UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Siim-Alexander Kütt

# SECURITY ANALYSIS OF TARTU SMART BIKE SHARE ANDROID APPLICATION

Bachelor's Thesis (9 ECTS)

Supervisor Arnis Paršovs

Tartu 2020

## Security Analysis of Tartu Smart Bike Share Android Application

**Abstract:** In June 2019, Tartu City Transport launched a smart bike share system, which allows the residents of Tartu to rent bikes for small commutes around the city. A month after the system first launched a privacy exposure was discovered and personal data of the users was leaked. It was not publicly disclosed where the fault had resided, but it was confirmed to have been fixed shortly after the developers were notified. The aim of this research was to analyze the security of the Tartu Smart Bike Share Android app and its communication with the web service. During the course of the research, several security and privacy issues were found, one of which allows any registered user to query information about the location of a bike and its current user. The thesis provides a general description of the system and its underlying architecture, outlines how and which aspects of the app functionality were analyzed and what results were found. Suggestions for improving the security and privacy aspects of the system are also provided.

**Keywords:**
Privacy, secure authentication, Android application security, web resource security, static code analysis, reverse engineering, HTTP requests

**CERCS:**
P170 - Computer science, numerical analysis, systems, control

## Tartu Rattaringluse Androidi rakenduse turvaanalüüs

**Lühikokkuvõte:** Tartu Linnatranspordi algatusel avati 2019 aasta juunis Tartu Rattaringlus, mis võimaldab Tartus viibijatel linnas liiklemiseks rattaid rentida. Kuu aega peale rattaringluse avamist avastati süsteemis andmeleke, mille tagajärjel lekkisid paljude kasutajate personaalsed andmed. Avalikkusele ei tehtud teatavaks, millises süsteemi osas viga avastati, kuid viga parandati vahetult peale seda, kui arendajaid oli teavitatud. Käesoleva uurimistöö eesmärk on analüüsida Tartu Rattaringluse Androidi rakenduse turvalisust ning antud rakenduse veebiteenusega suhtlemise turvalisust. Uurimistöö käigus avastati mitmeid turva- ning privaatsusriske, millest üks võimaldas igal kasutajal jälgida teise kasutaja rattasõidu kulgu reaalajas. Uurimistöö kirjeldab ka Tartu Rattaringluse süsteemi ja arhitektuuri, tuues välja kuidas ja milliseid süsteemi aspekte analüüsiti ning millised olid

tulemused. Lisaks pakutakse töös ka soovitusi, mille abil Tartu Rattaringluse süsteemi turvalisemaks muuta.

**Võtmesõnad:**
Privaatsus, turvaline autentimine, Androidi rakenduste turvalisus, veebiressursside turvalisus, pöördkonstrueerimine, HTTP päringud

**CERCS:**
P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimise teooria)

# Contents

# 1 Introduction

Tartu Smart Bike Android application was first released in June of 2019 as a part of the bike share system that the City Government of Tartu had installed. The system was installed to promote environmentally friendly mobility and reduce the carbon footprint of the city by allowing its users to conveniently rent public bikes for commuting around the city. The system consists of 750 bikes and a total of 69 stations across the city [1]. The system is available for everyone to use via a smartphone application and follows a simple payment plan. Additionally, it is possible to link a Tartu bus card to the system which allows the use of the bus card to unlock a bike from the station provided that a membership is active or a credit-card is linked.

A month after the system had first launched a privacy exposure was discovered in the system which allowed unauthorized users to view the real names, phone numbers and email addresses of nearly 20 000 users [2]. In addition 7180 personal identity codes were exposed due to the accounts having linked a personal Tartu bus card [2]. It was not publicly disclosed where the vulnerability had resided but the fault was confirmed to have been fixed shortly after the developers were notified [3].

The aim of the thesis is to describe the functionality provided by the service and analyze the security critical functionality of the Android mobile application and its communication with the web service. This involves studying how the application works on an architectural level, enumerating possible security and privacy issues, analyzing different attack scenarios and their results. An iOS app is also provided, but the Android app was chosen because it serves the core functionality of the system and a lot of previous research exists for analyzing Android applications. According to GlobalStats [4] Android OS also has a much greater market share in Estonia.

This thesis will focus on analyzing two different versions of the Tartu Smart Bike Share app. An older version (v18.0) that will be used for code analysis as it can be decompiled to Java code and a newer version (v81.0) that will be used to analyze web resources.

The core of the analysis will consist of code analysis and the corresponding web resource analysis. To conduct the code analysis the application will be disassembled and decompiled back into Java source code and then analyzed. The aim of code analysis is to determine the logic behind the applications user interface and discover potential

security and privacy issues. In order to analyze the web resources, HTTP requests will be constructed targeting the web resource URLs that were determined using code analysis.

The thesis is organized as follows. The next section provides a description of the system and how the main components of the system interact with each other. The subsections provide an overview on the involved parties and their roles, smart bikes and the complementary information system.

Section 3 introduces the object of research and describes the methods and tools that were used to conduct the analysis. The subsections provide a resultative overview of all the security critical functionality that was analyzed. The last section provides a summary of findings ranked by significance.

# 2 Tartu Smart Bike Share system

The following section and subsections introduce the Tartu Smart Bike Share system and describe the involved parties and how different elements of the system work and communicate.

## 2.1 Involved parties and their roles

Tartu City Transport is an institution managed by the Department of Communal Services of the Tartu City Government, which provides the Tartu Smart Bike Share service. Tartu City Transport also provides customer and technical support for the users and employs contractors to assist them in providing the service.

Bewegen Technologies Inc[1] is a Canadian company that has manufactured the smart bikes and stations for the Tartu City Government. In partnership with WeGoShare[2], a Portugese IT solutions company, Bewegen have also developed a complementary website and smartphone application that the users can use to access the system and the city government can use to maintain and operate the system. In cases where remote or on-site technical support is needed these companies will also act as maintainers[3].

Tartu City Government also employs the help of local contractors to move the bikes between stations during the night when the system is closed, perform routine maintenance and repair them as needed. The users of the system are residents of Tartu for the most part, but the system can be used by any visitor as well.

## 2.2 Smart bikes

According to Tartu City Transport Manager Roman Meeksa, there are a total of 750 bikes and what makes them smart is the fact that they exchange information in real-time with the bike share system. In addition, it was mentioned that there are GPS modules on each bike and a 3G internet connection which the bikes use to communicate with the back-end web service. The bikes use GPS to gather data about all the rides made, the total mileage of the bike and the start and end points of every single ride [5]. The bikes also have a

[1] Bewegen Technologies Inc. `https://bewegen.com/en` (03.03.2020)

[2] WeGoShare. `https://wegoshare.com/` (03.03.2020)

[3] WeGoShare job offer. `https://wegoshare.com/careers/support-technician/` (03.01.2020)

digital display on top of the frame that, while riding, shows data about the status of the bike, time spent driving and the total distance covered during the ride.

Along with the bikes the system also has 69 docking stations spread around the city of Tartu [5]. Each of these docking stations have an infographic board with instructions on how to use the system and docking slots where the bikes can be returned. A bike can be docked into any of the free and working docking slots. When a bike is returned, it locks into the station and begins recharging itself.

## 2.3 Functionality provided by the website and smartphone app

The purpose of the website[4] and the app[5] is to make access to the bike share system available to the public in a controlled and maintainable manner providing all the necessary tools for the users to use the system and for the operators to maintain and operate the system.

The users can use the website to register and gain access to the system by filling out their profile and payment details. Once registered the web service provides the users with a map of all the stations located in Tartu and their real-time operational status. The website provides instructions on how to use the system and general statistics about the system such as the most common start and return stations, total mileage across the system for the current year and in total. Users can also view their own bike usage history, payment history and lifetime ride statistics.

It is important to note that the bikes cannot be unlocked using the web service. Instead the users can use the smartphone application to unlock bikes. The smartphone app also includes the rest of the functionality that the website does. Alternatively, users can use the bus card to unlock bikes if they have one linked with their account. This provides convenience when pulling the bike out of the dock.

The maintainers can use the web service to manage the access and profiles of all the registered users. They also have the ability to reset or remove accounts. The maintainers can view information such as name, age, personal identity code, phone number, payment history, count of drives, bus card number and overall account status for each individual user [5].

[4]Tartu Bike share `https://ratas.tartu.ee/`. (12.01.2019).
[5]Tartu Bike share app `https://ratas.tartu.ee/about/app/`. (10.02.2019)

### 2.3.1 Registering and authenticating

To use the system the user must first register as a member. This can either be done in the smartphone application or at their website. The smartphone application can be found from Google Play or iOS App Store under the name Tartu Smart Bike. After launching the application the registration form can be used to create an account.

The registration process is straightforward and only the name, email and password are required from the user. After registering the system still cannot be used without first completing the profile and choosing a payment plan. The users can pay for any of the payment plans using a credit card. Alternatively the users can link any compatible card including the Tartu bus card and use that to pay for the service. If the profile is complete and all the fields are valid the registration can be finalized and the user can start using the service.

To authenticate with the service the user must log in using their email and password. If the login is successful the user is granted access to the system. The user can then use the functionality of the system such as unlocking bikes, viewing past rides and changing profile or membership data. The app also maintains the authentication state of the user as long as the user does not log out of the app.

### 2.3.2 Unlocking bikes

Unlocking a bike from the Android app requires the user to select a dock from which to unlock the bike and enter the bike serial number that is written on the frame. Unlocking a bike with the bus card requires the user to hold the card close to the center of the handlebars. Figure 1 illustrates the process of unlocking a bike through the app.

