University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Security Improvements for Object Tracking using LoRaWAN and The Things Network

Stefanie Ziltener
Zurich, Switzerland
Student ID: 11-727-856

Supervisor: Sina Rafati, Dr. Corinna Schmitt
Date of Submission: August 23, 2018

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

CSG
Department of Informatics
Communication Systems Group

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

This thesis documents the security audit and improvement of the security of the OTIoT solution developed at the University of Zurich. The Object Tracking using the Internet of Things (OTIoT) system was developed to provide a possibility to track user's belongings with the help of an Internet of Things (IoT) device. The system environment includes the IoT device (called Tag), the OTIoT backend, and the Pharos website for management purposes. Additionally, the publicly available The Things Network (TTN) architecture was utilized to provide communication between the Tag and the OTIoT backend.
In the security audit, these components and the communication between them is analysed, with a focus on the data flows between components of the system, the exposed Application Programming Interfaces (APIs), and the already supported security. First, a scope is defined to create a clear understanding of the boundary of the security analysis. Then, a threat model is applied to the OTIoT system. The vulnerabilities that are found during the analysis are prioritized according to the probability of a successful exploit and the impact exploitation would have on the system.
Subsequently, requirements to overcome the security issues are presented and applied for the design of a solution. Finally, the chosen security measures are implemented in OTIoT. The evaluation of the improved OTIoT system shows exploits that were used to find vulnerabilities in the beginning of the thesis do not work anymore. Finally, limitations to the presented work are discussed, such as security risks that were not part of the scope and dependencies of components that prevented a more thorough mitigation of security concerns.

Zusammenfassung

In dieser Masterarbeit wird eine Sicherheitsanalyse des OTIoT-Systems (kurz für Object Tracking using the Internet of Things) durchgeführt. Es werden Verbesserungsmassnahmen vorgestellt, implementiert und schlussendlich evaluiert. Das System wurde an der Universität Zürich entwickelt und bietet eine Möglichkeit, Alltagsgegenstände mittels eines IoT-Geräts (kurz für Internet of Things) zu verfolgen. Dafür wurde sowohl ein IoT-Gerät entwickelt, als auch ein Backend und eine Webseite, um das Gerät zu orten und mit ihm interagieren zu können. Für die Kommunikation zwischen dem IoT-Gerät, das Tag genannt wird, und dem Backend wurde die von The Things Network (TTN) bereitgestellte Architektur genutzt.
Der Fokus in der Sicherheitsanalyse des OTIoT-Systems liegt auf der Kommunikation zwischen den Komponenten, auf den verwendeten Application Programming Interfaces (APIs) und den bereits implementierten Sicherheitsmassnahmen. Vor der Analyse wird der Bereich, der analysiert wird, genauer eingeschränkt. Ein Bedrohungsmodell wird erstellt, um einen besseren Einblick in mögliche Sicherheitsrisiken zu erhalten. Die gefundenen Sicherheitslücken werden basierend auf zwei Faktoren priorisiert: Zunächst wird die Wahrscheinlichkeit, dass eine Lücke erfolgreich ausgenutzt wird, geschätzt. Der zweite Faktor, der zur Priorisierung betrachtet wird, ist der Schaden, den ein erfolgreicher Angriff durch die Ausnutzung der Lücke auf das System hätte.
Anschliessend werden Voraussetzungen für die Behebung der Sicherheitslücken definiert. Diese Voraussetzungen wurden für das Design der Lösung miteinbezogen. Schlussendlich werden die gewählten Sicherheitsmassnahmen in das bestehende System eingefügt. Die Prüfung des Systems zeigt, dass die Angriffe, die zu Beginn möglich waren und zur Entdeckung der Lücken geführt haben, nun nicht mehr durchführbar sind. Zum Schluss werden einige Einschränkungen zur bestehenden Arbeit erwähnt, unter anderem Sicherheitsrisiken, die nicht in den Rahmen der Arbeit gehören und Abhängigkeiten von Drittsoftware, die eine vollständige Beseitigung von Sicherheitsmängeln verhindern.

Acknowledgments

First, I would like to express my sincere gratitude to my supervisors, Dr. Corinna Schmitt and Sina Rafati, for the time they dedicated to providing me with support and guidance during my thesis.

I would like to thank Prof. Dr. Burkhard Stiller for the possibility to write this thesis at the Communication Systems Group at the Department of Informatics at the University of Zurich.

Last but not least, I would like to thank my partner and my family, for supporting and encouraging me not only during this thesis, but for the whole time of my studies.

Contents

Chapter 1

Introduction

Tracking the geographical position of an electronic device is a widely used feature nowadays. Examples range from general software solutions, like *Prey* [58], which work across the board for major computer and phone operating systems, to more specific apps such as the built-in functionality *Find My iPhone* [2] to find lost or stolen Apple devices or the *Find my device* [42] functionality of Microsoft, where one needs to be signed into a Microsoft account on the lost device to find it. To track non-electronic devices, for instance bikes or bags, there exist hardware solutions such as *Tile* [81]. Tile is a small, flat tracker object that can be attached on one's own belongings.

Students at the Communication Systems Group (CSG) have built an application called OTIoT - 'Object Tracking using the Internet of Things' - with the purpose of tracking an object's geographical location with an IoT[1] device [66]. The IoT devices that are used with the OTIoT system are called Tags and have both a Global Positioning System (GPS) and a motion sensor embedded. With help of the motion sensor, the GPS module will only collect positional data when the object to which the Tag is attached moves. The system includes a backend environment plus a website where users can manage and interact with their Tags.[8] [39]
To use the OTIoT application, users have to register their Tag on the website called Pharos. The website was developed specifically for this project. The Tag can be set to two modes, either 'armed' or 'unarmed'. In the armed mode, the user can define alarm settings, i.e. how far a Tag can move from the current position before an alarm is generated. An alert message will be sent per email or SMS to the Tag's owner if the object leaves the designated safe radius set by the user. In this scenario, the Tag is set to send continuous data position updates until the alarm is deactivated by an authorized user or the object re-enters the safe area. The alert message contains a link to deactivate the alarm, in case the alarm was left activated by mistake. An example for that would be if the Tag owner forgot to turn off the armed mode and moved the object out of the safe area. [39] [8]

[1] "The Internet of Things (IoT) is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment." [27]

1.1 Motivation

The focus of the OTIoT project was on developing the required tracking functionality and providing a seamless, user-friendly experience. Some of the features that were designed and planned to be integrated - like a general, fine-grained access management - could not be implemented in the prototype because of time constraints [39]. This may have consequences for the security of the whole application. A thorough analysis of the prototype system with focus on security aspects has been defined as future work.
Security is an important consideration for an application that is accessible over the internet. There is also a need to be extra careful because of the specific use case of the OTIoT system. The positional data that is collected with the tracking of belongings is private and can be considered sensitive. Consequently, there should be measures taken to protect it from theft or unauthorized access.

The collected position data must be protected as it can lead a malicious attacker to a valuable object, which is not in the interest of the user. More specifically, one of the use cases that were considered for the development of the application was that of tracking a bike and being alarmed by the system in case of theft. With positional data of a moving transport object like a bike over a longer time period, conclusions could be drawn to create a personal movement profile of the user.
Furthermore, the type of data that is collected and stored in the OTIoT application also falls under the category of personal data that specifically deserves protection in the recently updated General Data Protection Regulation issued to ”harmonize data privacy laws across Europe” [24]. The aim of this regulation is to give natural persons more rights to their personal data and to force organizations that collect and process such data to strengthen protection thereof [24].

1.2 Description of Work

The aim of this thesis is to analyse the existing OTIoT system to find security risks of the application. The focus of the examination is on the data flows between the components of the system, the exposed Application Programming Interfaces (APIs), and the already supported security. Found security vulnerabilities must be prioritized according to the threat they pose to the system and the damage that could potentially arise from exploitation. Subsequently, a solution will be designed and implemented for high priority security issues. Finally, the improved system will be evaluated.

1.3 Thesis Outline

In Chapter 2, an introduction to the OTIoT system and its components is given. Chapter 3 presents the security assessment and provides insight to the security measures already implemented in the project. After that, the design of strategies to mitigate security risks pertaining to the found vulnerabilities are propositioned in Chapter 4. The implementation of the improvements to the system security are examined in more detail in Chapter 5. Criteria for the evaluation are devised from the analysis of the vulnerabilities and consequently applied to the state of the updated OTIoT system in Chapter 6. Finally, Chapter 7 provides a summary of the thesis and an outlook to future work.

Chapter 2

Background

This chapter provides information on the context of this thesis. For the beginning, an introduction to the general architecture of the OTIoT system will be given in Section 2.1. Next, the components of The Things Network (TTN) [78] that are relevant to the understanding of this thesis will be discussed in Section 2.2. Subsequently, the OTIoT backend will be explained in more depth in Section 2.3 and the project code structure will finally be presented in Section 2.4.

2.1 General Architecture

The OTIoT system was created and implemented in two theses by students at the University of Zurich, see [39] and [8]. The following information is derived from these documents, as well as from analysis of additional material like source code and documentation.

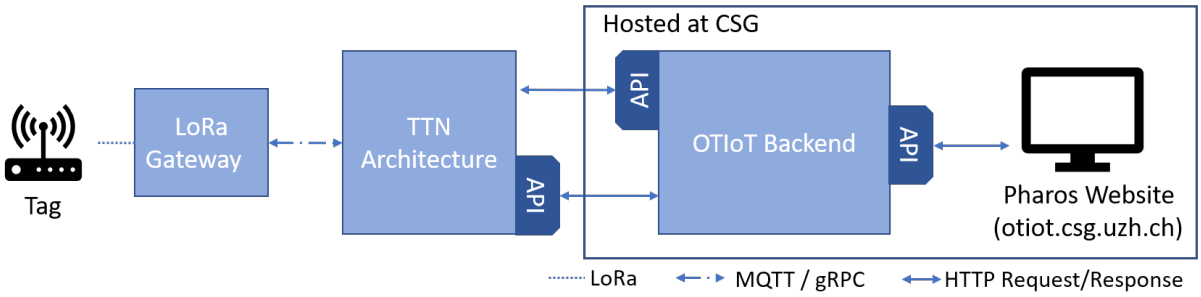

Figure 2.1: General Architecture of OTIoT

The main objectives of OTIoT are to provide a means for tracking physical objects and to alarm users in case of theft. Furthermore, the developers of the system considered additional requirements to increase user acceptance: first, the set-up and installation of OTIoT should be straightforward. Next, it should be easy to use the system and manage the tracking devices via a website. The tracking device, called **Tag**, must be reliable in collecting and sending positional data, robust in different environments as well as energy-efficient and affordable.

Additionally, the user's privacy and the security of the data were considered a priority in the design of the system. A coarse overview of the architecture used in OTIoT can be seen in Figure 2.1. It shows the components that interact with each other and the data flows between them. [39] [8]
The Tag is an IoT device equipped with sensors that can pinpoint the geographical location via GPS and detect movement by sensing acceleration. Furthermore, the Tag features a hardware chip, which can process the collected data and format it to comply with the used radio link protocol. That protocol is defined in the LoRaWAN network protocol specification [38]. LoRaWAN stands for Long Range Wide Area Network (WAN) and is described as "a Low Power, Wide Area (LPWA) networking protocol designed to wirelessly connect battery operated 'things' to the internet in regional, national or global networks, and targets key Internet of Things (IoT) requirements such as bi-directional communication, end-to-end security, mobility and localization services", according to the LoRa Alliance [38].
The LoRaWAN packets are sent by the Tag over a radio link and picked up by one or more **LoRa Gateways** in proximity. These gateways either drop or relay the packet to the **TTN Architecture** via the Internet Protocol (IP), using either Message Queuing Telemetry Transport (MQTT) [43] or gRPC [30], a protocol for Remote Procedure Calls (RPC) [77]. Among other things, TTN provides an infrastructure for the routing of LoRaWAN packets. In this project, it is used to send along all data packets it receives from Tags to the **OTIoT Backend** and to process management requests from the backend. More information about TTN is given in Section 2.2.
The **Pharos Website** uses a Representational State Transfer (REST) Application Programming Interface (API) of the backend to request information it needs to display and to manage data, e.g. users and Tags [25]. The backend and website are both hosted on a machine at the University of Zurich. The setup of the system consists of several Docker containers and will be explained in more detail in Section 2.3. [39] [8]

2.2 The Things Network

TTN describes itself as being "about enabling low power Devices to use long range gateways to connect to an open-source, decentralized Network to exchange data with Applications" [73]. It provides an infrastructure and a software stack to members of its community to build and benefit from an IoT network that is free of charge. Part of the infrastructure needed to work with TTN, like the gateways that are used to send and receive LoRaWAN packets, is provided by the community. TTN communities have developed over the world, an overview can be seen on their website [71]. Among them, the Zurich community is quite active with overall the most contributors and gateways. In this section, the usage of TTN for the OTIoT project will be explained. Moreover, an overview of the architecture will be given. Detailed information can be found in the TTN documentation on their website [73].

The TTN documentation uses the terms **Gateway** and **Application**, as shown in Figure 2.2. TTN is located between the gateways and the applications that rely on TTN to communicate with IoT devices. The IoT devices send packets, which will be received by gateways in their proximity. The gateways then use Wi-Fi, Ethernet or cellular connections to forward these packets to the network [65]. Applications are user-defined programs that utilize TTN. In the case of this project the application is the OTIoT backend which uses TTN to communicate with Tags.
Applications as well as gateways need to be registered on the **TTN Console**, which is integrated in their website. Besides that, the TTN console can be used to define application specific functionality. For example, encoder, decoder, converter, and validator functions can be defined to process the data sent between the IoT devices and the application. Through the console, one can also manage devices (i.e. the Tags for OTIoT), look at the data the devices send, or simulate uplink or downlink data transmission. The same functionality can also be accessed via the APIs, such as the Application Manager API [70], or integrations, such as the HTTP integration [72] that TTN provides. For authentication of access requests to the APIs or integrations, an Access Key is required. This key is unique and generated for each registered application and can be retrieved from the TTN console. [73]

In OTIoT, the Application Manager API is used to add and delete Tag devices. It can be accessed via HTTP, as used in the project, MQTT, or gRPC. The creation of Tags in Pharos calls a function in the OTIoT backend, which in turn sends a request to the Application Manager API. The values that are set in the Tag creation form are required to add new devices to a TTN application over the API. These values are the Tag ID, Dev EUI, and the App Key. The Tag creation form can be seen in a screenshot in the Security Analysis, namely in Figure 3.1. The Dev EUI, or Device Extended Unique Identifier, is used in the LoRa protocol and thus, by TTN to identify a specific device. The Dev EUI is programmed into the hardware chip of the Tag device. The Tag ID is used to assign the registered IoT device object on TTN to the corresponding Tag object in the OTIoT backend. Finally, the App Key is used to derive security keys when the IoT device joins TTN. These keys are used to secure communication between the Tags and TTN. Thus, the App Key and the Dev EUI have to be known to both TTN and the Tag. [73]
Opposed to the managing of Tag devices via Application Manager API, Tag positional data transfer from TTN to the OTIoT backend is done via the HTTP integration of TTN. In the TTN console, it can be defined which part of the incoming IoT device data is included in the HTTP requests body, which HTTP method is used, the value of the authorization header, and finally to which URL the request is sent.

The architecture of TTN can be seen in Figure 2.2. It shows the different components as well as the connections between them in the architecture. In this section, only the parts that are relevant for the project are explained. A more detailed description of the architecture can be found on the respective documentation page of TTN [75].
Starting from incoming, encrypted data from the IoT **Device**, one or more **Gateways** encapsulate the messages and forward them to a **Router** (R). The router handles communication from and to the gateway, or uplink and downlink messages, respectively. As different gateway protocols are supported, bridges are implemented on the router to translate them to the one format TTN uses internally.

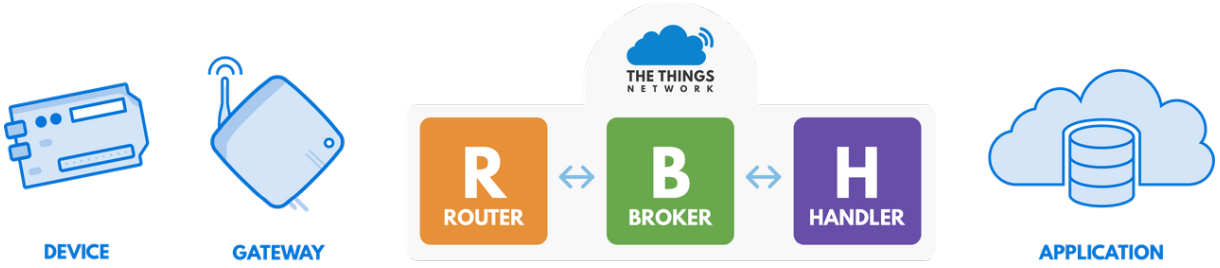

Figure 2.2: Architecture in TTN [68]

Downlink messages to the Tags can only be sent as responses to uplink messages and in a short time window after uplink messages are received. If a downlink message from the **Broker** (B) is received by the router in this time window, the router can forward it to the gateway. As the LoRaWAN protocol uses a radio link, the data can be picked up by several gateways and routers can receive data numerous times. The broker is responsible for de-duplication of incoming uplink data. Another task the broker has is the lookup of the device and application the message belongs to, since device addresses are not unique in TTN. The broker has other jobs, like cryptographic checks to mitigate replay attacks and validating the integrity of the message. Finally, the broker needs to forward downlink data to the applicable router. [75]
In the **Handler** (H), the payload data from the IoT device is decrypted and converted before it is relayed to the application. Consequently, messages that are sent from the application to the IoT device are converted and encrypted in the handler. Subsequently, the message is sent on to the broker. [75]
As one can see from the short description provided above, several security mechanisms are implemented in this architecture. Secrecy of the data is guaranteed through encrypted communication between the IoT device and the handler. Message Integrity Checks (MIC) are applied to mitigate manipulation of the data during the transport. Finally, there are measures implemented to mitigate Man-in-the-Middle (MITM) attacks. [76]

2.3 Existing Backend Structure

The OTIoT backend and the Pharos website are hosted on one machine at the University of Zurich. The previous developers decided to set up the whole backend in a Docker environment [39]. Docker is a containerization tool that is used in the project to separate different OTIoT components and handle communication between them [22]. The Docker containers communicate with each other via HTTP and a shared file space. A more detailed view of the architecture can be seen in Figure 2.3.
The architecture includes two containers that are set up one behind the other to handle incoming and outgoing data. They are both nginx reverse proxy servers [44], but their purpose is slightly different, which is reflected in their names. The first nginx container that incoming traffic passes is titled with **Let's Encrypt Proxy** in the figure. This server holds the TLS certificate issued by the Certificate Authority (CA) Let's Encrypt to enable HTTPS communication [37]. Additionally, the server handles incoming and outgoing HTTPS traffic.

If a request is sent to the server via HTTP, it is automatically redirected to use the HTTPS protocol. Then, the Let's Encrypt proxy forwards traffic as unencrypted HTTP to the **Reverse Proxy** nginx server. This second sever decides with routing rules to which Docker container HTTP requests from the Let's Encrypt nginx server need to be forwarded - which is most of the time the Django backend. Alternatively, HTTP requests for static files (like Tag icons, user avatars, etc.) are directly served from the nginx server. [39]
The Django backend runs on the container titled **Backend**. It contains most of the business logic of the web application, for example serving the REST API calls, such as `POST` requests for creating new Tags. The logic is handled in several Django apps, which build the backend together. Some of the apps include definitions of data models, which are migrated by Django at the setup of the backend; this means that Django automatically creates the corresponding database tables. The web API calls that can be used to interact with the backend, as well as some basic logic how the accessible website pages will look like, are defined in the `views.py` files. Some template designs like the dashboard navigation bar can be found in the Django backend code, although most of the frontend files are created in the respective container and then copied to a shared file system folder. [39]
The backend container can directly access the **Database**, i.e. the PostgreSQL database, or it can store tasks in the **Key-Value Storage** that runs on redis [60]. The tasks are then processed by the **Asynchronous Task Queue** built with celery [4]. As an example, storing positional data arriving from the TTN network is handled this way. The celery workers communicate with the Django backend via the redis container. Finally, most of the JavaScript and HTML files the website displays are created in the **Frontend** container at start-up and copied to the shared file system, from which they are used by the Django **Backend** to serve requests for views. [39]

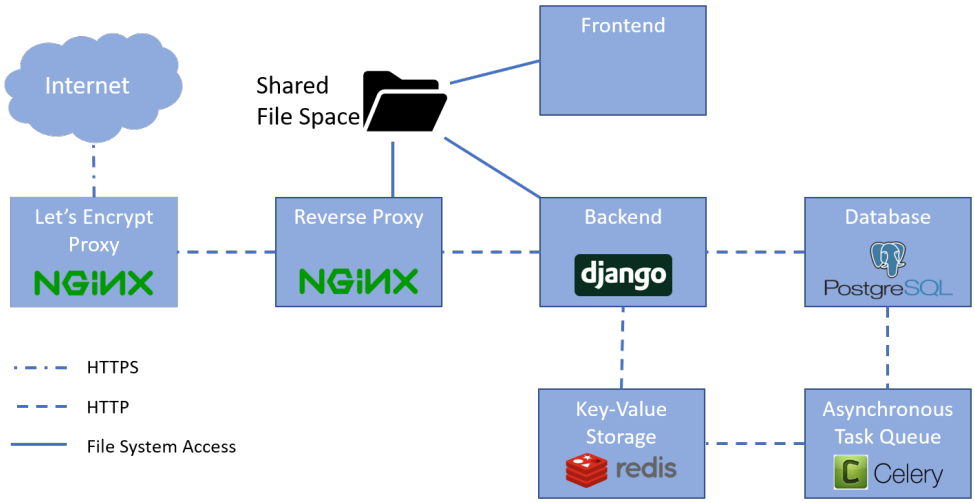

Figure 2.3: Docker Containers of the OTIoT Backend

2.4 Project Code Structure

In this section, the project structure is explained in more detail, with focus on the components that will be of importance in the implementation in Chapter 5. The code for the project is coarsely structured into the components that were explained in Section 2.3. The top level of the project structure contains folders for the Docker containers. Some of the folders are named differently than their container part, for example the Docker backend code can be found in the *web* folder and the Reverse Proxy code is in the *nginx* folder. An important file in the top-level of the project structure is the `docker-compose.yml` [21] file, which is used by Docker to set up the general environment of containers and volumes (that translate to the virtual storage the containers use). An excerpt of the `docker-compose.yml` file showing the definition of the *web* container can be seen in Listing 2.1. The *web* container is linked to the *redis* and *postgis* container as can be seen on lines 12 to 14. Linked containers can communicate with other containers via the exposed ports, which are defined in lines 9 to 11 in Listing 2.1 [21].

```
1  version: "2.1"
2  services:
3    web:
4      restart: always
5      build: ./web
6      container_name: web
7      depends_on:
8        - web_base
9      expose:
10       - 8000
11       - 535
12     links:
13       - postgis:postgis
14       - redis:redis
15     env_file: ${WEB_ENV_FILE}
16     command: ./run_web.sh
17     volumes:
18       - web:/usr/src/app
19       - web_static:/usr/src/app/static
```

Listing 2.1: Docker-compose File of the OTIoT Backend

For some containers, the compose file specifies a script that will be run when the setup has finished. For instance, the specified script for the web container is `run_web.sh`, as seen in line 16 in Listing 2.1. In the `run_web.sh` script, Django-specific management commands are carried out via the built-in tool `manage.py`. One of the tasks that are run is the migration of python models. The commands `makemigrations` and `migrate` are used to track changes made to the Django models and translate the models to a database schema that is then used in the PostgreSQL database [14]. The models in the Django project can all be found in `models.py` files, in their respective app folder.

The different parts of the Django application, located in the *web* folder, are structured into apps, including *alarm*, *dashboard*, *home*, *login_registration*, *master_thesis_backend*, *sensor_data*, *tags*, *user*, *utils*. Furthermore, two specific folders are defined for media files, *public* and *static*. While the content of *master_thesis_backend* consists mainly of settings files, the other apps store code for their particular part of the project. Most of the apps contain the same basic files that work together to provide a certain functionality. Taking the *tags* app as an example, the relevant files are shortly explained below:

Models.py
: Specifies the data structure of `Tag` and `SharedTag` classes and class-specific functions.

Permissions.py
: Defines functions to decide whether access to data should be given or not. Can be applied to classes defined in the `models.py` files.

Serializers.py
: Handles conversions from JSON to python format and back, including data validation functions.

Urls.py
: Lists the URL patterns that can be accessed through the website and the corresponding `views` or functions that are called and handle the business logic.

Views.py
: Holds the code for the functions that are carried out when the URLs are accessed. Defines the HTTP methods that can be called via REST API and includes permissions or other restrictions on access.

As an example of how these files work together in the Django framework, the following steps describe how an incoming HTTP request for a Tag object is processed in the Django backend:

1. The URL path of the request is filtered through the Django `urls.py` file in the *master_thesis_backend* folder. It includes all the `urls.py` files in the project.

