Santtu Kolehmainen

Security of firmware update mechanisms within SOHO routers

Master's Thesis in Information Technology

May 14, 2019

University of Jyväskylä

Faculty of Information Technology

Author: Santtu Kolehmainen

Contact information: santtu.o.kolehmainen@student.jyu.fi

Supervisor: Andrei Costin

Title: Security of firmware update mechanisms within SOHO routers

Työn nimi: Laiteohjelmistopäivityksien turvallisuus kotireitittimissä

Project: Master's Thesis

Study line: Cyber Security

Page count: 66+0

Abstract: Purpose of this thesis was to analyze the state of firmware update security within SOHO (Small Office/Home Office) routers as anecdotal claims of insecure routers are common and firmware updates are critical to the overall device security. A case study was performed, where 12 devices were analyzed using network and firmware level analysis. Analyzed devices were found to have trivial vulnerabilities where Man-In-The-Middle attacker could deny further updates or install malicious firmware through the network update mechanism. Results highlight the need for large-scale security analysis of similar devices and more secure development practices.

Keywords: Firmware updates, Embedded systems, Security analysis, Firmware analysis, Reverse engineering

Suomenkielinen tiivistelmä: Tutkielman tarkoituksena oli analysoida laiteohjelmistopäivityksien turvallisuutta kotireitittimissä, sillä anekdoottiset väitteet turvattomista reitittimistä ovat yleisiä ja laiteohjelmistopäivitykset ovat tärkeä osa reitittimen yleistä turvallisuutta. Tapaustutkimuksessa analysoitiin 12 laitetta sekä verkko- että laiteohjelmistotasolla. Laitteista löydettiin yksinkertaisia haavoittuvuuksia, joita hyödyntämällä ns. Man-In-The-Middle asemassa oleva hyökkääjä voi estää laitteen päivityksen tai asentaa haitallisen laiteohjelmiston päivityskanavan kautta. Tulokset korostavat laajemman tietoturva-analyysin ja turvallisempien kehityskäytänteiden tarvetta samankaltaisille laitteille.

Avainsanat: Laiteohjelmistopäivitykset, Sulautetut järjestelmät, Tietoturva-analyysi, Laiteohjelmistoanalyysi, Takaisinmallinnus

List of Figures

List of Tables

Contents

1 Introduction

In late 2016 several large Distributed Denial-of-Service (DDoS) attacks took place disrupting various services including Amazon, Netflix and Twitter. The attacks were accomplished using a botnet comprised of home routers, IP cameras, digital video recorders, printers and other network connected embedded and IoT devices. This botnet was formed using malware which specifically targets these resource constrained devices instead of traditional desktop computers. The malware, called Mirai, was able to gather consistent population of 200,000 - 300,000 devices to the botnet by guessing common login credentials used in the targeted devices. The large size of the botnet and simplicity of the attack vector raises questions about the state of embedded device security. Effects of the attacks also highlight the importance of security in embedded devices. (Antonakakis et al. 2017; Bertino and Islam 2017)

Security of embedded devices such as Small Office/Home Office (SOHO) or home routers is partly dependent on them being kept up-to-date. This is done by applying firmware updates which are used to fix discovered security vulnerabilities as well as introduce new functionality to the device. Therefore it is critical to have secure and efficient means of delivering, verifying and updating the firmware of an embedded device. Lack of proper update mechanisms will leave the device outdated and vulnerable to discovered exploits. In order to investigate and highlight the current situation, this thesis focuses on the study of firmware updates for embedded devices, in particular SOHO routers, and approaches this from security and practical standpoints.

Security of embedded devices has been researched before and insecure firmware update mechanisms are one of the common security issues ("OWASP Internet of Things Project" 2018). Practical exploitation has shown that vulnerabilities in update mechanisms are not only a theoretical issue (Ling et al. 2017; Rieck 2016). However, earlier research has given more focus to firmware modification attacks (Cui, Costello, and Stolfo 2013), where firmware update mechanisms are commonly exploited as the attack vectors to deliver an already tampered firmware payload. The lack of more focus on firmware update mechanisms combined with their importance makes it worthwhile to study issues surrounding the topic.

This thesis aims to provide information on the state of security in SOHO routers' update mechanisms. This information gives practical examples about update mechanisms, which can be used as a reference for security assessment, testing, and certification. It can also be used to construct signatures of known update mechanisms. In addition to that, results are expected to provide knowledge about challenges of automated analysis of network connected embedded device update mechanisms.

The objective of the thesis is to answers following practical and research questions:

RQ1: How update mechanisms are implemented in the selected cases of SOHO routers?

RQ2: What is the state of security of software/firmware update mechanisms in the selected cases of SOHO routers?

RQ3: What methods, techniques and tools can be used for (automated) security analysis of SOHO router update mechanisms? Also, is it feasible and practical to device a common assessment methodology?

Practical case study is conducted to determine the update mechanisms of the selected SOHO and home routers from multiple vendors. Objective of the case study is to collect practical information about how update mechanisms are implemented in real-life SOHO routers. Security of the found update mechanisms is analyzed as a part of the thesis to better understand threats against these mechanisms as well as possible countermeasures and fixes. A classification of the weaknesses in the update mechanisms is also attempted.

The case study uses both network-level and firmware-level file and binary analysis. Network-level analysis using packet capturing methods allows collecting network traffic originating from the target device to identify used protocols and possible weaknesses in the update mechanism. Firmware-level file and binary analysis is performed to supplement and extend the network-level analysis.

A comprehensive literature review is conducted to gain understanding of the current state and challenges in the firmware update mechanisms of Commercial-Off-The-Shelf (COTS) embedded devices such as SOHO network routers and DSL modems. Literature was searched from scientific databases and search engines (e.g. Google Scholar, Research Gate, IEEE

Xplore, ACM DL), preprint archives (e.g. arXiv.org), and relevant technical white papers.

The rest of the thesis is structured as follows: Chapter 2 establishes the general background of the topic. Methods and device set used in the study are described in Chapter 3. Then, Chapter 4 presents the results of the study. Discussion and future work are explored in Chapter 5. Chapter 6 concludes the thesis by summarizing the produced findings and key points.

2 Background

This chapter introduces the background of the thesis. First, the main object of interest, embedded system, is defined. Security issues surrounding embedded systems are introduced to the reader and security critical components of an embedded system are explained. Finally, the importance and problems of firmware and software updates are discussed to provide a background for the experimental part of the thesis.

2.1 Introduction to embedded systems

Embedded systems have integrated deeply into our everyday electronics and they can be found from most consumer and industrial devices. Common examples of devices that contain embedded systems include home routers, printers, video surveillance systems and mobile phones. Embedded systems are also present in industrial systems such as SCADA (supervisory control and data acquisition) and PLC (programmable logic controller) systems. (Costin et al. 2014)

Embedded system or embedded device is defined as a microprocessor-based system which purpose is to perform specialized tasks. In contrast to general-purpose computers where functionality is easily modified by changing software, embedded system's functionality is not designed to be changed by replacing software. This means that embedded devices are designed for singular task, whereas general-purpose computers can perform multitude of tasks by running different software. (Heath 2002)

Vahid and Givargis (1999) define embedded system as a computing system that is not desktop, laptop or mainframe computer. They note that precisely defining the term embedded system is difficult and instead propose three common characteristics shared by embedded systems: "Single functioned", "Tightly constrained" and "Reactive and real-time". Embedded system repeatedly executes the same program that it is designed for. This is significant difference compared to desktop systems where different programs are run frequently.

Constraints on embedded devices are generally tighter than those on other computing sys-

tems. Small size, low cost, limited power and real-time processing requirements are common design constraints for a embedded system. This creates its own set of challenges that differ from traditional computing systems. In addition to constraints real-time requirements are needed in some embedded systems, where results must be available with minimal delay and failing to meet these requirements can result in a full system failure. (Vahid and Givargis 1999)

Embedded devices are equipped with microprocessor which allows them to run program code that together with other data forms the software of the device. This software is usually called firmware. "IEEE Standard Glossary of Software Engineering Terminology" (1990) defines firmware as following: *"The combination of a hardware device and computer instructions and data that reside as read-only software on that device"*. Zaddach and Costin (2013) simplify the definition and call all code running on the embedded device's processor as firmware. Firmware contains the necessary software to accomplish the task the device was designed for. Firmware can contain a full operating system with complete file system combined with the used software libraries and applications. Full operating system can be replaced with smaller and often proprietary operating system that does not include all the features of a full blown operating system, where for example applications might not have separate privileges from the kernel. (Zaddach and Costin 2013)

Embedded system does not necessarily have a network connection, but with Internet of Things becoming more prevalent, the amount of interconnected embedded and other devices will steadily increase. Embedded devices can be equipped with IP stack to connect them to the Internet as it runs on small and constrained devices (Atzori, Iera, and Morabito 2010). Connecting devices to the Internet makes it possible to add more features that require communication with other devices, but also opens up additional security considerations regarding networking. This thesis focuses mainly on devices that are equipped with networking capabilities.

2.2 Security in embedded systems

Security is one of the important challenges in embedded systems. Embedded systems' role in mission critical systems makes consequences of a security breach a serious threat (Papp, Ma, and Buttyan 2015). As more and more embedded systems have the capability to connect to the Internet and complexity of these systems increases, security of embedded systems becomes more critical. When devices are connected to the Internet they are placed at a risk because they can be then accessed from anywhere in the world.

Embedded device security is a mix of technical problems and people problems. Technical problems can be solved with engineering efforts and software updates. Significant source of vulnerabilities is outdated development practices in the device industry. Old and widely known vulnerabilities and technical problems continue to appear in embedded and IoT devices even though they have been solved years ago in the computer industry. (Hyppönen and Nyman 2017)

People problems are related to learning and education of users. It is especially a problem in consumer devices where users do not necessarily know how to secure their device or they are not interested and ignore security of their devices (Hyppönen and Nyman 2017). User ignorance can make security features useless if they rely on user's knowledge and require user input. For example default passwords are known to be a security risk and it is common knowledge that they should changed. Nevertheless, default passwords are still successfully used to takeover devices, where even the basic security measures have been ignored (Antonakakis et al. 2017). These issues are pronounced in embedded systems where user interfaces are not always easily available. Similarly there are challenges with the maintenance and configuration aspect. Embedded systems need administrators as well, who ensure that the devices stay updated and secure (Koopman 2004). In home environments real administrators might not exists at all, which creates issues. To combat these problems manufacturers should adopt secure default configurations where ideally no additional effort from users is needed to make the device secure (Hyppönen and Nyman 2017).

Based on experiences of security vulnerabilities being found in different systems and partly anecdotal evidence, embedded systems are seen as insecure. However large scale analysis

of embedded firmware showed that the claim is not baseless (Costin et al. 2014). Cui and Stolfo (2010) also note that embedded devices are often thought to be less secure and easier to exploit than general computers, but there is a lack of scientific evidence for that claim. To investigate the claim they scan the Internet for embedded devices and show that over 540,000 embedded devices are accessible over the Internet using default credentials. In 2016 Mirai incident showed that the situation has not improved over time (Antonakakis et al. 2017).

Many different security issues in computing systems have been studied in the fields of computer and network security. However embedded systems present new challenges that are different from traditional computer environments. As earlier stated most embedded systems have resource constraints which require addressing before security can be implemented. Computational power of an embedded system is lower compared to a general-purpose computer. Security features consume part of the already low processing capabilities creating a trade-off between security and performance (Kocher et al. 2004). Increasing performance for security leads to additional costs which add up quickly when manufacturers produce millions of units each year while at the same time competing with other device makers (Koopman 2004). Another challenge comes from battery-driven devices, where the limited battery life is added to the list of constraints (Ravi et al. 2004). Attacks against battery can be used to perform denial of service type attacks even if deeper access to a device is not possible (Koopman 2004). For example flooding the device with continuous network requests increases the processor usage which then drains the battery. After it runs out the device becomes unavailable violating one of the important security goals. Real-time requirements create vulnerabilities, where attacks aim to disrupt the system by causing delays to the processing (Koopman 2004). With these requirements even the smallest of delays can cause unwanted effects or even system failures.

2.3 Attack surfaces in embedded systems

"OWASP Internet of Things Project" (2018) defines attack surface areas for IoT devices that must be understood in order to develop and manufacture secure IoT devices. Its objective is to give a holistic view of IoT security by listing out critical areas from security point of view and their common problems. The listing can be used as a guideline when testing security of

an IoT device. IoT and network connected embedded devices share similar security issues in the device component of the system. This makes device areas of the guideline applicable to embedded systems, which is why it is used to introduce common components of an embedded device and their potential security issues.

Device's physical interfaces present opportunities for the attacker. Vulnerabilities in this category require the attacker to have a physical access to the device. Open debug port such as universal asynchronous receiver-transmitter (UART) might allow attacker unauthenticated access to the bootloader or main operating system of the device. JTAG interface developed for debugging and testing can be used to control the processor over serial communication. These interfaces may allow adversary to gain command-line access to the device which can lead to device takeover or firmware extraction. Firmware extraction allows the attacker to reverse engineer device's functionality which helps in the search of other vulnerabilities and exploit development. ("OWASP Internet of Things Project" 2018; Skochinsky 2010)

If the device contains a web interface, it must be secured appropriately. Web interface vulnerabilities follow the common vulnerabilities found in other web environments ("OWASP Internet of Things Project" 2018). Vulnerabilities such as injection or sensitive data exposure might allow attacker to read confidential information, while broken authentication or access control flaws can result to unauthorized access to device functions ("OWASP Top Ten Project" 2017). Costin, Zarras, and Francillon (2016) analyzed web interfaces in embedded systems and their automatic system found severe web vulnerabilities in 185 unique firmware images, which proves that web security issues affect embedded devices as well.

