mobile Android devices.
Chapter 4

Attack Mitigation

We recommend disabling Busybox on devices where it is not needed to prevent such attacks. Furthermore, we present a multi-line defense approach to secure our home networks.

4.1 A Multi-line Defense Solution

In this section, we present a multi-line defense solution to defend against the attacks that can not be detected by both host-based and network-based solutions. Our multi-line defense relies on the analysis on the policy defined for each IoT device, the incoming and outgoing data flow from the device, and the payload analysis on each packet. The novelty is that we take the systematic information, both signature and behavior information, into consideration in order to defend against such stealthy attacks. Snort [25] alone with the available rulesets provided by the community is not enough to protect against these attacks.

The first line of defense is the policy, which is a configuration that allows access between in-network IoT devices in use during regular operational boundaries. The second line defense is the data traffic flow, which permits information between devices both locally and remotely, and specifies which devices can interact with one another. The third and last line of defense is content analysis, which involves the deep packet analysis to evaluate the content of individual packets.

To create the first line of defense we take a status of the standard policies required for the device to operate within the IoT. The rationale behind this is that IoT devices leave many ports open without necessary, but would be leveraged by attackers. We firstly identify and whitelist all protocols necessary for the device to communicate with other
network and remote devices. If a device should only communicate on a specific port, all other communications on any other ports should be flagged as suspicious. The flags are tailored to what testing shows as a standard packet size for operations (Dsize), to assess abnormalities that may be associated with data exfiltration. We then use these findings to inform our decisions in second line of defense, where we analyze the flow of the information across the network.

For the second line of defense, we try to capture the behavior of each IoT device within the network. Some IoT devices should never be permitted to communicate with each other. As an example from our test earlier discussed, the Android TV should never be allowed to communicate with the IP camera for any reason. The Android TV should not be able to communicate with any local network devices besides the gateway device which it uses to connect to remote sources (the router). We propose using the flow flags as well as the direction operator to capture such behavior, which gives us the capability to set acceptable network flow with regards to which devices can communicate and in which direction.

For the third and final line of defense, we propose to use the deep-packet analysis to examine the payload of the packets traversing the network and assess the risk of targeted payloads. This concurs with the signature-based approach. We propose to use Snort's content flag provides us with the ability to monitor which users log into and out of any system remotely, as well as detect various documented exploit signatures. This final layer uses the available signatures provided by the Snort GPL Community Rules and prevents attacks not only against the IoT space, but also local desktop and laptop devices.

The first step to attacking the network from the Android TV box is to scan the local network using the community rule sets which are included with Snort. In testing, all forms of scanning were detected and flagged. The default Snort community alert library did not recognize the attacks on the network devices as well as the screenshot attack until additional flags monitoring FTP, Curl, and Git requests were added.

On our test network, we placed a router which provides IDS and IPS functionality to all IoT devices on the network via the Snort software. The device runs in-line with traffic and is configured to flag or prevent attacks on the local network. In testing, the implementation of the instrument did not adversely affect latency, download or upload speed to devices hosted on the network. Using the methods detailed above, we can secure the home network environment against attacks on these IoT devices. The method requires configuring the policy, flow, and content that is acceptable for each device. We next evaluate the implementation of our approach on the network router. We benchmark the effects of this implementation on the home network. To evaluate
the bandwidth overhead in a home network, we benchmark the router on our testbed network. We alternated between our approach being on and off and ran these tests consecutively. We then aggregated the bandwidth of the network over time for these two scenarios. The study found that there is a slight negative impact on the bandwidth for network devices while our approach is running. However, such impact is not dramatic enough to dissuade users from the benefits of the proposed approach to securing their IoT devices. We also found zero false positives in our experiments, and other benchmarks, e.g., CPU, memory and storage, were found to be largely nominal in the test performed with our approach.

4.2 Disabling Busybox

Disabling Busybox can prevent the usage of tools and the script presented. Busybox is not the only source of utilities that can perform these attacks. Other platforms such as toybox [26] provide Unix based utilities that were used to create this attack. Disabling the access of these tools or removing them completely is a step toward securing your Android TV box without compromising functionality. The attacks shown here could use ToyBox in a way similar to how we have leveraged Busybox. The removal of Busybox may limit a users ability to use unauthorized software but does not effect its functionality as a TV box for legitimate usage in any way. Tests of streaming videos, playing games, and loading personal files were all completed successfully in various formats and sizes.

4.3 Supply Chain Concerns and Flashing Attacks

Malware has been installed by vendors within the supply-chain before the device has been received by a retailer in the past [27]. The method for this could be malicious or undesirable. Consumers see this on numerous products in the form of “bloatware” that is subsidized by a third-party wanting to increase the number of users accessing their services or viewing their marketing material. Vendors may do this to enhance the revenue received by the devices sold. They can do this by installing malware that replaces advertisements with links to the ad account of the vendor rather than the downloaded app’s[28].

