Expertise
and insight
for the future

Aino Nummikallio

Internet of Things Devices: Case Studies on Security

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

2 April 2019

Author Title Number of Pages Date	Aino Nummikallio Internet of Things Devices: Case Studies on Security 35 pages + 1 appendix 8 April 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Smart Systems
Instructors	Janne Manninen, Project Manager Kimmo Sauren, Head of Smart Systems

The purpose of this thesis is to analyze ethical hacking efforts and penetration testing of embedded IoT devices and their relation to recommended mitigations. Embedded devices present specific security challenges which are best mitigated during design and development phases. Penetration testing is contextualized with current approaches to introducing cyber security in organizations, in particular threat modeling and cyber security relevant regulations. Similar tests were performed on a variety of devices employing the same basic measures to demonstrate the commonly successful attacks which require minimal expertise, and show the common vulnerabilities they exploit. The results are used to derive practical measures for developers and management to employ during the development process to avoid the demonstrated flaws and vulnerabilities. The measures are organized in a simple list format which is intended to become a freely available internal resource for the client company.

Keywords	penetration test, cybersecurity, Tosibox, hackathon, threat modeling, employee guide, cyber security, mitigation, embedded Linux, internet of things

Contents

List of Abbreviations

AEP	Application Execution Platform
ARP	Address Resolution Protocol
CIA	Confidentiality, Integrity, Availability
CVE	Common Vulnerabilities and Exposures
DoS	Denial of Service
DDoS	Distributed Denial of Service
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
LoRa	Long Range (Radio)
LoRaWAN	Long Range (Radio) Wide Area Network
MITM	Man In The Middle
MQTT	Message Queueing Telemetry Transport
NIST	National Institute of Standards and Technology
OS	Operating System
OSINT	Open Source Intelligence
RF	Radio Frequency
RSA	Rivest-Shamir-Adleman cryptosystem

SSH	Secure Shell
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UPS	Uninterruptible Power Supply
USB	Universal Serial Bus
SSL	Secure Sockets Layer
TLS	Transport Layer Security
VAS	Vulnerability Assessment System
VPN	Virtual Private Network
Wi-Fi	Radio Wireless Networking Technology Trademark
WLAN	Wireless Local Area Network
WPA-2	WiFi Protected Access (version 2)
WPS	WiFi Protected Setup

1 Introduction

The subject of this thesis is the conversion of specific penetration test cases into employee guidance as part of security architecture on an organizational scale in an embedded device oriented company. A variety of Internet of Things (IoT) devices are tested using the same basic framework of tests to draw out common shared vulnerabilities. The results of the penetration tests will be combined with the results of an event organized by a customer of the client company, the “TosiHack hackathon” which was a crowdsourced competitive penetration testing event targeting an IoT VPN gateway (Tosibox Oy, 2018).

IoT devices are the primary focus of this thesis and the main product of the client company. They are embedded systems, devices of limited computing power and specific functionality, with sensors or other interfaces that allow them to monitor and interact with their environment without human intervention. These devices are then interconnected into a network and to the general Internet, where they operate in co-operation with both human users and other machines. (Smith, 2017)

The results of the penetration tests are analyzed in order to establish general mitigations for the commonly successful attacks. These mitigations will be arranged into an advisory checklist for employees, which can be found as Appendix 1. By guiding employees in how to anticipate threats and employ protective measures in the form of actionable measures it is hoped that some security breaches may be prevented entirely. The primary purpose of this thesis is to look at security from an organizational, practical perspective which integrates security into the practices of individual developers and focuses on the parts of security architecture which involve technical implementation.

For one particular device which was under the continuous control and development of the client company, some potential hardening measures are applied in order to briefly demonstrate the practical implementation of hardening. The implementation of some measures is further used to add suggested hardening measures to the checklist and shape the format of suggestions.

The purpose of this work is not to focus on the particular methodology of exploits in penetration testing, or indeed to analyze the overall process of penetration testing. Both

of these subjects are thoroughly explored in other works such as those by Weidman and Kim. (Kim, 2014) (Weidman, 2014) The basic penetration testing in this work was within a common framework using a beginner friendly toolset and simple common exploits. This approach is sufficient to demonstrate the simplest attacks faced by devices in the field as well as show how they can be mitigated.

Tests were either whitebox or greybox. Whitebox tests are those in which the engineer has full information on the functioning and construction of the device. This is in contrast with blackbox and greybox tests, terms also used in this work. Blackbox tests are blind tests in which the engineer knows only what they learn by testing, and greybox tests are tests in which the engineer has only partial information. (Khan & Khan, 2012)

The recommendations produced by this work concern the technical portion of the development process of new IoT devices using embedded Linux. The penetration tests conducted in the development of the recommendations are limited in scope, with efforts focused on gaining root Linux user access in on-site whitebox tests of IoT devices. Microcontrollers are excluded from the scope of review, as are non-embedded devices. Also outside of scope are analysis of the effectiveness of the methods used as compared to other potential penetration or security testing methods. Aspects of security architecture not directly related to the development process of new devices such as human resource management or systems administration on-site are out of scope.

2 Organizational Security Principles and Practice

2.1 Organizational Risk Management

Risk management is the process of assessing and containing risks on an organizational scale. (Pompon, 2016) A risk is a potential or hypothetical negative consequence of the current course of action. There are several potential models for cyber security risk management available such as the model suggested by NIST (National Institute of Standards and Technology, 2019) or those referenced by (Mead & Woody, 2017). They will not be examined in detail here, however some parts are briefly described. The gathering of data on security vulnerabilities and mitigations is a component of risk analysis which is a step

in risk management, therefore risk analysis is briefly described below. (Webb, et al., 2014) (Mead & Woody, 2017)

2.1.1 Risk Analysis

Risk analysis is a part of risk management which seeks to assess, categorize, and contextualize the risk within the lifecycle (Pompon, 2016). Like risk management in general, several models are available for risk analysis which serve different perspectives (Mead & Woody, 2017). Some conceptual tools used to evaluate risk and mitigation in this thesis are explored below. These conceptual tools require the addition of data on threats and mitigations, sometimes referred to as threat intelligence. The results of penetration tests in this paper form part of threat intelligence for future cases by examining specific cases representative of the technology developed at the client company.

2.1.2 Human Resources in Cyber Security

Human resource considerations intersect with cyber security considerations. Basic security measures such as account control often correspond to employee roles within the organization (Tevault, 2018). Employee education and training also have an integral role in protecting against human hacking and phishing attacks. Human hacking is intrusion based upon human manipulation, of which phishing is a particular type. Phishing is intrusion based upon convincing people to give up their credentials or other information under a false pretense (Okenyi & Owens, 2007). The threat intelligence gathered in this paper will be used to create a teaching tool for employees. Proper training will improve the coverage of the requested measures in the organization, improving overall security as well as the individual security awareness of the employees (Caldwell, 2013).

2.2 Conceptual Tools

Described below are some of the conceptual tools in use at the client company as part of the wider risk management framework. They are described in order to clarify the model upon which the tools and methods are built.

2.2.1 CIA & STRIDE

The CIA triad is the Confidentiality, Integrity, Availability model of security. Confidentiality, integrity, and availability are considered to be overlapping requirements of a secure device. (ISO/IEC, 2018, pp. 4.2.3, p.12) Confidentiality is defined by controlled access to data, and ensured using methods such as encryption, user access control, and database separation. Integrity is the verification of data as it moves from one point to the next, and the guarantee that data cannot be altered without trace and controls on alterations made. Availability is the overall accessibility and usability of the device, and particularly related to Denial of Service attacks. (Shostack, 2014, pp. 148-157)

STRIDE is a threat classification model. The acronym is composed of Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, Elevation of privilege. (Shostack, 2014, pp. 9-11) The term definitions can be refined for the context of embedded systems as follows. Spoofing refers to falsifying identity, which may be the identity of an account or a source or destination device. (Shostack, 2014, pp. 64-65) Tampering is making changes to information on the device or travelling to or from the device. Repudiation is the denial of responsibility for an attack, i.e. defeating forensic measures after a breach. Countering forensics includes activity such as deleting logs and command history which would allow the attack to be traced. Information disclosure is the leaking of information such as client details or business secrets. Denial of service is a particular type of attack, commonly shortened to DoS which prevents others from accessing a service or using a device by overburdening it and causing it to cease functioning. Elevation of privilege is a common part of attacks and refers to expanding access and capabilities after the initial access to a device is obtained. (Shostack, 2014, pp. 74-75) Connected to the threats of STRIDE are authentication and authorization. Authentication is the verification of the identity of an actor. (Shostack, 2014, pp. 146-148) Authorization is the verification of the right to access of an actor. (Shostack, 2014, pp. 157-159)

2.2.2 Threat Modeling

As part of security by design, or as a less involved alternative to a full security based architecture plan, threat modeling may be used. The steps and their relation to each other are shown in Figure 1 below. Threat modeling is a process wherein the developers of a device and security professionals consider the device as a hypothetical case study.

