Reverse Engineering IoT Devices: Effective Techniques and Methods

Omer Shwartz, *Student Member, IEEE*, Yael Mathov, Michael Bohadana, Yuval Elovici, and Yossi Oren, *Senior Member, IEEE*

***Abstract*—Recent Internet of Things (IoT) botnet attacks have called the attention to the fact that there are many vulnerable IoT devices connected to the Internet today. Some of these Web-connected devices lack even basic security practices such as strong password authentication. As a consequence, many IoT devices are already infected with malware and many more are vulnerable to exploitation. In this paper we analyze the security level of 16 popular IoT devices. We evaluate several low-cost black-box techniques for reverse engineering these devices, including software and fault injection-based techniques used to bypass 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 add these devices to a botnet. We also discuss how to improve the security of IoT devices without significantly increasing their cost or affecting their usability.**

***Index Terms*—Computer security, Internet of Things (IoT), IoT application design, IoT standardization, IoT system architecture, IoT test-bed, privacy, reverse engineering.**

I. Introduction

In THE past years, we have been witnessing a dramatic rise in the number of Internet connected devices and recently, wireless connected devices. 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. The number of IoT devices is constantly rising and estimated to reach 50 billion by 2020 [1].

Since the early days of computing, low-cost ubiquitous devices have largely been powered by simple microcontrollers and 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 design these devices around a fully featured operating system (OS), such as Linux, taking advantage of the existing code base and relative ease of development and debugging.

Manuscript received February 4, 2018; accepted October 2, 2018. Date of publication October 10, 2018; date of current version January 16, 2019. This work was supported by the Israel Science Foundation under Grant 702/16 and Grant 703/16. *(Corresponding author: Omer Shwartz.)*

The authors are with the Department of Software Information Systems Engineering, Ben-Gurion University of the Negev, Beersheba 8410501, Israel (e-mail: omershv@post.bgu.ac.il; yaelmath@post.bgu.ac.il; bohadana@post.bgu.ac.il; elovici@bgu.ac.il; yos@bgu.ac.il).

Digital Object Identifier 10.1109/JIOT.2018.2875240

The task of the device security engineer has also evolved with the move from application-specific integrated circuits (ASICs) and simple microcontrollers to complete Linux devices. Traditional hardware attack methods which target integrated circuits (ICs) are less effective today, 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 into the Web and residences and businesses was quickly followed by emerging security challenges [2]. The rapid growth in the number and variety of IoT devices created a scenario where millions of devices [3] are deployed yet consumers may know very little about the capabilities and security of the devices. This knowledge is especially crucial since IoT devices are often equipped with a wide array of sensors, connected to private networks and control a variety of physical systems, from entry gates and door locks to heating, ventilation and air conditioning (HVAC) systems [4]. The structure of a typical IoT device can be seen in Fig. 1.

In this paper we present a general methodology for "black-box" reverse engineering of complete stack IoT devices. The techniques presented address many use cases and can be used as a tutorial for gaining internal access to new devices. While most of the techniques we use are generally well known, to the best of our knowledge, this is 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 and make recommendations for more secure IoT design and implementation.

A. Contributions

This paper makes the following contributions.

1) We present a systematic reverse engineering workflow appropriate for full-stack OS IoT devices in a detailed and tutorial-like manner.
2) We apply this workflow to 16 IoT devices produced by different manufacturers.
3) We provide insight into the obstacles encountered and methods proven effective while evaluating the devices-under-test and discuss their common characteristics and security flaws.