2. In the `urls.py` file, the URL is mapped to the `TagViewSet` class in the `views.py` file in the *tags* folder.

3. The `TagViewSet` has the attribute `serializer_class` set to the `TagSerializer`. JSON data from the HTTP request is converted in that class before it is used in the appropriate function. This is explained in more detail in step four.

4. The conversion in the `serializer` is defined by a `Meta` class, which is in this case the `Tag` class from the `models.py` file. The attributes of the `Tag` object, like `name`, `uid`, or `dev_eui` are automatically retrieved from the JSON data. Optionally, validation of the retrieved data can be defined here.

5. The `TagViewSet` class defined in the file `views.py` can then use the `Tag` object instead of raw JSON data. It has default functions inherited from `ViewSet` for create, retrieve, update, and delete (CRUD) functionality, which can be mapped to the HTTP `PUT`, `POST`, `GET`, or `DELETE` methods. The Django framework maps the request type automatically to the matching function.

6. However, before the matched function is executed, a permission check is done. Similar to the `serializer_class`, the `TagViewSet` defines a variable called `permission_classes`. In this variable, the `ExtendedObjectPermissions` class from the `permissions.py` file is referenced.

7. Finally, if all permission checks succeed, the CRUD function is executed, and a HTTP response is sent.

Chapter 3

Security Analysis

In this chapter, the security of the existing OTIoT prototype will be analysed. First, the security model for this thesis and some general security background will be provided in more depth in Section 3.1. This includes the definition of security used in this report and an introduction to key security concepts this thesis will focus on. In Section 3.2, a threat model for the application is created, which makes it easier to argue about security risks to the system. It includes the scope of the assessment, an identification of possible threats to the system, the vulnerability analysis, and a prioritization of the issues found. Lastly, in Section 3.3, there will be a discussion of the security measures that were already implemented in OTIoT.

3.1 Security Background

Security in the context of IT systems is a broad subject that has many facets, so there exist several definitions. The Internet Security Glossary defines security in three ways: "1a. A system condition that results from the establishment and maintenance of measures to protect the system. 1b. A system condition in which system resources are free from unauthorized access and from unauthorized or accidental change, destruction, or loss. 2. Measures taken to protect a system." [67]. Another definition of security, provided by TechTarget, is "the defense of digital information and IT assets against internal and external, malicious and accidental threats. This defense includes detection, prevention and response to threats through the use of security policies, software tools and IT services" [61]. A different approach to security can be seen from Anderson, who defines enterprise information security as "a well-informed sense of assurance that information risks and control are in balance " [1]. Used in this thesis is the one by the International Telecommunication Union Telecommunication Standardization Sector (ITU-T) recommendation X.800, which uses the term security "in the sense of minimizing the vulnerabilities of assets and resources" [35]. To argue about security and vulnerabilities it is helpful to know some basic concepts, which are introduced in Section 3.1.1, as well as having a threat model, which is quickly explained in Section 3.2.3.

3.1.1 Security Concepts

For this thesis, the key concepts inherent to a secure system will be used to argue about security. The Federal Information Processing Standards Publication (FIPS PUB) 199 from the National Institute of Standards and Technology (NIST) defines security objectives for information systems [45]. Specifically, three main objectives are mentioned, i.e. confidentiality, integrity, and availability, which are also generally known as the CIA triad. The following definitions are from the aforementioned NIST specification [45]:

Confidentiality
: Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information. A loss of confidentiality is the unauthorized disclosure of information.

Integrity
: Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity. A loss of integrity is the unauthorized modification or destruction of information.

Availability
: Ensuring timely and reliable access to and use of information. A loss of availability is the disruption of access to or use of information or an information system.

Two further keywords in security that are mentioned in these definitions are authentication and authorization. **Authentication** is defined by the ITU-T X.800 recommendation as ”the assurance that the communicating entity is the one it claims to be” [35]. The definition includes data origin authenticity, which means the source of data sent is the one it claims to be. **Authorization** is done to give access to data or services, or decline it, depending on the rights and privileges of the authenticated user. In case one or more of the CIA properties are not held up, a security breach can happen. More general is the notion of a **threat**, ”a potential for violation of security, which exists when there is an entity, circumstance, capability, action, or event that could cause harm” as defined in the Internet Security Glossary RFC 4949 [67]. In the same glossary, a **vulnerability** is defined to be a concrete flaw in the system that can be exploited. However, not every vulnerability leads to attacks and not all exploited vulnerabilities have the same impact on the harmed asset. This will be relevant to prioritize the vulnerabilities later in this chapter.

In the context of the described system and security model, the existence of a malicious user or attacker who wants to cause harm to the system or get access to data is assumed. There are different ways attackers can try to breach the security of a system. Attacks can be broadly sorted into one of two categories, passive and active attacks. An example of a passive attack could be the **inspection of traffic** between two parties communicating over radio link, potentially obtaining information about the content of the messages. Another example of an attack that can be passive, meaning the traffic is not manipulated, is a passive **MITM** attack [54].

The goal is to intercept a connection, for example an HTTP connection between a client and a server, and split it into two separate connections, with the attacker between. The client and server both unknowingly communicate with the attacker who can read all the transferred data. This attack is generally possible using the HTTPS protocol as well, however, there are a few additional obstacles. In HTTPS, the server has to present a certificate to the client to prove its authenticity, before the secure channel is established, which would mitigate MITM attacks. Browsers warn users if the website they want to access presents an invalid certificate. However, this warning can be ignored by the user. The user can thus proceed to connect with the rogue website.
In the other category, active attacks, the attacker manipulates data or the system in such a way that it could be detected. A MITM can be active as well. It can be used in an exclusively passive way but is often combined with elements of an active attack. Since attackers receive all the data between client and server, they can also manipulate data without the other parties noticing. For example, if a client wants to get the price of an article on a website, the attacker could change the price the server sends to them to a higher value. Furthermore, the attacker could manipulate a request from the client to change the receiver or the amount of a money transfer.
Another active attack is a **Denial of Service** (DoS) attack, where the goal is to disturb or halt the normal function of a service. **Distributed DoS** (DDoS) attacks are a subcategory of DoS attacks, where the source of the attack is not a single point but many different devices. Those attacks have become a daily occurrence over the internet as can be seen on the website Digital Attack Map [29].
For example, a DDoS attack where a botnet consisting largely of IoT devices was used happened in 2016 [80]. The attacked company was Dyn, which provides infrastructure for the hosting of the Domain Name System (DNS) of the internet. As a result, for most of the day of the attack, internet users could not access Twitter, Netflix, PayPal, Reddit, and other popular sites. [35] [50]
To prevent both types of attacks, systems can be secured with several layers of security measures. However, it is impossible to secure a system against all possible attack scenarios. In particular, attacks against availability are often difficult to prevent, as the use case of Dyn shows. That is especially a problem for entities that cannot afford to purchase extensive DDoS protection. For this thesis, there exist time constraints that demand a focus on important security issues. Consequently, the kind of DDoS attack explained above is not covered in this thesis. Other attacks on availability might still be relevant though, see for instance the vulnerability concerning the Tag registration in Section 3.2.3.

3.2 Security Assessment

In this section, the threat model for the security of the application will be devised. The reason to use a threat model is to argue about the current state of the application security, the assessment scope, the identified vulnerabilities, and intended security measures. According to the Open Web Application Security Project (OWASP), thread modelling is used to ”optimize Network/ Application/ Internet Security by identifying objectives and vulnerabilities, and then defining countermeasures to prevent, or mitigate the effects of, threats to the system” [51].

The model that is created in this section is based on and follows the generic steps that are proposed by OWASP, slightly adapted to fit the scope of the thesis better. It consists of five steps. The first step is to define the assessment scope, in which the system will be looked at in all its parts and relevant security aspects for these parts are emphasized. In the next step, possible threats to the system will be discussed, including actors that may cause harm to the system. Following that is a vulnerability analysis and prioritization of the list of found issues. Lastly, the measures that are already implemented in the system to mitigate security risks will be discussed.

3.2.1 Scope

The basic architecture of the OTIoT system was already introduced in Figure 2.1. A more detailed view of the architecture of the OTIoT backend can be seen in Figure 2.3. The Tag communicates over a radio link, using the LoRa protocol for sending data. The signals are picked up by nearby gateways, which will then either accept the data and relay it to the TTN architecture. The data is routed in the TTN architecture and automatically forwarded via HTTPS to the OTIoT REST endpoint at `https://otiot.csg.uzh.ch/sensor-data/ttn/post/`.
The scope for this security assessment is limited according to the time constraint mentioned in Section 3.1.1. For the assessment, the scope extends from the OTIoT backend and website to the communication between the OTIoT API and the TTN API. The website, data flow between components and the API endpoints are of special interest in the assessment.
Most of the taken design decisions, among others the technologies used (i.e. LoRaWAN, TTN, Docker, Django, Vue.js, etc.), from the previous work for the OTIoT system are out of scope for this thesis. The reason for this is that it would be infeasible to evaluate all the used frameworks and software packages for their security benefits as well as drawbacks and decide whether to replace them with other technology. Following that, any hardware implementations (specifically on the Tag) and the Tag's software are excluded from the scope. It is instead of interest how the implementation of the technologies aside from the Tag and the utilized design ideas leave room for security threats, including how these threats can be mitigated.

3.2.2 Threat Identification

To identify threats, different types of attackers the OTIoT system could face and the type of threat they could pose are pointed out. Attacks can come from outsiders, who have nothing to do with the application, application users, or even 'insiders' - people who work with the system. Similar to the limitation in scope, there are concessions to be made in the focus on the type of threats to our system and the type of attackers. To enumerate all types of threats, no matter how likely the scenario in which they happen may be, would not be very helpful. The threats were arranged by the group of perpetrators that would most likely pose the threat. However, the focus lies on the attacks and risks they pose, not on the agents, so they will not be explored in depth.

Insider attack agents would be restricted to the administrator of the website, project members, and members of UZH that have access to the machine the application runs on. Threats that come from them include accidental or purposeful shut-down of the OTIoT backend (thus affecting the availability of the system), deletion or manipulation of the stored data, or leaking of data. At this point of the project, there is nothing done to mitigate these risks specifically. It is assumed that people in the mentioned positions are trustworthy and do not have the motivation to harm the system, nor the carelessness to do so accidentally. When the system runs in productive mode, there is the potential that data is leaked from an inside agent with administrator privileges. The backend could be shut down and made unavailable. However, the backend can be restarted in a short timespan because of its containerized structure. The worst case would be the deletion of the backend environment, which will cause harm as the database is lost in that case.

Another threat from the inside of the system bounds originates from agents of TTN. Their architecture is an inherent part of the OTIoT environment. The threat pertains to the access and leaking of data sent from the Tag objects. Although the usage of the TTN architecture is given and a change thereof out of scope for this thesis, it is worth to be mentioned. The data sent by the Tag is encrypted on the path from the Tag to the TTN Handler, where it is decrypted. This means that TTN has access to the plaintext. In this context it is assumed that TTN is trustworthy. If, at another time, this assumption does not hold anymore, or the risk associated would be too high, it is possible to run and use a private Handler (and other parts of TTN network) instead of using the ones provided by TTN [85].

The other group of agents that could harm our system are outside agents. One threat that any person who has access to the Tag could employ, is a physical attack on the Tag itself. The same holds for the machine that hosts the OTIoT application. If an attacker destroys the Tag or puts it in a situation where it can't send data anymore (e.g. into a Faraday cage), the system will fail to get updates and thus cannot serve its purpose anymore. That is a danger, which is inherent to IoT devices. The same can be said for the situation in which an attacker destroys the physical machine the OTIoT backend runs on. It would require physical barriers and security measures that could only be implemented by the university to mitigate the risk. These are examples of physical attacks that target availability.
A different kind of threat to the OTIoT system is a direct attack on the communication to decrease availability of OTIoT's services. One example for such an attack is a DoS attack or DDoS attacks, as mentioned in Section 3.1.1. DoS attacks may come as HTTP requests from the internet, but they exist in a diverse range of types that cover various layers of the ISO/OSI model. Most of the time, it is difficult to identify the perpetrators. The goal of these attacks is to disturb or halt the services of the application. These types of attacks are difficult to mitigate, and at this time there is nothing deployed for this threat. If the OTIoT system was fully active and in production, the damage of unavailability of the website would be remarkable. One of the motivations for building OTIoT was to track valuable belongings - if that is not possible anymore over a certain time period, the harm caused is two-fold: First, users' belongings could be stolen without them noticing, and second, the trust and thus reason to use the system would be lost.

However, the probability of a DDoS attack on our system is deemed low, but it should be a threat kept in mind when going productive.
The last threat from outsiders that will be discussed in this section is gaining unauthorized access to data belonging to OTIoT, including the manipulation of data. This threat will be important throughout the thesis, as the probability of an attack on a publicly available website is high and the damage done would be great. Loss of trust in the system and possibly loss of valuable belongings of users would be the result. Furthermore, with enough positional data points of moving objects, like bikes, it would be possible to generate movement profiles, which can lead to guesses about the person owning the Tag. As an example, someone who uses their bike daily to get to and from work, would be fairly easy to identify with such data. Leaking this kind of sensitive data can lead to a great loss of privacy for the user.
This threat can be caused by attacks in a multitude of ways. For example, an attacker could try to gain access to a user account with password cracking (e.g. with a brute force attack) and, if successful, look at the user data via the website. The system would not recognize that an unauthorized person accesses the data, since the user has been authenticated with the password. Another way to gain access to a legitimate user's account and data is session hijacking. Once a user is logged in, the requests are authenticated with a session token. If an attacker can get that token they have the same valid access to the user's account like with the possession of the password, as long as the session token is valid.
Access to this data could be retrieved by traffic sniffing or interception attacks. If network traffic is not encrypted, it can be caught and read using a network sniffer. An example for an interception attack would be, as mentioned previously in Section 3.1.1, a MITM attack. A further example of a specific attack that could threaten our system and belongs to this category is Cross-Site Request Forgery (CSRF) [52]. Its goal is to use an already existing, logged-in user to perform possibly malicious actions on a website. More information about these kinds of attacks can be found in [50] and [41].

3.2.3 Vulnerability Analysis and Prioritization

In this subsection, the found vulnerabilities will be presented and prioritized in the following paragraphs A-H. Additionally, the method with which the vulnerability was found will be shown in the paragraphs. Finally, an overview of the vulnerabilities found during the security assessment is shown in Table 3.1
Because the threat of data access and manipulation was estimated to be the most severe in probability of attack and damage caused, a large portion of the used vulnerability testing techniques were targeted on that. The NIST Special Publication 800-115 [64] and the OWASP website [56] both provide extensive information on security assessments, potential attacks and help in how to find vulnerabilities. Also, for the web application, the OWASP Top 10 Most Critical Web Application Security Risks [55] were a useful guide for learning about vulnerabilities and possible threats.
Before the vulnerabilities are presented, it is important to clarify the prioritization metrics used in the thesis. One of the ways this can be done is by rating the potential impact which the exploitation of a vulnerability would have on a system.

The NIST FIPS 199 Standard [45] introduces a potential impact rating according to the CIA security property that is affected by a vulnerability. The rating uses a low score for limited, a moderate score for serious, and a high score for severe or catastrophic adverse effects. An example for a more complex metric is the Common Vulnerability Scoring System (CVSS) [26]. The CVSS version 2.0 is composed of 3 groups of impacts - Base, Temporal, and Environmental - which include a number of metrics [40]. The National Vulnerability Database (NVD) uses the CVSS score to group vulnerabilities into different severity groups [46]. For the CVSS 2.0 rating, the range from 0.0 to 3.9 is considered of a low severity, from 4.0 to 6.9 is a medium severity, and finally, 7.0 to 10.0 is rated as a high severity.
For this thesis, a simple approach was chosen. Based on the risk they pose to the system, the vulnerabilities are deemed a low, medium, or high priority for the implementation of security measures. The risk is assessed based on the perceived probability of an exploit happening, combined with the potential damage that would be caused, i.e. the impact of the exploit. For the probability, the assessment is based on how easy it is to detect the vulnerability with conventional methods, and how difficult it is to carry out the exploit. The rating for the impact is done with the same reasoning as the NIST FIPS 199 Standard uses. The subjective assessment for probability and damage is stated for each vulnerability found in the thesis. For example, a medium probability and a medium damage would lead to a rating of a medium priority. If a vulnerability had severe consequences for the system or the users, but the probability of an exploit was estimated to be low, the priority could still be considered medium.

A Session Management

Through review of the documentation from the work previously done on the OTIoT project and the review of the code, a vulnerability in session management has been found. After a user authenticates in the login procedure, requests to the backend are validated with a session token. If the token is not valid, the request is treated as if an unauthenticated user sent it. The token is stored on the client machine as a cookie so that the browser can use it in further communication with the website. For example, with the Mozilla Firefox browser one can see which sites store cookies by going into the 'Tools' menu, then selecting 'Options'. In the tab 'Privacy & Security', there is a segment for cookies and website data. Selecting 'Manage data' opens a window where the websites that have stored cookies on the machine are listed.
In the project, session management is already in place, but implementation was not done correctly, as the session tokens do not have an expiration time. This means that once a user logs in the session is valid indefinitely, except the user specifically logs out of the website. An exploit for that vulnerability could be possible be if a user logs into the Pharos website on a public computer and forgets to log out. An attacker could use the same machine later and extract the session cookie from the browser, thus getting access to the former user's profile and data in the website. This vulnerability can be categorized in the OWASP Top 10 under "A2 Broken Authentication" [55].

Still, an attacker needs to have such an opportunity to get access to a session cookie, since the HTTP request (including the session token) is encrypted. The complexity for carrying out the exploit is medium, so the probability of an exploit is overall rated to be medium. As it carries the risk of unauthorized access and modification of the one user's data the session cookie was taken from and cannot impact other users, the impact is seen as medium. The vulnerability is thus of a medium priority for this thesis.

B Tag Registration

Another vulnerability that has been found through documentation review lies in the process of the Tag registration. It can be seen in Figure 3.1 that the form demands a name, a Tag ID, a Dev EUI, and an App Key. The Dev EUI and App Key are both necessary to communicate with the Tag, as explained in Section 2.2. The Dev EUI must be unique in an application context for TTN. If a user registers a Tag on the Pharos website with a specific Dev EUI, it will be forwarded to the TTN system. The EUI is then bound to this Tag instance. If another user wanted to register a Tag with the same EUI, this would not be accepted by TTN. While there is no error message shown on the website, the user will not be able to receive any data for that Tag. This can be exploited by a malicious user of the system by hogging Dev EUIs to their account. To be able to receive legitimate data from a Tag, the correct App Key must also be known, so it is unlikely the attacker would receive data from a Tag with a correct Dev EUI only. However, the damage is caused easily because the true owner of the Tag cannot use its Tag anymore for the desired purpose. The probability of the exploit happening is considered to be medium, while the impact is rated as medium.

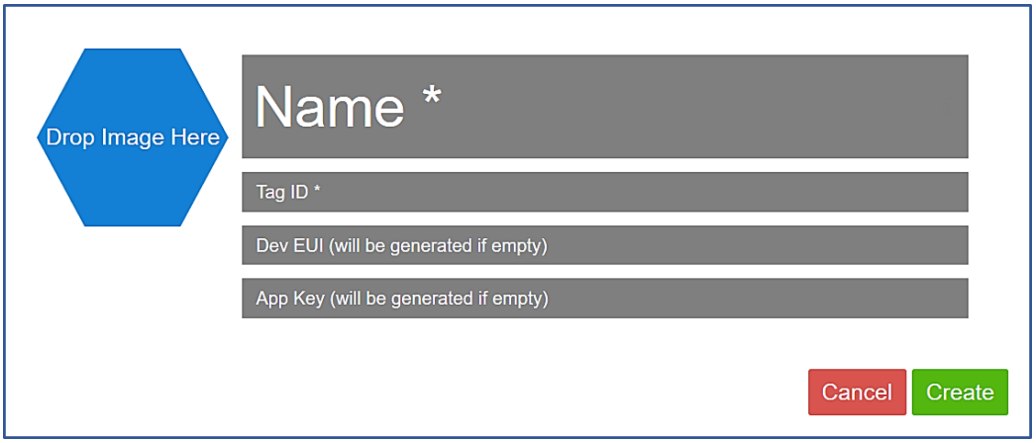

Figure 3.1: Tag Creation Form

The second part of the design flaw is the choice of the Tag ID. This ID is needed to reference positional data sent from the TTN server to our backend to specific Tags and therefore, must be set at registration. The ID must be unique to fulfill the reference function. This constraint is not communicated to the user, who is supposed to choose the ID. If the user's choice is rejected numerous times, it is frustrating and not user friendly.

A concern of the students that previously worked with OTIoT was that with knowledge of the Tag ID, all data of a tag can be seen in OTIoT. That behaviour could not be replicated, but assuming the concern holds, and the right exploit was simply not discovered, there would exist a risk of unauthorized data access by brute-force guessing Tag IDs. By letting the user choose the ID, the probability of values that are easy to guess is higher, which in turn increases the probability for a successful attack. Because the concern could not be validated, the probability and impact are rated as low.
The vulnerability introduced by the Tag registration poses mainly one threat to OTIoT, which is the denial of service to legitimate Tag owners and users of our system. The choice of the identifier of the Tag can potentially lead to an increased risk of unauthorized data access. The vulnerabilities can be exploited without a great effort from the side of attackers and the potential damage can be moderate, so the priority for this vulnerability is deemed medium.

C User Authentication

The Pharos website does have a weak authentication mechanism for login. It is only necessary to provide the user's email and password for standard login, even for the default administrator account. An exploit that could cause an attacker to get access to a user's account is password cracking. One example of a password cracking attack is brute force, where a large amount of automatically generated passwords consisting of letters, numbers, or special characters - configurable by the attacker - are tested to gain access to an account [49]. One existing measure to mitigate the threat of password cracking is called Two-factor Authentication (2FA). It extends the login procedure by a further step to strengthen the protection against account take-overs. The second step has to consist of a different method to prove the user's identity.
In a password-based login scheme, the first factor is a proof of knowledge of a secret. The second factor could be a proof of possession (like owning a credit card or a smartphone) or proof of having a physical feature (like a thumbprint). Two of these factors can be chosen for any 2FA scheme. The Pharos website includes a two-factor authentication scheme, for which the second step is offered when logging in for the first time. However, available methods for the second factor were de-activated because of problems in the project with the phone message and call step. This is not adequately portrayed in the user interface, so the phone option is still available to choose. The GUI even has two options for the phone: call and SMS, as shown in Figure 3.2. However, the setup of both options fails with an error message as seen in Figure 3.3. The only option at that point is to skip the two-factor setup or select the 'Token generator' method. The guess here is that most users will probably find the token generator less attractive than using the phone.

Another reason for skipping the two-factor setup could be that 'Token Generator' does not implicate a tangible method with which authentication can be done, like the 'Google Authenticator' smartphone app. If the preferred authentication choice leads to an error and the two-factor authentication scheme can be skipped as easily as it is the case here, the estimated outcome is that users will not bother to use two-factor authentication. The issues with the authentication increase the chance of a successful account take-over.

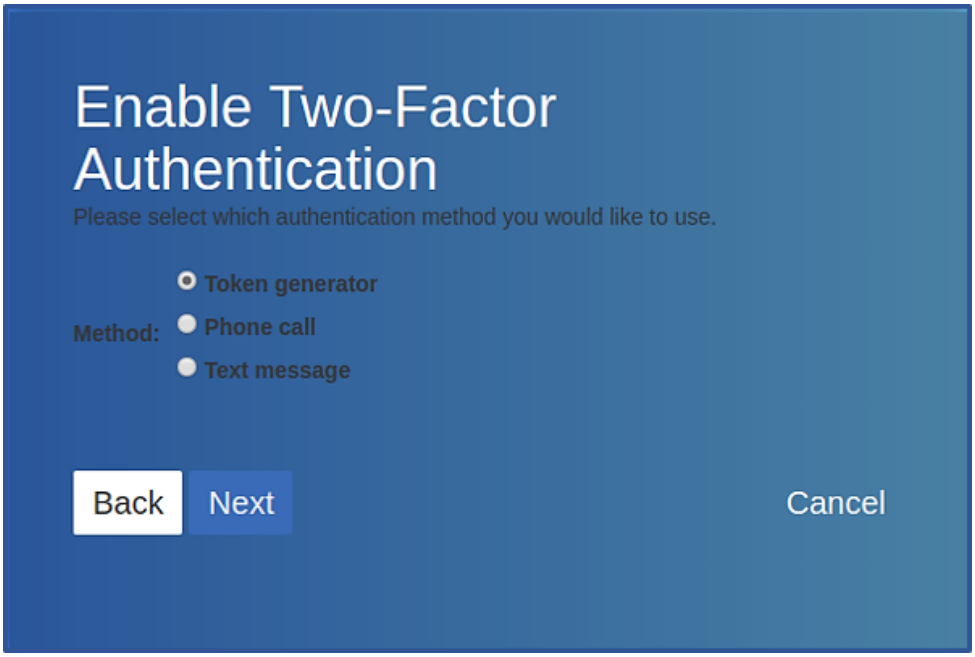

Figure 3.2: Options for Two-factor Authentication in Pharos

The administrator account can also be affected, which in turn means gaining unauthorized access and manipulation rights to all data. The damage in case of a take-over an account is considered to be medium, because the user has a chance to get the account back by resetting the password via their provided email address. The effort needed for the exploit is deemed medium. The priority of this vulnerability is thus set to medium. The OWASP Top 10 include Broken Authentication as one of the issues under place two in their ranking.