The case study is divided between development threat modeling and business threat modeling (also modeled as business impact analysis and privacy impact analysis. (Neil, 2018, pp. 33-34)).

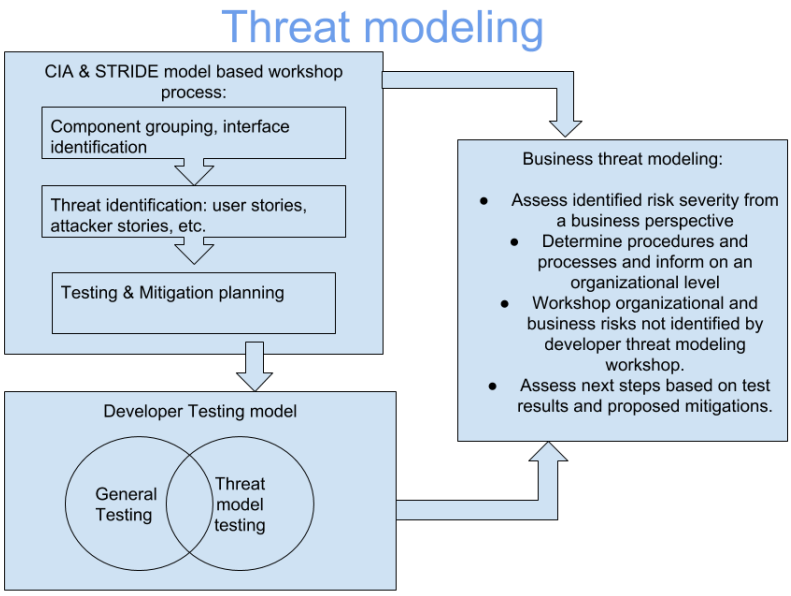

Figure 1. Threat modeling process

The threat modeling diagram process used during this thesis is as follows, with an example of the resulting diagram in Figure 2. Developers are asked to first set out the planned or existing architecture of the device, including the user interfaces, component devices, servers, and connections between parts. Then the developers are asked to section the diagram based on the transitional points of information. The components are grouped together such that those performing the same function in the workflow of the device are one unit. Each time the information is substantially altered or changes hands a new wall is created, referred to as trust boundaries. (Susi, 2018)

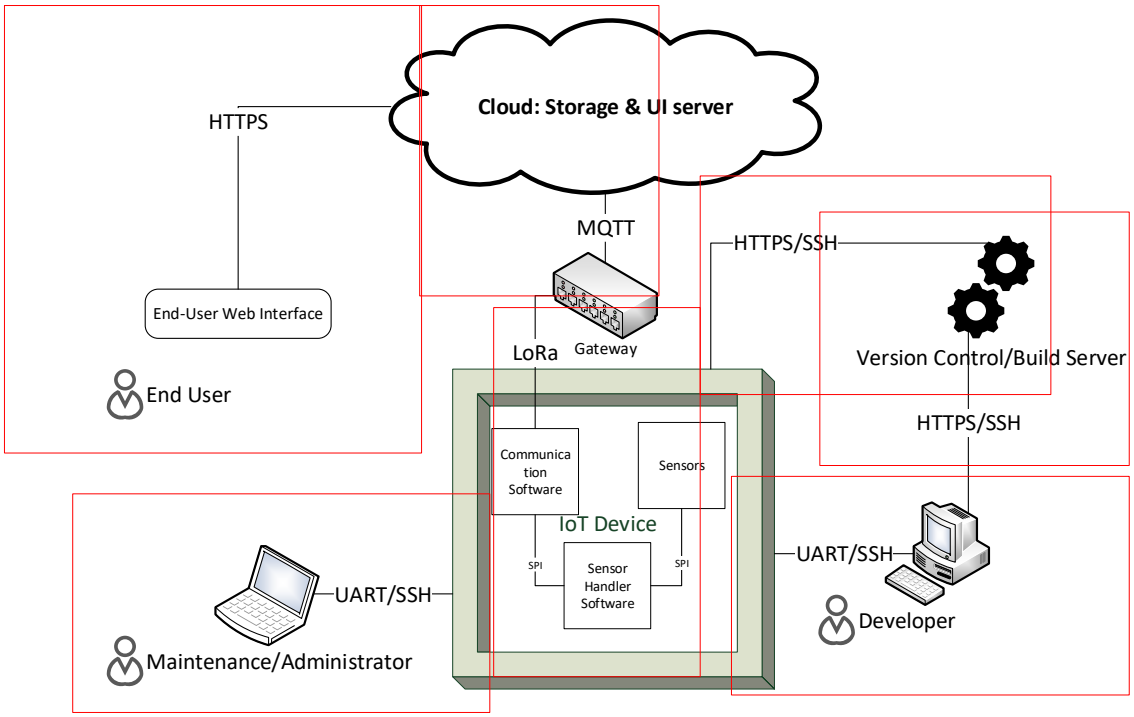

Figure 2. System diagram with trust boundaries

Once the information flow and system architecture is clarified, the developers are asked to imagine threats to the device. Brainstorming includes “malicious user”, “corporate espionage”, “incorrect usage”, and “outside attacker” stories, with particular attention paid to the transitional points between grouped areas as vulnerable points. Finally, the developers are asked to suggest mitigations to the discovered threats. (Shostack, 2014)

The basis of business threat modeling is the consideration of security threats from a financial and managerial perspective. (Shostack, 2014) The advantage of this is twofold. Firstly, the business is able to discuss the consequences of the threats discovered in software threat modeling, e.g. the potential for a costly GDPR breach, and the developers are therefore not required to do so. Secondly and more importantly, business threat modeling can examine the security practices and potential flaws of the organization itself using the context of the particular development process under consideration. Business threat modeling is carried out by the managerial staff of a project, using the findings of software threat modeling. The potential breaches are considered in terms of potential organizational consequences, to evaluate their severity and the necessity and cost-effectiveness of particular mitigations. Organizational processes are clarified by imagining the next steps in the event of a breach, i.e. to whom the breach must first be

reported, who will make required changes to policy, and how will these policy changes be conveyed to the organization. (Susi, 2018)

2.2.3 Hardening

Hardening is the improvement of the security of a device. Hardening comprises configuration changes, software addition and removal, and the improvement of specific software. Hardening comprises a wide variety of practices although it is possible to state some specific guidelines in the case of Linux devices. (Tevault, 2018)

Linux devices employ permissions for files and users which allow varying levels of system access or program execution. Controlling these permissions making sure they are as un-permissive as possible is a core tenet of Linux security, referred to as the principle of least privilege. Access control is employed both as a direct security measure and as preparation for forensics in the event of an attack. (Neil, 2018, pp. 15-17)

On a software level hardening includes code obfuscation, memory access control, and data protection. (Neil, 2018, pp. 221-225) Code obfuscation is the practice of modifying the appearance of the source to make it more difficult to use if it is revealed. Code obfuscation is most useful in the preservation of intellectual property and trade secrets. (Neil, 2018, p. 222)

Hardening can also include material changes to the kernel and operating system of the device. This can include removing features at a kernel level or including additional operating systems features to allow for better security. In particular networking behavior at the kernel level can affect device vulnerability, particularly to DoS attacks (Jaeger, et al., 2011).