Administrative interfaces are related to the web interfaces as it is possible that they are implemented together. In that case same web vulnerabilities must be taken into account in administrative interfaces. These interfaces should also implement secure credential management and limit the use of weak passwords. Default passwords should be prompted to be changed. Brute-force attacks can be defended against with account lockout policy or by throttling login attempts. ("OWASP Internet of Things Project" 2018)

Device firmware is an integral part of the device's ability to function. Firmware often contains multiple different software packages that provide services and functionality. Common

examples are a web server that is used to host the web interface of the device and a SSH server that might be used for remote administration. Outdated versions of software that are included in the device firmware form a security risk because the older software versions often contain known vulnerabilities of which information is publicly reported on vulnerability databases. Other firmware related vulnerabilities include hardcoded credentials or backdoor/debug accounts. Allowing firmware downgrading risks the possibility of reverting firmware version to a known vulnerable one. ("OWASP Internet of Things Project" 2018)

Device's network services are a critical part of the firmware. Every open network service increases the potential attack surface to threats coming through the network. Using unencrypted communication or poorly implemented cryptography will make communication channels insecure. Possible test or development services should be completely removed from production ready device. Generally all unused services should be turned off so they do not needlessly increase the attack surface. ("OWASP Internet of Things Project" 2018)

Update mechanisms also increase the attack surface of a device ("OWASP Internet of Things Project" 2018). Lack of update mechanism leaves the device unable to be updated which means that security patches can not be applied to fix found vulnerabilities. Adding update mechanism is not enough as included mechanisms must be implemented securely. Sending updates unencrypted over the network allows eavesdroppers to follow the process and extract the firmware image from network traffic. Update files should be signed so their authenticity can be verified. This provides protection against modification attacks, which have been shown to be effective in against real devices (Rieck 2016; Ling et al. 2017). However embedded device constraints and challenges such as limited device resources and untrusted infrastructure create difficulties in implementing secure update mechanisms compared to general purpose computers (Bellissimo, Burgess, and Fu 2006). Update mechanism should also have a fail-safe system in the case that update process fails to prevent situation where the device cannot be recovered from a failed installation without special methods.

2.4 Firmware and software updates in embedded systems

Maintenance is an important part of the software life cycle. It is estimated that over 60% of the life cycle is spent on maintenance of software (Gilliam et al. 2003). Updates or patches are the main method of software maintenance. They are used to fix faults and add new functionality to the software. From security perspective software and firmware updates are critical as new vulnerabilities are discovered frequently and will continue to be discovered over time placing systems at risk (Wash et al. 2014). Therefore, computing systems running software rely on updates to be secure. Software updates also have been observed as a potential security metric, as frequently updating software protects the user from latest vulnerabilities and can give an indication of the security state of a system (Khan, Bi, and Copeland 2012). Missing update functionality makes the device inherently insecure over time. It is important for security that all software running on the device is updatable (Hyppönen and Nyman 2017). This includes the operating system, software applications and used libraries.

Even if device firmware updates are available, challenges in applying updates hinder the patching process and might leave updates uninstalled (Simpson, Roesner, and Kohno 2017). For example if the user must manually apply updates this implies that the user should also know when updates are available. Keeping up with newest updates manually requires an effort from the user and even if notifications are received user must still access the device and apply it. In traditional desktop software users have been found reluctant to install updates based on negative experiences in the past or due to not understanding the necessity of updates (Vaniea, Rader, and Wash 2014). If a piece of software works properly for the user, it might not be updated based on the thought that there is no need to update working software. Users might also neglect updates when the purpose of the software is not completely understood.

To solve the problem of users not updating their devices, automation was added to the update process. Automatic updates have been the industry standard for modern desktop and mobile systems, but are still largely missing from embedded and IoT devices. They allow manufacturers efficiently fix bugs and provide new features with minimal user interaction. However designing and implementing automatic update system is not trivial as it requires modular design and fail-safe capabilities. (Antonakakis et al. 2017)

Removing human interaction from the update process creates new problems (Wash et al. 2014). Update process may require device restart in order to be fully installed which is a disruptive action and causes issues with availability if the restart is done forcefully. Automatic rebooting might cause serious disruptions if the system is critical. For example rebooting drone mid-flight might cause a crash and result in permanent damage (Hyppönen and Nyman 2017). Update itself might cause new bugs and user might want to postpone the update. Similar issue happens if an update removes or adds features that user does not agree with or is not ready to transition. The level of automation must be carefully thought out based on the requirements and use cases of the device as pure automation is not problem free solution.

Simpson, Roesner, and Kohno (2017) note that end of life presents one challenge for IoT devices. Similar challenges apply to embedded devices and other computing systems as well. When devices outlive the support of the manufacturer they will stop receiving updates leaving the systems outdated and eventually new vulnerabilities will be found for which no fixes will be provided. One example of end of life challenges comes from the PC world in the form of Microsoft Windows XP. Windows XP support ended in 2014, but it is still used in 2018 with about 2% of computers running it (Microsoft 2014; Statcounter 2018). Even though XP contains multiple serious vulnerabilities it continues to be used as old systems are not modernized. In the embedded and IoT world one point of speculation is that do the device vendors have motivation to provide long term update support and how long can this support be expected to last.

2.5 Introduction to firmware updates

Firmware update mechanisms can be implemented using various protocols and methods. Updates come in different formats and vary between devices and vendors. Sometimes customized protocols or methods may be used, which makes it impossible to provide comprehensive lists. Updates can also be delivered over a network or require physical storage media such as an USB drive or SD card. This section aims to introduce different technical characteristics of firmware updates by listing commonly used file systems, file formats and update process flows.

One example of a network firmware update process is shown in Figure 1. Firmware update process starts either automatically i.e. periodic checking or manually when triggered by the user. First device must check if a newer version of the firmware is available. This is typically done by downloading an update manifest from the update server. Manifest contains version information about the newest firmware, which is compared to the current version of the firmware residing in the device. If newer firmware is available, device then proceeds to download the updated firmware image. Lastly the device is updated with the downloaded firmware image which is flashed to the flash memory. Moran et al. (2019) present similar flow in recent work-in-progress Internet-Draft for IoT device firmware updates.

Key parts of the network update process are retrieving the manifest and update image. In order for the process to be secure, authenticity and integrity of both manifest and update image must be verified or the process will be left vulnerable. Neglecting manifest authenticity leads to attacks where the manifest is modified to contain old information, which denies important updates leaving the device vulnerable. Such attack can be performed automatically every time user tries to update their device provided that Man-In-The-Middle position is first accomplished and attacker knows the manifest format.

Weak update image integrity leads to more severe attacks where malicious firmware is planted to a device through the update mechanism. However, firmware modification attacks require high technical skill and knowledge about the device and its firmware. Open source tools are available to modify existing firmware images which lowers the barrier for creating malicious modification (Firmware-mod-kit 2019). Attack still requires Man-In-The-Middle position which increases the needed effort.

To defend against modification attacks digital signatures (Diffie and Hellman 1976) based on various cryptographic schemes can be used to verify both manifest and update image integrity and authenticity. For example public-key system consisting of private and public keys can be used to provide both integrity and authenticity by having the firmware update signed with vendor's private key. Device can then use corresponding public key to verify that the update has not been modified.

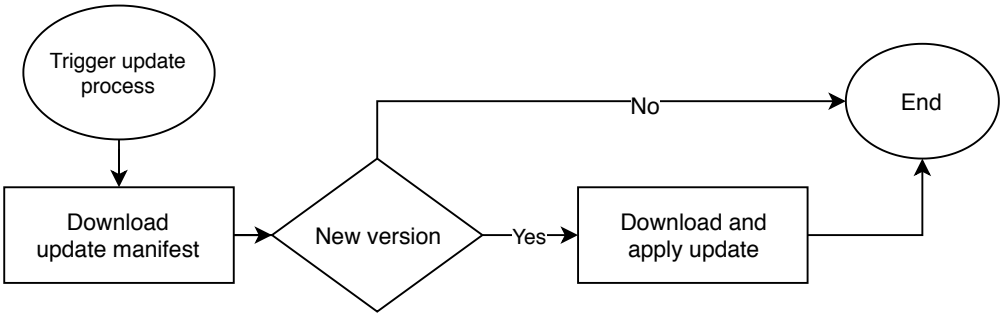

Figure 1: Overview of generic network firmware update process

2.5.1 Overview of firmware file formats

Firmware images may come in the form of more exotic file formats that are often meant for flash memory usage or microcontroller programming and are not commonly used in traditional computer systems. This section briefly introduces multiple file formats used in embedded systems.

Hexadecimal file formats encode binary data in plain text i.e. in printable characters using hexadecimal digits. This makes it possible to store and transmit the file on non-binary mediums and channels. Hex file can also be inspected easily with standard text editors as they are text files, but the downside of the ASCII hexadecimal encoding is increased file size as the encoding requires double the amount of bytes for data compared to a raw binary representation. (Intel 1988)

SREC or *Motorola S-record* is a hexadecimal file format developed by Motorola, where data is represented using ASCII character strings consisting of hexadecimal values. SREC file is formed from SREC records that are delimited by a line termination character. One record consists of 5 different fields (in order): type, record length, address, data, and checksum. Type field specifies the record type and can be one of the following values: S0, S1, S2, S3, S5, S6, S7, S8 or S9. As the type field always starts with a S character, files containing only SREC data will consist of lines starting with that character. Record length indicates the amount of hex digit pairs in address, data and checksum fields combined. Address points to a memory location where data should be loaded and the size of the address varies between 16, 24 and 32-bit depending on the record type. (Miller 2014; “Motorola S-records” 1998)

Intel HEX is another hexadecimal file format with similar structure as SREC. Record has six fields: record mark, record length, address, record type, data, and checksum. Record mark is always the colon (':') character, thus all records start with it. Record length specifies the number of data bytes in the record. Maximum value for record length is 255 as the field itself is one byte in size. All other fields except data are also fixed length. Intel HEX has 6 different valid record types: 00 to 05, which specifies the data field's functionality. Address field can only contain 16-bit addresses and longer addresses require first establishing a base address by using a record with one of the addressing record types. (Intel 1988)

Binary file formats contain raw binary data which makes them space efficient, but they can not be transmitted on channels that do not support all byte values. As binary files can contain non-printable characters, parts of the file are not human-readable. Binary files are interpreted according to some format that the device or software understand and uses. This means that same binary file can be interpreted differently in different devices or software, which makes it difficult to interpret binary files which format is unknown.

U-Boot *uImage* binary files are used in conjuction with the U-Boot open source boot loader. The file format consists of uImage header, which contains operating system, CPU architecture, and image type information combined with the actual data image (DENX 2019). Data image can for example contain operating system kernel, initial ramdisk and the actual file system (Yaghmour 2003). uImage files can be easily identified from their magic signature *0x27 0x05 0x19 0x56* (DENX 2019) located at start of the file and the U-Boot tools include utilities for creating and inspecting uImage files.

In some cases the firmware images come in very specialized formats such as printer firmware updates which are often proprietary and scarcely documented for general public, which increases the effort needed to perform further firmware analysis. For example some HP printers use special RFU (Remote Firmware Update) format which is intended for specific printer models and not publicly documented (Cui, Costello, and Stolfo 2013).

2.5.2 Overview of embedded file systems for firmware updates

File systems used in embedded systems are usually designed to be used together with flash memory devices (Skochinsky 2010). These flash file systems provide wear leveling which prolongs the lifetime of the underlying memory. File system can utilize compression to save storage space as space is often limited in embedded systems. Read-only file systems can be used to store permanent files to further reduce overhead and complexity. This section briefly introduces multiple flash file systems which are used in embedded systems.

Journalling Flash File System 2 (JFFS2) is a file system designed for embedded devices that utilize flash memory. JFFS2 is a log-structured writable file system and can be identified from it's magic signature 0x19 0x85 which is found at the start of every node in the log. It provides wear leveling and compression options. JFFS2 is included in the Linux kernel and unpacking can be done by mounting it in Linux or using public tools. (Woodhouse 2003; Skochinsky 2010)

SquashFS is a read-only file system included in Linux kernel. One of its intended use cases is constrained embedded devices where it can provide low overhead file system. Squashfs uses magic signature string "sqsh" (0x73 0x71 0x73 0x68 in hex) which can used to identify squashfs file systems in embedded firmware images. Squashfs file systems can be unpacked using publicly available tools such as *unsquashfs* found in common Linux distributions. The file system supports multiple compression algorithms such as LZMA and zlib. (Skochinsky 2010; Lougher 2011)

Cramfs is minimal read-only file system. Cramfs is intended to be used in very resource constrained environments where for example Squashfs might not be suitable. Cramfs can be identified from it's magic signature 0x28 0xCD 0x3D 0x45. It uses zlib compression to compress files while meta-data is left uncompressed. Cramfs limits maximum file size to 16MB and maximum file system size to 256MB. In systems where more memory is available and more features are needed Squashfs is recommended as an alternative. Cramfs file systems included in firmware images can be unpacked using *cramfsck*. (Skochinsky 2010; Pitre 2017)

Yet Another Flash File System (Yaffs) is a flash file system designed for embedded systems

with focus on speed and robustness. Yaffs is resistant to power failures during reading and writing which makes it suitable for uncertain environments. It has been used in Google's Android and aerospace systems. Yaffs does not have a dedicated magic signature which can make identifying it problematic especially for automatic systems. Found Yaffs file systems can be unpacked using open-source tools such as *yaffshiv* or *unyaffs*. (Skochinsky 2010; Aleph One Ltd. 2018)

FAT (File Allocation Table) is a file system made popular by its use in MS-DOS and can be nowadays found commonly on UBS sticks and memory cards. Its simplistic structure and good performance makes it suitable for embedded systems as well. FAT is based on an index table that holds information about clusters which represent storage areas. Table entries are used to infer file locations and free areas of the file system. FAT has three different versions: FAT12, FAT16, and FAT32, where the numbers indicate the amount of bits used for cluster count which directly affects maximum file system size. Maximum file size in FAT file systems is 4,294,967,295 bytes i.e. approximately 4GB, while maximum size of the file system is 256MB, 4GB or 2TB for FAT12, 16 and 32 respectively. (Keil 2018)

Myriad of other file systems optimized for flash memory, embedded devices or other special purposes exist with their own advantages and disadvantages and the choice of file system varies case by case. Examples include *SPIFFS*; intended for small memory devices and minimal RAM usage (Andersson 2017), *AXFS*; an execute-in-place filesystem (Corbet 2008), and *F2FS* which is designed for modern flash devices with performance in mind (Lee et al. 2015).

