Implementation and Security Evaluation of User-Customized Content in a Mobile Application

Implementering och säkerhetsutvärdering av användarbestämt innehåll i en mobilapplikation

Viktor Wällstedt

Examiner: Niklas Carlsson

External supervisor: Torbjörn Lundquist
Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Viktor Wällstedt
Abstract

Companies offering a service application targeting a broad audience often have difficulties meeting all user requirements since many users have unique needs. Allowing users to define and create content for service applications themselves, which addresses their specific needs, would be a welcomed solution. This would allow developers to focus on the main aspects of the service application, whereas the users themselves can include individual end-user aspects. User-customized content can be used as a selling point for the companies and opens up possibilities for providing a better user experience for each unique end user. This thesis describes the process of creating a prototype system that provides a solution for including user-customized content in a mobile application service system. First, we describe requirement elicitation followed by design and the actual implementation. Furthermore, security is a frequent topic whenever a digital application is discussed today. Therefore, the system creation process is followed up with an investigation of how the resulting application security aspect can be evaluated. After investigating different possibilities, a security evaluation case study on the application is performed. The results show a functioning system that allows customers to customize the content that is rendered inside a cross-platform mobile application. The results from the security evaluation investigation also show that the Open Web Application Security Project (OWASP) Mobile Security Testing Guide (MSTG) framework can be adapted and used for security evaluation of a cross-platform mobile application, even though it targets native applications. The resulting system satisfies most of the requirements for the targeted security level but does not satisfy all requirements for a normal production level mobile application according to the OWASP Mobile Application Security Verification Standard (MASVS). However, the results indicate that there is potential to reach the desired security level by adapting the system to use pure React Native with some native code additions.
Acknowledgments

I want to begin by thanking my supervisor and examiner Niklas Carlsson for taking time to answer all of my questions and guiding me throughout this thesis project. Your help has been much appreciated and a valuable asset.

I would also like to thank everyone at Senion for including me in the company and making me feel at home during my thesis work there. A special thanks to my external supervisors Torbjörn and Tommy for providing guidance and feedback, making the thesis project better and at the same time more enjoyable.

I would also like to thank Lukas Lindqvist for drafting a preparatory document for this thesis together with me during the course Vetenskaplig Metod TDDD89 and for always being available for discussions and insights whenever I got stuck during the thesis work itself.