Controlling user behavior via device changes can be an effective form of hardening, as some vulnerabilities are caused by misuse. This includes changes to the UI or credential requirements. (Kopecky, 2017) User interfaces can prevent user error as well as prevent the development of dangerous shortcuts in day to day usage of a device (e.g. shared accounts & passwords). Common hardening oriented credential requirements include password length and complexity requirements, banned usernames and passwords, password change frequency and so on. (Tevault, 2018)

2.2.4 Secure Design

Security by design is an idea referred to even in GPDR legislation. (2016, p. Art. 25) In this thesis security by design refers to the development and implementation of cybersecurity mechanisms in the device engineering itself, comparable to the technical implementation focused portions of security architecture as described by (Ross, et al., 2016, pp. 84-156). *"The security community has long argued that security must be designed into systems from the ground up; it can't be "bolted on" to an existing system at the last minute."* (Dirk, et al., 2004, p. 21) Security by design is also referred to as secure design patterns or secure design and is a methodology for securing devices pre-emptively. (Dougherty, et al., 2009)

In the case of embedded devices in particular updating and patching is not possible in the case of every device, and requires additional time and expense on the part of the manufacturer (Smith, 2017). Relying on an incident response model is not always possible for embedded devices because of this difficulty. Security critical patching and updates are a reactive approach to security which seeks to respond only after an exploit is discovered. Security by design attempts to secure against not only known exploits, but to anticipate the methods of the exploiter more generally and counter them in the fundamentals of the product (Smith, 2017). The requirements of security by design are weighted against the requirements of user friendliness and ease of use, with which they may conflict (Kopecky, 2017). If correctly implemented the promise of security by design is that the device will be a significantly less attractive target to a hacker, with a noticeably reduced attack service and limited exploitability.

Some of the measures and choices of security and design can be explained with a hypothetical example of the process for a cloud connected IoT device. Suppose a client purchases design and development services for a cloud connected healthcare device which monitors heart rate, blood oxygen level, and blood pressure. The client provides the relevant legislation and standards related to healthcare devices for their usage environment (e.g. maximum voltages, grounding requirements, personal information security laws). The specific choice of hardware and cloud services are left to the company.

In addition to taking into account the transmission and collection requirements for the data, the hardware designers take into account the integrity principle of the CIA triad in

designing data handling capabilities, as well as the confidentiality and authorization requirements. In doing so they implement tamper proof casing, and hardware which is capable of encryption.

The software designers begin with choosing the cloud service implementation. They consider the physical and software security of their servers, the authorization and privilege divisions of their cloud access, and the redundancy and overall resilience of their cloud. GDPR as well as logging and data verification best practices are taken into account when considering how long data is stored in the cloud, and which data, as well as the archiving process. The decision is made to limit the API of the cloud based on the particular connection source being handled.

Given the basic hardware of the device, the developers begin to design the software. The operating system of the device is decided based upon the hardware and the processing and connection requirements. At this stage solutions such as gateways or sensor-collector device chains are considered.

Supposing each device individually connects to the cloud, it can be assumed they have some running services. The minimum viable product concept is used to devise a list of absolutely necessary services. Thereafter, services required for testing and debugging are listed. Plans are made to include security testing in the test stage of development, and software security best practices are reviewed and modified for the particular project. Based upon the system type and project scope, both the development process software version control and updating are decided, as well as suggestions for post-delivery update security.

During the development process, code review considers not only code correctness and functionality, but also data security and obfuscation. Code suggesting known ‘antipatterns’ is identified and tested and modified or removed accordingly. Antipatterns are common code or process blunders which result in vulnerabilities (Smith, 2017).

In testing the tests most relevant to the running services and device type related risks are run, and their risks and possible harms are evaluated. The product is released to the client with security aspects included in documentation, to ensure proper disclosure.

Security by design is by no means a foolproof approach. The example provided is in fact fairly involved and idealistic compared to the realities of product design and development, and client preferences. As stated above, usability and user friendliness may be considered opposing concerns to security in this approach. Security by design seeks to close as many access methods and remove as many functionalities as possible, to reduce malicious access and the potential for malicious use of these functionalities. However excessive removal of functionality will eventually hobble the device and affect its attractiveness to customers. An example of this is the comparative ease of connecting a device to the Internet and to the cloud which serves it. An absolute security by design approach might necessitate long complex access keys for the device attempting to access the cloud, WPA-2 or better security for WLAN or a requirement for Ethernet internet access, a technician with root access to set up the device, or other security measures which require additional effort on the part of the user. By contrast many devices connect automatically to their backend once powered on and connected to the internet using API keys (which may be vulnerable to collection) and connect to the internet using WPS or connect to any available open network. The second device may be more attractive to the average user, and as such outperform the first in sales.

3 Embedded & IoT Device Security

In recent years the security of Internet of Things devices and embedded systems has become a more prominent issue, following several high profile large scale attacks. IoT devices in particular present their own unique security concerns and constraints. The devices are often in continuous communication with a cloud backend which necessitates internet connectivity, but also have hardware constraints which do not allow the full spectrum of security measures available on fully featured devices (Smith, 2017).

3.1 Legal and Regulatory Considerations

Pursuant to the recent change in EU law, embedded and IoT device security has gained further importance. IoT devices in particular may serve many functions which deal with personal data of the type protected by GDPR and the leaking of such data from a poorly secured device is a financial liability under the new legislation. GDPR legislation requires tight control of personal data, with users being allowed to control and access their own

personal data upon request. Personal data can include anything which can be used to identify a particular person. (The European Parliament and the Council of the European Union, 2016, Art. 24, 15)

3.2 Specialized Embedded Devices

Included in the devices developed at the client company were healthcare, security, and industrial devices. These particular devices have some unique security requirements, as well as unique restrictions on the hardware and software employed. Devices of these categories are governed by specific legislation and national or international standards. For example, the National Institute of Standards and Technology refers to several such industry cybersecurity standards in connection to their general cybersecurity framework guidance. (National Institute of Standards and Technology, 2019) In cybersecurity matters they have particularly stringent requirements which are beyond the scope of this paper to implement. Nevertheless a short discussion of the nature of these devices and implementing security measures in those cases is provided below.

3.2.1 Healthcare Devices

Healthcare devices must be designed to very tight specifications for patient safety. In particular this can result in limitations on broadcasting power and voltages as well as mandatory duplication of communication channels. Healthcare devices are likely to transport or contain sensitive data which may be covered by GDPR protections and as such special attention should be paid to their information security. Also of considerable importance is device availability in the case of a device with a measurable impact on patient wellbeing and hospital efficiency. The cyber security measures of the device cannot compromise the provision of care, providing an additional challenge with regard to user authentication measures. The cyber security requirements of medical devices are well known to the extent that they have been or will be codified into general standards and legal measures.

3.2.2 Security and Alarm Devices

In this category are various alarm systems, including burglar, fire, and personal safety alarms. Devices of this type are can serve as a vector of attack or accident more broadly than other IoT devices due to the risks associated with a malfunction. Availability and accuracy are paramount concerns. Devices should be designed with physical and software fail-safes, and tamper proof casing in just about every case. False alarms are preferable to a lack of alarm, however the accuracy of alarms can be lifesaving in these applications. The integrity of data should be verified as much as possible without impacting availability.

3.2.3 Industrial Devices

The term industrial devices is used to refer to devices used for a manufacturing process or otherwise in a particularly demanding physical environment. Industrial devices in an IoT context are sometimes referred to as Industrial Internet of Things (IIoT). These devices must typically function without intervention for long periods of time, and have high value components produced to greater environmental tolerances than consumer devices. They may be required to pass electrical, explosive, or radiation safety tests. From the perspective of cybersecurity industrial devices can cause large financial losses if successfully compromised and may even be a target of terrorism or cyber warfare. Dependability of the devices must be ensured, often with little possibility of update after installation. (Gupta, et al., 2019) As such these devices particularly benefit from extensive threat and device testing, and require a ‘future proof’ general threat based approach to security. They cannot depend exclusively on the current known vulnerabilities for security assessment, and must have controlled reactions in case of a breach or failure. (Smith, 2017)

4 Device Threat Evaluation Tools and Methods

4.1 Automated scanning

Two different automated scanning systems were tested as part of the reconnaissance phase of penetration testing. The automated scanners tested were F-Secure’s Radar

and an open source alternative, Greenbone OS4 (OpenVas). These systems need not necessarily be included in penetration testing, and can be used for network wide monitoring and vulnerability management (Scarfone, et al., 2008).