2.6 Introduction to firmware analysis

This section presents a general background for analysis and reverse engineering of embedded system firmware to support the later practical chapter of the thesis where some of the presented methods are used in practice to analyze multiple SOHO routers and their update mechanisms.

Firmware analysis or reverse engineering process aims to discover inner workings of embedded firmware in order to gain knowledge about its design and architecture. These objectives

are the same as with any reverse engineering process. The concept of reverse engineering is not limited to firmware or software as similar concepts are relevant when analyzing anything man-made, where the goal is to obtain otherwise unavailable information, ideas and designs (Eilam 2011). Traditionally reverse engineering has been used for example to analyze competing products or technology in various fields (Chikofsky and Cross 1990). In the field of cyber security, reverse engineering has applications for example in cryptography, malware and vulnerability research, where reverse engineering techniques allow skilled analyst to examine software programs of which no information is available in the form of documentation or source code to uncover vulnerabilities or malicious actions in case of malware (Eilam 2011).

Firmware analysis of embedded systems presents additional challenges compared to traditional PC-software reverse engineering. Embedded devices are a heterogeneous set which requires that human analysts and analysis tools must have the skill and ability to analyze different processor architectures, file formats and embedded operating systems combined with the limited amount of information available on some proprietary systems (Skochinsky 2010).

Figure 2 shows general firmware reverse engineering and analysis process. The process starts with firmware acquisition where firmware desired to be analyzed is retrieved from the device, vendor website or some other source. Acquired firmware is then inspected to identify the file format and sections of interests associated with the goal of the process. Interest areas can be analyzed further for example by disassembling code sections or inspecting strings contained in the file. Different functionality of the firmware can be then derived based on obtained information or additional methods such as hardware debugging or black box testing. (Basnight et al. 2013)

Next sections present the common steps of firmware analysis in more detail starting from obtaining firmware for analysis and continuing to unpacking. After unpacking, firmware can be analyzed both statically and dynamically in order to find security vulnerabilities or to obtain information about the firmware.

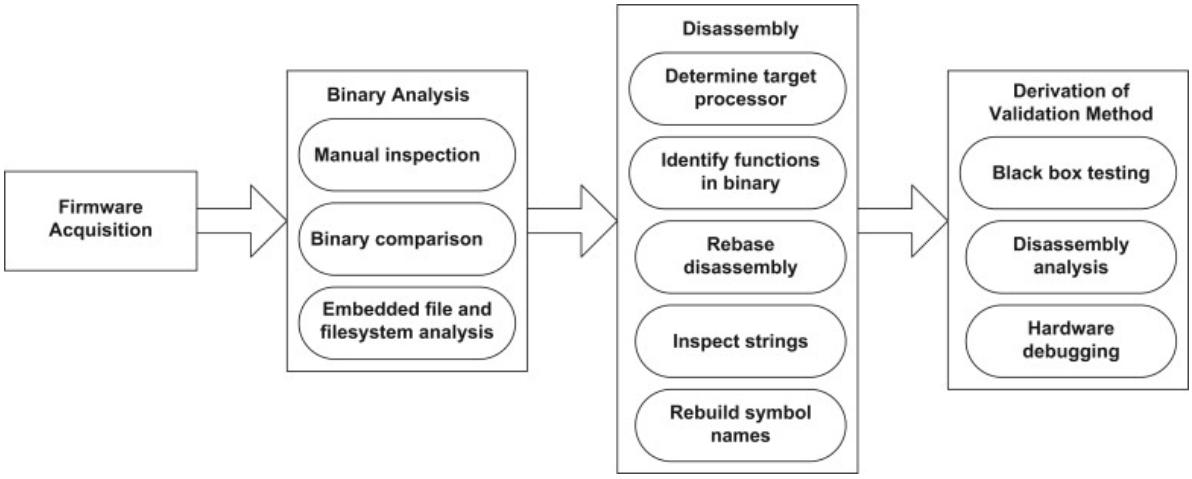

Figure 2: Overview of generic analysis process (Basnight et al. 2013)

2.6.1 Obtaining the firmware

Before firmware can be analyzed it must be first obtained. Firmware can be collected using multiple methods, but the chosen method depends on the situation and the device under analysis. Obtaining the firmware sample is not always trivial as firmware images might be encrypted or obfuscated or otherwise difficult to acquire (Skochinsky 2010).

One of the simplest ways of obtaining firmware is through update channels (Skochinsky 2010). Firmware update images can often be downloaded directly from the vendor's site. In this case direct access to the device is not needed which makes it resource effective method for obtaining firmware. This method can also be automated which makes it an effective way of collecting large amounts of firmware for analysis (Costin et al. 2014). If updates are not directly available for download, having access to a live device allows analyst to intercept traffic between the device and update server which can lead to a full firmware image.

For devices that allow remote command-line access via telnet or SSH it may be possible to dump the file system using aforementioned access. In Linux based embedded systems common utilities like Wget, curl, netcat (nc), SCP, FTP or TFTP can be used to transfer files over the network. (Shwartz et al. 2017)

Command-line access can also be obtained through UART port. This requires locating the physical port in the printed circuit board (PCB) and suitable equipment for connecting to

it. UART ports are commonly connected to a Linux console which is used for maintenance and prototyping during device's development stages. Therefore UART ports often give access to the underlying bootloader and/or operating system and can be invaluable method for accessing the device. (Skochinsky 2010; Shwartz et al. 2017)

When accessing a device through UART port, it is possible that a login process blocks access to the operating system. In cases like that it is recommended to try common username and password combinations like root or admin. If login is not successful it might be still possible to gain access to the bootloader. Bootloader is often accessed by pressing an arbitrary key during the boot process. Boot arguments can be then changed to bypass login processes completely. It is also possible that the bootloader itself is protected with password. However because bootloader's size is often heavily constrained, these password checks are commonly simple string comparisons and the password can be obtained using out-of-band methods. (Skochinsky 2010; Shwartz et al. 2017)

After command-line access is obtained, there are often multiple ways to copy the filesystem. For example simple read commands can be used to write files to the terminal output while receiving end stores them for later analysis (Shwartz et al. 2017). On Linux-based systems another method is to check for MTD (Memory Technonology Device), which provides layer between operating system and flash memory and allows analyst to copy its contents (Skochinsky 2010).

Lastly if other methods do not yield sufficient results it is possible to desolder the flash memory chip from the board and read it using special purpose memory readers (Skochinsky 2010; Shwartz et al. 2017). However this method damages the device and is only recommended to be used as a last resort or if surplus devices are available.

2.6.2 Unpacking the firmware

After firmware has been obtained, the next step in analysis is to locate the functional parts of the firmware for which further analysis can be performed. Firmware images generally consists of multiple files packed and compressed into a single file. Decompressing and extracting files from the image is called firmware unpacking. (Costin et al. 2014)

Unpacking phase presents additional challenges for the analyst. In comparison to file formats such as ZIP, TAR, PE or ELF which are publicly documented and commonly used in traditional software, firmware images have shortage of standards (Costin et al. 2014). Vendor modifications to compression algorithms and file systems create problems for unpacking as standard tools cannot be used to unpack such files (Hemel et al. 2011). Some vendors also create their own file formats which hinders the unpacking process (Costin et al. 2014). However, even some known file systems such as YAFFS2 can not be identified by magic bytes which makes them challenging to identify (Hemel et al. 2011). In addition firmware images can be obfuscated and encrypted to deliberately thwart analysis attempts.

Determining whether unpacking was successful is not trivial. When a file is extracted from the image and it is not recognized as any known file format, it is not possible to determine if the file is an unknown archive or just a plain data file. This problem makes fully automatic unpacking a significant challenge. (Costin et al. 2014)

To perform the actual unpacking in practice, publicly available tools can be used. Common tools include Binwalk (ReFirmLabs 2018), Binary Analysis Tool (BAT) (Tjaldur 2011; Hemel 2018b), and its successor Binary Analysis Next Generation (BANG) (Hemel 2018a). Binwalk utilizes known signatures and file carving methods to unpack firmware images. BAT was initially developed to find license compliance issues and can detect common bootloaders and file archives. BAT is no longer maintained and development has shifted to BANG which at the time of writing can detect and unpack (when applicable) over 100 file formats.

2.6.3 Static file and binary analysis

Static analysis is a program analysis method where a program is examined without executing it. From security perspective static analysis methods can be used to aid the process of finding security vulnerabilities, which might otherwise be missed. Static analysis is commonly performed with specialized tools, which use different techniques ranging from simple pattern and signature matching to more advanced abstract syntax tree methods to detect faults and vulnerabilities in the analyzed code. Manual auditing by human auditor can also be considered a form of static analysis, but is often a demanding task. Static analysis tools can

help human auditors to increase the efficiency of the analysis process, but similarly human auditors are needed to review the findings reported by an analysis tool for errors. (Chess and McGraw 2004)

Static analysis tools can make two kinds of errors: false positives and false negatives. False positives are findings that do not exist in the actual code but the tool erroneously reports them. In practice this means that human analyst must recognize and filter out false findings from true findings. In turn, false negatives are actual faults that the analysis tool does not recognize. (Chess and McGraw 2004; Costin, Zarras, and Francillon 2016)

Static analysis methods can be applied to source code or compiled program code (Chess and McGraw 2004). In firmware analysis and reverse engineering the latter is beneficial as source code of the firmware is rarely available. Possibility to analyze code without executing is also advantageous in firmware analysis as running the code often requires significantly more effort.

2.6.4 Dynamic binary analysis

In dynamic binary analysis programs are tested by running them in an executable environment. It includes techniques such as taint propagation, fuzz testing and symbolic and concolic execution (Zaddach et al. 2014). Dynamic analysis has several advantages, such as the ability to precisely identify and verify vulnerabilities. It can be used to verify findings from static analysis which are known to contain false positives (Costin, Zarras, and Francillon 2016). Dynamic analysis also allows the examination of obfuscated or packed code where static analysis methods fall short (Zaddach et al. 2014).

Disadvantages of dynamic analysis include the complex setup of testing environment, which takes time and effort compared to static analysis environments which are easily automated and scalable (Costin, Zarras, and Francillon 2016). This is especially problematic in embedded systems where devices can have proprietary and custom components combined with different processor architectures which makes it demanding for analysis tools (Zaddach et al. 2014). As dynamic analysis is based on executing the program, a basic requisite is an environment where the program can be ran. This environment can for example be either the

actual hardware where the program was intended to be executed, an emulated environment (Chen et al. 2016; Costin, Zarras, and Francillon 2016) or a hybrid approach (Zaddach et al. 2014).

Different environments all have their own problems. Hardware approach is precise as the environment corresponds to the intended one which means that executed programs can access all needed files and interfaces. However this approach requires procuring suitable physical device where the analysis can be performed. Even if the device is obtained the effort needed to prepare it for analysis might be significant or in the case of more advanced analysis even impossible. For these reasons the hardware approach does not scale and is not easily automated. (Chen et al. 2016)

Emulated environments overcome the requirement of having access to suitable physical device or hardware. This approach leverages software-based emulation which at the most complete level mimics the physical device and scales with the available computing resources. Emulated approach can be application-level emulation, where application data is extracted from firmware and executed outside the original context. Process-level emulation executes code inside the original file system, which allows programs to access for example configuration files residing in the file system. More complete approach is to use system-level emulation which aims to accurately emulate the original environment. System-level approach emulates different interfaces to peripherals which allows applications to access them, a problem which exists in the two earlier mentioned emulated approaches. However emulating embedded devices is not trivial and more complete approaches also require more implementation efforts with heterogeneity of devices increasing the complexity of more generic systems. (Chen et al. 2016)

Zaddach et al. (2014) present Avatar, a hybrid approach which combines both emulated and hardware environments. It aims to lessen the effort needed to implement system-level emulation by using hardware for I/O operations, while still using emulation to allow analysis that might not be possible to perform on the physical device. The approach allowed to analyze firmwares with unknown peripherals providing that the device was available.

2.7 Related and previous work

Firmware updates in embedded devices have been briefly studied in earlier research. More focus has been given to firmware modification attacks, but update mechanisms are commonly exploited as the attack vectors to perform these attacks. Thus, it is relevant to study existing modification attacks together with update mechanisms.