D Access Control

The access control to data stored in the database of OTIoT is lacking in several areas. The different ways this issue shows and how it was discovered are all explained in this paragraph.
In the OTIoT backend, the data and user profiles are referenced by IDs, which is reflected in the URLs used to access the data. Users can access data of other users even though this is not intended. Access control should enforce that only the corresponding user can access their ID. The lack of access control and the usage of predictable IDs can be exploited with forced browsing. Following, four examples of successful attacks using this vulnerability will be explained. A step-by-step guide for the exploits can also be found in the appendix in Chapter B.
A prerequisite needed for the success of these exploit is to have a user account on Pharos. Then, all traffic sent between the client machine and the Pharos REST backend was intercepted by a locally running program called Burp Suite [57]. It acts as a proxy and allows to analyse and manipulate HTTP requests and responses. By inspecting this traffic one can learn possible URL routes, which can also be done by an automatic scanner.

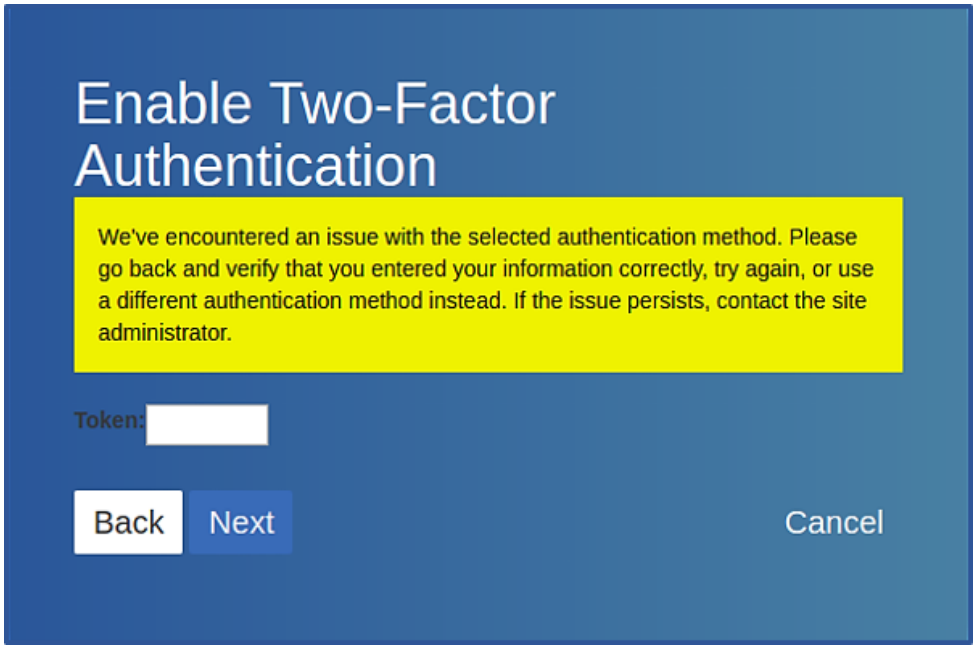

Figure 3.3: Error Message for Phone-based Authentication Methods

All the requests and responses shown in the following examples were captured, edited, and posted to the REST API with Burp, however, they were edited to exclude some of the information, like specific headers, to increase readability. The specific HTTP requests and responses that were used can be seen in the appendix in Chapter C.

Example 1: User enumeration In this example, information about other users could be collected by enumeration of IDs. The HTTP request used to gain information about other users can be seen in Listing 3.1. The '2' in the URL corresponds to the ID, which is increased by one with each new user created. Replacing the ID, one can go through all the users that exist for the website, among others, the administrator user. A response to that `GET` request can be seen in Listing 3.2, depending on the specific user requested.

Using this request and enumerating all possible users led to the collection of information like the username and the email address about them.

```
1 GET /user/rest/user/2/ HTTP/1.1
2 Cookie: csrftoken = AfLT7r4MglNmVLIX8d[...];
3 sessionid = zaxb640wc87xo5zvewu[...]
```

Listing 3.1: HTTP GET Request to Access User Data

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 387

{"url":"https://192.168.99.100/user/rest/user/2/",
"username":"admin","last_name":"","first_name":"",
"email":"pharos.otiot@gmail.com","pk":2,"conf":
 {"url":"https://192.168.99.100/user/rest/user_config/2/",
  "notify_by_email":false,"notify_by_sms":false,
  "has_active_email":"pharos.otiot@gmail.com",
  "has_active_phone":false,"avatar_2":
  "https://192.168.99.100/media/avatar_admin_Xn2vEk1.png"
  }}
```

Listing 3.2: HTTP Response With User Data

Example 2: User data manipulation In example two, an exploit could be carried out that uses the knowledge gained from forced browsing in the first example. The goal was to gain access to any user's account not secured by two-factor authentication, including the administrator account. With small changes to the collected user `PUT` requests, data of existing users, in this case the email address, could be edited. The request for this specific request can be seen in Listing 3.3. The administrator email address has then been set to one controlled by the attacker, which in this example request would be "malicious.user@domain.com".
As the last step, the password reset functionality of Pharos was used. Pharos sends a password reset link for the associated user in the OTIoT system to the email address specified in the input form. If one set the email address of every user to its own and used the password reset function, one would an email with a password reset links for every user with this address. The probability of this attack happening is deemed high and taking over user accounts and their data is easily done. The response to the `PUT` request that can be seen in Listing 3.4 indicates that the request was valid and has been carried out. The impact of the exploit, a permanent and easily repeatable account take-over for users with single-factor authentication, is considered to be high. Therefore, the vulnerability is prioritized as high.

Example 3: Sensor Data Manipulation Another instance of weak access control that has been found through testing out REST API calls pertains to the update of Tag position measurements. This should only be possible through the TTN API, possibly with an extension to the superuser for administrative purposes. With a valid `PUT` HTTP request, it is possible for any user to submit position measurement data for any Tag. To get a valid `PUT` request, information like the Tag ID has to be known or guessed. This attack, which targets the positional Tag data, results in unauthorized data access and is thus grouped into that category. The damage caused can be severe, as malicious users can forge positional data of tags. The backend cannot differentiate between benign and forged data, leading to a total loss of usability of the features of the OTIoT system. Ultimately, users cannot trust the data they see displayed for their Tags.

```
PUT /user/rest/user/2/ HTTP/1.1
Content-Type: application/json;charset=utf-8
X-CSRFToken: AfLT7r4MglNmVLIX8d7L[...]
Cookie: csrftoken=AfLT7r4MglNmVLIX8d7L[...];
sessionid=zaxb640wc87xo5zvewuy[...]
Connection: close

{"url":"https://192.168.99.100/user/rest/user/2/",
   "username":"admin","last_name":"","first_name":"",
   "email":"malicious.user@domain.com","pk":2,"conf":
{"url":"https://192.168.99.100/user/rest/user_config/2/",
   "notify_by_email":false,"notify_by_sms":false,"
   has_active_email":"malicious.user@domain.com","
   has_active_phone":false,"avatar_2":
"https://192.168.99.100/media/avatar_admin_Xn2vEk1.png"
}}
```

Listing 3.3: HTTP PUT Request to Modify User Data

```
HTTP/1.1 200 OK
Content-Type: application/json

{"url":"https://192.168.99.100/user/rest/user/2/",
   "username":"admin","last_name":"","first_name":"",
   "email":"malicious.user@domain.com","pk":2,"conf":
 {"url":"https://192.168.99.100/user/rest/user_config/2/",
  "notify_by_email":false,"notify_by_sms":false,
  "has_active_email":"malicious.user@domain.com",
  "has_active_phone":false,"avatar_2":
  "https://192.168.99.100/media/avatar_admin_Xn2vEk1.png"
  }}
```

Listing 3.4: HTTP Response with Modified User Data

Example 4: Shared Tag A smaller design flaw that leads to private information disclosure is found by using the functionality of Shared Tags. If one decides to share a Tag with another user, one can simply use the 'Share Tag' method that can be found in the Tag detail view. The selection of the user one can share the Tag with is done via a drop-down menu that includes all registered users, as can be seen in Figure 3.4. When the permission level has been selected (i.e. read, edit, read on alarm, admin) and the Tag is shared, the created Shared Tag entry does not show the username but the email address. This leaks information about other users that has not been authorized by that respective user. It is a violation of the user's privacy. The gained information can be used to spam the user's email. A list of all the registered users with their login name is also helpful for further exploits to gain access to user accounts.

The priority of the vulnerability caused by lacking access control is considered high. In the examples explained before, it was shown to be wide-spread through the web application and causing a lot of damage, expecially with the second exploit, which allows to take over accounts. This issue is covered by the OWASP Top 10 under place 5, Broken Access Control.

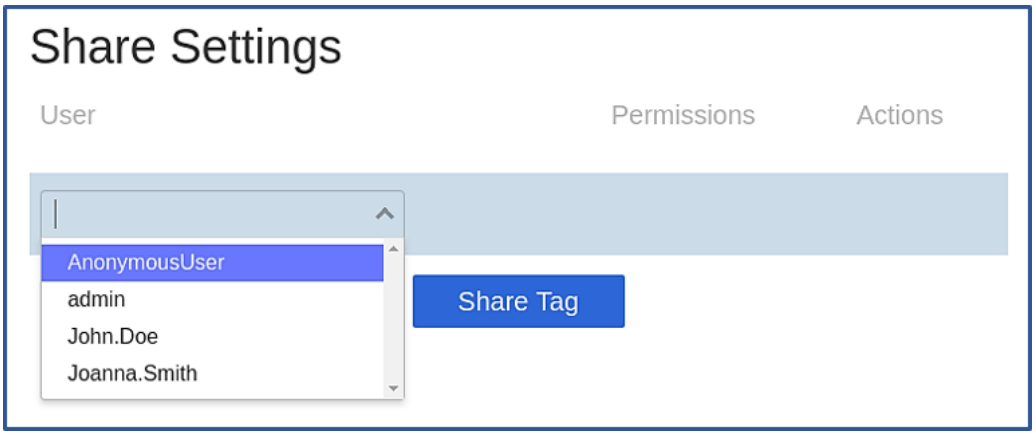

Figure 3.4: Share Tag Functionality in Pharos

E File Upload

The REST API reveals another vulnerability that is introduced with the functionality to upload files to the website. During the creation of a Tag the user has the choice to upload an image file as an avatar for the Tag. This is shown in the GUI of the website with a 'Drop Image Here' placeholder text in the icon space. Through analysis of the REST endpoint with Burp it became clear that file upload can be done without being logged in, the only thing needed is a valid CSRF token in an HTTP `POST` request. In Pharos, a CSRF token is already assigned to a site visitor before the login.
The uploaded file is not restricted to be an image, so it is possible to upload arbitrary content, e.g. an HTML file, through this method. When the file is dragged onto the referenced area on the website, in the background a first `POST` request is made so that the file is instantly uploaded onto the Django web server.

The response includes the relative URL to where the file is expected to be accessible as a temporary resource. Then, when the Tag is created by clicking on the create button, another `POST` request for the storage of the Tag object is sent. In this second `POST` request, the uploaded avatar file is referenced with the URL that was received before. In the backend, the avatar file is converted to a new, renamed Django `ImageFile` and saved in another folder than the originally uploaded file. The path to this file is then stored together with the Tag information. If the `ImageFile` is not a valid image (like a PNG or a JPG), the website will not show the picture in the Tag view because the conversion fails.
However, the temporary file created after the first `POST` request has not been deleted and is still accessible under the URL returned in the `POST` request. In the code and during experiments, nothing has been found that indicates the temporary file is ever deleted. In the Django documentation, it is repeatedly stated that the media upload functionality they use is not following security best practices [16]. There are not sufficient checks with regards to the content of the uploaded file. Another example they mention is the upload of a malicious HTML file with a valid PNG header. That file would pass a verification test, according to the documentation, and could introduce HTML code on the website.
Through the inclusion of the temporary file in the website and the possibility to access the file, cross-site scripting (XSS) can be achieved, e.g. if as mentioned one were to upload an HTML file. This specific form of XSS is categorized under Stored XSS. In the OWASP Top 10, XSS is ranked in place 7. Potentially, remote code execution can happen in the client's browser, e.g. if a HTML file with JavaScript code is uploaded and a user is tricked into clicking on the link that accesses that file. That exact example has been carried out as a proof of concept. The HTTP request can be seen in Listing 3.5. The file uploaded with the sample request was a simple HTML file with a JavaScript alert function.

