Opening Pandora's Box: Effective Techniques for Reverse Engineering IoT Devices

Omer Shwartz*, Yael Mathov*, Michael Bohadana*, Yuval Elovici, Yossi Oren
{omershv, yaelmath, bohadana}@post.bgu.ac.il, {elovici, yos}@bgu.ac.il

Ben-Gurion University of the Negev

Abstract. With the growth of the Internet of Things, many insecure embedded devices are entering into our homes and businesses. Some of these web-connected devices lack even basic security protections such as secure password authentication. As a result, thousands of IoT devices have already been infected with malware and enlisted into malicious botnets and many more are left vulnerable to exploitation.
In this paper we analyze the practical security level of 16 popular IoT devices from high-end and low-end manufacturers. We present several low-cost black-box techniques for reverse engineering these devices, including software and fault injection based techniques for bypassing password protection. We use these techniques to recover device firmware and passwords. We also discover several common design flaws which lead to previously unknown vulnerabilities. We demonstrate the effectiveness of our approach by modifying a laboratory version of the Mirai botnet to automatically include these devices. We also discuss how to improve the security of IoT devices without significantly increasing their cost.

1 Introduction

In the early days of computing, low-cost ubiquitous devices were generally powered by simple microcontrollers. These microcontrollers typically ran a very limited software stack, ranging from a fixed-function program running in a busy loop to a limited functionality real-time operating system (RTOS). As technology matured, it became more cost-effective to create these devices around a fully-featured operating system such as Linux, taking advantage of the existing code base and of the relative ease of development and debugging. This is especially the case in the Internet of Things (IoT), which can be defined as a network of smart electronic devices with internet connectivity. In the past years we have been witnessing a dramatic rise in the amount of connected devices and recently, wireless connected devices. The number of IoT devices is estimated to reach 50 billion by 2020 [17].

The task of the device security engineer has also evolved with the move from ASICs and simple microcontrollers to complete Linux devices. Traditional hardware attack methods which target ICs are less effective in this modern situation,

* These authors contributed equally to this paper

since the hardware can be assumed to be generic and even shared between different vendors. Ubiquitous network connectivity also changes the attack model, making it interesting to examine the vulnerability of devices to remote attacks, or the ability of an attacker to translate a single instance of physical access to widespread damage to many devices. Indeed, the introduction of these small, embedded devices unto the web and into residencies and businesses was quickly followed by emerging security challenges [39]. The rapid growth in the quantity and variety of IoT devices created a scenario where millions of devices [27] are deployed while the consumers may know very little about their composition and security. This is especially crucial since IoT devices are often equipped with a wide array of sensors, are connected to private networks and control a variety of physical systems, from entry gates and door locks to HVAC systems (Heating, Ventilation and Air Conditioning) [29].

In this work we present a general methodology for "black-box" reverse engineering of complete stack IoT devices. The techniques presented should answer many use cases and can be used as a tutorial for accessing new devices. While most of the techniques we use are generally well known, this is to the best of our knowledge the first time they are applied systematically to many different IoT devices. This allows us to make quantitative arguments about the state of IoT security today.

In detail, our paper makes the following contributions: We present a systematic reverse engineering workflow appropriate for complete-stack IoT devices in a detailed and tutorial-like manner. We apply this workflow to sixteen IoT devices produced by different manufacturers and discuss their common characteristics and security flaws. Finally, we offer some guidance to implementors interested in making these devices more secure.

1.1 Related Work

Mahmoud et al. [28] present a survey of the current concerns for IoT security. The authors describe the general architecture of IoT devices and the security challenges rising from this design, corresponding to the security principles of confidentiality, integrity, availability and authentication. Sicari et al. [36] claim that the network communication characteristics of IoT devices, combined with the increase in exchanged information, multiplies the potential for attacks on the system privacy leaks. Similar concerns were also raised by Alqassem et al. [6] and by Zhang et al. [40]. Interestingly, most of this analysis centers on security threats to the **user** of the IoT device (i.e. loss of confidentiality and availablility) and less on the risks to the **device itself** (e.g. counterfeiting).

Patton et al. [31] studied the extent of vulnerabilities found in network-accessible IoT devices. They reviewed several network scanners and focused on Shodan [35], a publicly available search engine for internet connected services. Using Shodan, the authors discovered many vulnerable IoT systems including a large number of SCADA (Supervisory Control And Data Acquisition) systems. Similar techniques can also be found in the work of Bodenheim et al. [10].

Tellez et al. [37] focused on WSN (Wireless Sensor Networks) and elements of their security. For their research, the authors chose the MSP430 MCU and investigated it. The BSL password (Bootstrap Loader) that protects the MCU from unauthorized access was presented as a main security feature of the MSP430 MCU. A flaw detected in the BSL password mechanism through reverse engineering techniques allowed the researches to easily break into a secured MCU. The authors also suggest ways for designing a Secure-BSL that can improve the MCUs protection.

Gubbi et al. [20] offered an all-in-one review of the WSN terrain along with the terminology that exists within it. Halderman et all. [21] showed techniques for recovering secrets from DRAM (Dynamic Random Access Memory) modules by transferring the modules into a new machine while minimizing data decays. Lanet et al. [24] showcase methods for reverse engineering EEPROM data of java memory cards. They describe forensics methods which enable the researcher to locate critical data within the memory image, account for errors and eventually rebuild the original applet code that is stored in the card.

Obermaier et al. [30] employs reverse engineering techniques on several wireless security cameras and shows how these are vulnerable to remote attackers with no physical access to the surroundings of the device. The authors show various encryption and communication faults that may allow and attacker to impersonate a camera and eavesdrop or sabotage its communication.