Cui, Costello, and Stolfo (2013) present a firmware modification attack against HP LaserJet printers. Vulnerability in the printer allows attacker to craft a malicious document which contains modified firmware. When this document is printed the printer will update its firmware. Analysis and discovery of the firmware format was done by reverse-engineering part of the boot code responsible for handling updates. The device itself was discovered to be running VxWorks operating system. Host-based security mechanisms do not exists within the operating system, which allows attacker to completely control the system. Based on the collected information a proof of concept malware was created with command and control capabilities.

Cui, Costello, and Stolfo (2013) also collected patch propagation data to track how many printers have been updated to a newer version where the discovered vulnerability has been fixed. After two months only 1.08% of the over 90 000 devices were updated, which gives slight indication that printers are rarely updated in timely manner.

Kemp, Czub, and Davidov (2016) studied the security of OEM (Original Equipment Manufacturer) PC vendors' software updaters. The findings were not positive as problems were found with every vendor that was inspected during the study. Common problems were lack of encryption when transmitting update manifests and updates, unsigned manifests and weak integrity checks. In total 12 different vulnerabilities were discovered.

Even though OEM software updaters are vastly different from embedded device update mechanisms, they still share common characteristics (and potential vulnerabilities (MITRE 2018)) with each other. It is worth to note that more mature PC software world has these problems, which makes it reasonable to study embedded device update mechanisms for similar issues.

Multiple different vulnerabilities were found from a smart plug system by Ling et al. (2017).

One of the demonstrated four attacks is a firmware attack, which takes advantage of insecure update procedure. Modified firmware is uploaded on the device and results in complete take-over.

Wurm et al. (2016) examine two IoT devices for security vulnerabilities. They are able to extract root password from consumer-grade home automation product and discover unencrypted firmware update using network analysis.

Rieck (2016) performs firmware modification attack on a fitness tracker. Update mechanism is identified as vulnerable and existing firmware image is captured from network traffic. Captured image is then reverse-engineered and modified. It is noticed that authenticity of the firmware is not verified which allows the attack to succeed. Similarly, Hanna et al. (2011) present a firmware modification attack targeted against a medical device. The tested device, an automated external defibrillator, was found to be vulnerable and counterfeit firmware could be uploaded to the device through the update mechanisms as firmware authenticity was not verified.

Basnight et al. (2013) studied the security of programmable logic controllers. They focused on firmware modification attacks and present a proof of concept attack against real PLC. Effort is made to describe general firmware reverse engineering process.

Simpson, Roesner, and Kohno (2017) discuss the problems and challenges of embedded and IoT device updates. They note that devices might outlive the manufacturer support or update capabilities may be missing altogether. There are also challenges in applying the released updates. Users do not necessarily update their systems and automatic updates might cause availability problems if the device must be restarted as part of the update process. "Central security manager" is proposed as a solution to manage vulnerabilities and updates in IoT networks.

Automatic methods and techniques for update mechanism analysis have not been studied widely, but there are some recent developments in the area. Visoottiviseth et al. (2018) present an automatic tool for home router firmware analysis with the capability to analyze update mechanisms. Their solution is based on keyword matching i.e. simple signatures. Automatic tools are one solution to the problem of scalability in embedded and IoT device

analysis as manual analysis of devices is time consuming task. The lack of earlier research needs to be addressed as embedded and IoT devices are becoming more widespread.

3 Methodology and experimental setup

This chapter describes the methods used in the update mechanism study. First the studied device set is presented to the reader by listing relevant information about each device. Background on SOHO routers and their security risks are discussed to rationalize the selection of devices used for the study. Second section introduces the analysis methods that were utilized in the practical part of the study with the aim that similar methods can be easily replicated in further analysis tasks and provides one approach to the task at hand.

3.1 Research methods

From research strategy point of view this thesis is based on empirical research as it is based on observations. Figure 3 illustrates the chosen strategies among other common research strategies. Case study was chosen as the main strategy because it allows to produce in-depth knowledge and analysis of the selected cases, which results in narrow but deep focus. This characteristic also fits with the overall aim of the study to produce more in-depth knowledge. In case study research cases should be selected so that clear boundaries for what defines a case can established (Patten and Newhart 2017). In this study each case is represented by a device from one category of embedded devices: commercially available routers intended for home and small office use with purpose of connecting other devices to the Internet. One additional advantage of case study research is the exploratory aspect of it. Case studies can be utilized as an initial exploratory study into the topic and used as information for further more generalizable studies (Patten and Newhart 2017).

As case study focuses more on the object of the study, it does not limit the available strategies that can be followed and can utilize both quantitative and qualitative approaches. In this thesis approach similar to qualitative approach is taken as the collected data is non-numerical, has qualitative properties and can not be clearly measured and the focus is not measuring, but understanding the characteristics and qualities of the target case. Due to technical nature of the cases and research questions, data collection and analysis methods must be suited accordingly and follow methods used in engineering. Data collection methods can also be

said to utilize observations and publicly available material such as technical documentation.

Figure 3: Overview of research strategies (narrower strategies are farther from the center) (University of Jyväskylä 2010)

3.2 Selection and details of analyzed devices

For this thesis' study 12 SOHO routers[1] from 6 different major and popular vendors were chosen. Table 1 shows the selected devices and their firmware versions at the time of study. The devices were chosen because of their availability within the resources of the study. The device set represents a sub-category of consumer-grade embedded devices and similar devices have been chosen for other router-security studies (Independent Security Evaluators 2013; The American Consumer Institute 2018). It is noted that release years listed in the Table 1 are based on available information gathered from online sources and the device, thus might not be completely accurate.

SOHO and home routers are among the most known and used embedded devices as they are necessary for Internet connectivity in common scenarios. Given their core networking role and central position, they process a lot of data that can have security and privacy value. Com-

1. ADSL modems were considered as routers based on their essentially identical functionality and architecture, except for the technology used to connect to the Internet.

promising a router opens up opportunities for further attacks. SOHO routers often include firewall capabilities, which are used to protect the network from unwanted traffic. Removing this layer of protection is advantageous for the attacker as it can expose the whole network behind the router to the Internet. Compromised routers like any other node on the network can be used to attack other devices connected to the same network, and are not only a threat to the owner of the device. Attacker can use compromised and captured routers alone as a foothold for further attacks which can be aimed at new targets, as demonstrated by the Mirai botnet (Antonakakis et al. 2017). All this context makes them a good and representative class of devices to study for security issues in embedded devices.

Vendor	Model	FW version	FW year	Device release year
ASUS	RT-N12+ B1	3.0.0.4.380_9880	2017	2016
ASUS	RT-N12E C1	3.0.0.4.380_9880	2017	2016
D-Link	DI-604	3.14	2005	2007
D-Link	DIR-655	1.37EU	2013	2006
Mediatrix	1102	4.5.7.54	-	-
Netgear	D1500	V1.0.0.25_1.0.1PE	2017	2016
Netgear	DG834GT	V1.03.23	2004	2004
Netgear	WNR612v3 N150	1.0.0.9	2016	2012
TP-Link	TL-WR841N	3.16.9	2016	2016
Zyxel	NBG-418N v2	V1.00(AARP.8)C0	2017	2014
Zyxel	NBG4115	V1.00(BFS.3)C0	2010	2009
Zyxel	NBG6602	V1.00(ABIL.0)C0	2017	2017

Table 1: Summary of analyzed devices

3.3 Analysis methods

An overview of used methods is depicted in Figure 4. Specific details about the methods follow the general descriptions. Weaknesses and challenges related to the test setup are included as part of the details to highlight the fact that one method is not always general enough to work against different devices and also to give more comprehensive view on the

topic.

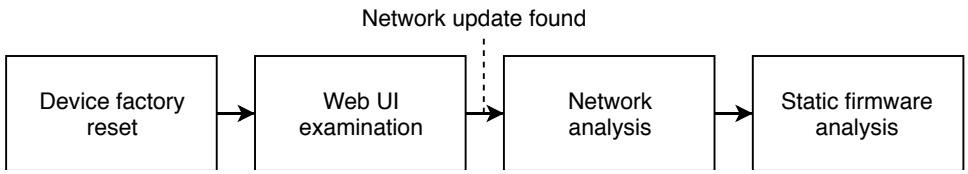

Figure 4: Overview of used methods

At the start of the examination a factory reset was performed for every device before they were investigated further. Resetting the device removes possible custom configurations and returns them to a state comparable to a new out-of-the-box device. In this case the factory reset does not affect the version of the firmware i.e. it does not downgrade or upgrade the device firmware.

Initial examination of the devices was done by browsing the web user interface. It was known beforehand that every selected device contains a web-based user interface which primary purpose is to allow user to configure the device. This interface is searched for firmware update feature to determine can the device be updated through manual means or over the network. These update mechanisms can be easily discovered from the user interface, commonly from under maintenance, firmware or admin sections. In this study manual update is considered to be an update process where user must manually determine if updates are available and then download the firmware image from vendor's support site. User then uploads the firmware image through the web user interface to update the device.

Network update is defined as a feature where user starts the update process manually but device checks and fetches the updates automatically over the network. Usually this done by clicking a specific button on the web interface. This can be thought of as user-initiated network update. Network update can also be completely automatic process where the device updates itself without user's input i.e device-initiated without user confirmation.

Other update mechanisms are excluded from this study as they can be said to be out of scope of a normal user. Some update mechanisms require more technical knowledge such as connecting to a device remotely and updating it via command-line utilities, setting up temporary FTP servers or accessing the device via serial interfaces such as UART, SPI or

I2C. Update mechanisms requiring additional physical storage devices like USB sticks or SD-cards were not considered in this thesis. More device specific mechanisms like special print commands for printers are also ruled out as not applicable for the chosen devices.

Devices containing network update functionality were chosen for further investigation. Network update mechanisms were examined by applying techniques utilized in earlier research. In this study two methods were chosen to cover both the network level and firmware level of the device. This allows for more comprehensive examination of the update mechanism. Firmware level analysis supplements the network level analysis as it is not always possible to see more specific details about the used update mechanisms such as what software is responsible for the detected network traffic. On the other hand performing network level analysis first provides preliminary information for the firmware level analysis and decreases the time spent on identifying software components related to the update mechanism.

3.3.1 Network level analysis

Wurm et al. (2016) use network analysis as one method to examine IoT device security. They capture network traffic with packet sniffing program and for example detect firmware update over unencrypted connection in their target device. Packet capturing can be used to understand the network traffic of an update mechanism. This method allows for reliable way of collecting network level information such as used protocols and presence of encryption because every outgoing and incoming packet can be logged for analysis.

In practice packet capturing device was implemented using Raspberry Pi 3 Model B+ running Raspbian Stretch which was configured as a router and packet analyzer. Figure 5 visualizes the used packet capture architecture. Packet capturing device is placed between the Internet and target device, which allows the packet capturing device to intercept and log all traffic originating from and destined to the target device. This position corresponds to what an attacker would obtain by performing a successful Man-In-The-Middle (MITM) attack. Target devices were allowed Internet connectivity with the intention that they can access and retrieve external resources, which all can be logged during the update process. This also frees the analysts from DNS lookup issues or networks checks that devices might perform

before they proceed with the update process. Client device is used to trigger target device's update process.

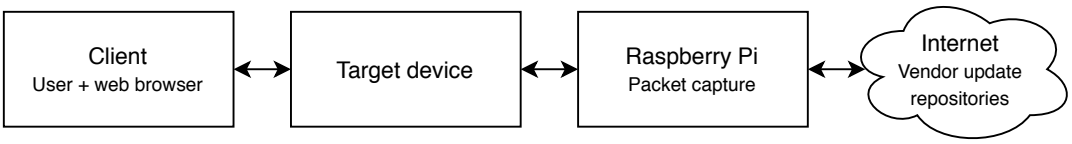

Figure 5: Architecture for generic capture of network packets during firmware update procedure

Wireshark (Wireshark 2018) was chosen to be used as the packet analyzer based on author's familiarity with the tool and its suitability for the task. It is used to log all incoming and outgoing packets that the target device sends or receives, which means it is effective against all protocols. Burp Suite (PortSwigger 2018) together with iptables (Netfilter 2018) are used to analyze HTTP and HTTPS traffic in more detail. Iptables is configured to redirect all TCP requests to ports 80 and 443 to Burp Suite's own proxy. Burp Suite can be then used to intercept and modify HTTP requests on the fly, which allows an easy method of testing different responses to HTTP requests that the device might perform.

Packet capturing can be hindered with the use of encryption (e.g. HTTPS, SSH), which can be implemented in multiple ways depending on used protocols and encryption methods. For example HTTPS (encrypted extension of HTTP) is commonly used in web traffic to protect against man-in-the-middle attacks by providing confidentiality and authentication. This defeats the shown network analysis method if HTTPS is used correctly.To test HTTPS usage Burp Suite's automatic TLS certificate generation is employed as it provides trivial way to detect issues where self-signed certificates are accepted. Burp generates self-signed certificate for each browsed host and serves them to the client. If HTTPS is fully and correctly used, client should deny self-signed certificates as they do not belong to any trusted certificate authority. Accepting self-signed certificates leaves the client vulnerable to man-in-the-middle attack as authenticity of the host can not be guaranteed. Other methods of encrypting network traffic were not considered in the planning phase based on earlier experiences with similar devices, where only HTTPS has been seen. It was also noted that analysis methods can be expanded based on examination of the device where more information is

available.