4.2 Penetration Testing

Penetration testing can be viewed as a methodology of other approaches to security or as a security approach of its own, depending on the fashion in which it is used. (Wyk, 2007, rev. 2013) The tools and the skill of the tester significantly affect the quality of penetration testing. While many tools are open source and accessible, knowledge of tools and how to use them is essential. (Caldwell, 2011) Penetration testing, shortened to pentesting is the combination of vulnerability analysis with actual execution of exploits (as much as possible) to ascertain the consequences of a successful attack. (Weidman, 2014, p. p.1)

The exact naming of the phases of a penetration test vary from source to source. Some sources use the term “kill-chain” to describe the phases of an exploit, others describe the individual sub-phases (e.g. lateral movement, privilege escalation). The general order described here is an overall summary of various sources, broadly divided into reconnaissance or information gathering, exploit, and forensic countermeasures. (Weidman, 2014) (Wyk, 2007, rev. 2013) (Neil, 2018) (Kim, 2014)

Passive reconnaissance is defined by gathering information about the target from public sources, avoiding direct contact with the target. (Neil, 2018, p. p.258) This is commonly referred to as Open Source Intelligence or OSINT. This involves reading product manuals, running searches against public databases, downloading and examining publicly available software, gathering information from company social media accounts to identify phishing targets and password dictionary entries, identifying company and website aliases and potential spoofing targets, and looking for potential vulnerabilities in the public websites of the company.

Once sufficient information is gathered, active reconnaissance begins. Active reconnaissance involves scanning of the network under attack, identifying possible attack vectors

and target devices on the network, and performing tests based on passive reconnaissance data to gather more specific information.

Active reconnaissance is considered more ‘dangerous’ than passive reconnaissance, because it risks detection by network administrators, and defensive software, and requires some access to the network.

A portion of active reconnaissance is fuzzing. Fuzzing is an approach which can be applied in a variety of ways, but in this context means gathering information on the open ports and network traffic to and from a device, and then sending packets to the device to collect the responses. Fuzzing allows the tester to determine what kind of information a particular device expects on a specific port as well as take advantage of any information leaked in the responses to the packet.

The exploit phase of the penetration test involves attempting to use the information gathered to successfully gain access to the device. This can be done either on the production network or in a test network. On a production network the test can further be done as a surprise to the network administrator, to test their response to a true threat. However the use of the production network requires the ability to guarantee no lasting harm to the environment and surrounding devices, including those used to gain access to the target device. Further, exploit attempts can be simplistic or complex. Simplistic attacking more closely models the conduct of an opportunistic attacker with no particular interest in the device itself, who will favor easy targets and public IP addresses. By contrast complex attacks are deep dives into a particular device, including writing custom exploits and focused repeated attempts.

Forensics countermeasures are used to avoid detection after an exploit is completed and a possible backdoor is opened. These include deleting the exploiting software, tampering with logs, and masking the specific target of the attack by attacking other random targets.

Forensics are used to identify the nature of the attack conducted and mitigate it in future devices. They are not necessarily included in all penetration tests, depending on the wishes of the target owner and the scope of the test. They are however part of a complete incident response plan in the case of a real attack.

The basic test framework used in this thesis was to identify open IP ports and services behind them and then launch suitable exploits, as well as password based attacks on login points. An attack was considered successful if root level shell access was obtained. In some cases DoS attacks were also tested as well as fuzzing results which may indicate unexplored potential attacks. This framework is very similar to some automated attacks faced by devices on the open internet which are outwardly visible and using default or near default configurations, and is similar to the attack framework suggested by Falco, et al. (2018). (Falco, et al., 2018)

4.3 Test Environment

Each penetration test is planned based on the needs of the particular device. In the case of the hackathon, results are compared against the stated priority list. The penetration test itself will be primarily carried out on a Kali Linux virtual machine which exists on a laptop dedicated for the purpose of security testing and isolated from the main network and domain. Kali Linux is a distribution of the Linux operating system which includes several security testing tools by default (Offensive Security, 2019). A security testing network will be used, which is a firewalled IP address space behind a switch, with no enterprise network connectivity. This is to minimize threat to any personal computer and allow maximum separation of the test environment from the organization network. In addition to a Kali Linux VM with open source tools, a Greenbone OS4 (OpenVAS) virtual machine will be used, as well as the proprietary F-Secure Radar tool. Greenbone OS or OpenVAS is a vulnerability scanner which can be run from a dedicated server or as a virtual machine on another machine (Greenbone Networks GmbH, 2019). F-Secure's Radar is a scanning tool which runs from a server and provides a continuing picture of the vulnerabilities of scanned devices as compared to several vulnerability databases (F-Secure, 2019).

The exploit phase of tests was carried out using Metasploit framework in several cases. Metasploit is a tool which contains a database of attacks in a software which can launch them, given certain configuration parameters. (Gupta & Kumar, 2015) Also used were dictionary attacks, which are attacks that use a wordlist as a "dictionary" and try to log in to an interface with different username and password combinations. This was expedited using hydra, a tool which connects to an interface and automatically sends login requests

until the connection is closed or login is successful. (Czagan, 2019) Both the dictionary attacks and the Metasploit based exploits used may be considered brute force attacks, which involve using many attempts of the same attack slightly modified repeatedly to eventually gain access. (Sowmya, et al., 2012)

An exception is the hackathon described below, wherein a Kali Linux LiveUSB system was used on the open Wi-Fi network provided by the hackathon organizers. The LiveUSB system is otherwise the same as the virtual machine, but it runs from within a USB stick.

4.4 TosiBox Hackathon

TurkuSEC members will be the primary "attackers" at the hackathon. Some teams were also composed of security professionals from TosiBox partner organizations. The Tosibox hack target device is the TosiBox Lock 500. This is an under development device intended for enterprise and industrial use. In the topology of Figure 3 below it takes the position of Tosibox Lock. The device will release in 2018, and as such documentation and user manuals are not yet available.

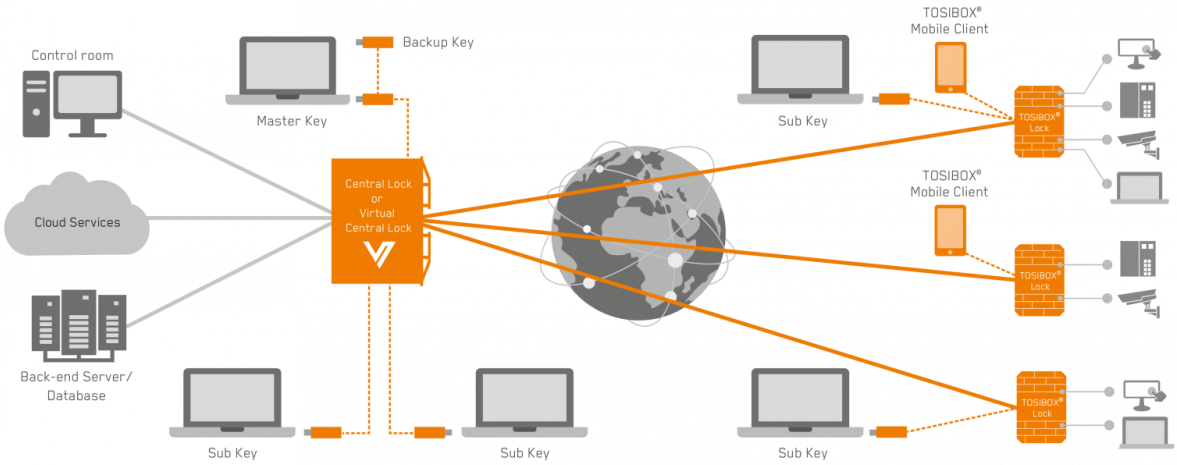

Figure 3. Tosibox Usage Diagram © Tosibox Oy, 2018

The device connects with a user device with a mobile client, SoftKey, MatchMaking service, or virtual Central Lock. These software are all software which authenticate the device to the lock or act as a lock to another device, forming a VPN tunnel endpoint or

passthrough. The traffic inside this tunnel is encrypted between the endpoints which can decrypt the traffic with keys. These keys are the most important information carried by the various devices and software, as evidence by the list of hackathon targets below. (Tosibox Oy, 2018)

Listing 1. TosiHack stated targets in priority order.