1.2 Embedded Device Software Architectures

Software architecture determines many of a device's properties and limitations, the architecture may include an OS (Operating System) or not. We differentiate between three main types of software architectures present in embedded devices. **Full-stack OS based devices** contain a modern operating system, such as Linux, that separates execution into kernel mode and user mode. While traditionally this architecture was preferred only when versatility and high performance was needed, [22], more and more low-cost devices are now based on Linux due to falling component costs and the ease of developing for this operating system. In particular, many of the cameras we surveyed had a complete-stack Linux implementation. **Partial stack OS based devices** are devices with a special-purpose real-time operating system (RTOS) such as VxWorks or vendor-provided OS implementation. These devices are generally very specifically crafted for their task [16], and tend to omit some of the features of complete-stack OS. Some lower-end or single tasked IoT devices use this architecture, with the RTOS handling WiFi, web protocols, and added vendor code in charge of gathering sensor data. Finally, **devices with no operating system** are embedded devices which directly execute compiled instructions, without any OS support for functionalities such as threading or interrupts. Devices with no OS can offer better raw performance and higher run-time predictability than other architectures, but tend to have increased difficulty of development, causing a longer time-to-market.

While it is the authors' belief that *partial stack OS devices* have the potential for security vulnerabilities, *full-stack OS devices* were chosen as the target for

reverse engineering in this paper. So far, all of the IoT attacks seen in the wild, and known to the authors, had targeted *full-stack OS devices,* as these are more generic and make use of many drivers and open source components that may have vulnerabilities.

2 Reverse Engineering Methodology

Fig. 1: The building blocks of black-box reverse engineering

Following is a description of the flow of actions performed in order to gain access to the software of IoT devices, run foreign applications on it and extract secrets such as credentials used for accessing the device. This section focuses on reverse engineering "black-box" devices where no previous knowledge about the device is required. The tools used for assessing these techniques can be seen in Table 3 in Appendix.

Our black-box reverse engineering process follows a standard workflow that can be seen in Figure 1:

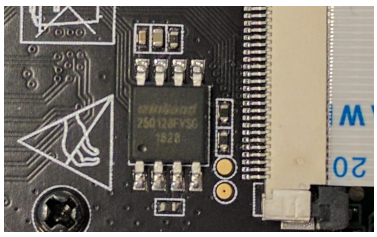
1. Physical inspection of the device.
2. Extraction of the device firmware image and file system:
   (a) Bypass boot-time security and recover the firmware image.
   (b) Recover the data with out-of-band means.
3. Analysis the firmware image and recovery of the secrets inside.

2.1 Inspection of the Device

Most of the devices can be carefully opened without damaging neither the exterior of the device nor the internal components.

Locating and identifying memory components Smart devices that run the Linux operating system require enough non-volatile memory for storing the kernel and additional mandatory file system components. A cheap and efficient way for engineering such devices is placing the memory module outside of the main processor package. Devices engineered in such configurations usually employ a processor that is capable of loading and running instructions directly from SPI (Serial Parallel Interface) Flash memory or EEPROM (Erasable Programmable Read-Only Memory) devices.

(a) F59L1G81A 1GB NAND Flash module inside Xtreamer Cloud Camera



(b) W25Q128FVSG 16MB SPI Flash module inside Ennio SYWIFI002 Wireless Doorbell

Fig. 2: Examples of onboard memory

Understanding the memory technology is crucial for performing firmware extraction when there is no capability to run commands on the tested device, see more details in Subsection 2.2.

It is common to find a memory module that uses technology consistent with the required capacity inside devices. Common examples are: 25XX \ 26XX series eight-pin SPI flash memory with up to 32MB of storage space; larger SPI Flash devices with sixteen or thirty-two pins; NAND Flash devices that come in various capacities and shapes and are usually coupled with a 24XX EEPROM module for holding initial configuration; eMMC (embedded Multi-Media Controller) modules or cards usually containing more than a GB of data. Examples of memory modules can be seen in Figure 2.

Identification of the memory module can be performed by searching of the engraved device codes on the IC (Integrated Chip) package. In most cases the modules used are commonly known and available off-the-shelf with public datasheets.

Locating UART terminals UART (Universal Asynchronous Receiver / Transmitter) ports can be found on many smart devices. UART ports are commonly used for development and maintenance via a Linux console that the port is bound to. UART ports' communication is based on a specified protocol in predetermined baud rate, typically 9600, 57000, or 115200 bits-per-second.

In many cases, UART terminals are embedded into the PCB (Printed Circuit Board) in the prototyping stages of a product's life and are kept in the design during production either to reduce costs of redesign or maintain access for future maintenance. In certain cases, UART terminals are placed in a visible and accessible locations, occasionally marked with their purpose. In other cases the terminals are purposefully or unpurposefully hidden between many other test points exposed on the boards for post-production testing. Connecting to UART terminals allows easy access for communication with the OS, and may also form a beachhead for the effort of reverse engineering.

Basic UART communication requires only three electrical lines: TX (Transmit), RX (Receive) and GND (Ground). A typical UART terminal has two to

four exposed copper pads aligned in a row; when having two pads, the TX pad is pulled electrically towards +1.8v, +3.3v or +5v and the RX pad might not be pulled to either directions; when having three pads, the additional pad is usually the GND pad and should have continuity to the ground plane of the PCB; when having four pads, the last pad is generally VCC and shows up as +1.8v, +3.3v or +5v when powered on.

By using the known properties and appearances of UART terminals it is possible to locate suspected terminals using a Multimeter and verify them by attaching a Digital Analyzer capable of analyzing UART communication. Figures showing various placements of UART terminals can be seen in Figure 4.

UART discovery assistant module In order to assist with the detection of UART terminals on PCBs that contain a large amount of test points, a small device that generates audible beeps when probing an active UART TX line was designed. The device is composed of an ATtiny13A [9] programmable micro-controller along with auxiliary electronics with custom code that switches between three popular UART baud rates, and it beeps when encountering a threshold amount of English printable ASCII characters (characters larger than 0x20 and smaller than 0x7F). The device can be seen in Figure 3 in Appendix. The source code for the module is publicly available in the authors' github repository [8].