3.3.2 Firmware level analysis

Network analysis is supplemented with firmware image analysis. Firmware analysis techniques have been presented in earlier literature. Costin et al. (2014) unpack large amount of firmware images and statically analyze them to find vulnerabilities. Skochinsky (2010) presents techniques for analyzing and reverse engineering embedded firmware. Basnight et al. (2013) reverse engineer PLC firmware and present their process.

Firmware images for all devices containing network update feature were successfully obtained by downloading them from vendors' official support sites. Firmware image is then unpacked using binwalk (ReFirmLabs 2018), which attempts to carve and extract files inside the image. Unpacked files can be then analyzed to identify software libraries, executable files, used operating system and update mechanism components.

Unpacked file system is searched for common filenames of utilities used for transferring files over the network to identify relevant software components. This method follows the techniques Visoottiviseth et al. (2018) present in their research. Using filenames to identify software components has its own disadvantages. Filenames are not necessarily correct and might be faked in order to disguise malware or functionality of the analyzed file (Costin, Zarras, and Francillon 2017). However, in this case it can be argued that device vendors do not possess malicious intent to disguise software utilities with fake names. Nevertheless, vendors might have the motivation to obfuscate names in order to make reverse engineering more difficult. Bigger disadvantage is the fact that file names do not give strong indication of update mechanisms. For example searching for common FTP or HTTP clients may reveal that firmware contains said clients, but this information alone does not indicate an update mechanism. As update clients or scripts are often custom-made, it is difficult to compile a list of potential names beforehand which decreases the efficiency. This method mainly provides small pieces of information for manual analysis to fasten the process.

Software component identification is augmented using version string matching. In this method signatures of version strings inside a specific software component are created. How-

ever creating signatures requires a sample version of the targeted software component combined with manual work where suitable version strings are identified, which makes building large amount of signatures a demanding task. After a signature has been created it can be used to efficiently detect the targeted component. Signature method is not suitable for detecting unknown or new components that have not been encountered before. Zhang et al. (2018) use this method to extract version strings of binaries inside IoT firmware to identify software components and their versions.

Second part of the file analysis is URL extraction. URLs are extracted using two regular expressions. One expression matches valid URLs, while the other recognizes URLs containing format characters and variable placeholders from common scripting and programming languages. URLs have to contain the scheme name to be recognized. In these expressions following schemes were considered: FTP, HTTP and HTTPS. These URLs can be used to find update servers when network traffic analysis is not possible or was hindered by encryption. If network analysis was successful then it is likely that update server URLs are already known. However this knowledge can then be used to find files that contain said string, which makes it easier to identify relevant components.

The presented firmware analysis methods supplement traditional manual work where the file system of the firmware is explored and relevant files are identified based on analysts earlier knowledge and experience. After interesting files are discovered they can be analyzed further using text or hex editors, disassemblers and other binary utilities depending on the format of the analyzed file.

4 Results and case studies

This chapter presents the collected results from the update mechanism study. Device related results are divided in separate sections based on the device. Main results of the study are knowledge about design and architecture of update mechanisms used in selected devices which includes used communication protocols, update manifests and security issues found in said mechanisms.

11 devices out of 12 were found to have manual update mechanisms. Mediatrix 1102 is the only device that can not be updated through the web user interface but instead requires more advanced knowledge in order to be updated.[1] Thus, Mediatrix 1102 was not considered to have an easily accessible manual update mechanism making it the only device in selected set to not have one.

Network update is available in 7 devices out of 12. These update mechanisms use HTTP, HTTPS and FTP as their communication protocols. Table 2 partly summarizes the results of the update mechanism study. It is noted that all devices function as a client and not as a server. All 7 network update mechanisms look similar to the user. User navigates to firmware update page in the web interface and clicks a button to check for updates. Figure 6 shows a typical firmware update interface on SOHO routers. In this example the device provides both manual and network updates.

Security related results are summarized in Table 3. Overall level of security in update mechanisms were found to be from low to medium depending on the device. Firmware integrity was verified only in two devices and these two devices were the only ones that utilized encryption in connections to the update server. However, firmware manifest integrity was not verified in any of the devices, which is also important from security perspective. This means that all devices had at least one security issue, which could lead to update denial or firmware modification attacks.

Firmware unpacking was considered to be successful in 4 firmware out of 7, but as both Asus

1. TFTP server configuration is required: `http://wiki.media5corp.com/wiki/index.php?title=Firmware_-_Software_Upgrade_(SIP/MGCP)`

Vendor	Model	FW version	Network update	Manual update	FW unpack
Asus	RT-N12+ B1	3.0.0.4.380_9880	✓ (https)	✓	✓
Asus	RT-N12E C1	3.0.0.4.380_9880	✓ (https)	✓	✓
D-Link	DI-604	3.14	✗	✓	-
D-Link	DIR-655	1.37EU	✓ (http)	✓	✗
Mediatrix	1102	4.5.7.54	✗	✗	-
Netgear	D1500	V1.0.0.25_1.0.1PE	✓ (ftp)	✓	✗
Netgear	DG834GT	V1.03.23	✗	✓	-
Netgear	WNR612v3 N150	1.0.0.9	✓ (ftp)	✓	✓
TP-Link	TL-WR841N	3.16.9	✗	✓	-
Zyxel	NBG-418N v2	V1.00(AARP.8)C0	✓ (ftp)	✓	✗
Zyxel	NBG4115	V1.00(BFS.3)C0	✗	✓	-
Zyxel	NBG6602	V1.00(ABIL.0)C0	✓ (ftp)	✓	✓

Table 2: Summary of results

Orange items (HTTPS) were vulnerable to MITM due ignored certificates. Red protocols do not contain any security checks and are similarly vulnerable to MITM attacks.

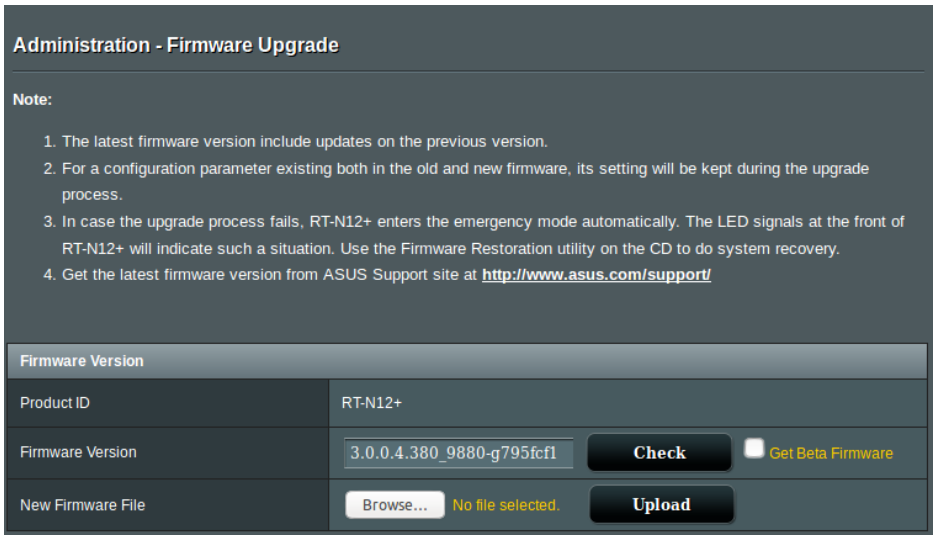

Figure 6: Typical SOHO router Web UI update interface (example of Asus RT-N12+)

Vendor	Model	Conn. encrypted	Manifest integrity	Firmware integrity
Asus	RT-N12+ B1	✓	✗	✓
Asus	RT-N12E C1	✓	✗	✓
D-Link	DIR-655	✗	-	-
Netgear	D1500	✗	-	-
Netgear	WNR612v3 N150	✗	✗	✗
Zyxel	NBG-418N v2	✗	-	-
Zyxel	NBG6602	✗	✗	✗

Table 3: Summary of network update mechanisms from security perspective

devices were found to be using the same firmware, the total amount of unique extractions was 3, which results in a 50 % unpacking rate for the study. All 3 unpacked firmware were similar in the sense that they were based on Linux kernel, contained SquashFS file system and came equipped with BusyBox utility. Table 4 collects file paths of executable files used for firmware updates from each of the unpacked firmware. Firmware that could not be unpacked were not analyzed further in the firmware analysis stage and results obtained for them are based on the network analysis, which results in significantly lower amount of information collected from those pieces of firmware.

Device	Manifest downloader	Image downloader	Image flasher
RT-N12+ B1	/usr/sbin/webs_update.sh	/usr/sbin/webs_upgrade.sh	/sbin/rc
RT-N12E C1	/usr/sbin/webs_update.sh	/usr/sbin/webs_upgrade.sh	/sbin/rc
WNR612v3 N150	/usr/www/cgi-bin/webupg	/usr/www/cgi-bin/webupg	/usr/bin/upgrader
NBG6602	/bin/get_online_info	/bin/get_online_fw	/sbin/sysupgrade

Table 4: Summary of executable files used for firmware updates

4.1 Asus RT-N12+ B1 & RT-N12E C1

Both Asus devices were found to be using the same firmware and update mechanism. The devices are based on Linux and contain kernel version 2.6.36 compiled for 32-bit little endian

MIPS. Common Unix utilities can be found from the firmware as it comes equipped with BusyBox 1.17.4. Command-line access to the device can be acquired over SSH, provided it is enabled first from the admin configuration.

The devices download manifest and update files over HTTPS, which is shown in Figure 7 and Figure 8 . Firmware analysis reveals that the update process consists of two shell scripts, that are used to download the update manifest to check the availability of a new firmware and to retrieve and apply the newest firmware image. Wget [2] is used to retrieve the update manifest and the update image. Used Wget options confirm that SSL/TLS certificates are ignored which degrades the security of the transfer. Connection is still encrypted as HTTPS is used which prevents eavesdropping, but because certificates are ignored it is possible for the attacker to impersonate the update server (e.g. Man-In-The-Middle (MITM) attacks) and for example attempt to deny updates.

File extensions of the ASUS update files indicate that the files are compressed zip files. Interestingly, this is not the case as examining the update manifest and release notes reveals that these files are just simple text files with UNIX-style line endings. Similarly Linux file command reports the update image as an U-boot Linux kernel image even though the file extension indicates a zip file.

Before update is applied, the script executes *firmware_check* and *rsasign_check* named binaries, which indicates that the firmware image is validated before it is applied to the device. The *firmware_check* is believed to check for correct file format so the device can not be bricked with random files. A closer look at the update files shows that a file ending in *_rsa.zip* is downloaded together with the update image as seen in Figure 8. Based on the filenames it is reasonable to assume that RSA signature is used to verify the image authenticity.

2048-bit public key can be found inside the firmware from path */usr/sbin/public.pem*. The downloaded signature contains 256 bytes of data which fits to the length of the public key. From security standpoint this scheme protects against modified firmware images as the integrity of the image is verified using RSA-based digital signature. The key length is recommended as safe at the time of writing (Barker and Roginsky 2019). To confirm that the

2. Wget open source network file retrieval tool: `https://www.gnu.org/software/wget/`

signature verification process works properly modified firmware images were uploaded to the device from man-in-the-middle position, which abuses the earlier found option to ignore certificates. Based on brief testing the signature verification process appears to work properly as otherwise correct firmware images are rejected when provided with a signature for another version of the firmware. It is concluded that the update mechanism protects against firmware modification attacks that come from the network. However the mechanism is not secure against attacks which attempt to deny updates for the device. This is possible as the attacker can easily impersonate the update server from man-in-the-middle position by redirecting traffic to his own server even though HTTPS is used. Because certificates are ignored, the update process does not stop when it receives invalid certificate from the server. After the connection is established attacker can return update manifest where firmware version is modified to be lower than the current one. For example setting the firmware version in the manifest to 0 denies updates for all firmware versions. Figure 9 shows an excerpt of the manifest containing version information. From user's perspective an attack against the manifest looks exactly the same as the case where device actually has the latest firmware, because device is forced to report that no new firmware could be found.

Update script is generalized to multiple different models as it branches according to what model is detected during the update process. It is also possible that some models do not support RSA signing as the update script checks if the required dependencies are present in the system and enables the RSA signature check based on that. In this case both of the analyzed devices supported RSA signing.