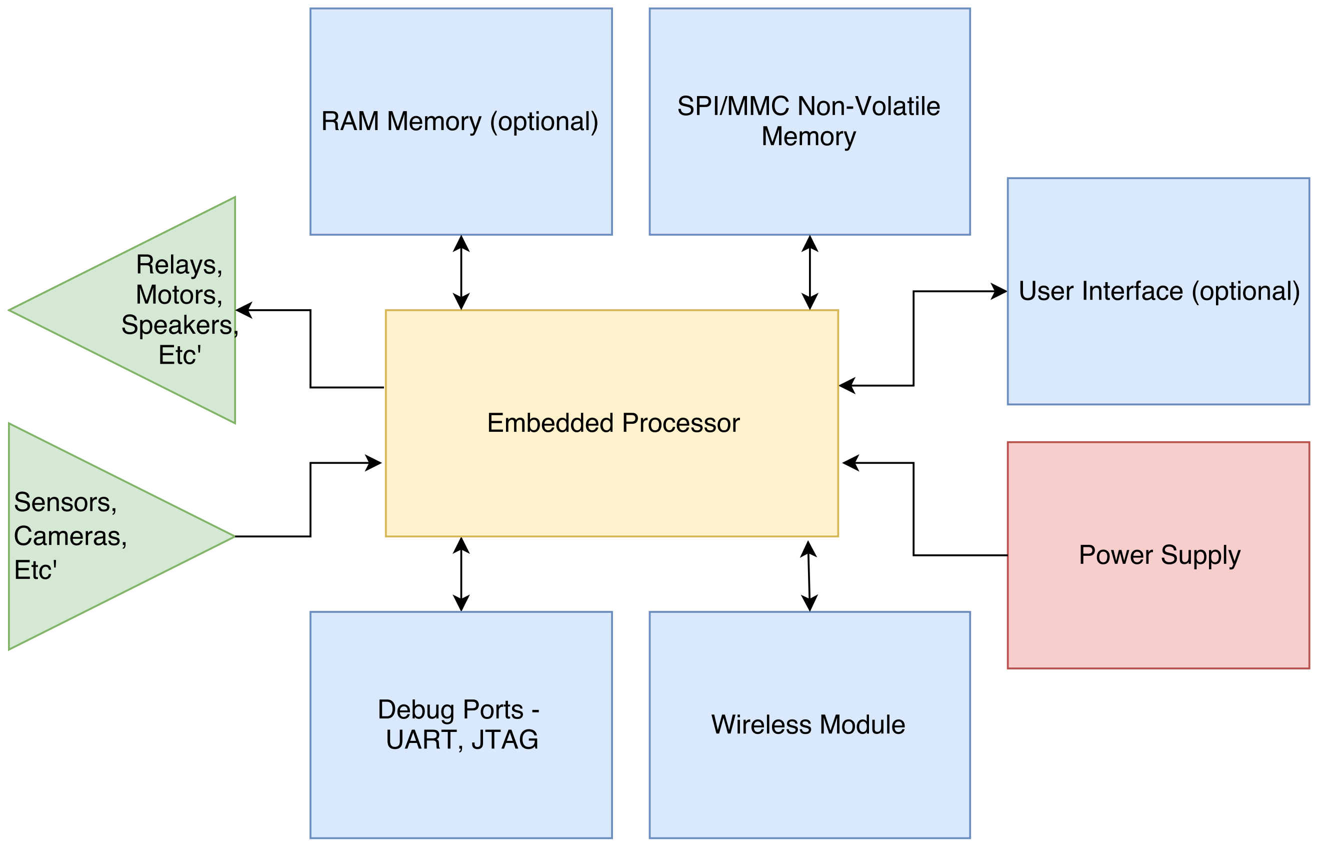


Fig. 1. Block diagram describing the structure of a typical IoT device.

4) We analyze some of the properties and vulnerabilities discovered to implement new attacks and suggest theoretical exploits.
5) We suggest some nonmalicious uses for the reverse engineering process that may benefit consumers.
6) We offer a list of recommendations for those interested in making these devices more secure.

B. Related Work

Mahmoud *et al.* [5] presented a survey of the current concerns in IoT security. The authors described the general architecture of IoT devices and the security challenges stemming from this design, corresponding to the security principles of confidentiality, integrity, availability, and authentication. Sicari *et al.* [6] claimed that the network communication characteristics of IoT devices, combined with the increase in exchanged information, multiply the potential for attacks that breach system privacy. Alqassem and Svetinovic [7], Zhang *et al.* [8], and Yang *et al.* [9] also raised similar concerns. Interestingly, most of this analysis centers on security threats to the user of the IoT device (i.e., loss of confidentiality and availability) and less on the risks to the device itself (e.g., counterfeiting).

Lin *et al.* [10] presented a broad and highly specific collection of definitions regarding IoT types and taxonomies. In addition to providing a long list of communication methods and standards they also described six security features of IoT devices and 18 different attacks which were categorized into three classes.

Patton *et al.* [11] studied the extent of vulnerabilities found in network-accessible IoT devices. They reviewed several network scanners and focused on Shodan [12], a network scanning project that yielded a publicly available search engine for Internet connected services. Using Shodan, the authors discovered many vulnerable IoT systems, including a large number of supervisory control and data acquisition systems. Similar techniques can also be found in the work of Bodenheim *et al.* [13].

Tellez *et al.* [14] focused on wireless sensor networks (WSNs) and elements of their security. In this paper, the authors investigated the MSP430 MCU. The bootstrap loader (BSL) password 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 suggested ways to design a secure-BSL that can improve the protection of MCUs.

Gubbi *et al.* [15] offered a comprehensive review of the WSN terrain, including the terminology associated with it. Halderman *et al.* [16] showed techniques for recovering secrets from dynamic random access memory modules by transferring the modules into a new machine while minimizing data decays. Several improvements that allow a memory module to remain unpowered for several minutes without losing most of the data and various effects that can be achieved by recovering secrets belonging to different algorithms or applications were also demonstrated in this paper.

Lanet *et al.* [17] showcased methods for reverse engineering Java memory cards' erasable programmable read-only memory (EEPROM) data. They described forensic methods which enable researchers 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 and Hutle [18] employed reverse engineering techniques on several wireless security cameras and showed how the cameras are vulnerable to remote attackers without physical access to the surroundings of the device. The authors presented various encryption and communication weaknesses that may allow an attacker to impersonate a camera and eavesdrop or sabotage its communication.

C. Embedded Device Software Architectures

Software architecture (which may include an OS) determines many of the properties and limitations of a device. We differentiate between three main types of software architectures that are present in embedded devices.

1) *Full Stack OS-Based Devices*—Devices that contain a modern OS, 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 [19], more and more low-cost devices are now based on Linux due to falling component costs and the ease of developing for this OS. In particular, many of the cameras we surveyed had a full stack Linux implementation.
2) *Partial Stack OS-Based Devices*—Devices with a specifically designed RTOS, such as VxWorks or vendor-provided OS implementation. These devices are generally specially crafted for their task [20], and tend to omit some of the features of full stack OSs. Some lower-end or single tasked IoT devices use this architecture, with the RTOS handling Wi-Fi and Web protocols, and additional vendor code responsible for gathering sensor data.
3) *Devices With No OS*—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 architecture, but tend to be more difficult to development, and therefore have a longer time-to-market.

Thus, far, all of the IoT attacks seen in the wild and known to the authors, have targeted *full stack OS devices,* as these are more generic and make use of various drivers and open source components that may have known vulnerabilities. For this reason, *full stack OS devices* were chosen as the target for reverse engineering in this paper, although we believe that *partial stack OS devices* have significant potential for security vulnerabilities as well.

TABLE I
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 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 the Windows 10 operating system and the Ubuntu 16.04.4 operating system on a virtual machine.
7	Intel i7-6900K server with four Titan X (Pascal) Nvidia GPUs running the 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 - the latest version pulled from GitHub repository on 30/07/2017 and compiled locally

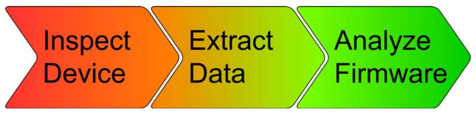


Fig. 2. Building blocks of black-box reverse engineering.

II. REVERSE ENGINEERING METHODOLOGY

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

Our black-box reverse engineering process follows a standard workflow that can be seen in Fig. 2.

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 it.

A. Inspection of the Device

Electronic devices are usually held together by screws or plastic clips. Most of the devices can be opened without damaging the exterior of the device or the internal components. Some device manufacturers also use glue during assembly, thus making opening the device more difficult.