2.2 Extraction of Firmware and Data

Handling Bootloader and Linux Passwords While booting, the bootloader loads the kernel and passes over the boot arguments to the kernel. Commonly, within the boot argument is the path for a user-mode process that starts when the kernel completes booting.

After booting, the Linux kernel transfers control of the console to the user-mode process. Traditionally, after executing a list of scripts, the init process may transfer control either to the login or the shell process. When the login process is started, it requests and verifies the user's credentials and instantiates a shell process for the user to control. The login process is protected from brute-force attempts and employs a delay between consecutive password guessing attempts.

When encountered with a login request in an embedded device, a simple technique is to replace the init part of the boot argument with a path to /bin/sh or any other process that can assist with gaining access to the system. This change can be done from within the bootloader terminal, that can be accessed when the boot process begins.

Access to the bootloader is usually done by pressing some key at the first stages of boot. In certain cases the bootloader is protected by password. Since the bootloader has a very small memory footprint, it usually lacks the infrastructure for password hashing and only performs string comparison against a hard-coded password. The password string may be recovered from memory blobs obtained via out-of-band methods.

Using physical attacks for bypassing passwords or recovering passwords Fault injections have a significant role in reverse engineering [32]. The usage of fault injections allows the researcher to generate a hardware fault at any given time and manipulate the underlying software. Countermeasures for fault injection attacks are under constant research [19], but they are rarely implemented in devices that are not designed to be tamper-proof. We discovered that hardware faults which cause the initialization process to fail can cause the system to fall back into a highly-privileged shell process. This can be done by disconnecting or shorting various hardware components. For example, shorting the GND and MISO pins of an SPI Flash module will cause any reads from the device to be malformed. Of course, this procedure carries the risk of damaging the device or its memory.

While side-channel attacks can also be used for recovering passwords [15], they tend to be better suited to systems with a simpler design such as ASICs or FPGAs. They are more difficult in our black-box scenario which includes a fully-featured multitasking operating system. Many other physical attacks exist for the determined researcher, some of which are even effective against tamper-resistant devices [7], but none of the devices we investigated required these methods.

Uploading additional tools into the device Embedded systems are often designed with the minimal set of features and components required for their task, as such, their software is designed similarly. Embedded Linux may contain only a small subset of the Linux utilities and features that desktop Linux users are used to having. BusyBox [38] provides many known Linux utilities in reduced size and pre-compiled for many common architectures. Using common utilities such as FTP (File Transfer Protocol utility), TFTP (Tiny File Transfer Protocol utility), Wget or NetCat can mediate data and file transfer to and from the device and over the network.

When network utilities are unavailable, data can be infiltrated through crude methods such as scripting the use of the Unix Bash Echo command for writing binary data into files. A simple python script that uses Echo for transferring files over UART is publicly available in the authors' github repository [8].

Obtaining the firmware Extracting a copy of the firmware and file system is an important stage for reverse engineering since analysis of the firmware can reveal secrets and vulnerabilities. Firmware analysis is further discussed in Section 4.

When network connection and console access are available, Flash memory MTD (Memory Technology Device) partitions can be streamed into NetCat and sent to a remote computer. A copy of the file system may also be compressed using the Tar utility and streamed using NetCat. Doing so will eliminate the need for unpacking the file system, which is not always a trivial task.

If a network connection is unavilable, memory contents can be read over UART from bootloader or Linux console. Bootloaders consoles often contain

memory read/write/display primitives and can be used to slowly dump an image of the memory into the UART console. A script on the receiving end can convert the hexadecimal-displayed data into binary format; such script is publicly available in the authors' github repository [8].

When the bootloader and Linux console are inaccessible, flash memory contents can be dumped via out-of-band methods. There are several ways in which the researcher can gain access to partial or complete data belonging to the device's memory. A minimally intrusive option is connecting a logic analyzer to the pins of the memory module and recording the signals while the device is booting up. Partial memory images can be extracted from the communications on the memory bus, depending on the actual addresses that were accessed during the recording. A simple script can convert the logic analyzer output to usable binary, such script is publicly available in the authors' github repository [8].

In order to gain a full and accurate image of the device memory, it is possible to desolder the memory chip and connect it to off-the-shelf memory readers such as the CH341A. If the memory module is not compatible with off-the-shelf readers, a custom reader can be built using a general purpose USB adapter such as FT2232H or a programmable micro-controller.

More advanced techniques have been proposed [13] but are outside the scope of this paper due to their costs and effort requirements.

2.3 Analyzing the Firmware

Unpacking memory images Once a memory image had been obtained, it is necessary to unpack it in order to view the data it holds. The community-maintained Binwalk utility has the ability to unpack and extract most common embedded file systems, and even some proprietary file systems. When used with the '-Z' argument, Binwalk detects raw compression streams that may be hidden from default scans and is able to extract them. A collection of utilities named firmware-mod-kit [2] contains several file formats and variations that Binwalk does not support.

Brute-forcing passwords One of the more interesting feats of reverse engineering is password extraction. Native Linux passwords are used by default over SSH (Secure Shell) and Telnet (Telecommunication Network) connections and in cases also for other services such as HTTP and FTP. An observation about the Mirai IoT malware is that the infection method was connecting to IoT devices over SSH/Telnet with default credentials. Many devices today have credentials that are not as trivial as 'root', 'admin' or '123456' but are still not complex enough to withstand exhaustive password search.