The update server does not list directories which means that clients have to know the exact URL to the update in order to download it, which makes it especially hard for automatic scrapers to gather firmware images for research purposes as filenames were noticed to contain long version and revision information. The update scripts also contain multiple URLs where the updates are downloaded based on the device model.

https://dlcdnets.asus.com	GET	/pub/ASUS/LiveUpdate/Release/Wireless/wlan_update_v2.zip
https://dlcdnets.asus.com	GET	/pub/ASUS/wireless/ASUSWRT/RT-N11P_B1_3004_380_10410-gea9e8bc_EN_note.zip
https://dlcdnets.asus.com	GET	/pub/ASUS/wireless/ASUSWRT/RT-N11P_B1_3004_380_10410-gea9e8bc_US_note.zip
https://dlcdnets.asus.com	GET	/pub/ASUS/wireless/ASUSWRT/RT-N11P_B1___EN_note.zip
https://dlcdnets.asus.com	GET	/pub/ASUS/wireless/ASUSWRT/RT-N11P_B1___US_note.zip

Figure 7: Asus RT-N12 update manifest query

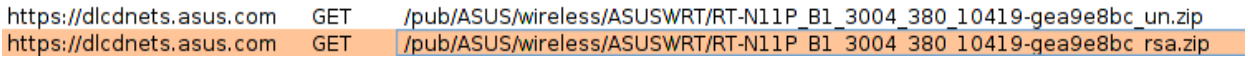

Figure 8: Asus RT-N12 update firmware query [3]

```
RT-N12VP#FW3004100#EXT0-gfe219b7#URL#UT4208#BETAFW#BETAEXT#
RT-N11P#FW3004380#EXT7378-g7a25649#URL#UT4208#BETAFW#BETAEXT#
RT-N11P_B1#FW3004380#EXT10410-gea9e8bc#URL#UT4208#BETAFW#BETAEXT#
RT-N10P_V3#FW3004100#EXT0-g84fa08d#URL#UT4208#BETAFW#BETAEXT#
RP-AC87#FW3004382#EXT18537-g255d5da#URL#UT4208#BETAFW#BETAEXT#
```

Figure 9: Asus RT-N12 update manifest (excerpt)

4.2 D-Link DIR-655

D-Link DIR-655 uses HTTP to fetch update manifests. Network traffic between the device and the update server reveals that the server is likely no longer supported as it responds with HTTP status code 404 - Not Found. This means that the device can not be updated using this channel, which makes it harder for end users to keep the device up-to-date and basically makes the device obsolete. Figure 10 shows captured HTTP communication between DIR-655 and the update server.

Unpacking the firmware for DIR-655 was not fully successful. The initial image contains archives compressed with ARJ which can be decompressed with publicly available tools. Decompressing leads to another image file which does not unpack successfully. With the update server not responding and unsuccessful unpacking the amount of information that could be collected from DIR-655 was very limited so proper analysis of its firmware update mechanism can not be provided. However, an assumption that HTTP is used through the update process would mean that no encrypted connections are used in this update mechanism.

Interestingly, even though the update server is unreachable and the device cannot confirm if updates are available it's web interface still displays a message to unaware users indicating that current firmware is the latest one, which leads to false sense of security for the user instead of notifying about the error. Figure 11 illustrates a problem with the user interface.

3. Version ends with 10419 instead of 10410 as the request was modified in flight to prevent the update

```
GET /router/firmware/query.asp?model=DIR-655_AX_EU HTTP/1.1
Host: wrpd.dlink.com.tw

HTTP/1.1 404 Not Found
Content-Type: text/plain; charset=utf-8
Date: Tue, 11 Sep 2018 09:58:12 GMT
ETag: W/"9-0gXL1ngzMqISxa6S1zx3F4wtLyg"
X-Powered-By: Express
Content-Length: 9
Connection: Close

Not Found
```

Figure 10: D-Link DIR-655 update query

Figure 11: D-Link DIR-655 misleading update interface

4.3 Netgear D1500 & WNR612v3

Netgear's D1500 and WNR612v3 have similar update mechanisms protocol-wise as both retrieve updates over FTP. Figures 12 and 13 show the network activity during firmware manifest download. Both devices retrieve the manifest from *updates1.netgear.com* which resolves to IP addresses included in Figures 12 and 13. Interestingly, WNR612v3 provides a password (possible client default of Busybox: busybox@) for the FTP server, but the password is not actually needed to access the files.

Firmware image for the WNR612v3 contains Squashfs filesystem which can be unpacked confidently, but the image for the D1500 consists of LZMA compressed data chunks which in this case did not unpack to clear separate files. Because the unpacking for D1500 can be

said to be only a partial success it is listed as an unsuccessful unpack.

Unpacked WNR612v3 firmware image was investigated in more detail for possible information about the update mechanism. The firmware itself is based on Linux kernel version 2.6.30.9 compiled for 32-bit big endian MIPS CPU architecture and contains BusyBox 1.6.1 which in turn contains many common Unix utilities. Files */usr/www/cgi-bin/webupg* and */usr/bin/upgrader* were identified to be potential candidates responsible for the update mechanism. Cursory reverse-engineering reveals that */usr/www/cgi-bin/webupg* is the actual firmware update downloader.

The downloader contains URLs for Netgear's update servers: *updates1.netgear.com, updates2.netgear.com* and *updates3.netgear.com*. At the time of writing only the *updates1.netgear.com* -domain appears to be working correctly i.e. resolves to an IP address and URLs/File paths shown in Figures 12 and 13 return meaningful content. Domain *updates2.netgear.com* does not resolve to any IP address and domain *updates3.netgear.com* resolves successfully, but does not respond to any HTTP/HTTPS/FTP connections.

The update process consists of retrieving the update manifest to check for newer versions. The update manifest is shown in Figure 14. It contains the filename and MD5 hash of the newest available firmware for the device. Update manifest is downloaded into */var/fileinfo.txt* where it is parsed for version information. Firmware update image is then downloaded into */var/image/img* and verified by calculating its MD5 hash which is compared to the one received in the manifest.

Verifying the firmware using MD5 hash provides protection against transmission errors. If the file changes during transmission, the calculated MD5 hash does not match the one provided in the manifest and the error can be detected. However this does not provide any security against deliberate modification attacks as an attacker can first fake the manifest and provide suitable MD5 hash for the modified firmware. Such attack works because the router cannot determine the authenticity of the communication peer. No public keys were found inside the firmware, which strengthens the observation that digital signature type schemes are not used to verify integrity and authenticity of the firmware image.

For comparison a manifest for D1500 is shown in Figure 15. Both devices have similar man-

ifests with the exception that WNR612v3 contains separate language files used to translate the router's web-based user interface. D1500 is suspected to have similar update mechanism based on the manifest and network analysis even though full confirmation cannot be given at the time of study because its firmware could not be fully unpacked.

Interestingly the downloader contains strings which indicate that it may support multiple network protocols. FTP was already detected during the network analysis, but in addition HTTP and HTTPS are referenced in the downloader. In fact the downloader contains full commands to download firmware images using wget with both protocols. Browsing the update site with a standard browser reveals that it supports HTTP but not HTTPS. From research standpoint both FTP and HTTP server have disabled directory listing which makes it difficult to scrape firmware from the site for further analysis.

Finally the */usr/bin/upgrader* is used to write images to the flash memory. It is called from */usr/www/cgi-bin/webupg* after an image has been downloaded and verified.

Source	Destination	Protocol	Length	Info
34.241.19.122	192.168.10.19	FTP	86	Response: 220 (vsFTPd 2.2.2)
192.168.10.19	34.241.19.122	FTP	82	Request: USER anonymous
34.241.19.122	192.168.10.19	FTP	100	Response: 331 Please specify the password.
192.168.10.19	34.241.19.122	FTP	73	Request: PASS
34.241.19.122	192.168.10.19	FTP	89	Response: 230 Login successful.
192.168.10.19	34.241.19.122	FTP	74	Request: TYPE I
34.241.19.122	192.168.10.19	FTP	97	Response: 200 Switching to Binary mode.
192.168.10.19	34.241.19.122	FTP	72	Request: PASV
34.241.19.122	192.168.10.19	FTP	118	Response: 227 Entering Passive Mode (34,241,19,122,244,132).
192.168.10.19	34.241.19.122	FTP	95	Request: RETR /D1500/pe/fileinfo.txt
34.241.19.122	192.168.10.19	FTP-DATA	404	FTP Data: 338 bytes (PASV) (RETR /D1500/pe/fileinfo.txt)
34.241.19.122	192.168.10.19	FTP	147	Response: 150 Opening BINARY mode data connection for /D1500/
192.168.10.19	34.241.19.122	FTP	72	Request: QUIT
34.241.19.122	192.168.10.19	FTP	90	Response: 226 Transfer complete.
34.241.19.122	192.168.10.19	FTP	80	Response: 221 Goodbye.

Figure 12: Netgear D1500 version check

Source	Destination	Protocol	Length	Info
34.243.216.129	192.168.10.26	FTP	86	Response: 220 (vsFTPd 2.2.2)
192.168.10.26	34.243.216.129	FTP	82	Request: USER anonymous
34.243.216.129	192.168.10.26	FTP	100	Response: 331 Please specify the password.
192.168.10.26	34.243.216.129	FTP	81	Request: PASS busybox@
34.243.216.129	192.168.10.26	FTP	89	Response: 230 Login successful.
192.168.10.26	34.243.216.129	FTP	74	Request: TYPE I
34.243.216.129	192.168.10.26	FTP	97	Response: 200 Switching to Binary mode.
192.168.10.26	34.243.216.129	FTP	97	Request: SIZE wnr612v3/ww/fileinfo.txt
34.243.216.129	192.168.10.26	FTP	76	Response: 213 6436
192.168.10.26	34.243.216.129	FTP	72	Request: PASV
34.243.216.129	192.168.10.26	FTP	117	Response: 227 Entering Passive Mode (34,243,216,129,238,0).
192.168.10.26	34.243.216.129	FTP	97	Request: RETR wnr612v3/ww/fileinfo.txt
34.243.216.129	192.168.10.26	FTP	150	Response: 150 Opening BINARY mode data connection for wnr612v3/
34.243.216.129	192.168.10.26	FTP-DATA	1454	FTP Data: 1388 bytes (PASV) (RETR wnr612v3/ww/fileinfo.txt)
34.243.216.129	192.168.10.26	FTP-DATA	1454	FTP Data: 1388 bytes (PASV) (RETR wnr612v3/ww/fileinfo.txt)
34.243.216.129	192.168.10.26	FTP-DATA	1454	FTP Data: 1388 bytes (PASV) (RETR wnr612v3/ww/fileinfo.txt)
34.243.216.129	192.168.10.26	FTP-DATA	1454	FTP Data: 1388 bytes (PASV) (RETR wnr612v3/ww/fileinfo.txt)
34.243.216.129	192.168.10.26	FTP-DATA	950	FTP Data: 884 bytes (PASV) (RETR wnr612v3/ww/fileinfo.txt)
34.243.216.129	192.168.10.26	FTP	90	Response: 226 Transfer complete.
192.168.10.26	34.243.216.129	FTP	72	Request: QUIT
34.243.216.129	192.168.10.26	FTP	80	Response: 221 Goodbye.

Figure 13: Netgear WNR612v3 version check

```
[Major1]
file=wnr612v3_wnr500-v1.0.0.26.img
md5=f0375127246784c64b9a86688b46b90d
size=3629060
o1=<MSG101>
o2=<MSG102>
o3=<MSG103>

[wnr612v3-1.0.0.26-Arabic-language-path]
file=wnr612v3-1.0.0.26-Arabic.tar.gz
md5=9740e2bab14a0e2f72619fb325821941
size=100631

[wnr612v3-1.0.0.26-Czech-language-path]
file=wnr612v3-1.0.0.26-Czech.tar.gz
md5=5ef02fc0915b2be7d3e7cf1b3e754a76
size=102325
```

Figure 14: Netgear WNR612v3 update manifest (excerpt)

```
[Major1]
file=D1500-V1.0.0.27_1.0.1PE.img
md5=3FC4A1B823D9BEDE309AB0212598910D
size=1927944
o18=<MSG018>
o20=<MSG020>
o24=<MSG024>
o25=<MSG025>
o27=<MSG027>
```

Figure 15: Netgear D1500 update manifest

4.4 Zyxel NBG-418N & NBG6602

Zyxel's NBG-418N and NBG6602 use FTP to download firmware updates. Packet captures of the version check process are shown in Figures 16 and 17. The process is essentially the same for both devices, which suggests that similar update mechanism is used in these models. Zyxel's update server is hosted at *ftp2.zyxel.com* and contains firmware for multitude of devices. Browsing the server for firmware is trivial as directory listing is enabled and allows obtaining firmware, release notes and/or manuals for 726 different device models. To clarify the amount, all of the objects are not necessarily available for all devices so the amount of available items might not equal the total amount of devices.

Unpacking success rate for selected Zyxel devices was 50%. Firmware for the NBG-418N contains multiple LZMA compressed data chunks, which in this case, similarly to one of earlier analyzed firmware, did not unpack into sensible separate files and was classified as unsuccessful unpack. NBG6602's firmware unpacks successfully allowing further analysis.

Initial examination of NBG6602's firmware reveals a Linux based firmware with a Squashfs filesystem. The Linux kernel is compiled for 32-bit little endian MIPS CPU and the kernel version is 3.10.14. Deeper look into the firmware shows that it is based on OpenWrt [4] Barrier Breaker 14.07, an open source router firmware, that has been modified for NBG6602 by the vendor.

4. OpenWrt: `https://openwrt.org/`

NBG6602's firmware update consists of series of shell scripts. Files */bin/get_online_info* and */bin/get_online_fw* are the downloaders responsible for retrieving update manifests and firmware update images. Wget is used as the client to retrieve both the manifest and update file. Update server URL can be found from separate configuration file, which matches with the observations during network analysis. Firmware update manifest for NBG6602 is shown in Figure 18 . For comparison the manifest for NBG-418N is provided in Figure 19 . Manifest format is identical in both devices, which further indicates similarities in their update mechanisms. Script named */sbin/sysupgrade* is used to update the device after a firmware image has been downloaded. This script is based on the one included in OpenWrt. Before this file is executed, device performs sanity checks that the update fits to the flash memory and that the image is intended for the device, which is done using string comparison with information that is read from the update image header. The update process itself is triggered from the web interface which uses Lua programming language in the backend to process requests.