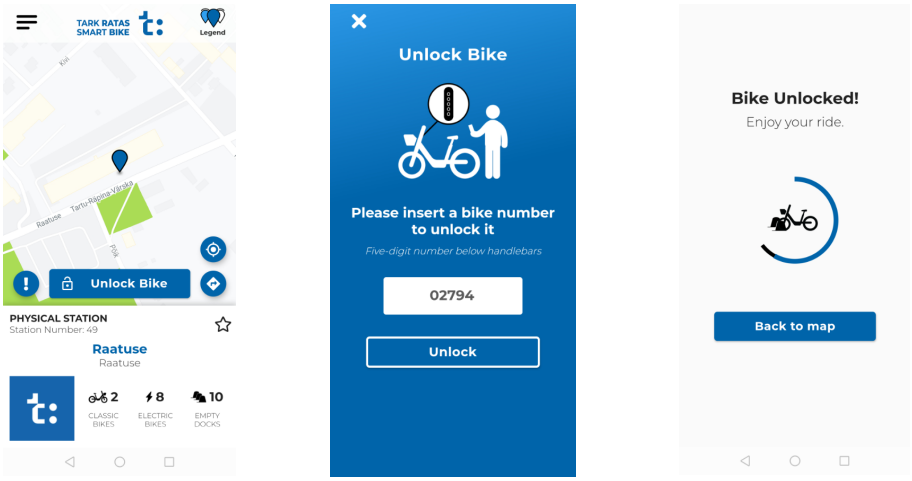



Figure 1. Bike unlocking process using the Android app

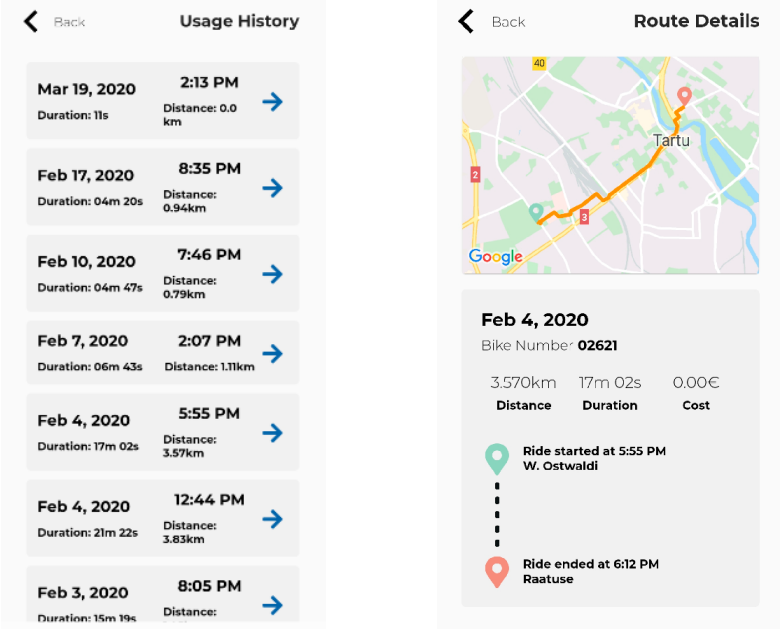
Unlocking the bike from the app prompts the smartphone application to send a request to the back-end web service which then, over a 3G connection, sends a command to the bike telling it to unlock itself. Upon a successful unlock request the bike will be unlocked for 10 seconds. If the bike is not removed from the dock within those 10 seconds the bike will lock itself again and the process must be repeated.

Once the bike is removed from the dock a ride session will begin. During a ride session the bikes will collect information about the route and the state of the bike and send it back to the web service over a 3G connection [5]. In order to end a ride, the bike must be pushed into any of the available docking stations where the bike will then lock itself and emit a beeping sound.

### 2.3.3 Ride details and usage history

Overview of all the routes taken can be viewed from either the website or the mobile application. The overview itself provides only few details of each ride such as the date, distance and duration. Figure 2a illustrates the overview of usage history as seen from the mobile application.

After a ride has ended, details about that specific ride will become available for viewing. The ride details contain data such as the date, bike number, distance, duration, cost and the start and end stations as can be seen in Figure 2b.




(a) General overview of ride history (b) Detailed overview of a single ride

Figure 2. Overview of ride history and ride details

# 3 Security analysis of Android app

The following section and subsections introduce the object of research, describe in detail how the code and web resource analysis of the Android app were conducted, what methods were used, which functionality was tested and what were the results.

Android application security principles are not too different from general software security principles. The same ideas still apply and must be taken into consideration by the developers as they are most responsible for the security of the application. Security practices like a strong password policy, secure authentication and session management and server side validation of data are vital to the security of any information system and will also be covered by the analysis.

Security practices, however, are only one aspect of a healthy system and user privacy must also be considered by the developers. Generally, when we talk about smartphone application privacy, we refer to the necessity of the amount of personal data processed by the system. Mobile devices hold a lot of personal data and the creators of smartphone applications can learn a lot about their users by gathering and processing different data from the devices that are using their applications. This can impose a privacy risk if the developers are not careful about the data they gather or expose. A study [6] on app developers found that privacy is not often the main concern of the developers. The study also described key privacy behaviours that applications should enforce such as limiting data collection to only what is needed by the application. In the analysis we will also take a look at Android permissions, describe Android permission best practices and compare them to the practices implemented by the Tartu Smart Bike Share Android app.

Since the Tartu Smart Bike Share Android application has released newer versions over the course of writing this thesis and as it is impossible to analyze all the updates, the thesis will focus on two versions.

The first app version that will be analyzed is the version 18 that was live from 08.07.2019 to 11.10.2019. This version will be referred to as the old version. The old version of the application was built using Android Studio and can be decompiled back to Java code and further used for static code analysis. This version, however, has since become deprecated and the corresponding server back-end is offline and does not respond to HTTP requests.

The second app version that will be analyzed is the version 81 that was released on 12.02.2020. This version was the latest at the time of writing this thesis and will be referred to as the new version. The new version has the most recent server interaction logic and can be further used for web resource analysis. The new version was not built using Android Studio, but using Flutter [7]. Flutter is a mobile development toolkit developed by Google. Flutter provides native app support and better performance by compiling down to native code. Flutter uses the Dart programming language behind the scenes and decompilers cannot disassemble Dart functions. This means that decompilers cannot restore the original source code and that makes analyzing static data of the application harder, however, not impossible. Both of the versions can be downloaded from APKPure website [8] and analyzed by anyone.

## 3.1 Methods and tools

Popular methods for analyzing the security of Android applications are code analysis and analysis of the network communications. Static-code analysis is a method of analyzing the source code of a program without executing the program itself. By viewing the source code of the Android application we can learn how the application works and communicates behind the scenes and possibly identify security or privacy risks.

Android applications are shipped as Android packages. Android package files or APKs are installer files that the Android OS uses to install an application. Essentially they are nothing more than archive files containing the compiled source and resources of the application [9]. Compiled sources are unreadable for the most part and little information can be interpreted. This is where tools like JADX[6] can help. JADX can decode an Android APK that was built using Android studio to nearly its original development form allowing us to view the source code and resources used when the application was built. Other similar tools exist such as APKTool [10] but were not chosen due to them not being capable of decompiling an apk back to Java code. JADX also provides a powerful GUI to work with that can considerably speed up code analysis. Since JADX is unable to decompile the new version due to Flutter, tools like Ghidra[7] will be used to investigate

[6]JADX. `https://github.com/skylot/jadx`. (21.03.2020)
[7]Ghidra. `https://ghidra-sre.org/`. (26.03.2020)

underlying code. Ghidra is an open-source reverse engineering tool developed by the NSA that we can use to investigate the code and static symbols of Flutter apps.

Static code analysis was used to find the server's web resource URLs and analyze the structure of HTTP requests that were sent to the server. With this data it is was possible to work out the authentication logic and construct our own HTTP requests to these URLs and observe the server's response to them. These web resources, if not properly secured, could potentially accept malicious or unauthorized requests and this could potentially reveal private data or allow unrestricted access to the system.

The old version was used to study the app logic from the decompiled source code, but the new version was used to find out the web resource URLs. Once the web resource URLs were known, Postman[8] was used to construct HTTP requests to specific web resources and observe the HTTP responses returned by the server.

Using JADX we can view the logic of the app itself, different metadata, assets and also all external 3rd party libraries used. JADX's GUI also allows us to navigate and analyze the source code of multiple classes conveniently as Figure 3 illustrates.

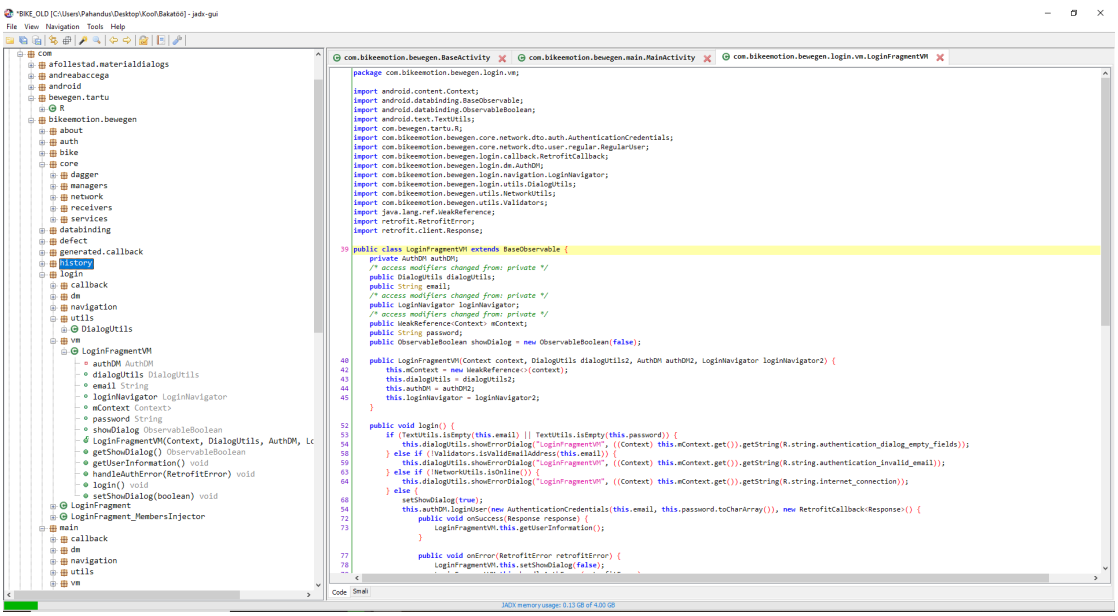


Figure 3. The source code and resources of the old app in JADX

[8]Postman. `https://www.postman.com/`. (21.03.2020)

We can now take a look at how security critical functionality is implemented on the client side and try to identify possible code that might impose a security vulnerability or a privacy risk.

For the new version of the app we can still use JADX to view resources and assets like the `AndroidManifest.xml` which can be useful when analyzing permissions but the source code is no longer viewable with JADX due to the new app being built using Flutter. We can instead use the Ghidra code browser tool to view the disassembled code and static data of the new version as Figure 4 illustrates.