```
1  POST /utils/file-upload/ HTTP/1.1
2  Content-Type: multipart/form-data;
3  boundary=---------------------------8970094813384053[...]
4
5  -----------------------------8970094813384053[...]
6  Content-Disposition: form-data; name="image";
7  filename="helloworld.html"
8  Content-Type: text/html
9
10 <p>Hello World</p>
11 <script>alert(1)</script>
12 -----------------------------8970094813384053[...]--
```

Listing 3.5: HTTP POST Request for File Upload

It is also possible to upload a file that is downloadable by the browser, e.g. an .exe file. For the user, that content comes from a website that is trusted (i.e. it has a valid TLS certificate and the browser displays a green padlock or a similar symbol, implying a secure connection). It is thus assumed that a download would be trusted and possibly accepted, leading to the risk of introducing malware to the user's machine.

With this method, it is also possible to create one's own content that is hosted on Pharos, for example a whole new one-page website. This enables an attacker to launch phishing attacks, which aim to steal user credentials, other sensitive information, or steal the CSRF token and use it for further CSRF attacks.
In the analysis of this vulnerability, a proof-of-concept exploit as explained above was carried out. With the file upload function, an HTML file with JavaScript code was uploaded. The file loads a page that looks like the Pharos login page. However, the file was changed so that when hitting the login button, the username and password values are read out and shown in an alert message, as can be seen in Figure 3.5. Instead of showing an alert, that information could be sent to the attacker and used to compromise the user account. The URL to the fake login page could be sent via email to a potential victim.
As explained, the potential damage can be severe for the user, the user's machine, and the trust one sets into the Pharos website. This is even the case if the uploaded file is only temporarily available on the website. Moreover, there are automated tools that can detect such vulnerabilities, for example the Burp Suite provides such functionality [57]. Combined with the low complexity of the exploit, this leads to a high probability rating. Considering the high impact and probability, it was decided that the remediation of this issue is of a high priority.

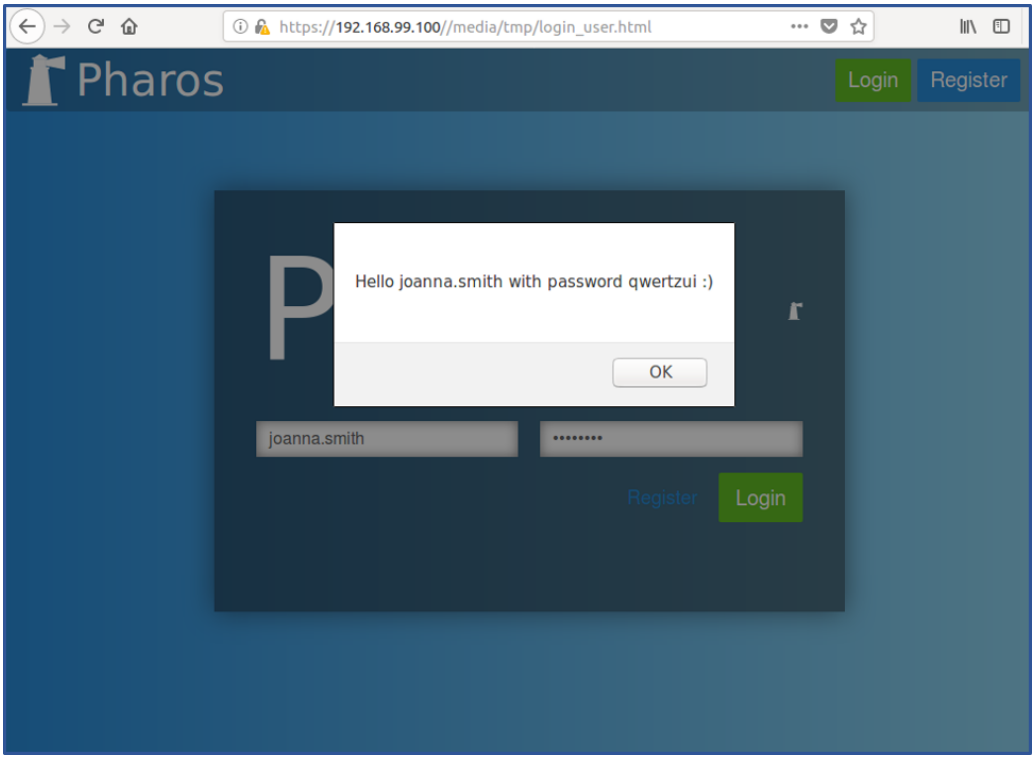

Figure 3.5: Login Via the Uploaded File login_user.html

F TTN Traffic Authentication

An issue that has been found that relates to TTN lies in the way the HTTP traffic between TTN and the OTIoT backend is authenticated on the respective platform. The `PUT` request used by TTN to push positional data to OTIoT is authenticated with Basic Auth, which means that the username and password are Base64 encoded before it is sent in an HTTPS request. On its own, this is defensible and will be discussed later in Section 3.3. However, the credentials used by TTN are those of the Pharos administrator, which is a enormous data leak. This data should not be given to outside agents since one can access the admin account and also the Django admin interface with these credentials. In a Django project, the admin interface can generally be used to look at all data and change that data (like username, email address, Tag name, Tag owner, Tag alarm configurations, etc.) [9]. The damage that can be done to the system if those credentials are used maliciously is enormous. Additionally, the complexity of exploiting the vulnerability is very low, as the means for the exploit is freely provided to an external party. Based on these factors, the priority of this issue is considered high.

The HTTP requests sent from OTIoT to the TTN Application Manager API are also authenticated with Basic Auth, but the requests are not sent over HTTPS. This means we cannot expect that the authentication data is not read by other agents, as it is sent unencrypted. If an attacker captured one of the HTTP packets, they could read the contents and use them to send their own messages to the TTN Application Manager API, while using the credentials of OTIoT. This REST API is used by OTIoT to manage the Tag devices, i.e. creation, update of the alarm configuration, and change of the armed status. In the scenario defined above, they could manipulate Tag devices through this vulnerability and disturb the service OTIoT provides. This issue is considered to be of medium priority, because an attacker needs either access to the network communication or access logs to learn the credentials.
The combination of these issues constitutes a vulnerability classified as a high priority.

G Default Configuration

A review of the system configuration files reveals highly insecure default passwords for database users and the Django default admin user. Protecting the admin with a stronger password is necessary to lower the risk of account take-over via a password-cracking attack. Likewise, the chosen values for database usernames and passwords are weak, which could be exploited to gain access to the database if a password-cracking or -guessing attack was executed. The impact on the system is high, as the database can be accessed and manipulated if the exploit succeeds. The potential of that happening is considered low, because the database of the system is not directly accessible from the internet and protected through a reverse proxy. Only if any of the containers with access to the database or the host machine itself were compromised, this protection would be futile. However, the security improvement of default configuration can still be seemed as worthwhile. A principle in security called defense-in-depth states that it is better to layer security mechanisms so that when one fails, others may still provide protection against attacks [53]. Overall, the priority of this vulnerability is deemed low.

H Dependency Management

Finally, dependencies that are out-dated and have known security issues constitute the last vulnerability discovered. This issue is listed in the OWASP Top 10 under position 9, using components with known vulnerabilities [55]. If a component with a known vulnerability is exploited, it can cause damage to the entire system. According to OWASP, there is a risk of unauthorized data access or data loss, or server take-over [55]. However, the impact depends on the specific software vulnerabilities and can be expressed in a number of metrics, among which the CVSS is one of the best known [26]. The dependencies of the project like Django, Django extensions, Node packages, etc. have been locked by their version number in the Docker setup. Since the project has been finished in early 2017, the project dependencies have not been updated and some of them have become deprecated.

The used Django version, 1.10.5, is no longer supported and is called insecure in the Django documentation [10]. Vulnerabilities pertaining to this Django version can be found in the security vulnerability datasource CVEDetails [5].
An example is 'CVE-2017-7233', which can lead to XSS and received a CVSS score of 5.8, which translates to a medium rated risk [5]. Some of the Node packages used in the project have known vulnerabilities, which could be detected with the audit functionality of the Node Package Manager (NPM) [47]. The more vulnerabilities exist in the dependencies, the higher the risk of exploitation which can cause damage to the OTIoT system. Therefore, the priority of this vulnerability was decided to be medium.

Table 3.1: Vulnerability Overview

ID	Name	Description	Priority
V1	Session Management	Session tokens do not expire on their own. Possibility of session theft.	Medium
V2	Tag Registration	Tags can be registered and blocked by anyone. Knowledge of Tag ID can lead to access to Tag's data.	Medium
V3	User Authentication	Default login is password based. Implemented two-factor authentication is broken.	Medium
V4	Access Control	Data from other users can be seen per HTTP request. Data from other users can be manipulated per HTTP request. Tag position data can be forged via HTTP request.	High
V5	File Upload	File upload possible without login. XSS is possible through file upload.	High
V6	TTN Traffic Authentication	HTTPS requests from TTN and HTTP requests to TTN only use Basic Auth. TTN possesses the administrator credentials for authentication.	High
V7	Default configuration	Default passwords and usernames for database users are weak.	Low
V8	Dependency Management	Python and Node.js dependencies are outdated. Some contain known vulnerabilities that can be exploited.	Medium

3.3 Security Measures

In the OTIoT project, several measures have already been taken before this thesis to secure the system. The backend and website were designed with security in mind, as can be seen, for instance, in the architecture of the Docker containers [39]. In this section, previously taken design decisions and implemented measures to increase the security of the web application will be discussed.

3.3.1 Security of Communication

The design of the backend includes a reverse proxy server as the handler of traffic that enters the OTIoT backend environment and the traffic headed for the outside. In fact, there are two proxy servers, the Let's Encrypt Proxy and the Reverse Proxy containers, as explained in Section 2.3. Per design, a proxy server adds a layer of abstraction to a system by shielding one side of the environment from the other. In our case, the concrete containers are protected against direct access from the internet and the architecture is obscured. The Let's Encrypt Proxy container provides the certificates for the otiot.csg.uzh.ch domain and enforces HTTPS as the traffic protocol. The SSL/TLS certificates authenticate the OTIoT server to the client. In essence, HTTPS (which is HTTP over TLS) provides the same security guarantees as TLS, namely confidentiality and integrity by encryption of data and authenticity for the communication between client and server. This is documented in RFC 5246 [7]. [39]
Inside the backend, the containers communicate with each other over HTTP [39]. Consequently, data is not encrypted, its integrity is not guaranteed, and the communication partners are not authenticated. This is acceptable if the OTIoT containers and the virtual network connecting them are assumed to be secure. If this did not hold and an attacker managed to eavesdrop on traffic between containers, they would be able to read the data in clear text. To protect the OTIoT environment, the nginx server running the Let's Encrypt reverse proxy must be up to date, since it is exposed to the internet and accepts client connections. On the application layer, the same holds for the web container hosting the web application. These two components are the most obvious attack vector for adversaries. Therefore, they have been more thoroughly assessed than less exposed containers such as the celery worker or the database container. Especially the website can be seen as the door to our application, since it is open for anyone to use, even malicious users without much technical knowledge. [39]

3.3.2 Authentication

It was mentioned in Section 3.2.3 that two-factor authentication for users is partly implemented with the python framework *Django Two-Factor Authentication* [34]. The functionality depends on other components, such as *Django-OTP* [63] (which provides a method for one-time passwords) and the built-in authentication framework of Django [19]. However, authentication over phone was broken and could not be used in the application at the beginning of this thesis.

A desire to offer an alternative method to phone and token generator was expressed in the documentation left by the students who created OTIoT [39] [8].
The security of the OTIoT backend REST API was examined in the analysis. For the transmission of positional data from the Tag, TTN uses Basic Auth to authenticate requests to the backend. This means that the password and username are Base64-encoded and appended to the HTTP request without encryption. However, since the API calls are done over HTTPS, the credentials are sufficiently protected. Even though transmitting short-lived tokens (e.g. JSON web tokens (JWT)) is preferable for API access, submitting long-term passwords over a TLS connection is deemed secure enough for this project [36].

3.3.3 Partial Access Control

Data access control was not implemented for all functions. From the analysis it is apparent that the user and user configuration data are vulnerable to manipulation and unauthorized access. However, this does not hold for the Tag and the Tag data, which can only be accessed by the rightful owner and authorized users. The restrictions for accessing and manipulating the Tag data are implemented with the help of the permissions models from Django and the Django REST framework [15].

3.3.4 Security Mechanisms of Django

Finally, Django itself provides some easy to activate security measures, as can be seen in the documentation [16]. One example for these is Cross Site Scripting (XSS) protection. This protection is limited however and does not protect the app from XSS vulnerabilities introduced by programming errors. Django also offers protection against Cross Site Request Forgery (CSRF), but it must be included in the project specifically. In the case of this project, both security mechanisms were used as recommended. The same can be said for the Clickjacking protection, the TLS / HTTPS recommendations, and Host header validation. All these measures protect the backend from various attacks but, as the vulnerability analysis has shown, there still are ways to bypass certain measures and compromise the security of the system and the data. This is especially visible in the session security, for which the measures Django offers are only partially implemented and leave the system vulnerable. [39]

Chapter 4

Design

In this chapter, a design is introduced with the goal to mitigate the security issues identified in Chapter 3. In Section 4.1, security requirements are identified based on the vulnerability analysis that was executed in Section 3.2.3. Following, design propositions to address these requirements are presented. The design ideas are processed in the order of the priority assessment of their corresponding vulnerability. First, the issues that received a high priority are treated in Section 4.2. Subsequently, the medium prioritized issues are treated in Section 4.3. Finally, the issue with a low priority is discussed in Section 4.4.

4.1 Security Requirements

This section presents a list of security requirements. Meeting these requirements will allow to successfully tackle and resolve the vulnerabilities and security issues discovered in the last chapter, specifically in Section 3.2.3. Table 4.1 presents the requirements that are devised from the vulnerabilities discovered in Section 3.2.3. Since some vulnerabilities resulted in several requirements, the column 'Category' in Table 4.1 corresponds to the vulnerability name in Table 3.1.

Table 4.1: Security Requirements

Category	Requirement	ID
Access Control	User and user configuration data can only be accessed and manipulated by their owner.	AC-1
Access Control	Tag data can only be accessed and manipulated by their owner.	AC-2
Access Control	Shared Tag data can only be accessed and manipulated by the owner and users that were given explicit permission.	AC-3
Access Control	Position measurement data can only be added and updated by TTN.	AC-4
Access Control	PIN creation utilities can only be accessed by the administrator.	AC-5
User Authentication	All displayed two-factor authentication methods can be chosen and used without error.	UA-1
User Authentication	A replacement to the phone-based authentication must be provided.	UA-2
User Authentication	Users must set up two-factor authentication to be able to use the website.	UA-3
Tag Registration	Tag Dev EUIs can only be registered by the owners of the Tag with the Dev EUI.	T-1
Tag Registration	Tag IDs are not easy to guess.	T-2
Session Management	Session tokens are randomly generated after login.	S-1
Session Management	Session tokens are stored securely.	S-2
Session Management	Session tokens are not included in the URL.	S-3
Session Management	Session tokens are invalidated at user logout.	S-4
Session Management	Session tokens are invalidated after a reasonable time period has passed since their creation.	S-5
File Upload	File upload is not possible for unauthenticated users.	F-1
File Upload	No files can be uploaded except for image files in the .PNG format with valid content.	F-2
TTN Traffic Authentication	TTN authentication to OTIoT Backend is done with a dedicated user role.	TA-1
TTN Traffic Authentication	OTIoT authentication to TTN is done over a secure channel.	TA-2
Default Configuration	Default configuration security is strengthened.	DC-1
Dependency Management	Django version is updated to a secure version.	DM-1
Dependency Management	No components with known vulnerabilities are used.	DM-2

4.2 High Priority Issues

Design ideas for the highly prioritized vulnerabilities are discussed in this section. The vulnerabilities belonging to this category, as referenced in Table 3.1, are: the Access Control (*V4*), the File Upload (*V5*), and finally, the TTN Traffic Authentication (*V7*).

4.2.1 Access Control

One of the re-occurring issues mentioned in the security analysis concerns the unauthorized access to information in OTIoT. Positional data and personal data, like the email address, are not kept as secure in the OTIoT system as they should be. They can be accessed by unauthorized agents, which is a privacy concern. Additionally, this leaves space for exploits as shown in the vulnerability analysis in Section 3.2.3. Not only is the reading of the data a problem, the manipulation of other user's data and Tag positional data is also an issue. The requirements *AC-1* and *AC-4* state the desired access control for user and user configuration data, as well as Tag position data.
The problem that leads to this security vulnerability is broken access control. OWASP defines broken access control in the following way: "Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other user's accounts, view sensitive files, modify other user's data, change access rights, etc." [56]. The restrictions of user's actions are taken directly from the Master Thesis from Meier, where a list of requirements for the OTIoT system is carved out in the Findings and Requirements chapter [39]. The requirement with ID *F8* is specifically about privilege management. It demands that "access to the stored data can be fine controlled on a per-user basis" [39]. Further, privacy is mentioned in the requirement *Q11*, which demands that "data privacy is assured" [39]. Finally, the point *Q17* requires that the "GUI treats the user's data with care to avoid privacy issues" [39]. These three requirements were the base for the access policy for users. However, only the Tag and shared Tag access right management, as expressed in the requirement *AC-2* and *AC-3* in Table 4.1, could be implemented as a proof of concept in the prototype. The other cases of access control were left as future work. [39]
If one only uses the methods from the graphical user interface (GUI) of the Pharos website to access data, some of the access control issues detected in the vulnerability analysis in Section 3.2.3 are not obvious. For instance, it is not possible to look at or alter another user's data through the GUI or forge positional data for a Tag. In the process of sharing a Tag, however, this is still violated as personal information can be seen by users without having the permission of the owner of this information. What is lacking is server-side validation of the request sender's authorization to access the desired backend API function.

The following points are options specified by OWASP to prevent broken access control [55]:

Deny by default
Access should be denied by default except for resources meant to be publicly available.

Re-use Definition
Access control rules should be defined in one place and re-used through the application.

Record Ownership
Where model data is concerned, no user should be able to create, read, update or delete (CRUD) every possible record. Instead, ownership of data records should be observed.

The goal for the mitigation of the broken access control is to implement server-side mechanisms to prevent unauthorized data access. This will be done both on the data model level - as advised by OWASP, data owners should be able to read, edit, and delete their data - as well as on the function level. Some of the functions should not be accessible for regular OTIoT users, e.g. the sending of positional data from the Tag (which should only be handled by TTN), or the generation of the Tag activation PINs (which is handled by the administrator only). These restrictions of functionality to specific user roles are covered in the requirements *AC-4* and *AC-5*, respectively. Exempted for these mechanisms is the Django admin user, which is a superuser and is assigned all permissions by default for the data CRUD methods.
For the functionality of shared Tags, the requirement of *AC-3* is already fulfilled in the OTIoT implementation [39]. The same holds for the requirement *AC-2* [39]. However, the design still has to be changed such that no private information about other users (i.e. their username and email address) is leaked through the website. It would be best if a user who wanted to share a Tag must enter username or email address of the party they want to grant access to their Tag. It is assumed that users would know this information of people they want to share their Tags with. It is possible to register multiple users with the same email address at the moment, so the username approach is chosen. Additionally, this functionality could be implemented as a two-way permission granting mechanism, where the user granted access must agree to this (for example via an activation link sent via email) before the Tag is definitely considered as shared. However, there is no reason to assume that users would reject such shared Tag requests, so that additional functionality is not deemed to be crucial.

4.2.2 File Upload

The file upload carries a high risk of introducing damage to the OTIoT system. First, any person with access to the Pharos website can upload files. As seen in requirement *F-1*, it should not be possible to upload files to the website for unauthenticated users, as the functionality is only intended to give users the option to personalize their Tag avatar.

However, as elaborated in the vulnerability analysis (cf. Section 3.2.3), the uploaded file is not examined at all before it is uploaded the first time.

While it should only be possible to upload image files in the PNG format, as seen in requirement *F-2*, this is not enforced by the backend. Additionally, as mentioned in Section 3.2.3, Django advises caution for using that exact functionality of permitting users to upload media content files [11]. It is explicitly said that even if one were to implement restrictions on the uploaded file in size (to prevent DoS attacks) and file type (i.e. PNG), it would be not enough of a security measure to prevent all attacks: "No bulletproof technical solution exists at the framework level to safely validate all user uploaded file content" [16]. They recommend considering using a could service or a Content Delivery Network (CDN) to avoid security issues with file uploads. However, that is considered to be too much effort for the actual functionality it provides. After all, the file upload is only used to assign an image to a Tag, serving as its avatar. It is deemed to be unnecessary for a positive user experience, while bringing enormous risks to the OTIoT backend. So, the easiest and most secure option to solve this issue is to provide default Tag icons and abandon the file upload function for users.

4.2.3 TTN Traffic Authentication

As already discussed in the vulnerability analysis, the issue with authentication with TTN is that both endpoints use Basic Auth. The authentication at the OTIoT API endpoint is only considered to be a minor issue, since the HTTPS traffic is encrypted. Still, as stated in Section 3.2.3, TTN uses the credentials of the Django administrator account. Sharing these credentials with TTN is an unacceptable leak of sensitive information. A better method to allow TTN to authenticate at the backend is thus to create a dedicated user account. The user account should have the exclusive privilege to access the sensor data REST endpoint via the POST or PUT method, as required by *TA-1*.
For authentication of OTIoT at the TTN API, there needs to be an alternative implemented to HTTP, as the credentials are not encrypted and can basically be read as clear text in requests. The need for a secure authentication channel is expressed in requirement *TA-1*. TTN does not offer an HTTPS endpoint as a secure option to send requests to their API. Instead, they recommend using the Data API over the MQTT protocol [74]. According to a forum post of a TTN team member, there is a secure way to connect via TLS to the Data API endpoint, as they use Let's Encrypt certificates to authenticate [84]. This is the intended design choice to solve the issue of unencrypted and unsecure traffic coming from OTIoT to TTN. However, at this moment, with MQTT one can only send data to the IoT device. The messages OTIoT sends to the Application Manager API that consist of device activation and deletion do not have a corresponding MQTT counterpart yet. There seem to be plans to implement such functions, as can be seen in the source code for the TTN python module [79], but the functions are not implemented at this time.

4.3 Medium Priority Issues

In this section, the vulnerabilities that were assigned a medium priority for the project are discussed. They consist of the Session Management (*V1*), the Tag Registration (*V2*), the User Authentication (*V3*), and the Dependency Management (*V9*).

4.3.1 Session Management

Concerning the session management, the recommendation of OWASP is to "use a server-side, secure, built-in session manager" to create random session IDs after login [55]. That recommendation was directly translated to the requirement *S-1*. Furthermore, session IDs should be stored securely, not be included in the URL, and be invalidated at logout. These restrictions concerning session management are demanded in requirements *S-2* to *S-4*. Finally, after some time of inactivity on the site and after an absolute timespan, the session tokens should be invalidated according to OWASP. The requirement *S-5* combines them to an invalidation of session tokens after a reasonable amount of time has passed. [55]
The intended solution to improve session management is based on this idea, except for the implementation of an inactivity timeout. The reason that is that in the case of Pharos, users are never seen as inactive due to JavaScript functionality. The client-side JavaScript code generates HTTP requests periodically to check for data updates in the backend. This guarantees a live view of Tag data, among other things. However, this cannot be easily differentiated to real user behaviour. Moreover, a session timeout due to inactivity cannot prevent an attacker that has stolen a session cookie from using it further, assuming the attacker has knowledge of idle timeouts and keeps the stolen session constantly active. Absolute timeouts are thus more sensible as they cannot be avoided by an attacker. Furthermore, since the website does not provide a wide array of functionality to keep users active, it seems reasonable to keep the absolute timeout to a short time, such as 30 minutes. This lowers the risk of giving an attacker enough time to guess a session ID while also keeping the time an attacker could use a stolen session ID short.

4.3.2 Tag Registration

The problem discussed in *V2*, Tag registration, is that the Dev EUI needed for the creation of a Tag is only able to be used once for TTN. However, this is not enforced in the frontend and works on the first come, first served basis. A malicious user could thus hoard devices and block the usage of valid Tag devices. To prevent that from happening, requirement *T-1* demands that only owners may be able to register their Tag on the Pharos website.
As a solution to the stated problem, the Tag registration process was re-designed. The first, smaller change is that the Tag ID, which is used as the primary key for identifying Tags between TTN and the OTIoT backend, is now automatically assigned by the system at Tag creation. The frustration of having to create a unique Tag ID is taken from the user, without restricting their ability to name the Tag as they want it via the Tag name.

As the ID is not necessary for the user to identify their Tag, that function did in fact not provide any benefit to the user. A positive side-effect is that random numbers for IDs make it harder to guess Tag IDs, which is demanded in requirement *T-2*.
The other design change is the introduction of an activation PIN. As the OTIoT project evolved, requirements of the Tag registration changed. Two possibilities pertaining to the choice of Dev EUI and App Key were described, with different effects on the registration and Tag setup process.

The first possibility is that those values can be freely chosen by the user or generated by OTIoT - this was indicated in the GUI by the "will be generated if empty" hint in the Tag registration form. However, this meant that users would have to program these generated values into the Tag software themselves. For this, the Arduino development environment would have to be downloaded, the software for the Tag would need to be imported and changed in the places where those values are defined, and finally, the code would have to be deployed onto the Tag device. This is not a user-friendly process and it is questionable whether users would be willing to accept this effort. [8] [39]

The second possibility is that the Tag is pre-programmed with those two values and sold to users with an instruction guide where the values are written down [8] [39]. This approach is found nowadays in several electronic devices, like default passwords in routers or activation PINs for SIM cards. So, additionally to the Dev EUI and the App Key value, the activation PIN would also have to be entered into the Tag creation form. This PIN has to be entered at Tag creation to validate that the user creating the Tag is actually the owner who bought it. Also, the PIN should depend on the Dev EUI, which is only made available to the authorized user, because the goal is to protect this value from being used by others. The chosen design for the PIN is to use a Message Authentication Code (MAC). In this case, the authenticated message is the Dev EUI. More specifically, the choice was made to use a hash-based MAC (HMAC) algorithm to generate a code from the Dev EUI and a secret key. From that code, 12 digits were chosen as the activation PIN. The process of the PIN generation can be seen in Figure 4.1. This will be verified in the backend every time a Tag creation request is sent.

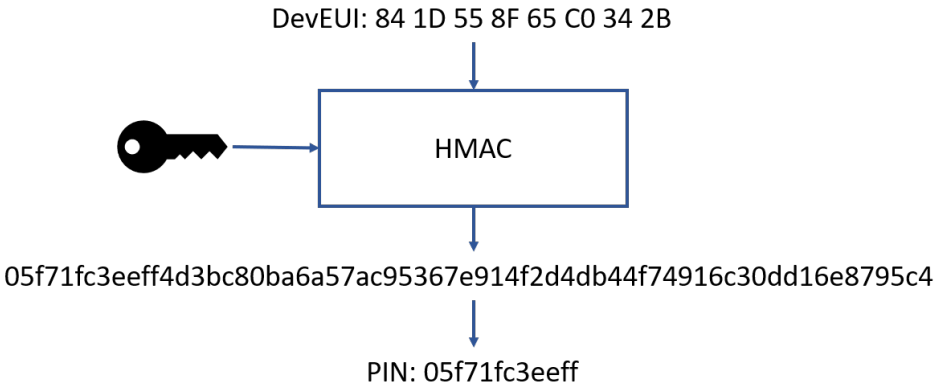

Figure 4.1: PIN Generation Process

It was also decided that the PIN generation should be possible through the website. This was done so the secret key and specific random seeds do not have to be stored in multiple places. Also, it would be unhandy to need a distinct, second application just for PIN creation. However, there need to be additional security measures to prevent malicious users from accessing the PIN generation method through the website. For this, the same access control mechanisms as for the other sensitive areas of OTIoT will be implemented.

4.3.3 User Authentication

Two vulnerabilities have been found that belong to the OWASP Top 10 category 2, Broken Authentication. Those are *V1* and *V3*, session management and user authentication, respectively. Regarding user authentication, OWASP strongly recommends implementing a multi-factor authentication "to prevent automated, credential stuffing, brute force, and stolen credentials re-use attacks" [55]. Following this recommendation and to lower the risk of an account take-over, it was also decided to make 2FA mandatory for every user, instead of leaving the choice to the user as it was done before. Requirement *UA-3* was formed to conform to that. [55]

A problem of the already implemented 2FA is that two of the three available methods ended in an error message. As can be seen in requirement *UA-1*, it must be possible for users to utilize any of the displayed authentication methods. There are different proposals to solve this issue. First, one could abandon the implemented framework and choose another to replace it. However, the problem didn't lie within the framework but with the authentication method over phone. As mentioned in Section 3.2.3, the phone-based methods were chosen to be deactivated by the previous developers. This resulted in an error when those methods are selected in the two-factor authentication setup.
Alternatively, the required functionality could be implemented from scratch, which would make it possible to customize the authentication to the exact requirements that are needed. The drawback of this approach would be that it is time intensive to create and maintain. Furthermore, the code could be more susceptible to flaws that may not be noticed right away. Mostly, frameworks that are actively used by a community are made aware of errors or security flaws by their users and are steadily improved. The framework currently used is still updated on a regular basis and has 42 contributors on GitHub [33]. There is also evidence for an active discussion about issues and change requests on GitHub [33].
After these considerations it was decided to keep the Django Two-Factor Authentication framework and just make adjustments to the code where needed. To offer a replacement for the broken authentication options, as required in *UA-2*, and provide a more diverse choice to users, a method that is comparable in its simplicity to the phone-based authentication will be included - the email-based token generation. This is possible because the code is open source and stands under the MIT license [48]. To simplify the migration of the adjustments to updated versions of the framework, the changes were kept to a minimum.

4.3.4 Dependency Management

For the dependency management, two different requirements are stated, *DM-1*, which focuses on the Django framework itself, and *DM-2*. For the vulnerable components mentioned in *DM-2*, there exist different categories of dependencies that are outdated and need to be considered for this issue.
One group consists of the python dependencies that Django uses. The other category consists of JavaScript dependencies that are used in the frontend container. The approach for both is to update dependencies that are outdated. If no update exists, there would be either the possibility to dispose of the dependency and look for another framework that provides the functionality needed, or to edit the source code directly where the vulnerability is introduced, if that is possible. The second approach is not recommended, as changing code can always lead to the inclusion of more unnoticed security issues.
So, the chosen approach is to update critical dependencies where possible. For some vulnerable parts of dependencies, the risk is tied to the execution of specific methods, in which case it would only be acceptable to continue using the code if those risky methods are not executed. In some cases, it may be that the dependency cannot be replaced or edited, so the vulnerability has to be accepted and treated with caution in further security related plans for the project.

4.4 Low Priority Issue

Finally, the single low prioritized vulnerability that was detected in the OTIoT system is discussed in this section. The design consideration for the issues of the Default Configuration (*V8*) is explained in the following paragraph.

4.4.1 Default Configuration

The requirement *DC-1* is about strengthening the security of the default configuration. It can be approached by using stronger passwords and changing the user names that were specified. The question that emerges from this is what makes one password stronger than another. There has been some discussion over the years on how to generate strong passwords. Standard advice to users, as can for example be seen from Google [28], is to include letters, numbers, and special characters in a password, to not include personal information or commonly known words, and to avoid using passwords again for other websites . Additionally, the longer a password is, the harder it is to guess in a brute force attack. The design to solve this issue is to provide stronger passwords for the pre-set user accounts and roles.

Chapter 5

Implementation

After assessing vulnerabilities in Chapter 3 and carving out requirements and a design for tackling the issues in Chapter 4, the implemented solution is explained in this chapter. First, an explanation of the initial changes made to set up a running OTIoT backend is given in Section 5.1. In the following sections the security improvements are examined. The changes to the access control mechanisms are shown in Section 5.2.
In Section 5.3, the measures deployed to improve user authentication are focused on. Then, the adaptations to the Tag registration process are discussed in Section 5.4. Finally, smaller changes to the project, including the dependency updates, the file upload, and the session management, are summarized in Section 5.5. For a better readability, changes to the default configuration will be introduced and discussed in Section 6.1.

5.1 Environment Setup

Before the backend was able to run in a Linux operating system environment, a few adjustments had to be made to the project code. For transparency, these changes are summarized here because they could have an influence on the rest of the code and its functionality, although this is not expected.
Several changes were necessary because the original developers used a MacOS to set up their environment, while for this thesis, a Linux Ubuntu OS was used. So, for local setup, the `Makefile` for the web container had to be adapted to match the names of packages to install. Instead of `brew` commands, the `apt-get` package management tool was used to download dependencies. In the same folder, the `Dockerfile` was slightly adapted to prevent errors. Moreover, the shell script file called `run_web.sh` had a missing `import` statement, which caused an error in the creation of the superuser for the web application.
The file upload of OTIoT did not work as intended. Files could be uploaded, but they could not be accessed because the files were stored in a place that the machine serving them had no access to. To prove that the file upload is a security vulnerability, additional changes to the code were introduced to restore the upload functionality. To fix the problem, the entry for the `web_media` volume was added to the web container definition of the Docker compose file.

Additionally, the value of the `WEBSERVER_ROOT` was changed to the path `/usr/src/app`. This was done to match the place the images are saved to where they are searched for by the nginx server that is responsible for serving the files to the website.
Finally, the connection between the Celery workers and the Redis queue caused various errors and did not work correctly. The change that was done to fix this issue was a simple rename of the `REDIS` variable in the environment settings to address the `redis` docker machine instead of an IP address. All these changes should be clear in the context of the step-by-step setup in Chapter A of the appendix.

5.2 Restrictions On Data Access

For the access control of OTIoT data, several requirements were defined in Section 4.1. Excluded from the implemented restrictions on data access is the superuser, i.e. the administrator, as explained in Section 4.2.1.
First, only data owners should be able to access their data and manipulate it, except for the Shared Tags, which are examined later. This affects several models and their defined HTTP endpoints in the Django `views.py` files.

5.2.1 User And User Configuration Data

In requirement *AC-1*, the access restrictions on user and user configuration data is defined, as can be seen in Table 4.1. To implement that, the `UserViewSet` and `UserConfigurationViewSet` classes were extended to include permissions. The permissions were written into a newly created `permissions.py` file. The two permission classes `UserPermission` and `UserConfigurationPermission` are defined in that file.
In Listing 5.1, only the `UserPermission` is shown, but the `UserConfigurationPermission` is implemented similarly.

They both override the Django REST Framework `BasePermissions` class [15], as can be seen in line 1 of the Listing 5.1. Two levels of permissions can be defined through this, the first method listed on line 2 is on the view-level. It can be seen in lines 3, 6, and 8 that the permission check depends on the request type (i.e. `list, create, retrieve, update, destroy`) that is stored in the `view` variable. For instance, in line 4 and 5 the function only returns true if the request was sent from an authenticated superuser. This means that only those users are allowed to request a list of all users. All other request types that fall into `create, retrieve, update, partial_update`, or `destroy` are available to all authenticated users, as seen in lines 7 and 10.
The second function, `has_object_permission`, is only executed if the first returns `True` and is an access control check on the object-level. It is used to restrict access to the owners of the data instance. The permission for `retrieve`, `update`, and `partial_update` is granted only if the sender of the request can be mapped exactly to the object that is requested or if the sender is a superuser, which can be seen in lines 19 and 21.

Moreover, only an administrator is allowed to delete user (and user configuration) data. These permissions are included in the `UserViewSet` (and `UserConfigurationViewSet`) class by setting the attribute `permission_classes` to `UserPermission` (or `UserConfigurationPermission`, respectively). This attribute originates from the base class the two `ViewSet` classes have overridden, which is the `ModelViewSet` class from the Django REST framework [20].

```
1  class UserPermission(permissions.BasePermission):
2    def has_permission(self, request, view):
3      if view.action == 'list':
4        return (request.user.is_authenticated and
5        request.user.is_superuser)
6      elif view.action == 'create':
7        return request.user.is_authenticated
8      elif view.action in
9      ['retrieve', 'update', 'partial_update', 'destroy']:
10       return request.user.is_authenticated
11     else:
12       return False
13
14   def has_object_permission(self, request, view, obj):
15     # Deny actions on objects if the user is not
     authenticated
16     if not request.user.is_authenticated():
17       return False
18     if view.action == 'retrieve':
19       return obj == request.user or request.user.
     is_superuser
20     elif view.action in ['update', 'partial_update']:
21       return obj == request.user or request.user.
     is_superuser
22     elif view.action == 'destroy':
23       return request.user.is_admin
24     else:
25       return False
```

Listing 5.1: UserPermission Class in user/permissions.py

5.2.2 PIN Generation

The implementation of the permission check for the PIN creation functionality is less complicated than the one for user data. Because only the administrator user is required to have access to the PIN creation, as seen in *AC-5* in Table 4.1, the permission check simply consists of checking whether the sender of the request has administrator privileges.

The permission check for the pin generation functionality is done directly in the `urls.py` file. As can be seen in Listing 5.2 in line 6, for the permission check the decorator function `user_passes_test`, which was imported from the built-in Django authentication framework, was used. Only if that method returns `True`, the method `get_pin` in the `views.py` file is executed, as seen in line 7 of the Listing 5.2.