Linux user passwords are usually stored in the special file '/etc/passwd' or its companion '/etc/shadow' in a hashed format, using the crypt(3) [1] utility. The password hash files can be read freely by users with sufficient credentials and can also be extracted from the firmware.

crypt(3) supports several hashing algorithms, but two are the most observed in IoT devices: **Descrypt** - A DES (Data Encryption Standard) based password

hashing algorithm. A modern high-end GPU (Graphical processing using) is capable of calculating over 9*10^8 descrypt hashes per second. **Md5crypt** - An MD5 (Message-Digest Algorithm 5) based password hashing algorithm. A modern high-end GPU is capable of calculating over 10^6 (Ten million) md5crypt hashes per second.

While simple passwords can be recovered using a generic password recovery tools such as John the Ripper [4], advanced password cracking can be done with Hashcat [3]. Hashcat supports advanced rules and patterns and is designed for GPU hashing. The usage of Hashcat requires more knowledge than using John the Ripper ans it is widely used for recovery of difficult passwords.

In order to perform efficient password cracking, a word-list or pattern file is required. Many patterns and word-lists are available online but none had proved effective enough against hard to guess IoT device passwords. A few observations by the authors about known and newly discovered passwords allowed the creation and sorting of a password pattern list that proved effective against IoT device passwords. The pattern generation rules consist of: up to two symbol characters; up to two three uppercase characters; any amount of digits and lowercase characters; up to 8 characters total.

Another observation was that many elements of password difficulty inversely correlates with password selection. For example: symbol characters are expensive to search and used less often than other characters; digits are easy to search and are widely used; uppercase characters are used less than lowercase characters. This allowed sorting the pattern list according to increasing difficulty levels while expecting to guess passwords in the early stages of testing the list. More on the results of password cracking in Section 3. A python script for generating and sorting the pattern list is publicly available in the authors' github repository [8].

Detecting vulnerabilities within the firmware As firmware images contain the operating system and code controlling the device behavior, further analysis may expose underlying vulnerabilities. While in-depth reverse engineering techniques of the firmware are beyond the scope of this paper, there are many previous researches done in this field [30,25,12,11,26].

3 Results

3.1 Devices Under Inspection

Table 1 describes 16 IoT devices that were subjected to reverse engineering. As shown in the Table, the survey included devices from many different vendors and with prices which varied by an order of magnitude. Most of the devices with the properties selected for this work contain cameras. Additionally there are two smart doorbells that are capable of streaming video, audio, initiating VOIP sessions and also opening an entry door or a gate. A smart thermostat was also analyzed. This device can control an entire household's HVAC systems. A list of all of the devices and their properties can be seen in Table 1. All of the devices contained the embedded Linux operating system.

Table 1: List of devices reverse engineered

Device ID	Device Type	Manufacturer	Model	Video recording	Additional Capabilities	Price (USD)
1	IP Camera	Xtreamer	Cloud Camera	Yes	None	84
2	IP Camera	Simple Home	XCS7_1001	Yes	None	54
3	IP Camera	Simple Home	XCS7_1002	Yes	None	47
4	IP Camera	Simple Home	XCS7_1003	Yes	None	142
5	IP Camera	Foscam	FI9816P	Yes	None	70
6	IP Camera	Foscam	C1	Yes	None	58
7	IP Camera	Samsung	SNH-1011N	Yes	None	68
8	IP Camera	Xiaomi	YI Dome	Yes	None	40
9	IP Camera	Provision	PT-838	Yes	None	163
10	IP Camera	Provision	PT-737E	Yes	None	102
11	IP Camera	TP-Link	NC250	Yes	None	70
12	Baby Monitor	Phillips	B120N	Yes	None	46
13	Baby Monitor	Motorola	FOCUS86T	Yes	None	145
14	Doorbell	Danmini	WiFi Doorbell	Yes	Open door/gate	80
15	Doorbell	Ennio	SYWIFI002	Yes	Open door/gate	119
16	Thermostat	Ecobee	3 (golden firmware)	No	HVAC control	170

3.2 Techniques Used on Devices

Table 4 shows a sample of the devices inspected along with the properties that allow or hinder reverse engineering. In the table there are also the techniques shown effective against these devices.

3.3 Discoveries Made During the Evaluation

Login credentials One of the most significant steps of reverse engineering an IoT device is to identifying all of the user accounts within the device. Every device contains at least one effective account which is the root account. The root account is the most privileged account on a Unix system. The root account has the ability to carry out all facets of system administration, including adding accounts, changing user passwords, accessing the file system, and installing software. Once a hashed password is recovered and its underlying plaintext password revealed, the ability of logging into the device with root user privileges is achieved. As can be seen in Table 2, eight of the devices contained password hashed with the descrypt algorithm, while the other eight devices employed md5crypt. The selection of hashing algorithm is critical for resisting password cracking, descrypt hashing can be as much as ninety times faster than md5crypt, as described in subsection 2.3.

Table 2: Discovered device properties

Device ID	Similar products	Password Hash Type	Open Services for Remote Access	Password complexity	Contains Private Keys
1	Closeli Simplicam	descrypt	-	Medium	Yes
2	-	md5crypt	Telnet	Very Low	-
3	-	descrypt	Telnet	Low	-
4	Tenvis TH692	md5crypt	Telnet	Low	-
5	-	md5crypt	FTP	Unknown	Yes
6	-	md5crypt	FTP	Unknown	-
7	-	md5crypt	-	Unknown	-
8	-	md5crypt	-	None	-
9	VStarcam D38	descrypt	-	Low	-
10	VStarcam C23S	descrypt	Telnet	Low	-
11	-	md5crypt	-	Very Low	-
12	-	descrypt	SSH	Medium	Yes
13	-	md5crypt	-	Unknown	-
14	-	descrypt	Telnet	Very Low	-
15	-	descrypt	Telnet	Very Low	-
16	-	descrypt	-	Low	-

The pattern based password recovery described in Subsection 2.3 was used against all of the extracted password hashes. Figure 5 in Appendix shows the theoretical duration of password recovery using the proposed 48,820 patterns that cover all of the password possibilities previously mentioned. The patterns were sorted in order of rising complexity. For example, the pattern for six consecutive digits contains 1,000,000 possibilities and was sorted before the pattern for five consecutive English characters that has 11,881,376 possibilities. As the figure shows, most observed non-empty passwords were recovered within the first 5,000 patterns, after testing only 5.22e+11 passwords. The theoretical bound for testing that many passwords on a strong GPU server is 2.4 minutes for descrypt hashes and 217 minutes for md5crypt. Actual password recovery can have significant overheads over the theoretical bounds.