Viktor Wållstedt
Linköping, June 2019
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>iv</td>
</tr>
<tr>
<td>Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>viii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>ix</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Aim</td>
<td>1</td>
</tr>
<tr>
<td>1.3 Research Questions</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Delimitations</td>
<td>2</td>
</tr>
<tr>
<td><strong>2 Background</strong></td>
<td>3</td>
</tr>
<tr>
<td>2.1 Thesis Environment</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Mobile Application</td>
<td>3</td>
</tr>
<tr>
<td>2.2.1 React Native</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Mobile Application Security</td>
<td>4</td>
</tr>
<tr>
<td>2.3.1 OWASP Mobile Security Testing Guide</td>
<td>5</td>
</tr>
<tr>
<td>2.3.2 Privacy</td>
<td>5</td>
</tr>
<tr>
<td>2.3.3 Access Control</td>
<td>5</td>
</tr>
<tr>
<td>2.3.4 Availability</td>
<td>6</td>
</tr>
<tr>
<td>2.3.5 Vulnerability Detection</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Web Application</td>
<td>7</td>
</tr>
<tr>
<td>2.4.1 React</td>
<td>7</td>
</tr>
<tr>
<td>2.4.2 Redux</td>
<td>7</td>
</tr>
<tr>
<td>2.4.3 Semantic UI</td>
<td>7</td>
</tr>
<tr>
<td>2.5 Web Server</td>
<td>8</td>
</tr>
<tr>
<td>2.5.1 Node</td>
<td>8</td>
</tr>
<tr>
<td>2.5.2 Java</td>
<td>8</td>
</tr>
<tr>
<td>2.5.3 Kotlin</td>
<td>8</td>
</tr>
<tr>
<td>2.6 JavaScript and Security</td>
<td>9</td>
</tr>
<tr>
<td>2.6.1 Changes to Browsers and Standards</td>
<td>9</td>
</tr>
<tr>
<td>2.6.2 Restricting Scripts to a Safe Subset of JavaScript</td>
<td>9</td>
</tr>
<tr>
<td>2.6.3 Code Transformation and Runtime Monitoring Approaches</td>
<td>9</td>
</tr>
<tr>
<td>2.7 Cross-Site Scripting</td>
<td>10</td>
</tr>
<tr>
<td>2.7.1 Stored</td>
<td>10</td>
</tr>
<tr>
<td>2.7.2 Reflective</td>
<td>10</td>
</tr>
<tr>
<td>2.8 Cross-Site Request Forgery</td>
<td>10</td>
</tr>
<tr>
<td>2.9 NoSQL injection</td>
<td>10</td>
</tr>
<tr>
<td>Chapter</td>
<td>Section</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>2.10</td>
<td>CSS Injection</td>
</tr>
<tr>
<td>2.11</td>
<td>Authentication</td>
</tr>
<tr>
<td>2.11.1</td>
<td>Username and Password</td>
</tr>
<tr>
<td>2.11.2</td>
<td>Cookies</td>
</tr>
<tr>
<td>2.11.3</td>
<td>JSON Web Tokens</td>
</tr>
<tr>
<td>2.12</td>
<td>Authorization</td>
</tr>
<tr>
<td>2.13</td>
<td>Encryption</td>
</tr>
<tr>
<td>2.13.1</td>
<td>Symmetric Encryption</td>
</tr>
<tr>
<td>2.13.2</td>
<td>Asymmetric Encryption</td>
</tr>
<tr>
<td>2.14</td>
<td>Hash Functions</td>
</tr>
<tr>
<td>2.15</td>
<td>Digital Signatures</td>
</tr>
<tr>
<td>2.16</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>2.17</td>
<td>Related Work</td>
</tr>
<tr>
<td>3</td>
<td>System Design</td>
</tr>
<tr>
<td>3.1</td>
<td>Requirements Elicitation</td>
</tr>
<tr>
<td>3.2</td>
<td>System Architecture</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Static Content</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Dynamic Content</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Rendering Content</td>
</tr>
<tr>
<td>4</td>
<td>Implementation</td>
</tr>
<tr>
<td>4.1</td>
<td>Mobile Application</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Components</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Authentication</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Mobile Application - Evaluation</td>
</tr>
<tr>
<td>4.2</td>
<td>Web Server</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Routing</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Authentication</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Database</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Web Server - Evaluation</td>
</tr>
<tr>
<td>4.3</td>
<td>User Interface</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Authentication</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Static Configuration</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Dynamic Configuration</td>
</tr>
<tr>
<td>4.3.4</td>
<td>User Interface - Evaluation</td>
</tr>
<tr>
<td>4.4</td>
<td>Customer Server</td>
</tr>
<tr>
<td>4.4.1</td>
<td>User Interface - Further Evaluation</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Updating Content</td>
</tr>
<tr>
<td>4.4.3</td>
<td>External Data</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Customer Server - Evaluation</td>
</tr>
<tr>
<td>5</td>
<td>Case Study</td>
</tr>
<tr>
<td>5.1</td>
<td>Method</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Architecture, Design and Threat Modeling</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Data Storage and Privacy</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Cryptography</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Authentication and Session Management</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Network Communication</td>
</tr>
<tr>
<td>5.1.6</td>
<td>Platform Interaction</td>
</tr>
<tr>
<td>5.1.7</td>
<td>Code Quality and Build Setting</td>
</tr>
<tr>
<td>5.2</td>
<td>Results</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Architecture, Design and Threat Modeling</td>
</tr>
</tbody>
</table>
5.2.2 Data Storage and Privacy ............................................. 35
5.2.3 Cryptography ......................................................... 36
5.2.4 Authentication and Session Management ......................... 37
5.2.5 Network Communication ........................................... 37
5.2.6 Platform Interaction ................................................ 39
5.2.7 Code Quality and Build Setting ................................ 39