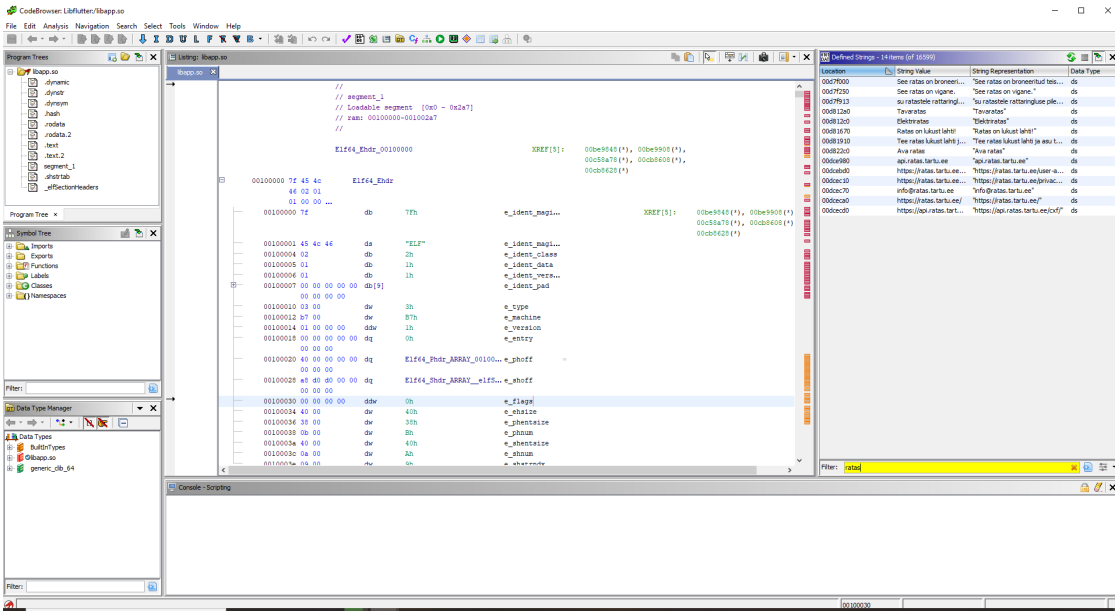


Figure 4. The disassembled code and strings in Ghidra Code browser

An API (Application Programming Interface) can be thought of as an intermediary bridge that defines interactions between the application and the server. To accomplish this task, the server exposes a set of web addresses referred to as resource URLs or endpoints to which the mobile application can send HTTP requests and receive HTTP responses. The most common types of HTTP requests are the GET and POST requests which are used to request and send data respectively [11]. However, only authorized users should be able to make requests to most of these web resources. To check if the web request is authenticated, HTTP request headers usually include some sort of access token that has been issued after the user has authorized with the service.

In order to analyze the implementation and security of the web resources it is necessary to find out the resource URLs where the application is connecting to. We can first analyze the code of the old application to get a general idea of how the API is structured. A simple string search over the source code of the old application reveals the `BikeemotionApi.class` which defines all the resource URLs that the server exposes. Listing 1 illustrates some of the resource URLs found in the `BikeemotionApi.class`.

```
public interface BikeemotionApi {
    @Headers({"Content-Type: application/json"})
    @POST("/cxf/am/station/addFavorite/{id}")
    void addFavoriteStation(@Path("id") UUID uuid, Callback<Response> callback);

    @POST("/cxf/sm/auth")
    void authenticateUser(@Body AuthenticationCredentials authenticationCredentials,
    Callback<Response> callback);

    @POST("/cxf/sm/auth")
    void authenticateUser(@Body RedirectCredentials redirectCredentials, Callback<
    RedirectResponse> callback);

    @GET("/cxf/sm/auth")
    void authenticateUser(Callback<Response> callback);

    @POST("/cxf/um/users/individual/my/")
    void changeIndividualUserInfo(@Body UserInformation userInformation, Callback<
    Response> callback);

    @POST("/cxf/um/users/mobile/my/")
    void changeMobileUserInfo(@Body UserInformation userInformation, Callback<Response>
     callback);

    @POST("/cxf/sm/password/my/")
    void changePassword(@Body Password password, Callback<Response> callback);

    @GET("/cxf/am/cyclecommunications/{id}")
    void checkCycleCommunication(@Path("id") String str, Callback<CycleConnection>
    callback);

    @GET("/cxf/am/cycle/connected/{imei}")
    void checkCycleConnectionState(@Path("imei") String str, Callback<Boolean> callback
    )
```

Listing 1. List of resource URLs in the old app source code

The class defines a list of resource URLs that were used in the old application. It also defines their request type, the request body and the expected return type which is useful when constructing our own test requests. We can note the general structure of how every web resource begins with `/cxf`. Since the addresses have likely changed since then, we can still use this information when analyzing the new version of the application.

By decompiling the newer application using Ghidra, we no longer get a complete and readable source code due to it being built using Flutter. However, using the data we get from the source code of the old app as a guide, we can still narrow down the associated resource URLs. A string search over the code of the new application reveals all the new resource URLs and the main web address used in the new version of the application. Figure 5 illustrates the result of the `cxf` string search using Ghidra.

| Code Unit | String View | String Type |
|---|---|---|
| ds "/cxf/ms/mail/my" | "/cxf/ms/mail/my" | string |
| ds "/cxf/sm/password/reset/start/" | "/cxf/sm/password/reset/start/" | string |
| ds "/cxf/sm/password/my" | "/cxf/sm/password/my" | string |
| ds "/cxf/am/rfidtoken/my/" | "/cxf/am/rfidtoken/my/" | string |
| ds "/cxf/am/cycle/cycleNumber/" | "/cxf/am/cycle/cycleNumber/" | string |
| ds "/cxf/am/cyclecommunications/" | "/cxf/am/cyclecommunications/" | string |
| ds "/cxf/am/cycle/connected/" | "/cxf/am/cycle/connected/" | string |
| ds "/cxf/am/cycle/findcyclestation/" | "/cxf/am/cycle/findcyclestation/" | string |
| ds "/cxf/rm/routes/" | "/cxf/rm/routes/" | string |
| ds "/cxf/um/code/generate" | "/cxf/um/code/generate" | string |
| ds "/cxf/um/code/cancel/" | "/cxf/um/code/cancel/" | string |
| ds "/cxf/am/rfidtoken/my" | "/cxf/am/rfidtoken/my" | string |
| ds "/cxf/rm/routes/my" | "/cxf/rm/routes/my" | string |
| ds "/cxf/am/station/takeFavorite/" | "/cxf/am/station/takeFavorite/" | string |
| ds "/cxf/am/station/addFavorite/" | "/cxf/am/station/addFavorite/" | string |
| ds "/cxf/am/station/map/my" | "/cxf/am/station/map/my" | string |
| ds "/cxf/am/station/map/search" | "/cxf/am/station/map/search" | string |
| ds "/cxf/zm/zone/map/search" | "/cxf/zm/zone/map/search" | string |
| ds "/cxf/um/code/my" | "/cxf/um/code/my" | string |
| ds "/cxf/um/users/myProfile" | "/cxf/um/users/myProfile" | string |
| ds "/cxf/um/country" | "/cxf/um/country" | string |
| ds "/cxf/mm/subscription/plans/" | "/cxf/mm/subscription/plans/" | string |
| ds "/cxf/es/ridango/validate/" | "/cxf/es/ridango/validate/" | string |
| ds "/cxf/mm/subscription/info" | "/cxf/mm/subscription/info" | string |
| ds "/cxf/cm/settings/membership/" | "/cxf/cm/settings/membership/" | string |
| ds "/cxf/um/users/mobile" | "/cxf/um/users/mobile" | string |
| ds "/cxf/cm/settings/search/MOBILE" | "/cxf/cm/settings/search/MOBILE" | string |
| ds "/cxf/billing/creditcard/token/validate/my" | "/cxf/billing/creditcard/token/validate/my" | string |
| ds "/cxf/billing/creditcard/token/preauth" | "/cxf/billing/creditcard/token/preauth" | string |
| ds "/cxf/billing/my/creditcard" | "/cxf/billing/my/creditcard" | string |
| ds "/cxf/am/rfidtoken/" | "/cxf/am/rfidtoken/" | string |
| ds "/cxf/am/rfidtoken/associate/activate/my" | "/cxf/am/rfidtoken/associate/activate/my" | string |
| ds "/cxf/um/users/individual/my" | "/cxf/um/users/individual/my" | string |
| ds "/cxf/um/users/mobile/my" | "/cxf/um/users/mobile/my" | string |
| ds "/cxf/billing/bail/my" | "/cxf/billing/bail/my" | string |
| ds "/cxf/mm/subscription/my" | "/cxf/mm/subscription/my" | string |
| ds "/cxf/sm/auth/" | "/cxf/sm/auth/" | string |
| ds "https://api.ratas.tartu.ee/cxf/" | "https://api.ratas.tartu.ee/cxf/" | string |

Figure 5. List of resource URLs used in the new app as seen in Ghidra

We can observe that some resource URLs have been added and some have been removed but most remain the same. We can also observe the URL in the last line in Figure 5 and note it down as the main API URL upon which all the web resources are built. This URL exposes a list of SOAP services where WSDL[9] is used to describe all the web resources provided by the system. Using this and the old source code as a guide we can construct proper HTTP request bodies and discover expected HTTP responses. The IP address of `https://api.ratas.tartu.ee` is `35.195.62.234` which corresponds to a Google Cloud Platform hosted in USA.

[9]WSDL. `https://www.w3.org/TR/wsdl.html`. (30.04.2020)

## 3.2 Functionality analysis

This chapter focuses on analyzing both the client and server side of the security critical functionality.

### 3.2.1 Account creation

The first step in using the Tartu Smart Bike Share system is to create an account in the app or on the website. In order for the registration functionality to be secure the communication must happen over HTTPS. Registration text fields such as name, password and email must be validated at least on the server side. To confirm the users email account the server must also send an email to the user containing a unique confirmation link. This confirmation link must be randomly generated so that potential attackers could not guess or predict the link.

From the code analysis of the new version we know that the app communicates with the server over HTTPS hence the communication is secure. The email confirmation link is also uniquely generated and cannot be guessed or predicted. An example email confirmation link is shown below:

```
https://ratas.tartu.ee/users/confirm-account/?token=ba452b07-aadc-4b75-97d3-
  abf7f2f9ba30
```

To figure out the client-side requirements we entered and observed the app's response to the attempt of trying to register with invalid data. Figure 6 illustrates error messages that the app produces when trying to register submitting invalid data.

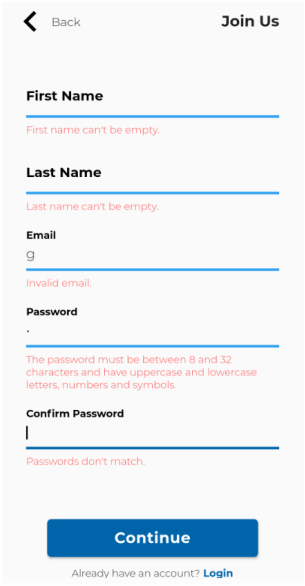



Figure 6. Text field errors when submitting invalid data

We can observe that the app validates all the fields and requires all fields to first have content. Furthermore, the email field must match the email regular expression and the passwords must match and also adhere to the password policy which is described and analyzed in Section 3.2.3. All of these checks are good coding practices and to verify whether these checks are also enforced on the server side, we performed the following experiment.