Eleven non-empty passwords were recovered, one device contained an empty password. Four passwords were not yet recovered to the time of writing this paper and are expected to be revealed within several weeks. Table 2 shows password complexities that varied between very low complexity (e.g. “abcd”) to medium complexity (e.g. “AbC123de”), undiscovered passwords were given the complexity rating “Unknown”. All the discovered passwords were verified as the credentials in multiple devices of the same model. Two devices made by the same manufacturer were discovered to have the same passwords but different hash values due to random salt.

Remote access A simple port scan using Nmap [18] revealed that many of the tested devices have administration services bound to open ports such as SSH or Telnet, which allows a remote access. Remote access allows a user to log-in to a device as an authorized user without being in the proximity of the device, depending on the network topology. Six of the devices maintain a Telnet service, one device has an accessible SSH port and two devices allow communication to open FTP ports as can be seen in Table 2.Although some of the devices do not allow communication through an administration port, by accessing the UART console it is possible to set up network services performing the desired functions.

WiFi credentials IoT devices must be connected to the internet in order to function properly. In order to maintain wireless connection persistency across reboots and power shortages, a configuration file that holds the WiFi credentials is located in all of the tested devices. The configuration file is located in the mounted file system, usually under the "config" or the "NetworkManager" paths, and contains all of the WiFi settings, including the SSID (Service Set Identifier) and non-encrypted password. Retrieval of the correct file from an extracted file system can be done simply by searching for relevant keywords.

Embedded private keys A private key is an object that is used by an encryption algorithm for encrypting and decrypting messages and plays an important role in asymmetric cryptography. In three of the devices a hard-coded private key used for secure communication were found, as shown in Table 2. With the private key exposed, secure communication may be rendered insecure and exposed to violations such as man-in-the-middle attack.

Rebranded devices In the IoT market, a rebranded device is one where the internal design, architecture and file system are purchased from one manufacturer, and cosmetic modifications link the device to a new brand and manufacturer. Identifying rebranded devices means that discovered private keys, hashed passwords, account credentials and even the application vulnerabilities may be identical across several devices. Four devices inspected were found to share a non-trivial password or hashed password with products from different manufactures, strongly implying a similarity between them. The devices were found using a simple web search for the passwords and hashes and encountering forum posts that specify hashes and passwords of other devices.

4 Analysis

The techniques that were shown in Section 2 may be used for both malicious and benign activities. This section serves to demonstrate and discuss some of the possibilities that emerge from making the reverse engineering process more generic and streamlined, and considering the results seen in the last section.

4.1 Extension of Existing Attacks into New Platforms

Creation of a personalized Mirai botnet with increased capabilities The infamous Mirai botnet had gained publicity after it was used against several online web sites. After witnessing a large-scale DDoS (Distributed Denial of Service) attack on KrebsOnSecurity.com, Martin McKeay, Akamai's senior security advocate was quoted saying "Someone has a botnet with capabilities we haven't seen before. We looked at the traffic coming from the attacking systems, and they weren't just from one region of the world or from a small subset of networks — they were everywhere." [23]. Mirai malware infects IoT devices with an open Telnet port and default login credentials and add them to the attacker's botnet army. The source code for Mirai was leaked to the internet on and can be modified and used by anyone who desires [5]. By using the reverse engineering process we were able to extract new and previously unknown Telnet and SSH credentials belonging to several IoT devices that were never a part of the Mirai botnet. In order to create a customized version of the Mirai botnet, the source code was modified by adding the new passwords to the malware's source code. After building an isolated network and infecting it with the modified Mirai botnet. The bot activity over the network was monitored and the infection could be seen spreading to the IoT devices that were added to the network.

An interesting example case is that after extracting the login credentials of the ProVision PT-838 security camera, the modified botnet was able to successfully connect to the ProVision PT-737E security camera due to the shared credentials between the cameras of the same manufacturer. The aforementioned process allows the number of devices that are vulnerable to Mirai to be extended.

Remote access to IoT devices by unauthorized parties Remote connection to an IoT device, via Telnet or SSH, can be performed not only by malware but also be used as an easy and quick way for an attacker to gain control over a device remotely. The Philips In.Sight Wireless HD Baby Monitor (B120N/10) was designed to allow parents to watch, listen and talk to their newborn [33]. During the reverse engineering process several critical engineering faults that allows an outsider to use this device were discovered. Credentials were revealed that allows anyone to connect through the open SSH port in all Philips In.Sight B120N monitors. Additionally, SSL private keys that allow an attacker to perform a man-in-the-middle attacks on device communication were discovered. Furthermore, as shown in the previous section, after gaining access to an IoT device the attacker can extract sensitive information about the device and its owner such as WiFi credentials.

Execution of arbitrary code on IoT devices During the reverse engineering process, software is often uploaded into the device in various ways. The ability to upload software and even have it maintain persistency after restarts has a great implication on device security. Since it was shown how to gain complete device control when physical access is available, physical access to a device can be used to modify the device's behavior even after the device is no longer in proximity.