6 Discussion ................................................................. 41
6.1 Results ................................................................. 41
6.1.1 System Design ....................................................... 41
6.1.2 Implementation .................................................... 41
6.1.3 Case Study ......................................................... 42
6.2 Method ................................................................. 43
6.2.1 Design ............................................................... 43
6.2.2 Implementation .................................................... 43
6.2.3 Case Study ......................................................... 44
6.3 Future Development .................................................. 44
6.4 The Work in a Wider Context ...................................... 45

7 Conclusion ................................................................. 47

Bibliography ................................................................. 49

A Mobile Application ...................................................... 53
A.1 Notes Scenario ....................................................... 53
A.2 Screens ............................................................... 55
A.3 Building Scenario ................................................... 55
A.4 Links Scenario ....................................................... 56
A.5 Lunch Scenario ....................................................... 56
A.6 News Scenario ....................................................... 57
A.7 Parking Scenario ..................................................... 57
A.8 Smart Office Control Scenario .................................... 58
A.9 Service Desk Scenario ............................................. 58
A.10 Multiple Cards ....................................................... 59
A.11 Updating Content - Input ......................................... 60
A.12 Updating Content - Periodic ..................................... 61

B User Interface ............................................................ 62
B.1 Sign In ............................................................... 62
B.2 Sign Up ............................................................... 63
B.3 Static Content ...................................................... 63
B.4 Dynamic Content ................................................... 64
B.5 Dynamic Content - Example ....................................... 65

C Case Study ................................................................. 68
C.1 Network Traffic ....................................................... 68
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Problem description</td>
<td>2</td>
</tr>
<tr>
<td>3.1</td>
<td>Static configuration architecture</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Dynamic configuration architecture</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>Configuration usage</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td>Content rendered from parsed JSON file</td>
<td>23</td>
</tr>
<tr>
<td>4.2</td>
<td>Server architecture</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>Figure illustrating customized content for two users</td>
<td>26</td>
</tr>
<tr>
<td>4.4</td>
<td>Part of the links configuration</td>
<td>28</td>
</tr>
<tr>
<td>4.5</td>
<td>Rendered links configuration</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>Security tab for the company server inside Firefox</td>
<td>33</td>
</tr>
<tr>
<td>5.2</td>
<td>Figure illustrating the network traffic for the mobile application during usage</td>
<td>38</td>
</tr>
<tr>
<td>A.1</td>
<td>Display notes for a room in an office</td>
<td>53</td>
</tr>
<tr>
<td>A.3</td>
<td>Display building information</td>
<td>55</td>
</tr>
<tr>
<td>A.4</td>
<td>Provide links to frequently used websites</td>
<td>56</td>
</tr>
<tr>
<td>A.5</td>
<td>Display lunch menu of local restaurants</td>
<td>56</td>
</tr>
<tr>
<td>A.6</td>
<td>Provide the latest news for the company</td>
<td>57</td>
</tr>
<tr>
<td>A.7</td>
<td>Display available parking slots</td>
<td>57</td>
</tr>
<tr>
<td>A.8</td>
<td>Display office control information</td>
<td>58</td>
</tr>
<tr>
<td>A.9</td>
<td>Display all known issues</td>
<td>58</td>
</tr>
<tr>
<td>A.10</td>
<td>Report a new issue</td>
<td>58</td>
</tr>
<tr>
<td>A.11</td>
<td>Multiple configured cards</td>
<td>59</td>
</tr>
<tr>
<td>A.12</td>
<td>Multiple configured cards</td>
<td>59</td>
</tr>
<tr>
<td>A.13</td>
<td>Add a new note inside the mobile application</td>
<td>60</td>
</tr>
<tr>
<td>A.14</td>
<td>The customer server directly updates the configuration</td>
<td>60</td>
</tr>
<tr>
<td>A.15</td>
<td>Current available parking slots</td>
<td>61</td>
</tr>
<tr>
<td>A.16</td>
<td>Available parking slots a moment later, automatically updated</td>
<td>61</td>
</tr>
<tr>
<td>B.1</td>
<td>Sign in page</td>
<td>62</td>
</tr>
<tr>
<td>B.2</td>
<td>Sign up page</td>
<td>63</td>
</tr>
<tr>
<td>B.3</td>
<td>Default static content page</td>
<td>63</td>
</tr>
<tr>
<td>B.4</td>
<td>Default dynamic content page</td>
<td>64</td>
</tr>
<tr>
<td>B.5</td>
<td>Configuration using four cards</td>
<td>65</td>
</tr>
<tr>
<td>B.6</td>
<td>Received configuration inspection tool</td>
<td>66</td>
</tr>
<tr>
<td>B.7</td>
<td>First part of configured content</td>
<td>67</td>
</tr>
<tr>
<td>B.8</td>
<td>Second part of configured content</td>
<td>67</td>
</tr>
<tr>
<td>C.1</td>
<td>Sample of the raw unfiltered network traffic</td>
<td>68</td>
</tr>
</tbody>
</table>
List of Tables