From the old source code we found that the app sends an HTTP PUT request to the URL `https://api.ratas.tartu.ee/cxf/um/users/mobile` with the name, email, password and date of birth in the request body. Using that as a template we constructed our own registration request bypassing the client-side validation. Listing 2 illustrates an HTTP PUT request and response that has a single character password which does not adhere to the password policy.

```
PUT /cxf/um/users/mobile HTTP/1.1
Content-Type: application/json
Accept: */*
Cache-Control: no-cache
Host: api.ratas.tartu.ee
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

```
Content-Length: 175
{
    "firstName": "Test1",
    "lastName": "Test2",
    "email": "tempmailtest1@svpmail.com",
    "password": [
        "q"
    ],
    "dateOfBirth": "872630000000"
}


HTTP/1.1 200 OK
Server: nginx
Date: Mon, 13 Apr 2020 14:31:03 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
be-tenant: tartu
Strict-Transport-Security: max-age=63072000;
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET,HEAD,OPTIONS,POST,PUT
Access-Control-Allow-Headers: *
Content-Encoding: gzip
"70f0d40a-53c2-4e31-b831-1d61decce0cc"
```

Listing 2. HTTP PUT request to register an account

As we can see the server allowed this registration to proceed and in the HTTP response body returned the user's unique ID. This means that an account with this email was created on the back end, however, we observed that no confirmation email arrives and logging in with the password does not work. Requesting a password change worked and the password had to be changed to a password that complied with the password policy. Once the password was changed the user could log in and bypass the initially required profile and membership creation, when otherwise new users would have had to complete their profile and choose a payment plan before they would be able to proceed to the main view. Viewing any account details or ride history on the website produces a server error, while the app and the API do not return an error, but just an empty response. This is unintended behaviour that may or may not lead to an exploitable vulnerability. The back end should have refused the creation of an account with such a password similarly as the email and name were validated when they were incorrect or empty.

We conclude that the app handles the registration well by verifying all the registrational fields but the server allows the password to ignore the password policy which

leads to some unintended behaviour. The email confirmation link seems to be correctly generated as it cannot be guessed or predicted.

### 3.2.2 Authentication and session management

Secure authentication is at the heart of every information system and can be considered the most critical for the security of the system. The username and password must travel safely across the network and the resulting authentication state must be kept secure.

Tartu Smart Bike system relies on passwords for user authentication. After sending credentials to the server an access token is issued to the user that has to be included in the headers of further HTTP requests as a proof of identity. The access token is usually a unique string of characters following a general standard for representing claims such as the JWT [12] or it could be customly designed by the developers themselves in order to suit their needs. This token must be kept secure by using HTTPS connections and implementing an expiration date for the token so that no access token would be valid forever [13].

In order to inspect how the Tartu Smart Bike app implements its authentication logic we inspected the login functionality of the old app. Listing 3 illustrates the code logic related to login functionality found in the `AuthDM.class` file.

```
public void loginUser(AuthenticationCredentials authenticationCredentials, final
    RetrofitCallback<Response> retrofitCallback) {
        this.preferencesManager.setUsername(authenticationCredentials.getAuthId());
        this.preferencesManager.setPassword(String.valueOf(authenticationCredentials.
    getPassword()));
        this.bikeemotionApi.authenticateUser(authenticationCredentials, (Callback<
    Response>) new Callback<Response>() {
            public void success(Response response, Response response2) {
                for (Header next : response.getHeaders()) {
                    if (next.getName() != null && next.getName().equalsIgnoreCase("be-
    token")) {
   AuthDM.this.preferencesManager.setToken(next.getValue().split(",")[0]);
                    }
                }
                retrofitCallback.onSuccess(response);
            } });
}
```

Listing 3. Login functionality of the old app

We can observe from the code snippet above that the app sends a request to the web service including the username and password of the user in that request. If the

HTTP request response is a success all of the response headers are checked. The name of each response header is compared against the value `be-token` which we can assume to contain the resulting access token upon successful login. If a header with such name exists in the HTTP response we can assume the login was successful. In some cases the response header also contains an additional field `be-user` which represents the user ID of the user to whom the access token was issued for.

We can observe from Listing 1 that the username and password are sent to the `/cxf/sm/auth` web resource as a HTTP POST request and that the body of the request consists of data as described in the `AuthenticationCredentials.class`. The class itself consists of a variable `authId` and character array as can be seen in Listing 4.

```
public class AuthenticationCredentials {
    private String authId;
    private char[] password;

    public AuthenticationCredentials(String str, char[] cArr) {
        this.authId = str;
        this.password = cArr;
    }
}
```

Listing 4. AuthenticationCredentials class defined in the old app

This means that the body of the authentication request consists of these two parameters and we can make a test request to see if we get a valid access token back in the response request headers as discussed above. Listing 5 illustrates an authentication request and response that has the author's credentials as the payload.

```
POST /cxf/sm/auth HTTP/1.1
Content-Type: application/json
Accept: */*
Cache-Control: no-cache
Host: api.ratas.tartu.ee
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 222
{
    "authId": "siim3214@gmail.com",
    "password": [
        "q", "W", "E", "R", "T", "Y", "1", "2", "3", "4", "!"
    ]
}


HTTP/1.1 200 OK
```

```
Server: nginx
Date: Mon, 13 Apr 2020 14:33:26 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
access-control-expose-headers: be-token
be-tenant: tartu
be-token: 6f647a8c-264c-4d21-8c1c-fdf3fb52f021
Strict-Transport-Security: max-age=63072000;
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET,HEAD,OPTIONS,POST,PUT
Access-Control-Allow-Headers: *
Content-Encoding: gzip
{"passwordChangeNeed":false,"email":"siim3214@gmail.com"}
```

Listing 5. Authentication HTTP request

We can see that the response headers returned by the server include a header `be-token` that serves as an access token for further HTTP requests. Including this token in the header of further HTTP requests allows us to make HTTP requests to protected web resources. We can also observe a boolean `passwordChangeNeed` in the response body of the request. This field was used when the password policy was different and is further described in Section 3.2.3

Now that we have the access token we can check if it ever expires. To check if a token will expire we can try to reuse the token after some time. When trying to use this token on the following day the server returned an error saying that the token was invalid. This means that the system invalidates the token after a fixed amount of time. To figure out the expiration period of the token a binary search approach was used. Using this method it was discovered that the expiration period of the token is 12 hours. This means that the system invalidates the token 12 hours after issuing it. The app, however, maintains the authentication state by storing credentials in memory and does not require the user to reauthenticate every 12 hours.

We conclude that since the user credentials and the token are sent over HTTPS they can safely move through the network and thus the authentication logic is secure. The system also uses safe token-based session management where access tokens seem to have high entropy and therefore are not guessable. The access tokens also have a reasonable expiration period.

### 3.2.3 Analysis of password policy

The password policy used in the old version was quite different from the policy that is enforced in the new version. We can use the source code of the old app to see what were the password requirements in the old version. Listing 6 illustrates password change functionality that is present in the old old app's `PasswordEditionVM.class`.

```
public void onSubmit() {
if (StringUtils.isBlank(this.newPassword.get()) || StringUtils.isBlank(this.oldPassword
    .get()) || StringUtils.isBlank(this.confirmPassword.get())) {
           setErrorMessage(this.resources.getString(R.string.
    authentication_dialog_empty_fields));
        } else if (this.newPassword.get().trim().length() < 4) {
            setErrorMessage(this.resources.getString(R.string.password_small));
        } else if (!this.newPassword.get().trim().equals(this.confirmPassword.get().
    trim())) {
            setErrorMessage(this.resources.getString(R.string.passwords_not_equals));
        } else {
            Password password = new Password(this.newPassword.get(), this.oldPassword.
    get());
            this.passwordEditionCallback.onPasswordEditionStarted();
            this.bikeemotionApi.changePassword(password, new Callback<Response>() {
                public void success(Response response, Response response2) {
                    PasswordEditionVM.this.passwordEditionCallback.
    onPasswordEditionCompleted(true);
                    PasswordEditionVM.this.onSuccess();
                }
                public void failure(RetrofitError retrofitError) {
                    PasswordEditionVM.this.passwordEditionCallback.
    onPasswordEditionCompleted(false);
                    PasswordEditionVM.this.onFailure(retrofitError);
                }
            });
        }}
```

Listing 6. Password change code logic in the old app

We can see that the first check of the function validates that all the fields have content, the second check validates whether length of the new password is at least 4 characters and that the third check validates if the new passwords match. According to this logic the only password requirement back then was that the password is of greater length than 4 characters. We also cannot check if the old server enforced any additional checks because it is unavailable. It goes without saying that such a password policy allows the users to choose weak passwords that are prone to brute-force attacks. The new password policy encourages the users to choose a strong password by requiring the length of the password to be at least 8 characters including uppercase letters and at

least one number and one symbol. This was verified experimentally by observing error messages returned by the server for obvious noncompliant passwords. The maximum length of the password is limited to 32 characters which was confirmed experimentally.

We can evaluate the strength of the policy by comparing it to the general digital identity guidelines outlined by NIST [14]. According to NIST any password chosen by the user should be at least 8 characters in length and be allowed a maximum size of 64 characters. Additionally, all ASCII characters, including whitespace, should be supported and login lockout should be forced after 10 incorrect password attempts to protect against brute-force attacks. NIST guidelines also outline that passwords should be salted with a salt of at least 32 bits and hashed using a one-way key derivation function for each user to make the password hashes harder to crack.

The website login form is protected from brute-force attacks by a Google reCaptcha. The authorization web resource is also protected against brute-force attacks as 10 incorrect login requests yielded a temporary error by the server which blocked logins to that user account for approximately a minute.

We can calculate the entropy of the password that complies with the password policy to assess the security level of the password. In the worst case scenario, where a password will have exactly 8 random characters, we can calculate the number of required computations to break the hash as follows:

- The number of lower-case letters in the English alphabet: 26
- The number of upper-case letters in the English alphabet: 26
- The number of valid numeric characters (0-9): 10
- The approximate number of allowed special symbols: 33

$$(27 + 27 + 10 + 33)^8 \approx 2^{52}$$

This means that the security level of the password hash during the worst case scenario is 52 bits. Assuming that passwords are generated randomly the attacker would have to perform at least $2^{52}$ operations to crack the password hash, which may require considerable effort from the attacker. Additionally, if password salting is not used, the attacker will be able to brute-force several password hashes in parallel.

The reason behind a 32 character password limit may be that the database field for the password has been restricted to 32 characters and the passwords are stored in the database as a plain text, which is a bad security practice because the passwords can leak if the database gets compromised. Additionally, the database administrators would be able to view the passwords of the users. If the password would be hashed, it would not matter how big the input password is, because the hash output would be fixed to a set amount of bytes.

We conclude that at some point the Tartu Smart Bike Share system was using a weak password policy. The new password policy, however, adheres to most of the digital identity guidelines listed by NIST [14]. The system is also protected against trivial online and offline brute-force attacks. The maximum password length restriction of 32 characters may suggest that passwords are being kept in the database as plain text.

### 3.2.4 Password change functionality

The app allows the user's to change their passwords once they have logged in. To change the password the user must enter the old password, the new password and confirm the new password. There must be checks in place on both the client side and the server side to ensure that the new password of the user adheres to password policy discussed in Section 3.2.3. If all the client-side checks pass, the app sends a request to the server to change the users password.

Since we know the web resource URL we can construct our own password change request bypassing all the client-side checks. Listing 7 illustrates a constructed HTTP POST request which tries to change the author's account password to a value which does not comply with the password policy.

