Bachelor Thesis

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Security analysis of UniNow app

Keven Zimmermann
Department of Computer Science
Chair of Computer Science II (Secure Software Systems)

Prof. Dr.-Ing. Alexandra Dmitrienko
Reviewer

B.Sc. Filipp Roos
Advisor

Submission
29. June 2021

Abstract

We performed a security analysis of the *UniNow* application, focusing on private user data. The application was chosen for its high popularity and amount of sensitive information handled through its new contact tracing functionality. This functionality is used by many universities for mandatory contact tracing against COVID-19. Our analysis concerned the Android mobile application and all web services. After mapping the attack surface using many tools to automate the process, we tested the potential issues in order of impact.

We reveal multiple security issues with low to critical severity. Most of these vulnerabilities threatened private user data directly. In particular, we found multiple disclosed secret keys that were used to encrypt user data on the client side, among others. Furthermore, we show how an in-app browser as well as an open redirect vulnerability were easing phishing attacks, and how university access tokens were shared with *UniNow*. Our more severe findings include how sensitive data was sent automatically to Google servers through backup files, how strict SSL certificate checking was disabled deliberately for many university endpoints, how JavaScript could be injected into the email viewer, and how appointment invitations could be brute forced to get sensitive information about past and future appointments. The two critical security issues concern how user accounts could be taken over by using an account identifier, and how an internal service was accessible by anyone, allowing to change university configurations and potentially make user devices sent their user's university credentials to the attacker.

We performed a responsible disclosure, which serves as a reference to *UniNow*, enabling them to fix the discovered issues. Our work tangibly improves the security of student data handled by the company.

Zusammenfassung

Wir haben eine Sicherheitsanalyse der *UniNow* app durchgeführt, wobei wir uns auf private Nutzerdaten fokussierten. Wir haben diese Applikation aufgrund ihrer hohen Popularität und verarbeiteten Datenmenge, die durch die neue Kontaktverfolgungsfunktion erhoben wird, gewählt. Diese Funktion wird von vielen Universitäten für eine verpflichtende Kontaktverfolgung gegen eine Ausbreitung von COVID-19 eingesetzt. Unsere Analyse hat sich mit der mobilen Android Applikation und allen Internetdiensten auseinandergesetzt. Nachdem wir die Angriffsfläche mit Hilfe vieler automatischer Programme erkundet haben, haben wir die möglichen Sicherheitsprobleme der Reihe nach ihrem potenziellen Schaden getestet.

Wir enthüllen mehrere Sicherheitsprobleme mit einer niedrigen bis kritischen Auswirkung. Die meisten dieser Schwachstellen bedrohten private Nutzerdaten direkt. Genauer gesagt, haben wir mehrere veröffentlichte geheime Schlüssel gefunden, welche unter Anderem für die Verschlüsselung von Nutzerdaten auf der Clientseite genutzt wurden. Außerdem zeigen wir wie ein app-interner Webbrowser und eine Open-Redirect Schwachstelle jeweils Phishingattacken erleichtert haben, und wie Zugangsdaten für Universitätskonten an *UniNow* gesendet werden. Unsere ernsteren Befunde beinhalten wie sensitive Daten automatisch an Google Server gesendet wurden, wie die strikte Prüfung von SSL Zertifikaten für viele Universitätsdienste absichtlich ausgeschaltet wurde, wie JavaScript in die Email-Ansicht injiziert werden konnte, und wie Termineinladungen erraten werden konnten, um sensitive Information über vergangene und zukünftige Termine zu erlangen. Die zwei kritischen Sicherheitsprobleme sind der Fakt, dass man fremde Nutzerkonten mit einer Kontokennung übernehmen konnte, und das jeder Zugang zu einem internen Dienst hatte, was einem Angreifer theoretisch erlaubt hat Universitätskonfigurationen zu verändern und potenziell die Zugangsdaten für Universitätsdienste von anderen Nutzern zu stehlen.

Wir haben eine Responsible Disclosure durchgeführt, welche von *UniNow* genutzt werden kann, um die gefundenen Probleme zu beheben. Unsere Arbeit verbessert wahrnehmbar die Sicherheit der, vom Unternehmen verarbeiteten, Studentendaten.

Contents

1. Introduction

The COVID-19 pandemic sparked a lot of development on innovative countermeasures. An important way to slow down the spread of an infectious disease is contact tracing [1]. It is used to reconstruct chains of infection, allowing the quarantining of people who have a high chance of being infected before they can infect others. This process is currently part of people's everyday life. Whether going to a restaurant or a class in university, people are often required to fill out a form with their contact information or register themselves in an attendance list.

Contact tracing had to be digitalized, in order to deal with the high amount of infections during the pandemic. Therefore, different kinds of applications emerged. There are mobile applications, like the German *Corona-Warn-App* [2], which use Bluetooth technology to communicate with nearby smartphones to calculate the distance between them and the time of exposure. Most of these applications notify users if they had contact with somebody who was later diagnosed with COVID-19. Other applications are online attendance lists, used by businesses like restaurants or swimming pools to trace their customers. These attendance lists save contact information on a time and date basis, allowing government agencies to make conclusions on potential contacts and to inform the participants.

The pressure for solutions came with an increased development speed, which often reduced security and, combined with private data handling, jeopardized the privacy of users. This risk of exposing private data can be reduced by security audits. Multiple security audits of contact tracing applications have revealed unknown vulnerabilities [3, 4, 5, 6, 7, 8].

One application that added a contact tracing feature is *UniNow*. The app provides many features to manage university related tasks. For example, users can send and receive their university emails or access their grades. *UniNow* enjoys widespread use and allows universities to manage online attendance lists to trace their students. To make sure that the mandatory usage of the application is secure and to contribute to the effort of securing private data, we performed a security analysis of the *UniNow* mobile application with a focus on the protection of user data.

The security analysis was divided into three parts. First, we tried to find as much attack surface as possible, while the scope of the attack surface was already defined beforehand in our planned approach. Then a vulnerability testing order was determined. Lastly, the vulnerabilities were tested for, documenting the results. This process ensured that as many vulnerabilities as possible were tested for, starting with the high priority vulnerabilities. The analysis was guided by multiple OWASP resources, like the OWASP Mobile Security

Testing Guide (OWASP MSTG) [9] and the OWASP Web Security Testing Guide (OWASP WSTG) [10].

Our work checks the security claims made by *UniNow* and points out any security issues that we found. In other words, we try to improve the security of the user data handled by the application.

Subsequent to this introduction, the thesis continues with Chapter 2, containing needed background information. In particular, information about the target of the security analysis, the used resources, and other concepts. Afterwards, we explain how the thesis fits into the existing work done on related topics in Chapter 3.

After the first chapters have talked about work by others, we start with our own work in Chapter 4, explaining how the security analysis was approached. Specifically, how the scope of the security analysis was defined, how we mapped out the attack surface, how we approached the construction of the testing order, and how the testing was performed.

In Chapter 5, we explain how and what attack surface was found, separating the attack surface into the assets found through functionalities in the mobile application and web services. Continuing in Chapter 6, we list and explain the found security and privacy issues. Other tested attack vectors, which were tested and found to not exist, are listed in Chapter 7.

Following the explanation of the results of the security analysis, Chapter 8 is about how we disclosed the found issues to *UniNow* and what their point of view on these issues was. Lastly, Chapter 9 concludes our work, giving suggestions on future work on the same application and on others.

2. Background

In this chapter, we are going to provide some background information needed for the thesis.

2.1. UniNow Mobile Application

UniNow is a mobile application available on Android and iOS smartphones. It is being developed by the *UniNow GmbH* residing in Magdeburg, Germany. With a download count of more than 100,000 on the Google Play Store [11] and the top position in the "Apps for University" category of the iOS App Store [12], *UniNow* is mainly used by university students. Also, more than 810 universities use it, generating over 3.5 million screen views per month [13]. The application advertises features for students, student councils, universities and companies [13]. We are going to describe these features in the following few paragraphs.

Students can join their university, if it has signed up for *UniNow*, to be able to import their university calendar, check their grades, retrieve and send university emails, get the schedule of their university cafeteria, renew borrowed books from the university library, discover companies and jobs, and get access to the news feed of their university.

Universities, on the other hand, have to sign up for *UniNow* before they can manage and use electronic ID cards, provide e-learning resources for their students, provide access to their radio station, manage user roles and privileges, provide information about campus facilities to users, use the news feed as a communication channel, and provide a list of important website URLs. Furthermore, they can use the attendance list feature. Universities can create QR codes and put them in front of lecture halls or other rooms and areas. Students should then scan the QR code using the *UniNow* mobile application before physically entering the corresponding area. After scanning a QR code, the university may also ask the student to fill out an additional form in the application. The collected data is then encrypted and stored on a remote server in Germany. Only the corresponding university has a key to decrypt the data. The stored data gets deleted after four weeks.

Universities can create profiles for student councils. Students can then follow these profiles to get notified about posts made by them.

Companies can create profiles on *UniNow*, which work like the student council profiles. Additionally, they can publish open job positions with individual descriptions. They can also run advertisements for themselves and their job positions. Lastly, companies and their published positions can be filtered and searched for by the students.

2.2. Open Web Application Security Project (OWASP)

The Open Web Application Security Project is a nonprofit organization, which publicizes different resources on software security [14]. Relevant to our work are the resources it keeps up to date on web application security and mobile application security. These were used heavily to guide the security analysis.

The OWASP Mobile Top Ten [15] and the OWASP Top Ten [16] rank the most critical security risks to mobile applications and web applications, respectively. The ranking is based on a broad consensus that is determined by collecting data from a variety of sources. These include global surveys, security vendors, consultancies, bug bounties, and company contributions. Both lists are updated every few years. At the time of writing, the latest version of the OWASP Mobile Top Ten is from 2016, while the latest version of the OWASP Top Ten is from 2017. We used both rankings to determine the testing order of the potential vulnerabilities.

The OWASP Mobile Security Testing Guide (OWASP MSTG) [9] and the OWASP Web Security Testing Guide (OWASP WSTG) [10] are comprehensive testing guides for mobile applications and web applications respectively. They contain mostly descriptions of various vulnerabilities and security requirements, as well as step-by-step guides on how to test for them. As we started with little knowledge about the technical aspects of *UniNow*, the testing guides helped us by saving us time on many occasions. For example, the OWASP MSTG has many techniques on how to bypass certificate pinning for native, Java and hybrid applications, which saved us a lot of time spent researching. The same applies for testing vulnerabilities. Not knowing the technical aspects of the application also meant that we did not know what potential vulnerabilities there might exist. If we would have found that cryptography libraries are used, for example, then the chapters on cryptography would have been very helpful in checking if they are used correctly. So, in summary, having one source for all of our initial questions saved us a lot of time.

The OWASP MSTG is accompanied by a project called OWASP Mobile App Security Requirements and Verification Standard (OWASP MASVS) [9], which is a standard for mobile application security. It lists the security requirements of various mobile application categories and references parts of the OWASP MSTG, that show how to verify the requirements, as well as Common Weakness Enumeration (CWE) IDs, elaborating on weaknesses, that may exist. The OWASP WSTG, on the other hand, is supported by the OWASP Application Security Verification Standard (OWASP ASVS) [17], which is a standard for web application security. It is structured similarly to the OWASP MASVS. These standards also helped us in constructing the testing order. Furthermore, they improved our understanding of what security level we could and should have expected from the application.

2.3. JSON Web Token (JWT)

A JSON Web Token [18] is a standard for securely sharing claims between two parties. JWTs are made up of three parts: A header containing the algorithm used to generate the signature, a payload containing a set of claims, and a signature of the header and the payload. The header and payload are in JavaScript Object Notation (JSON) format. Every part is first Base64 [19] encoded and then joined with a "." in the order from before. The tokens can be verified with the private or public key of the signature, depending on the used algorithm.

JWTs can be used for Single Sign-On (SSO) purposes, where a single authority is responsible for handling the login process, allowing the user account to be used across multiple independent systems. The stored claims usually reference a user account, such that a client can prove that it is logged in with this account.

```
query {
  student {
    firstName
    lastName
  }
}
```

Listing 2.1.: Example GraphQL query.

2.4. GraphQL

GraphQL [20] is a query language for Application Programming Interfaces (APIs) and a runtime for fulfilling those queries with existing data. There are multiple ways for a client to send the queries to the server. Usually, HTTP POST requests are used containing the query and any additional parameters. Listing 2.1 shows an example query to get the first and last name of all students in the database.

2.5. WebSocket Protocol

The WebSocket protocol [21] provides a communication channel over a TCP connection, allowing two parties to exchange data while keeping the connection alive. It is compatible with HTTP, making it possible for WebSockets to be initialized through HTTP requests. After a connection is established, WebSockets enable servers to send data to a client without it having to request the data first. This permits communication with less overhead than with alternatives like HTTP polling, where the client has to periodically request data from the server in order to receive it shortly after it becomes available.

3. Related Work

Android is developed in an open source environment and is the most used operating system on smartphones. Therefore, most of the work in mobile security is concentrated on Google's Android operating system. The following sections are going to examine what work related to ours already exists and how it relates exactly.

3.1. Mobile Applications

We want to start with work that revealed vulnerabilities in mobile applications without contact tracing functionalities.

There is a lot of work on security issues in mobile banking applications. Haupert et al. [22], for example, pointed out multiple security issues in the *N26* banking application. Vulnerabilities in the mobile application allowed them to manipulate monetary transactions, and inject HTML as well as JavaScript into a WebView. The backend also contained some vulnerabilities, like a sensitive information leak. Haupert et al. show how a combination of the found vulnerabilities could be used to take over user accounts.

Haupert and Müller [23] found ways to manipulate bank transactions in the mobile applications of *Deutsche Bank*, *Commerzbank*, and *Norisbank* by abusing their photoTAN implementation. For this they built on earlier work [24], where they performed a security analysis of the *Sparkasse* and *S-pushTAN-App* and found a way to freely manipulate monetary transactions initiated through a smartphone that is also set up to use the app based TAN procedure.

Furthermore, Dmitrienko et al. [25] investigated multiple CrontoSign/photoTAN two-factor authentication implementations, finding various weaknesses that allow bypassing them with cross-platform attacks. The vulnerable applications include numerous banking applications and other well-known applications. Moreover, they analyze many malicious Android applications targeting two-factor authentication schemes.

Other types of applications can also have severe vulnerabilities. The very popular mobile Android application *SHAREit*, for example, was found to contain multiple vulnerabilities [26]. A malicious application could be used to leak sensitive user data and execute arbitrary code by making the application install an arbitrary application.

Lastly, a Google employee found a vulnerability in another popular application called *Fortnite* [27]. This security issue allowed any app to substitute the actual *Fortnite* installation

image after it was downloaded and before the installation process. Therefore, causing any application to be installed instead.

The goal of this group is to report vulnerabilities to the developers of the corresponding applications, who in turn can fix these issues. This goal equals ours and improves the security of user data. We also believe that the process of finding the mentioned vulnerabilities was similar to our approach, making this group very similar to our work.

3.2. Contact Tracing

There also already exists much work on applications with contact tracing features. Some of this work is going to be listed below.

Muñoz [4] found and reported a vulnerability in the German *Corona-Warn-App*, which allowed anyone to compromise the server. It could be exploited by performing an unauthenticated HTTP request. This allowed an attacker to steal database credentials, for example. Fortunately, the data is stored in a completely anonymous manner.

Amnesty International [5] reported a critical security issue in the configuration of Qatar's *EHTERAZ* contact tracing application. This vulnerability, on the other hand, gave access to highly sensitive personal information of over a million users. The problem was an unauthenticated API endpoint that could be used to retrieve the name of any user, the location of confinement, if in confinement, and the name of medical facilities in which the user is being treated, if they are treated at the moment. Amnesty International [28] also analyzed other applications on their user privacy and pointed out any concerns.

There is also some work on the security of online attendance lists starting with Faßbender et al. [6], who found a vulnerability that exposed private data on the web application *forAtable*. Anyone could exploit this security issue by changing a parameter in the URL of the confirmation web page, giving them access to the sensitive data of others. One gets redirected to this page after being successfully registered in the attendance list.

Neumann reported two similar problems [7, 8] on different attendance lists which where found by the *Chaos Computer Club*. In the first report [7] they talk about many vulnerabilities in a cloud service by *gastronovi*. A combination of these issues allowed them to access millions of datasets going back up to one decade. Most of the datasets were reservations for restaurants.

The second report [8] is about another multitude of vulnerabilities found in the *darfichrein.de* application. Similar to the previous report, one could access hundreds of thousands of datasets using a combination of issues in their application. Fortunately, all the data was encrypted and therefore unusable.

Lastly, Stadler et al. [29] analyzed the potential harms of a large-scale deployment of the *Luca* contact tracing system. Among others, they found that sensitive information was leaked to the backend server which can be problematic if such a server is compromised or malicious in the first place.

The goals of security analyses of applications with contact tracing features align with our goals. The difference between this kind of work and security analyses of other applications is a slight discrepancy in the motivation. Such work is most often motivated by the COVID-19 pandemic and the resulting contact tracing applications, making them more similar to our work than the work on other applications.

4. Approach

After comparing our work to existing work, the approach to the security analysis is going to be explained in the following sections. An approach is like a plan. By clearly defining each step and the results of the step, time-consuming and less important actions can be avoided, increasing the amount of time available for the important work.

Our approach is divided into four parts: define the scope, find as much attack surface as possible, define a testing order, and test for the vulnerabilities.

4.1. Scope

Before doing any practical work, we defined the scope of our security analysis. A scope is a broad definition of the attack surface space of the target. Without it there are no boundaries to the analysis, making it possible to analyze third party services in order to find a vulnerability in the actual target. This, however, can be very time-consuming, complex, and inefficient. Therefore, we only focused on assets which were in control of *UniNow*.

a) UniNow Android Application and Interfaces Subject of the security analysis was the *UniNow* mobile Android application [11] and all interfaces it uses to fetch data, handle data and provide data. The analysis was limited to the Android application, as iOS applications require additional hardware and can not be emulated on any device.

b) Services of *.uninow.de Additionally, all services of all subdomains of `uninow.de` were in scope, because these had a high chance of handling important data. The main web application contained login pages for universities and student councils, for example.

4.2. Reconnaissance

After defining the scope, we created a map of the items in the attack surface of the target. We tried to find as many actual assets in the scope as possible. This step is important, as the more assets we know about, the better we understand how the application functions, and the more places we can analyze for potential vulnerabilities.

a) UniNow Android Application and Interfaces The first step of analyzing an application is installing it. This can be done using an Android smartphone or an emulator,

like the *AVD Manager* included in the *Android SDK* [30]. Emulators have some advantages over physical devices, as the environment of the application can be manipulated and monitored easier when using an emulator. Furthermore, multiple emulators can be used easily to simulate multiple devices, while using physical devices can be more costly.

Next, we wanted to identify all the available privilege levels. We used the privilege levels to divide the functionalities into groups. Therefore, a map of the functionalities of every privilege level was created by using the app manually or by decompiling the installation image and searching the source code. The second approach can yield functionalities which are implemented but unused, like API endpoints.

Going through every found functionality, tools were used to identify what interfaces are used by the application to fetch the data needed, what data it provides to do so and how it handles the data. The interfaces of particular importance were Inter-Process Communication (IPC) interfaces, network services, persistent storage, and the Android API.

b) Services of *.uninow.de First, we wanted to try to find subdomains of `uninow.de` by doing research online and by using brute force tools such as OWASP's *amass* [31]. Next, we used tools, e.g. *nmap* [32], to probe the ports of the found domains, to find what ports are open, and what services are running on these ports. The found services were then searched for their functionalities.