5.1 Architecture, design and threat modeling requirements. ........................................ 35
5.2 Data storage and privacy requirements. ................................................................. 35
5.3 Cryptography requirements. .................................................................................. 36
5.4 Authentication and session management requirements. ........................................ 37
5.5 Network communication requirements. ............................................................... 39
5.6 Platform interaction requirements. ....................................................................... 40
5.7 Code quality and build setting requirements. ....................................................... 40
1 Introduction

1.1 Motivation

Technical services used in society today are supposed to be easy to use, readily available and respectful of the user’s privacy [1]. Because of high expectations for availability, a natural development of how technical services are made available is through implementations suitable for use in mobile devices [2],[3]. Everyone knows how to use their phone and takes it with them wherever they go, which makes it a perfect device for convenient service applications. One concern for both users and companies with this development of technology is security [4]. Today, most companies keep records about their employees as well as their customers digitally. In most cases, companies highly value the confidentiality and integrity of this stored data, and since companies increasingly keep records digitally, the need to keep digital data secure is growing.

Imagine being a company offering a service in the form of a mobile application. A possible scenario would be to offer the same service to several customer companies, where each of these customers and their end users have their point of view on how this service could be improved. In most cases, it will not be feasible for the developers of the mobile application to address each of the customers’ wishes due to the amount of time it would require. An example of such a scenario is illustrated in Figure 1.1. In this scenario, it is clear that Customer 1 and Customer 3 have completely contradicting wishes. If the company wants to satisfy both customers, they need to offer two different versions of the same service. At the same time, Customer 2 wants a way of updating the number of currently free parking slots at the parking garage close to the office. To be able to satisfy this request, the app must be able to receive and react to regular updates from Customer 2 as the number changes. To solve this problem the company would need some way of allowing customers to add customized content for themselves.

1.2 Aim

This thesis examines the possibility of allowing customers to define modules that can be included in mobile applications. Ideally, these modules should be able to receive updated data
from the customer and thereby change and update themselves dynamically. If a mobile application could support such modules, it would be a solution to customers wanting different versions of the same application as well as customers wanting the ability to update data inside the application.