4.2 Possible Theoretical Attacks

Discovery of new vulnerabilities By using the black-box reverse engineering process, an attacker with the possession of an unknown device (e.g. a security camera with no identification markings printed on it) that was obtained from a public area may extract crucial or sensitive information. While analyzing the results we found out that many IoT devices had old OS or firmware version that was outdated, when many issues were fixed in later versions. After learning about the firmware or OS version, the attacker can search the internet for known vulnerabilities or even find this information in the release notes of more updated versions. Furthermore, after obtaining the firmware the attacker can scan the software for security holes using static analysis methods [14,12].

Extraction of secrets from outdoor IoT devices Many IoT devices are marketed for an outdoor installation (e.g. security cameras, smart doorbells etc.). These products are mounted outside or in large halls and can be accessed by strangers. For example, the Ennio Doorbell (SYWIFI002) contains a camera, microphone and speaker in order to monitor and control an entrance and can also be wired to a door or a gate for remote unlocking. The doorbell is usually installed outside and may be accessed by a stranger. A direct result of the device's accessibility is the ability of an attacker to physically modify sabotage the device. However, it is not just the device that may be affected, secrets may be extracted from the device giving the attacker access to the whole network.

Supply chain attacks Malicious activity can also be performed as a part of the supply chain. An untrustworthy seller or courier can reverse engineer a device without having any previous knowledge about the it and perform modifications to the device. The recipient of an IoT device may use it without knowing it was tampered with, perhaps even equipped with a backdoor, or some other malware.

4.3 Constructive Uses to the Reverse Engineering Process

There are uses of reverse engineering that can benefit the owner. Lower-end products are often received with insufficient information about the hardware or software inside. A concerned customer can use the described process and discover properties of the device she bought. If the device is rebranded she could search the internet for the similar devices by well other vendors. The consumer gains the ability to learn about the device's vulnerabilities and perhaps even upgrade the firmware and secure the device. This process can be performed on many types of IoT devices and may also assist with products that no longer have support.

Learning about the device's software and hardware can not only help the customer identify their product, but also allows her to customize it to her own needs. After gathering the desired information the owner can manipulate the firmware or configuration. She can also develop her own system that will operate the device and even add missing functionality. Modification of stock devices can also be used to hinder censorship and other information blocking instruments.

5 Discussion

The IoT market is evolving and so does the competition among the vendors for being the first to create better and cheaper devices. This pressure may affect the product's design and lead to devices with critical security issues being released. Time is not the only obstacle for creating a secure product; as competition drives the prices down, the production process must be become cheaper. Employing penetration testers and security analysts may be very expensive, while the hardware engineers that built the product might lack knowledge of cyber security. This trade-off between money and security is usually inclined towards cheaper but less safe products. The reverse engineering process empowers consumers and researchers with abilities to discover important details about devices available in the market and benchmark their security.

5.1 Recommendations for Implementers

The results and analysis shown in this paper support several recommendations for better securing IoT devices.

1. Disable UART ports or remove their terminals from the board design. If a UART port is required, it can be set up as read-only.
2. If a UART port is required and must be write-enabled, protect UART ports in a similar fashion to JTAG protection [34].
3. Use unique strong passwords for every single device hashed with a strong hashing algorithm. Passwords must be user replaceable in a convenient way.
4. If possible, encrypt all of the device's writable memory. Otherwise, encrypt all sensitive data stored on the device.

5.2 Conclusion

The increase in IoT technology popularity holds many benefits but on the other hand this surge of new, innovative and cheap devices reveals complex security and privacy challenges. Vulnerabilities and design flaws in innocent IoT devices are an opening for an adversary to exploit and misuse. As shown in Section 4, an attacker that gains remote or physical access to an IoT device may snoop on the owner's personal or sensitive information and even use the device's capabilities for her own desire. The evolution of cyber crime didn't pass over the IoT and in the last years we are witnessing new types of cyber attacks that involve IoT devices. Accessibility of the black-box reverse engineering process may accelerate the attacker's work and introduce new IoT cyber threats.

References

1. crypt(3) man page. Linux Programmer's Manual `http://man7.org/linux/man-pages/man3/crypt.3.html`

2. Firmware-mod-kit github repository https://github.com/mirror/firmware-mod-kit
3. Hashcat password recovery tool https://hashcat.net/
4. John the ripper password cracker http://www.openwall.com/john/
5. Mirai github repository https://github.com/jgamblin/Mirai-Source-Code
6. Alqassem, I., Svetinovic, D.: A taxonomy of security and privacy requirements for the internet of things (iot). In: 2014 IEEE International Conference on Industrial Engineering and Engineering Management, IEEM 2014, Selangor Darul Ehsan, Malaysia, December 9-12, 2014. pp. 1244–1248. IEEE (2014), https://doi.org/10.1109/IEEM.2014.7058837
7. Anderson, R.J., Kuhn, M.G.: Low cost attacks on tamper resistant devices. In: Christianson, B., Crispo, B., Lomas, T.M.A., Roe, M. (eds.) Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1361, pp. 125–136. Springer (1997), https://doi.org/10.1007/BFb0028165
8. Anonymous: The author's github repository. details omitted for anonymous submission (2017)
9. Atmel Corporation: ATtiny13A Datasheet (May 2012), http://www.atmel.com/images/doc8126.pdf
10. Bodenheim, R., Butts, J., Dunlap, S., Mullins, B.E.: Evaluation of the ability of the shodan search engine to identify internet-facing industrial control devices. IJCIP 7(2), 114–123 (2014), https://doi.org/10.1016/j.ijcip.2014.03.001
11. Chen, D.D., Woo, M., Brumley, D., Egele, M.: Towards automated dynamic analysis for linux-based embedded firmware. In: NDSS (2016)
12. Costin, A., Zaddach, J., Francillon, A., Balzarotti, D.: A large-scale analysis of the security of embedded firmwares. In: Fu, K., Jung, J. (eds.) Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014. pp. 95–110. USENIX Association (2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin
13. Courbon, F., Skorobogatov, S., Woods, C.: Reverse engineering flash EEPROM memories using scanning electron microscopy. In: Lemke-Rust, K., Tunstall, M. (eds.) Smart Card Research and Advanced Applications - 15th International Conference, CARDIS 2016, Cannes, France, November 7-9, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10146, pp. 57–72. Springer (2016), https://doi.org/10.1007/978-3-319-54669-8_4
14. Cui, A., Costello, M., Stolfo, S.J.: When firmware modifications attack: A case study of embedded exploitation. In: 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013. The Internet Society (2013), http://internetsociety.org/doc/when-firmware-modifications-attack-case-study-embedded-exploitation
15. DaRolt, J., Das, A., Natale, G.D., Flottes, M., Rouzeyre, B., Verbauwhede, I.: Test versus security: Past and present. IEEE Trans. Emerging Topics Comput. 2(1), 50–62 (2014), https://doi.org/10.1109/TETC.2014.2304492
16. Davis, R., Merriam, N., Tracey, N.: How embedded applications using an rtos can stay within on-chip memory limits. In: 12th EuroMicro Conference on Real-Time Systems. pp. 71–77 (2000)
17. Gartner: Gartner says 4.9 billion connected "things" will be in use in 2015. Gartner.com (2014), http://www.gartner.com/newsroom/id/2905717
18. Gordon Lyon: Nmap security scanner https://nmap.org/