The result of the reconnaissance was a map of the attack surface, which also contained some relations between its assets. However, as we did not have an inside perspective of the systems involved, the relations were mostly based on assumptions.

4.3. Testing Order

Testing of the entire attack surface was most likely going to be too time-consuming. Therefore, we planned to limit the attack surface and the vulnerabilities we wanted to test for. A testing order allowed us to prioritize vulnerabilities, enabling us to test as many of the more important vulnerabilities as possible. The approach in devising the order was the following.

First, every vulnerability which might disclose private data and that is part of the OWASP MSTG or the OWASP WSTG, was assigned to the targets which might have it. Afterwards, the target-vulnerability pairs were ordered in descending order by (1) the security importance of the private data, which the target provides, handles or consumes, and by (2) the impact of the vulnerability and the ease of abusing the vulnerability. The OWASP resources mentioned in Chapter 2.2 helped us in dealing with the latter aspect.

This order allowed us to test for the most critical vulnerabilities first, while also getting into high to medium criticality depending on how much time we had left.

4.4. Testing

The testing was going to start with the first item in the vulnerability list. It was guided by different OWASP resources among others. Furthermore, as the analysis was done under a time constraint, we stopped testing when the time planned for testing was over.

Based on the vulnerability and target at hand, testing can require different approaches. One way to test a vulnerability is by reading the source code of the application and then testing the vulnerability depending on the acquired information. This is the most reliable

way, as the complete application logic is available and no assumptions need to be made. If the source code is not available, on the other hand, then many tests might be necessary to get a good understanding of the used logic and to ensure the correctness of the tests.

Not only the approach to testing, but also the actual testing can vary in many cases. This can be seen in our analysis. There are often tools available that can either just support the testing or even perform whole tests on their own. However, this is not always the case, forcing the creation of tailored tools or manual testing if the complexity of the tests is low enough.

5. Reconnaissance

This chapter explains how the reconnaissance step was performed on the mobile application and the network services of *UniNow*.

5.1. Mobile Application

The Android mobile application in scope has the ID *de.mocama.uninow* [11]. In particular the version `3.86.0` was used. We used the *PlaystoreDownloader* [33] to download the installation image from the Google Play Store.

To analyze the application dynamically, we used the *AVD Manager* from the *Android SDK* [30] to download and install an emulator. Furthermore, we used the *Frida* toolkit [34] to bypass the SSL certificate pinning, and *Zaproxy* [35] to analyze the network traffic from the application.

For the static analysis on the other hand, we used *UnZip* [36] to extract the data from the Android Package (APK) [37] installation image and get access to the byte code in the DEX format [38]. We also used the *Apktool* [39] to get access to the decrypted Android manifest file and to disassemble the DEX byte code further into Smali assembly [40]. *enjarify* [41] was used to convert the DEX files into jar files, which were than used with the *cfr* decompiler [42] to generate Java source files. Later on, the *jadx* [43] decompiler was used on the APK to get a second version of the Java source code. This version was easier to analyze in some cases.

While analyzing these files, we figured out that the application is a JavaScript hybrid and is using the *React Native* framework [44]. Although the JavaScript source code is being shipped with the installation image, it was compiled to a byte code that is interpreted by Facebook's *Hermes* JavaScript engine [45] during runtime. Since this made it quite difficult to analyze the source code, we used the *hbctool* [46] to disassemble the *Hermes* byte code to an assembler like code. The resulting code was still hard to analyze, but there was no other way to improve its readability.

However, many important functionalities were found to be implemented in Java, easing our analysis as the decompiled source code was hardly obfuscated. Nonetheless, we also found many other functionalities to be implemented on the JavaScript side, increasing the difficulty of analyzing them drastically, as we only had access to the *Hermes* byte code, which is still very new compared to Java and, therefore, lacks mature static analysis tools.

5.1.1. Privileges

When opening the application for the first time, users are registered automatically once the Terms of Use are accepted. Before being able to do anything, users have to choose a university. Afterwards, users have to choose a role. These roles seem to be configured by the universities, with common roles being *Student*, *Employee*, or *Guest*. The roles mandate what functionalities should be exposed to users and which university interfaces the application should use to interact with university accounts.

Users can freely change the university and the role without providing any authentication. Authentication is needed to access functionalities which need data provided by the university. One such example would be the calendar. It can import the schedule of one's university account, if the university has enabled it.

Further authentication is only needed to access functionalities of student councils or companies.

5.1.2. Functionalities and their Interfaces

The difficulty of analyzing the application statically as well as the fact that we only have one university student account to test with, made it in turn difficult to find and test all possible functionalities. We certainly missed functionalities, just because we can not test any functionality accessible to student councils or companies, as well as features that need authentication, but are disabled for our university. What follows is a list of the functionalities discovered and examined by us.

Cafeteria: Users can get the menus of the selected university's cafeterias without authenticating with their university account. This works by requesting the menu from `https://scraping.uninow.com/socket.io` using a WebSocket connection. This service is subsequently also called "scraping service".

Grades: The grades of a university account can be retrieved after logging in to the account once. The application uses a WebSocket connection to notify the scraping service about the authentication attempt. The scraping service then instructs the client on what HTTP request to perform with which headers and body. The client performs the request, replacing username and password placeholders if needed. The client then encodes the body of the HTTP response and sends it, including its headers, back to the *UniNow* server. The scraping service seems to scrape the needed information from the received response, formatting it if necessary, and responding to the client. In the case of authentication, this process is repeated multiple times until all needed information is collected. Finally, the credentials are stored locally on each user's phone. This seems to be the general procedure to keep the scraping process on the server side and to save the credentials on the client side only.

After authenticating successfully once, users can request their grades. Similarly to above, the client is again receiving instructions from the scraping service and performing the requests that need authentication. At last, the client receives the grades from the scraping service.

Users can also create custom grades which are stored locally on each user's device.

Calendar: The calendar service is mainly provided by *UniNow*. All data is stored on the server side and all information is exchanged with an API at `https://scraping.uninow.com`. This time through standard HTTP requests. Every request has to include an authentication token specifically for the calendar functionalities. Users can create, edit or cancel events, and backup, restore or reset their calendar. Moreover,

users can create appointments with multiple people by inviting them through a generated invitation link. Lastly, users can import all calendar items from the schedule of their university account. This process is done through WebSocket communication with the scraping service and is similar to the process of retrieving the grades.

After backing up the calendar, users are given a backup code. This code needs to be entered to restore the calendar. Specifically, the client sends the calendar authentication token and the backup code to `https://scraping.uninow.com/schedule/restore`. If the code is valid, the response is going to contain a new calendar authentication token that is used by the client in the future to access the old calendar. Therefore, the backups are stored on the *UniNow* servers.

The invitation links are shortened URLs to `https://meeting.uninow.com` with the appointment ID being sent over the *id* query parameter. After a user opens the URL on their mobile device, the *UniNow* application is opened, and the appointment information is retrieved using the GraphQL API at `https://graphql.uninow.com`. If the ID is correct, users can join the appointment and it is copied to their own schedule.

To-Dos: The data of the to-do list feature is saved on the server side by communicating with the GraphQL API at `https://graphql.uninow.com`. However, any settings are saved client side.

Library: Users can login to their university library account to see what books they have borrowed. Borrowed books can also be renewed, such that the users can keep them longer. All communication for this happens over a WebSocket connection with the scraping service.

Mail: The application can be linked to a university email account to send and receive emails. The authentication and all data transfer is done directly with the university mail server.

Campus Check-In: Users can register their attendance by scanning a QR code or by entering a check-in code. The code is then used to get the corresponding public PGP key to encrypt the data and a form is shown that requests the user to enter additional information. After filling out the form, all information is encrypted with the PGP key and sent to *UniNow* servers. Specifically, the GraphQL endpoint at `https://api.checkin.uninow.com/v1/graphql` is used for all of these purposes.

Radio: Universities can configure a radio station which can be listened to by users.

Sports: This module is not enabled for the *University of Würzburg*. Therefore, we have little information about it. It seems like users can log in to their university account to get access to their identity card. There also exists a schedule presumably for sport meetings. This schedule seems to be stored on *UniNow* servers.

Freshman: Student councils seem to be able to create so-called freshman areas. These areas contain a specific feed for freshmen, and additional information like tailored maps or links. Furthermore, student councils can choose that users require an access key to enter their area. The access tokens are checked by sending them to the GraphQL API at `https://graphql.uninow.com`.

Links: This is just a collection of links that is most likely configured by the university. The links are opened in the default web browser of the device.

Places: Universities seem also be able to configure a list of locations for the places module. Users can then use this module to get information about these locations.

In-App Browser: Instead of using the links module to provide a list of useful URLs to the user, a different type of module can be used. This module opens the configured link, unlike the links module, in a browser activity in the app itself. This module is often used by universities to give faster access to their e-learning platform. The *University of Würzburg* uses it for their *Moodle* [47] e-learning platform, for example.

Mini Games: Users can unlock this module by being active at least once in 5 of the last 7 days, allowing them to choose between a small amount of mini games to play.

Feeds: Feeds are made up of posts. Users can share, like or hide posts, follow and view feeds by companies or student councils, and report posts or feeds. All of these functionalities work through requests to the GraphQL API at `https://graphql.uninow.com`. The feeds also include job offers and other ads.

Support Chat: Almost every activity includes a button in the top right corner to open a support chat. These support chats are identified by an ID. Messages are sent by sending them with HTTP requests to an endpoint at `https://scraping.uninow.com`. Furthermore, the client gets messages by continually performing requests to the same endpoint as before.

App Locks: Users can choose to lock the *UniNow* application with their fingerprint or with a PIN. Afterwards, users are forced to enter the PIN or scan the fingerprint before being able to use the application. These checks are performed completely on the client side.

5.2. Network Services

While analyzing the mobile application we found that there are three domains of interest: `uninow.de`, `uninow.com`, and `uninow.io`. As these domains contain critical infrastructure, we assume that they are in control of the *UniNow GmbH*. Therefore, we added `uninow.com`, `uninow.io`, and their respective subdomains to the scope of the analysis.

5.2.1. Subdomain Enumeration

In the first step we created a list of valid resolvers by running the DNS server validation tool *dnsvalidator* [48] on a list of public DNS servers [49]. This resulting list increases the speed of DNS lookups for many applications, because it allows the applications to spread their lookups out among the name servers, decreasing delays from rate limits implemented by individual DNS resolvers.

We started off with brute forcing subdomains by using *gobuster* [50] with a popular word list [51] from the *SecLists* project. We continued running *gobuster* throughout this enumeration process after finding new subdomains, to check their existence on the other domains in scope.

We also used OWASP's *amass* [31], *assetfinder* [52], *findomain* [53], and *subfinder* [54] on every domain to find exposed subdomains on the internet. All of these tools were configured to use all freely-available sources, and to use our DNS resolvers list if possible. The found subdomains were checked using the *massdns* [55] DNS resolver.

Finally, using *altdns* [56] and *dnsgen* [57] allowed us to create permutations, alterations and mutations of the found domains. We checked these candidates with *massdns*.

This process yielded a combined 191 domains, of which 40 domains were subdomains of `uninow.de`, 85 were subdomains of `uninow.com`, and 65 were subdomains of `uninow.io`. A list of all found domains is located in Appendix A.

5.2.2. Port Scanning

We found that a lot of the domains where referencing the same IP addresses. Therefore, we could save time by only scanning these addresses. A script was created to group the domains by IP address, run *nmap* [32] on every IP address, and output the results. We only scanned all ports of the TCP side.

In this case the output showed that the ports 80 and 443 were open on the majority of the IP addresses. These were used for HTTP and HTTPS respectively. The only outlier was the domain `images.uninow.com` which had 13 open ports. A detailed summary of the results can be found in Appendix B.

5.2.3. Content Discovery

As most of the domains seemed to be running HTTP/HTTPS servers, we started of with visiting their websites to figure out what they are used for. We used *aquatone* [58] to get a fast overview of the home pages, *ffuf* [59] to brute force URL paths with a word list from *Assetnote* [60], and *waybackurls* [61] as well as *commoncrawl* [62] to get the URLs indexed by the *Wayback Machine* [63] and *Common Crawl* [64] respectively.

For the non-HTTP ports of `images.uninow.com` we used *nmap* to figure out what protocols they were using. Afterwards, we tried to find and use corresponding programs to talk to the services behind the open ports. None of the services answered our requests successfully.

We are going to list the noteworthy functionalities and the interfaces that we were able to identify in the following. There are a lot more web applications with no known purpose or only a partially known one. A list of all found web applications can be found in Appendix C.

https://uninow.com is the homepage of *UniNow*.

https://feed.uninow.com is a login page for student councils. The JavaScript source code suggests, that this website is used by student councils to create posts among others.

https://recruiting.uninow.com is a login page for companies. Since JavaScript sources can again give us a clue about the functionalities of this website, we know that companies seem to be able to edit, create, and archive ads and job offers using this web application.

https://schedule.uninow.com is a way to access the calendar from a browser. The website sends an ID to the scraping service through a WebSocket connection and displays the same ID in a QR code, which users are supposed to scan with the QR code scanner in the calendar module of the mobile application. After scanning the QR code, the mobile application sends a JWT that is scoped to the schedule functionalities to the scraping service with the scanned ID. Lastly, the latter sends the same token to the browser through the existing WebSocket connection. The browser can now use the token to make authenticated requests like the mobile application.

https://checkin.uninow.com hosts a web version of the Campus Check-In. `https://checkin.uninow.com/scan` offers a QR code scanner, `https://checkin.uninow.com/input` can be used to enter the check-in code manually, and `https://checkin.uninow.com/enter-data` can be used to enter data after the fact. The data is encoded with the public PGP key of the corresponding university like in the Campus Check-In module of the mobile application.

https://admin.checkin.uninow.com contains a login page and is probably used for administrating the Campus Check-In feature.

https://api.checkin.uninow.com is a GraphQL API for the Campus Check-In feature.

https://sport.uninow.com is again protected by a login page, but the source code tells us that the website allows universities to configure the sports module of the mobile application. Furthermore, it tells us that users can manage courses with the ability to change the location and the lecturer, or schedule and cancel appointments.

https://hasura-sport.uninow.io is a *Hasura* GraphQL Server [65] used for the sports module.

https://accounts.uninow.com contains a login page, requiring an email address and password to log in. An OAuth API at `https://accounts.uninow.com/api/v1/oauth` is used to authenticate. The same API is used to authenticate from a mobile device. In the latter case a *deviceId* and a *nativeId* are mainly used to do so. This service also contains other functionalities for managing account information.

https://support.uninow.io contains a login page presumably for customer support purposes.

https://support.uninow.com offers the ability to send account credentials (username and password) to *UniNow* through `https://support.uninow.com/accountfreigabe`. Users just need to append a valid chat ID through the *chat* query parameter. The chat IDs from the in-app support chat work too. This functionality probably exists so *UniNow* can use the user's account to resolve any complicated issues.

https://cooperation.uninow.com contains a login page for employees. This webpage seems to provide functionalities to change university banners, view contracts, and other functionalities to edit university settings or view statistics.

https://statistics.cooperation.uninow.com contains a login page to some internal company service.

https://statistics.uninow.com is used to collect usage data for the mobile application. Almost every user action is sent there together with the user's authentication token.

https://npe.uninow.io has a login page. Unfortunately, the JavaScript source code did not give us a lot of insight into what exactly the purpose of this web app is. However, we were able to trick the frontend into thinking we were authenticated, giving us information about the visible functionalities. This webpage is used by *UniNow* employees to change university configurations, view university status, and view user errors.

https://npe-backend.uninow.com is the API used by `https://npe.uninow.io`.

https://admin.open.uninow.com hosts an administration page for an instance of the *Shlink* [66] link shortener. An API key is needed to access any information or sensitive functionality.

https://open.uninow.com is probably the *Shlink* instance that can be configured from `https://admin.open.uninow.com`.

https://qrc.uninow.com has an endpoint at `https://qrc.uninow.com/template` which allows one to create a Campus Check-In information PDF by supplying a check-in code through the *code* query parameter. The *template* query parameter can also be used in combination with an integer value to get different templates. Furthermore, when requesting `https://qrc.uninow.com/pdf` the server responds with a few bytes that look like the beginning of a PDF file, but then drops the connection.

https://meeting.uninow.com is used to relay appointment invitations to the mobile application or prompt the visitor to download the application.

https://scraping.uninow.com is a scraping service. It enables all university account related functionalities for the mobile application and more.

https://graphql.uninow.com hosts a GraphQL API, which is used for many purposes, like retrieving information about a meeting, or updating user data. A valid JWT is needed to access it.

https://wapi.uninow.com seems to be the API that is used by many of the web based functionalities, like student council and company functionalities.

https://errors.uninow.io is used for tracking application errors with *Sentry* [67].

https://app.search.uninow.com is used by the mobile application to search for universities, jobs, and companies.

6. Security and Privacy Issues

During the reconnaissance we found many clues for potential vulnerabilities. As this decreased the time needed for testing, we started with testing them first. Afterwards, we worked through the testing order that we created after the reconnaissance step.

The following sections contain the security problems found during our analysis, the impact of the problems, and our recommendations for fixing them. The problems are sorted by their criticality in ascending order.

6.1. Disclosure of Multiple Secret Keys

UniNow uses the *Redux* [68] open-source JavaScript library in multiple occasions. This allows them to have a global state, which can be accessed from all parts of the application at hand. Furthermore, they are often using the *redux-persist* [69] library to persist the state of the *Redux* store across browser sessions. On top of that, they are using the *redux-persist-transform-encrypt* [70] library to encrypt the values in this persisted store with a key.

Two such keys were found to be stored in JavaScript files. The first of these keys `S1qBjwHR5ZNd209TykRndZ8hrwYm5h` was found in `https://feed.uninow.com/config.js`, and the second key `dannys-super-duper-secret-key-of-keys` was found in the main JavaScript chunk of `https://recruiting.uninow.com`.

The problem with these keys is that they seem to be used for all users of the application. The author of the library explicitly warns about this problem: "You SHOULD NOT use a single secret key for all users of your application, as this negates any potential security benefits of encrypting the store in the first place" [70].

In the first instance the store is persisted using the local storage of the browser. Therefore, we were able to successfully test that this is indeed the key used to encrypt the store by using the same JavaScript library for decryption as *redux-persist-transform-encrypt*, namely *crypto-js* [71].

Unfortunately, we were not able to test the second key, because the store only seems to be persisted after the user logged in. Lacking an account with the right access rights, we had no way to test this secret key.

The JavaScript files of `https://npe.uninow.io` also seem to contain very sensitive information. `https://npe.uninow.io/config.js` contains a JWT. This JWT expired a few

days before we found it, and it was created over a year ago. It seems to give access rights that were not seen on any other JWT accessible to us. `https://graphql.uninow.com` is explicitly listed in the audience section of the payload, suggesting that it might have had access to sensitive data of this GraphQL service.

Furthermore, the main JavaScript chunk of `https://npe.uninow.io` contains a public and private key called `REACT_APP_PUBLIC_KEY` and `REACT_APP_PRIVATE_KEY` respectively. We were not able to find what they are used for, as the JavaScript code does not seem to use these keys for anything. Nonetheless, their names make them noteworthy, as they suggest their usage in a cryptographic context.