In addition, Android Trojans, which are capable of downloading and executing APK files, can be bundled with firmware[29]. The attack works by covertly installing its own ads within a box that overlays existing apps to glean ad revenue off of the users of all applications on the device. When attempting to delete the software the malware will download and reinstall itself[29].
There is potential for a rogue employee of a company distributing these devices to flash an infected OS onto an Android TV box. The firmware for these devices is readily available from the manufacturer’s website and adding the malicious payload to the firmware is trivial. Individuals should beware of downloading the firmware from outside software vendors for the purpose of flashing their device. Numerous third-party websites link to free un-governed file-hosting websites not affiliated with the product and verifying a file's validity is not a possibility without a complete audit of the firmware being installed [30].
Chapter 5

Related Work

In this section, we will explore related work surrounding the Android platform, IoT, and embedded devices.

5.1 Embedded Devices

Brasser et al. [31] showed how embedded devices are growing in popularity. The persistence of DoS and replay attacks within the IoT creates a greater need of certifying connections between devices. Desnity et al.[32] showed that optimizing security for light weight devices can be troublesome; insuring that devices are small yet powerful creates security concerns. Making decisions about the efficiency of security techniques is key to their usage within IoT. The low overhead ARM trusted execution environments developed by Chang et al.[33] performed attacks based on user interaction rather than simply running the application and can evade dynamic analysis tools. This is done by avoiding the action unless user-like behavior is detected. Until then the application will not use malicious techniques. Using the key identifiers for embedded systems within the IoT gives security professionals new ways to circumvent side channel attacks. These methods have been documented by Zheng et al.[34] where they detailed the communications and execution of machine code on local and remote hardware. Embedded devices running various operating systems need distinct considerations to security practice. What is effective for some architectures and hardware platforms is insufficient for others as shown by Jabeen et al.[35] and his work documenting the various approaches to these devices.

A major concern with these embedded devices is the debug ports available to attackers. The deactivation of specific areas of connection can build trust in embedded devices, but
this is not enough to sufficiently secure a platform. This approach also has drawbacks for developers as shown by Francillon et al. [36]. Valente et al. [37] have used heuristics to analyze bugs within embedded devices which do not share the code and binaries loaded onto these devices.

Embedded systems can be leveraged to stop attacks as well after security considerations are covered effectively by these frameworks. The detection of malicious usage of Wi-Fi networks can be accomplished by small embedded devices previously insufficient to do analysis of wireless networks [38]. A variety of attacks from session hijacking to wireless scanning can be monitored.

5.2 Android Security

With the advent of Android 6.0 new permission systems are changing in complexity and not necessarily for the better. A group of researchers, Tchakount et al. [39], found third-parties are still able to gain footing in the Android system. To combat this, researchers are developing application forensic analysis frameworks that generate graphical representations of Android applications’ behavioral patterns for audits by developers. Some of these frameworks also assist in the analysis of spyware [40]. Even with these frameworks, the current issue is in the lack of granularity in regard to current Android privacy configurations. Jain et al. [41] showed an unprivileged application can leverage the privileges of an approved application to gain unauthorized access.

Yang et al. [42] presented methods for dynamically analyzing Android applications to find the actions of an executable and create a risk score to catalog the likelihood of the executable being malicious. To document low level malware detection through Linux level detection methods, Isohara et al. [43] developed analysis methods to yield signatures which relate to archived attacks at the kernel level. These signature methods are useful, as profiling methods for Android applications are currently not available within the capabilities of the Google play store. Wei et al. [44] presented methods to profile these platforms by detecting and delineating their usage patterns in the ProfileDroid framework. With the direction of these frameworks, machine learning algorithms coupled with behavior analysis can yield new and improved Anti-virus systems; the goal of Sen et al. [45] being to eliminate both false negatives and false positives. Not all apps are intentionally malicious, Android applications can exhibit negative behaviors due to poor developer practices. Tools by Bost et al. [46] profile and identify behaviors that are not malicious but have negative effects on the usability of the OS.
New approaches to attacking Android systems are always being developed. Johnson et al. [47] showed denial of service attacks on the Android platform which cyclically reboot the attacked device disabling it, until restoration by outside media or factory reset. The attack can effectively disable embedded Android systems. Attacks on the user interface called GUI hijacking create overlapping user interfaces which lead to confusion with users and assist malicious parties [48]. The goal of which is to coax users into leaking private information. Hyuk et al. [49] have developed a method to recover data that would have otherwise been destroyed by malware in the attacks on the users device, which may assist in profiling these attacks. Even micro-controllers are starting to use Android as a platform for integration with IoT as shown by Chtourou et al. [50] who demonstrates controlling an array of electronics via Android services.

Research surrounding the privacy leaks from within the Android Operating System are well documented [51]. The researchers defined what constitutes sensitive information, such as GPS location. All of the information noted by the researchers is mobile exclusive except WiFi. This means the exploration of these apps and the information leaked to them are not relevant for non-mobile devices. However, their methodology can be used to evaluate the ramifications of these applications on non-mobile devices. The source of many of these leaks are similar: applications, and advertisements. The researchers found that particular ad libraries leaked additional phone information. These ad libraries are leveraging location information with the ability to tailor ads to the location the device is being used to further grow their revenue stream through clicks.