Security-wise NBG6602's update mechanism uses unencrypted connection and does not verify the authenticity of the downloaded firmware which makes it vulnerable to firmware modification attacks. Similarly the integrity and authenticity of the update manifest is not verified, which allows attackers to forge firmware manifests to block updates. Firmware does not contain any public keys which could be used as part of a digital signature scheme and analysis of the code reveals that such feature is not present. Based on the code, NBG6602 also allows firmware downgrading as version comparison is done using negation operator, which means that firmware versions that are not same as the current one will pass the check. This is seen in Figure 20 and Figure 21, which are code snippets from the Lua web interface controllers. In Figure 21 the *on_line_check_fw=1* parameter tells the corresponding HTML template file to show upgrade button to the user i.e. version check has passed and the device can be updated.

Source	Destination	Protocol	Length	Info
66.171.116.101	192.168.10.22	FTP	86	Response: 220 (vsFTPd 2.2.2)
192.168.10.22	66.171.116.101	FTP	82	Request: USER anonymous
66.171.116.101	192.168.10.22	FTP	100	Response: 331 Please specify the password.
192.168.10.22	66.171.116.101	FTP	83	Request: PASS anonymous@
66.171.116.101	192.168.10.22	FTP	89	Response: 230 Login successful.
192.168.10.22	66.171.116.101	FTP	74	Request: TYPE I
66.171.116.101	192.168.10.22	FTP	97	Response: 200 Switching to Binary mode.
192.168.10.22	66.171.116.101	FTP	72	Request: PASV
66.171.116.101	192.168.10.22	FTP	118	Response: 227 Entering Passive Mode (66,171,116,101,41,151).
192.168.10.22	66.171.116.101	FTP	104	Request: SIZE /NBG-418N_v2/firmware/zyfw_info
66.171.116.101	192.168.10.22	FTP	75	Response: 213 189
192.168.10.22	66.171.116.101	FTP	104	Request: RETR /NBG-418N_v2/firmware/zyfw_info
66.171.116.101	192.168.10.22	FTP-DATA	255	FTP Data: 189 bytes (PASV) (SIZE /NBG-418N_v2/firmware/zyfw_info)
66.171.116.101	192.168.10.22	FTP	156	Response: 150 Opening BINARY mode data connection for /NBG-418N_v2/
66.171.116.101	192.168.10.22	FTP	90	Response: 226 Transfer complete.
66.171.116.101	192.168.10.22	FTP	76	Response: 500 OOPS:
66.171.116.101	192.168.10.22	FTP	96	Response: vsf_sysutil_recv_peek: no data
66.171.116.101	192.168.10.22	FTP	78	Response:
66.171.116.101	192.168.10.22	FTP	76	Response: child died
66.171.116.101	192.168.10.22	FTP	68	Response:

Figure 16: Zyxel NBG-418N version check

Source	Destination	Protocol	Length	Info
66.171.116.101	192.168.10.23	FTP	86	Response: 220 (vsFTPd 2.2.2)
192.168.10.23	66.171.116.101	FTP	82	Request: USER anonymous
66.171.116.101	192.168.10.23	FTP	100	Response: 331 Please specify the password.
192.168.10.23	66.171.116.101	FTP	81	Request: PASS busybox@
66.171.116.101	192.168.10.23	FTP	89	Response: 230 Login successful.
192.168.10.23	66.171.116.101	FTP	74	Request: TYPE I
66.171.116.101	192.168.10.23	FTP	97	Response: 200 Switching to Binary mode.
192.168.10.23	66.171.116.101	FTP	102	Request: SIZE NBG6602_co/firmware/zyfw_info
66.171.116.101	192.168.10.23	FTP	75	Response: 213 182
192.168.10.23	66.171.116.101	FTP	72	Request: PASV
66.171.116.101	192.168.10.23	FTP	118	Response: 227 Entering Passive Mode (66,171,116,101,42,127).
192.168.10.23	66.171.116.101	FTP	102	Request: RETR NBG6602_co/firmware/zyfw_info
66.171.116.101	192.168.10.23	FTP-DATA	248	FTP Data: 182 bytes (PASV) (RETR NBG6602_co/firmware/zyfw_info)
66.171.116.101	192.168.10.23	FTP	154	Response: 150 Opening BINARY mode data connection for NBG6602_co/
66.171.116.101	192.168.10.23	FTP	90	Response: 226 Transfer complete.
66.171.116.101	192.168.10.23	FTP	76	Response: 500 OOPS:
66.171.116.101	192.168.10.23	FTP	96	Response: vsf_sysutil_recv_peek: no data
66.171.116.101	192.168.10.23	FTP	78	Response:
66.171.116.101	192.168.10.23	FTP	76	Response: child died
66.171.116.101	192.168.10.23	FTP	68	Response:

Figure 17: Zyxel NBG6602 version check

```
Model: NBG6602
FW file: V1.00(ABIL.0)C0.bin
FW version: V1.00(ABIL.0)C0
Revision: 19527
Release date: 2017-05-05
Release note: NBG6602_V1.00(ABIL.0)C0_release_note.pdf
Size: 5835797
```

Figure 18: Zyxel NBG6602 update manifest

```
Model:NBG-418N V2
FW file: V1.00(AARP.9)C0.bin
FW version: V1.00(AARP.9)C0
Revision:
Release date: 2017-12-20
Release note: NBG-418Nv2_V1.00(AARP.9)C0_release_note.pdf
Size: 1695312 bytes
```

Figure 19: Zyxel NBG-418N update manifest

```
if ( fw_ver_now ~= fw_version ) and (fw_version ~= nil) then
        sys.exec("echo newfirmware > /tmp/newfirmware")
else
        sys.exec("echo nonewfirmware > /tmp/newfirmware")
end
```

Figure 20: Zyxel NBG6602 new version check of "online_firmware_check.lua" and its flawed logic

```
if ( fw_ver_now == fw_version ) then
        luci.template.render("expert_maintenance/fw", {on_line_check_fw=3})
else
        luci.template.render("expert_maintenance/fw", {on_line_check_fw=1,
```

Figure 21: Zyxel NBG6602 new version check of "firmware_upgrade.lua" and its flawed logic

5 Discussion and future work

Results obtained in this study strengthen the large amount of anecdotal evidence that SOHO routers contain simple vulnerabilities, and the security level of said devices is not particularly high. However, these results can not be generalized to other SOHO routers or embedded devices and only present these particular cases. Emphasis is placed on this fact so no premature conclusions of SOHO routers in general are made. This study presents illustrative examples of different security issues in firmware update mechanisms and also present the process how these issues can be detected. Exploratory side of this study serves as initial investigation into the topic and knowledge gained can be used in future studies where preferably large scale analysis should be conducted to obtain more general results.

In technical sense all found update mechanisms were similar as they utilized standard network protocols and no proprietary protocols were encountered. General update flow was also highly similar in all analyzed devices. Devices first fetch an update manifest to determine the availability of new updates and new firmware image is then fetched if available.

Found security issues were limited to security issues specific to update mechanisms and more general vulnerabilities such as buffer overflows in the mechanisms itself were not considered in this study. The main issues found from update mechanisms were firmware downgrading, weak manifest and firmware image integrity and unencrypted connections. Similar issues repeated between different devices which shows the lack of attention to security in this device set. The severity of these vulnerabilities can be argued as exploiting network update mechanisms can be difficult. Adversary must be able to intercept the target's traffic and provide highly targeted exploit based on the device. However the high effort comes with high reward as firmware modification attacks result in complete device takeover and instead of underestimating the risk, it should be understood and controlled. To remedy these vulnerabilities existing solutions are available that do not require complex implementation work and neglecting this area shows ignorance towards good security practices, which can result in a loss of reputation when new vulnerabilities are found.

Solving the issue by closing the update mechanism by only accepting cryptographically

signed and verified firmware images protects against firmware modification attacks, but at the same time it blocks open source firmware. Open source firmware provides alternative options to vendor provided firmware and can increase the longevity of a device in the form of updates when support for the official vendor firmware has ended. Using open source firmware does not limit the ability of using signed images for integrity verification in later updates as similar security controls can be implemented in open source firmware as well.

In cases where the firmware update process is carried over unencrypted connection, the transmission is insecure as it is possible to eavesdrop the connection and modify transmitted data. Modifying the transmitted data poses a significant security risk as it can cause users to receive malicious update files. However this does not necessarily make the update process vulnerable to modification attack as the updater can check the authenticity and integrity of an update image after the transmission if a digital signature is used. This method is seen as a sufficient way to verify that downloaded data was not modified during or before the transmission and originates from a trusted source. Nevertheless using unencrypted connections when downloading raises privacy questions as it possible for a third party to see what is being downloaded, but in the case of firmware updates criticality of this issue is debatable. However, even with HTTPS it is still possible to see where data is being transmitted from but the data itself will be encrypted. In some cases it is possible to infer even what data is being downloaded even though HTTPS is used because size of the transfer and connection host is revealed. Similar information leakage issues leading to decreased privacy have been found from other encrypted protocols in IoT device (Apthorpe, Reisman, and Feamster 2017) and VoIP (Wright et al. 2007; Wright et al. 2008; Chang et al. 2008) traffic as well.

The analysis process shows that firmware analysis of embedded devices is not trivial. Analysis depends highly on the initial results of the unpacking phase, where components of the firmware are extracted from the firmware image. This phase is hindered by multitude of file formats, which can sometimes be modified versions of known ones or completely custom-made. Images can also be encrypted to further hinder the process. If firmware image can not be properly unpacked, further analysis is limited significantly. In this thesis 3 firmwares could not be unpacked from total of 6 unique firmwares which reduced the amount firmwares that could be analyzed properly by 50%. Improving unpacking techniques and tools would

allow for more diverse analysis device-wise. Shown unpacking tools were effective against Linux-based firmwares, but these devices form only a limited subset of embedded and IoT devices.

Reverse engineering firmware is often a slow process, which requires considerable amounts of patience, focus and technical knowledge. This highlights the need for advanced tools to help fasten the analysis process. Many of the traditional PC tools are not suitable for firmware analysis because of different processor architectures, file formats and hardware peripherals. The high heterogeneity of embedded and IoT devices increases the difficulty of building practical analysis tools as highly targeted tools require significant development efforts when the large amount of different devices is taken into account. More generic methods suffer from the lack of device specific knowledge, which can decrease the quality of analysis.

Future research ideas include extending the device set to a more heterogeneous set, which should include recent IoT devices and more recent SOHO and home routers. Chosen device set was partially outdated as it contained multiple over 5 year old SOHO routers. However older devices provide a good comparison point to see how security features might have improved over time. Extending the device set would also allow to cover more update mechanisms and network protocols especially when including devices outside of SOHO routers. Large scale analysis of SOHO routers should be conducted to gain more representative results and could also improve methods of analysis as such study would require automation. Automated analysis will be one of the key areas in embedded and IoT security as manual analysis does not scale favorable with the increasing amount of devices. This gives more ideas for future research to develop, improve and test suitable methods for automatically detecting vulnerabilities, software components and sensitive information from embedded and IoT firmware. Going one step further automatic solutions could also offer fix suggestions or in the extreme case even provide automatic vulnerability patching. The scope of this study was limited to firmware update mechanisms, but research should not be constrained and instead should study all the different components found within firmware. Author firmly believes that more research is needed in embedded firmware and device analysis as a whole in the coming years to account for the increase in number and heterogeneity of IoT devices.

6 Conclusion

In this thesis we studied the security of firmware update mechanisms of 12 different SOHO routers. Main objectives of the study were to examine and reverse engineer how firmware update mechanisms are implemented in Commercial-Off-The-Shelf SOHO routers and to discover potential security issues in said mechanisms.

In this study we found that 11 out of 12 devices offered a manual update mechanism over the network with user provided firmware image, while 7 devices were also capable of updating their firmware over various network protocols from an external host. The 7 devices containing network update mechanism were studied in depth utilizing both network level and firmware level file and binary analysis. Devices were found to be using standard network protocols to retrieve firmware update files from vendor's servers. Out of the 7 devices supporting network firmware update mechanisms, 4 devices used FTP as their protocol of choice while 2 devices utilized HTTPS and 1 device employed HTTP.

Using the available tools and resources, we were able to examine 4 (2 using HTTPS, 2 using FTP) devices in more detail. Their firmware was successfully unpacked, which enabled more options to analyze the update mechanism. These firmware update mechanisms were found to have security issues which could lead to firmware modification attacks or denied updates. 2 devices did not verify the authenticity of downloaded firmware which allows attackers to install malicious firmware to the device through the network update mechanism. Remaining 2 devices employed RSA signature based method to verify the firmware image which should in theory protect the device from malicious tampering of the updates. However, validation of full and correct verification of digital signatures is left as future work. Firmware update manifests' integrity was not verified in any of the 4 devices, which makes them vulnerable to forged manifests that can be used to deny further updates, otherwise control the update flow, or implement other types of attacks that are currently unknown.

Results of this study show that security flaws in update mechanisms can be found from common SOHO routers with simple, but time-consuming analysis methods. Literature review of earlier work revealed gaps in studies focusing on security of update mechanisms in

embedded and IoT devices. Automated practical solutions targeted for update mechanism security analysis do not exist and the amount of embedded and IoT devices is only going to increase which highlights the need for more practical and complete analysis solutions. Similarly device vendors should pay more attention to the security of produced devices as attacks exploiting network connected IoT and embedded devices are already reality.

Acknowledgments

We acknowledge grants of computer capacity from the Finnish Grid and Cloud Infrastructure (persistent identifier urn:nbn:fi:research-infras-2016072533).