1) Locating and Identifying Memory Components: Smart devices that run the Linux OS require sufficient nonvolatile memory to store the kernel and other mandatory file system components. A cheap and efficient way for engineering such devices is to place the memory module outside of the main processor package. Devices engineered in such configurations usually employ a processor that is capable of loading and

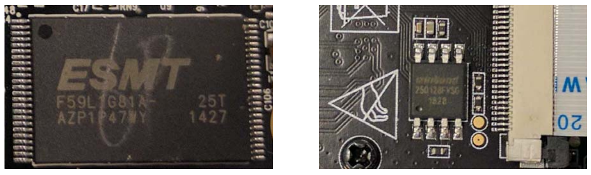


Fig. 3. Examples of onboard memory. (a) F59L1G81A 1 GB NAND Flash module inside Xtreamer Cloud Camera. (b) W25Q128FVSG 16MB SPI Flash module inside Ennio SYWi-Fi002 wireless doorbell.

running instructions directly from the serial parallel interface (SPI) Flash memory or EEPROM devices.

Understanding the memory technology is crucial for performing firmware extraction when there is no possibility of running commands on the tested device (see additional details in Section II-B).

A memory module that uses technology consistent with the required capacity is often found in devices. Common examples are: 25XX\26XX series eight-pin SPI Flash memory with up to 32 MB of storage space; larger SPI Flash devices with 16 or 32 pins; NAND Flash devices that come in various capacities and shapes and are usually coupled with a 24XX EEPROM module for holding initial configuration; embedded multimedia controller modules or cards usually containing more than a gigabyte of data. Examples of memory modules can be seen in Fig. 3.

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

2) Locating UART Terminals: Universal asynchronous receiver/transmitter (UART) ports are embedded universal communication channels with many purposes. Within 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 standard protocol at a predetermined baud rate. Typical baud rates for UART communication with embedded devices are 9600, 57 000, and 115 200 bits/s.

In many cases, UART terminals are embedded into the printed circuit board (PCB) 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 visible and accessible locations, occasionally marked with their purpose. In other cases the terminals are hidden (intentionally or unintentionally) between 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 benefit of reverse engineering.

Basic UART communication requires only three electrical lines: 1) transmit (TX); 2) receive (RX); and 3) ground (GND). A typical UART terminal has two to four exposed copper pads aligned in a row; when there are two pads, the TX pad is pulled electrically toward +1.8 V, +3.3 V, or +5 V and the RX pad might not be pulled to either directions; when there are three pads, the additional pad is usually the GND pad which should have continuity to the ground plane of the PCB; when there are four pads, the last pad is generally a VCC (positive voltage supply) which shows up as +1.8 V, +3.3 V, or +5 V when powered on.

By using the typical 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 Fig. 4.

3) UART Discovery Assistant Module: In order to assist in 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 [21] programmable micro-controller along with auxiliary electronics with custom code that switches between three popular UART baud rates, and it beeps when encountering a specified number of English printable ASCII characters (characters with an ASCII denotation larger than 0×20 and smaller than $0 \times 7F$). The device can be seen in Fig. 5. The source code for the module is publicly available in the authors' GitHub repository [22].

B. Extraction of Firmware and Data

Retrieval of data from within the device is a key component of reverse engineering. This section describes practices that facilitate data access, acquisition, and transfer from the device.

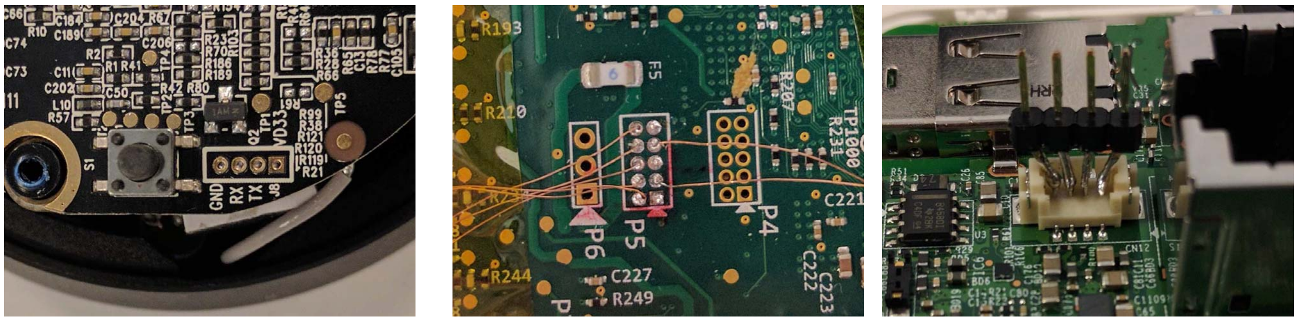
1) Handling Bootloader and Linux Passwords: While booting, the bootloader loads the kernel and passes the boot arguments to the kernel. Typically, the path for a user mode process that starts when the kernel completes booting is within the boot argument.

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 to the login or shell process. When the login process starts, it requests and verifies the user's credentials and spawns a shell process for the user to control. The login process is protected from brute force attempts and imposes a delay between consecutive password guessing attempts.

During reverse engineering, when encountering 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 in gaining access to the system. This change can be done from within the bootloader terminal, which can be accessed when the boot process begins.

Access to the bootloader is usually accomplished by pressing some key during the initial booting stages. In some cases the bootloader is protected by a 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. Such hard-coded password strings may be recovered from memory blobs obtained via out-of-band methods.

2) Using Physical Attacks to Bypass Passwords or Recover Passwords: Fault injections have a significant role in reverse engineering [23]. The use of fault injections allows the



(a) (b) (c)

Fig. 4. Examples of UART terminals. (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.

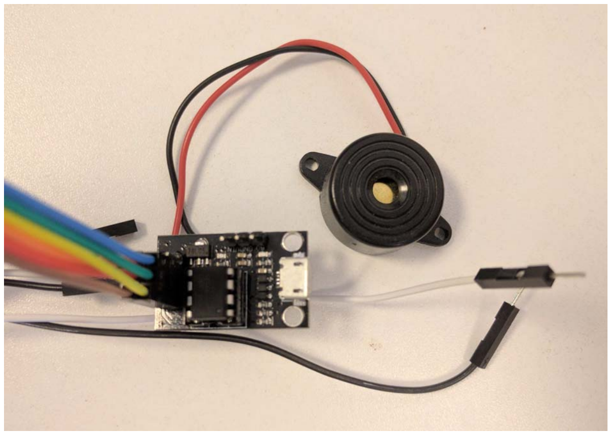


Fig. 5. UART discovery assistant module.

researcher to generate a hardware fault at any given time and manipulate the underlying software. Countermeasures for fault injection attacks constantly being investigated [24], 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 master in slave out (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 [25], they tend to be better suited to systems with a simpler design, such as ASICs or field-programmable gate arrays. They are more difficult to implement in our black-box scenario which includes a fully featured multitasking OS. Many other types of physical attacks exist for the determined researcher, some of which are even effective against tamper resistant devices [26], but none of the devices we investigated had properties that required these methods.