6.1.1. Impact

These problems best fit into the CWE-200 "Exposure of Sensitive Information to an Unauthorized Actor" category.

The impact heavily depends on the secret value looked at. The first two keys are used to encrypt sensitive user data which is most likely stored on the client side, making it hard for an attacker to reach.

The JWT, on the other hand, might have been a critical security concern with the potential for an attacker to fetch sensitive user data without restrictions. These assumptions, however, need to be verified.

The same can be said about the last two values. Their impact varies depending on their current usage. They might be used for the generation of JWTs, or for a completely different benign purpose.

6.1.2. Recommendations

We strongly recommend to never store any secret keys or administration access tokens in a publicly accessible way.

The found keys should be verified. If they were usable after their disclosure, then an investigation into whether they were abused or not should be conducted. Furthermore, all sensitive keys should be removed, and the process of generating the first two keys should be remodeled, such that they are created and stored on the server side for every user. And, users should only receive their key to unlock their locally persisted *Redux* data after they log in successfully.

6.2. No Address Bar or Location Restrictions in In-App Browser

The *UniNow* mobile application offers the ability for universities to add links. These links are most often used to provide an easy access to the university's e-learning platform. However, they can also be used for other purposes.

Instead of opening the links in an external web browser, *UniNow* chose to use an in-app solution. We found that this solution lacks important security features of most other web browsers. In particular, there is no address bar to see the current location. Furthermore, there is no way of knowing where a link leads to when looking at it. An example web page opened in this browser can be seen in Figure 6.1.

The address bar of a browser is one of the only ways to see if the current website can be trusted or not. It is paramount in identifying phishing websites, which are tricking users into sharing sensitive information by pretending to be a trusted source.

The normal way of identifying phishing websites before visiting them in mobile browsers is to long press on the link, such that an information panel appears containing the full URL. This feature is also disabled for the in-app browser solution.

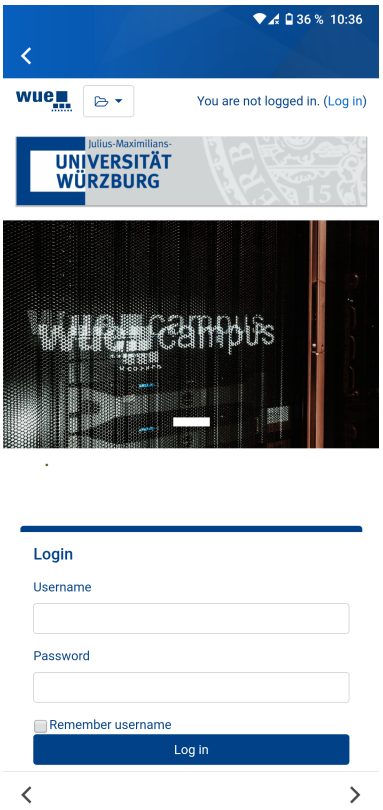


Figure 6.1.: Moodle login page of the *University of Würzburg* opened in the in-app browser.

6.2.1. Impact

This vulnerability fits best into the CWE-451 "User Interface (UI) Misrepresentation of Critical Information" weakness category.

Users of the in-app browser solution can easily be tricked into visiting and sharing sensitive information with phishing websites. This link module is often used by universities to allow users to easily access their e-learning platforms. The platforms often contain messaging systems, through which an attacker might be able to distribute phishing URLs.

6.2.2. Recommendations

Users have to be able to identify when they leave a secure web site. There are multiple possible ways of achieving this.

One way would be to enable the two missing features from above, delivering a familiar user interface. This, however, leaves little difference between the in-app browser and an external browser. Therefore, we believe it to be easier and more secure to use the default browser of the device instead.

Another solution would be to restrict the location of the user, and warn them when they are trying to visit an out of scope website. These websites could then be opened in an external browser to further separate both environments.

Our second suggestion is more complicated, as it has to be able to deal with certain edge cases, like e-learning platforms that use multiple domains. Consequently, we believe that the first solution should be preferred over the second one.

```
https://accounts.uninow.com/api/v1/verify/email/?email=me@example.com&
↪ redirect_url=https://accounts.uninow.com/auth/verify-email&
↪ signature=<SIGNATURE>&expires=1622724020
```

Listing 6.1.: Example URL to verify an email address.

```
https://accounts.uninow.com/api/v1/verify/email/?email=support@uninow.de&
↪ redirect_url=https://www.uni-wuerzburg.de/startseite
```

Listing 6.2.: Example URL to redirect to the homepage of the *University of Würzburg*.

6.3. Open Redirect in Email Verification URL

Open redirects are vulnerabilities, which redirect a user from a seemingly secure URL to an insecure one after visiting it.

We found an open redirect in the email verification URL of *UniNow*. Listing 6.1 shows an example of a benign URL generated by the service. One can see the *redirect_url* query parameter in the same listing. This parameter is used to redirect the user to a confirmation page, after the email was verified. Unfortunately, any value is accepted for this parameter, and other than this parameter only the *email* query parameter is required which also accepts any value.

Therefore, an attacker can construct a URL like in Listing 6.2 to trick their victim into visiting any website, this example redirects to the homepage of the *University of Würzburg*.

6.3.1. Impact

Open redirects are part of the CWE-601 "URL Redirection to Untrusted Site ('Open Redirect')" category.

They are often used in phishing attacks, but more severe attacks are also possible. For example, they can be used to steal OAuth 2.0 [72, 73] access tokens [74].

OAuth is an authorization framework used by web applications to request access to an account on another application. The application at `https://feed.uninow.com` seems to use OAuth 2.0 to get access to the user's Instagram account.

A *redirect_uri* query parameter has to be specified as a callback for OAuth 2.0. The access token is then sent to the requester by appending it in some way to the redirect URL specified in this parameter. Therefore, a whitelist is often used to restrict the values for this parameter. However, an open redirect can be used to bypass the whitelist and redirect the access token to an attacker controlled server, giving them full access to the account.

To abuse this hypothetical vulnerability, an attacker would have to prepare a URL with their redirect URL and trick the user into visiting this URL. The latter part can be done by sending phishing emails to university email addresses. When the user visits the URL, they are presented with the *UniNow* OAuth 2.0 application, asking for certain access rights to their account. If the user allows *UniNow* to access their account, they are redirected to the URL specified by the attacker, and the OAuth 2.0 access token is appended to the URL, effectively sharing it with the attacker.

The OAuth 2.0 application from *UniNow* on Instagram does not whitelist `https://accounts.uninow.com`, protecting its users from such an attack. This is the correct and

secure behaviour. Only if `https://accounts.uninow.com` was whitelisted, then such an attack would be possible. However, changes in the configuration of this OAuth 2.0 application or new applications registered by *UniNow* might be abusable with this open redirect in the future, if `https://accounts.uninow.com` is in their whitelist. Consequently, this open redirect alone may be a low severity issue, but with a possibility of causing larger issues later.

Open redirects can also be used to bypass other whitelists, enabling attacks through Server-Side Request Forgery (SSRF), for example [75]. SSRFs are weaknesses through which a server can be used to perform requests on behalf of the attacker. Fortunately, we were not able to find any such vulnerability on any *UniNow* service. But again, new functionalities in the future might enable this open redirect to be a severe issue, because it can be used to bypass whitelists.

6.3.2. Recommendations

We recommend to implement a whitelist for the *redirect_url* parameter, which only includes the used redirect URLs. This whitelist should be made up of explicit URLs, which should only be accepted if they match completely. Dynamic definitions using prefixes or regular expressions are often a source of errors, allowing an attacker to bypass the whitelist.

6.4. Access Tokens are Shared With UniNow

UniNow advertises the way university credentials are handled. They are supposedly never shared with them and are only stored locally on the user's device [76]. This is true for username and password pairs, for example, but not always true for access tokens.

Request instructions from `https://scraping.uninow.com/socket.io` over WebSockets are replied to with the results of the last performed request in a redirect chain. Therefore, if the first HTTP response is not a redirect, or if the *followRedirect* and the *followAllRedirects* instruction options are set to *false*, then the first response is returned to the scraping service. Otherwise, only the last response in the redirect chain is returned.

The reply is made up of a JSON array, where the only element is a stringified JSON object. As can be seen in Listing 6.3, the replies can carry important information forcing the mobile application to at least strip out any user credentials before sending the reply to *UniNow*. In particular, the headers, the URL, and the Base64 encoded body of the HTTP response should be searched for credentials.

Cookies are the most common way of handling authentication tokens, which is why we started looking for a leak in the *Set-Cookie* response header. We found the disassembled Java source code presumably responsible for creating the reply JSON object. It shows that only the last occurrence of a header stays in the reply. Consequently, a *Set-Cookie* header can only exist in the final reply if its HTTP response is the last in the redirect chain, and if it is the last *Set-Cookie* header in the HTTP response.

We were not able to find such an endpoint because of the small amount of university accounts we have access to. The login endpoint of the *University of Würzburg*, used to fetch student grades, sets another cookie after the one carrying the access token. But, if we use *Zaproxy* to remove the second *Set-Cookie* header before the response reaches the mobile application, then the authentication cookie is included in the reply to the scraping service. This only suggests that the client does not remove access tokens from the reply headers, but we can not be sure, because we had to tamper with the HTTP response to get this result.

```
{
  "headers": {
    "": "HTTP/1.1 200 200",
    "accept-ranges": "bytes",
    "age": "0",
    "cache-control": "no-store, must-revalidate, max-age=120",
    "content-type": "text/html;charset=UTF-8",
    "date": "Wed, 26 May 2021 18:46:31 GMT",
    "set-cookie": "JSESSIONID=X; Path=/qisserver/; Expires=Wed, 20 Sep
    ↪  2017 22:23:01 GMT; HttpOnly",
    "strict-transport-security": "max-age=15768000; includeSubDomains",
    "vary": "Accept-Encoding",
    "X-Android-Received-Millis": "1622054790014",
    "X-Android-Response-Source": "NETWORK 200",
    "X-Android-Selected-Protocol": "http/1.1",
    "X-Android-Sent-Millis": "1622054789807",
    "x-cache-hits": "0",
    "x-cache": "MISS",
    "x-content-type-options": "nosniff",
    "x-frame-options": "SAMEORIGIN",
    "x-ua-compatible": "IE=Edge",
    "x-varnish": "27258714"
  },
  "method": "GET",
  "statusCode": "200",
  "responseUrl": "https://wuestudy.zv.uni-wuerzburg.de/qisserver/pages/
  ↪  cs/sys/portal/hisinoneStartPage.faces?chco=y",
  "body": "dGhpcyBpcyBhbiBleGFtcGxlIHJlc3BvbnNlIGJvZHkgYXdvZnl1dG5hd2Yg
  ↪  YXd5ZnRvIG5hd2Zv\\ndHUgbmF3Zm90eXUgbmF3b2Z1dAo="
}
```


Listing 6.3.: Example reply to a request instruction from the scraping service.

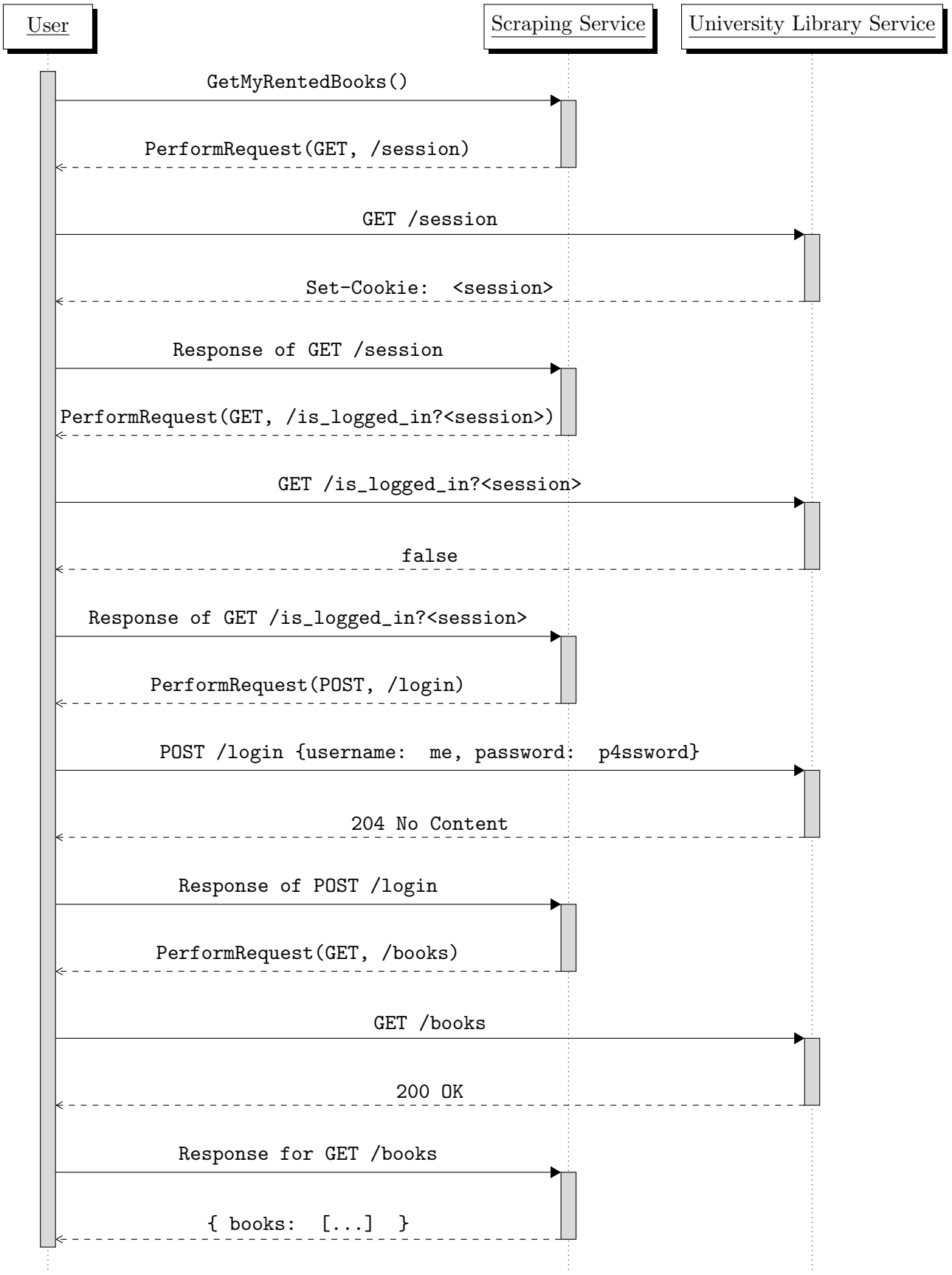


Figure 6.2.: Simplified sequence diagram of how the application fetches the user's borrowed books for the library of the *University of Würzburg*. `<session>` is the value of a session cookie.

Next, we checked whether access tokens are removed from the response body and noticed how the library module, again, for the *University of Würzburg* exposes the access token of the library account to the scraping service. The library service uses a session cookie which is set when visiting any of its web sites for the first time. When logging in, the session cookie is not replaced with a new one, but is linked internally to the logged in user.

This particular login process is made up of four request instructions. A sequence diagram of the communication between the client, the scraping service, and the library server can be seen in Figure 6.2. The first request is made to a seemingly arbitrary library web site. The HTTP response sets the session cookie if it was not set before, and a harmless cookie is returned through the *headers* JSON object to the scraping service. However, the session cookie is sent to the scraping service through the Base64 encoded response body.

Next, the scraping service instructs the client to perform another request to a URL which contains the session token from the first response. Hence, the server had to scrape its body, find the session cookie value, and construct the new URL. This request looks like it is checking whether the session token is linked to a user.

If the session token is not linked to a user, then the scraping service asks the mobile device to perform a login request with the user's username and password. The library server links the logged in user to the given session cookie, if the credentials were correct. Furthermore, this request, like the second one, does not return any sensitive data to the scraping service.

Finally, the client is instructed to request a page containing the borrowed books of the user from the university server. The response of the server is returned to the scraping service with no sensitive information in it. The borrowed books are scraped from the HTML body by the server and returned in a standardized format to the application.

This chain of request instructions shows how a valid access token can be shared with *UniNow* through a returned request body, when the university endpoint meets the right requirements. Our testing also suggests that access tokens can be shared through the returned HTTP headers. We were not able to find any university endpoint accessible to us, that returns sensitive data through the *responseUrl*. However, there are university endpoints which include session cookies in the URL they redirect to, indicating the possibility of such a leak.

6.4.1. Impact

We believe the impact to be rather low, as only *UniNow* has access to these access tokens and they are most likely not going to exploit any of them deliberately. However, these access tokens might be stored or handled in an insecure fashion.

The front end files of *npe.uninow.io* suggest that failed requests are stored in some kind of log system. These log files seem to be searchable by *UniNow* employees, so they can detect errors in the scraping system. Therefore, they might contain further context for each error, like the whole request chain. If this is the case, then the access tokens would be logged, if an unrelated error in the scraping system occurs.

As *UniNow* might be unaware of the sensitivity of the data in these logs, they might handle this data in a less secure manner. Not deleting this data periodically, or sharing it through insecure channels could jeopardize its security and, therefore, the security of the access tokens.

6.4.2. Recommendations

The scraping system is closed source and operated on the server side, leaving us with little information about how it works and what data it relies on. Nonetheless, if *UniNow* wants

to stick to its claim about how university credentials are handled, then the client has to remove all access data from its replies to the scraping service.

We recommend to remove every potential access token, as an individual configuration for every endpoint, that only removes the access token of this endpoint, can introduce many errors. In contrast, removing all potential access tokens can be done easier by stripping all cookie values, authentication header values, and *responseUrl* values from the reply.

Such a solution is going to break all functionalities of the scraping service that rely on this kind of data. Therefore, workarounds and other approaches should be devised to enable these functionalities again.

6.5. Sensitive Data in Application Backups

Data used in Android applications can be backed up. There are multiple methods of creating such a backup. Google added an automatic backup feature for applications which automatically syncs at most 25MB of application data with the user's Google Drive account [9]. This feature is automatically enabled for applications that target Android 6.0 (API level 23) and above.

The *UniNow* mobile Android application has the *android:allowBackup* flag set to *true* and targets an API level of 29, allowing such backups to be performed. Any data present in the backup files is transferred to Google servers without informing the user. Hence, there should not be any sensitive data in these backup files.

After using the `adb shell bmgr` command to set a local transport, backing up the application data, finding the backup file, and extracting the data using the *tar* [77] command, we found multiple sensitive chunks of data. The *deviceId* used in the account takeover of Section 6.9, session data and cookies from the in-app browser module, the SQLite database used for email storage, and more were present in an unencrypted state.

6.5.1. Impact

Anybody who has access to the backup files of the *UniNow* mobile application has access to sensitive information about the user, like the data named above. Google also warns about this problem, emphasizing that Android cannot guarantee the security of the backed up data, because the backup transport may differ from device to device. Therefore, they suggest to be careful about storing sensitive data in backup files [78].

6.5.2. Recommendations

The sensitive data should either be encrypted or removed from the backup file. Whole SQLite databases can be encrypted with a password through the *SQLCipher* [79] library, among others. To exclude files from backups, the *android:fullBackupContent* flag can be used to specify which files to include or exclude in the backups [80].

6.6. Disabled Strict SSL on University Endpoints

The scraping service seems to only instruct the mobile application to perform requests that need to be authenticated. Hence, when fetching public information like cafeteria menus, there is no need for a university account to do so and the scraping service fetches the data on its own.