Bibliography

Aleph One Ltd. 2018. “Yaffs”. Visited on March 12, 2019. `https://yaffs.net/`.

Andersson, Peter. 2017. “SPI Flash File System”. Visited on May 6, 2019. `https://github.com/pellepl/spiffs`.

Antonakakis, Manos, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. “Understanding the mirai botnet”. In *USENIX Security Symposium,* 1092–1110.

Apthorpe, Noah, Dillon Reisman, and Nick Feamster. 2017. “A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic”. *arXiv preprint arXiv:1705.06805.*

Atzori, Luigi, Antonio Iera, and Giacomo Morabito. 2010. “The internet of things: A survey”. *Computer networks* 54 (15): 2787–2805.

Barker, Elaine, and Allen Roginsky. 2019. *Transitioning the Use of Cryptographic Algorithms and Key Lengths.* Technical report. National Institute of Standards and Technology.

Basnight, Zachry, Jonathan Butts, Juan Lopez Jr, and Thomas Dube. 2013. “Firmware modification attacks on programmable logic controllers”. *International Journal of Critical Infrastructure Protection* 6 (2): 76–84.

Bellissimo, Anthony, John Burgess, and Kevin Fu. 2006. “Secure Software Updates: Disappointments and New Challenges.” In *HotSec.*

Bertino, Elisa, and Nayeem Islam. 2017. “Botnets and internet of things security”. *Computer,* number 2: 76–79.

Chang, Yu-Chun, Kuan-Ta Chen, Chen-Chi Wu, and Chin-Laung Lei. 2008. “Inferring speech activity from encrypted Skype traffic”. In *IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference,* 1–5. IEEE.

Chen, Daming D, Maverick Woo, David Brumley, and Manuel Egele. 2016. “Towards Automated Dynamic Analysis for Linux-based Embedded Firmware.” In *NDSS,* 1–16.

Chess, Brian, and Gary McGraw. 2004. “Static analysis for security”. *IEEE security & privacy* 2 (6): 76–79.

Chikofsky, Elliot J., and James H Cross. 1990. “Reverse engineering and design recovery: A taxonomy”. *IEEE software* 7 (1): 13–17.

Corbet, Jonathan. 2008. “AXFS: a compressed, execute-in-place filesystem”. Visited on May 6, 2019. `https://lwn.net/Articles/295545/`.

Costin, Andrei, Jonas Zaddach, Aurélien Francillon, Davide Balzarotti, and Sophia Antipolis. 2014. “A Large-Scale Analysis of the Security of Embedded Firmwares.” In *USENIX Security Symposium,* 95–110.

Costin, Andrei, Apostolis Zarras, and Aurélien Francillon. 2016. “Automated dynamic firmware analysis at scale: a case study on embedded web interfaces”. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security,* 437–448. ACM.

———. 2017. “Towards automated classification of firmware images and identification of embedded devices”. In *IFIP International Conference on ICT Systems Security and Privacy Protection,* 233–247. Springer.

Cui, Ang, Michael Costello, and Salvatore Stolfo. 2013. “When firmware modifications attack: A case study of embedded exploitation”.

Cui, Ang, and Salvatore J Stolfo. 2010. “A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan”. In *Proceedings of the 26th Annual Computer Security Applications Conference,* 97–106. ACM.

DENX. 2019. “U-Boot Source Code”. Visited on May 1, 2019. `https://git.denx.de/?p=u-boot.git;a=blob;f=include/image.h;h=889305cbefdb932c994a290174f51fa850aa0832;hb=refs/heads/master`.

Diffie, Whitfield, and Martin Hellman. 1976. “New directions in cryptography”. *IEEE transactions on Information Theory* 22 (6): 644–654.

Eilam, Eldad. 2011. *Reversing: secrets of reverse engineering.* John Wiley & Sons. ISBN: 9780764574818.

Firmware-mod-kit. 2019. “Firmware Mod Kit”. Visited on May 5, 2019. https://github.com/rampageX/firmware-mod-kit/wiki.

Gilliam, David P, Thomas L Wolfe, Joseph S Sherif, and Matt Bishop. 2003. “Software security checklist for the software life cycle”. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on,* 243–248. IEEE.

Hanna, Steve, Rolf Rolles, Andrés Molina-Markham, Pongsin Poosankam, Jeremiah Blocki, Kevin Fu, and Dawn Song. 2011. “Take Two Software Updates and See Me in the Morning: The Case for Software Security Evaluations of Medical Devices.” In *HealthSec.*

Heath, S. 2002. *Embedded Systems Design.* Elsevier Science. ISBN: 9780080477565.

Hemel, Armijn. 2018a. “Binary Analysis Next Generation”. Visited on February 26, 2019. https://github.com/armijnhemel/binaryanalysis-ng.

———. 2018b. “Binary Analysis Tool”. Visited on May 3, 2019. https://github.com/armijnhemel/binaryanalysis.

Hemel, Armijn, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Dolstra. 2011. “Finding software license violations through binary code clone detection”. In *Proceedings of the 8th Working Conference on Mining Software Repositories,* 63–72. ACM.

Hyppönen, Mikko, and Linus Nyman. 2017. “The Internet of (Vulnerable) Things: On Hypponen’s Law, Security Engineering, and IoT Legislation”. *Technology Innovation Management Review.*

“IEEE Standard Glossary of Software Engineering Terminology”. 1990. *IEEE Std 610.12-1990* (): 1–84. doi:10.1109/IEEESTD.1990.101064.

Independent Security Evaluators. 2013. “Exploiting SOHO Routers”. Visited on April 29, 2019. https://www.securityevaluators.com/casestudies/exploiting-soho-routers/.

Intel. 1988. “Intel Hexadecimal Object File Format Specification”. Visited on May 1, 2019. http://www.interlog.com/~speff/usefulinfo/Hexfrmt.pdf.

Keil. 2018. “FAT File System”. Visited on May 6, 2019. https://www.keil.com/pack/doc/mw/FileSystem/html/fat_fs.html.

Kemp, Darren, Chris Czub, and Mikhail Davidov. 2016. “Out-of-Box Exploitation”. *Duo Security.*

Khan, Moazzam, Zehui Bi, and John A Copeland. 2012. “Software updates as a security metric: Passive identification of update trends and effect on machine infection”. In *MILCOM 2012-2012 IEEE Military Communications Conference,* 1–6. IEEE.

Kocher, Paul, Ruby Lee, Gary McGraw, Anand Raghunathan, and Srivaths Moderator-Ravi. 2004. “Security as a new dimension in embedded system design”. In *Proceedings of the 41st annual Design Automation Conference,* 753–760. ACM.

Koopman, Philip. 2004. “Embedded system security”. *Computer* 37 (7): 95–97.

Lee, Changman, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. 2015. “F2FS: A New File System for Flash Storage”. In *13th USENIX Conference on File and Storage Technologies (FAST 15),* 273–286. USENIX Association.

Ling, Zhen, Junzhou Luo, Yiling Xu, Chao Gao, Kui Wu, and Xinwen Fu. 2017. “Security vulnerabilities of internet of things: A case study of the smart plug system”. *IEEE Internet of Things Journal* 4 (6): 1899–1909.

Lougher, Phillip. 2011. “Squashfs”. Visited on March 12, 2019. http://squashfs.sourceforge.net/.

Microsoft. 2014. “Support for Windows XP ended”. Visited on November 28, 2018. https://www.microsoft.com/en-us/WindowsForBusiness/end-of-xp-support.

Miller, Peter. 2014. “Motorola S-Record”. Visited on April 29, 2019. http://srecord.sourceforge.net/man/man5/srec_motorola.html.

MITRE. 2018. “CAPEC-186: Malicious Software Update”. Visited on April 28, 2019. https://capec.mitre.org/data/definitions/186.html.

Moran, Brendan, Milosch Meriac, Hannes Tschofenig, and David Brown. 2019. *A Firmware Update Architecture for Internet of Things Devices.* Internet-Draft draft-ietf-suit-architecture-05. Work in Progress. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-suit-architecture-05.

"Motorola S-records". 1998. Visited on April 29, 2019. http://www.amelek.gda.pl/avr/uisp/srecord.htm.

Netfilter. 2018. "iptables". Visited on October 25, 2018. https://www.netfilter.org/projects/iptables/index.html.

"OWASP Internet of Things Project". 2018. Visited on October 21, 2018. https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project.

"OWASP Top Ten Project". 2017. Visited on November 26, 2018. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

Papp, Dorottya, Zhendong Ma, and Levente Buttyan. 2015. "Embedded systems security: Threats, vulnerabilities, and attack taxonomy". In *Privacy, Security and Trust (PST), 2015 13th Annual Conference on,* 145–152. IEEE.

Patten, Mildred L., and Michelle Newhart. 2017. *Understanding Research Methods.* 10. Taylor / Francis. ISBN: 9781315213033.

Pitre, Nicolas. 2017. "Cramfs". Visited on March 12, 2019. https://www.kernel.org/doc/Documentation/filesystems/cramfs.txt.

PortSwigger. 2018. "Burp Suite". Visited on October 25, 2018. https://portswigger.net/.

Ravi, Srivaths, Anand Raghunathan, Paul Kocher, and Sunil Hattangady. 2004. "Security in embedded systems: Design challenges". *ACM Transactions on Embedded Computing Systems (TECS)* 3 (3): 461–491.

ReFirmLabs. 2018. "binwalk". Visited on October 15, 2018. https://github.com/ReFirmLabs/binwalk.

Rieck, Jakob. 2016. “Attacks on fitness trackers revisited: A case-study of unfit firmware security”. *arXiv preprint arXiv:1604.03313.*

Shwartz, Omer, Yael Mathov, Michael Bohadana, Yuval Elovici, and Yossi Oren. 2017. “Opening Pandora’s box: effective techniques for reverse engineering IoT devices”. In *International Conference on Smart Card Research and Advanced Applications,* 1–21. Springer.

Simpson, Anna Kornfeld, Franziska Roesner, and Tadayoshi Kohno. 2017. “Securing vulnerable home IoT devices with an in-hub security manager”. In *Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on,* 551–556. IEEE.

Skochinsky, Igor. 2010. “Intro to embedded reverse engineering for pc reversers”. In *REcon conference, Montreal, Canada.*

Statcounter. 2018. “Desktop Windows Version Market Share Worldwide”. Visited on November 28, 2018. `http://gs.statcounter.com/windows-version-market-share/desktop/worldwide/`.

The American Consumer Institute. 2018. “Securing IoT Devices: How Safe Is Your Wi-Fi Router?” Visited on April 29, 2019. `https://www.theamericanconsumer.org/wp-content/uploads/2018/09/FINAL-Wi-Fi-Router-Vulnerabilities.pdf`.

Tjaldur. 2011. “Binary Analysis Tool”. Visited on February 26, 2019. `http://www.binaryanalysis.org/`.

University of Jyväskylä. 2010. “Strategies”. Visited on May 5, 2019. `https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/en/methodmap/strategies/strategies`.

Vahid, Frank, and Tony Givargis. 1999. “Embedded system design: A unified hardware/software approach”. *Department of Computer Science and Engineering University of California.*

Vaniea, Kami E, Emilee Rader, and Rick Wash. 2014. “Betrayed by updates: how negative experiences affect future security”. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* 2671–2674. ACM.

Visoottiviseth, Vasaka, Pongnapat Jutadhammakorn, Natthamon Pongchanchai, and Pongjarun Kosolyudhthasarn. 2018. “Firmaster: Analysis Tool for Home Router Firmware”. In *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE),* 1–6. IEEE.

Wash, Rick, Emilee Rader, Kami Vaniea, and Michelle Rizor. 2014. “Out of the loop: How automated software updates cause unintended security consequences”. In *Symposium on Usable Privacy and Security (SOUPS),* 89–104.

Wireshark. 2018. “Wireshark”. Visited on October 25, 2018. `https://www.wireshark.org/`.

Woodhouse, David. 2003. “Journalling Flash File System 2”. Visited on March 12, 2019. `http://www.sourceware.org/jffs2/`.

Wright, Charles V, Lucas Ballard, Scott E Coull, Fabian Monrose, and Gerald M Masson. 2008. “Spot me if you can: Uncovering spoken phrases in encrypted voip conversations”. In *2008 IEEE Symposium on Security and Privacy (sp 2008),* 35–49. IEEE.

Wright, Charles V, Lucas Ballard, Fabian Monrose, and Gerald M Masson. 2007. “Language identification of encrypted voip traffic: Alejandra y roberto or alice and bob?” In *USENIX Security Symposium,* 3:43–54.

Wurm, Jacob, Khoa Hoang, Orlando Arias, Ahmad-Reza Sadeghi, and Yier Jin. 2016. “Security analysis on consumer and industrial iot devices”. In *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific,* 519–524. IEEE.

Yaghmour, Karim. 2003. *Building embedded Linux systems.* O’Reilly Media, Inc. ISBN: 9780596550486.

Zaddach, Jonas, Luca Bruno, Aurelien Francillon, Davide Balzarotti, et al. 2014. “AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems’ Firmwares.” In *NDSS,* 1–16.

Zaddach, Jonas, and Andrei Costin. 2013. “Embedded devices security and firmware reverse engineering”. *Black-Hat USA*.

Zhang, Weidong, Yu Chen, Hong Li, Zhi Li, and Limin Sun. 2018. “PANDORA: A Scalable and Efficient Scheme to Extract Version of Binaries in IoT Firmwares”. In *2018 IEEE International Conference on Communications (ICC),* 1–6. IEEE.