3) Uploading Additional Tools Onto the Device: Embedded systems are often designed with the minimal set of features and components required for their designated task, and their software design reflects this. Embedded Linux may contain only a small subset of the Linux utilities and features that desktop Linux users are used to having. BusyBox [27] provides many known Linux utilities in reduced size and precompiled for many common architectures. Using common utilities, such as file transfer protocol utility (FTP), trivial FTP utility, Wget, and NetCat can be used to 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 [22].

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

When network connection and console access are available, flash memory 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 unavailable, memory contents can be read over UART from a 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 [22].

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 to connect a logic analyzer to the pins of the memory module in order to record 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, and such script is publicly available in the authors' GitHub repository [22].

In order to obtain a complete 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 [28] but they are outside the scope of this paper due to their costs and effort required.

C. Analyzing the Firmware

1) Unpacking Memory Images: Once a memory image has 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. Firmware-mod-kit [29], a collection of utilities contains several file formats and variations that binwalk does not support.

2) Brute Forcing Passwords: One of the more interesting feats of reverse engineering is password extraction. Native Linux passwords are used by default over secure shell (SSH) and telecommunication network (Telnet) connections and in some cases also for other services, such as HTTP and FTP. An known 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 may not be as trivial as "root," "admin" or "123456" but are still not complex enough to withstand an 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) [30] utility. The password hash files can be read freely by users with sufficient credentials and can also be extracted from the file system residing in the firmware.

This utility supports several hashing algorithms, but there are two that are most commonly observed in IoT devices.

1) *Descrypt*—A data encryption standard-based password hashing algorithm that uses a two character salt with 4096 different combinations. Although passwords may exceed eight bytes, only the first eight bytes are hashed and tested. A modern high-end graphical processing unit (GPU) is capable of calculating over 9 × 10^8 descrypt hashes per second.
2) *Md5crypt*—An message digest algorithm 5-based password hashing algorithm that supports a salt value of 12–48 bits allowing up to 2^{48} different combinations. md5crypt do not impose any length limitation on passwords. A modern high-end GPU is capable of calculating over 10^6 md5crypt hashes per second.

While simple passwords can be recovered using generic password recovery tools, such as John the Ripper [31], advanced password cracking can be achieved with hashcat [32]. Hashcat is an advanced password cracking program which supports advanced rules and patterns and is designed for GPU hashing. Hashcat use requires more knowledge than using John the Ripper and it is widely used for the recovery of difficult passwords.

In order to perform efficient password cracking, a word list or password generation 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 more effective against tested IoT device passwords. The pattern generation rules employed consist of: up to two symbol characters; up to two three uppercase characters; any amount of digits and lowercase characters; and up to eight characters total.

Another observation we made was that many elements of password difficulty inversely correlates with their frequency in password selection. For example, nonalphabetic characters are difficult to guess but are used less often than other characters; digits are easy to guess and are widely used; and uppercase characters are used less than lowercase characters. This allowed sorting the pattern list according to increasing guess difficulty levels while expecting to guess passwords in the early stages of evaluating the list. More on the results of password cracking can be found in Section III. A Python script for generating and sorting the pattern list is publicly available in the authors' GitHub repository [22].

3) Detecting Vulnerabilities Within the Firmware: As firmware images contain the OS 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, extensive research has been conducted in this field [18], [33]–[36].

III. Results

A. Devices Under Inspection

Table II describes 16 IoT devices that were subjected to reverse engineering. As shown in the table, our 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 paper contain cameras. In addition, there are two smart doorbells that are capable of streaming video, audio, initiating VOIP sessions as well as 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 II. All of them contained the embedded Linux OS.

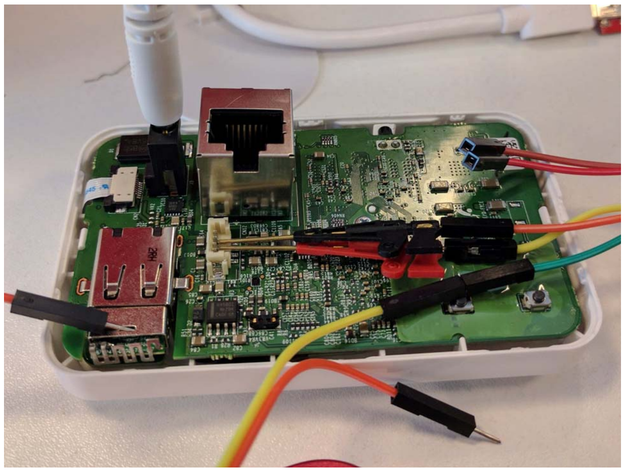
Fig. 6 shows an example of how an investigator may easily attach probes to a target device without the need of soldering. Temporary probe connections, such as the ones used during the evaluations in this paper, leave no evidence of tempering with the device other than opening it.