The request instruction message is a JSON array of length 2. The message is identified as a request instruction, if the first element of the array is the string `"request"`. Listing 6.4

```
{
  "url": "https://wuestudy.zv.uni-wuerzburg.de:443/qisserver/
  ↪  rds?state=user&type=1&category=auth.login",
  "method": "POST",
  "strictSSL": false,
  "dontUpgrade": true,
  "omitRequestBody": false,
  "timeout": 30000,
  "headers": {
    "User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
    ↪  (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36"
  },
  "form": {
    "javaScriptSupported": "false",
    "userInfo": "",
    "ajax-token": "c239fe80-b71c-11eb-9cf4-b0dc47b4dec6",
    "submit": "",
    "asdf": "[[USER:da41f894216695aff8380ab452ed3061]]",
    "fdsa": "[[PASSWORD:da41f894216695aff8380ab452ed3061]]"
  },
  "jar": "ByBs9Axltu"
}
```

Listing 6.4.: Example request instruction from the scraping service.

only contains the details of the request instructions which are found in the second element of the JSON array as a stringified JSON object. The HTTP request instructions from the scraping service contain many details on how the mobile application should perform a request.

The option *strictSSL* seemed to be of particular importance, as a value of `false` would indicate that the application should not impose strict SSL rules for the given request. We found the functions handling the request instructions in the disassembled Java source code. They showed that any certificate from any issuer is accepted when the *dontUpgrade* and the *strictSSL* options are set to `false` and an HTTPS URL is targeted.

While searching manually for universities with other properties, we stumbled upon some endpoints which meet these requirements and are used to transfer sensitive data. We chose the library login request of the *Mainz University of Applied Sciences* to test our theory. As the WebSocket connection to the scraping service uses TLS and its certificate is checked correctly, we had to use a DNS server to serve the IP address of our proxy instead of the IP address of the login server of the university library. This way all other TLS connections were untouched, and the request instructions were able to be delivered.

We used the *Dnsmasq* [81] DNS server and a *Zaproxy* instance. Furthermore, we made sure to disable any installed CA certificates on the mobile device and regenerate the CA certificate used by *Zaproxy*. And as expected, the application accepted the certificate from our proxy and sent the request, including the tried username and password, to our *Zaproxy* instance.

It is also noteworthy, that the disassembled Java source code indicated that the application only allows connections using the protocols *TLSv1*, *TLSv1.1*, and *TLSv1.2*, when HTTPS is used and *dontUpgrade* is set to `false` which is the default value. However, *TLSv1*

```
{
  "port": 143,
  "protocol": "IMAP",
  "baseURL": null,
  "hostname": "mail.hs-rm.de",
  "connectionType": "STARTTLS",
  "authType": null,
  "checkCertificate": false,
  "__typename": "MailConfig",
  "ingoing": true,
  "useEmail": false
}

{
  "port": 587,
  "protocol": "SMTP",
  "baseURL": null,
  "hostname": "mail.hs-rm.de",
  "connectionType": "STARTTLS",
  "authType": null,
  "checkCertificate": false,
  "__typename": "MailConfig",
  "ingoing": false,
  "useEmail": false
}
```


Listing 6.5.: Example email configurations for incoming and outgoing emails.

as well as *TLSv1.1* are both deprecated since March 2021 [82]. We also tested this by using *Dnsmasq* and *Zaproxy*. *Dnsmasq* was needed because only the requests from request instructions allow these two protocols and the request instructions can not be transfered using them. *Zaproxy* allows the user to configure the allowed protocols for the proxy connections. The tests showed the correctness of our hypothesis.

While looking at the configuration files of the mobile application, we noticed an option called *checkCertificate* in the email server configurations. The name of this option and what we believe to be the responsible source code indicate that a value of `false` instructs the application to trust all given SSL certificates.

The application fetches the corresponding email configurations, after the user switches to another university. Multiple configurations for outgoing and incoming emails are possible, such that the user can choose between the different options. Listing 6.5 shows such configurations of the *RheinMain University of Applied Sciences*.

As the *checkCertificate* option in all configurations of the *University of Würzburg* were set to `null`, a script was used to iterate through a list of all universities to fetch the corresponding configurations, and output all configurations that have this option set to an actual boolean value.

The used list contained 800 universities and was found in a JavaScript file on an *UniNow* website. 145 configurations had the *checkCertificate* option set to a boolean value. 143 of them had it set to `false`, affecting 38 universities. A full list of these configurations can be found in Appendix E.

A student of the *RheinMain University of Applied Sciences* let us use her university email account to test our hypothesis. We used the *striptls* [83] auditing proxy with the *GENERIC.Intercept* test, and *Dnsmasq* to only redirect communication to the configured mail server through the proxy.

This resulted in us being able to eavesdrop on the communication between the mobile application and the email servers. This only works when the *checkCertificate* option is set to `false`. `null` values are interpreted as `true`.

6.6.1. Impact

The first two issues belong into the CWE-295 "Improper Certificate Validation" category.

An attacker could set up a malicious hotspot in a university with a vulnerable endpoint or mail server configuration. Anyone, who uses this hotspot and triggers the *UniNow* mobile application to exchange data using a bad configuration, would enable the attacker to eavesdrop on the exchange. Many university endpoints could have *strictSSL* set to `false`, as we stumbled upon a few by looking at a small number of universities manually.

But, even if *strictSSL* is set to `true`, which is the default value, then a *dontUpgrade* value of `false`, also the default value, can enable the class of TLS downgrade attacks. These can be used to trick the client and server into using one of the deprecated protocols instead of the most secure one, making it potentially possible to abuse vulnerabilities in these protocols [84].

6.6.2. Recommendations

Our research on the JavaScript files of `https://npe.uninow.io` suggests that *UniNow* employees set *strictSSL* to `false` when error messages indicate a certificate error. If this is true, it would create the need for a system which checks the certificate of each endpoint periodically and resets *strictSSL* when the certificate works again. So, either this system does not exist, is broken, or does not check the certificates often enough.

However, there should not exist a switch to turn off strict SSL checking in the first place. *UniNow* sacrifices the security of their users' data for an improved user experience in rare situations. So they use insecure connections, when secure connections do not work. Instead, there should be no connections, when secure connections do not work.

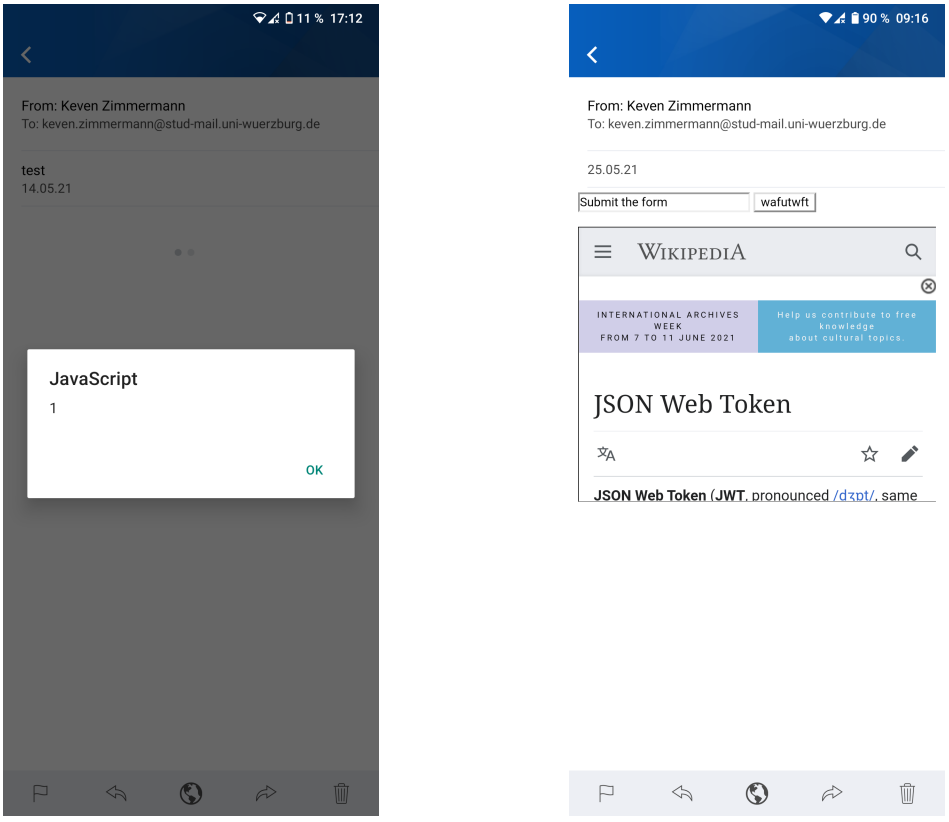
Therefore, we recommend to keep *strictSSL* as well as *checkCertificate* always enabled, and implement an error message which is displayed when the creation of a secure connection to a university server failed.

The usage of deprecated TLS protocols can be fixed easily by removing the deprecated protocols from the list of allowed ones.

6.7. JavaScript Injection in Mail Module

The mail module is used to send and receive emails from a university email account. These emails can contain HTML in their body, which is rendered by the application. We found that `script` tags are also taken into account. The JavaScript source code inside them is interpreted and executed after opening the email.

Hence, opening an email with a body like `<script>alert(1)</script>` is rendering a blank email, whereupon an alert box containing the text `1` is opened immediately. Figure 6.3a shows the resulting view.



(a) Alert call of a JavaScript injection using the `script` tag.

(b) HTML form and an iFrame of a Wikipedia website.

Figure 6.3.: Result of opening an email containing insecure HTML tags.

```
var request = new XMLHttpRequest();
request.open("GET","https://requestbin.io/10c1zv11");
request.send();
```

Listing 6.6.: Example JavaScript payload for making a request to `https://requestbin.io/10c1zv11`.

Sending another email with the body `<script>alert(Object.keys(window))</script>` lists the keys of the *window* object. The *window* object is often globally available in many browser contexts. The resulting list contained a key called *ReactNativeWebView*. This suggests the use of the *React Native WebView* [85] for rendering emails.

Furthermore, we tested if outbound connections are possible from the JavaScript context by sending ourselves an email with the JavaScript source code from Listing 6.6. After opening the email, an HTTP GET request was successfully sent to our *RequestBin* [86] HTTP request recorder, proving that outbound connections are not restricted.

Lastly, some tests for other HTML tags were performed. These showed the usability of the *iframe* tag, as well as the *form* tag combined with the *input* and *button* tags. As can be seen in Figure 6.3b, the *iframe* tag allowed us to embed a web site in the email, and the *form* tag allowed us to create a HTML form which upon submitting opened the target web site in an external browser.

6.7.1. Impact

An attacker can execute arbitrary JavaScript code in a WebView environment by sending their victim an email. This allows the attacker to perform requests that are only restricted by the same-origin policy [87].

Furthermore, the attacker could use the working *iframe* tag, as well as the *form* tag to trick the victim into sharing sensitive information, by letting them fill in forms. Other mail clients filter out these, and other, tags to prevent such an attack.

Lastly, there are multiple ways of performing requests to external servers, allowing users to be tracked. HTML mail clients often disable any requests to external servers, until the user allows it, to counteract this privacy issue.

6.7.2. Recommendations

Secure HTML email views are hard to create, because of the wide variety of abusable HTML tags. Therefore, we recommend the use of an existing solution, as building a new one is most certainly going to result in unforeseen errors.

If nonetheless a custom solution is still preferred, we recommend to make sure that every possibility of executing JavaScript is disabled, and that all abusable HTML tags are also disabled. A whitelist of secure tags can also be created. Again, a similar project or guide should be used as a reference to avoid any errors.

We believe that this vulnerability belongs into the CWE-74 “Improper Neutralization of Special Elements in Output Used by a Downstream Component (‘Injection’)” category, because if a WebView is used, then abusable tags of HTML mails should be removed before rendering.

6.8. Brute-Forceable Appointment Invitations

As mentioned above, users are able to create appointments in the calendar module of the mobile application. This also automatically creates an invitation URL that can be used to invite others to the appointment. Unfortunately, we found a way to enumerate these links and extract sensitive user information using them.

The invitation links look like this: `https://uninow.page.link/<SHORT_LINK_ID>`. The `<SHORT_LINK_ID>` placeholder is replaced by a string of four or more upper or lower case letters, and digits. `https://uninow.page.link/4tU3`, for example, might be a valid invitation link. The domain `uninow.page.link` tells us that the application uses *Firebase Dynamic Links* [88] to create them. In particular, the developers seem to have set the *suffix* option to *SHORT*, which makes the links a minimum of 4 characters long. Note that the *Firebase* documentation warns about this attack vector: "Use this method if sensitive information would not be exposed if a short Dynamic Link URL were guessed" [89].

We used a small script to create a list of all possible invitation links with an ID of length 4 and used *ffuf* with 40 threads and without a delay between the requests to find the valid ones. The link shortener does not seem to have a rate limit. This allowed us to test about 1.5 million of 14.7 million 4-character combinations in 40 minutes, averaging 625 requests per second. Furthermore, we managed to find about 2850 valid link IDs. To categorize these IDs, we had to look at their redirect location. For this we simply used *ffuf*'s verbose flag, which outputs them.

320 of the IDs were for appointment invitations. The other valid combinations redirected to post shares, company shares, job shares, cafeteria links, book links, or others that did not expose any user data. The invitation short links on the other hand redirect to URLs looking like this: `https://uninow.de/ScheduleLinkDetail?id=<APPOINTMENT_ID>` with `<APPOINTMENT_ID>` being the ID of the appointment.

These IDs can be used with the GraphQL endpoint at `https://graphql.uninow.com` to get more details about the appointment. Specifically, the mobile application is using the GraphQL query seen in Appendix D. No special authentication is needed to perform this query. Therefore, anybody can enumerate these appointment invitations and get potentially sensitive information. Listing 6.7 shows a real example of the information an attacker can get.

6.8.1. Impact

While sampling appointments, we found some that range back to April of 2019, raising the question whether the appointments or their short links ever get deleted. This would allow anyone to enumerate all appointments that ever existed. Fortunately, the sensitive fields are not all mandatory to fill out when creating an appointment, causing the names of the members, the name of the host, the title of the appointment, and its notes to often be arbitrary. This makes it more difficult to track users across appointments. Nonetheless, an attacker can use the *memberID*s, or other recurring data between appointments to do that. In particular, the name of the host and the members is often the same, as the application recommends the last used name for the next appointment too.

By manually sampling a few of the found appointments, we came up with an appointment at the headquarters of *UniNow*.

6.8.2. Recommendations

This problem is made up of the following two parts: (1) Unnecessarily broad access rights after the appointment is over, and (2) missing cryptographic security or a rate limit to prevent brute force attacks.

```
{
  "data": {
    "meeting": {
      "id": "<REDACTED>",
      "title": "Erdbeer pflücken und Triathlon",
      "type": "Treffen",
      "color": "#78DBDC",
      "blocks": [{
        "location": null,
        "allDay": false,
        "duration": "6:00",
        "notes": "Hallo MH, Lust vor dem Triathlon mit mir Erdbeeren zu
        ↪  pflücken und danach an den See zu fahren? Ich hoffe, 15 Uhr
        ↪  Feierabend zu machen :) ",
        "repeat": 4,
        "__typename": "Block"
      }],
      "slots": [{
        "_id": "<REDACTED>",
        "startTime": 15,
        "date": "Wed Jun 05 2019 00:00:00 GMT+0000 (Coordinated Universal
        ↪  Time)",
        "__typename": "Slot"
      }],
      "members": [{
        "memberID": "<REDACTED>",
        "name": "",
        "status": "pending",
        "slots": [],
        "__typename": "Member"
      }],
      "host": {
        "name": "MH",
        "__typename": "Host"
      },
      "chosenSlot": null,
      "status": "pending",
      "ownStatus": "pending",
      "hasTime": false,
      "links": {
        "forSend": "https://uninow.page.link/<REDACTED>",
        "__typename": "Links"
      },
      "__typename": "Meeting"
    }
  }
}
```


Listing 6.7.: Example of an appointment found by brute-forcing.

```
{
  "userAgent": {
    "system": "ANDROID",
    "version": "3.86.0",
    "systemVersion": "11"
  },
  "device": {
    "deviceId": "8109371f-f60b-44e1-be77-66c34b78fcc5",
    "legacyId": null,
    "nativeId": "Z74BzTLplDQ8FdVBNTJfR4HjrXgNa2YNUV8fyqno3m4=\n"
  },
  "applicationId": "de.mocama.UniNow",
  "grantType": "hmac"
}
```


Listing 6.8.: An example request body of a login request performed by the mobile application.

The first problem can be addressed, e.g., by only allowing users who have voted for an appointment and the host to access its information after the appointment took place. This would preserve the information for the participants only.

We believe that the second problem can be solved by setting the *suffix* option of the *Firebase* links to *UNGUESSABLE*, which would make the path component of each link 17 characters long. The existing links should be invalidated and regenerated.

This weakness seems to fit best into the CWE-334 “Small Space of Random Values” category.

6.9. Account Takeover Using Device IDs

The mobile application does not provide a user account in the traditional sense, as users do not need to register manually and provide a username and password, for example, to log in. Instead, *UniNow* is creating and assigning a user account automatically to every mobile phone. Therefore, sharing user data between phones is not possible without tampering with the application or its configuration.

We found a way to take over these user accounts. To understand this process, we first need to explain how the application is authenticating its users. All the services, that we found, use a JWT to authenticate incoming HTTP requests. The mobile application, in particular, gets its JWT by performing a POST request to `https://accounts.uninow.com/api/v1/oauth/token`. An example JSON body of such a request can be found in Listing 6.8.

When looking at the decoded payload of the JWTs, like in Listing 6.9, one can see that the JWTs only store the given *deviceId*. Therefore, the *deviceId* has to be used by the servers to uniquely identify the referenced account. It is also worth mentioning, that the *deviceId* in the login request is optional. When providing *null* as its value, the JWT is going to contain a newly generated *deviceId*, and is therefore presumably referencing a new user.

To make this authentication system work, the server would need to interpret each login request in the following way. The *deviceId* should be treated like the username in a traditional authentication system, and the *nativeId* as the password. This way the *deviceId*

```
{
  "sub": "8109371f-f60b-44e1-be77-66c34b78fcc5",
  "acr": "device",
  "jti": "04b29914-43dd-4531-a6b4-33222b84fe16",
  "https://uninow.com/jwt/claims": {
    "x-uninow-allowed-profiles": [],
    "x-uninow-default-profile": "",
    "x-uninow-device-id": "8109371f-f60b-44e1-be77-66c34b78fcc5"
  },
  "aud": [
    "*.uninow.com"
  ],
  "iss": "accounts.uninow.com",
  "iat": 1620914316,
  "exp": 1621000716
}
```


Listing 6.9.: An example of a JWT used by the mobile application for authentication.

would reference the user's account, and the *nativeId* would serve as the key which can only be provided by the user's device. Unfortunately, we can only assume that it was supposed to work this way, because we do not know much about the *nativeId* due to the difficulties of analyzing the application statically. We only know that the *nativeId* seems to stay the same across multiple installations on the same device.

Nonetheless, the fact is that the login endpoint is not checking the *nativeId* before creating a valid JWT with the provided *deviceId*. Thus, anyone can create a valid authentication JWT, if they know the *deviceId* of the user. We found a few ways on how an attacker could get this identifier of their victim's account.