Creating a network of devices running arbitrary software is not beyond manufacturers means. This device could be used as a means to control the bandwidth of a victims network to complete denial-of-service attacks. A number of studies and individuals have hypothesized large Android Botnet attacks, which is an expanding focus as the popularity of Android devices booms [24, 52].

### 5.3 Internet of Things

Sain et al. [53] evaluated the security of IoT devices and cataloged the challenges that develop over time. Tracking the history and growth of these devices assists in managing research direction for this industry. Sen et al. [54] showed that analysis of the resources needed within changing environments using IoT devices to secure these platforms may not be sufficient as they do not focus on more than privacy and access control. Frameworks exist to map potential security attacks for IoT devices. Systematic classification such as the taxonomy created by Nawir et al.[55] assists developers in understanding
risks. The implementation of these frameworks within businesses can be difficult. Irshad [56] has gone so far as to systematically evaluate the frameworks presented by other researchers to establish the most successful one within an organization.

Challenge codes for IoT cameras such as those were presented by Cojocar et al. [57] to insure the integrity of video recording systems. These codes added a new defense and work to improve trust in the image being seen. Algorithms such as those presented by Arseni et al. [58] must be crafted to secure IoT infrastructures and must be supported by a variety of architectures. There are an array of security frameworks surrounding the expansion of IoT. Testbeds with IP cameras and other smart devices are an effective way to evaluate the privacy issues faced by IoT devices. This form of testbed was used by Tekeoglu et al. [59] to observe insecure protocols, outdated software, and information leakage. The possibility of denial of service attacks from IoT devices grows larger and stronger [60]. These devices must be secured, but currently lack the controls we need to secure such systems. The hardware of many of these devices is insecure; wireless sensor networks can be penetrated gaining access to micro controllers through simple brute forcing [61]. Securing the bootstrap loader password can deter attackers by lengthening the time required to penetrate these devices. Securing the data leakage of these hardware platforms is a necessity, specifically with privacy sensitive IoT devices. Sharma et al. [62] detailed how blood banks are using IoT to assist individuals in donating blood, which has definite security considerations.

5.4 The Drammer Attack

There are many remote exploitation attacks that have taken place against Android-powered devices. The Drammer attack is a recent discovery that can be used to attack modern devices [63]. This exploit does not require user permissions or any software to exploit as it is attacking the system inherent in both ARM based devices and x86 machines. This is accomplished by attacking the patterns used by memory allocators and flipping the bits of said memory strings using a row hammer attack. The fault of the system being the DRAM chips used within these devices.

With the direct memory access or DMA cache bypassing process, the researchers were able to hammer on the rows without flushing the cache. DMA as a process is fully available to the attackers and could be used to flip the bits within these rows. The attackers allocate a large single block of memory, de-allocate, then re-allocate a significant number of smaller blocks of memory. This creates a series of page tables within a region of memory defined by the attackers. The attackers template the memory, figuring out which characters can be flipped, then fragment the memory. This creates an error
within the page management system which results in OS controlled memory becoming available within the applications user memory. This gives full read and write access, which adversaries can leverage to give themselves root access.

5.5 Dirty Cow Exploit

The Dirty Cow exploit is an attack against the Linux OS and by proxy the Android Operating System. The exploit with the reference CVE-2016-5195 has proven robust against new and old versions of Android by leveraging a copy-on-write error by attacking a race condition within the memory of the OS [64]. The exploit allows the attackers the ability to gain root. This issue is not new, rather it has existed for a significant amount of time, but it was never appropriately patched.

The attack works by opening a read-only file, then uses “nmap” to create a memory segment. Using a flag that creates a private copy of a file and maps it into memory, this is called copy-on-write or COW. The copy-on-write function means the area in memory is copied outside of its physical location and relocated when a write process initiates. This means a file is being mapped into your memory, in which you can read or write to a copy of the file; this should not result in the user being able to make changes to the file.

The attack then uses a flag to tell the operating system the memory can be reallocated and likely will not be used soon, and reloading of the mapped file can commence. This error occurs because the paging files are both pointing to the same space in physical memory, the file is updated. The kernel writes the update to disk with its system privilege despite the malicious action not having the permission to do so.
Chapter 6

Conclusion

In this thesis, we have demonstrated that the TV box device can be quickly compromised by remote and local exploitation. The compromised TV box device could launch powerful attacks to the home networks via the Busybox. Our study has shown that the Busybox is a double-edged sword, which could equip attackers with more attack capacity. Furthermore, we have presented an effective multi-line defense approach to secure home networks. We also recommend auditing the devices chosen to be within any home landscape. These devices should be viewed as personal computers connecting to any other home network with similar attacks and considerations. The lack of governance for these products is both a feature and a danger for those individuals choosing to use them.
Bibliography