- eavesdropping of the VPN connection / defeating encryption
- obtaining VPN authentication RSA keys remotely from any product
- remote root login / remote code execution in Lock / (Virtual) Central Lock from WAN interface
- obtaining RSA keys or key material from a TOSIBOX® Key (token)
- obtaining RSA keys or key material from Mobile Client app
- obtaining RSA keys or key material from SoftKey
- obtaining RSA keys or key material from a Lock 500
- obtaining the password of a TOSIBOX® Key (token)
- obtaining the password of a SoftKey
- impersonation: making Lock / Key / SoftKey / Mobile Client / (Virtual) Central Lock connect with a fake end point
- impersonating a TOSIBOX® device towards MatchMaking service
- obtaining RSA keys or key material from Lock / (Virtual) Central Lock
- remote root login / remote code execution in Lock / (Virtual) Central Lock from LAN / service port
- root login on Lock with physical access to device

Hackathon testing was of the greybox model. A full set of devices, software, and user end usage instructions were provided. The results described in this thesis were those obtained by one of the teams comprised of employees of the client company.

4.5 Device Test Cases

One client owned and one company owned device were tested for comparison. The client device was a network connected UPS, which was then hardened by the client company and returned. Testing was of the blackbox model. The operating system was not revealed by the client, and the user interface was a web interface.

The company owned device was tested with a whitebox model. The company device was a LoRa enabled IoT gateway used to collect data and transfer it to servers. The gateway uses a Linux based operating system. The particular model used in this instance had a web interface using Linux operating system (later upgraded to a purely command line Linux system), additional RF antennas and a LoRa module. The device was connected to the Internet via ethernet during the test, and the LoRa module was operational but not in continuous use.

5 Results

5.1 Tosibox Hackathon Outcome

The hackathon group was a mixture of professionals and beginners in security. Hackathon participation began before the hackathon event with reconnaissance and reverse engineering attempts. Reverse engineering focused on software packages such as the SoftKey application and the mobile client. Primarily the search was for data leaks and fuzzing targets. Secondarily a MITM attack via the software update channel attempt was considered.

Reverse engineering was done by downloading the publicly available software packages and reviewing the raw binaries for patterns matching keys, as well as using proprietary software to attempt to translate the binary first into machine instructions and then into code. Some public key material was discovered as well as potential vulnerabilities in how the mobile application code was structured.

The results of reverse engineering suggested the potential vulnerability in the upgrade channel due to the necessity for some authentication of the firmware package. The upgrade channel of the device required it to be switched to an upgrade mode, from which it might be more vulnerable to other attacks, if it were possible to find this upgrade mode switch.

Hackathon results mostly focused on potential vulnerabilities rather than fully executed exploits, compared to the fully implemented approach of penetration testing. Reverse engineering also played a much more significant role, revealing the possible vulnerabilities of the application in use, as well as data which could potentially be used for fuzzing keys and passwords.

5.2 Embedded Gateway Results

As with the other test cases, social engineering and OSINT were minimized for these device cases, due to the nature of the company-client relationship with the device manufacturer. It should be noted that the device firmware updates are freely available on the manufacturer webpages, as well as a variety of information regarding the device, aiding in possible reverse engineering efforts. However employee and customer information is not advertised on the website, discouraging social engineering.

The penetration test began with general scanning of the cybersecurity network, trying to deduce the IP address and open ports of the target gateway. Also on the same network were the firewall protecting it, a Windows server, another gateway, and the computer used for testing. The first tool used was masscan, which was able to deduce the IP addresses of the live hosts on the network in a more passive fashion. Masscan is an internet scanner which sends TCP packets to probe the network it is used on (Graham, 2019). Thereafter a several scans were run in nmap. Nmap is another open source network scanner which provides a wider set of scans and tests, particularly port and service scanning (Lyon, 2019). Nmap scans provided the IP addresses of the live hosts and their open ports. The open port list allowed the exclusion of the firewall and the Windows server, narrowing the field to the two gateways.

Further nmap scanning provided an OS guess and running services. The OS guess was inaccurate, but the running services were accurate, and combined with the patterns of the IP addresses on the network allowed guessing which IP address belonged to the target.

F-Secure Radar was also used as a comparison point to open source tools on the device. The scan found vulnerabilities primarily related to the SSH version of the device, and was accurately able to show the open TCP & UDP ports. It also provided a more accurate OS guess than the nmap scan. The firmware of the device was the latest version, and there were no immediate updates available to patch the SSH version to a newer one, which was the suggested fix of the SSH vulnerability according to the CVE database. The Common Vulnerabilities and Exposures (CVE) database is an open source database of known vulnerabilities in applications and devices (The MITRE Corporation, 2019). The findings of the OpenVas/Greenbone scanner were almost identical, with differences only in the OS guess results of the scanner, and the names used for the findings in reports generated.

The software of the device was then updated to the version recommended for LoRaWAN server functionality, a common use of this particular gateway. Some vulnerabilities were thereby eliminated, namely those SSL/TLS vulnerabilities related to the running webserver for the AEP user interface. However, notably, the SSH vulnerabilities were unaffected.

A MITM attack via ARP poisoning was used to monitor the traffic to and from the device. ARP poisoning is giving false information to the gateway on a network in order to receive packets bound for another machine (Gibson, 2005).The traffic was further triggered by logging in over SSH from a different machine to check for leaking information.

Based on the results of the active reconnaissance the SSH service had several potential vulnerabilities. The device was vulnerable to DoS attack via the SSH port, using TCP-packets. Further a MITM attack on the SSH service was tested, which succeeded in capturing SSH packets.

Having established that the SSH version was vulnerable to multithreaded login brute forcing, a hand generated wordlist based on common usernames (admin, root etc.) as

well as passwords and password patterns (root, CompanyName, companyname, device-CurrentYear) was used with hydra to attempt login. This attempt was successful and the login credentials were used to create a permanent backdoor. This was done using a cronjob exploit to start a reverse TCP shell. Crontab is a Linux administration tool which causes the device to run a particular program or task at the specified time or at a specified interval. These tasks are called cronjobs (Ubuntu Foundation, 2016). In this instance the program was used maliciously to inconspicuously check the existence of the backdoor and re-establish a connection to the attacking computer if necessary. Further device files were downloaded for future reference.

The exploit was done in a clumsy fashion, with little attempt to mask the connection to the device. As such network log files clearly show the IP address of the attacking computer and the new cronjob is also easy to notice. These particular changes however highlight one important security feature, frequent logfile and cronjob comparison in order to notice changes. These comparisons can be automated (for example, notify administrator if a new IP address connects to an unusual port).

The gateway is a largely closed system without the possibility of installing new software. However some measures can be taken to make it more secure. Firstly, the SSH should be reconfigured to use SSH-keys. The firewall should be configured to reject most requests, and the SSH port should be moved from the default port.

As an alternative a custom firmware was built using the Yocto build system. Yocto is a framework which allows the user to customize a Linux operating system for an embedded device by selecting layers which contain particular limited programs and configurations to create their own version of the Linux operating system (Linux Foundation, 2018). The manufacturer provides a base Yocto layer with the necessary board defines and basic software. This allows the removal of all but the most necessary programs from the device operating system. Furthermore necessary custom software can be wrapped into a Yocto layer allowing the employing of all available code obfuscation and encryption measures.

The example system built was one which contained only MQTT functionality and LoRa functionality over the base operating system. Further the ports were changed from default ports, and a firewall was set up to reject connections to all other ports. The device

includes the ipkg package manager which is a lightweight software which allows for the download and installation of other software on the device after the initial installation (Chetwynd, et al., 2004).The assumed update method of the device is using the ipkg package manager to update the software, including software added by the client company. This can be done by configuring the source list of the ipkg package manager.