```
1 from django.conf.urls import url
2 from . import views
3 from django.contrib.auth.decorators import
     user_passes_test
4
5 urlpatterns = [
6   url(r'get-pin', user_passes_test(lambda u:
7     u.is_superuser)(views.get_pin), name='get_pin'),
8 ]
```

Listing 5.2: Permission Check in /utils/urls.py

5.2.3 TTN Traffic and Tag Position Measurement

In Section 4.1, the requirement *AC-4* was defined to tackle the vulnerability of unauthorized Tag position data access. First, the usage of the administrator credentials by TTN was discontinued by deleting the credentials from the TTN console website. Besides that, the password of the administrator was changed.
A dedicated user account was created for TTN, called 'ttn_http_api', according to the requirement *TA-1*. This is done in the bash script `run_web.sh`. An excerpt of the script, which is written in python, can be seen in Listing 5.3. As explained in Section 2.4, the script is run after the web container is fully set up. First, in that file, the user for TTN is created in line 4 to 5. Then, the permissions for updating and adding position measurements were extracted from the database in line 6 to 7 and 9, respectively. Finally, in line 8 and 10, the permissions are assigned to the TTN user account. These permissions are then checked in the file `views.py` when a `POST` request is made to the API. Instead of using object-based or view-based permission checks such as with the user permission check, the check runs with a `has_perm` function executed on the request user object.
The other requirement pertaining to TTN traffic is *TA-2*, see Table 4.1. It requires the authentication of OTIoT over a secure channel. In Section 4.2.3 in the design chapter, the options provided by TTN for a secure data transmission have been discussed. The usage of MQTT to access the Data API seems to be the recommended approach by TTN, with a potential future extension to the Application Manager API. To use the MQTT protocol in the code, the Eclipse Paho MQTT python client library [23] was downloaded and imported in the Tag `models.py` file. The function `send_config_downlink` that can be seen in Listing 5.4 is used to send downlink messages to the Tag.

```
1  import os
2  from django.contrib.auth.models import Permission
3  from django.contrib.auth.models import User
4  user_ttn = User.objects.create_user('ttn_http_api',
5    'pharos.otiot@gmail.com', os.environ.get('
     TTN_USER_PASSWORD'))
6  permission = Permission.objects.get(codename =
7    'update_position_data')
8  user_ttn.user_permissions.add(permission)
9  permission = Permission.objects.get(codename = '
     add_positionmeasurement')
10 user_ttn.user_permissions.add(permission)
```

Listing 5.3: TTN User Creation

In lines 2 to 4, the payload data for the message is encoded into a payload format. Before, when the TTN HTTP integration was used, this was done by TTN. The encode function had to be translated from the one described in Diez' thesis [8], which was also implemented in the TTN console, to python.
The Tag expects the Tag values `active`, `acc_sensitivity`, `pgs_tolerance`, and `moving_send_delay` to be encoded in nine bytes, which is done through the `encode` function. The `topic`, which is defined in lines 9 and 10, can be seen as similar to a HTTP request path and is prescribed by TTN [69], as are the host and port values. The port that is defined on line 12 is specified by TTN to be used for TLS encrypted traffic. Additionally, a TLS certificate file that was previously downloaded from TTN has to be included, as seen in line 16. Authorization is done the same way as before, with HTTP requests, and can be seen in lines 13 to 15. The `publish.single` function on line 18 is a shortcut provided by the paho-mqtt framework that includes connection start, publishing of the message to the MQTT endpoint, and connection teardown.

5.2.4 Tag Sharing Process

As explained in Section 4.2.1, the functionality of sharing a Tag has to be adapted. The change was necessary because user data, such as email address and username, should only be able to be accessed by the corresponding owner of the data. So, the dropdown menu that can be seen in Figure 3.4 did not show anything after the access control changes were implemented. Instead of the dropdown menu, a text input field was defined that must be used to enter the desired username, as can be seen in Figure 5.1. To provide the same Tag sharing functionality as before, minor adaptions in the related HTML, JavaScript, `views.py` and `models.py` files were necessary.

```
def send_config_downlink(self):
    byte_payload = encode(self.active,
     self.acc_sensitivity, self.gps_tolerance,
     self.moving_send_delay)
    payload_raw = {
     "port": int(os.environ.get('TTN_PORT_CONFIG')),
     "confirmed": False,
     "payload_raw": byte_payload}
    topic = str(os.environ.get('TTN_APPLICATION_ID') +
     "/devices/" + self.uid + "/down")
    host = "eu.thethings.network"
    port = 8883
    authorization = {
     "username": str(os.environ.get('TTN_APPLICATION_ID'))
     "password": str(os.environ.get('TTN_ACCESS_KEY'))}
    tls = {"ca_certs": "/usr/src/app/mqtt-ca.pem",
     "tls_version": ssl.PROTOCOL_TLSv1}
    publish.single(topic, payload=json.dumps(payload_raw),
   hostname=host, port=port, auth=authorization, tls=tls)
```

Listing 5.4: MQTT Data API Usage

5.3 Two-factor Authentication

The requirements for changes in the two-factor authentication procedure are defined in Table 4.1 in *UA-1* to *UA-3*. For both *UA-1* and *UA-2*, the necessary changes were made directly in the source code of the 'Django Two-Factor Authentication' framework [34]. Instead of including the framework as a third-party app in the settings of Django, the source code was downloaded and added into the project structure like the local apps, as was explained in Section 2.4. The next step was to adapt the authentication method to match the custom design that was declared in the last chapter.
First, the phone based two-factor authentication was disabled. This was done by adapting the `models.py` file and changing the method `get_available_phone_methods` so that it returned an empty set. The next step was to add the email one-time password functionality. A new function, shaped after the now defunct `get_available_phone_methods` function, was written to include the choice of the email method in the setup process. The exact changes can be seen in Chapter D in the appendix.
At last, the functionality of the email-based authentication method had to be implemented. The Django two-factor framework depends on 'Django OTP' [63]. It already provides a model class, called `EmailDevice`, which delivers the desired functionality. Finally, the email message and subject also had to be configured.

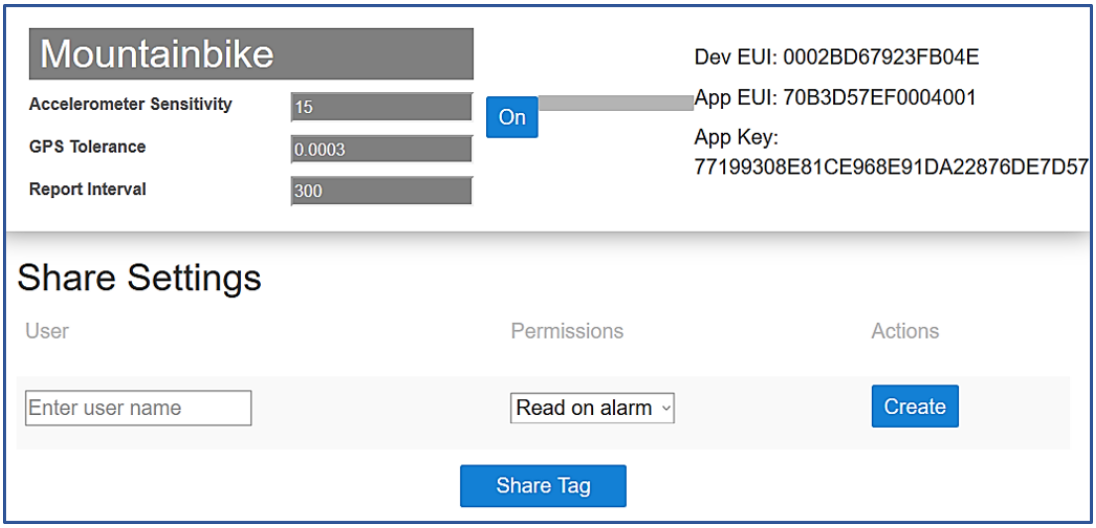

Figure 5.1: Adapted Share Tag Functionality in Pharos

The design in Section 4.3.3 additionally specified the requirement to make two-factor authentication mandatory instead of optional. The documentation of the Django framework mentions that enforcing two-factor authentication is not implemented. However, the framework provides decorators and mixins, which both are mechanisms to extend functionality of classes in python. These mechanisms can be used to limit access to a view to users that have been verified using two-factor authentication. The `OTPRequiredMixin` is used to limit users to access views. This mixin has been used to decorate the `DashboardBaseMixin`, which is included in all `views.py` classes and subclasses that exist in the *dashboard* and the other apps that were introduced in Section 2.4. The effect this has is that a non-verified user is redirected to the setup of the second factor if a URL is requested that is not the login page.

5.4 Tag Registration

The Tag Registration process requirement *T-1* demands a method of ensuring only authenticated Tag owners can register their Tags in Pharos. In Section 4.3.2 in the design chapter, a way to achieve Tag owner authentication was proposed. For that functionality, three measures had to be implemented: a method for the administrator to generate activation PIN codes, a change in the registration form to include the PIN, and the backend method to check whether a PIN is correct for the given Tag Dev EUI. Additionally, the Tag ID was changed to be generated automatically and at random, corresponding to requirement *T-2*. This functionality had to be changed in the Tag `models.py` file.
The Tag ID was chosen to be generated by default by the python module `uuid`, which implements the RFC standard 41222 [59]. It provides functions to create Universally Unique Identifiers (UUID). From those, the chosen function is `uuid4()` because the number it creates is random, which was important to increase resistance against guessing the ID. For the usage in the database and for readability, the used ID is the 32-character hexadecimal conversion of the generate UUID.

The first step for the PIN generation functionality was to implement an additional GUI component in the dashboard that only the administrator could access, as can be seen in Figure 5.2. The page is held in a simple design; there is only an input field to enter the Dev EUI and a button to submit the request. In the backend, the API functionality is defined as a function that checks the user authentication and privilege, as described in Section 5.2.2. The called API view function utilizes the `generate_pin` function that can be seen in Listing 5.5 and returns a simple JSON object including the PIN value.

Figure 5.2: Pin Utils View

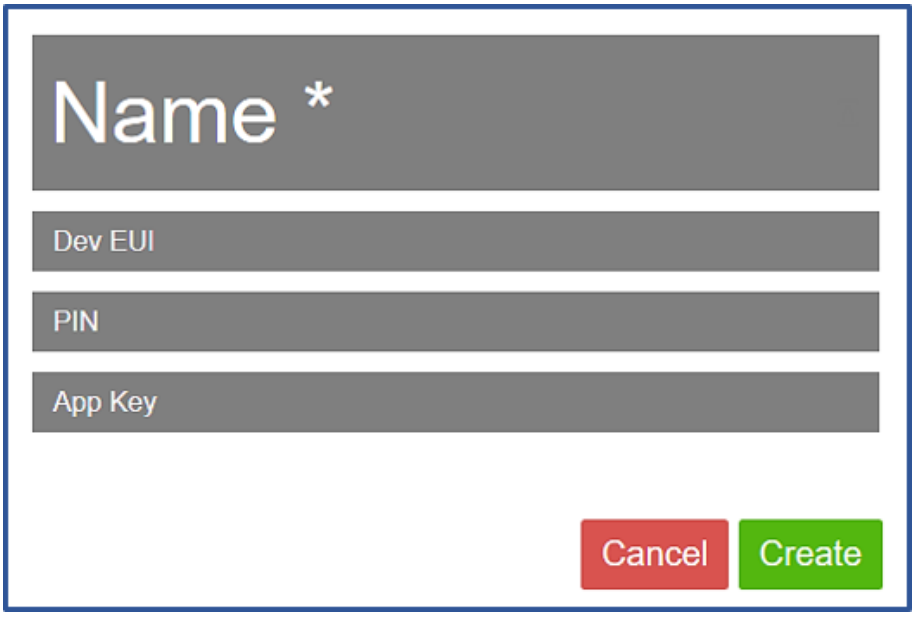

Figure 5.3: Modified Tag Creation Form

The `SECRET_KEY` that is used by Django for cryptographic purposes, such as signing messages [18], was used as the key input for the HMAC algorithm, as can be seen in line 4. The message to be signed is the Dev EUI, as explained in the design choices in Section 4.3.2. Python provides a HMAC generation function with the choice of different hash algorithms. In lines 6 and 7 of Listing 5.5, the HMAC generator object is created with the choice of SHA256 as the underlying Hash algorithm. SHA256 was chosen from the available choices because it is still considered an approved, secure hash algorithm for the purpose of MAC by NIST [6]. Also, as advised in the publication for the truncated message digest, the leftmost bits were used to finally create a hex value of 12 digits, as can be seen in line 9. Collision-resistance was not considered important for the usage of the MAC.
The Tag registration process form on the website was extended with an input area for the PIN, as can be seen in Figure 5.3. In the backend, the PIN value included in the Tag creation request data was handled in the `TagSerializer` class. The PIN is not stored in the database because it is only needed at creation time. The value must thus be filtered from the request data before the `TagSerializer`, devised from the Django REST framework `Serializer` class, can convert the JSON data to the Python `Tag` model object [17], as described in Section 2.4. The `create` method the serializer uses for the Tag object creation can be seen in lines 3 to 10 in Listing 5.6. If the request data does not contain the PIN (seen in line 4) or the PIN value is wrong (as seen in line 8), the REST API call will be returned with an error message. To ensure the sent PIN is valid, in line 6 the expected PIN is generated from the Dev EUI that was included in the HTTP request. If the check for equality passes, the PIN value is deleted from the validated data the serializer uses in the function, as can be seen in line 9, and the `Tag` object is created from the remaining validated data in line 10.

```
1  def generate_pin(dev_eui):
2    pin = ""
3    if not (dev_eui == None or dev_eui == ""):
4     key = str(os.environ.get('SECRET_KEY')).encode('utf-8')
5     sig_enc = dev_eui.encode('utf-8')
6     hmac_generator = hmac.new(key, sig_enc,
7      hashlib.sha256)
8     hmac_generated = hmac_generator.hexdigest()
9     pin = hmac_generated[0:12]
10   return pin
```

Listing 5.5: Generate Pin in utils/views.py

```
1  class TagSerializer(HyperlinkedModelSerializer):
2    pin = serializers.CharField(write_only=True, required=
     False)
3    def create(self, validated_data):
4      if not validated_data['pin']:
5        raise serializers.ValidationError("Pin is wrong for
     this Tag.")
6      expected_pin = generate_pin(validated_data['dev_eui'])
7      if (expected_pin != validated_data['pin']):
8        raise serializers.ValidationError("Pin is wrong for
     this Tag.")
9      validated_data.pop('pin', None)
10     return Tag.objects.create(**validated_data)
```

Listing 5.6: Excerpt of the TagSerializer Class in tags/serializers.py

5.5 Modifications to Upload and Management Functionality

In this section, the implementation pertaining to upload and management functionality is discussed. Specifically, the modifications to the file upload functionality, the session management, and the dependency management are implemented according to the requirements stated in Table 4.1.

5.5.1 File Upload Restriction

The requirements *F-1* and *F-2* demand restrictions on the file upload to avoid the vulnerability detected in Section 3.2.3. In Section 4.2.2 in the design chapter the difficulty of restricting media upload to a Django backend has been mentioned. To prevent the security issues that are inherently connected with a file upload in a Django project, the functionality was deleted.
The code was structured to provide a default icon if the user did not choose to upload an image at Tag creation. This was done in the HTML template file responsible for displaying Tags, as can be seen in Listing 5.7. Now, instead of having the `v-if` and `v-else` options, that can be seen in line 3 and 4, only the default option of `src="'/img/icons/marker.svg' | static"` remains. Likewise, the Tag `models.py` and `views.py` files were adapted by deleting the code handling custom avatar file upload. At the end, the `urls.py` file was changed to not list the file upload as a API function anymore. So even if the user cannot upload a Tag avatar, the default icon functionality is unchanged.