1.3 Research Questions

Allowing customers to define dynamic content and provide data to it needs to be carefully implemented since the content received potentially could be malicious or faulty. Blindly trusting content provided by customers could introduce severe security risks. How dynamic can the dynamic content potentially be? Can it be predefined text, templates or even executable code; what is suitable? The purpose of this thesis is therefore to answer the following question:

- How can user-customized content in a mobile application be realized with considerations to security and how can the security level of such an application be evaluated?

1.4 Delimitations

The configuration web page used by the customers should be implemented using React and the mobile application using React Native.
2 Background

2.1 Thesis Environment

The company Senion AB[1] is an example of a company offering a convenient service in the form of a mobile application. Senion offers a workplace assistant which uses their cutting-edge indoor positioning system to help increase the efficiency and convenience in work environments, such as offices. Senion’s customers have expressed a need for being able to configure the assistant to their specific work environments in more detail than what is currently possible. This dynamic behavior of the application requires a configuration platform for the customers where they can define their specific content for their workplace assistant. The way in which the customers should be able to define their dynamic content is challenging from a security perspective.

The work assistant application in its current state uses modules which are predefined by Senion. The first step in allowing customers to define their modules is to create a way of allowing customers to specify which additional modules and data to show in their application. One example of a specific feature which a customer request is that they want to be able to show their local restaurants menus in the application. Furthermore, a user should be able to interact with this module in some way, such as pressing buttons or submitting some text. This is impossible in the applications current state since there is no tool available to the customers which allows them to supply their own data or modules. This thesis aims to fill the above gaps, while simultaneously keeping security in mind.

The frameworks and languages used in this study are influenced by the frameworks and languages which Senion suggests for integration with their existing mobile application.

2.2 Mobile Application

Mobile applications may be developed targeting a specific operating system such as Android or iOS, or by using a cross-platform framework, such as React Native, which targets both platforms.

2.2.1  React Native

React Native [5] is a framework built by Facebook to help developers quickly and easily build mobile applications using JavaScript. Web developers with previous knowledge of using JavaScript in web programming will have an easier time getting started with mobile application development since they can use the same language and techniques with React Native as they do in React. Another strength of using React Native compared to using traditional Java for Android and Objective-C for iOS is that both Android and iOS can use much of the code, hence allowing developers to write most of the code for both platforms simultaneously. At the same time, developers may write platform specific code if needed since there exist files for targeting either Android or iOS. However, running and testing an application requires the developer to configure a working Android environment in Android Studio [2] for the Android version and a working iOS environment using Xcode [3] for the iOS version.

Expo

Expo [6] is a set of tools created to help developers more easily develop mobile applications using React Native. Instead of having to go through a tedious setup, Expo enables developers to get started with development as soon as possible with very minimal setup. Basically, the setup is to install Expo, initialize an Expo project and open it in a text editor. At this early stage, one can run the command expo start and already have a functioning application which runs on both Android and iOS alike. Furthermore, developers may test the application on their mobile device directly without any additional setup by having their phone and computer on the same network and scanning the QR-code generated when running the command expo start. The code is pure JavaScript, specific files for Android and iOS does not exist, and the project runs on both platforms out of the box. The downside of using Expo is that developers are limited to the functionality which the Expo ecosystem supports. Currently, compared to native Android and iOS development, there are some features which Expo does not support, such as background code execution. In conclusion, pure React Native is more reliable and suitable for production-level software, whereas Expo is great for developing prototypes quickly that are not planned to be used in production.

React Native Elements

React Native Elements [7] is a third party library for React Native apps that help developers style their apps by using pre-designed components such as buttons, inputs, and icons. React Native Elements is very useful for developing a mobile application quickly but at the same time somewhat good looking.