A debug port is available on the back of the device and this was used to configure the running system. The modified build by default does not allow remote login, so this debug port must be used.

The device lacks a user application for testing, however as a general principle applications run as part of an embedded solution should not run as root. To this end a user account on the machine with reduced privileges should be set up and used wherever possible.

Further hardening measures are a crucial step with regard to any custom applications developed for the device, but the creation of a software implementation is beyond the scope of this paper. In general terms these include measures such as data encryption, separate compilation and installation via a Yocto Layer, authentication, and logging. This methodology should be used to set up a firewall, protected remote login, and up to date software on the device.

5.3 UPS Results

Passive reconnaissance using the Radar and Nmap tools resulted in open ports for telnet (23), SSH (22), and HTTP (80). An AllegroSoft ROM overflow/misfortune cookie vulnerability was detected, with an OS guess of WindRiver/VXWorks. The ROM overflow/misfortune cookie vulnerability is a buffer overflow flaw which makes it theoretically possible to cause a DoS or execute arbitrary code on the device. It is referred to as AllegroSoft in these results as it has been discovered on devices from that manufacturer (National Institute of Standards and Technology, 2016). The OS guess WindRiver/VXWorks refers to a real time operating system called VXWorks developed by WindRiver (WindRiver, 2019). OpenVAS/Greenbone guessed that the OS was for a HP printer, and found the same open ports and vulnerabilities with the addition of a default login HTTP

interface. The particular OS used by HP printers is not revealed by HP, however it is not the operating system of this device, though it may be similar.

Direct connection to port 80, and 22 was refused. Connecting to the HTTP interface via web browser was successful and revealed a great deal of information about the device, including manufacturer and use case.

To begin a series of basic brute force Metasploit exploits were tested. Notably, the NetGear telnetd exploit resulted in a successful root shell, a command prompt with the maximum level of access to the device. This exploit is a vulnerability found in NetGear devices as well as other embedded and IoT devices which takes advantage of the configuration of the telnet process of the device (Metasploit, 2018). Also possible was login using a dictionary attack against the web login page of the device. This allowed access (via another common default password) to a software update utility. Development of malicious firmware was out of the scope of this device but would theoretically have been uploaded from this interface.

5.4 UPS With New Network Card

Preliminary scanning was done both with the automated OpenVas/Greenbone and Radar tools as well as manually using nmap and Wireshark. As a result open ports 22 (SSH), 443 (SHTTP), and 80 (HTTP) were discovered. All tools returned the same operating system guess of ‘Linux’ with no detail available on exact Linux version. The SSH service visible through port 22 had an up to date version of OpenSSH at time of test.

The HTTP port was selected as the most likely vector of exploitation. HTTP fuzzing was done with a variety of malformed requests. Oversized requests resulted in a closed connection. All other requests were received without reply.

The login page served from the port was devoid of unnecessary information, which provided less basis for dictionary attacks. In many devices these credentials remain unchanged, creating an easy attack vector. Dictionary and offline attacks were not used, however common default credentials were tried. Default credentials are those which are available to the user from user manuals or web pages, which are intended to be used for

first login and then changed (OWASP, 2016). Excessive attempts did not lock the interface, allowing brute forcing.

Attacks using Metasploit framework were launched against the device. The chosen Metasploit modules were unsuccessful. Further probing of the device suggested up to date software and an obscured OS. SSH modules and HTTP modules were preferred for attempted exploits and were unsuccessful. Certificate and key based SSH attacks were not tested. Certificate and key based attacks are those which do not rely on a keyboard based login, instead attacking SSH connection via cryptographic credentials (SSH Communications Security Inc., 2018).

Overall no successful exploit was completed. Other potential attack vectors and security relevant configuration decisions were identified in the implementation of the web interface and the SSH connection but were not tested due to time constraints. These concerns were discussed with the manufacturer along with results from tests on the prior model.

6 Discussion

6.1 Hackathon

Results of the hackathon were somewhat hampered by time constraints. As noted above in the description of the results, reverse engineering proved a surprisingly large part of successful efforts. This was done on publicly available software intended for end users of the product, which happened to contain data to facilitate connection to the cloud backend. Likewise the connection between the gateway and the user endpoint was potentially vulnerable to a MITM attack.

The test device of the hackathon was a fairly classic IoT device, and had the issues of such a device. Namely the need to make software easily available to end users also made it easy to obtain for reverse engineering. Furthermore, the device had connections to several endpoints, some remote, and each connection outwards was a potential attack vector.

Also of interest was the difference between the preparedness and skill level of the teams and between team members. The experience and skill of each penetration tester mattered a great deal, as well as the amount of preparation done before the hackathon. Teamwork was of importance, but most attacks were facilitated by one primary penetration tester focused on a particular attack vector.

6.2 Gateway Test Case

The automated scanning carried out in the passive reconnaissance phase revealed one of the difficulties of embedded systems security. The results of the scanners were so similar because although they used different techniques for OS detection and therefore reported different results, they rely on the same databases for vulnerability intelligence and detection methods. These databases often contain limited information on any particular embedded system OS or hardware compared to conventional computer software and operating systems. The vulnerabilities found were based on Linux SSH vulnerabilities, however there are important differences in how Linux is implemented in an embedded system. It is unwise to assume an embedded Linux system has all the same vulnerabilities or strengths as a desktop Linux OS due to certain differences in the underlying software. Generally embedded Linux devices do not use the bash shell, instead using a more limited sh or ash shell. They tend to have fewer services than a desktop Linux device due to the general lack of a full desktop environment and only specified online services (Linux Foundation, 2018).

In contrast to other test cases the gateway revealed a large amount of information in casual scans. The default credentials are available online and are easily cracked with a dictionary of common usernames and passwords. The login provides root access immediately, with no less privileged users.

The SSH version vulnerabilities were not patched along with other updates to the device. Once login was achieved there were no further protections to dumping the device memory. 'Dumping' device memory is the copying of all the contents of the device memory. The danger of dumping device memory is reverse engineering and the potential loss of keys or other sensitive material stored in the memory (Sianipar, et al., 2018).

While the closed build Linux operating system presents an advantage in terms of software standardization and customization, it complicates the hardening process. The closed buildroot/Yocto style operating system does not allow free installation of packages, which makes the system more secure, but this also prevents the optimization of software and removal of unnecessary packages if the vendor does not reveal the necessary information about the hardware. In this case it was available for use but was not open source. Without source code it is time consuming to assess the implementation of the system, and as such impossible to easily assess the security of the architecture.

The base system provided was used to build a custom system without “unnecessary” packages which had no open ports. The installation of up to date utilities might require customized updated packages not available from manufacturer repos. This was not tested here, but it was noted that the minimum system remains a black box with regard to the packages installed.

6.3 UPS

The UPS test case is an example of the core tenets of hardening. Unnecessary ports were closed, and credentials were changed. These two changes countered the most simplistic attacks. Further the login page was redesigned, which is an important preventative measure. Underlying software was updated and all encryption measures the device could support were enabled. The front page of the device no longer revealed any information other than the manufacturer name and the need for a username and password. The manufacturer states that there are further plans to close port 80 and require HTTPS by default. The option to opt-in to using port 80 and HTTP traffic will be provided to customers, who can choose to use this method if necessary.

6.4 Comparative Analysis of Cases

Common to all of the penetration testing conducted was helpful information leaks from vendor webpages or manuals which led to more focused plans of attack. These leaks were generally related to either passwords or cryptographic keys. In the case of keys safe storage is clearly material to the security of the device, as even partial key material is useful for gaining access.

Also of note was a slight correspondence between open ports and active services and the overall quality of the security of the device, suggesting that a design which seeks to limit ports is more likely to take other security factors into account. Each service or port can provide different potential attack vectors.

Reverse engineering formed a material part of hackathon efforts, and revealed surprising insights. The information left behind in an executable file may be somewhat unpredictable, and may contain keys or other information helpful to an attacker.