B. Techniques Applied on Devices

During the evaluation of the techniques presented in Section II, various approaches were used on the devices under test. Table III presents a sample of the devices inspected along

TABLE II
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	Wi-Fi Doorbell	Yes	Open door/gate	80
15	Doorbell	Ennio	SYWi-Fi002	Yes	Open door/gate	119
16	Thermostat	Ecobee	3 (golden firmware)	No	HVAC control	170


(a)

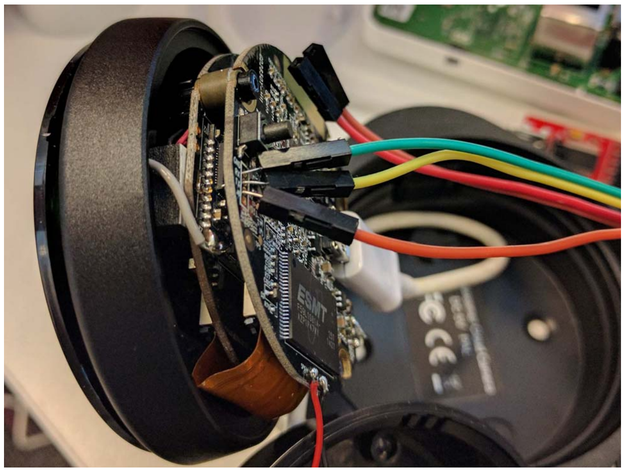

(b)

Fig. 6. Temporary probe connections to UART terminals of inspected IoT devices, temporary connections are useful for making quick evaluations of devices. (a) Two micro grabbing probes connected to the Samsung SNH-1011N smart camera UART terminal while a third probe is connected to the chassis of the USB port for grounding. (b) Three probes connected to the Xtreamer Cloud Camera UART terminals, the probes are being held in place by friction.

with the properties that allow or hinder reverse engineering. The table also mentions the techniques that were shown to be effective against these devices.

We describe some of the obstacles encountered and how they were overcome below.

1) Device 2—Bootloader in Read-Only Mode: The bootloader on device 2 did not allow keyboard input, leading to difficulty bypassing the Linux password. The approach selected in this case was to cause a physical fault during the boot process and observe the results.

A paperclip was used to bridge the memory MISO pin and the ground pin, causing any read from the memory chip to fail. After several attempts, the boot process fell back into a shell process allowing filesystem manipulation.

2) Device 5—Bootloader Was Password Protected: Device 5 contained a password-protected bootloader, and two approaches were successfully used to retrieve the password. The first approach included reading the memory chip using a CH341A USB adapter connected to an SOIC-8 clip for easy attachment to the chip; the dumped memory was later analyzed, and the password was found within. The second approach utilized the Saleae Logic Pro 8 logic analyzer which was attached to the memory chip; by analyzing the outgoing signals we were able to obtain a more narrow memory image including only the bootloader code where the password was easily found.

3) Device 8—No Network Tools Were Available on the Device: While analyzing device 8, no network tools could be found to facilitate the transfer of files to and from the device. In order to facilitate the transfer of the NetCat tool executable, the echo technique was used in which raw binary data is translated to "echo" commands that are automatically sent over the UART interface using a Python script.

4) Device 16—UART Interface Pads Were Not Easily Identified: A plethora of test pads made the task of locating UART terminals a difficult task when inspecting device 16.

TABLE III
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 fall back	Used Wget to download NetCat
5	Unmarked pads*	No	No	-	Physically read the onboard 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 inspecting the PCB (with assistance from the UART discovery assistant module described in subsection II-A3.
** Bootloader password was recovered using a logic analyzer to sniff communication on the memory bus.

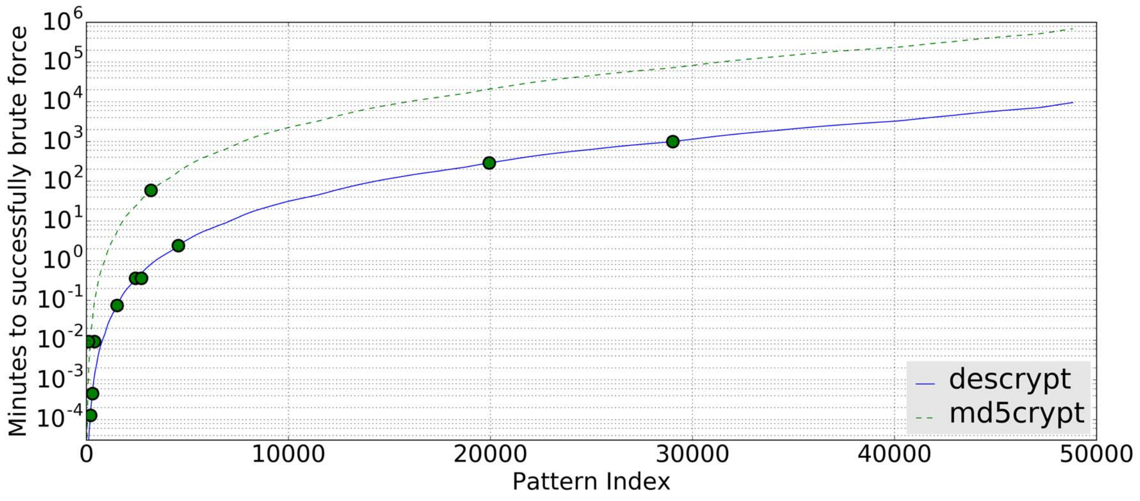


Fig. 7. Time required for password recovery using the GPU server described in Table I. Each marking on the graph represents a successfully recovered password of an inspected device.

Using the UART discovery assistant module described in Section II-A3, the detection of the UART terminals became an effortless endeavor. The UART terminal location can be seen in Fig. 4(b).

C. Discoveries Made During the Evaluation

1) Login Credentials: One of the most significant steps of reverse engineering an IoT device is to identify 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 onto the device with root user privileges is obtained. As can be seen in Table IV, eight of the devices contained passwords hashed with the descrypt algorithm, whereas the other eight devices employed md5crypt. The correct selection of the hashing algorithm is critical for resisting password cracking, for example, descrypt hashing can be as much as 90 times faster than md5crypt, as described in Section II-C.

The pattern-based password recovery described in Section II-C was used against all of the extracted password hashes. Fig. 7 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 increasing 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 nonempty passwords were recovered within the first 5000 patterns, after testing only $5.22e + 11$ passwords. The theoretical bound for testing that many passwords on a strong GPU server is 2.4 min for descrypt hashes and 217 min for md5crypt. Actual password recovery can impose significant overhead on theoretical bounds.

Eleven nonempty passwords were recovered, and one device contained an empty password. Four passwords still had not yet

TABLE IV
DISCOVERED DEVICE PROPERTIES

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

been recovered and are expected to be revealed within several weeks. Table IV shows the password complexities which varied between very low complexity (e.g., “abcd”) to medium complexity (e.g., “AbC123de”); undiscovered passwords were given the complexity rating “Unknown.” All the passwords discovered 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.