2.3 Mobile Application Security

Given the technological movement from traditional desktop and web applications to more convenient mobile applications, a new threat landscape has emerged. New technology introduces new security complications, and mobile applications are no exception. As more services are made available through mobile applications, more users start using them, resulting in more money involved. Popularity and money attract attackers, which in turn means that security becomes more important. It is easy to think that the mobile application threat landscape should be very similar to desktop or web applications, and can be seen as the same application just put inside a smaller device. However, these smaller devices use other operating systems, which means that they have a different way of doing things. Because of these differences in the core principles of how the applications run, the security threat landscape also becomes different. According to Bernhard Mueller, mobile security is all about

2https://developer.android.com/studio/
3https://developer.apple.com/xcode/
data protection [8]. Mobile phones often contain much information about the owner’s life, and if an attacker gains access to a mobile phone, they could potentially be able to access much of this sensitive information. It is therefore important to develop mobile applications with considerations to security by implementing security measures according to the different threat landscape. According to Mueller, the key factors that need to be carefully implemented and secured inside mobile applications are secure network communication, proper usage of cryptographic APIs, inter-app communication and data storage.

2.3.1 OWASP Mobile Security Testing Guide

Understanding that mobile application security is important and different from traditional application security is a good first step, but how can developers and project managers tackle this in their projects? Jeroen Willemsen, Bernhard Mueller and Sven Schleier at Open Web Application Security Project (OWASP) identified this problem and decided to start the OWASP Mobile Security Testing Guide (MSTG) project [9]. The purpose of the MSTG project is to help security testers, developers and product managers create mobile application security requirements, test them and verify their security enhancements according to the applications required security level. The MSTG [10] itself contains detailed information on how to test certain security aspects in mobile applications. Alongside the MSTG, the project also resulted in the Mobile Application Security Verification Standard (MASVS) [8] which can be used to help verify security requirements after implementing them and a Mobile App Security Checklist to keep track of the progress.

2.3.2 Privacy

Privacy is a matter receiving increased attention lately, both by users and creators of digital services. An important question to ask as a developer is how can users’ privacy be protected? Balebako and Cranor [11] investigated privacy concerns in apps and how they could be improved. First of all, they acknowledge some important privacy guidelines from the US, Australia, and Europe. These guidelines include minimizing the amount of data collected, avoid retention of old data, communication through a privacy policy to the users, as well as encrypting transmitted and stored data. In the article, the authors realize that developers want to improve the privacy for their users but that they need help by the surrounding ecosystem of app development. Their conclusion was that privacy would only improve if the pressure to prioritize privacy increased. Today, this help is noticeable through privacy regulations such as the General Data Protection Regulation (GDPR) in the EU. Such regulations help developers prioritize privacy and make it a primary task in any company handling user information. It is now a priority to remember these privacy regulations and be sure to implement them through privacy policies and handling user data with care.

2.3.3 Access Control

Mobile applications are limited in what they can access on the device they are installed on via their respective permission systems. A few years ago, an application would directly ask users for all of the permissions the developer had specified in a manifest file. Today, the permissions in Android and iOS are dynamic as opposed to static, and from a purely functional perspective Android and iOS works the same with regards to permissions. If an application offers some feature that makes use of the camera, then the application will ask for camera usage permission only if a user tries to use the specific feature [12],[13]. This is helpful since the mobile device will continuously inform the user what permission the application gain

---

3https://www.android.com/
2https://www.apple.com/se/ios/ios-12/
2.3. Mobile Application Security

access to on the mobile device. If an application wants to use something which the user does not agree with they can easily block the request \cite{12,13}. As shown in the research done by Squires, W., and Centonze, P. \cite{14}, this evolution of permissions has resulted in mobile applications no longer having too many or too few privileges. On the other hand, developers may now include frameworks for features that are never used. Developers should be careful not to include frameworks they do not need, not only because of bloat but also because of security risks, since any third-party software potentially can be malicious.