```
POST /cxf/sm/password/my HTTP/1.1
be-token: 6f647a8c-264c-4d21-8c1c-fdf3fb52f021
Content-Type: application/json
Accept: */*
Cache-Control: no-cache
Host: api.ratas.tartu.ee
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 64
{
    "newPassword": "ggggg",
    "oldPassword": "qWERTY1234!"
}
```

```
HTTP/1.1 400 Bad Request
Server: nginx
Date: Mon, 13 Apr 2020 14:37:06 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
be-tenant: tartu
be-token: 6f647a8c-264c-4d21-8c1c-fdf3fb52f021
be-user: 34c8bed7-2f4f-46ab-90fa-0dd18a93e6af
Strict-Transport-Security: max-age=63072000;
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET,HEAD,OPTIONS,POST,PUT
Access-Control-Allow-Headers: *
{"code":"7082","message":"com.wegoshare.sm.bll.authentication.exception.
    InvalidPasswordException: Password is invalid.","requestId":"03bc15f3-fa9f-4d34-85
    d2-ddc6ac66db30"}
```

Listing 7. Password change request using a noncompliant new password

As Listing 7 illustrates the server rejected our request and this is true for all cases that do not match the password policy described in Section 3.2.3. It was also verified experimentally that the server verifies the `oldPassword` field. If the new password adheres to the password policy, the request will be successful. When trying to change the password to a previously used one the server returns an error stating that the password must be different from the last four passwords. This means that the server is doing an additional check and compares the new password against the last four. This means that the system is storing the last four passwords or the hashes of the last four passwords. It was verified experimentally that the system indeed disallowed exactly the last four passwords. This poses a security risk, because an attacker that has gained access to the database will learn the last four previous passwords of the user, some of which might still be used in other systems. The processing of the last four passwords is not mentioned in the privacy policy (see Section 3.2.8 for the analysis of the privacy policy).

We conclude that the password policy is enforced for the new password not only on the client side, but also on the server side, which is a good coding practice. The server is also retaining the last four passwords of the user which is never mentioned in the privacy policy.

### 3.2.5 Password reset functionality

The system also allows the user to reset their password should the user forget it. In order for this system to be safe it should be implemented in such a way that upon requesting a password reset the user receives a unique one-time use link in their registred email that allows them to reset the password. This link should have a reasonable expiration timer and requesting a new reset link should invalidate the previous one.

To reset the password the user must enter their email address into the app. The app then sends a HTTP PUT request to the resource URL `/cxf/sm/password/reset/start/` + the users email address which is appended to the path of the URL. The server then generates a unique password reset link and sends it to the user's email. The link is indeed unique as observing five different password reset links revealed no obvious repeating patterns:

```
https://ratas.tartu.ee/recover-password/reset/69d0a3c9-b568-4e0c-a32b-7d61548f4467
https://ratas.tartu.ee/recover-password/reset/3e0234e5-ed78-4403-9aff-1a4e505d3555
https://ratas.tartu.ee/recover-password/reset/384af21a-1189-4800-82bc-3e63be8ef96e
https://ratas.tartu.ee/recover-password/reset/e8aa9d28-8e85-4887-b37d-ae331b1593ed
https://ratas.tartu.ee/recover-password/reset/1c039ac1-889b-4bf8-9d68-bf7a4cbeec1c
```

The reset link leads the user to the web application and presents a form for entering the new password. No password reset form is available on the app and it must be reset in the web application. Figure 7 illustrates the password reset from as seen on the website.

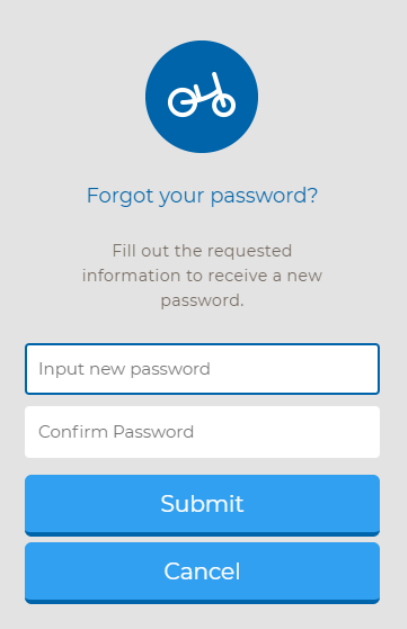



Figure 7. Password reset form on the website

The user can then change their password to a value that adheres to the password policy. After submitting the form, a HTTP POST request is made to the same URL. Listing 8 illustrates the HTTP request and response sent after submitting the password reset form.

```
POST /recover-password/reset/8da97e92-ae03-457f-a3d2-ba06c6abeeab/ HTTP/1.1
Host: ratas.tartu.ee
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-CSRFToken: Pwm3B8qqMMSnwlc0UBjYjWjdSwTfoPJf
X-Requested-With: XMLHttpRequest
Content-Length: 105
Origin: https://ratas.tartu.ee
Connection: keep-alive
Referer: https://ratas.tartu.ee/recover-password/reset/8da97e92-ae03-457f-a3d2-
    ba06c6abeeab/
Cookie: sessionid=xqnjkj9d0jjljyhvl7alikx73n0nqhcm; csrftoken=
    Pwm3B8qqMMSnwlc0UBjYjWjdSwTfoPJf; _ga=GA1.2.1774012266.1587642801; _gid=GA1
    .2.2127043801.1587642801
Payload: {"csrfmiddlewaretoken":"Pwm3B8qqMMSnwlc0UBjYjWjdSwTfoPJf","password":"qWERTY!1
    ","confirmPassword":"qWERTY!1"}


HTTP/2 200 OK
server: nginx
date: Thu, 23 Apr 2020 11:58:31 GMT
```

```
content-type: application/json
vary: Accept-Encoding
vary: Cookie, Accept-Language
content-language: et-ee
set-cookie: csrftoken=XkAgTPCx0eC6iyC8Flb06AV95E0SVmGA; expires=Thu, 22-Apr-2021 11
    :58:31 GMT; Max-Age=31449600; Path=/; secure
set-cookie: sessionid=mck20owgximxc74syqoe1a88xr12b6n8; expires=Thu, 23-Apr-2020 23
    :58:31 GMT; httponly; Max-Age=43200; Path=/; secure
content-encoding: gzip
strict-transport-security: max-age=63072000;
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
content-security-policy: script-src 'self' 'unsafe-inline' 'unsafe-eval' *.googleapis.
    com *.gstatic.com *.github.io *.google-analytics.com *.cloudflare.com *.wpcc.io
    wpcc.io *.jsdelivr.net *.google.com *.jquery.com *.twitter.com *.twimg.com *.
    instagram.com *.facebook.net; object-src 'self'
X-Firefox-Spdy: h2
Response Payload:{"EDITOR_CONFIG":{"text":"{\"url\": \"/users/my-account/\", \"
    redirect_to\": \"url\"}","mode":"application/json"}}
```

Listing 8. Password reset HTTP request and response

After having successfully changed the password the reset link becomes invalid and can no longer be used. Requesting multiple different password reset links reveals that the first one does not actually become invalid after other reset links are requested. In addition, the links are not protected with a reasonable expiration timer as the author was able to use the link 8 days after requesting it. It is not known whether the password reset links ever expire. This is not a good practice because in cases where the user accidentally requests multiple password resets additional working password reset links are generated that will never be used by the user and can possibly leak. Furthermore, in case an attacker requests multiple password resets for a user, since they all are active, the attacker has a bigger chance (while still negligent since a randomly generated token is included in the link) to guess the token for a working link.

We conclude that the password reset functionality could use additional security measures. The password reset links should expire and only one password reset link should be valid at all times for a given user.

### 3.2.6 Web resource authorization checks

Since all the resource URLs seen in Figure 5 are known, we can consider different attack vectors against them. One possible attack vector is that unauthorized users could be allowed to make requests to web resources that require authorization similar to how authorized users make them. To verify that only authorized users can use web resources

that require authorization we can construct valid requests to these resource URLs but exclude the access token. As an example let's consider the bike unlocking functionality. We know that only users who are logged in and have an active membership should be able to unlock bikes. To verify that only authorized users can unlock bikes we can again analyze the source code of the old application to get a general idea of the logic behind how the app is unlocking the bikes. The `BikeUnlockFragment.class` reveals the following code logic related to bike unlocking ( Listing 9).

```
public void onClick(DialogInterface dialogInterface, int i) {
                    if (cycle.getCycleStatus() == CycleStatus.Available) {
                        BikeUnlockFragment.this.bikeemotionApi.putBikeUnlock(cycle.
    getSerialNumber(), new BikeUnlockCallback());
                    } else if (cycle.getCycleStatus() == CycleStatus.IntermediateStop)
    {
                        BikeUnlockFragment.this.bikeemotionApi.postBikeUnlock(cycle.
    getSerialNumber(), new BikeUnlockCallback());
                    }
                }
```

Listing 9. Bike unlock functionality in the old app

This method already assumes that a bike with the entered serial number exists and is also available. The app then sends a request to the server with the associated serial number with the intent of unlocking the bike. The request is sent to the resource URL illustrated by Listing 10.

```
@PUT("/cxf/rm/routes/{cycleNumber}")
void putBikeUnlock(@Path("cycleNumber") String str, Callback<Response> callback);
```

Listing 10. Bike unlock web resource URL

We can observe that the serial number of the bike is appended to the path of the address in this case. Knowing this we can again construct our own request to this resource URL, with a valid bike serial number but an invalid access token and observe the server's response to the request. Listing 11 illustrates the constructed PUT request with a valid and available bike serial but an invalid access token.