2) Remote Access: A simple port scan using Nmap [37] revealed that many of the tested devices have administration services bound to open ports, such as SSH or Telnet, which allows 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 IV. 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.

3) Wi-Fi Credentials: IoT devices must be connected to the Internet in order to function properly. In order to maintain wireless connection persistency after reboots and power shortages, a configuration file that holds the Wi-Fi credentials is typically stored by Linux and was located in all of the tested devices. The configuration file is located in the mounted file system, usually under the “config” or “NetworkManager” paths, and contains all of the Wi-Fi settings, including the service set identifier and nonencrypted passwords. Retrieval of the correct file from an extracted file system can be done simply by searching the filesystem for relevant keywords.

4) 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. The usage of a public/private key communication scheme relies on the private key being secret and proper practice dictates that the server’s private key should only be present on the server.

In three of the devices a hard-coded private key used for secure communication was found, as shown in Table IV. With the private key exposed, secure communication is rendered insecure and exposed to violations, such as man-in-the-middle and communication sniffing attacks.

5) Rebranded Devices: Rebranding is the creation of a new look and feel for an established product or company. 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 are applied the device giving it a new brand and manufacturer. Identifying rebranded devices means that previously discovered private keys, hashed passwords, account credentials, and even the application vulnerabilities may be identical across several devices. Four of the inspected devices were found to share a nontrivial password or hashed password with products from different manufacturers, strongly implying a similarity between them. The devices were found using a simple Web search for the passwords and hashes that resulted in a number of forum posts that specify hashes and passwords of other devices.

IV. ANALYSIS

The techniques presented in Section II may be used for both malicious and beneficial activities. In this section, we demonstrate and discuss some of the possibilities that emerge from making the reverse engineering process more generic and streamlined, while considering the results seen in the previous section.

A. Extension of Existing Attacks Into New Platforms

1) Creation of Personalized Mirai Botnet With Increased Capabilities: The relative ease of compromising modern IoT devices combined with limited knowledge on the part of consumers regarding the security of their devices has created fertile ground for malicious exploitation of such devices.

Numerous distributed denial-of-service (DDoS) attacks have been traced and were shown to originate from networks of infected IoT devices commonly referred to as botnets.

The infamous Mirai botnet gained publicity after it was used against several online websites. After witnessing a large-scale DDoS attack on KrebsOnSecurity.com, Martin McKeay, Akamai's senior security advocate was quoted as saying, "Someone has a botnet with capabilities we have not seen before. We looked at the traffic coming from the attacking systems, and they were not just from one region of the world or from a small subset of networks, they were everywhere" [38]. Mirai malware infects IoT devices with an open Telnet port and default login credentials and adds them to the attacker's botnet army. The source code for Mirai was leaked later on to the Internet on and can be modified and used by anyone who desires [39].

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 it. 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 subsequently added to the network.

In this paper, we observed an interesting case involving ProVision security cameras; 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 credentials shared between the cameras of the same manufacturer. The aforementioned process allows the number of devices that are vulnerable to Mirai and similar botnets to increase. Considering that both cameras have been found to be rebranded (as seen in Table IV), other camera models will likely to be vulnerable to our modified Mirai malware without any further efforts on our part.

2) Remote Access to IoT Devices by Unauthorized Parties: Remote connection to an IoT device, via Telnet or SSH which can be performed by malware for various purposes, can 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 [40]. During the reverse engineering process several critical engineering faults that allow an outsider to use this device were discovered. Credentials were revealed that allow 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 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 (for example, the credentials for the Wi-Fi network from the devices unencrypted configuration file).

3) Execution of Arbitrary Code on IoT Devices: During the reverse engineering process, software is often uploaded onto the device in various ways. The ability to upload software and maintain persistency after restarts has significant implications on device security. It has been shown that it is possible to gain complete control of a device when physical access is available, and physical access to a device can be used to modify the device's behavior even when the device is no longer in proximity.

B. Possible Theoretical Attacks

1) Discovery of New Vulnerabilities: By using the black box reverse engineering process, an attacker that possesses 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. During the reverse engineering we found out that many IoT devices had old OS or firmware versions that are now outdated, or have been patched when vulnerabilities were discovered or fixed in later versions. After identifying the firmware or OS version of an IoT device, 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 [34], [41].

2) Extraction of Secrets From Publicly Accessible IoT Devices: Many IoT devices are intended for 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 entrance to a facility; the doorbell can also be wired to a door or a gate for remote unlocking. The doorbell is typically installed outside and may be accessed from the street. A direct result of the device's accessibility is the ability of an attacker to physically modify or sabotage the device. However, it is not just the device that may be affected, since confidential material may be extracted from the device giving the attacker access to the whole network.

3) Supply Chain Attacks: Malicious modification can also be performed as a part of the supply chain. An untrustworthy vendor or courier can reverse engineer a device without having any previous knowledge about it and perform modifications to the device. Once access had been gained, the software can be modified in ways that are not visible to the consumer. The user of an IoT device may utilize it without knowing it was tampered with, and perhaps even equipped with a backdoor or some other malware.

C. Beneficial Uses of the Reverse Engineering Process

There are uses of reverse engineering that are not malicious or illegal and can benefit the owner. Low-end products are often accompanied with insufficient information about their hardware or software. A concerned consumer can use the process we have presented to learn about the device and its properties. If the device has been rebranded the consumer could search the Internet for similar devices provided by other vendors. The consumer obtains 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 be helpful in securing products no longer supported by vendors. Becoming more knowledgeable and informed regarding the device's software and hardware can not only help the customer get to know their product; it also allows the owner to customize the device to meet his/her needs. After gathering the desired information the owner can manipulate the firmware or configuration and develop the device further, and even add missing functionality. Modification of stock devices can also be used to hinder censorship and other information blocking instruments.

V. Discussion

As the IoT market evolves, the competition among vendors in the race to be the first to create better and cheaper devices increases. This pressure may affect the products' design and lead to the release of devices with critical security weaknesses. Time is not the only obstacle for creating a secure product; as competition drives the prices down, the production process must also become cheaper. Although the hardware engineers designing the product often lack cyber security knowledge, employing penetration testers, and security analysts may be very expensive. This tradeoff between money and security usually favors the production of cheaper but less safe products. A gap exists between the amount that is known about new devices on the marketplace and what is needed for assessing and ensuring their security. The reverse engineering process empowers consumers and researchers with abilities to discover important details regarding devices available on the market and benchmark their security.