```
<td class="table__cell">
    <div class="table__content--center">
      <img v-if="tag.avatar" class="table__icon
    table__icon--hexagon" :src="tag.avatar"></img>
      <img v-else class="table__icon" :src="'/img/icons/
    marker.svg'| static"></img>
    </div>
  </td>
```

Listing 5.7: Previous Avatar Choice in tag_table_entry.html

5.5.2 Session Management Adjustments

The requirements pertaining to the session management are listed in Table 4.1 as *S-1* to *S-5*. The first step taken to implement proper session management was to set an absolute timeout for session cookies. Django provides a middleware that offers session management functionality, the `SessionMiddleware` [12]. To use the functionality, the middleware has to be included in the Django settings file. This was already done in the initial OTIoT project version. With the standard functionality of the Django class `SessionMiddleware`, requirements *S-1*, *S-3*, and *S-4* are implemented [12].

For the session token storage, Django offers the choice between using database-backed, cached, file-based, or cookie-based session backends. The decision fell on the database-backed solution, because it is the most stable choice and additionally, it is easy to configure. Furthermore, as a secure storage option was specifically required in *S-2*, so the database seemed a good choice. The cache does have a slightly better performance, but there can be problems such as data loss occurring with a cache backend, according to the official documentation [12]. The setup and configuration process for this solution was considered to be too complex and error prone to justify the speedup achieved. Furthermore, the overhead introduced by the database is not an issue on a web application of this scale, as the number of sessions is not that high. The cookie-based solution was discarded because of the lack of a freshness guarantee Django warns of and the performance that can suffer with the method. Lastly, the file-based backend was dismissed because of a subjective preference of the databases approach. The file-based approach also does not offer any advantage compared to the database backend. [12]
To include the database-backed session in the Django project, some variables had to be set in the settings file. The absolute timeout for session cookies can be set via the variable `SESSION_COOKIE_AGE`, which was set to 1800 seconds. This is a reasonable expiration time for session cookies, conforming to the requirement *S-5*. Additionally, the `SESSION_EXPIRE_AT_BROWSER_CLOSE` was activated, so cookies will expire after closing the browser. However, one should not rely on this, as different browsers can choose to not implement that functionality. With the storage of session cookies in the database, the necessity to delete expired tokens appears, to prevent the database size from increasing continuously. This was solved with the implementation of a cronjob that is scheduled to run every hour [82]. The task runs a script that calls the Django management function `clearsessions` to clean expired sessions from the database [9].

5.5.3 Dependency Management Updates

The previously used Django version, 1.10.5, is not supported by Django anymore [31]. The requirement *DM-1* demands the usage secure Django version, so it was updated to the long-term supported (LTS) release 1.11, specifically the most recent version 1.11.13. This means that until at least April 2020, the release is supported with security updates. The most recent Django version overall is 2.1.1. However, that version of Django does not support python 2.7 anymore, on which various dependencies in the project rely on. So, Django version 1.11.13 is the most recent that still supports Python 2.7 and was chosen for that reason. As a consequence of the Django update, other python dependencies defined in the `requirements.txt` file had to be updated. All version changes can be seen in Table 5.1. Most of these updates did not cause any breaking changes and the errors caused by an upgrade were fixed quickly.

Table 5.1: Overview of Updated Python Requirements

Python Package Name	Initial Version	Updated Version
Django	1.10.5	1.11.13
Celery	4.0.2	4.1.1
Django-filter	1.0.1	1.1.0
Django-registration-redux	1.4	2.4
Gunicorn	19.3.0	19.5.0
Sqlparse	0.1.18	0.2.4
Django-debug-toolbar	1.4	1.8

Subsequently, the Node package dependencies on which the frontend relies were updated. Vulnerabilities that could be mitigated with updates that cause no breaking changes were done with help of the `npm audit fix` function [47]. One of the updates that had to be installed manually was the Gulp package [32], which is a framework for automating tasks. The version was updated to 4.0.0 and affected the `gulpfile.js` file in the *frontend* container. That file consists of tasks, which, among other things, automatically compile the sass files included in the project. The method definition and sequences of several gulp tasks had to be re-arranged to fit the task system change gulp introduced with the version update. After the breaks could be fixed, various other dependencies used by gulp (like `gulp-cli, gulp-sass, browser-sync, gulp-jscs`, etc.) could be updated without problems.
At the end, not all components with known components could be updated, as required by *DM-2*. Consequently, not all security threats could be mitigated. However, the Node Package Manager audit indicated that only seven of 635 vulnerabilities that existed in the previous OTIoT setup remained. Among those, six are rated low and one is rated as critical, compared to the previous 539 low, 52 moderate, 41 high, and three critical vulnerabilities. For all of the left-over vulnerabilities, no patch was available, or the vulnerable part was an irreplaceable dependency of one of the Node packages. More details to the audit report for the initial OTIoT system and the current audit report can be found in the accompanying CD in the folder *additionalMaterial*.

Chapter 6

Evaluation

This chapter presents an evaluation of the implemented methods with respect to the security issuse they are meant to solve. This chapter consists of two parts. First, the evaluation of the contributions will be presented in Section 6.1, following the structure that was applied in Chapter 4. Following that, some limitations concerning the evaluation method will be discussed in Section 6.2.
The criteria that are used for this evaluation are derived from the security requirements stated in Table 4.1. It was deemed a reasonable choice because those requirements were defined to address the direct cause of the found security issues. Moreover, they were also used as a guide in design and implementation.

6.1 Vulnerability Mitigation Check

In this section, the implemented mechanisms to mitigate security issues are evaluated. The following sections provide more detailed information about the methods that were applied to check the requirements. Summarizing, all the requirements were fulfilled, except for requirements *TA-2* and *DM-2*, secure authentication of OTIoT to TTN and using no components with known vulnerabilities, respectively. Both these requirements could only be partly fulfilled.

A Access Control (V4)

The adapted implementation restricts access to data at every HTTP endpoint, as explained in the implementation in Section 5.2. The requirements *AC-2* and *AC-3* were already implemented into the initial OTIoT system, as mentioned in 4.2.1. To ensure that only authorized users can access data, these restrictions were tested manually. The same method that was used in the security analysis (i.e. HTTP requests from a logged in user) is applied to check if the implementation fulfills the access control requirements. The collection of requests used can be found in the appendix Chapter C.

A specific request that worked before the change but is denied now is the `GET` request for other user's data. In the same way, the `PUT` request to change other user's data that was used in the vulnerability analysis, as seen in Listing 3.3, is not successful. This can be seen in Figure 6.1, in which a `PUT` request with an ID of another user is executed.The server response on the right side shows the permission for this action is denied and thus, the request for a data modification is not granted. The same HTTP requests were carried out for the user configuration data with the same results. With this, *AC-1*, the restriction of access to user and user configuration data to their owner, is fulfilled.
For the other requirements on access control, *AC-2* to *AC-5*, the method to evaluate whether the implementation fulfills the requirements uses the same mechanism of sending HTTP requests with Burp. The evaluation of the Tag and shared Tag access control was done exactly as the one for *AC-1*.

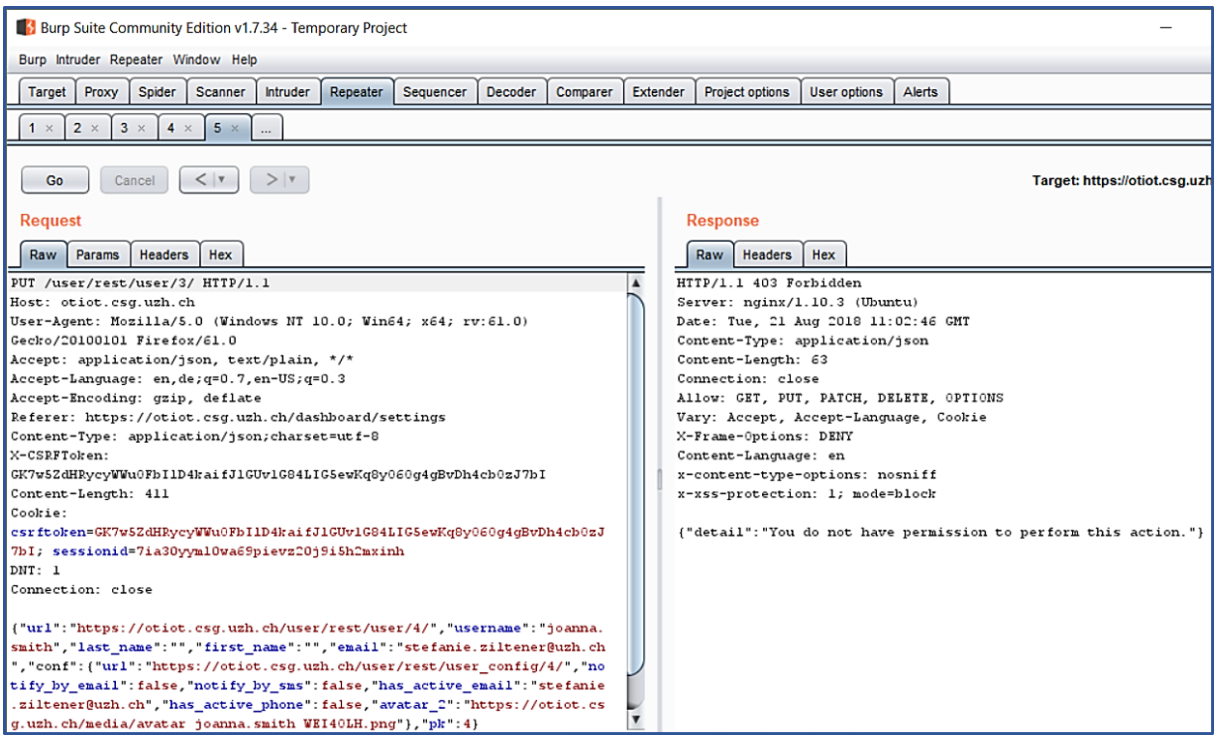

Figure 6.1: HTTP PUT Request and Response for User Data Without Authorization

For the position measurement access, as required in *AC-4*, it was tested whether the dedicated TTN user, `ttn_http_api` is able to send `POST` data to the OTIoT backend, which succeeded. This can also be verified when a Tag is registered, set to armed mode, and moved. When the device is moved, it starts to send positional data, which is displayed in the Pharos dashboard after a few seconds. Additionally, the same request was sent with credentials of a normal user without the specific permissions, which can be seen in Figure 6.4. The same method was used to assure that only the administrator can create a PIN for a DevEUI, which corresponds to *AC-5*. If a user who has no superuser rights (and thus, is not authorized) tries to access the PIN generation functionality, a redirect to the login page is returned (see the Burp screenshot in Figure 6.2). The PIN functionality is only shown if superuser credentials and the second factor are successfully entered. The Burp screenshot shown in Figure 6.3 shows the difference to the first case. The tests confirmed that the website granted access to the required degree specified in *AC-1* to *AC-5*.

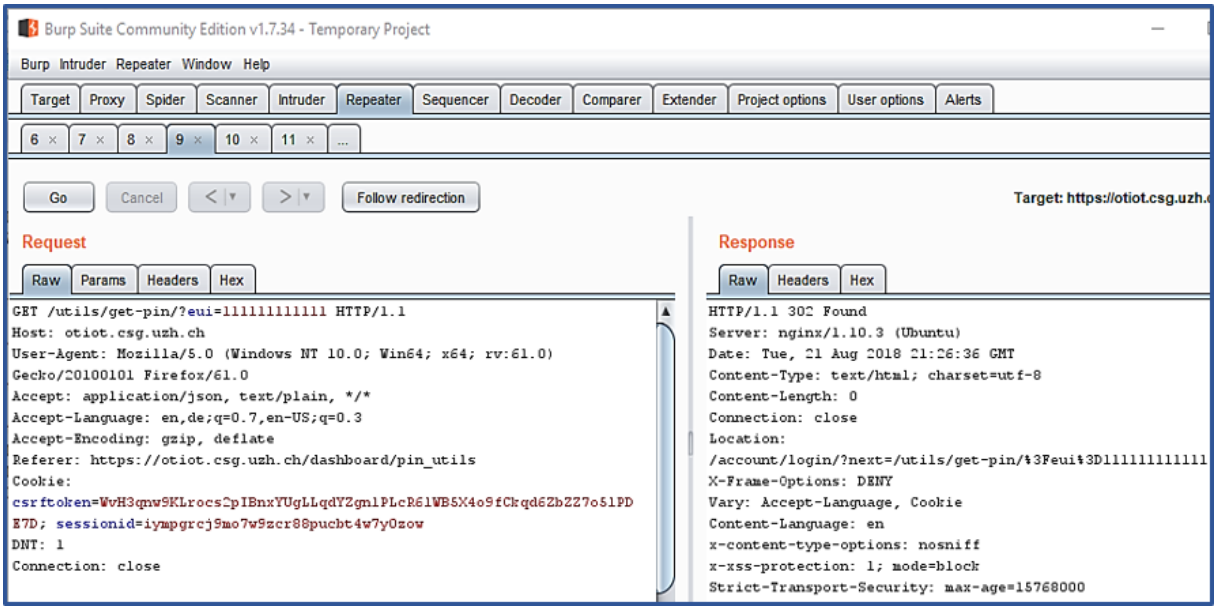

Figure 6.2: HTTP GET Request and Response for PIN Without Authorization

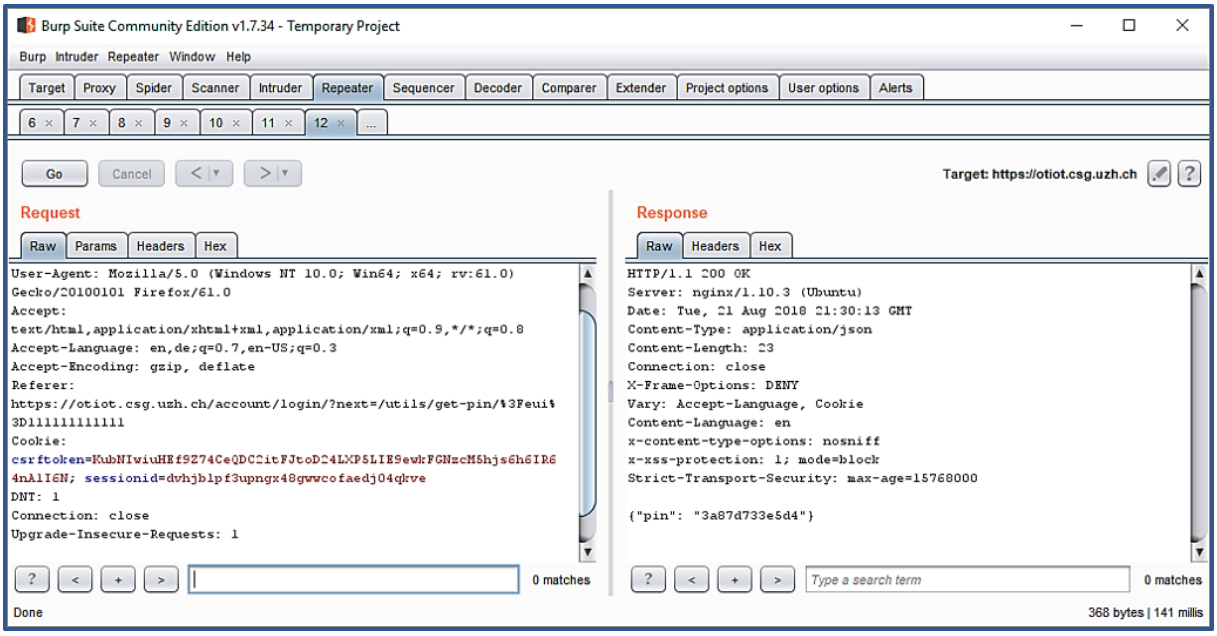

Figure 6.3: HTTP GET Request and Response for PIN from Superuser

B User Authentication (V3)

The first requirement concerning user authentication states that all displayed two-factor authentication methods must be choosable (*UA-1*. It was fulfilled by deleting the phone-based authentication method. The now available methods can be seen in Figure 6.5.

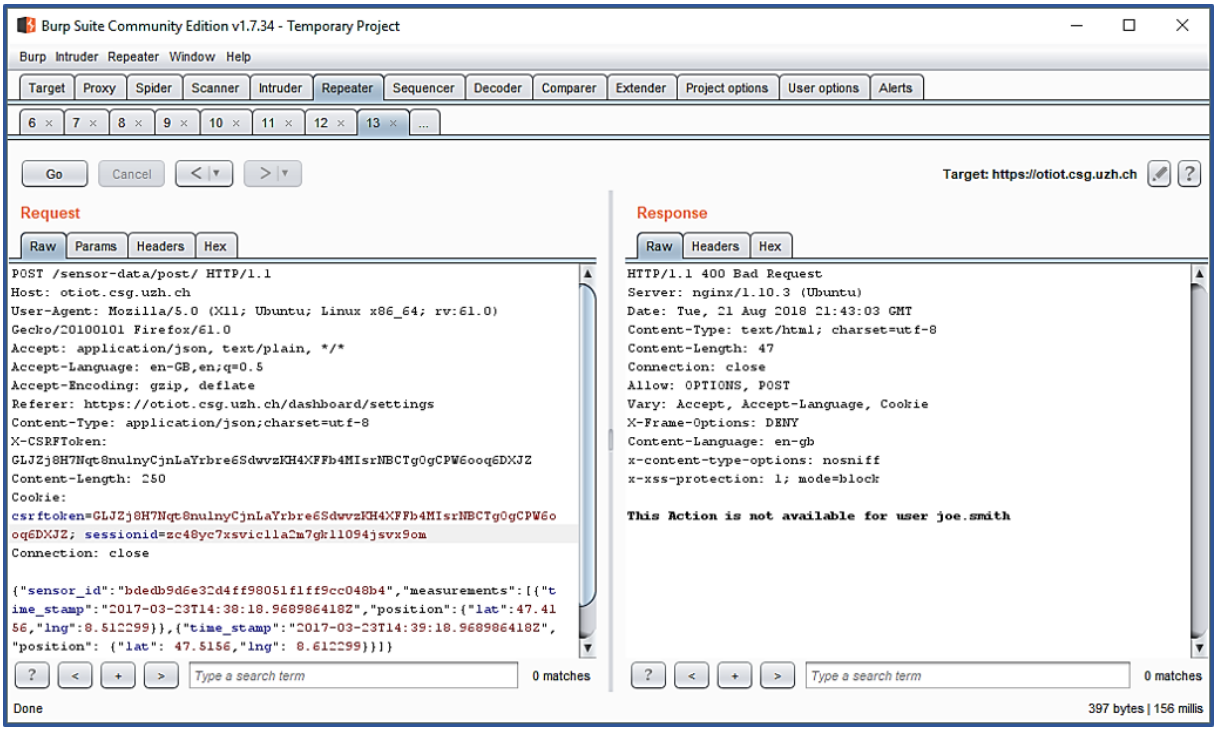

Figure 6.4: HTTP POST Request and Response for Tag Position Data

The email-based two-step authentication was tested through the website. After selecting the email option, the user is logged in without further steps. After that first login, subsequently, the user needs to enter the token sent to the user's email address into the website, as seen in Figure 6.6, as a second authentication step.
The page view for entering the email OTP is the same as for the token generator option. Similarly to the token generator token, the email token must also be entered within a minute. The token generator was already working in the initial OTIoT project and still performs successfully. With this, *UA-1* is deemed correctly implemented.
For the requirement *UA-2*, as a replacement to the phone-based methods, the email-based one-time password approach was implemented, as explained in Section 5.3. Finally, for requirement *UA-3*, it was ascertained that users without two-factor authentication activated could not access pages, except for the login and setup of the two-factor authenticaion. With a manual check of the website pages this could be confirmed, which fulfills requirement *UA-3*. If a page, such as the settings, is requested by a user that has not set up two-factor authentication, an error message is shown, as can be seen in Figure 6.7.

C Tag Registration (V2)

The problem of the Tag registration was approached with the choice of a secure HMAC algorithm to create an activation PIN. The probability that two HMACs of two different messages (which is the DevEUI in our applied algorithm) is negligible. The probabiliy that two PINs are the same because only 12 digits from the generated HMAC were taken exists but is also small with $N = 36^{12}$ different PINs available. The purpose of the PIN

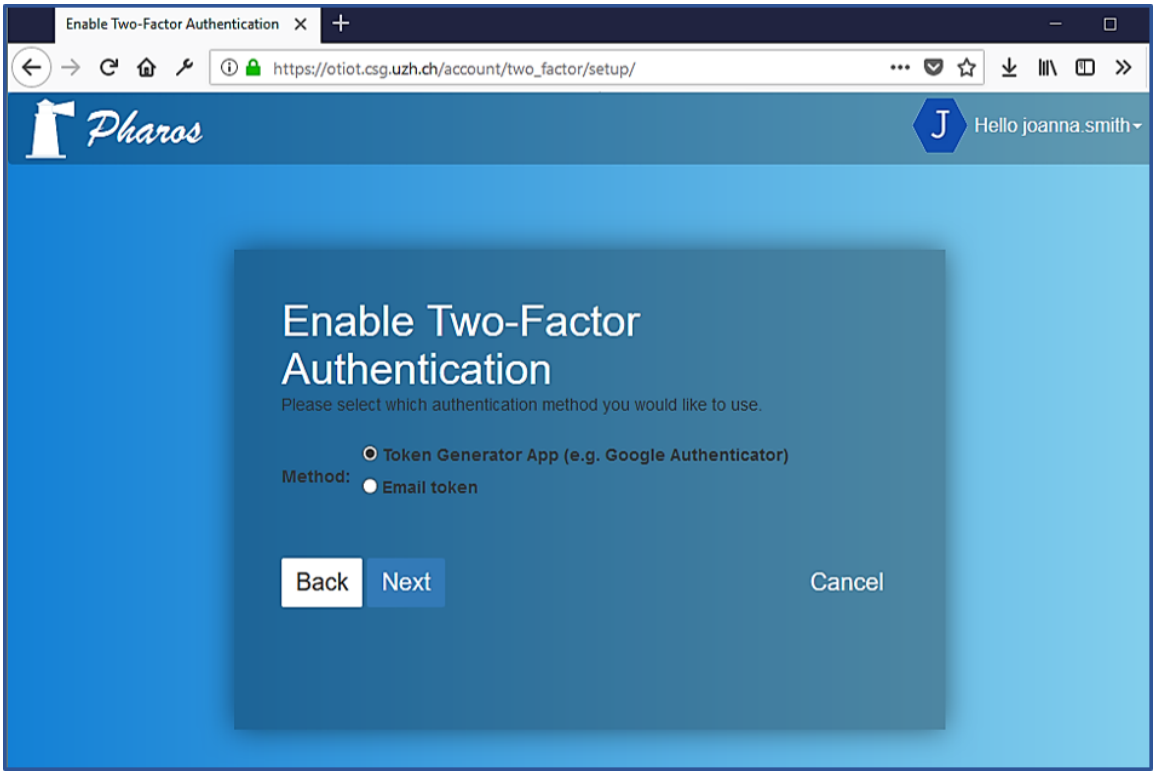

Figure 6.5: Choices for Two-factor Authentication Method

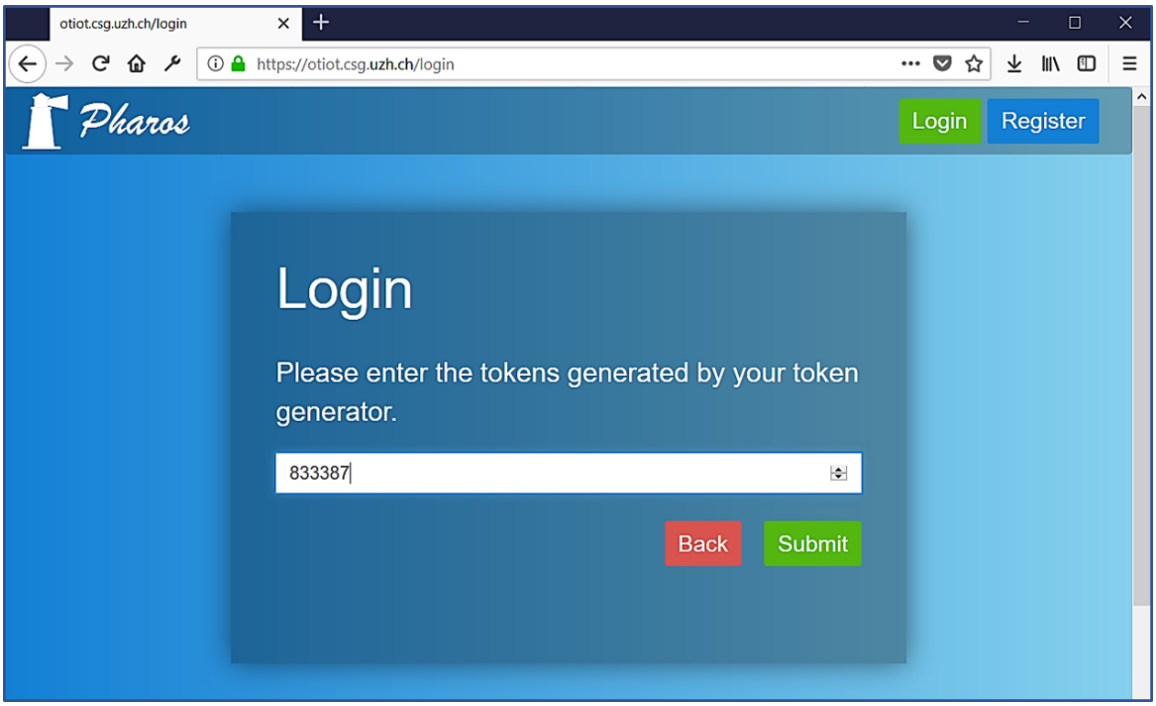

Figure 6.6: Email-based Authentication with Token

was not to be unique, but to be hard to guess and dependent on the DevEUI. So, for better usability, the PIN was limited to 12 digits. If the uniqueness or length of the PIN became a concern in the future, longer PINs might be considered. Essentially, the PIN

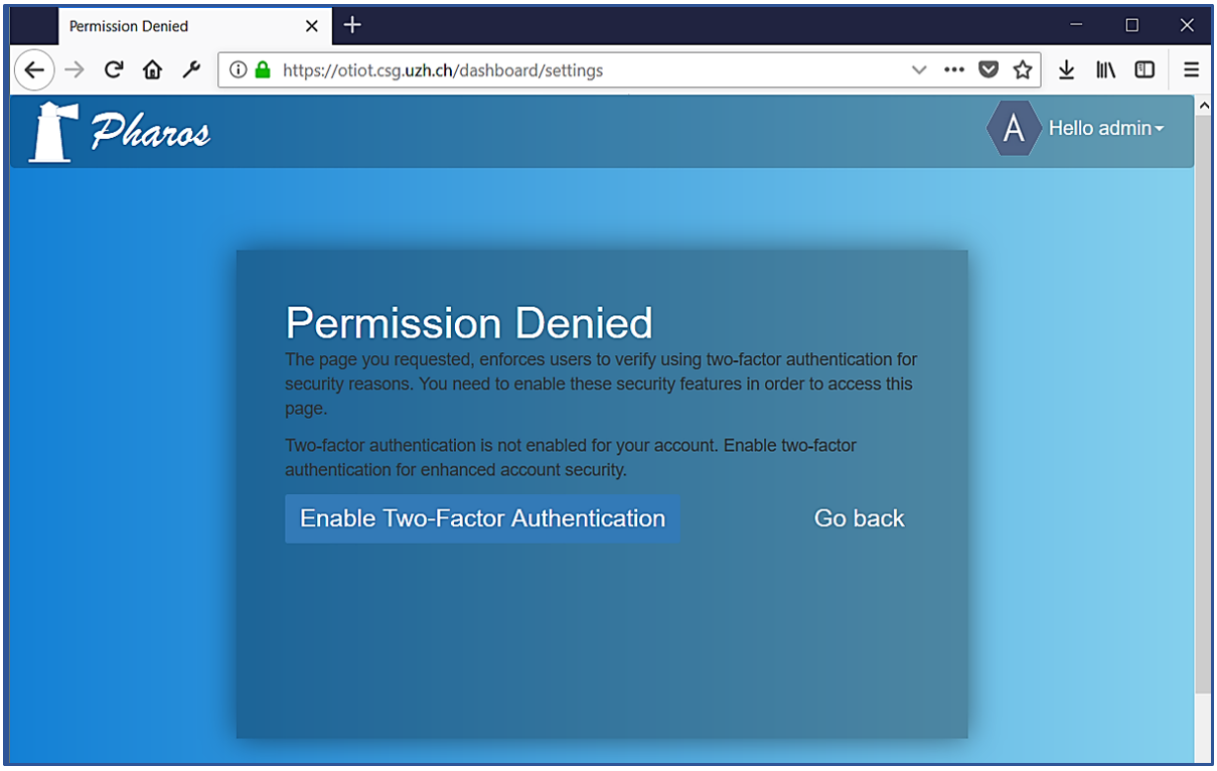

Figure 6.7: Permission Denied for Unauthorized User

functionality conforms to *T-1*, which requests a method to authenticate Tag owners. If the wrong PIN or an empty PIN is entered for a Tag, an error is returned, as can be seen in Figure 6.8. The default Tag ID is chosen to be random and the length of 32 characters is enough to satisfy requirement *T-2*, requiring Tag IDs to be hard to guess. The justification for that is the same as used in the choice for longer passwords in the default configuration setup, see Section G.

D Session Management (V1)

The requirements *S-1*, *S-3*, and *S-4* (session tokens are generated randomly, are not included in the URL, and are invalidated at logout) are default behaviour for the session tokens if the Django session management has been configured correctly, as was checked in the implementation in Section 5.5.2. For the requirement *S-5*, the chosen strategy was to deploy a session token expiration after 30 minutes, as explained in the discussion in Section 4.3.1. The tokens are stored in the database. If the security of the database is assumed, as was done already in Meier's thesis [39], requirement *S-2* is considered fulfilled. As explained in Section 3.3.1, the security of the Docker containers (including the database) is dependant on the security of the Let's Encrypt proxy and the hosting of the Docker environment.

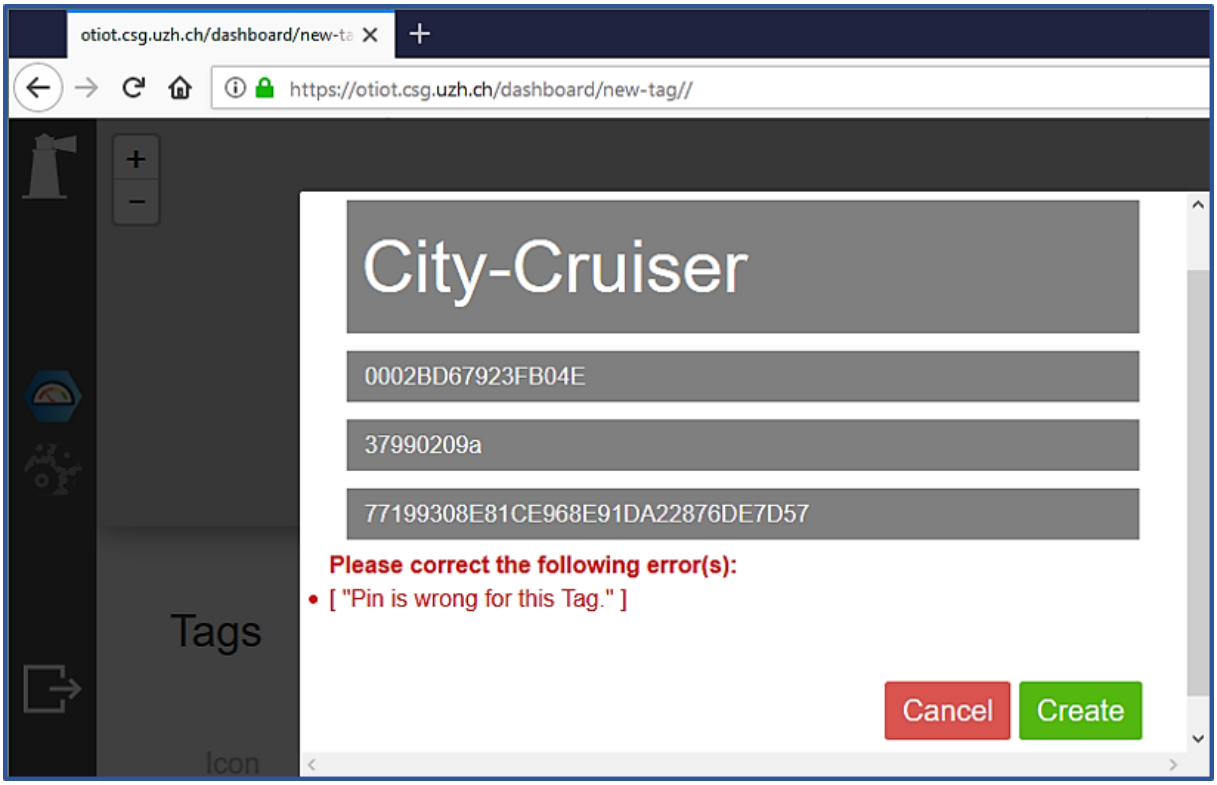

Figure 6.8: Wrong PIN Entry

E File Upload (V5)

In Section 4.2.2, the deletion of the file upload functionality was chosen as the most secure and practical approach to mitigate the vulnerability issue detected in Section 3.2.3. This approach satisfies both *F-1*, which requires that unauthenticated users cannot upload any files, and *F-2*, as no invalid or harmful files can be uploaded anymore.
The proof for that can be seen in Figure 6.9, where a `POST` request for a file upload was sent to the backend. As the path does not lead to any URL that is registered with Django, a 'Page not found' error is shown.

F TTN Traffic Authentication (V6)

The requirement for a dedicated user role for TTN authentication has been implemented in the project with the creation of the `ttn_http_api` user, as seen in Section 5.2.3. The credentials of the user in the TTN console were exchanged for the ones of the `ttn_http_api`, thus satisfying *TA-1*. The solution derived in the design chapter for *TA-2* could only be implemented partly in the oTIoT software, as discussed in Section 4.2.3, because of dependencies on TTN interfaces that do not yet provide the required functionality.

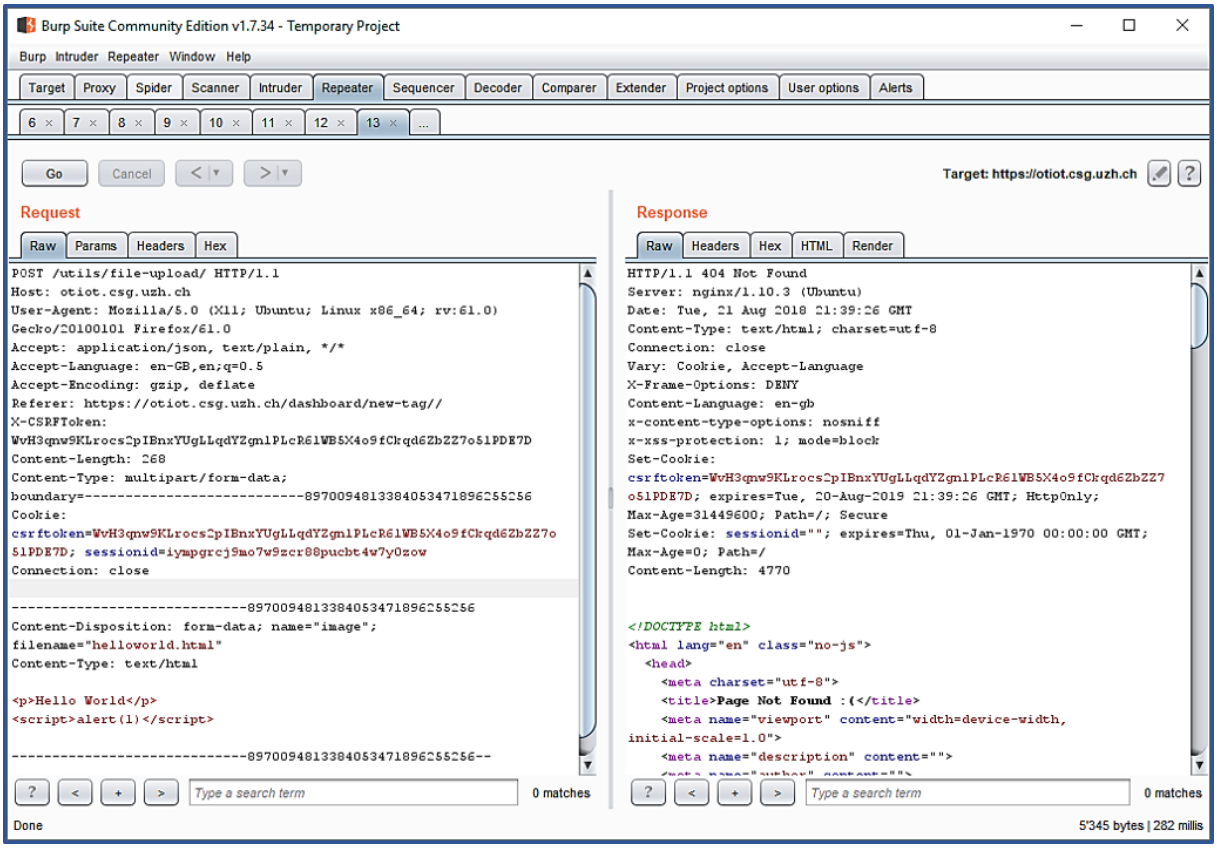

Figure 6.9: HTTP POST Request and Response for HTML File Upload

G Default Configuration (V7)

As mentioned in part G of Section 3.2.3, the default usernames and passwords were chosen notably insecure. One example for this is the PostgresSQL user 'docker' that is specified in the .env configuration file. The passwords before this change were set to be the same as the username, which is easily guessable. It is reasonable to say that one of the first attempts to guess a password would be to try the username.
Additionally, the administrator password had been leaked to TTN, so it was also exchanged for a new, more secure one.
To address *DC-1*, the new passwords were chosen to make them hard to crack using brute-force attacks, according to probability theory. The argument is based on the entropy that is calculated from the way the passwords are created. The formula for calculating the entropy H is the following:

$$H = log_2 N^L \tag{6.1}$$

In this case, $N = 62$, as every position of the password is randomly chosen from 62 possible characters. L stands for the length of the password, specifically $L = 20$. Each password has the same likelihood of being chosen because the password is created at random. That means for one guess of the password, the probability that it's correct is $p = 1/k$ with k being the number of possible keys. As we take a look at the password formed in bits, we can say $k = N^L$. Assuming an attacker would have to try half of all possible passwords on average before guessing the right one, he would need to try out

approximately $n = 3.52 * 10^{35}$ passwords. The strengthening of the default configurations, specifically the user configurations, that corresponds to criteria *DC-1* is thus seen as satisfied.

H Dependency Management (V8)

The purpose of updating the python dependencies was mainly to change an unsupported, insecure Django version, namely 1.10.5, to the LTS release 1.11.13 with security support for an extended time period, as can also be seen in Table 5.5.3. Consequently, requirement *DM-1* is considered to be fulfilled. Several JavaScript dependencies with known vulnerabilities were updated. However, there were seven issues that could not be fixed this way, so the project still includes few components with known security issues. Among them, six are rated with a low risk, one with a critical risk pertaining to command injection. Although the evaluation criteria *DM-2* could not be fulfilled completely, it is safe to say that components with known vulnerabilities were significantly reduced, as can be seen in the comparison of the NPM audit files, which can be found on the CD submitted with the thesis. The risk of the remaining components with vulnerabilities is bearable. A criterion which would be more sensible for this project would be to reduce the usage of components with known vulnerabilities as far as possible.

6.2 Limitations

It can be seen from the criteria that the evaluation is done on a functional level. Some of the solved issues were verified by manual checking on the website, with the same methods used during the security analysis. The numerous vulnerabilities that were found this way had a significant impact on the security of OTIoT. However, there are far more complex methods that can dig deeper into the system and find more vulnerabilities.

More extensive penetration testing of the OTIoT system by an expert, for instance, would bear a more comprehensive outlook on the security issues within the application. It cannot be ruled out that such techniques would find gaps in the implemented features.
Besides that, a more thorough security audit could also include the configuration of the virtual machine and even the physical host on which OTIoT is deployed. Thus, the result of this security evaluation must be considered with knowledge of these limitations.
One example of a security risk that was out of scope, as defined in Section 3.2.1, is that data is not encrypted end-to-end. This concern was briefly mentioned in Section 3.2.2. While traffic between the Tag and TTN is encrypted, uplink data sent from the Tag is decrypted at the Handler before it is sent to the OTIoT backend. While that data transfer is done over HTTPS and thus, encrypted again, the data is seen in clear text in TTN. The same applies for downlink data.

Another limitation that must be mentioned concerns the two-factor authentication methods. As explained, the authentication demands two different factors be chosen. However, strictly speaking, the choice of using email can be seen as the same factor as the password,

namely the factor of 'knowing a secret'. The reason for this is that anyone knowing the password for the email account of the user can access the OTP that is sent for authentication. This means if a user selects the email method, it is a two-step authentication instead of a two-factor one [62]. The difference for this project is deemed small.
However, the framework that provides the `EmailDevice` implementation used in this authentication warns that the email step does not guarantee any security benefit if the website offers password reset through email. The author of this thesis disagrees in part because in the scenario of an account take-over via password guessing, the user is still protected with another layer should the password be found out. The attacker would also have to overcome the hurdle of breaking the email account of the user. If this risk was not acceptable, the password reset could be performed over another channel, such as answering pre-defined security questions. Ultimately, one can say that a two-step method does usually not provide the same protection as a two-factor method, but it does provide more security compared to a one-step password-based authentication.

Chapter 7

Summary and Conclusions

In this master thesis, the OTIoT system developed at the University of Zurich has been introduced, analysed for security vulnerabilities, and adapted to mitigate found risks. Furthermore, the implemented changes have been evaluated as to whether they solve the issues detected.

The focus in the security analysis of the system was put on the communication between the components as well as on the Pharos website, specifically the HTTP endpoints the Django backend provides. Additionally, some security concerns from past developers of OTIoT were included. The result of this analysis provided a list of security issues that were prioritized according to potential damage and the perceived risk of an attack being carried out successfully. During the analysis, OTIoT was found to lack proper access mechanisms for the stored data. Because the nature of the data is sensitive to leakage, the priority was high for fixing any kind of unauthorized access to data and manipulation of data. Other focus points lay on increasing the security of the login process with a fixed two-factor authentication setup, the proper management of session tokens, and the improvement of the Tag registration process.
In the design chapter, recommendations and approaches to solve the problems were laid out for all detected vulnerabilities. Most of the design choices were carried on in the subsequent chapter and integrated into the OTIoT software. The evaluation concretizes which vulnerabilities were mitigated through the implemented changes and lists some limitations for the examination of the system, both in the security analysis, the implementation, and also in the evaluation.

Not all the found security vulnerabilities could be resolved in this thesis. The access to the Application Manager API of TTN over HTTP, which OTIoT utilizes, is not secure. The implementation of this access with MQTT over TLS, as done for the Data API (shown in Section 5.2.3), would fulfill the requirement *TA-2*. As this functionality is not provided by TTN yet, this is considered future work.
Another issue that was not addressed in this thesis is the lack of end-to-end encryption as mentioned in both Section 3.2.2 and in Section 6.2. It will be future work to evaluate the options for mitigating this concern. One example solution is to deploy part of the TTN architecture privately.

For this thesis of special interest would be the Handler and possibly the Broker component because of the integrity checks it handles. TTN provides a guide on how to set up and run parts of the architecture, as can bee seen in [85] and [83].

During the thesis, several challenges resulting from the project setup have been encountered. For one, the OTIoT system was designed and meant to be easy to set up, with basically only needing to run a script to start up the docker containers. As mentioned in the implementation chapter, several changes had to be implemented before the system could be started up. Some parts of the system, namely the file upload or the connection between celery and redis, had to be fixed before they could be examined for security risks. The lack of a thorough documentation made navigating the code and the search for the source of errors harder. Combining that with the wide array of technologies and frameworks used made working with the system challenging. All in all, a lot could be learned about those used components in the course of the thesis and after a while, working with the code became rewarding as the insight into several software and security domains grew.

Due to the fact that this was the first security analysis of the system, it may very well be that several vulnerabilities were not detected. Also, as mentioned in the limitations part in the evaluation chapter, the methodology to detect security risks was rather simple compared to what a determined attacker may use. For future work, it would be interesting to focus on parts of the OTIoT system and analyse it in more depth, for instance the web application could be specifically examined with penetration testing methods.
Lastly, an interesting aspect that was only briefly touched on in the beginning of this thesis pertains to the consequences the EU General Data Protection Regulation, which is applicable as of May 25th, on this project. The assumption made in the introduction was that positional data of a belonging falls into the sensitive data category the regulation protects. If that is the case, the way data is collected and handled in the system probably has to be adapted to conform the regulation. Part of the regulation is already fulfilled, as users have the right to delete their Tag and Tag positional data. However, it is not possible at this time for a user to delete their account from Pharos. A thorough check of how the OTIoT system conforms to the regulation is thus another concern for future work.

Bibliography

[1] J. M. Anderson: *Why We Need A New Definition Of Information Security*, 2003. Computers & Security. 22 (4): 308-313.

[2] Apple Support: *If Your IPhone, IPad, Or IPod Touch Is Lost Or Stolen*, 2018. [Online]. Available: `https://support.apple.com/en-us/ht201472`. Accessed: June 26, 2018.

[3] L. Balcerzak: *Django-Guardian Overview*. [Online]. Available: `https://django-guardian.readthedocs.io/en/stable/overview.html`. Accessed: July 27, 2018.

[4] Celery Project: *Celery: Distributed Task Queue*. [Online]. Available: `http://www.celeryproject.org/`. Accessed: July 30, 2018.

[5] CVE Details: *Django 1.10.5: Security Vulnerabilities*. [Online]. Available: `https://www.cvedetails.com/vulnerability-list/vendor_id-10199/product_id-18211/version_id-211636/Djangoproject-Django-1.10.5.html`. Accessed: August 13, 2018.

[6] Q. Dang: *Recommendation For Applications Using Approved Hash Algorithms*, NIST SP 800-107, 2012. [Online]. Available: `https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-107r1.pdf`. Accessed: July 24, 2018.

[7] T. Dierks and E. Rescorla: *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246 (Proposed Standard), IETF, Fermont, CA, USA, August 2008.

[8] M. Diez: *Secure Position Data Transmission For Object Tracking Using LoRaWAN*, Master's thesis, Department of Informatics (IFI), University of Zurich (UZH), 2017.

[9] Django Software Foundation: *Django-Admin And Manage.py*. [Online]. Available: `https://docs.djangoproject.com/en/1.11/ref/django-admin/#clearsessions`. Accessed: July 24, 2018.

[10] Django Software Foundation: *Django 1.10 Release Notes*. [Online]. Available: `https://docs.djangoproject.com/en/1.10/releases/1.10/`. Accessed: July 24, 2018.

[11] Django Software Foundation: *File Uploads*. [Online]. Available: `https://docs.djangoproject.com/en/1.11/topics/http/file-uploads/`. Accessed: August 1, 2018.

[12] Django Software Foundation: *How To Use Sessions*. [Online]. Available: `https://docs.djangoproject.com/en/1.11/topics/http/sessions/`. Accessed: July 24, 2018.

[13] Django Software Foundation: *Meet Djano*. [Online]. Available: `https://www.djangoproject.com/`. Accessed: August 17, 2018.

[14] Django Software Foundation: *Migrations*. [Online]. Available: `https://docs.djangoproject.com/el/1.11/topics/migrations/`. Accessed: August 12, 2018.

[15] Django REST Framework API Guide: *Permissions*. [Online]. Available: `https://www.django-rest-framework.org/api-guide/permissions/`. Accessed: July 27, 2018.

[16] Django Software Foundation: *Security In Django*. [Online]. Available: `https://docs.djangoproject.com/en/1.11/topics/security/`. Accessed: July 27, 2018.

[17] Django REST Framework API Guide: *Serializers*. [Online]. Available: `https://www.django-rest-framework.org/api-guide/serializers/`. Accessed: July 27, 2018.

[18] Django Software Foundation: *Settings*. [Online]. Available: `https://docs.djangoproject.com/en/1.11/ref/settings/#secret-key`. Accessed: August 18, 2018

[19] Django Software Foundation: *User Authentication In Django*. [Online]. Available: `https://docs.djangoproject.com/en/2.1/topics/auth/`. Accessed: August 18, 2018.

[20] Django REST Framework API Guide: *ViewSets*. [Online]. Available: `https://www.django-rest-framework.org/api-guide/viewsets/`. Accessed: July 27, 2018.

[21] Docker: *Compose File Version 2 Reference*. [Online]. Available: `https://docs.docker.com/compose/compose-file/compose-file-v2/`. Accessed: August 18, 2018.

[22] Docker: *What Is Docker?*. [Online]. Available: `https://www.docker.com/what-docker`. Accessed: July 30, 2018.

[23] Eclipse Paho: *Python Client*. [Online]. Available: `http://www.eclipse.org/paho/clients/python/`. Accessed: August 19, 2018

[24] The European Parliament and the Council of the European Union: *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*, Official Journal of the European Union, Vol. L119, pp. 1-88, May 2016

[25] R. Fielding: *Architectural Styles And The Design Of Network-based Software Architectures*, Doctoral dissertation, University of California, Irvine, 2000.

[26] First.org, Inc.: *Common Vulnerability Scoring System SIG*. [Online]. Available: `https://www.first.org/cvss/`. Accessed: August 13, 2018.

[27] Gartner IT Glossary: *Internet Of Things Defined - Tech Definitions By Gartner*, 2018. [Online]. Available: `https://www.gartner.com/it-glossary/internet-of-things/`. Accessed: June 26, 2018.

[28] Google Account Help: *Create A Strong Password*. [Online]. Available: `https://support.google.com/accounts/answer/32040?hl=en`. Accessed: August 7, 2018.

[29] Google Ideas, Arbor Networks: *Digital Attack Map*, 2013. [Online]. Available: `https://www.digitalattackmap.com/`. Accessed: July 21, 2018.

[30] Google Inc. *About GRPC*. [Online]. Available: `https://grpc.io/about/`. Accessed: July 30, 2018.

[31] T. Graham: *Django's Roadmap*, 2015. [Online]. Available: `https://docs.python.org/2/library/uuid.html`. Accessed: August 8, 2018.

[32] Gulpjs: *Gulp API Docs*, 2015. [Online]. Available: `https://github.com/gulpjs/gulp/blob/v3.9.1/docs/API.md`. Accessed: August 7, 2018.

[33] B. Haarsma: *Django Two-Factor Authentication* on GitHub. [Online]. Available: `https://github.com/Bouke/django-two-factor-auth`. Accessed: August 7, 2018.

[34] B. Haarsma: *Django Two-Factor Authentication Project Description*. [Online]. Available: `https://pypi.org/project/django-two-factor-auth/1.7.0/#description`. Accessed: July 27, 2018.

[35] *X.800: Security Architecture For Open Systems Interconnection For CCITT Applications*, ITU-T Recommendation X.800. 1991. [Online]. Available: `https://www.itu.int/rec/T-REC-X.800-199103-I/e`. Accessed: July 21, 2018.

[36] JWT.IO: *Introduction To JSON Web Tokens*. [Online]. Available: `https://jwt.io/introduction/`. Accessed: August 7, 2018.

[37] Let's Encrypt: *About Let's Encrypt*, [Online]. Available: `https://letsencrypt.org/about/`. Accessed: July 30, 2018.

[38] LoRa Alliance™: *What is the LoRaWAN™ Specification?*. [Online]. Available: `https://lora-alliance.org/about-lorawan`. Accessed: August 18, 2018.

[39] J. Meier: *Design, Implementation, And Evaluation Of An Object Tracking Motion Detection System*, Master's thesis, Department of Informatics (IFI), University of Zurich (UZH), 2017.

[40] P. Mell, K. Scarfone, and S. Romanosky: *A Complete Guide To The Common Vulnerability Scoring System Version 2.0*. [Online]. Available: `https://www.first.org/cvss/v2/guide`. Accessed: August 18, 2018.

[41] Microsoft: *Common Types Of Network Attacks*. [Online]. Available: `https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc959354(v=technet.10)`. Accessed: July 21, 2018.

[42] Microsoft: *Find And Lock A Lost Windows Device*, 2017. [Online]. Available: https://support.microsoft.com/en-us/help/11579/microsoft-account-find-and-lock-lost-windows-device. Accessed: June 26, 2018.

[43] MQTT.org: *Frequently Asked Questions*. [Online]. Available: http://mqtt.org/faq. Accessed: August 18, 2018.

[44] Nginx: *What Is A Reverse Proxy Server?*. [Online]. Available: https://www.nginx.com/resources/glossary/reverse-proxy-server/. Accessed: July 30, 2018.

[45] *Standards For Security Categorization Of Federal Information And Information Systems*, NIST FIPS 199, 2004. [Online]. Available: https://csrc.nist.gov/publications/detail/fips/199/final. Accessed: July 21, 2018.

[46] NIST National Vulnerability Database: *Vulnerability Metrics*. [Online]. Available: https://nvd.nist.gov/vuln-metrics/cvss. Accessed: August 18, 2018.

[47] Node Package Manager Documentation: *About Security Audits*. [Online]. Available: https://docs.npmjs.com/getting-started/running-a-security-audit. Accessed: July 24, 2018.

[48] Open Source Initiative: *The MIT License*. [Online]. Available: https://opensource.org/licenses/MIT. Accessed: August 7, 2018.

[49] Open Web Application Security Project: *Brute Force Attack*, 2016. [Online]. https://www.owasp.org/index.php/Brute_force_attack. Accessed: August 12, 2018.

[50] Open Web Application Security Project: *Category: Attack*. [Online]. Available: https://www.owasp.org/index.php/Category:Attack. Accessed: July 21, 2018.

[51] Open Web Application Security Project: *Category: Threat Modeling*, 2018. [Online]. Available: https://www.owasp.org/index.php/Category:Threat_Modeling. Accessed: July 21, 2018.

[52] Open Web Application Security Project: *Cross-Site Request Forgery (CSRF)*, 2018. [Online]. Available: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF). Accessed: August 18, 2018.

[53] Open Web Application Security Project: *Defense In Depth*. [Online]. Available: https://www.owasp.org/index.php/Defense_in_depth. Accessed: August 7, 2018.

[54] Open Web Application Security Project: *Man-in-the-Middle Attack*, 2015. [Online]. Available: https://www.owasp.org/index.php/Man-in-the-middle_attack. Accessed: July 26, 2018.

[55] Open Web Application Security Project: *OWASP Top 10 - 2017*. [Online]. Available: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_\%28en\%29.pdf.pdf. Accessed: July 21, 2018.

[56] Open Web Application Security Project: *Welcome To OWASP*. [Online]. Available: `https://www.owasp.org/index.php/Main_Page`. Accessed: July 24, 2018.

[57] Portswigger Web Security: *Burp Suite Scanner*. [Online]. Avaliable: `https://portswigger.net/burp`. Accessed: July 24, 2018.

[58] Prey: *Track & Find Your Stolen Phones, Laptops & Tablets | Prey Anti-theft*, 2018. [Online]. Available: `https://www.preyproject.com/`. Accessed: June 26, 2018.

[59] Python Software Foundation: *UUID Objects According To RFC 4122*. [Online]. Available: `https://docs.djangoproject.com/en/1.10/releases/1.10/`. Accessed: July 24, 2018.

[60] Redis Labs: *Redis*. [Online]. Available: `https://redis.io/`. Accessed: July 30, 2018.

[61] M. Rouse and M. Bacon: *Security*, 2017. [Online]. Available: `https://searchsecurity.techtarget.com/definition/security`. Accessed: July 26, 2018.

[62] M. Rouse, I. Wigmore: *Two-Step Verification*. [Online]. Available: `https://searchsecurity.techtarget.com/definition/two-step-verification`. Accessed: August 20, 2018.

[63] P. Sagerson: *Django OTP Project Description*. [Online]. Available: `https://pypi.org/project/django-otp/#description`. Accessed: July 27, 2018.

[64] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh: *Technical Guide To Information Security Testing And Assessment*, NIST SP 800-115. 2008, [Online]. Available: `https://csrc.nist.gov/publications/detail/sp/800-115/final`. Accessed: July 24, 2018.

[65] J. Schiller: *Mobile Communications*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2003.

[66] C. Schmitt, J. Meier, M. Diez, and B. Stiller: *OTIoT - A Browser-based Object Tracking Solution For The Internet Of Things*, IEEE 4rd World Forum on Internet of Things (WF-IoT 2016), New Vork, NY, USA, pp. 1-6, February 2018.

[67] R. Shirley: *Internet Security Glossary, Version 2*, RFC 4949 (Informational), IETF, Fremont, CA, USA, August 2007.

[68] J. Stokking: *The Things Network Architecture*, May 2017. [Online]. Available: `https://www.thethingsnetwork.org/article/the-things-network-architecture-1`. Accessed: August 17, 2018.

[69] The Things Network: *API Reference*. [Online]. Available: `https://www.thethingsnetwork.org/docs/applications/mqtt/api.html`. Accessed: August 18, 2018.

[70] The Things Network: *Application Manager API*. [Online]. Available: `https://www.thethingsnetwork.org/docs/applications/manager/`. Accessed: July 30, 2018.

[71] The Things Network: *Communities*. [Online]. Available: https://www.thethingsnetwork.org/community. Accessed: July 30, 2018.

[72] The Things Network: *HTTP Integration*. [Online]. Available: https://www.thethingsnetwork.org/docs/applications/http/. Accessed: July 30, 2018.

[73] The Things Network: *Learn*. [Online]. Available: https://www.thethingsnetwork.org/docs/. Accessed: July 30, 2018.

[74] The Things Network: *MQTT*. [Online]. Available: https://www.thethingsnetwork.org/docs/applications/mqtt. Accessed: August 7, 2018.

[75] The Things Network: *Network Architecture*. [Online]. Available: https://www.thethingsnetwork.org/docs/network/architecture.html. Accessed: July 30, 2018.

[76] The Things Network: *Network Security*. [Online]. Available: https://www.thethingsnetwork.org/docs/network/security.html. Accessed: July 30, 2018.

[77] The Things Network: *Packet Forwarders*. [Online]. Available: https://www.thethingsnetwork.org/docs/gateways/packet-forwarder/. Accessed: August 18, 2018.

[78] The Things Network: *Support*. [Online]. Available: https:www.thethingsnetwork.org/support. Accessed: June 26, 2018.

[79] The Things Network: *The Things Network Application SDK For Python* on GitHub. [Online]. Available: https://github.com/TheThingsNetwork/python-app-sdk. Accessed: August 16, 2018.

[80] S. Thierlman and C. Johnston: *Major Cyber Attack Disrupts Internet Service Across Europe And US*, The Guardian, 2016. [Online]. Available: https://www.theguardian.com/technology/2016/oct/21/ddos-attack-dyn-internet-denial-service. Accessed: July 21, 2018.

[81] Tile Inc.: *Finden Sie Mit Der Tile App Und Dem Bluetooth-Tracker Ihre Schlüssel, Ihre Brieftasche Oder Ihr Handy | Tile*, 2018. [Online]. Available: https://www.thetileapp.com/de-de/. Accessed: June 26, 2018.

[82] Ubuntu Documentation: *CronHowto*, 2016. [Online]. Available: https://help.ubuntu.com/community/CronHowto. Accessed: August 7, 2018.

[83] H. Visser: *Deploying A Private Routing Environment With Docker-Compose*. [Online]. Available: https://www.thethingsnetwork.org/article/deploying-a-private-routing-environment-with-docker-compose. Accessed: August 7, 2018.

[84] H. Visser: *Secure MQTT APIs Now Using Let's Encrypt Certificates*. [Online]. Available: https://www.thethingsnetwork.org/forum/t/secure-mqtt-apis-now-using-lets-encrypt-certificates/8774. Accessed: August 7, 2018.

[85] H. Visser: *Setting Up A Private Handler Connected To The Public Community Network*. [Online]. Available: `https://www.thethingsnetwork.org/article/setting-up-a-private-handler-connected-to-the-public-community-network`. Accessed: August 7, 2018.

[86] Yubico: *The YubiKey Series*. [Online]. Available: `https://www.yubico.com/products/yubikey-hardware/`. Accessed: July 24, 2018.

Abbreviations

OTIoT	Object Tracking using the Internet of Things
IoT	Internet of Things
TTN	The Things Network
API	Application Programming Interface
CSG	Communication Systems Group
GPS	Global Positioning System
SMS	Short Message Service
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
IP	Internet Protocol
MQTT	Message Queuing Telemetry Transport
gRPC	gRPC Remote Procedure Call
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
MIC	Message Integrity Check
MITM	Man-in-the-Middle
TLS	Transport Layer Security
CA	Certificate Authority
HTTPS	HTTP Secure
DoS	Denial of Service
DDoS	Distributed Denial of Service
DNS	Domain Name System
OWASP	Open Web Application Security Project
CSRF	Cross-Site Request Forgery
MAC	Message Authentication Code
HMAC	Hash-based MAC
JSON	JavaScript Object Notation

Glossary

OTIoT A system that uses IoT devices to track user's belongings and alarm them in case of theft.

IoT The Internet of Things (IoT) is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment.

Tag The physical IoT device used in OTIoT. Is also used to refer to the `Tag` class or object in the data model, or in the TTN Console view.

Pharos The website used with OTIoT. Users can register and manage their Tags in Pharos and track belongings in real time.

General Data Protection Regulation A regulation created by the European Parliament and Council. It has been updated recently to harmonize data protection laws across Europe and strengthen the rights of natural persons, while penalizing organizations that collect and process sensitive data if they fail to meet the regulations.

Shared Tag A Tag that has been shared with other users of OTIoT. Is also used to refer to the `SharedTag` class or object in the data model.

The Things Network The Things Network is building a network for the Internet of Things. TTN is about enabling low power devices to use long range gateways to connect to an open-source, decentralized network to exchange data with applications.

LoRaWAN The LoRaWAN specification is a Low Power, Wide Area (LPWA) networking protocol designed to wirelessly connect battery operated 'things' to the internet in regional, national, or global networks, and targets key IoT requirements such as bi-directional communication, end-to-end security, mobility and localization services.

LoRa Gateway A gateway equipped to deal with the LoRaWAN protocol.

MQTT A machine-to-machine / Internet of Things connectivity protocol. Designed as a publish / subscribe messaging transport.

OTIoT Backend Consists of Docker containers that provide the OTIoT architecture, including a Let's Encrypt proxy server, a remote proxy server, a PostgreSQL database, a redis message broker, a celery task queue, a Django backend, and a frontend.

REST REST is an architectural style for distributed hypermedia systems. It proposes a set of constraints that are applied to elements within the architecture.

Docker Docker is a containerization framework that is used in the project to separate different OTIoT components and handle communication and shared file space between them.

Docker Container A Docker container is a lightweight, standalone, executable package of software that includes everything needed to run an application.

TTN Console A website over which IoT devices, TTN applications, and TTN gateways can be managed.

TTN Access Key Required to access the TTN APIs. Is generated for each TTN application during creation.

TTN Application Manager API API for managing TTN application and devices that belong to the application. Can be accessed over MQTT or gRPC.

TTN Data API API for sending and receiving messages to and from IoT devices.

Tag ID Identifier to reference Tag devices between TTN and OTIoT.

DevEUI Device Extended Unique Identifier. Used in the LoRa protocol to identify a specific IoT device. Is unique in the context of an TTN application.

App Key Used to derive security keys (for encryption etc.) when an IoT device joins a TTN network.

TTN Router Handles communication between the gateway and the Broker. Schedules downlink communication.

TTN Broker Responsible for communication between the Router and the Handler. Deduplicates incoming uplink data, looks up device and application a message belongs to, does cryptographic checks on data to mitigate replay attacks and to validate the integrity of a message.

TTN Handler Encrypts downlink data and decrypts uplink data. Also applied converter and encoder functions defined in the TTN console.

HTTPS HTTP over a secured TLS connection. Provides confidentiality, integrity, and secrecy.

Security Minimizing the vulnerabilities of assets and resources.

Confidentiality Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information. A loss of confidentiality is the unauthorized disclosure of information.

Integrity Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity. A loss of integrity is the unauthorized modification or destruction of information.

Availability Ensuring timely and reliable access to and use of information. A loss of availability is the disruption of access to or use of information or an information system.

Authentication The assurance that the communicating entity is the one it claims to be.

Data Origin Authenticity The source of sent data is the one it claims to be.

Authorization Process to give access to data or services, or decline it, depending on the rights and privileges of the authenticated user.

Threat A potential for violation of security, which exists when there is an entity, circumstance, capability, action, or event that could cause harm.

Vulnerability A concrete flaw in a IT system that can be exploited.

MITM attack Intercept a connection between two communicating entities and split it into two connections. The traffic between the entities can be read and manipulated.

DoS attack Attack with the goal to hinder of halt the normal function of a service.

Threat Model A model that can be used to optimize network / application / internet security by identifying objectives and vulnerabilities, and then defining countermeasures to prevent, or mitigate the effects of threats to the system.

List of Figures

List of Tables

Appendix A

Setup Guidelines for OTIoT Snvironment

These two Guidelines are derived from the original guide of the OTIoT platform, which can be found on GitHub (`https://github.com/OTIoT/otiot-platform`). It describes how to set up the original OTIoT project. For deployment of the new OTIoT project code, the steps 7-10 of the remote setup or the steps 9-21 for the local setup have to be carried out. This guide assumes that the required programs have already been installed on the machine, among others Docker, Docker-compose, Docker-machine, NPM, Python, Gulp.

Setup of the remote CSG machine

1. Comment out the last `COPY` line out in the Dockerfile with #
2. Go into web folder, in file `run_web.sh` prepend `import os;` to the echo statement that creates the superuser
3. Go to `docker-compose.yml`, add `web_media:/usr/src/app/public/media` to volumes in web
4. In the `web/.env` file, change the redis service to `REDIS_SERVICE=redis`
5. In the file `/web/master_thesis_backend/settings/production.py` change the following:
   (a) Change the host IP, in this case to '192.41.136.236' and include that and '127.0.0.1' in `ALLOWED_HOSTS`
   (b) Set `WEBSERVER_ROOT` to `/usr/src/app/`
   (c) Set `DEBUG = False`
   (d) Add `django_extensions` to `INSTALLED_APPS`
6. In the `set_env.sh` script, append the following lines:

```
If [[ $1 == csg2_dev ]]; then
Echo "export WEB_ENV_FILE=web/.env.csg2_dev"
Echo "export ENV_FILE=.env.csg2_dev"
Exit 0;
fi
```

7. If there is no setup yet of the remote machine run `docker-machine create -driver generic generic-ip-address=192.41.136.236 -generic-ssh-key /.ssh/id_rsa`
   (a) IP address and location of the SSH key to connect to the remote machine depend on configuration
8. Run `eval $(./set_env.sh dev)`
9. Run `deploy_csg2-full.sh` for a new deployment, deleting the backend plus database
10. Run `deploy_csg2.sh` for new deployment of frontend and web, which resets the admin and TTN users

Deployment on local Ubuntu Linux machine

1. Start the program Oracle VM VirtualBox
2. Select the option 'Host Network Manager' in the 'File ' tab
3. Remove any virtual network adapters
4. Change `Makefile` in `./web`
   (a) Exchange all `brew` with `apt-get`
   (b) Change `brew install gdal` to `apt-get install gdal-bin python-gdal python3-gdal`
   (c) Change `brew install libgeoip` to `apt-get install libgeoip-dev`
5. Comment out the last `COPY` line out in the `Dockerfile` with #
6. Go into web folder, in file `run_web.sh` add `import os;` to the echo statement that creates the superuser
7. Open `docker-compose.yml`, add `web_media:/usr/src/app/public/media` to `volumes` in web container configuration
8. In the `web/.env` file, change the redis service to `REDIS_SERVICE=redis`
9. In `set_env.sh` change value `otiot-dev` to `dev` in the dev and local clauses
10. If there is no docker machine created yet, run: `docker-machine create -d virtualbox dev`
11. Else run `docker-machine start dev`
12. Run `eval $(./set_env.sh dev)`
13. Run `cd frontend`
14. Run `yarn install`
15. Run `gulp build`
16. Run `docker-compose build -no-cache frontend`
17. Run `Cd ..`
18. Run `Docker-compose build`
19. Run `Docker-compose up -d`
20. Check successful start with `docker-compose ps`
21. If, for some reason, the letsencrypt-proxy container has the status code exit 137, just run `docker-compose restart letsencrypt-proxy`, it should work then
22. Reach 192.168.99.100

Sometimes, the setup process can fail. If that happens, it can help to restart the docker container nginx and letsencrypt-proxy with `docker-compose restart containername`, in that order. If there still exist errors after waiting for a minute, it would be good to enter the containers with `docker exec -it containername bash` and look at the error files. On the nginx and letsencrpyt-proxy containers, those files can be found in /var/log/nginx/

for server issues. In the letsencrypt-proxy, errors pertaining to the TLS certificates can be found in /var/log/letsencrypt/. In the web container, the error and log files can be found in the /usr/src/app/ folder.

Appendix B

Exploits

For carrying out the exploits, HTTP requests were used often. To capture requests and responses, the Burp Suite was used. Instead of using Burp and trying to capture data, one can also use just another tool to interact with APIs, like Postman. All required requests can be found in the appendix in Chapter C, they just need to be adapted for some specific requests.
The following explanation is a step-by-step guide to replicate the results that were discussed in Section 3.2.3. The exploits can be used on the previous version of the OTIoT website.

Preparation for exploits

1. Start up Burp and set it up as a proxy for the used browser.
2. Register a new user.
3. Activate via email code.
4. Cancel the 2FA setup process.
5. Click on Settings.
6. Look for `PUT /user/rest/user/*/` request in Burp HTTP History, send to repeater via right-click. In this case, *=7.
7. Look for `PUT /user/rest/user_config/7/` request Burp HTTP History, send to repeater.
8. Create a Tag with bogus data.
9. Go to the dashboard on the website.
10. Look for `GET /sensor-data/rest/position_measurements/?uid=*` request in Burp HTTP History, send to repeater. In this case, *=Hans-Bicycle.
11. Log out with that user, create another user account for later actions to simulate a user trying to get access to data of another user.

Find out more about other users

Use the user request:

1. Take the `PUT /user/rest/user/7/` request, change the method from PUT to a GET.

2. Test out `GET` requests one after another, from 1-6 instead of 7.

Or you can use the user_config request:

1. Take the `PUT /user/rest/user_config/7/` request, change the method from `PUT` to a `GET`.
2. Test out `GET` requests one after another, from 1-6 instead of 7.

Take over admin account

1. Find user named 'admin', in this case it was user 6.
2. Take the JSON struct in the response body from the `GET user/rest/user/6/` HTTP request.
3. Exchange the body from the `PUT /user/rest/user/7/` request with the JSON struct from above.
4. Change both the 'email' and the 'has_active_email' attributed to own email address, in this case 'stefanie.ziltener@uzh.ch'.
5. Send the adapted `PUT /user/rest/user/6/` HTTP request.
6. Log out if user is logged in.
7. Enter the administrator username without a password, click login button.
8. Click on 'Reset Password'.
9. Enter the before set email address, in this case 'stefanie.ziltener@uzh.ch'.
10. Go to email account, log in, find the email sent by Pharos.
11. Click on the link to reset the admin password.
12. Submit a new password.
13. Log in with new password and username 'admin'.

Send position measurement data to OTIoT backend with knowledge of Tag ID

1. Create a new `POST` request, sending it to `/sensor-data/post/` with the structure of the one found in the appendix.
2. Exchange the 'sensor_id' with a known ID.
3. Exchange 'lat' and 'lng' to desired values.
4. Send created `POST` request.

File Upload

This exploit can be carried out without being logged in, it just needs a current, valid `csrftoken`.

1. Access the website otiot.csg.uzh.ch.
2. Look in Burp for a `GET /` request, collect `crsftoken`.
3. Create a `POST /utils/file-upload/` request like the one found in the appendix.
4. If the desired file content is of type `text/html`, fill into request and send.
5. If the desired file content is a picture, use the website GUI to upload the file.
6. Collect the response, take URL from it.

7. With otiot.csg.uzh.ch/URL you can access the uploaded file.
8. You can send the link to other people so they, too, can access the file.

Appendix C

HTTP Requests and Responses

To keep the appendix short, only one of each HTTP request / response pair that was used in the exploits is shown here. Other requests can be formed similar to their structure.

User
Request:

```
GET /user/rest/user/2/ HTTP/1.1
Host: 192.168.99.100
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
  rv:61.0) Gecko/20100101 Firefox/61.0
Accept: application/json, text/plain, */*
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https:// 192.168.99.100/dashboard/settings
Cookie: csrftoken=
   AfLT7r4MglNmVLIX8d7LjnXaYtYuy1XpvAiPyAjDCLX
  qGc2ElXuYuaS1u3yMClN2; sessionid=
   zaxb640wc87xo5zvewuyw2ksi1owedjg
Connection: close
```

Response:

```
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Sat, 11 Aug 2018 10:55:20 GMT
Content-Type: application/json
Connection: close
Allow: GET, PUT, PATCH, DELETE, OPTIONS
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: DENY
Content-Language: en-gb
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
Strict-Transport-Security: max-age=15768000
Content-Length: 397
{"url":"https://192.168.99.100/user/rest/user/2/",
"username":"admin","last_name":"","first_name":"",
"email":"pharos.otiot@gmail.com","pk":2,"conf":{
"url":"https://192.168.99.100/user/rest/user_config/2/",
"notify_by_email":false,"notify_by_sms":false,
"has_active_email":"pharos.otiot@gmail.com",
"has_active_phone":false,"avatar_2":
"https://192.168.99.100/media/avatar_admin_Xn2vEk1.png"}}
```

User Config

Request:

```
GET /user/rest/user_config/2/ HTTP/1.1
Host: 192.168.99.100
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
  rv:61.0) Gecko/20100101 Firefox/61.0
Accept: application/json, text/plain, */*
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://192.168.99.100/dashboard/settings
Content-Length: 240
Cookie: csrftoken=
   AfLT7r4MglNmVLIX8d7LjnXaYtYuy1XpvAiPyAjDCLXqGc2ElXuYua
   S1u3yMClN2; sessionid=zaxb640wc87xo5zvewuyw2ksi1owedjg
Connection: close
```

Response:

```
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Sat, 11 Aug 2018 10:52:57 GMT
Content-Type: application/json
Connection: close
Allow: GET, PUT, PATCH, DELETE, OPTIONS
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: DENY
Content-Language: en-gb
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
Strict-Transport-Security: max-age=15768000
Content-Length: 239
{"url":"https://192.168.99.100/user/rest/user_config/2/",
"notify_by_email":false,"notify_by_sms":false,
"has_active_email":"pharos.otiot@gmail.com",
"has_active_phone":false,"avatar_2":
"https://192.168.99.100/media/avatar_admin_Xn2vEk1.png"}
```

Request:

```
PUT /user/rest/user/2/ HTTP/1.1
Host: 192.168.99.100
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv
   :61.0)
  Gecko/20100101 Firefox/61.0
Accept: application/json, text/plain, */*
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://192.168.99.100/dashboard/settings
Content-Type: application/json;charset=utf-8
X-CSRFToken:
   AfLT7r4MglNmVLIX8d7LjnXaYtYuy1XpvAiPyAjDCLXqGc2
  ElXuYuaS1u3yMClN2
Content-Length: 405
Cookie: csrftoken=
   AfLT7r4MglNmVLIX8d7LjnXaYtYuy1XpvAiPyAjDCL
  XqGc2ElXuYuaS1u3yMClN2;
sessionid=zaxb640wc87xo5zvewuyw2ksi1owedjg
Connection: close

{"url":"https://192.168.99.100/user/rest/user/2/",
"username":"admin","last_name":"","first_name":"",
"email":"stefanie.ziltener@uzh.ch","pk":2,"conf":{
"url":"https://192.168.99.100/user/rest/user_config/2/",
"notify_by_email":false,"notify_by_sms":false,
"has_active_email":"stefanie.ziltener@uzh.ch",
"has_active_phone":false,"avatar_2":
"https://192.168.99.100/media/avatar_admin_Xn2vEk1.png"}}
```

Response:

```
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Sat, 11 Aug 2018 11:08:58 GMT
Content-Type: application/json
Connection: close
Allow: GET, PUT, PATCH, DELETE, OPTIONS
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: DENY
Content-Language: en-gb
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
Strict-Transport-Security: max-age=15768000
Content-Length: 403
{"url":"https://192.168.99.100/user/rest/user/2/",
"username":"admin","last_name":"","first_name":"",
"email":"stefanie.ziltener@uzh.ch","pk":2,"conf":
{"url":"https://192.168.99.100/user/rest/user_config/2/",
"notify_by_email":false,"notify_by_sms":false,
"has_active_email":"stefanie.ziltener@uzh.ch",
"has_active_phone":false,"avatar_2":
"https://192.168.99.100/media/avatar_admin_Xn2vEk1.png"}}
```

Position Measurement

Request:

```
GET /sensor-data/rest/position_measurements/?uid=Hans-
   Bicycle HTTP/1.1
Host: 192.168.99.100
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv
   :61.0)
  Gecko/20100101 Firefox/61.0
Accept: application/json, text/plain, */*
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://192.168.99.100/dashboard/
Cookie: csrftoken=
   gr35c0ci9cAwwMiIsre430KoH9liK4OFwrJDhFf2gMy
  vYuGXiILTK1AVIb5O5xtZ;
sessionid=mp2qm4emyw3snhgzciv1uwgrt1bd8we4
Connection: close
```

Response:

```
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Sat, 11 Aug 2018 11:20:45 GMT
Content-Type: application/json
Connection: close
Allow: GET, POST, OPTIONS
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: DENY
Content-Language: en-gb
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
Strict-Transport-Security: max-age=15768000
Content-Length: 52

{"count":0,"next":null,"previous":null,"results":[]}
```

Request:

```
POST /sensor-data/post/ HTTP/1.1
Host: 192.168.99.100
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv
   :61.0)
  Gecko/20100101 Firefox/61.0
Accept: application/json, text/plain, */*
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://192.168.99.100/dashboard/settings
Content-Type: application/json;charset=utf-8
X-CSRFToken:
   QkFKzPujsk800p1vUrBog09SjxJnHsKeZojbT93TviCjzJyp
  SiTDOydjksc4YAxf
Content-Length: 248
Cookie: csrftoken=
   QkFKzPujsk800p1vUrBog09SjxJnHsKeZojbT93TviC
  jzJypSiTDOydjksc4YAxf;
sessionid=gxl2y9bty09zv7v3t3988fap1apbpmd2
Connection: close
{"sensor_id":"Hans-Bicycle","measurements":[{"time_stamp":
"2017-03-23T14:38:18.968986418Z","position":
{"lat":47.4156,"lng":8.512299}},{"time_stamp":
"2017-03-23T14:39:18.968986418Z",
"position": {"lat": 47.5156,"lng": 8.612299}}]}
```

Response:

```
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Mon, 13 Aug 2018 08:31:45 GMT
Content-Type: application/json
Content-Length: 12
Connection: close
Allow: POST, OPTIONS
Vary: Accept, Accept-Language, Cookie
X-Frame-Options: DENY
Content-Language: en-gb
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
Strict-Transport-Security: max-age=15768000
{"status": "ok"}
```

File Upload

Request:

```
POST /utils/file-upload/ HTTP/1.1
Host: 192.168.99.100
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
rv:61.0) Gecko/20100101 Firefox/61.0
Accept: application/json, text/plain, */*
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://192.168.99.100/dashboard/new-tag//
X-CSRFToken:
   d4zTHVqKWlBKUhgPggeRET3qOHHED8ucTN6zOwqMKxZih7D
  aW79rVhYdtVqPUhOH
Content-Length: 268
Content-Type: multipart/form-data;
boundary
   =------------------------897009481338405347189625256
Cookie: csrftoken=
   d4zTHVqKWlBKUhgPggeRET3qOHHED8ucTN6zOwqMKxZih7DaW79
rVhYdtVqPUhOH;sessionid=pkosdq12hjgyz3eirc40ua7r6ilogleo
Connection: close
--------------------------897009481338405347189625256
Content-Disposition: form-data; name="image";
filename="helloworld.html"
Content-Type: text/html
<p>Hello World</p>
<script>alert(1)</script>
--------------------------897009481338405347189625256--
```

Response:

```
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Thu, 02 Aug 2018 16:00:07 GMT
Content-Type: application/json
Content-Length: 37
Connection: close
X-Frame-Options: DENY
Vary: Accept-Language, Cookie
Content-Language: en-gb
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
Strict-Transport-Security: max-age=15768000
{"url": "/media/tmp/helloworld.html"}
```

Appendix D

Django Two-Factor Authentication Framework Changes

The following list is a collection of all changes implemented to the downloaded source code of the Django two-factor authentication framework, as seen in [34].

1. Download the *two-factor* folder from the GitHub repository of the Django Two-factor Authentication framework,
   see `https://github.com/Bouke/django-two-factor-auth`
2. Add the following python modules to the `/web/requirements.txt` file:
   (a) django-formtools
   (b) django-otp>=0.3.4,<0.99
   (c) django-phonenumber-field>=1.1.0,<1.99
   (d) qrcode==4.0.4
3. Delete the 'django-two-factor-auth==1.6.0' module from the `/web/requirements.txt` file
4. Go to `/web/master_thesis_backend/settings/base.py` file and change the following:
   (a) Add 'django_otp.plugins.otp_hotp','otp_yubikey', 'django_otp.plugins.otp_email' to the `THIRD_PARTY_APPS` variable
   (b) Add 'two-factor' to the`LOCAL_APPS` variable
   (c) Add the variable `OTP_EMAIL_SENDER = 'pharos.otiot@gmail.com'`
   (d) Add the variable `OTP_EMAIL_SUBJECT = 'Pharos OTP Token'`
5. Go to `/web/master_thesis_backend/urls.py`, exchange
   `url(r' ', include('two-factor.urls', 'two_factor'))` with
   `url(r' ', include('two-factor.urls', namespace='two_factor'))`
6. In file `/web/two_factor/models.py`, change the following:
   (a) Change function `get_available_phone_methods` to return [ ]
   (b) Add the function:
   ```
   def get_available_email_methods():
       methods = [ ]
       methods.append(('email', _('Email token')))
       return methods
   ```

   (c) Add to the function `get_available_methods():` the line
   `methods.extend(get_available_email_methods())`
7. Go to file `/web/two_factor/views/core.py`
   (a) Add import statement:
   `from django_otp.plugins.otp_email.models import EmailDevice`
   (b) In class `SetupView` add to variable `condition_dict`
   `'email': lambda self: self.get_method() == 'email',`
   (c) Adapt line 296 to include choice 'email' so it looks like:
   `elif self.get_method() in ('call', 'sms', 'email', 'yubikey'):`
   (d) In function `def get_device(self, **kwargs):` add, approximately at line 341:
   `if method in ('email'):`
   `return EmailDevice(**kwargs)`

Appendix E

Contents of the CD

The attached CD consists of the following folders and files:

- `Zusfsg.txt`: Plain text version of the German abstract.
- `Abstract.txt`: Plain text version of the English abstract.
- `Thesis.pdf`: The PDF version of the submitted Master thesis.
- `/latex`: Directory containing the LaTeX source files for the Master thesis.
- `/code`: Directory containing the source code of the OTIoT Platform.
- `/additionalMaterial`: Directory containing additional material.
  - `npm_audit.log`: The current Node Package Manager audit log file.
  - `npm_audit_initial.log`: The initial Node Package Manager audit log file.
  - `/screenshots`: Directory containing screen shots of the initial and final Pharos website.
  - `/exploits`: Directory containing the collection of HTTP requests and the files that were used in the exploits in the security analysis and evaluation.
- `/presentation`: Directory containing the midterm presentation and used graphics.
- `/sources`: Directory containing referenced sources.