The first way is pretty simple and does not need a lot of explanation, as it was already explained above. The *memberID*s which represent the users that responded to an appointment invitation are the *deviceId*s of the user accounts. This means that a victim just has to accept an appointment invitation, of which the attacker knows the unique invitation ID, to take over their account without them being able to notice.

Furthermore, we found that the calendar web application at `https://schedule.uninow.com` can be mimicked to trick somebody into leaking it. After scanning the QR code with the QR code scanner in the schedule activity of the mobile application, the web application receives a JWT that can only be used for the schedule functionalities. This JWT also contains the *deviceId* of the referenced user, allowing somebody who knows this JWT to escalate their privileges by acquiring a JWT with all privileges.

An attacker can therefore replicate this process to trick victims by doing the following. First, they create a WebSocket connection to `https://scraping.uninow.com/socket.io`. When creating such a WebSocket the server sends the ID of the socket connection to the client. The attacker now inserts their socket ID into the URL `uninow://schedule/qrCodeID<SOCKET_ID>` by replacing the placeholder `<SOCKET_ID>` with their actual socket ID. Lastly, the attacker has to create a QR code of the resulting URL and trick the victim into scanning it with the QR code scanner of their calendar. As long as the WebSocket connection is alive, the attacker is going to receive the JWTs of their victims from the server.

Tricking somebody into scanning their QR code should also not be very hard, as the application is often used to scan QR codes for other purposes. The trap is therefore already building on the user's habits. An attacker could, for example, hang up a poster next to a Campus Check-in QR code which advertises that by scanning the provided QR code, with the QR code scanner of the calendar, students can add important dates to their calendar.

6.9.1. Impact

The OWASP MSTG suggests the CWE-287 "Improper Authentication" category for this vulnerability.

We consider the impact to be critical, as taking over somebodies account gives the attacker access to a lot of personal data. This data includes, all calendar entries, all to-do list entries, all liked jobs and companies, as well as all feeds the user follows. There might of course be more information which we do not know about, because we did not have access to all modules. Furthermore, an attacker can completely impersonate their victim and perform all actions on their behalf.

Additionally, getting the *deviceId* requires minimal user interaction, as an attacker can enumerate appointment invitations at a reasonable speed. Targeted attacks need more interaction from the victim, because the victim has to accept an appointment which the attacker knows the ID of, or scan the attackers QR code.

6.9.2. Recommendations

We assume that the *nativeId* is unique per app installation. We believe that this takeover exists, because the endpoint does not check whether the given *deviceId* was first created with the given *nativeId*. Implementing this check on the server side and never sharing the *nativeId* is enough to fix this problem.

Furthermore, the developers should make sure that both IDs are generated in a cryptographically secure manner. Their length would then make them hard to brute force. We also recommend the implementation of an IP based rate limit to secure them even better.

6.10. Unauthorized Access to `https://npe.uninow.io`

Judging from the JavaScript files, `https://npe.uninow.io` seemed to have important functionalities for managing the *UniNow* university configurations. Therefore, it is secured with a login webpage, which does require a specific account unaccessible with the known registration options.

However, we noticed that it uses the *redux-persist* JavaScript library to persist the used *Redux* store across sessions by saving it to the web browsers local storage. The local storage can be accessed and edited using the *DevTools* in *Chromium* [90] based browsers.

There we found a single stored key/value-pair. The value of this pair is the current configuration. The default configuration can be observed in Listing 6.10. There one can also see a parameter called *token* in the *auth* property. Inserting a JWT generated by *UniNow* for the mobile application, for example, allowed us to access the user interface behind the login page. However, no functionalities worked, as the backend server always returned a status `401 Unauthorized` on every request.

Towards the end of our analysis we came back to this problem. After, playing around with the user settings, like the username or avatar image URL, we were able to access multiple internal functionalities. Retesting it with a different JWT that had nothing in

```
{
  "filter": "{\"university\":{\"search\":\"\",\"editor\":
  ↪  null,\"status\":null,\"fileStatus\":null},\"plugins\":{\"search\":
  ↪  \"\",\"editor\":null,\"group\":null,\"fileStatus\":null}}",
  "universities":
  ↪  "[{\"id\":\"de_ovgu\",\"name\":\"Otto-von-Guericke-Universität
  ↪  Magdeburg\",\"status\":\"LIVE\",\"modules\":{}},{\"id\":
  ↪  \"de_tud\",\"name\":\"Technische Universität
  ↪  Dresden\",\"status\":\"LIVE\",\"modules\":{}}]",
  "auth":
  ↪  "{\"role\":\"guest\",\"token\":null,\"name\":null,\"id\":null}",
  "checklist": "{\"checked\":[]}",
  "account": "{\"avatar\":null}",
  "_persist": "{\"version\":-1,\"rehydrated\":true}"
}
```


Listing 6.10.: The default configuration of `https://npe.uninow.com`

common with the first one, showed that we did not have to change any settings to gain the same result. Consequently, either our first try somehow changed the configuration of the backend server, or *UniNow* made a change which allowed anyone with a valid *UniNow* JWT to access this internal service.

We confirmed access to the following functionalities: Reading university configurations, searching university configurations for static strings, viewing plugin configurations, and finding user errors by device ID. The last functionality was not tested completely, because we were not able to find a device ID which had produced an error before, and were also not able to find any error produced by us after provoking one by tampering with the scraping request instructions.

University configurations are made up of university information, as well as account configurations and module configurations for the scraping service. Plugin configurations seem to be used as a base for the module configurations, enabling similar parts of the module configurations to be reused.

As we did not want to cause any denial of service, we did not edit any university configurations. However, we did go to an edit view which provides in-browser editors to edit the selected configuration. After doing that, the configuration switched from the status *OPEN* to *LOCKED*. Furthermore, not only was our selected username displayed with the date of last edit, but the configuration was the only one with the *LOCKED* status that we were able to edit again. All other configuration with the same status had a disabled edit button.

All of this suggests, that we were indeed able to edit the university and plugin configurations.

Lastly, we found that the secrets for the schedule JWTs are located in the corresponding schedule configurations. They can also be searched for using the university configuration search tool.

6.10.1. Impact

We believe this to be the most severe of all the found vulnerabilities, as there is a host of attack possibilities available to an attacker.

An attacker would have control over the mail configurations of every university, allowing them to redirect the email to a server controlled by them or abuse the *checkCertificate* flag to make man-in-the-middle attacks possible for email exchanges. Furthermore, the leaked JWT secrets could be used to access a user's schedule data by using their device ID.

The most severe attack, however, can be conducted by changing the login URLs of the universities to a attacker controlled URL. The attacker could set up their server to proxy the incoming requests to the actual university server, allowing them to steal all the login credentials without their victims being able to notice. Similar attacks are possible by changing the URLs of other configurations to steal incoming cookies, for example.

Lastly, having access to these configurations would allow an attacker to change them arbitrarily causing a denial of service.

6.10.2. Recommendations

We believe that any internal servers should only be accessible through a Virtual Private Network (VPN). This would add a second authentication layer for all endpoints by only allowing certain IP addresses to communicate with the servers. This is a widely used approach and is used by the *University of Würzburg*, for example, on `https://www.uni-wuerzburg.de/typo3/` among others.

7. Other Tested Attack Vectors

While the last chapter concerned itself with the found security problems, this chapter is going to contain a list of the attack vectors which we tested for and found to not exist. The following sections are either on a tested type of attack or on a tested part of the attack surface.

7.1. Privilege Escalations

The OWASP Top Ten contains an item for the *Broken Access Control* vulnerability category, suggesting that access controls are often enforced poorly. Therefore, we tried many different API endpoints in an effort to escalate the privileges of a user account. Our goal was to access restricted API endpoints which had a high chance of being able to provide sensitive information of other users.

Our main testing target was `https://accounts.uninow.com` which was responsible for registering new users, as well as providing new JWTs for existing users. We also targeted other APIs like the GraphQL API at `https://hasura-sport.uninow.io/v1/graphql`, as it seemed to also provide a way to register new accounts.

We tried fuzzing hidden request parameters and API endpoints using the *ffuf* fuzzing tool. Although we were able to find new attack surface, we were not able to escalate the privileges of any of our test accounts in any meaningful way.

7.2. GraphQL APIs

The *Sensitive Data Exposure* vulnerability category, ranked 3rd place in the 2017 OWASP Top Ten, encouraged us to test ways of accessing sensitive data using the GraphQL APIs at `https://api.checkin.uninow.com/v1/graphql`, `https://graphql.uninow.com`, and `https://hasura-sport.uninow.io/v1/graphql`. In particular by using email/password accounts created through requests to `https://accounts.uninow.com/api/v1/register`, as well as device ID accounts.

We did not find any way to access `https://hasura-sport.uninow.io/v1/graphql`, as it required unaccessible privileges for every functionality. `https://api.checkin.uninow.com/v1/graphql` and `https://graphql.uninow.com`, on the other hand, were mostly accessible with only a few restricted functionalities.

`https://api.checkin.uninow.com/v1/graphql` was of particular interest to us, as it is used to enable the contact tracing functionalities of *UniNow*, which were the motivation for this work.

A regular expression was used to find all GraphQL queries and mutations in all found JavaScript files, as well as the disassembled mobile application. This data was used to get a better grasp of the available functionalities. We also used *clairvoyance* [91] and *GraphQLmap* [92] to find more accessible functionalities. The gathered data was especially useful for the API at `https://graphql.uninow.com`, as it did not allow introspection.

We tested every found query and mutation, as well as most of the fields and schemas. But despite these efforts, we were not able to find anything abusable in any of the mentioned APIs. We also tried mutations to add new public keys for contact tracing using its API.

7.3. SQL Injections

SQL injections are possible when user provided data is not sanitized before the data is inserted into a SQL query. This allows an attacker to manipulate the performed SQL query and extract sensitive data from the used database or perform dangerous actions [10]. They belong to the first ranked category *Injection* of the 2017 OWASP Top Ten, and are one of the most critical security risks [16].

We used the *sqlmap* [93] automatic SQL injection tool to test various endpoints, which seemed to be vulnerable because of error messages or other clues. Fortunately, none of the tested endpoints and parameters was vulnerable to this kind of attack.

7.4. Cross-Site Scripting (XSS)

Cross-Site Scripting can be abused by an attacker to make their victims browser execute arbitrary JavaScript code [10]. This type of vulnerability is contained in the *Cross-Site Scripting XSS* category, which is ranked 7th on the 2017 OWASP Top Ten.

We were particularly interested in finding a XSS vulnerability in one of the web applications, because most of them store sensitive data, including the JWT of the user, in the browser's local storage, which can be accessed using JavaScript [94]. This would not be possible, if the data was stored using cookies with the *HttpOnly* flag [95].

There were multiple web applications handling sensitive user data. And, whenever we had the option of providing data, we took a closer look on how this data is sanitized by testing various symbols.

We did not find a single instance where user data is not sanitized properly before being inserted into JavaScript source code or HTML.

7.5. Cloud Storage

UniNow uses multiple cloud storage solutions to provide content to their users. We have identified the following three systems: the *Amazon AWS S3* buckets [96] at `https://image.uninow.com` and `https://uninow-advertising.s3.amazonaws.com`, as well as the *Contentful* space [97] at `https://cdn.contentful.com/spaces/9t2pd5q435yg` using an access token found in a JavaScript file at `https://admin.checkin.uninow.com`.

The access rights to all three systems were checked by using the *s3scanner* and the *Contentful* JavaScript SDK *contentful.js*. Both buckets were only readable, allowing one to get the content of an item only by knowing the exact URL. This is the most secure configuration.

The *Contentful* space, on the other hand, allowed us to list all contents of the space, and read the contents of every item. However, no sensitive information was found while searching it.

8. Disclosure

As we found some security problems with a high to critical severity, we chose to do a responsible disclosure by giving the developers at least 30 days to fix the most severe issues before publishing our work.

We prepared a disclosure document containing an unfinished version of Chapter 6, and first contacted the support of *UniNow* by email on the 9th of June in the late afternoon. We explained the context of our analysis and asked where to send our results to. An answer from *Stefan Wegener*, one of the CEOs and a founders, was received in the late evening of the same day, asking for the details of the found security issues. These were sent to them in the morning of the next day. We received an almost immediate reply from Mr. Wegener, where they explained that they are going to work through the vulnerabilities and take actions accordingly. Finally, we received another response in the late afternoon of the same day, explaining their point of view on our findings.

The disclosure proceeded with further communication through emails and video conversations. *UniNow* appreciated our efforts and communicated their points in a highly professional manner, focusing on solutions to our findings. We are going to explain further details of the found vulnerabilities, *UniNow*'s point of view, and their proposed solutions in the following sections.

8.1. Disclosure of Multiple Secret Keys

UniNow verified that they are using *redux-persist* in multiple applications to store the application state. As sensitive data is only stored after the user has logged in, *UniNow* chose to stop encrypting the locally saved *Redux* store. Furthermore, they are going to remove all other keys and secrets found by us. Some of them were only used in earlier versions of the corresponding application.

8.2. No Address Bar or Location Restrictions in In-App Browser

This problem was acknowledge and is going to be addressed by showing a notification when the user is using a link to visit a URL which is out of the secure scope. They are also going to check the feasibility of an address bar.

8.3. Open Redirect in Email Verification URL

UniNow was not sure about the open redirect vulnerability. We understand that this type of vulnerability is a point of debate on whether or not it is an actual vulnerability on its own. For example, Google does not reward a found open redirect vulnerability in their bug bounty program, if it can not be used as leverage for more severe vulnerabilities [98]. We explained our concerns about the potential future impact of this vulnerability. However, it is *UniNow*'s choice on whether they want to fix this problem or not.

8.4. Access Tokens are Shared With UniNow

UniNow explained that some functionalities are not possible without sharing the access tokens with their scraping service. Furthermore, they pointed out how their Terms of Use implicitly allow this. Nonetheless, we suggested to clarify their statements regarding the sharing of access data in places other than their Terms of Use, and suggested ways of using a key-value store on the client side to store and reference these access tokens. They already use key-value stores in a similar fashion for university credentials.

They also explained that access tokens are replaced in error logs with random characters, making them unusable.

8.5. Sensitive Data in Application Backups

This flaw was acknowledged. They are going to check the feasibility of deactivating the backup functionality completely, as they do not see any advantages in it.

8.6. Disabled Strict SSL on University Endpoints

We assumed that certificate checking for a university endpoint gets disabled, when errors indicate a fault in the used certificate. *UniNow* verified this assumption and admitted that a system for checking and re-enabling the certificates does not exist. Therefore, they are going to develop such a system. We agree with this solution. However, we also pointed out how the user should know about whether or not their connections are secure. We suggested to implement a warning, similar to how browsers warn their users about bad HTTPS connections.

8.7. JavaScript Injection in Mail Module

UniNow was able to reproduce our findings. They are checking the feasibility of a settings option which allows users to choose to enable or disable JavaScript in emails.

8.8. Brute-Forceable Appointment Invitations

Old appointments are going to be deleted automatically in the future and the length of the invitation links is going to be checked. *UniNow* are also working on a new calendar system, which is going to replace the current one.

8.9. Account Takeover Using Device IDs

UniNow acknowledges this problem. Currently, they are working on a traditional email and password account system to substitute their current system. Users are going to have to create an account before using functionalities, for which data is stored on their servers. Consequently, account takeovers using device IDs are not going to be possible, as they are not going to be needed for to log in to an account.

8.10. Unauthorized Access to https://npe.uninow.io

UniNow already uses a VPN for their internal services, as recommended by us. However, a change in a regular expression allowed anyone to bypass it. This verifies that anyone was able to get full access to `https://npe.uninow.io`. The bug was fixed immediately and an investigation into how similar problems can be prevented in the future is going to be conducted.

9. Conclusion

We performed a security analysis of the *UniNow* application in a four step process: Definition of the scope, reconnaissance, construction of the testing order, and the testing itself. We were particularly focused on the security of sensitive user data and their new COVID-19 contact tracing feature. The following sections are about the results of our work and what similar future work might look like.

9.1. Results

Our analysis revealed multiple security problems concerning user data, ranging from low to critical severity. We were able to show, how sensitive user data was threatened by phishing attacks through a lack of location information in the in-app browser, and discovered an open redirect, which could lead to exploitable vulnerabilities later. Furthermore, we explained how, contrary to what the company claims, access tokens are shared with *UniNow*, sensitive user data was backed up automatically to Google servers, SSL certificate checking was disabled deliberately for many university endpoints, and how JavaScript injections were possible in the mail module.

One of the more severe problems was that appointment invitations were brute-forceable, enabling an attacker to access data about the time and date, the location, as well as the participants of the appointment. There was appointment data ranging back as far as a few years. We also found a way to use the participant information, among others, to take over the participants' user accounts, giving an attacker access to all of their calendar, to-do list, and information about what jobs and posts they liked as well as what feeds they are following. Finally, we discovered an accessible internal service, which allowed anyone to edit university configurations giving them the ability to set up an attack through which they could hijack the credentials to the university accounts of all users.

These findings prompted us to perform a responsible disclosure by communicating our results to *UniNow* and working with them in choosing the right solutions. *UniNow* took our concerns very seriously and appreciated our work. They responded swiftly and asked for our point of view on potential solutions. We were able to explain our position on the findings and suggest improvements to their proposed solutions. They accepted our suggestions and promised to check the feasibility of implementing them. We are staying in contact with *UniNow* until all problems are resolved.

Our work tangibly improved the security of user data handled by *UniNow*, as the developers now know about the found security problems and are going to address them. Some of

the vulnerabilities had the potential of dealing tremendous damage to the users of the application. Therefore, our work mostly benefited university students and employees. Moreover, the analysis also underlined the security of the contact tracing solution, which is used by many universities, including our own. This also encourages other universities to start using or move to this solution. *UniNow* was surprised by our findings and is working on improving their security auditing process. For example, they are considering to host a bug bounty program, allowing third-party security researchers to test their application in a given scope without fearing any legal action. Researches are also often rewarded by such programs for finding vulnerabilities, encouraging further testing on the program. If such a bug bounty program was in place before, then another researcher may have been interested in testing the program, which could have revealed some of these vulnerabilities. Therefore, we believe this to be a step in the right direction.

9.2. Future Work

Future work in this area of research can be of two kinds. Firstly, we want to encourage more security analyses on popular applications that handle a lot of user data in the future. Such analyses benefit the users of the application directly, and hold the developers accountable for lacking security principles. The results of our work show the feasibility of security analyses conducted in this manner. Although *UniNow* performs annual security audits, we were still able to find critical vulnerabilities that threatened sensitive user data. However, such analyses do not have to find new vulnerabilities, as the fact that no vulnerabilities were found would underline the security promises of the audited application, serving to reassure its users, and benefiting security focused developers.

Secondly, as there are still many vulnerabilities and attack surfaces that we were not able to test for in the *UniNow* application, our work can be built upon to test the rest of the vulnerabilities. In particular, we were not able to test any of the functionalities which were not accessible to us. This includes the ID card functionality which seems to provide an electronic ID card to university students, as well as the sports module which seems to be used to schedule sports related university events. The latter is known to handle sensitive user information like appointments, including their time and location. This module is also accompanied by a web application, which might be vulnerable to web based attacks. Furthermore, we also did not have access to any student council or company account, as well as many other services. These could contain more vulnerabilities. Specifically, the way access tokens are stored in the browser's local storage allows XSS vulnerabilities to be abused to steal them, making XSS vulnerabilities a high priority vulnerability to test for in the various web applications. *UniNow* may be inclined to provide testing accounts for another analysis, which would provide the needed access rights to test these applications.