A. Recommendations for Implementers

Based on the analysis performed and results obtained in this paper, we make the following recommendations for improving the security of IoT devices.

1) Removing UART Ports: UART ports typically have no function in mature devices. While obfuscation of UART ports is a widely used technique for hindering reverse engineering, it may not be effective in the face of devices, such as the UART discovery assistant module described in Section II-A3. Whenever possible, UART ports should be removed from finalized products, and their terminals should not appear in board designs.

2) Restricting Access to UART Ports: In situations where UART ports are essential in consumer products, they can be set up as read-only. An example can be seen in the debug port available in Google Nexus phones that can be accessed through the headphone port; The system logs are funneled using UART communication while maintaining a read-only mode.

3) Protecting UART Ports: If a UART port is required and must be write-enabled, certain protective measures should be considered. Previous works suggested safeguarding UART ports in a similar fashion to JTAG protection [42].

4) Hardening Bootloader Security: Hardening of bootloader security should be considered; bootloaders can be protected by physical means so that they only go into debug mode when specified electrical criteria are met or when using passwords. Although bootloader passwords were observed during our survey, retrieval of the passwords was easy using communication dumps, meaning that more sophisticated defenses should be employed.

5) Usage of Unique Passwords: Using the same passwords in devices of the same model or manufacturer enables a low resource attack to be amplified across many devices. In addition to hashing passwords with a strong hashing algorithm, such as SHA-516 crypt, strong unique passwords should be used for each and every device.

6) Facilitating Password Replacement: Hard-coding passwords should be avoided. Users must be able and encouraged to replace passwords frequently and easily.

7) Encryption of Device Memory: When possible all of the device's memory should be encrypted, similarly to what is done with mobile devices.

8) Encryption of Sensitive Data: All sensitive data stored on the device, including configuration, should be encryption.

9) Pen-Testing Devices: Many of the issues uncovered in this paper could have been easily detected prior to product launch. Therefore, devices should be pentested before being deployed. The techniques shown in this paper and others, such as those shown by Ling *et al.* [33] can be used to create an infrastructure for device audit.

B. Conclusion

The increase in IoT technology's popularity holds many benefits, but this surge of new, innovative, and cheap devices is accompanied by complex security and privacy challenges. Vulnerabilities and design flaws in seemingly innocent and ubiquitous IoT devices are an opening for an adversary to exploit and misuse. As shown in Section IV, an attacker that gains remote or physical access to an IoT device may snoop on the owner's personal or sensitive information and use the device's capabilities for their own benefit. The evolution of cyber crime has not bypassed the IoT, and in recent years we have witnessed new types of cyber attacks that involve IoT devices. The accessibility of the black box reverse engineering process may accelerate the attacker's work and introduce new IoT cyber threats.

References

[1] Gartner. (2014). *Gartner Says 4.9 Billion Connected 'Things' Will Be in Use in 2015*. [Online]. Available: http://www.gartner.com/newsroom/id/2905717

[2] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the Internet-of-Things," in *Proc. 14th ACM Workshop Hot Topics Netw.* Philadelphia, PA, USA, Nov. 2015, pp. 1–5.

[3] D. Lund, C. MacGillivray, V. Turner, and M. Morales, "Worldwide and regional Internet of Things (IoT) 2014–2020 forecast: A virtuous circle of proven value and demand," Int. Data Corporat., Framingham, MA, USA, Rep. IDC #248451, 2014.

[4] Nest Labs. *Nest Learning Smart Thermostat*. Accessed: Sep. 15, 2018. [Online]. Available: https://nest.com/thermostat/meet-nest-thermostat/

[5] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, "Internet of Things (IoT) security: Current status, challenges and prospective measures," in *Proc. 10th Int. Conf. Internet Technol. Secured Trans. (ICITST)*, London, U.K., Dec. 2015, pp. 336–341.

[6] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Comput. Netw.*, vol. 76, pp. 146–164, Jan. 2015.

[7] I. Alqassem and D. Svetinovic, "A taxonomy of security and privacy requirements for the Internet of Things (IoT)," in *Proc. IEEE Int. Conf. Ind. Eng. Eng. Manag. (IEEM)*, Petaling, Malaysia, Dec. 2014, pp. 1244–1248.

[8] Z.-K. Zhang *et al.*, "IoT security: Ongoing challenges and research opportunities," in *Proc. 7th IEEE Int. Conf. Service Orient. Comput. Appl. (SOCA)*, Matsue, Japan, Nov. 2014, pp. 230–234.

[9] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in Internet-of-Things," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1250–1258, Oct. 2017.

[10] J. Lin *et al.*, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.

[11] M. Patton *et al.*, "Uninvited connections: A study of vulnerable devices on the Internet of Things (IoT)," in *Proc. IEEE Joint Intell. Security Inf. Conf. (JISIC)*, The Hague, The Netherlands, Sep. 2014, pp. 232–235.

[12] Shodan. *Shodan Is the World's First Search Engine for Internet-Connected Devices*. Accessed: Sep. 15, 2018. [Online]. Available: https://www.shodan.io/

[13] R. Bodenheim, J. Butts, S. Dunlap, and B. Mullins, "Evaluation of the ability of the shodan search engine to identify Internet-facing industrial control devices," *Int. J. Critical Infrastruct. Protect.*, vol. 7, no. 2, pp. 114–123, 2014.

[14] M. Tellez, S. El-Tawab, and H. M. Heydari, "Improving the security of wireless sensor networks in an IoT environmental monitoring system," in *Proc. IEEE Syst. Inf. Eng. Design Symp. (SIEDS)*, 2016, pp. 72–77.

[15] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[16] J. A. Halderman *et al.*, "Lest we remember: Cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[17] J.-L. Lanet *et al.*, "Memory forensics of a Java card dump," in *Proc. 13th Int. Conf. Smart Card Res. Adv. Appl. (CARDIS)*, Paris, France, Nov. 2014, pp. 3–17.

[18] J. Obermaier and M. Hutle, "Analyzing the security and privacy of cloud-based video surveillance systems," in *Proc. 2nd ACM Int. Workshop IoT Privacy Trust Security*, 2016, pp. 22–28.