```
PUT /cxf/rm/routes/02558 HTTP/1.1
be-token: bf6c7307-940f-47b2-8a2e-acb9846b4e9b
Accept: */*
Cache-Control: no-cache
Host: api.ratas.tartu.ee
```

```
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 0


HTTP/1.1 401 Unauthorized
Server: nginx
Date: Mon, 13 Apr 2020 14:40:01 GMT
Content-Length: 0
Connection: keep-alive
be-tenant: tartu
be-token: bf6c7307-940f-47b2-8a2e-acb9846b4e9b
Strict-Transport-Security: max-age=63072000;
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET,HEAD,OPTIONS,POST,PUT
Access-Control-Allow-Headers: *
```

Listing 11. Bike unlock request using invalid access token

As the figure illustrates the server rejected our request of trying to unlock a valid bike with an invalid access token. This means that the server is indeed checking the identity and validating the token, which is a secure coding practice.

Another possible attack vector is that users without an active membership can use the bike unlock functionality. This would allow registered users, that have not yet paid for the membership, to unlock the bikes. To verify this we can create a new account but leave the profile and payment data unfinished. This way we can acquire an access token but should not be able to unlock a bike. Listing 12 illustrates the constructed PUT request with a valid and available bike serial and a valid access token with no active membership.

```
PUT /cxf/rm/routes/02558 HTTP/1.1
be-token: bf6c7307-940f-47b2-8a2e-acb9846b4e9b
Accept: */*
Cache-Control: no-cache
Host: api.ratas.tartu.ee
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 0


HTTP/1.1 400 Bad Request
Server: nginx
Date: Mon, 13 Apr 2020 14:41:00 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunkedConnection: keep-alive
be-tenant: tartu
be-token: bf6c7307-940f-47b2-8a2e-acb9846b4e9b
be-user: 529bd4da-e4eb-4237-9e72-af957ba21040
Strict-Transport-Security: max-age=63072000;
Access-Control-Allow-Origin: *
```

```
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET,HEAD,OPTIONS,POST,PUT
Access-Control-Allow-Headers: *
{"code":"2001","message":"com.wegoshare.common.exception.IllegalOperationException: The
     user with id [529bd4da-e4eb-4237-9e72-af957ba21040] is not active.","requestId":"
    14fe2a7b-2626-4e14-8f46-fa23248ae468"}
```

Listing 12. Authorized bike unlock request without active membership

As we can observe, the server returned an error stating that the account was not active which is what it should do. This means that only authorized users with an active membership can actually unlock bikes and that the web resource is protected. It seems that the web resources are not vulnerable to at least trivial authentication bypass flaws. All of the resource URLs seen in Figure 5 follow the same standard. No web resource, that requires authorization, was found accessible without an access token.

We conclude that the web resources are not vulnerable to at least trivial authentication bypass flaws and that the server is checking whether the request includes a valid access token or whether the requesting user has an active membership.

### 3.2.7 Bicycle privacy exposure

While verifying access conditions for different web resource URLs that should only be accessible to authorized users, it was discovered that the systems web resource URL `/cxf/am/cycle/cycleNumber/` + bike serial number returns more data about the bike than it should. The data includes, but is not limited to, the bikes real-time location, user, defects, condition, last unlock time and total usage and mileage. Data from this web resource can be requested for any bike and the user making the request does not need to have an active membership.

This web resource was used in the old app to check if the bike, that is being unlocked by the user, is actually available and not in use. This means that in order to check if a bike is available for use the app made a HTTP request to the aforementioned web resource which, in addition to the bike availability, returned a lot more data behind the scenes. To illustrate what data is returned by the web resource, we constructed our own HTTP GET request. Listing 13 displays a HTTP GET request to query info about a bike with the serial number 02794 which was not being used and was available.

```
GET /cxf/am/cycle/cycleNumber/02794 HTTP/1.1
be-token: 606ccb26-01c0-42e1-aa91-94331c58a430
Accept: */*
Cache-Control: no-cache
Host: api.ratas.tartu.ee
Accept-Encoding: gzip, deflate, br
Connection: keep-alive


HTTP/1.1 200 OK
Server: nginx
Date: Mon, 13 Apr 2020 14:42:37 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
be-tenant: tartu
be-token: 606ccb26-01c0-42e1-aa91-94331c58a430
be-user: 34c8bed7-2f4f-46ab-90fa-0dd18a93e6af
Strict-Transport-Security: max-age=63072000;
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET,HEAD,OPTIONS,POST,PUT
Access-Control-Allow-Headers: *
Content-Encoding: gzip
{"id":"0308c3c1-9f21-4a42-a522-6c86c764fcc8","serialNumber":"02794","notes":"09.12.2019
     KK- Kaigud ei toota\n05.07.2019 KK - Keskjooks vajab hooldust\nCWB901171 -
    BWGK52190624","status":true,"code":"VEH02641","cycleType":"Bike","sponsor":"Tartu",
    "advertisement":"Tartu","cycleStatus":"Available","cycleCondition":"Live","
    distanceTraveled":2884.290002949536,"usageTime":2731926819,"location":{"@class":"
    GeoPoint","latitude":58.37737833333333,"longitude":26.730515}
```

Listing 13. Bicycle data GET request

The full response body can be found in the Appendix. We can see that the response includes the location of the bike, defects of the bike and other administrative data. We can already consider this web resource to be too revealing as it is not necessary for regular users to be able to see such administrative data.

We observed that if a bike has been unlocked and is being used by a user the web resource returns additional data. As an example, the response of the bike that at that time was being used by the author can be seen in Listing 17.

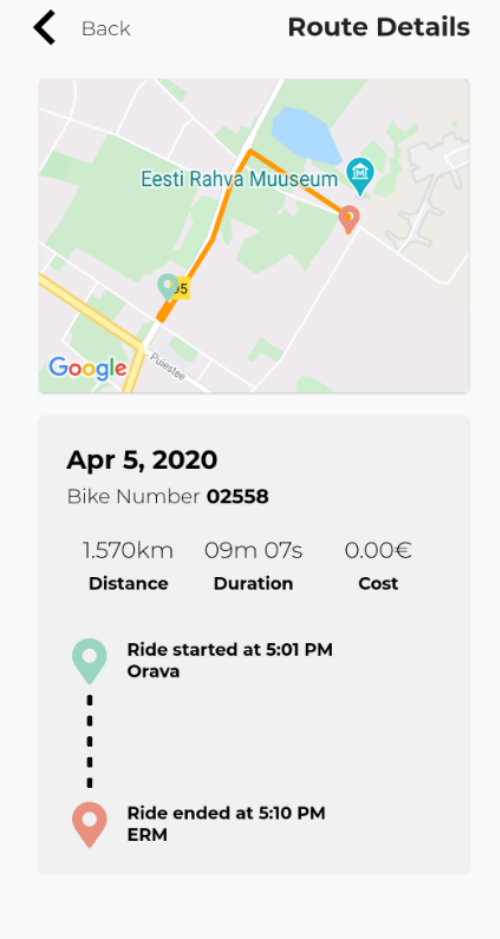
We can observe from the response that the status of the bike is now "InUse" and in addition to all the previous data we also see the `userID` of the user that is currently using the bike and the `rfidCode` field. Removing the first eight numbers from the rfidCode field exposes the user's bus card number if the user has one linked. We verified that the bus card number is exposed even if the bike itself was not unlocked using the bus card.

Even though the user's full name is never revealed, the unique identifier of the user is revealed. There are several ways how the person, to whom the user ID has been

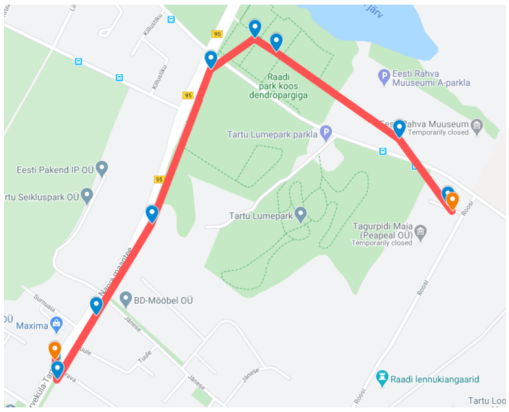
assigned, could be identified. It is possible to just follow the bike to its location and find out the person using the bike. A script can be written to regularly iterate over the data of all the bikes in order to obtain location data of all bikes and their users. We verified the range of serial numbers via a sequential search starting from any valid serial to find the lower and upper bound of valid serials, discovering that valid bike serial numbers are in the range 02043 and 02795. There are special cases such as an invalid serial number 02663 and an additional SparkLAB demo bike with the serial number 02050, which means that there are actually 751 bikes. If data is queried about a serial number that is out of the aforementioned bounds an error is returned by the server stating that the bike does not exist.

According to OWASP [15] this vulnerability falls into the top 10 mobile application vulnerabilities and is a clear breach of privacy as the web resource should never reveal so much data about the bike and its current user to an ordinary member of the system. A similar privacy exposure was found in June 2019 [2] and was likely also in one of the web resources. A better implementation could be that each user can only see this data about themselves or about the bike they are riding and still it would be unnecessary for them to be able to see all the administrative metadata such as defects. Since these requests can be constructed by anyone that has an account in the Tartu Smart Bike System we can essentially consider all the data seen in Listings 16 and 17 public.

To demonstrate how the availability of this information compromises user's privacy the author tracked the bike he was riding. The data about the bike's location was queried throughout the ride and later reconstructed into a full route and then compared against the route that the server generated in the app. Figure 8a and 8b illustrate both the real route as shown by the app and the route that was reconstructed using the data from the web resource.




(a) The route shown by the app




(b) The route reconstructed using 9 queries

Figure 8. The authors bike tracking experiment

Data about the bike was queried every minute up to a total of nine times during the ride and we can see that the precision of the reconstructed route is already alarming. Increasing the number of queries will result in an even greater precision and allow for real-time tracking of users.

We conclude that there is a privacy exposure within the system that allows for any user to track the real-time location and condition of any bike. In addition if the bikes are being used, user specific data such as the unique ID of the user and their Tartu bus card number are also revealed if the users have it linked.

### 3.2.8 Analysis of privacy policy

A good privacy policy is easy to understand and informs the user of how exactly their data is processed and for what purpose. The Tartu Smart Bike Share privacy policy [16] outlines the collection and processing of the following user data:

- First name and surname, age, gender
- Contact details (telephone number, e-mail address and place of residence)
- Payment information (credit card details, personal identification code and data of public transport card or equivalent card)
- Ride information (time and location of renting and returning bike and route data)

According to the privacy policy the profile and contact details will only be used to process claims and provide information and notices related to using the service. The payment information will be processed to enable the user to pay for the service. The personal identification code will be processed if the user wishes to link their Tartu bus card. Ride information will be processed to provide customer support, display route information and identify potential violations of the service agreement.

The privacy policy also notes that the user can update or correct the personal data submitted to the system at any time by logging in and that the cooperation partners may only process data to the extent that is necessary for provision of service. All user data is stored for as long as the user has an account and the user can request the deletion of their account. User data is not deleted in cases where obligation to retain the data is provided by law.

By our observations the app is not uploading any data that is not listed above. It is clear as to why this data is being collected and for what purpose it is being processed. The privacy policy does not, however, note the fact that the system stores the last four passwords of the user. It also fails to mention how the passwords are being protected in the case of a database leak. Additionally, the date and version of the privacy policy are missing, while being present in the Tartu Bike Share Terms of Use[10] where privacy policy is referenced.

[10]Tartu Bike Share Terms of Use. `https://ratas.tartu.ee/user-agreement/(30.04.2020)`

We conclude that the Tartu Smart Bike Share system's privacy policy does not outline all the data that is being collected and processed. The policy should include additional entries on how the current and past passwords are retained.

### 3.2.9 Analysis of requested Android permissions

The Android operating system controls the flow of information from the system to the application via Android permissions [17]. The sole purpose of Android permissions is to protect the user's privacy. The permissions are divided into different protection layers. Normal permissions, such as accessing the network state, imply a low privacy risk and are granted automatically by the Android operating system [17]. Dangerous permissions, such as the ability to read the users contacts or send an SMS, are considered a high privacy risk due to the nature of the information that they disclose to the app and therefore require the user to approve the permission instead of it being granted by default [17]. Android applications should only require permissions absolutely critical for the functionality of the application and only use them for their intended purposes. Requiring permissions for functionalities that an application does not even use or are highly unnecessary is a bad practice and puts the users privacy at risk.

The Android operating system allows its users to view permissions required by any application in the Android system settings. However, the names of these permissions are very abstract. For example, the device shows that the app uses the *LOCATION* permission but does not specify which ones since there multiple variations and in order to see the exact permissions being required it is necessary to disassemble the app. By design each Android app must have a manifest file called `AndroidManifest.xml` that describes the required permissions and components of the app to various elements of the operating system [18]. Most importantly, the manifest describes permissions the app needs from the device in order to serve its functionality.

By disassembling the Android app we can take a look at the contents of its manifest file and figure out the exact Android permissions that are being used by the app and then decide whether or not they are really necessary for the functionality of the app. We expect the Tartu Smart Bike Share Android app to only require permissions that are critical for functionality and not require any dangerous permissions that could be

considered unnecessary for the functionality of the app and possibly impose a privacy risk.
After disassembling the old version of the app it was discovered that the app required exactly seven Android permissions:

1. INTERNET
2. ACCESS NETWORK STATE
3. WRITE EXTERNAL STORAGE
4. ACCESS COARSE LOCATION
5. ACCESS FINE LOCATION
6. VIBRATE
7. CALL PHONE

It is obvious why the app might need permissions such as *INTERNET* and *ACCESS LOCATION* as they can be considered critical due to the nature of the app revolving around navigation. The difference between *ACCESS COARSE LOCATION* and *ACCESS FINE LOCATION* is the precision of the location. Fine location returns precise GPS data while coarse location returns approximate network based location data [19]. The rest of the permissions, however, raise a question as to why exactly are they needed for the functionality of the app.

According to the Android permissions manifest [**?**] two of these permissions, *WRITE EXTERNAL STORAGE* and *CALL PHONE* are considered dangerous while *VIBRATE* has a normal protection level. In addition the manifest outlines that the *CALL PHONE* permission allows the app to make a phone call without ever having the user confirm it. *WRITE EXTERNAL STORAGE* permission allows the app to write data to the external storage of the phone which contains media files such as images, videos and audio files [21]. Since the *VIBRATE* permission has normal protection level and does not impose a privacy risk even when unnecessarily used, a closer look as to where and why the permission is used, will not be taken. The necessity of the *WRITE EXTERNAL STORAGE* permission is also hard to evaluate due to any file activity potentially needing it and will not be discussed further.

To find out what for the permission *CALL PHONE* was used, we took a look at the source code of the app. Listing 14 illustrates the code logic found in `MainActivity.class` that is associated with dialing customer support through the app.