Furthermore, we did not have enough time to test the scraping service. This service is essential to all university functionalities and has the ability to instruct user devices to perform HTTP requests. It receives a lot of different data from its users, encouraging input based attacks. There might also exist more scraping functionalities that we are unaware of, increasing the attack surface and possibilities of potential exploitation.

We also refrained from testing any brute force attacks on any of the login pages, as we did not want to cause a denial of service by locking the account of an employee. We found some promising employee email addresses. A missing rate limit combined with simple passwords could lead to a successful brute force attack, allowing an attacker to access important infrastructure, like the support service, and extract sensitive data.

We found many working services, but only know the purpose of some of them. These might have APIs which can be used to access sensitive data. Old, unused APIs have an especially

high chance of not enforcing new access rights, potentially enabling unauthorized access to the data of other users.

Lastly, *UniNow* might implement the solutions to the found problems incorrectly, making another analysis after the implementations interesting. Specifically, the new login system as well as the new calendar system have a high complexity, increasing the chance of security flaws in their corresponding implementations.

In conclusion, there are many places left to look for vulnerabilities in the *UniNow* application and many more applications to perform security analyses on. We therefore encourage the continuation of our work on this application and look forward to more work on applications similar to *UniNow*.

List of Figures

List of Listings

Acronyms

APK Android Package

API Application Programming Interface

CWE Common Weakness Enumeration

IPC Inter-Process Communication

JSON JavaScript Object Notation

JWT JSON Web Token

OWASP Open Web Application Security Project

OWASP MSTG OWASP Mobile Security Testing Guide

OWASP WSTG OWASP Web Security Testing Guide

OWASP MASVS OWASP Mobile App Security Requirements and Verification Standard

OWASP ASVS OWASP Application Security Verification Standard

SSO Single Sign-On

SSRF Server-Side Request Forgery

VPN Virtual Private Network

XSS Cross-Site Scripting

Bibliography

[1] K. T. Eames and M. J. Keeling, "Contact tracing and disease control," *Proceedings of the Royal Society of London. Series B: Biological Sciences*, vol. 270, no. 1533, pp. 2565–2571, 2003.

[2] Robert-Koch-Institut, "Corona-Warn-App." [Online]. Available: https://play.google.com/store/apps/details?id=de.rki.coronawarnapp

[3] R. Sun, W. Wang, M. Xue, G. Tyson, S. Camtepe, and D. Ranasinghe, "An Empirical Assessment of Global COVID-19 Contact Tracing Applications," 2021. [Online]. Available: https://europepmc.org/article/PPR/PPR272765

[4] A. Muñoz, "Securing the fight against COVID-19 through open source." [Online]. Available: https://securitylab.github.com/research/securing-the-fight-against-covid19-through-oss

[5] Amnesty International, "Qatar: Contact tracing app security flaw exposed sensitive personal details of more than one million." [Online]. Available: https://www.amnesty.org/en/latest/news/2020/05/qatar-covid19-contact-tracing-app-security-flaw/

[6] S. Faßbender, J. Gunzenreiner, and T. Schröder, "Mit Webapps gegen COVID-19," Jul 2020. [Online]. Available: https://www.modzero.com/modlog/archives/2020/07/06/mit_webapps_gegen_covid-19/index.html

[7] L. Neumann, "CCC hackt digitale 'Corona-Listen'," Aug 2020. [Online]. Available: https://www.ccc.de/de/updates/2020/digitale-corona-listen

[8] ——, "CCC meldet Schwachstellen bei weiterer digitaler 'Corona-Liste'," Sep 2020. [Online]. Available: https://www.ccc.de/en/updates/2020/ccc-meldet-schwachstellen-bei-weiterer-digitaler-corona-liste

[9] OWASP, "OWASP Mobile Security Testing Guide." [Online]. Available: https://owasp.org/www-project-mobile-security-testing-guide/

[10] ——, "OWASP Web Security Testing Guide." [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/

[11] UniNow GmbH, "UniNow App." [Online]. Available: https://play.google.com/store/apps/details?id=de.mocama.UniNow

[12] Apple Inc., "iOS App Store: UniNow - Studium & Karriere." [Online]. Available: https://apps.apple.com/de/app/uninow-bequem-durchs-studium/id969806189

[13] UniNow GmbH, "UniNow." [Online]. Available: https://uninow.de

[14] OWASP, "OWASP Foundation | Open Source Foundation for Application Security." [Online]. Available: https://owasp.org/

[15] ——, "OWASP Mobile Top Ten." [Online]. Available: https://owasp.org/www-project-mobile-top-10/

[16] ——, "OWASP Top Ten Web Application Security Risks." [Online]. Available: https://owasp.org/www-project-top-ten/

[17] ——, "OWASP Application Security Verification Standard." [Online]. Available: https://owasp.org/www-project-application-security-verification-standard/

[18] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC 7519, May 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7519.txt

[19] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," RFC 4648, Oct. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4648.txt

[20] Facebook, "GraphQL | A query language for your API." [Online]. Available: https://graphql.org/

[21] A. Melnikov and I. Fette, "The WebSocket Protocol," RFC 6455, Dec. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6455.txt

[22] V. Haupert, D. Maier, and T. Müller, "Paying the price for disruption: How a fintech allowed account takeover," in *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*, 2017, pp. 1–10.

[23] V. Haupert and T. Müller, "On app-based matrix code authentication in online banking." in *ICISSP*, 2018, pp. 149–160.

[24] ——, "Auf dem Weg verTAN: Über die Sicherheit App-basierter TAN-Verfahren," *Sicherheit 2016-Sicherheit, Schutz und Zuverlässigkeit*, 2016.

[25] A. Dmitrienko, C. Liebchen, C. Rossow, and A.-R. Sadeghi, "On the (in) security of mobile two-factor authentication," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 365–383.

[26] J. Chang and E. Duan, "SHAREit Flaw Could Lead to Remote Code Execution," Feb 2021. [Online]. Available: https://web.archive.org/web/20210405095518/https://www.trendmicro.com/en_us/research/21/b/shareit-flaw-could-lead-to-remote-code-execution.html

[27] Google LLC., "Fortnite Installer downloads are vulnerable to hijacking," Aug 2018. [Online]. Available: https://issuetracker.google.com/issues/112630336

[28] Amnesty International, "Bahrain, Kuwait and Norway contact tracing apps among most dangerous for privacy." [Online]. Available: https://www.amnesty.org/en/latest/news/2020/06/bahrain-kuwait-norway-contact-tracing-apps-danger-for-privacy/

[29] T. Stadler, W. Lueks, K. Kohls, and C. Troncoso, "Preliminary Analysis of Potential Harms in the Luca Tracing System," *arXiv preprint arXiv:2103.11958*, 2021.

[30] Google Developers, "Android Studio." [Online]. Available: https://developer.android.com

[31] OWASP, "OWASP Amass." [Online]. Available: https://owasp.org/www-project-amass/

[32] Nmap, "Nmap: the Network Mapper - Free Security Scanner." [Online]. Available: https://nmap.org/

[33] G. C. Georgiu, "PlaystoreDownloader." [Online]. Available: https://github.com/ClaudiuGeorgiu/PlaystoreDownloader

[34] "Frida - A world-class dynamic Instrumentation framework." [Online]. Available: https://frida.re

[35] OWASP, "OWASP Zed Attack Proxy (ZAP)." [Online]. Available: https://www.zaproxy.org/

[36] Info-ZIP, "UnZip." [Online]. Available: http://infozip.sourceforge.net/UnZip.html

[37] Android Open Source Project, "Application Fundamentals." [Online]. Available: https://developer.android.com/guide/components/fundamentals

[38] ——, "Dalvik Executable format." [Online]. Available: https://source.android.com/devices/tech/dalvik/dex-format

[39] C. Tumbleson, "Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps." [Online]. Available: https://ibotpeaches.github.io/Apktool

[40] B. Gruver, "smali/baksmali." [Online]. Available: https://github.com/JesusFreke/smali

[41] Google and R. Groose, "enjarify." [Online]. Available: https://github.com/Storyyeller/enjarify

[42] L. Benfield, "cfr." [Online]. Available: https://github.com/leibnitz27/cfr

[43] skylot, "jadx - Dex to Java decompiler." [Online]. Available: https://github.com/skylot/jadx

[44] Facebook, "React Native - Learn once, write anywhere." [Online]. Available: https://reactnative.dev/

[45] ——, "hermes." [Online]. Available: https://github.com/facebook/hermes

[46] P. Sommalai, "hbctool." [Online]. Available: https://github.com/bongtrop/hbctool

[47] Moodle HQ, "Moodle - Open-source learning platform." [Online]. Available: https://moodle.org/

[48] vortexau, "dnsvalidator." [Online]. Available: https://github.com/vortexau/dnsvalidator

[49] Public DNS Server List, "nameservers.txt." [Online]. Available: https://public-dns.info/nameservers.txt

[50] O. Reeves, "gobuster." [Online]. Available: https://github.com/OJ/gobuster

[51] D. Miessler, J. Haddix, and g0tmi1k, "subdomains-top1million-110000.txt." [Online]. Available: https://raw.githubusercontent.com/danielmiessler/SecLists/dc04568e57db11935584f7b63ec4c50719e9af46/Discovery/DNS/subdomains-top1million-110000.txt

[52] T. Hudson, "Find domains and subdomains related to a given domain." [Online]. Available: https://github.com/tomnomnom/assetfinder

[53] E. Tolosa, "The complete solution for domain recognition." [Online]. Available: https://github.com/Findomain/Findomain

[54] ProjectDiscovery, Inc, “subfinder - Fast passive subdomain enumeration tool.” [Online]. Available: https://github.com/projectdiscovery/subfinder

[55] blechschmidt, “massdns.” [Online]. Available: https://github.com/blechschmidt/massdns

[56] S. Shah, “altdns.” [Online]. Available: https://github.com/infosec-au/altdns

[57] ProjectAnte, “dnsgen.” [Online]. Available: https://github.com/ProjectAnte/dnsgen

[58] M. Henriksen, “aquatone.” [Online]. Available: https://github.com/michenriksen/aquatone

[59] J. Hoikkala, “ffuf.” [Online]. Available: https://github.com/ffuf/ffuf

[60] A. Pty.Ltd., “raft-large-words-lowercase.txt.” [Online]. Available: https://wordlists-cdn.assetnote.io/data/manual/raft-large-words-lowercase.txt

[61] T. Hudson, “waybackurls.” [Online]. Available: https://github.com/tomnomnom/waybackurls

[62] ghostlulzhacks, “commoncrawl.” [Online]. Available: https://github.com/ghostlulzhacks/commoncrawl

[63] Internet Archive, “Wayback Machine.” [Online]. Available: https://web.archive.org/

[64] Common Crawl, “Common Crawl.” [Online]. Available: https://commoncrawl.org/

[65] Hasura Inc., “Hasura - Instant GraphQL APIs for your data.” [Online]. Available: https://hasura.io/

[66] A. Celaya, “Shlink - The URL shortener.” [Online]. Available: https://shlink.io/

[67] Functional Software, Inc., “Application Monitoring and Error Tracking Software.” [Online]. Available: https://sentry.io/welcome/

[68] Dan Abramov and the Redux documentation authors, “Redux - A Predictable State Container for JS Apps.” [Online]. Available: https://redux.js.org

[69] Z. Story, “Persist and rehydrate a redux store.” [Online]. Available: https://github.com/rt2zz/redux-persist

[70] M. Bowers, “Encrypt your Redux Store.” [Online]. Available: https://github.com/maxdeviant/redux-persist-transform-encrypt

[71] E. Vosberg, “crypto-js: JavaScript library of crypto standards.” [Online]. Available: https://github.com/brix/crypto-js

[72] D. Hardt, “The OAuth 2.0 Authorization Framework,” RFC 6749, Oct. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6749.txt

[73] M. Jones and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” RFC 6750, Oct. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6750.txt

[74] V. Li, “Stealing OAuth Tokens With Open Redirects.” [Online]. Available: https://sec.okta.com/articles/2021/02/stealing-oauth-tokens-open-redirects

[75] detectify, “The real impact of an Open Redirect vulnerability.” [Online]. Available: https://blog.detectify.com/2019/05/16/the-real-impact-of-an-open-redirect/

[76] UniNow GmbH, "FAQ: Wer fragt gewinnt." [Online]. Available: https://uninow.de/en/faq

[77] Free Software Foundation, Inc., "GNU Tar." [Online]. Available: https://www.gnu.org/software/tar/

[78] Google Developers, "Encrypt your Redux Store." [Online]. Available: https://github.com/maxdeviant/redux-persist-transform-encrypt

[79] Zetetic LLC, "SQLCipher." [Online]. Available: https://www.zetetic.net/sqlcipher/

[80] Google Developers, "Back up user data with Auto Backup." [Online]. Available: https://developer.android.com/guide/topics/data/autobackup.html

[81] S. Kelley, "Dnsmasq - network services for small networks." [Online]. Available: https://thekelleys.org.uk/dnsmasq/doc.html

[82] K. Moriarty and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1," RFC 8996, Mar. 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc8996.txt

[83] M. Ortner, "striptls - auditing proxy." [Online]. Available: https://github.com/tintinweb/striptls

[84] E. S. Alashwali and K. Rasmussen, "What's in a Downgrade? A Taxonomy of Downgrade Attacks in the TLS Protocol and Application Protocols Using TLS," in *Security and Privacy in Communication Networks*, R. Beyah, B. Chang, Y. Li, and S. Zhu, Eds. Cham: Springer International Publishing, 2018, pp. 468–487.

[85] React Native Community, "React Native Cross-Plattform WebView." [Online]. Available: https://github.com/react-native-webview/react-native-webview

[86] Runscope Inc., "React Native Cross-Plattform WebView." [Online]. Available: https://github.com/react-native-webview/react-native-webview

[87] Mozilla, "Same-origin policy." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

[88] Google LLC., "Firebase Dynamic Links." [Online]. Available: https://firebase.google.com/docs/dynamic-links

[89] ——, "Firebase Dynamic Links." [Online]. Available: https://firebase.google.com/docs/dynamic-links/rest

[90] ——, "Chromium." [Online]. Available: https://www.chromium.org/Home

[91] N. Stupin, "Clairvoyance - Obtain GraphQL API schema despite disabled introspection." [Online]. Available: https://github.com/nikitastupin/clairvoyance

[92] Swissky, "GraphQLmap." [Online]. Available: https://github.com/swisskyrepo/GraphQLmap

[93] B. D. A. Guimaraes and M. Stampar, "sqlmap: automatic SQL injection and database takeover tool." [Online]. Available: https://sqlmap.org/

[94] Mozilla, "Using the Web Storage API." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API

[95] ——, "Using HTTP cookies." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

[96] Amazon Web Services, Inc., "What is Amazon S3?" [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[97] Contentful, "Spaces and organizations." [Online]. Available: https://www.contentful.com/help/spaces-and-organizations/

[98] Google LLC., "Open redirectors." [Online]. Available: https://sites.google.com/site/bughunteruniversity/nonvuln/open-redirect

[99] Metabase, "Metabase." [Online]. Available: https://github.com/metabase/metabase

Appendix

A. Subdomain Enumeration Results

A.1. Subdomains of uninow.com

accounts.uninow.com
account.uninow.com
admin.checkin.uninow.com
admin.open.uninow.com
analytics.uninow.com
answers.uninow.com
api.answers.uninow.com
api.assets.uninow.com
api.checkin.uninow.com
api.facility.uninow.com
api-legacy.uninow.com
api.uninow.com
api.user-feedback.uninow.com
app.search.uninow.com
assets.uninow.com
auth-legacy.uninow.com
auth.uninow.com
business.uninow.com
capacity.checkin.uninow.com
checkin.uninow.com
cooperation-develop.uninow.com
cooperation.uninow.com
c.uninow.com
features.uninow.com
feed.uninow.com
games.uninow.com
gql-legacy.uninow.com
gql.uninow.com
graphql.uninow.com
images.uninow.com
kooperation.uninow.com
log.uninow.com
mail.uninow.com
meeting.uninow.com
open.uninow.com
purchase.uninow.com
qrc.uninow.com
recruiting-dashboard.uninow.com
recruiting-develop.uninow.com
recruiting.uninow.com
schedule.uninow.com
scraping.uninow.com
search.answers.uninow.com
sport.uninow.com
staging.admin.checkin.uninow.com
staging.api.checkin.uninow.com
staging.api.facility.uninow.com
staging.checkin.uninow.com
staging.cooperation.uninow.com
staging.feed.uninow.com
staging-gql.uninow.com
staging-graphql.uninow.com
staging.meeting.uninow.com
staging.qrc.uninow.com
staging.recruiting.uninow.com
staging.schedule.uninow.com
staging.scraping.uninow.com
staging.sport.uninow.com
staging.support.uninow.com
staging.uninow.com

staging.universities.uninow.com
staging-wapi.uninow.com
statistics.cooperation.uninow.com
statistics.uninow.com
status.uninow.com
stundenplan.uninow.com
support.uninow.com
testing-gql.uninow.com
testing.scraping.uninow.com
testing.uninow.com
wapi.uninow.com
web.api.uninow.com
www.business.uninow.com
www.checkin.uninow.com
www.cooperation-develop.uninow.com
www.cooperation.uninow.com
www.kooperation.uninow.com
www.meeting.uninow.com
www.recruiting-dashboard.uninow.com
www.recruiting-develop.uninow.com
www.recruiting.uninow.com
www.schedule.uninow.com
www.sport.uninow.com
www.stundenplan.uninow.com
www.support.uninow.com
www.uninow.com

A.2. Subdomains of uninow.de

admin.checkin.uninow.de
analytics.uninow.de
api.uninow.de
app.uninow.de
business.uninow.de
capacity.checkin.uninow.de
chat.uninow.de
checkin.uninow.de
cooperation-develop.uninow.de
cooperation.uninow.de
c.uninow.de
feed.uninow.de
game-legacy.uninow.de
game.uninow.de
kooperation.uninow.de
meeting.uninow.de
npe.uninow.de
recruiting-dashboard.uninow.de
recruiting.uninow.de
schedule-develop.uninow.de
schedule.uninow.de
sport.uninow.de
staging.feed.uninow.de
stundenplan.uninow.de
support.uninow.de
www.business.uninow.de
www.checkin.uninow.de
www.cooperation-develop.uninow.de
www.cooperation.uninow.de
www.feed.uninow.de
www.kooperation.uninow.de
www.meeting.uninow.de
www.npe.uninow.de
www.recruiting-dashboard.uninow.de
www.recruiting.uninow.de
www.schedule.uninow.de
www.stundenplan.uninow.de
www.support.uninow.de
www.uninow.de