[19] C. Hollabaugh, *Embedded Linux: Hardware, Software, and Interfacing*. Boston, MA, USA: Addison-Wesley, 2002.

[20] R. Davis, N. Merriam, and N. Tracey, "How embedded applications using an RTOS can stay within on-chip memory limits," in *Proc. 12th EuroMicro Conf. Real-Time Syst.*, 2000, pp. 71–77.

[21] Atmel Corporation. (May 2012). *Attiny13a Datasheet*. [Online]. Available: http://www.atmel.com/images/doc8126.pdf

[22] Anonymous. (2017). *The Author's Github Repository. Details Omitted for Anonymous Submission*. [Online]. Available: http://www.qqq.com

[23] M. S. Pedro, M. Soos, and S. Guilley, "FIRE: Fault injection for reverse engineering," in *Proc. 5th Int. Workshop Inf. Security Theory Pract. Security Privacy Mobile Devices Wireless Commun. (IFIP) (WG)*, Heraklion, Greece, Jun. 2011, pp. 280–293.

[24] L. Goubet, K. Heydemann, E. Encrenaz, and R. D. Keulenaer, "Efficient design and evaluation of countermeasures against fault attacks using formal verification," in *Proc. 14th Int. Conf. Smart Card Res. Adv. Appl. (CARDIS)*, Bochum, Germany, Nov. 2015, pp. 177–192.

[25] J. Da Rolt *et al.*, "Test versus security: Past and present," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 1, pp. 50–62, Mar. 2014.

[26] R. J. Anderson and M. G. Kuhn, "Low cost attacks on tamper resistant devices," in *Proc. 5th Int. Workshop Security Protocols*, Paris, France, Apr. 1997, pp. 125–136.

[27] D. Vlasenko. *Busybox: The Swiss Army Knife of Embedded Linux*. Accessed: Sep. 15, 2018. [Online]. Available: https://busybox.net/

[28] F. Courbon, S. Skorobogatov, and C. Woods, "Reverse engineering flash EEPROM memories using scanning electron microscopy," in *Proc. 15th Int. Conf. Smart Card Res. Adv. Appl. (CARDIS)*, Cannes, France, Nov. 2016, pp. 57–72.

[29] *Firmware-Mod-Kit Github Repository*. Accessed: Sep. 15, 2018. [Online]. Available: https://github.com/mirror/firmware-mod-kit

[30] *Crypt(3) Man Page*. Accessed: Sep. 15, 2018. [Online]. Available: http://man7.org/linux/man-pages/man3/crypt.3.html

[31] *John the Ripper Password Cracker*. Accessed: Sep. 15, 2018. [Online]. Available: http://www.openwall.com/john/

[32] *Hashcat Password Recovery Tool*. Accessed: Sep. 15, 2018. [Online]. Available: https://hashcat.net/

[33] Z. Ling *et al.*, "Security vulnerabilities of Internet of Things: A case study of the smart plug system," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 1899–1909, Dec. 2017.

[34] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *Proc. 23rd USENIX Security Symp.*, San Diego, CA, USA, Aug. 2014, pp. 95–110.

[35] D. D. Chen, M. Woo, D. Brumley, and M. Egele, "Towards automated dynamic analysis for Linux-based embedded firmware," in *Proc. 23rd Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, Feb. 2016.

[36] M. Liu, Y. Zhang, J. Li, J. Shu, and D. Gu, "Security analysis of vendor customized code in firmware of embedded device," in *Proc. Int. Conf. Security Privacy Commun. Syst.*, 2016, pp. 722–739.

[37] Gordon Lyon. *Nmap Security Scanner*. Accessed: Sep. 15, 2018. [Online]. Available: https://nmap.org/

[38] B. Krebs. *Krebsonsecurity Hit With Record DDoS*. Accessed: Sep. 15, 2018. [Online]. Available: https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/

[39] *Mirai Github Repository*. Accessed: Sep. 15, 2018. [Online]. Available: https://github.com/jgamblin/Mirai-Source-Code

[40] Philips. *Philips In.Sight Wireless HD Baby Monitor*. Accessed: Sep. 15, 2018. [Online]. Available: http://www.philips.co.uk/c-p/B120N_10/in.sight-wireless-hd-baby-monitor/overview

[41] A. Cui, M. Costello, and S. J. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," in *Proc. 20th Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, Feb. 2013, pp. 134–141.

[42] K. Rosenfeld and R. Karri, "Attacks and defenses for JTAG," *IEEE Design Test Comput.*, vol. 27, no. 1, pp. 36–47, Jan./Feb. 2010.

Omer Shwartz (S'17) received the M.Sc. degree from the Ben-Gurion University of the Negev, Beersheba, Israel, in 2018, where he is currently pursuing the Doctoral degree at the Department of Software and Information Systems Engineering.

His current research interests include hardware security and software security in smart device context.

Yael Mathov received the B.Sc. degree in computer science from the Ben-Gurion University of the Negev, Beersheba, Israel, where she is currently pursuing the M.Sc. degree at the Department of Software and Information Systems Engineering.

Her current research interests include IoT security, reverse engineering, cyber security, and machine learning.

Michael Bohadana is currently pursuing the M.Sc. degree at the Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Beersheba, Israel.

Yuval Elovici received the Ph.D. degree in information systems from Tel-Aviv University, Tel Aviv, Israel.

He is the Director of Deutsche Telecom Laboratories, Ben-Gurion University of the Negev, Beersheba, Israel, where he is a member of the Information Systems Engineering Department. His current research interests include computer and network security, information retrieval, and data mining.

Yossi Oren (SM'17) received the M.Sc. degree in computer science from the Weizmann Institute of Science, Rehovot, Israel, in 2008, and the Ph.D. degree in electrical engineering from Tel Aviv University, Tel Aviv, Israel, in 2013.

He is a Senior Lecturer (an Assistant Professor) with the Department of Software and Information Systems Engineering, Ben-Gurion University, Beersheba, Israel. His current research interests include implementation security (power analysis and other hardware attacks and countermeasures; low-resource cryptographic constructions for lightweight computers) and cryptography in the real world (consumer and voter privacy in the digital era; Web application security).