```
private void onNeedHelp() {
   try {
        Intent intent = new Intent("android.intent.action.DIAL");
        intent.setData(Uri.parse("tel:+372 5305 5000"));
        startActivity(intent);
    } catch (ActivityNotFoundException e) {
        Log.e("MainActivity", "onNeedHelp: " + e.getLocalizedMessage(), e);
    } catch (IllegalStateException e2) {
        Log.e("MainActivity", "onNeedHelp: " + e2.getLocalizedMessage(), e2);
    }
}
```

Listing 14. Customer support dial logic in the old app

We can observe from the code snippet that the app constructs an Android intent. Android intents [22] are messaging objects that are used to request an action from another app on the device. The intent *action.DIAL* is used here to forward the customer support phone number to the device's native Phone app so that the users could make the call themselves. The intent *action.DIAL* requires no permissions from the device according to [23] and therefore using it is a good practice.

According to Google's Android application security best practices [24] applications should indeed delegate such tasks to other applications that already have the necessary permission thus avoiding the need to require dangerous permissions from the user. Except that the old app still required the *CALL PHONE* permission for no apparent reason.

The permission *CALL PHONE* allows the app to use the intent *action.CALL* which in turn allows the app to make a phone call without the user's consent [23] but there was no trace of the intent being used in the app's source code and thus the permission was never actually used.

We conclude that the app never even uses the potential of the CALL PHONE permission making the permission entirely unnecessary and imposing an unnecessary privacy risk for the users by allowing the app to potentially make a phone call without the consent of the user.

After disassembling the new version of the app it was discovered that these permissions in question have indeed since been removed. The app now requires only four permissions:

1. INTERNET
2. ACCESS NETWORK STATE
3. ACCESS COARSE LOCATION
4. ACCESS FINE LOCATION

All of these permissions can be considered critical for the functionality of the system and in the latest update no unnecessary permissions are required. This also proves that the app can serve it's functionality just fine without requiring the permissions used in the old version.

# 4 Summary of findings

Below is a list of security and privacy issues found, ranked by significance.

1. A privacy exposure is present in the the system that allows any user to track the real-time location and condition of any bike. Additionally if the bikes are being used, the user's unique ID and bus card number are also exposed if the users have it linked (Section 3.2.7).

2. The system used to enforce a weak password policy. The current password policy is strong and adheres to most of the digital identity guidelines outlined by NIST [14], however, the maximum password length is restricted to 32 characters which may suggest that passwords are being kept in the database as plain text (Section 3.2.3).

3. The password reset functionality allows for more than one active password reset link and generating a new one does not invalidate the previous links. The password reset links also do not expire which greatly increases the chance of a working password reset link making it to the hands of an attacker (Section 3.2.5).

4. The reuse of the last four passwords is forbidden meaning that the server is also retaining the last four passwords which could prove to be a security risk should the database ever leak as some passwords could still potentially be used in other systems (Section 3.2.4).

5. By disabling client-side checks, it is possible to register an account with a password that does comply with the password policy and this leads to some unintended behaviour (Section 3.2.1).

6. The old version of the app was requesting unnecessary Android permissions (Section 3.2.9) which imposed a privacy risk for the users.

7. The privacy policy of Tartu Smart Bike Share system does not mention password among the personal data it processes and is missing the information about the previous passwords being retained by the system. (Section 3.2.8).

# Conclusion

The objectives set in the introduction were achieved. The functionality of the Tartu Smart Bike Share system was described and security critical functionality of the app and its communication with the web service were analyzed. Several security and privacy aspects were discovered that need to be taken into account when using the Tartu Smart Bike Share system.

Most of the security critical functionality is properly implemented and safe from trivial attacks. The app enforces a strong password policy and all web resources that should require authentication do require authentication. Access tokens and special links are unique and seem to contain enough entropy to be safe from brute-force guessing attacks.

However, several security and privacy issues were still found in different functionalities. One web resource is revealing too much data about the bikes and exposes the user and the real-time location of the bike to all users of the system. The password reset links do not expire and generating new ones does not invalidate the previous ones. In some cases registering can lead to unintended behaviour and the privacy policy could be improved by describing how the system retains user's passwords.

The use of a weak password policy and the use of unnecessary Android permissions in the old version of the app have since been fixed. The fixes for the rest of the findings outlined in the thesis should not be too difficult to implement and should not impact the uptime or performance of the service.

## Disclosure and vendor response

On `04.05.2020`, we contacted Tartu City Transport sharing the thesis draft and the findings. On `05.05.2020`, we received a response that Bewegen has been notified and necessary steps are taken to evaluate the breach. On `06.05.2020`, we received an update that the privacy issue has been fixed in the system. In addition, we received information that the passwords are actually stored in the database as hashes and there are plans to extend the maximum password length from 32 to 64 characters. We also received a confirmation that the shortcomings of the privacy policy will be addressed and the risks related to passwords that are highlighted in the points 2 to 5 in the Summary of Findings

are taken into account and the developer will be requested to make corresponding improvements to the system.

Listing 15 illustrates the HTTP GET request and response after the fault has been fixed. As can be seen in Listing 15 the response no longer includes unnecessary additional data. The web resource now returns the bikes `cycleStatus` without exposing the bikes location, user ID or the users bus card. We verified that the value of `currentUserId` is always set to zeros, even if the bike is being used.