19. Goubet, L., Heydemann, K., Encrenaz, E., Keulenaer, R.D.: Efficient design and evaluation of countermeasures against fault attacks using formal verification. In: Homma, N., Medwed, M. (eds.) Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers. Lecture Notes in Computer Science, vol. 9514, pp. 177–192. Springer (2015), https://doi.org/10.1007/978-3-319-31271-2_11
20. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future Generation Comp. Syst. 29(7), 1645–1660 (2013), https://doi.org/10.1016/j.future.2013.01.010
21. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: cold-boot attacks on encryption keys. Commun. ACM 52(5), 91–98 (2009), http://doi.acm.org/10.1145/1506409.1506429
22. Hollabaugh, C.: Embedded Linux : hardware, software, and interfacing. Addison-Wesley, Boston (2002)
23. Krebs, B.: Krebsonsecurity hit with record ddos https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/
24. Lanet, J., Bouffard, G., Lamrani, R., Chakra, R., Mestiri, A., Monsif, M., Fandi, A.: Memory forensics of a java card dump. In: Joye, M., Moradi, A. (eds.) Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8968, pp. 3–17. Springer (2014), https://doi.org/10.1007/978-3-319-16763-3_1
25. Ling, Z., Luo, J., Xu, Y., Gao, C., Wu, K., Fu, X.: Security vulnerabilities of internet of things: A case study of the smart plug system. IEEE Internet of Things Journal (2017)
26. Liu, M., Zhang, Y., Li, J., Shu, J., Gu, D.: Security analysis of vendor customized code in firmware of embedded device. In: International Conference on Security and Privacy in Communication Systems. pp. 722–739. Springer (2016)
27. Lund, D., MacGillivray, C., Turner, V., Morales, M.: Worldwide and regional internet of things (iot) 2014–2020 forecast: A virtuous circle of proven value and demand. International Data Corporation (IDC), Tech. Rep (2014)
28. Mahmoud, R., Yousuf, T., Aloul, F.A., Zualkernan, I.A.: Internet of things (iot) security: Current status, challenges and prospective measures. In: 10th International Conference for Internet Technology and Secured Transactions, ICITST 2015, London, United Kingdom, December 14-16, 2015. pp. 336–341. IEEE (2015), https://doi.org/10.1109/ICITST.2015.7412116
29. Nest Labs: Nest learning smart thermostat https://nest.com/thermostat/meet-nest-thermostat/
30. Obermaier, J., Hutle, M.: Analyzing the security and privacy of cloud-based video surveillance systems. In: Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security. pp. 22–28. ACM (2016)
31. Patton, M.W., Gross, E., Chinn, R., Forbis, S., Walker, L., Chen, H.: Uninvited connections: A study of vulnerable devices on the internet of things (iot). In: IEEE Joint Intelligence and Security Informatics Conference, JISIC 2014, The Hague, The Netherlands, 24-26 September, 2014. pp. 232–235. IEEE (2014), https://doi.org/10.1109/JISIC.2014.43
32. Pedro, M.S., Soos, M., Guilley, S.: FIRE: fault injection for reverse engineering. In: Ardagna, C.A., Zhou, J. (eds.) Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication - 5th IFIP WG 11.2

International Workshop, WISTP 2011, Heraklion, Crete, Greece, June 1-3, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6633, pp. 280–293. Springer (2011), https://doi.org/10.1007/978-3-642-21040-2_20
33. Philips: Philips in.sight wireless hd baby monitor http://www.philips.co.uk/c-p/B120N_10/in.sight-wireless-hd-baby-monitor/overview
34. Rosenfeld, K., Karri, R.: Attacks and defenses for JTAG. IEEE Design & Test of Computers 27(1), 36–47 (2010), https://doi.org/10.1109/MDT.2010.9
35. Shodan: Shodan is the world's first search engine for internet-connected devices https://www.shodan.io/
36. Sicari, S., Rizzardi, A., Grieco, L.A., Coen-Porisini, A.: Security, privacy and trust in internet of things: The road ahead. Computer Networks 76, 146–164 (2015), https://doi.org/10.1016/j.comnet.2014.11.008
37. Tellez, M., El-Tawab, S., Heydari, H.M.: Improving the security of wireless sensor networks in an iot environmental monitoring system. In: Systems and Information Engineering Design Symposium (SIEDS), 2016 IEEE. pp. 72–77. IEEE (2016)
38. Vlasenko, D.: Busybox: The swiss army knife of embedded linux https://busybox.net/
39. Yu, T., Sekar, V., Seshan, S., Agarwal, Y., Xu, C.: Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In: de Oliveira, J., Smith, J., Argyraki, K.J., Levis, P. (eds.) Proceedings of the 14th ACM Workshop on Hot Topics in Networks, Philadelphia, PA, USA, November 16 - 17, 2015. pp. 5:1–5:7. ACM (2015), http://doi.acm.org/10.1145/2834050.2834095
40. Zhang, Z., Cho, M.C.Y., Wang, C., Hsu, C., Chen, C.K., Shieh, S.: Iot security: Ongoing challenges and research opportunities. In: 7th IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2014, Matsue, Japan, November 17-19, 2014. pp. 230–234. IEEE Computer Society (2014), https://doi.org/10.1109/SOCA.2014.58