A.3. Subdomains of uninow.io

accounts.uninow.io
alpha.uninow.io
analytics-mirror.uninow.io
analytics.uninow.io
api.sales.uninow.io
api.uninow.io
argocd.uninow.io
auth.uninow.io
charlie.uninow.io
ciam.uninow.io
docs-account-api.uninow.io
engage-api.uninow.io
errors.uninow.io
feature-flags.uninow.io
features.uninow.io
function.uninow.io
gaming.uninow.io
google-chat-webhooks.uninow.io
gql.uninow.io
hasura-sales.uninow.io
hasura-sport.uninow.io
ics.uninow.io

internal.uninow.io
k8s-echo.uninow.io
k8s-recruiting-dashboard.uninow.io
k8s-recruiting.uninow.io
k8s-schedule.uninow.io
k8s.uninow.io
k8s-www.uninow.io
mcp.uninow.io
mike.uninow.io
mimir.uninow.io
m.uninow.io
november.uninow.io
npe-backend-k8s.uninow.io
npe-backend.uninow.io
npe-mongodb.uninow.io
npe.uninow.io
odin.uninow.io
poseidon.uninow.io
recruiting-dashboard.uninow.io
sales.uninow.io
staging.accounts.uninow.io
staging.answers.uninow.io
staging.api.answers.uninow.io
staging.api.sales-new.uninow.io
staging.argocd.uninow.io
staging.events.answers.uninow.io
staging.hasura-sport.uninow.io
staging.k8s-echo.uninow.io
staging.k8s.uninow.io
staging.sales-new.uninow.io
staging.sales.uninow.io
staging.support.uninow.io
status.uninow.io
support.uninow.io
test.m.uninow.io
thor.uninow.io
vpn.uninow.io
wapi.uninow.io
web-ui.storybook.uninow.io
wh.uninow.io
www.recruiting-dashboard.uninow.io
www.support.uninow.io

B. Port Scanning Results

B.1. IP 45.129.181.34 (Ports: 80, 443)

accounts.uninow.com
account.uninow.com
admin.checkin.uninow.com
admin.checkin.uninow.de
admin.open.uninow.com
analytics-mirror.uninow.io
analytics.uninow.com
analytics.uninow.de
answers.uninow.com
api.answers.uninow.com
api.assets.uninow.com
api.checkin.uninow.com
api.facility.uninow.com
api-legacy.uninow.com
api.sales.uninow.io
api.uninow.com
api.uninow.de
api.user-feedback.uninow.com
app.search.uninow.com
app.uninow.de
argocd.uninow.io
assets.uninow.com
auth-legacy.uninow.com
auth.uninow.com
business.uninow.com
business.uninow.de
capacity.checkin.uninow.com
capacity.checkin.uninow.de
chat.uninow.de
checkin.uninow.com
checkin.uninow.de
ciam.uninow.io
cooperation-develop.uninow.com
cooperation-develop.uninow.de
cooperation.uninow.com
cooperation.uninow.de
c.uninow.com
c.uninow.de
docs-account-api.uninow.io
engage-api.uninow.io
feature-flags.uninow.io
feed.uninow.com
feed.uninow.de
function.uninow.io
game-legacy.uninow.de
games.uninow.com
game.uninow.de
google-chat-webhooks.uninow.io
gql-legacy.uninow.com
gql.uninow.com
graphql.uninow.com
hasura-sales.uninow.io
hasura-sport.uninow.io
ics.uninow.io
internal.uninow.io
k8s-echo.uninow.io
k8s-recruiting-dashboard.uninow.io
k8s-recruiting.uninow.io
k8s-schedule.uninow.io
k8s.uninow.io
k8s-www.uninow.io
kooperation.uninow.com
kooperation.uninow.de
log.uninow.com
mcp.uninow.io
meeting.uninow.com
meeting.uninow.de
m.uninow.io
npe-backend-k8s.uninow.io
npe-backend.uninow.io
npe-mongodb.uninow.io
npe.uninow.de
npe.uninow.io
open.uninow.com
purchase.uninow.com
qrc.uninow.com
recruiting-dashboard.uninow.com
recruiting-dashboard.uninow.de
recruiting-dashboard.uninow.io
recruiting-develop.uninow.com
recruiting.uninow.com
recruiting.uninow.de
sales.uninow.io
schedule-develop.uninow.de
schedule.uninow.com
schedule.uninow.de
scraping.uninow.com
search.answers.uninow.com
sport.uninow.com
sport.uninow.de
staging.cooperation.uninow.com
staging.feed.uninow.com
staging.feed.uninow.de
staging-graphql.uninow.com
staging.hasura-sport.uninow.io
staging.meeting.uninow.com

staging.recruiting.uninow.com
staging.sales.uninow.io
staging.schedule.uninow.com
staging.scraping.uninow.com
staging.sport.uninow.com
staging.support.uninow.com
staging.support.uninow.io
statistics.cooperation.uninow.com
statistics.uninow.com
status.uninow.com
status.uninow.io
stundenplan.uninow.com
stundenplan.uninow.de
support.uninow.com
support.uninow.de
support.uninow.io
testing.scraping.uninow.com
test.m.uninow.io
uninow.de
uninow.io
wapi.uninow.com
web.api.uninow.com
web-ui.storybook.uninow.io
wh.uninow.io
www.business.uninow.com
www.business.uninow.de
www.checkin.uninow.com
www.checkin.uninow.de
www.cooperation-develop.uninow.com
www.cooperation-develop.uninow.de
www.cooperation.uninow.com
www.cooperation.uninow.de
www.feed.uninow.de
www.kooperation.uninow.com
www.kooperation.uninow.de
www.meeting.uninow.com
www.meeting.uninow.de
www.npe.uninow.de
www.recruiting-dashboard.uninow.com
www.recruiting-dashboard.uninow.de
www.recruiting-dashboard.uninow.io
www.recruiting-develop.uninow.com
www.recruiting.uninow.com
www.recruiting.uninow.de
www.schedule.uninow.com
www.schedule.uninow.de
www.sport.uninow.com
www.stundenplan.uninow.com
www.stundenplan.uninow.de
www.support.uninow.com
www.support.uninow.de
www.support.uninow.io
www.uninow.com
www.uninow.de

B.2. IP 45.129.180.174 (Ports: 80, 443)

accounts.uninow.com
account.uninow.com
admin.checkin.uninow.com
admin.checkin.uninow.de
admin.open.uninow.com
analytics-mirror.uninow.io
analytics.uninow.com
analytics.uninow.de
answers.uninow.com
api.answers.uninow.com
api.assets.uninow.com
api.checkin.uninow.com
api.facility.uninow.com
api-legacy.uninow.com
api.sales.uninow.io
api.uninow.com
api.uninow.de
api.user-feedback.uninow.com
app.search.uninow.com
app.uninow.de
argocd.uninow.io
assets.uninow.com
auth-legacy.uninow.com
auth.uninow.com
business.uninow.com
business.uninow.de
capacity.checkin.uninow.com
capacity.checkin.uninow.de
charlie.uninow.io
chat.uninow.de
checkin.uninow.com
checkin.uninow.de
ciam.uninow.io
cooperation-develop.uninow.com
cooperation-develop.uninow.de
cooperation.uninow.com
cooperation.uninow.de
c.uninow.com
c.uninow.de
docs-account-api.uninow.io
engage-api.uninow.io
feature-flags.uninow.io

feed.uninow.com
feed.uninow.de
function.uninow.io
game-legacy.uninow.de
games.uninow.com
game.uninow.de
google-chat-webhooks.uninow.io
gql-legacy.uninow.com
gql.uninow.com
graphql.uninow.com
hasura-sales.uninow.io
hasura-sport.uninow.io
ics.uninow.io
internal.uninow.io
k8s-echo.uninow.io
k8s-recruiting-dashboard.uninow.io
k8s-recruiting.uninow.io
k8s-schedule.uninow.io
k8s.uninow.io
k8s-www.uninow.io
kooperation.uninow.com
kooperation.uninow.de
log.uninow.com
mcp.uninow.io
meeting.uninow.com
meeting.uninow.de
m.uninow.io
npe-backend-k8s.uninow.io
npe-backend.uninow.io
npe-mongodb.uninow.io
npe.uninow.de
npe.uninow.io
open.uninow.com
purchase.uninow.com
qrc.uninow.com
recruiting-dashboard.uninow.com
recruiting-dashboard.uninow.de
recruiting-dashboard.uninow.io
recruiting-develop.uninow.com
recruiting.uninow.com
recruiting.uninow.de
sales.uninow.io
schedule-develop.uninow.de
schedule.uninow.com
schedule.uninow.de
scraping.uninow.com
search.answers.uninow.com
sport.uninow.com
sport.uninow.de
staging.cooperation.uninow.com
staging.feed.uninow.com
staging.feed.uninow.de
staging-graphql.uninow.com
staging.hasura-sport.uninow.io
staging.meeting.uninow.com
staging.recruiting.uninow.com
staging.sales.uninow.io
staging.schedule.uninow.com
staging.scraping.uninow.com
staging.sport.uninow.com
staging.support.uninow.com
staging.support.uninow.io
statistics.cooperation.uninow.com
statistics.uninow.com
status.uninow.com
status.uninow.io
stundenplan.uninow.com
stundenplan.uninow.de
support.uninow.com
support.uninow.de
support.uninow.io
testing.scraping.uninow.com
test.m.uninow.io
uninow.de
uninow.io
wapi.uninow.com
web.api.uninow.com
web-ui.storybook.uninow.io
wh.uninow.io
www.business.uninow.com
www.business.uninow.de
www.checkin.uninow.com
www.checkin.uninow.de
www.cooperation-develop.uninow.com
www.cooperation-develop.uninow.de
www.cooperation.uninow.com
www.cooperation.uninow.de
www.feed.uninow.de
www.kooperation.uninow.com
www.kooperation.uninow.de
www.meeting.uninow.com
www.meeting.uninow.de
www.npe.uninow.de
www.recruiting-dashboard.uninow.com
www.recruiting-dashboard.uninow.de
www.recruiting-dashboard.uninow.io
www.recruiting-develop.uninow.com
www.recruiting.uninow.com
www.recruiting.uninow.de
www.schedule.uninow.com
www.schedule.uninow.de
www.sport.uninow.com
www.stundenplan.uninow.com

www.stundenplan.uninow.de
www.support.uninow.com
www.support.uninow.de
www.support.uninow.io
www.uninow.com
www.uninow.de

B.3. IP 193.31.27.35 (Ports: 80, 443)

accounts.uninow.io
features.uninow.com
features.uninow.io
mimir.uninow.io
staging-gql.uninow.com
staging-wapi.uninow.com
testing-gql.uninow.com
testing.uninow.com

B.4. IP 195.128.101.234 (Ports: 80, 443)

auth.uninow.io
errors.uninow.io
gaming.uninow.io
gql.uninow.io
odin.uninow.io
wapi.uninow.io

B.5. IP 45.129.180.83 (Ports: 80, 443)

mike.uninow.io
vpn.uninow.io

B.6. IP 104.21.44.46 (Ports: 80, 443, 2052, 2053, 2082, 2083, 2086, 2087, 2095, 2096, 8080, 8443, 8880)

images.uninow.com

B.7. IP 172.67.194.211 (Ports: 80, 443, 2052, 2053, 2082, 2083, 2086, 2087, 2095, 2096, 8080, 8443, 8880)

images.uninow.com

B.8. IP 142.250.185.211 (Ports: 80, 443)

mail.uninow.com

C. Found Web Applications

Services with a description of "-" have an unknown purpose.

Service	Description
`https://accounts.uninow.com`	Account management requiring an email and password account
`http://accounts.uninow.com`	Redirects to the HTTPS version
`https://auth.uninow.io`	*Sentry* administration
`http://auth.uninow.io`	-
`https://admin.checkin.uninow.com`	Administration app for the Campus Check-In feature
`http://admin.checkin.uninow.com`	Redirects to the HTTPS version
`https://admin.checkin.uninow.de`	Administration app for the Campus Check-In feature
`http://admin.checkin.uninow.de`	Redirects to the HTTPS version
`https://admin.open.uninow.com`	*Shlink* administration app
`http://admin.open.uninow.com`	Redirects to the HTTPS version
`https://analytics-mirror.uninow.io`	Unknown purpose; accepts requests for path `/mirror`
`http://analytics-mirror.uninow.io`	Redirects to the HTTPS version
`https://analytics.uninow.com`	-
`http://analytics.uninow.com`	-
`https://analytics.uninow.de`	-
`http://analytics.uninow.de`	-
`https://answers.uninow.com`	-
`http://answers.uninow.com`	Redirects to the HTTPS version
`https://api.answers.uninow.com`	-
`http://api.answers.uninow.com`	Redirects to the HTTPS version
`https://api.assets.uninow.com`	-
`http://api.assets.uninow.com`	Redirects to the HTTPS version
`https://api.checkin.uninow.com`	*Hasura* GraphQL API for the Campus Check-In feature
`http://api.checkin.uninow.com`	Redirects to the HTTPS version
`https://api.facility.uninow.com`	-
`http://api.facility.uninow.com`	Redirects to the HTTPS version
`https://api-legacy.uninow.com`	-
`http://api-legacy.uninow.com`	-
`https://api.sales.uninow.io`	-
`http://api.sales.uninow.io`	Redirects to the HTTPS version
`https://api.uninow.com`	-
`http://api.uninow.com`	-
`https://api.uninow.de`	-
`http://api.uninow.de`	-
`https://api.user-feedback.uninow.com`	-
`http://api.user-feedback.uninow.com`	Redirects to the HTTPS version
`https://app.search.uninow.com`	API used to search universities, jobs, and companies
`http://app.search.uninow.com`	Redirects to the HTTPS version
`https://app.uninow.de`	Same as `https://uninow.de`
`http://app.uninow.de`	Redirects to the HTTPS version

https://argocd.uninow.io	-
http://argocd.uninow.io	Redirects to the HTTPS version
https://assets.uninow.com	-
http://assets.uninow.com	Redirects to the HTTPS version
https://auth-legacy.uninow.com	-
http://auth-legacy.uninow.com	-
https://auth.uninow.com	-
http://auth.uninow.com	-
https://business.uninow.com	Management web application for companies
http://business.uninow.com	Redirects to the HTTPS version
https://business.uninow.de	Same as https://business.uninow.com
http://business.uninow.de	Redirects to the HTTPS version
https://capacity.checkin.uninow.com	-
http://capacity.checkin.uninow.com	-
https://capacity.checkin.uninow.de	Web version of the campus check-in feature
http://capacity.checkin.uninow.de	Redirects to the HTTPS version
https://chat.uninow.de	-
http://chat.uninow.de	-
https://checkin.uninow.com	Web version of the campus check-in feature
http://checkin.uninow.com	Redirects to the HTTPS version
https://checkin.uninow.de	Same as https://checkin.uninow.com
http://checkin.uninow.de	Redirects to the HTTPS version
https://ciam.uninow.io	-
http://ciam.uninow.io	Redirects to the HTTPS version
https://cooperation-develop.uninow.com	-
http://cooperation-develop.uninow.com	-
https://cooperation-develop.uninow.de	-
http://cooperation-develop.uninow.de	-
https://cooperation.uninow.com	Some internal service protected by a login page
http://cooperation.uninow.com	Redirects to the HTTPS version
https://cooperation.uninow.de	-
http://cooperation.uninow.de	-
https://c.uninow.com	Web version of the campus check-in feature
http://c.uninow.com	Redirects to the HTTPS version
https://c.uninow.de	
http://c.uninow.de	Redirects to the HTTPS version
https://docs-account-api.uninow.io	-
http://docs-account-api.uninow.io	-
https://engage-api.uninow.io	-
http://engage-api.uninow.io	-
https://errors.uninow.io	Same as https://auth.uninow.io; used by the mobile application to log errors
http://errors.uninow.io	Redirects to the HTTPS version
https://feature-flags.uninow.io	-
http://feature-flags.uninow.io	-
https://feed.uninow.com	Web application for university and student council accounts

http://feed.uninow.com	Redirects to the HTTPS version
https://feed.uninow.de	Same as https://feed.uninow.com
http://feed.uninow.de	Redirects to the HTTPS version
https://function.uninow.io	-
http://function.uninow.io	-
https://game-legacy.uninow.de	-
http://game-legacy.uninow.de	-
https://gaming.uninow.io	Same as https://auth.uninow.io
http://gaming.uninow.io	-
https://games.uninow.com	-
http://games.uninow.com	Redirects to the HTTPS version
https://game.uninow.de	-
http://game.uninow.de	-
https://google-chat-webhooks.uninow.io	-
http://google-chat-webhooks.uninow.io	-
https://gql-legacy.uninow.com	-
http://gql-legacy.uninow.com	-
https://gql.uninow.com	-
http://gql.uninow.com	-
https://gql.uninow.io	Same as https://auth.uninow.io
http://gql.uninow.io	-
https://graphql.uninow.com	GraphQL API with many account related purposes
http://graphql.uninow.com	Redirects to the HTTPS version
https://hasura-sales.uninow.io	-
http://hasura-sales.uninow.io	Redirects to the HTTPS version
https://hasura-sport.uninow.io	*Hasura* GraphQL API for the sport module
http://hasura-sport.uninow.io	Redirects to the HTTPS version
https://ics.uninow.io	-
http://ics.uninow.io	Redirects to the HTTPS version
https://images.uninow.de	*S3* bucket used to provide static content to multiple applications
http://images.uninow.de	Redirects to the HTTPS version
https://internal.uninow.io	-
http://internal.uninow.io	Redirects to the HTTPS version
https://k8s-echo.uninow.io	-
http://k8s-echo.uninow.io	Redirects to the HTTPS version
https://k8s-recruiting-dashboard.uninow.io	-
http://k8s-recruiting-dashboard.uninow.io	-
https://k8s-recruiting.uninow.io	-
http://k8s-recruiting.uninow.io	-
https://k8s-schedule.uninow.io	-
http://k8s-schedule.uninow.io	-
https://k8s.uninow.io	-
http://k8s.uninow.io	-
https://k8s-www.uninow.io	-
http://k8s-www.uninow.io	-

https://kooperation.uninow.com	-
http://kooperation.uninow.com	-
https://kooperation.uninow.de	-
http://kooperation.uninow.de	-
https://log.uninow.com	-
http://log.uninow.com	Redirects to the HTTPS version
https://mail.uninow.com	*Gmail* login page
http://mail.uninow.com	Redirects to the HTTPS version
https://mcp.uninow.io	*Metabase* [99] instance
http://mcp.uninow.io	Redirects to the HTTPS version
https://meeting.uninow.com	Opened instead of the application when *UniNow* mobile application is not installed; shows download instructions
http://meeting.uninow.com	Redirects to the HTTPS version
https://meeting.uninow.de	Same as https://metting.uninow.com
http://meeting.uninow.de	Redirects to the HTTPS version
https://mike.uninow.io	-
http://mike.uninow.io	-
https://m.uninow.io	-
http://m.uninow.io	-
https://npe-backend-k8s.uninow.io	-
http://npe-backend-k8s.uninow.io	-
https://npe-backend.uninow.io	The API used by https://npe.uninow.io
http://npe-backend.uninow.io	Redirects to the HTTPS version
https://npe-mongodb.uninow.io	-
http://npe-mongodb.uninow.io	-
https://npe.uninow.de	-
http://npe.uninow.de	-
https://npe.uninow.io	Internal service to manage university configurations, view university status, and access error messages
http://npe.uninow.io	Redirects to the HTTPS version
https://odin.uninow.io	Sames as https://auth.uninow.io
http://odin.uninow.io	-
https://open.uninow.com	Might be the *Shlink* instance managed by https://admin.open.uninow.com
http://open.uninow.com	Redirects to the HTTPS version
https://purchase.uninow.com	-
http://purchase.uninow.com	Redirects to the HTTPS version
https://qrc.uninow.com	Used to create PDFs with QR codes for the Campus Check-In feature
http://qrc.uninow.com	Redirects to the HTTPS version
https://recruiting-dashboard.uninow.com	-
http://recruiting-dashboard.uninow.com	-
https://recruiting-dashboard.uninow.de	-
http://recruiting-dashboard.uninow.de	-