Both the original UPS design and the original gateway setup shared a problem with the easy accessibility of the software upgrade utility once initial access was gained. This made the conversion of the attack into a more complex tampering attack trivial. Security should take into account the pivoting steps of an attacker after initial access.

In the case of the gateway the device was very vulnerable to DoS attacks, despite its gateway function. This might have been avoided with stress testing during development. Comparatively the UPS and the Tosibox Lock were resistant, closing connections if excessive packets were sent.

Overall although some of the attacks might have been mitigated by better configuration. Several attacks used default passwords and many attacks leveraged device features or design side effects. These may not have been available if the device was configured less permissively. As such the viewpoint found in literature review which suggested that security is difficult to add after the fact proved true.

In the case of the redesigned UPS organizational factors and organizational interest in security were material in facilitating a much improved product. Many of the risk factors and attacks identified were in fact known to the client company and taken into account in the redesign.

In the case of the gateway, the improved software developed for this thesis was heavily supported by documentation and up to date software and tools provided by the manufacturer. The maintenance of this documentation also serves as organizational memory of the security status of the device.

Some of the services and software in the original gateway software were included for debugging purposes. Debugging tools are services and software which assist the development of the device or other software on it by revealing information about its operation and allowing monitoring. Debugging tools used by developers can be particularly helpful attack vectors if not removed from the final product. They may also cause problems if accidentally discovered by the end user.

6.5 Converting Lessons Learned into Recommendations

The overall takeaway from these tests is the need for an organized and systematic approach to security which encompasses the entire organization and begins with the design of the device. Developers must maintain documentation, avoid behavior which is predictable by automatic tools, test against common vulnerabilities, and remove all unnecessary debug tools, design features for minimum attack surface, and guide the user through the design and the documentation towards correct configuration of the device. These steps take into account several stages of the development process, despite being used by relatively simple attacks.

The organization must understand the information carried by each device and the security risks associated with devices. The potential consequences of these risks should be assessed and organizational memory should be created for risk management process. Developers should be facilitated in passing on this knowledge. The organizational framework and awareness is necessary due to the spread of basis of the vulnerability over the entire development process.

These overall principles are converted into concrete steps which can be taken at appropriate points in the development process in the attached Appendix 1, along with some additional points directed towards managerial staff.

6.6 Next Steps and Future Developments

The next steps of this work after the publication of this thesis are automation of the testing process used and integration into a Continuous Integration (CI) pipeline. A CI pipeline is

a sequence of tests run automatically on a server by various software during development, which then return their results to the developer (ThoughtWorks Inc., 2019). This will provide the most efficient results and allow the development of a further device specific test process. Automation is a key next step of a timely approach to penetration testing because of the marked increase in attacks, in particular simple automated attacks which mirror the exploits attempted in this work. Automated penetration testing and fuzzing will be used as part of the CI pipeline to free time to do more involved and nuanced penetration tests and raise the basic level of security of devices developed at the client company.

The automated test system will also help to measure the effectiveness of the distributed guide. Improvements and expansions to the guide will be developed. Coverage of topics exclusive to microcontrollers, physical security topics, and cryptography advice will be added at a later date after more data is collected.

7 Conclusion

The techniques employed for this thesis were all basic penetration testing techniques. However in the case of at least two devices, those techniques were still able to achieve full access rights remotely. These kinds of simple attacks can be automated and are fairly commonplace given the wide variety of simple to use tools available and ready to download (Offensive Security, 2019). The successful attacks suggest a series of basic mitigations, enumerated in the Handbook (Appendix 1). Based upon the findings of this thesis penetration testers can be a part of the wider security architecture design process in embedded devices. The findings of a penetration test are valuable information for the hardening and forensic measures taken to prevent a security incident.

More importantly, the preventative measures of the attacks are best implemented by each developer. Devices of several different types had similar vulnerabilities, suggesting a lack of the related mitigations. Each of these mitigations can be stated simply and plainly, and they do not require specific knowledge of the related attacks or vulnerabilities. The attacks are however related to changes or missed steps at several stages of the development. Consequently, the application of mitigation is in small steps at various

stages. These steps, as listed, may lead to safer devices by preventing attacks by design.

References

Caldwell, T., 2011. Ethical Hackers: putting on the white hat. *Network Security,* 2011(7), pp. p.10-13.

Caldwell, T., 2013. *Risky business: why security awareness is crucial for employees.* [Online]
Available at: https://www.theguardian.com/media-network/media-network-blog/2013/feb/12/business-cyber-security-risks-employees
[Accessed 3 4 2019].

Chetwynd, J. et al., 2004. *Handhelds.org: Ipkg.* [Online]
Available at: https://web.archive.org/web/20100823030002/http://www.handhelds.org/moin/moin.cgi/Ipkg
[Accessed 8 4 2019].

Czagan, D., 2019. *https://resources.infosecinstitute.com/online-dictionary-attack-with-hydra/.* [Online]
Available at: https://resources.infosecinstitute.com/online-dictionary-attack-with-hydra/
[Accessed 8 4 2019].

Dirk, B., Durfee, G. & Smetters, D., 2004. In Search of Usable Security: Five Lessons from the Field. *IEEE Security & Privacy,* September/October .pp. 19-24.

Dougherty, C. et al., 2009. *Secure Design Patterns,* Pittsburgh, PA, U.S.A.: Carnegie Mellon Software Engineering Institute.

Falco, G., Viswanathan, A., Caldera, C. & Shrobe, H., 2018. A Master Attack Methodology for an AI-Based Automated Attack Planner for Smart Cities. *IEEE Access,* 6(2018), pp. 48360-48373.

F-Secure, 2019. *F-Secure Corporation: F-Secure Radar.* [Online]
Available at: https://www.f-secure.com/documents/10192/1566545/Radar+brochure
[Accessed 2019].

Gibson, S., 2005. *ARP Cache Poisoning.* [Online]
Available at: http://www.grc.com/nat/arp.htm
[Accessed 8 4 2019].

Graham, R., 2019. *Masscan Github Repo.* [Online]
Available at: https://github.com/robertdavidgraham/masscan
[Accessed 6 April 2019].

Greenbone Networks GmbH, 2019. *OpenVAS.* [Online] Available at: http://www.openvas.org/ [Accessed 8 4 2019].

Gupta, A. A. A. et al., 2019. Prevailing and emerging cyber threats and security practices in IoT-Enabled smart grids: A survey. *Journal of Network and Computer Applications,* Volume 132, pp. 118-148.

Gupta, H. & Kumar, R., 2015. *Protection against penetration attacks using Metasploit.* [Online] Available at: http://ieeexplore.ieee.org/document/7359226 [Accessed 8 4 2019].

ISO/IEC, 2018. *27000:2018.* New York, U.S.A.: ISO/IEC.

Jaeger, T., van Oorschot, P. C. & Wurster, G., 2011. Countering Unauthorized Code Execution on Commodity Kernels: A survey of common interfaces allowing kernel code modification. *Computers & Security,* 30(8), pp. 571-579.

Khan, M. E. & Khan, F., 2012. A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. *International Journal of Advanced Computer Science and Applications,* 3(6), pp. 12-15.

Kim, P., 2014. *The Hacker Playbook: Practical Guide To Penetration Testing.* North Charleston(South Carolina): Secure Planet LLC.

Kopecky, S., 2017. *Cyber Paradox from a User's View Point.* London, IEEE, pp. 783-787.

Linux Foundation, 2018. *Yocto Project.* [Online] Available at: https://www.yoctoproject.org/ [Accessed 8 4 2019].

Lyon, G., 2019. *Nmap Reference Guide.* [Online] Available at: https://nmap.org/book/man.html [Accessed 8 4 2019].

Mead, N. R. & Woody, C. C., 2017. *Cyber Security Engineering: A Practical Approach for Systems and Software Assurance.* 1st ed. Boston: Addison-Wesley.

Metasploit, 2018. *Exploit Database: NetGear "TelnetEnable' Magic Packet.* [Online] Available at: https://www.exploit-db.com/exploits/44245 [Accessed 8 4 2019].

National Institute of Standards and Technology, 2016. *National Vulnerability Database: CVE-2014-9223 Detail.* [Online]

Available at: https://nvd.nist.gov/vuln/detail/CVE-2014-9223 [Accessed 8 4 2019].