Appendix

Table 3: A list of hardware and software tools used

1	Screwdrivers and plastic spudgers including common and uncommon drive bits such as Philips, Torx, Security Torx and various star configurations.
2	BK 2712 Multimeter.
3	FTDI FT232R USB UART interface module.
4	Saleae Logic Pro 8 logic analyzer with the Logic 1.2.12 software.
5	CH341A USB EEPROM and Flash memory programmer module with software version 1.29.
6	Intel i7-4790 desktop PC running Windows 10 operating system and Ubuntu 16.04.4 on a virtual machine.
7	Intel i7-6900K server with four Titan X (Pascal) Nvidia GPUs running Ubuntu 16.04.2 LTS operating system with Nvidia driver version 375.66.
8	John The Ripper 1.8.0 CPU password cracking software.
9	Hashcat 3.6.0 multiple architecture password recovery software.
10	Binwalk Firmware Analysis Tool - latest version pulled from github repository on 30/07/2017 and compiled locally, including all dependencies.
11	firmware-mod-kit - latest version pulled from github repository on 30/07/2017 and compiled locally

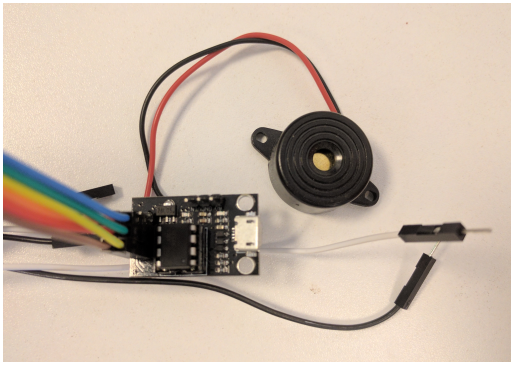
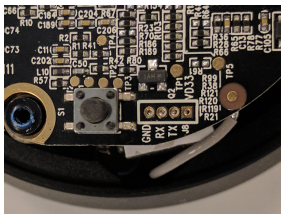


Fig. 3: UART discovery assistant module

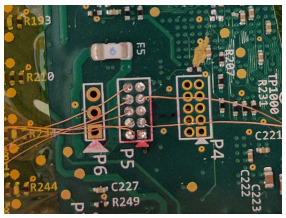
Table 4: Inspected devices and the techniques effective on them

Device ID	UART location*	Bootloader password	Terminal password	Terminal password bypass technique	Data extraction technique
2	Marked pads	No	Yes	Shorted memory caused fallback	Used Wget to download NetCat
5	Unmarked pads*	No	No	-	Physically read the the on-board flash
8	Unmarked pads*	No	No	-	Used "echo" to transfer NetCat over UART
10	Unmarked pads*	Yes**	Yes	Set bootcmd in bootloader	Used NetCat
11	Unmarked pads*	No	Yes	Trivial password	Used Wget to download NetCat
12	Marked pads	No	Yes	Set bootcmd in bootloader	Used NetCat
15	Unmarked pads*	No	Yes	Trivial password	Used TFTP to download NetCat
16	Unmarked pads*	No	No	-	Used NetCat

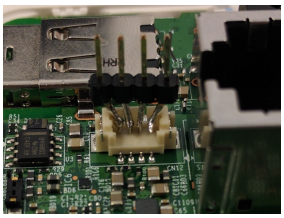
* Unmarked pads were discovered by inspection of the PCB assisted with the UART discovery assistant module 3.
** Bootloader password was recovered using a logic analyzer that sniffs communication on the memory bus.



(a) UART terminals with marking inside Xtreamer Cloud Camera



(b) Wires soldered to a header pads that includes UART connections inside the Ecobee 3 Smart Thermostat



(c) Male pin header soldered on top of UART socket inside Samsung SNH-1011N Smart Camera

Fig. 4: Examples of UART terminals

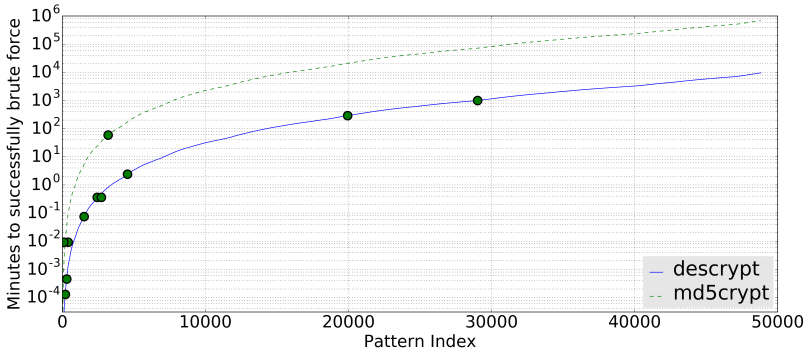


Fig. 5: Password recovery duration using the GPU server described in Table 3. Each marking on the graph is a successfully recovered password belonging to a device inspected.