https://recruiting-dashboard.uninow.io	-
http://recruiting-dashboard.uninow.io	Redirects to the HTTPS version
https://recruiting-develop.uninow.com	-
http://recruiting-develop.uninow.com	-
https://recruiting.uninow.com	Same as https://business.uninow.com
http://recruiting.uninow.com	Redirects to the HTTPS version
https://recruiting.uninow.de	Same as https://business.uninow.com
http://recruiting.uninow.de	Redirects to the HTTPS version
https://sales.uninow.io	-
http://sales.uninow.io	-
https://schedule-develop.uninow.de	-
http://schedule-develop.uninow.de	-
https://schedule.uninow.com	Web version of the calendar module
http://schedule.uninow.com	Redirects to the HTTPS version
https://schedule.uninow.de	Same as https://schedule.uninow.com
http://schedule.uninow.de	Redirects to the HTTPS version
https://scraping.uninow.com	The scraping service
http://scraping.uninow.com	Redirects to the HTTPS version
https://search.answers.uninow.com	-
http://search.answers.uninow.com	-
https://sport.uninow.com	Web app presumably for administering the sports module
http://sport.uninow.com	Redirects to the HTTPS version
https://sport.uninow.de	-
http://sport.uninow.de	-
https://staging.cooperation.uninow.com	-
http://staging.cooperation.uninow.com	Redirects to the HTTPS version
https://staging.feed.uninow.com	-
http://staging.feed.uninow.com	-
https://staging.feed.uninow.de	-
http://staging.feed.uninow.de	-
https://staging-graphql.uninow.com	-
http://staging-graphql.uninow.com	-
https://staging.hasura-sport.uninow.io	-
http://staging.hasura-sport.uninow.io	-
https://staging.meeting.uninow.com	-
http://staging.meeting.uninow.com	-
https://staging.recruiting.uninow.com	-
http://staging.recruiting.uninow.com	-
https://staging.sales.uninow.io	-
http://staging.sales.uninow.io	-
https://staging.schedule.uninow.com	-
http://staging.schedule.uninow.com	-
https://staging.scraping.uninow.com	-
http://staging.scraping.uninow.com	Redirects to the HTTPS version
https://staging.sport.uninow.com	-

http://staging.sport.uninow.com	-
https://staging.support.uninow.com	-
http://staging.support.uninow.com	-
https://staging.support.uninow.io	-
http://staging.support.uninow.io	-
https://statistics.cooperation.uninow.com	Login page for some internal service
http://statistics.cooperation.uninow.com	Redirects to the HTTPS version
https://statistics.uninow.com	
http://statistics.uninow.com	Redirects to the HTTPS version
https://status.uninow.com	-
http://status.uninow.com	Redirects to the HTTPS version
https://status.uninow.io	-
http://status.uninow.io	-
https://stundenplan.uninow.com	Redirects to https://schedule.uninow.com
http://stundenplan.uninow.com	Redirects to the HTTPS version
https://stundenplan.uninow.de	Redirects to https://schedule.uninow.com
http://stundenplan.uninow.de	Redirects to the HTTPS version
https://support.uninow.com	Web page to share username and password with *UniNow*
http://support.uninow.com	Redirects to the HTTPS version
https://support.uninow.de	Same as https://support.uninow.com
http://support.uninow.de	Redirects to the HTTPS version
https://support.uninow.io	Login page for internal support service
http://support.uninow.io	Redirects to the HTTPS version
https://testing.scraping.uninow.com	-
http://testing.scraping.uninow.com	Redirects to the HTTPS version
https://test.m.uninow.io	-
http://test.m.uninow.io	-
https://uninow.de	Homepage of *UniNow*
http://uninow.de	Redirects to the HTTPS version
https://uninow.io	-
http://uninow.io	-
https://vpn.uninow.io	-
http://vpn.uninow.io	Redirects to the HTTPS version
https://wapi.uninow.com	API used by many web applications
http://wapi.uninow.com	Redirects to the HTTPS version
https://wapi.uninow.io	Same as https://auth.uninow.io
http://wapi.uninow.io	-
https://web.api.uninow.com	-
http://web.api.uninow.com	-
https://web-ui.storybook.uninow.io	-
http://web-ui.storybook.uninow.io	Redirects to the HTTPS version
https://wh.uninow.io	-
http://wh.uninow.io	-
https://www.business.uninow.com	Same as https://business.uninow.com
http://www.business.uninow.com	Redirects to the HTTPS version

https://www.business.uninow.de	Same as https://business.uninow.com
http://www.business.uninow.de	Redirects to the HTTPS version
https://www.checkin.uninow.com	Same as https://checkin.uninow.com
http://www.checkin.uninow.com	Redirects to the HTTPS version
https://www.checkin.uninow.de	Same as https://checkin.uninow.com
http://www.checkin.uninow.de	Redirects to the HTTPS version
https://www.cooperation-develop.uninow.com	-
http://www.cooperation-develop.uninow.com	-
https://www.cooperation-develop.uninow.de	-
http://www.cooperation-develop.uninow.de	-
https://www.cooperation.uninow.com	-
http://www.cooperation.uninow.com	-
https://www.cooperation.uninow.de	-
http://www.cooperation.uninow.de	-
https://www.feed.uninow.de	Same as https://feed.uninow.de
http://www.feed.uninow.de	Redirects to the HTTPS version
https://www.kooperation.uninow.com	-
http://www.kooperation.uninow.com	-
https://www.kooperation.uninow.de	-
http://www.kooperation.uninow.de	-
https://www.meeting.uninow.com	-
http://www.meeting.uninow.com	-
https://www.meeting.uninow.de	-
http://www.meeting.uninow.de	-
https://www.npe.uninow.de	-
http://www.npe.uninow.de	-
https://www.recruiting-dashboard.uninow.com	-
http://www.recruiting-dashboard.uninow.com	-
https://www.recruiting-dashboard.uninow.de	-
http://www.recruiting-dashboard.uninow.de	-
https://www.recruiting-dashboard.uninow.io	-
http://www.recruiting-dashboard.uninow.io	-
https://www.recruiting-develop.uninow.com	-
http://www.recruiting-develop.uninow.com	-
https://www.recruiting.uninow.com	Same as https://recruiting.uninow.com
http://www.recruiting.uninow.com	Redirects to the HTTPS version
https://www.recruiting.uninow.de	Same as https://recruiting.uninow.com
http://www.recruiting.uninow.de	Redirects to the HTTPS version

https://www.schedule.uninow.com	Redirects to https://schedule.uninow.com
http://www.schedule.uninow.com	Redirects to the HTTPS version
https://www.schedule.uninow.de	Redirects to https://schedule.uninow.com
http://www.schedule.uninow.de	Redirects to the HTTPS version
https://www.sport.uninow.com	Same as https://sport.uninow.com
http://www.sport.uninow.com	Redirects to the HTTPS version
https://www.stundenplan.uninow.com	Redirects to https://schedule.uninow.com
http://www.stundenplan.uninow.com	Redirects to the HTTPS version
https://www.stundenplan.uninow.de	Redirects to https://schedule.uninow.com
http://www.stundenplan.uninow.de	Redirects to the HTTPS version
https://www.support.uninow.com	Same as https://www.support.uninow.com
http://www.support.uninow.com	Redirects to the HTTPS version
https://www.support.uninow.de	Same as https://www.support.uninow.com
http://www.support.uninow.de	Redirects to the HTTPS version
https://www.support.uninow.io	-
http://www.support.uninow.io	Redirects to the HTTPS version
https://www.uninow.com	Same as https://uninow.com
http://www.uninow.com	Redirects to the HTTPS version
https://www.uninow.de	Same as https://uninow.com
http://www.uninow.de	Redirects to the HTTPS version

D. Appointment Invitation GraphQL Query

```
query ($id: String!) {
  meeting(id: $id) {
    id
    title
    type
    color

    blocks {
      location
      allDay
      duration
      notes
      repeat
      __typename
    }

    slots {
      _id
      startTime
      date
      __typename
    }

    members {
      memberID
      name
      status
      slots
      __typename
    }

    host {
      name
      __typename
    }

    chosenSlot {
      startTime
      date
      __typename
    }

    status
    ownStatus
    hasTime

    links {
      forSend
      __typename
    }
```

```
    __typename
  }
}
```

E. Email Server Configurations with *checkCertificate* not *null*

University	Hostname	Port	Type	Connection Type	checkCertificate
Alanus Hochschule für Kunst und Gesellschaft	mail2.alanus.edu	587	outgoing	STARTTLS	false
	mail2.alanus.edu	993	ingoing	TLS	false
Bauhaus-Universität Weimar	mailgate.uni-weimar.de	143	ingoing	CLEAR	false
	mailgate.uni-weimar.de	143	ingoing	SSL	false
	mailgate.uni-weimar.de	25	outgoing	CLEAR	false
	mailgate.uni-weimar.de	25	outgoing	TLS	false
	mailgate.uni-weimar.de	465	outgoing	TLS	false
	mailgate.uni-weimar.de	587	outgoing	STARTTLS	false
	mailgate.uni-weimar.de	993	ingoing	SSL	false
Europa-Universität Viadrina Frankfurt (Oder)	owa.europa-uni.de	25	outgoing	SSL	false
	owa.europa-uni.de	25	outgoing	TLS	false
	owa.europa-uni.de	465	outgoing	SSL	false
	owa.europa-uni.de	587	outgoing	STARTTLS	false
	owa.europa-uni.de	993	ingoing	STARTTLS	false
	owa.europa-uni.de	993	ingoing	TLS	false
Evangelische Hochschule für Kirchenmusik Halle	mail.uni-halle.de	993	ingoing	TLS	false
	smtpauth.uni-halle.de	587	outgoing	STARTTLS	false
Frankfurt University of Applied Sciences	mail.frankfurt-university.de	143	ingoing	SSL	false
	mail.frankfurt-university.de	143	ingoing	TLS	false
	mail.frankfurt-university.de	25	outgoing	SSL	false
	mail.frankfurt-university.de	25	outgoing	TLS	false
	mail.frankfurt-university.de	465	outgoing	SSL	false
	mail.frankfurt-university.de	465	outgoing	TLS	false
	mail.frankfurt-university.de	993	ingoing	SSL	false
	mail.frankfurt-university.de	993	ingoing	TLS	false
Hochschule Aalen - Technik und Wirtschaft	studmail.htw-aalen.de	587	outgoing	STARTTLS	false
	studmail.htw-aalen.de	993	ingoing	TLS	false
Hochschule Bonn-Rhein-Sieg	imap.inf.h-brs.de	993	ingoing	TLS	false
	owa.stud.h-brs.de	587	outgoing	STARTTLS	false
	owa.stud.h-brs.de	993	ingoing	TLS	false
	smtp.inf.h-brs.de	465	outgoing	TLS	false
Hochschule Fulda	mail.hs-fulda.de	143	ingoing	STARTTLS	false
	mail.hs-fulda.de	993	ingoing	SSL	false
	mail.hs-fulda.de	993	ingoing	TLS	false
	smtp.hs-fulda.de	587	outgoing	STARTTLS	false
	smtp.hs-fulda.de	587	outgoing	TLS	false
Hochschule für Angewandte Wissenschaften Hamburg	haw-mailer.haw-hamburg.de	587	outgoing	STARTTLS	false
	haw-mailer.haw-hamburg.de	993	ingoing	TLS	false
Hochschule für angewandte Wissenschaften Kempten	mail-s.hs-kempten.de	143	ingoing	STARTTLS	false
	mail-s.hs-kempten.de	587	outgoing	STARTTLS	false
Hochschule Ludwigshafen am Rhein	imaps.hwg-lu.de	143	ingoing	STARTTLS	false
	imaps.hwg-lu.de	143	ingoing	STARTTLS	true

Hochschule Mannheim	mail.hs-mannheim.de	465	outgoing	TLS	false
	mail.hs-mannheim.de	993	ingoing	TLS	false
	stud.hs-mannheim.de	465	outgoing	TLS	false
	stud.hs-mannheim.de	993	ingoing	TLS	false
Hochschule Nordhausen	imap-mail.outlook.com	993	ingoing	TLS	false
	smtp-mail.outlook.com	587	outgoing	STARTTLS	false
	xmail.hs-nordhausen.de	25	outgoing	STARTTLS	false
	xmail.hs-nordhausen.de	993	ingoing	TLS	false
Hochschule Offenburg	imap.hs-offenburg.de	993	ingoing	SSL	false
	imap.hs-offenburg.de	993	ingoing	TLS	false
	smtp.hs-offenburg.de	465	outgoing	TLS	false
	smtp.hs-offenburg.de	587	outgoing	STARTTLS	false
Hochschule RheinMain	mail.hs-rm.de	143	ingoing	STARTTLS	false
	mail.hs-rm.de	465	outgoing	SSL	false
	mail.hs-rm.de	465	outgoing	TLS	false
	mail.hs-rm.de	587	outgoing	STARTTLS	false
	mail.hs-rm.de	587	outgoing	STARTTLS	true
	mail.hs-rm.de	993	ingoing	SSL	false
	mail.hs-rm.de	993	ingoing	TLS	false
	mail.student.hs-rm.de	25	outgoing	TLS	false
	mail.student.hs-rm.de	993	ingoing	TLS	false
Hochschule Rhein-Waal	mail-kam.hsrw.org	587	outgoing	TLS	false
	mail-kam.hsrw.org	993	ingoing	TLS	false
	mail-kle.hsrw.org	587	outgoing	TLS	false
	mail-kle.hsrw.org	993	ingoing	TLS	false
Hochschule Zittau/Görlitz	mail.hszg.de	993	ingoing	TLS	false
	smtp.hszg.de	587	outgoing	STARTTLS	false
Johannes Gutenberg-Universität Mainz	mail.uni-mainz.de	587	outgoing	STARTTLS	false
	mail.uni-mainz.de	993	ingoing	SSL	false
	mail.uni-mainz.de	993	ingoing	TLS	false
Justus-Liebig-Universität Gießen	imap.stud.uni-giessen.de	143	ingoing	STARTTLS	false
	imap.stud.uni-giessen.de	143	ingoing	TLS	false
	imap.stud.uni-giessen.de	993	ingoing	TLS	false
Katholische Hochschule Mainz	mail.students.kh-mz.de	465	outgoing	SSL	false
	mail.students.kh-mz.de	465	outgoing	STARTTLS	false
	mail.students.kh-mz.de	993	ingoing	SSL	false
	mail.students.kh-mz.de	993	ingoing	STARTTLS	false
	mail.students.kh-mz.de	993	ingoing	TLS	false
Martin-Luther-Universität Halle-Wittenberg	mail.uni-halle.de	143	ingoing	STARTTLS	false
	mail.uni-halle.de	993	ingoing	STARTTLS	false
	mail.uni-halle.de	993	ingoing	TLS	false
	smtpauth.uni-halle.de	587	outgoing	STARTTLS	false
	smtpauth.uni-halle.de	587	outgoing	TLS	false
Medizinische Universität Innsbruck	mail.i-med.ac.at	993	ingoing	TLS	false
Pädagogische Hochschule Karlsruhe	imap.ph-karlsruhe.de	993	ingoing	TLS	false
Pädagogische Hochschule Schwäbisch Gmünd	imap.ph-gmuend.de	993	ingoing	TLS	false

	smtp.ph-gmuend.de	465	outgoing	SSL	false
	smtp.ph-gmuend.de	465	outgoing	STARTTLS	false
	smtp.ph-gmuend.de	465	outgoing	TLS	false
Pädagogische Hochschule Weingarten	imap.ph-bw.de	993	ingoing	TLS	false
	imap.ph-weingarten.de	993	ingoing	TLS	false
	smtp.ph-weingarten.de	25	outgoing	STARTTLS	false
	smtp.ph-weingarten.de	25	outgoing	TLS	false
	smtp.ph-weingarten.de	465	outgoing	STARTTLS	false
	smtp.ph-weingarten.de	465	outgoing	TLS	false
Technische Hochschule Bingen	mail.zdv.net	587	outgoing	STARTTLS	false
	mail.zdv.net	993	ingoing	SSL	false
	mail.zdv.net	993	ingoing	TLS	false
Technische Hochschule Köln	imap.intranet.fh-koeln.de	993	ingoing	TLS	false
	smtp.intranet.fh-koeln.de	587	outgoing	STARTTLS	false
Technische Hochschule Nürnberg Georg Simon Ohm	my.ohmportal.de	465	outgoing	TLS	false
	my.ohmportal.de	993	ingoing	TLS	false
Technische Universität Darmstadt	imap.stud.tu-darmstadt.de	993	ingoing	TLS	false
	smtp.tu-darmstadt.de	465	outgoing	TLS	false
Technische Universität Dresden	mail.zih.tu-dresden.de	143	ingoing	CLEAR	false
	mail.zih.tu-dresden.de	143	ingoing	STARTTLS	false
	mail.zih.tu-dresden.de	25	outgoing	STARTTLS	false
	mail.zih.tu-dresden.de	465	outgoing	TLS	false
	mail.zih.tu-dresden.de	587	outgoing	TLS	false
	mail.zih.tu-dresden.de	993	ingoing	CLEAR	false
	mail.zih.tu-dresden.de	993	ingoing	TLS	false
	msx.tu-dresden.de	143	ingoing	CLEAR	false
	msx.tu-dresden.de	143	ingoing	STARTTLS	false
	msx.tu-dresden.de	465	outgoing	TLS	false
	msx.tu-dresden.de	587	outgoing	STARTTLS	false
	msx.tu-dresden.de	993	ingoing	CLEAR	false
	msx.tu-dresden.de	993	ingoing	SSL	false
Theologische Hochschule Friedensau	mail.stud.thh-friedensau.de	465	outgoing	STARTTLS	false
	mail.stud.thh-friedensau.de	587	outgoing	STARTTLS	false
Universität Regensburg	imap.uni-regensburg.de	993	ingoing	CLEAR	false
	imap.uni-regensburg.de	993	ingoing	TLS	false
Universität Siegen	mail.uni-siegen.de	143	ingoing	STARTTLS	false
	mail.uni-siegen.de	25	outgoing	STARTTLS	false
	mail.uni-siegen.de	587	outgoing	STARTTLS	false
	mail.uni-siegen.de	993	ingoing	TLS	false
Universität Stuttgart	imap.uni-stuttgart.de	993	ingoing	TLS	false
	smtp.uni-stuttgart.de	587	outgoing	STARTTLS	false
Universität Ulm	imap.uni-ulm.de	587	outgoing	STARTTLS	false
	imap.uni-ulm.de	993	ingoing	TLS	false
Universität Wien	imap.univie.ac.at	993	ingoing	SSL	false
	imap.univie.ac.at	993	ingoing	TLS	false
	mail.univie.ac.at	465	outgoing	SSL	false
	mail.univie.ac.at	465	outgoing	TLS	false

Westfälische Hochschule	mailx.w-hs.de	993	ingoing	TLS	false
	smtpx.w-hs.de	587	outgoing	STARTTLS	false
	studmail.w-hs.de	465	outgoing	TLS	false
	studmail.w-hs.de	587	outgoing	STARTTLS	false
	studmail.w-hs.de	993	ingoing	TLS	false
Westfälische Wilhelms-Universität Münster	imap.uni-muenster.de	993	ingoing	TLS	false
	secmail.uni-muenster.de	587	outgoing	STARTTLS	false
	secmail.uni-muenster.de	587	outgoing	TLS	false

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Würzburg, 29. June 2021

.....
(Keven Zimmermann)