```
GET /cxf/am/cycle/cycleNumber/02621 HTTP/1.1
be-token: 99fb08e2-e87f-42ae-ab39-b98a46146c4a
Accept: */*
Cache-Control: no-cache
Host: api.ratas.tartu.ee
Accept-Encoding: gzip, deflate, br
Connection: keep-alive



HTTP/1.1 200 OK
Server: nginx
Date: Wed, 06 May 2020 11:33:50 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
be-tenant: tartu
be-token: 99fb08e2-e87f-42ae-ab39-b98a46146c4a
be-user: 34c8bed7-2f4f-46ab-90fa-0dd18a93e6af
Strict-Transport-Security: max-age=63072000;
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET,HEAD,OPTIONS,POST,PUT
Access-Control-Allow-Headers: *
Content-Encoding: gzip
{
   "id":"be2f7382-d4a6-4c6f-b511-cdfd31dc6430",
   "serialNumber":"02621",
   "cycleStatus":"Available",
   "cycleCondition":"Live",
   "currentUserId":"00000000-0000-0000-0000-000000000000",
   "cycleCommunicationsList":[
      {
         "id":"a967d827-d0c1-46bc-9d2d-cdda1623af5b",
         "delete":false
      }
   ]
}
}
```

Listing 15. Bicycle data GET request after the fix

# References


[1] Tartu Bike share website. Tartu City Government. https://www.tartu.ee/en/bikeshare (12.01.2020)

[2] ERR News. Olerexi, Tartu rattaringluse ja kirjatarvete e-poe klientandmed olid avalikult ripakil (in Estonian). Client data of Olerex, Tartu Bike Share and a stationery web store were publicly disclosed, 2019. https://www.err.ee/960809/olerexi-tartu-rattaringluse-ja-kirjatarvete-e-poe-kliendiandmed-olid-avalikult-ripakil (12.01.2020)

[3] Tartu News. Tartu rattaringluse infosüsteemis esinenud turvanõrkus on kõrvaldatud (in Estonian). The security vulnerability residing in the Tartu Smart Bike system has been fixed. Tartu City Government, 2019. https://tartu.ee/et/uudised/tartu-rattaringluse-infosusteemis-esinenud-turvanorkus-on-korvaldatud (06.05.2020)

[4] Mobile operating system market share in Estonia. GS Statcounter. https://gs.statcounter.com/os-market-share/mobile/estonia (30.04.2020)

[5] Meeksa, Roman. Presentation in Data Science Seminar: Tartu Smart Bike Share – how and what do we see, University of Tartu, 2019. https://www.uttv.ee/naita?id=28980 (12.01.2020)

[6] Balebako, R & Cranor, L. Improving App Privacy: Nudging App Developers to Protect User Privacy. Security & Privacy, IEEE, 2014, 55-58.

[7] Flutter. Google. https://flutter.dev/docs/resources/faq (26.03.2020)

[8] APKPure. https://apkpure.com/tartu-smart-bike/com.bewegen.tartu. (21.03.2020)

[9] Application Fundamentals. Google Developers. https://developer.android.com/guide/components/fundamentals (24.03.2020)

[10] APKTool. A tool for reverse engineering Android apk files. https://ibotpeaches.github.io/Apktool/ (03.03.2020)

[11] HTTP request methods. Mozilla. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (26.03.2020)

[12] JSON Web Token. https://jwt.io/ (26.03.2020)

[13] JSON Web Token Best Practices. Auth0. https://auth0.com/learn/token-based-authentication-made-easy/ (26.03.2020)

[14] Digital Identity Guidelines. NIST, 2017. https://pages.nist.gov/800-63-3/sp800-63b.html#sec5 (04.15.2020)

[15] OWASP Mobile Top 10. OWASP Foundation. https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage (04.14.2020)

[16] Tartu Smart Bike Share privacy policy. Tartu City Government. https://ratas.tartu.ee/privacy-policy/ (20.04.2020)

[17] Permissions Overview. Google. https://developer.android.com/guide/topics/permissions/overview#normal_permissions (29.02.2019)

[18] App Manifest Overview. Google. https://developer.android.com/guide/topics/manifest/manifest-intro (29.02.2019)

[19] Common Android Intents. Google. https://developer.android.com/guide/components/intents-common (21.03.2020)

[20] Android permission manifest. Google. https://developer.android.com/reference/android/Manifest.permission (03.03.2020)

[21] Data and file storage overview. Google. https://developer.android.com/training/data-storage (03.03.2020)

[22] Intents and filters. Google. https://developer.android.com/guide/components/intents-filters (05.03.2020)

[23] Common Android Intents. Google. https://developer.android.com/guide/components/intents-common (21.03.2020)

[24] App security best practices. Google. https://developer.android.com/topic/security/best-practices (05.03.2020)

# Appendix

## I. Server responses for bike data queries

Listing 16 shows a HTTP GET request and response when querying data about a bike that at the time of the query was not used by any user. Listing 17 shows a HTTP GET request and response when querying data about a bike that the author was riding. Both queries were made to analyze that data returned by the `/cxf/am/cycle/cycleNumber/` web resource and are further described in Section 3.2.7.

```
GET /cxf/am/cycle/cycleNumber/02794 HTTP/1.1
be-token: 0b7fb04a-969c-4b53-b186-350c2bcb15d5
Accept: */*
Cache-Control: no-cache
Host: api.ratas.tartu.ee
Accept-Encoding: gzip, deflate, br
Connection: keep-alive


HTTP/1.1 200 OK
Server: nginx
Date: Sat, 13 Apr 2020 14:42:37 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
be-tenant: tartu
be-token: 0b7fb04a-969c-4b53-b186-350c2bcb15d5
be-user: 34c8bed7-2f4f-46ab-90fa-0dd18a93e6af
Strict-Transport-Security: max-age=63072000;
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET,HEAD,OPTIONS,POST,PUT
Access-Control-Allow-Headers: *
Content-Encoding: gzip

{
   "id":"0308c3c1-9f21-4a42-a522-6c86c764fcc8",
   "serialNumber":"02794",
  "notes":"09.12.2019 KK- Kaigud ei toota\n05.07.2019 KK - Keskjooks vajab hooldust\
    nCWB901171 - BWGK52190624",
   "status":true,
   "code":"VEH02641",
   "cycleType":"Bike",
   "sponsor":"Tartu",
   "advertisement":"Tartu",
   "cycleStatus":"Available",
   "cycleCondition":"Live",
   "distanceTraveled":2803.450002754107,
   "usageTime":2681899478,
   "location":{
      "@class":"GeoPoint",
      "latitude":58.371766666666666,
```

```
      "longitude":26.763915
   },
   "lockType":"Primary",
   "stationName":"Kaunase puiestee",
   "stationId":"3d853caf-5b61-4c30-b395-b72ddca5df3d",
   "batteryLevel":89.0,
   "unlockAt":1586342469947,
   "lockAt":1586343680000,
   "condition":"25711e33-1b7c-4273-1111-73729b141051",
   "cycleHardwareList":[

   ],
   "rfidReaderList":[

   ],
   "cycleBatteryList":[
      {
         "id":"d6fc0e47-2ffe-4f47-9a28-1518f4f6014d",
         "designation":"com.wegoshare.am.bll.impl.battery.cycle.CycleBatteryEntity",
         "delete":false,
         "serialNumber":"2500119-00032"
      }
   ],
   "electricMotorList":[

   ],
   "motorControllerPCBList":[

   ],
   "boardPCBList":[
      {
         "id":"ca61c273-ecd8-47ac-8d1e-88ecf6b1da1a",
         "designation":"com.wegoshare.am.bll.impl.boardpcb.BoardPCBEntity",
         "delete":false,
         "serialNumber":"BWGN52190732"
      }
   ],
   "cycleCommunicationsList":[
      {
         "id":"515a398b-506a-419b-a76f-9c54ba82baff",
         "designation":"com.wegoshare.am.bll.impl.communications.cyclecommunications.
    CycleCommunicationsEntity",
         "delete":false,
         "serialNumber":"BWGS52196349"
      }
   ],
   "displayList":[

   ],
   "motorType":"RS485",
   "motor":"0",
   "frame":"BW011997",
   "locationUpdatedAt":1586364829420
}
```


Listing 16. Server response for a bike that is available

```
GET /cxf/am/cycle/cycleNumber/02794 HTTP/1.1
be-token: 0b7fb04a-969c-4b53-b186-350c2bcb15d5
Accept: */*
Cache-Control: no-cache
Host: api.ratas.tartu.ee
Accept-Encoding: gzip, deflate, br
Connection: keep-alive


HTTP/1.1 200 OK
Server: nginx
Date: Sat, 13 Apr 2020 14:45:33 GMT
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
be-tenant: tartu
be-token: 0b7fb04a-969c-4b53-b186-350c2bcb15d5
be-user: 34c8bed7-2f4f-46ab-90fa-0dd18a93e6af
Strict-Transport-Security: max-age=63072000;
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET,HEAD,OPTIONS,POST,PUT
Access-Control-Allow-Headers: *
Content-Encoding: gzip
{
  "id": "dd973116-6a13-46cf-97a9-e47ee4d49508",
  "serialNumber": "02558",
  "notes": "28.03.2020 KK- vasak pedaal hakkab ara lagunema \nCWB901171 - BWGK52190712"
    ,
  "status": true,
  "code": "VEH02243",
  "cycleType": "Bike",
  "sponsor": "Tartu",
  "advertisement": "Tartu",
  "cycleStatus": "InUse",
  "cycleCondition": "Live",
  "distanceTraveled": 2487.3699965029955,
  "usageTime": 2900963507,
  "location": {
    "@class": "GeoPoint",
    "latitude": 58.391423333333336,
    "longitude": 26.73027666666667
  },
  "currentUserId": "34c8bed7-2f4f-46ab-90fa-0dd18a93e6af",
  "batteryLevel": 97,
  "rfidCode": "3086490099501414959",
  "unlockAt": 1586095278809,
  "lockAt": 1586077736929,
  "condition": "25711e33-1b7c-4273-1111-73729b141051",
  "cycleHardwareList": [],
  "rfidReaderList": [],
  "cycleBatteryList": [
    {
      "id": "7d18d92f-d7ce-4e9b-b360-58fba372aee5",
      "designation": "com.wegoshare.am.bll.impl.battery.cycle.CycleBatteryEntity",
      "delete": false,
      "serialNumber": "2500119-00128"
    }
```

```
  ],
  "electricMotorList": [],
  "motorControllerPCBList": [],
  "boardPCBList": [
    {
      "id": "2a48acf4-7e8f-4732-ad9f-caf917cbf6cd",
      "designation": "com.wegoshare.am.bll.impl.boardpcb.BoardPCBEntity",
      "delete": false,
      "serialNumber": "BWGN52190564"
    }
  ],
  "cycleCommunicationsList": [
    {
      "id": "62acdd16-76fe-4a62-8294-c6956e1def24",
      "designation": "com.wegoshare.am.bll.impl.communications.cyclecommunications.
    CycleCommunicationsEntity",
      "delete": false,
      "serialNumber": "BWGS52196187"
    }
  ],
  "displayList": [],
  "motorType": "RS485",
  "motor": "0",
  "frame": "BW012238",
  "locationUpdatedAt": 1586095280014
}
```


Listing 17. Server response for a bike that is being used

# II. Licence

## Non-exclusive licence to reproduce thesis and make thesis public

I, **Siim-Alexander Kütt**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   **Security Analysis of Tartu Smart Bike Share Android Application**,

   supervised by Arnis Paršovs.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Siim-Alexander Kütt
***08.05.2020***