National Institute of Standards and Technology, 2019. *Cybersecurity Framework: Critical Infrastructure Resources.* [Online] Available at: https://www.nist.gov/cyberframework/critical-infrastructure-resources [Accessed 31 January 2019].

National Institute of Standards and Technology, 2019. *NIST Risk Management Framework.* [Online] Available at: http://csrc.nist.gov/groups/SMA/fisma/framework.html [Accessed 2 4 2019].

Neil, I., 2018. *CompTIA Security+ Certification Guide.* 1st ed. Birmingham: Packt Publishing.

Offensive Security, 2019. *What is Kali Linux?.* [Online] Available at: http://docs.kali.org/introduction/what-is-kali-linux [Accessed 8 4 2019].

Okenyi, P. O. & Owens, T. J., 2007. On the Anatomy of Human Hacking. *Information Systems Security,* 16(6), pp. 302-314.

OWASP, 2016. *Testing for Default Credentials (OTG-AUTHN-002).* [Online] Available at: https://www.owasp.org/index.php/Testing_for_default_credentials_(OTG-AUTHN-002) [Accessed 8 4 2019].

Pompon, R., 2016. *IT Security Risk Control Management: An Audit Preparation Plan.* 1st ed. Seattle: Apress.

Ross, R., McEvilley, M. & Oren, J., 2016. *Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems,* Gaithersburg: National Institute of Standards and Technology.

Scarfone, K., Souppaya, M., Cody, A. & Orebaugh, A., 2008. *Technical Guide to Information Security Testing: Recommandations of the National Institute of Standards and Technology,* Gaithersburg: National Institute of Standards and Technology, U.S. Department of Commerce.

Shostack, A., 2014. *Threat Modeling: Designing for Security.* 1st ed. Hoboken, N. J., U.S.A.: John Wiley & Sons.

Sianipar, J., Sukmana, M. & Meinel, C., 2018. *Moving Sensitive Data Against Memory Dumping, Spectre and Meltdown Attacks.* Sydney, Australia, IEEE.

Smith, S., 2017. *The Internet of Risky Things.* 1st ed. Sebastopol, CA, U.S.A.: O'Reilly.

Sowmya, G., D.Jamuna & Reddy, M. K., 2012. Blocking of Brute Force Attack. *International journal of engineering research and technology,* 1(6), pp. 1-4.

SSH Communications Security Inc., 2018. *SSH.com: Attack: Man-In-The-Middle Attack.* [Online]
Available at: https://www.ssh.com/attack/man-in-the-middle
[Accessed 8 4 2019].

Susi, H., 2018. *Threat Modeling [Internal Documentation].* Espoo: Etteplan Oy .

Tevault, D. A., 2018. *Mastering Linux Security and Hardening.* 1 ed. Birmingham, UK: Packt Publishing.

The European Parliament and the Council of the European Union, 2016. *Regulation (EU) 2016/679 of the European Parliament and of the Council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection).* Brussels, Belgium: The European Parliament and the Council of the European Union.

The MITRE Corporation, 2019. *Common Vulnerabilities and Exposures (CVE).* [Online]
Available at: http://cve.mitre.org/
[Accessed 8 4 2019].

ThoughtWorks Inc., 2019. *Continuous Integration.* [Online]
Available at: https://www.thoughtworks.com/continuous-integration
[Accessed 8 4 2019].

Tosibox Oy, 2018. *TosiHack.fi.* [Online]
Available at: www.tosihack.fi
[Accessed 4 February 2019].

Ubuntu Foundation, 2016. *Ubuntu Cron HowTo.* [Online]
Available at: http://help.ubuntu.com/community/CronHowto
[Accessed 8 4 2019].

Webb, J., Maynard, S. B., Ahmad, A. & Shanks, G. G., 2014. Information Security Risk Management: An Intelligence-Driven Approach. *Australasian Journal of Information Systems,* 18(3), pp. 391-404.

Weidman, G., 2014. *Penetration Testing: A Hands-On Introduction to Hacking.* 1st ed. San Francisco, CA, U.S.A.: No Starch Press.

WindRiver, 2019. *WindRiver VXWorks.* [Online]
Available at: https://www.windriver.com/products/vxworks/#products
[Accessed 8 4 2019].

Wyk, K. v., 2007, rev. 2013. *Build Security In: Adapting Penetration Testing for Software Development Purposes.* [Online] Available at: https://www.us-cert.gov/bsi/articles/best-practices/security-testing/adapting-penetration-testing-software-development-purposes [Accessed 31 January 2019].

Appendix 1

General checklist for IoT device security

- Clarify with the project manager the person responsible for security concerns.
- Include threat modeling and proposed mitigations in the planning phase of development.
- Set up a unified and documented development environment.
- If network functionality testing is necessary, ensure a closed test network is used.
- If automated scanning is available, utilize it regularly during development to track potential vulnerabilities and the efficacy of hardening measures.
- Document the active services and software of the device (list format is sufficient).
- In particular make a note of software and services used for debugging purposes.
- Prefer a 'whitelisting' approach where each open port, service, and application added is included with a justification, rather than including all by default and then closing vulnerabilities.
- Avoid default ports, but remain within the privileged 0-1023 if possible. In particular focus on ports that expect a user configured connection on the service such as SSH or MQTT. These ports can be moved and the users informed with relative ease. This will defeat a variety of simple automated attack tools which scan networks looking for weak targets.
- Include security best practices along with other code standardizations in the development process.
- During code review, refer to an accepted code standard.

- Before release use the port, software, and service list generated previously to remove all non-critical software and debugging tools from the device.

- Incorporate unit testing during development. In addition to resulting in more robust code, unit testing can be used to test the security of individual components of the code.

- During the testing phase, include basic stress testing and fuzzing. Do not assume the device will always be used as intended. Test edge cases such as overly large inputs, wrong formats, and unusual encoding. Purposely ‘misuse’ the device. If possible invite a colleague unfamiliar with the project to use it to get an idea of ‘new user’ mistakes. Particularly for health and industrial use cases, perform simple denial of service attacks on open ports and test the robustness of the device.

- Dump device memory and check for passwords and sensitive data

- If possible force default credentials to be changed on first login. Make the default credentials harder to guess. While defaults must understandably be straightforward, avoid credentials such as: admin, password, root, 12345678, 0000, user, administrator. Consider a randomly generated per device password, as a sufficiently long random alphanumeric string is difficult to guess using automated methods, even if it never changed by the end user. If this is not possible, try to use words that are known mainly within the company and the client such as project_name, over terms that can be found publicly such as CompanyName.

- An attacker with physical access to the device poses a particularly large threat. With this in mind, recommend tamper proof casing and hardware fail-safes.

- Finally, maintain transparency with the client. Internal documentation should be clear with regard to active programs and services on the device. Known vulnerabilities should be documented and checked with the client [in non-public documents]. Client requests and/or approval for security reducing changes and non-action on vulnerabilities should be obtained in writing whenever possible.

- In writing external documentation keep in mind the publicity level of the documentation. If this is not already the case, suggest sectioning customer documentation and public documentation and include information accordingly.

Business Management & Legal specific steps:

1.) Review contracts, in particular warranty clauses on a per-project basis. Devices may be vulnerable to breakage in response to testing, and such testing should only be provided on an as-is basis.

2.) Isolate the company from responsibility for future security breaches. Explicit contractual refutation of responsibility constructive or otherwise for future security flaws and breaches based upon security services provided is preferred. In the negotiation process strive to ensure the client has a realistic view of the possible consequences of security findings and recommendations, and the limitations of such services.

3.) As the first point of contact for your developers, clarify the steps to be taken in the event of a security breach or major threat disclosure.

4.) Managers should pay particular attention to the paper trail and transparency of the project, maintaining wherever possible written evidence of provided security information and advice, and client responses thereto.

5.) While most projects have understandable resource constraints, the addition of code review and CI pipelines to projects can provide significant security advantage. A second developer can provide a vital sounding board for implementation decisions and as well as CI code review can flag issues earlier when they are easier to address.