Data leakage has been widely studied in mobile applications. In their research, Canbay et al. \cite{15} present one method of examining android applications to determine if they are benign or malicious. Their focus is on permissions, what permissions are used, and how they are used. Since permissions are not very granular and often grant wide access, it is important not to use any permissions one does not need. The authors found that the permission with most offenses was the permission to read SMS. However, this is probably prone to change since new messaging application alternatives such as Facebook Messenger\footnote{https://www.facebook.com/} and WhatsApp\footnote{https://www.whatsapp.com/} lead to SMS usage declining \cite{16}. In the paper by Canbay et al. they focused on four major permission categories, READ, ACCESS, GET and SEND. After some research, they came to the conclusion that the READ category was the most volatile one. Therefore it seems like the main permission to restrict the usage of and handle with care is the READ permission. As a developer, it would be good practice to limit the usage of these permissions unless they are necessary.

2.3.4 Availability

Schmeelk and Aho \cite{17} have examined availability risks in Android applications to help fill the gap in research on the subject. According to Schmeelk and Aho, the availability of Android applications is something often overlooked in the research of security risks. In their report, Schmeelk and Aho decide to divide availability risks and threats into application-level threats and system-level threats. Application-level threats which they discuss are Android permissions, vulnerable APIs, embedded libraries, network requests, and inter-application interfaces. As for system-level threats, they mention changing network configurations, disabling the device keyguard and killing background processes. After stating these risks, the authors then describe defense solutions for these parts of Android applications and the system itself. For system-level defense, the authors state that effective defense includes rate-limiting API requests for specific vulnerable APIs to avoid abuse, as well as monitoring the frequency of device configuration changes. Furthermore, Schmeelk and Aho say that system level defense could be developed straight into the operating system, but a watchdog application could also handle it. Currently, it seems like the operating system does not handle everything and using watchdog applications could be needed to ensure protection. Regarding the application-level defense, the authors say that there are three main defensive solutions to protect against Denial of Service (DoS). The first defense is to guard against sending plaintext messages over the internet, the second to guard inter-application and inter-system communications, and the third defense is to use a watchdog application which can help monitor the behavior of an application on a device. In their conclusions, Schmeelk and Aho claim that their research is unique in identifying a more complete picture of the landscape of availability attacks. The prospect of the authors is that this research will aid Android operating system developers, application security analysts, as well as application analysis malware detection tools in protecting against availability risks.
2.3.5 Vulnerability Detection

Detecting security vulnerabilities in mobile applications is an important step to be able to understand where the vulnerabilities are located. When the vulnerabilities are known and located, they can be mitigated and perhaps fully prevented. Whereas if one does not know where to look for the vulnerabilities, it is impossible to mitigate them. Zhou et al. [18] have studied how mobile application security can be tested, and vulnerabilities detected. After dividing the security concern into client-side and server-side vulnerabilities, Zhou et al. came to the conclusion that combining static and dynamic security detection methods is the proposed method for detecting vulnerabilities on the client-side of applications. Vulnerability detection for server-side is approached using a fuzzy testing method based on the HTTP protocol. After examining where and how to look for vulnerabilities in general applications, Zhou et al. were able to construct a platform for scanning Android applications to find vulnerabilities both on the client-side as well as server-side. The article is concluded by using this platform to scan three Android applications (which are popular in China) for vulnerabilities. The platform was able to detect several high-risk vulnerabilities such as sensitive information leakage, secondary packaging and misuse of permissions in these applications. Because of this, the authors claim that the platform can be used to great effect for detecting vulnerabilities in general mobile applications and help companies strengthen the security of mobile applications.

2.4 Web Application

There exists tons of different frameworks and tools for helping developers create web applications quickly and easily.

2.4.1 React

React [19] is a JavaScript library used for building user interfaces, whose primary focus is to be fast, scalable, and simple. One key selling point for React is the efficient update and rendering of only affected components when data changes by using a virtual DOM (document object model). React also promotes reuse of components which can save time and make modifications easier since the code is split up into individual building blocks which do not affect each other. Another feature is JSX, which is a syntax extension allowing developers to mix HTML with JavaScript. Furthermore, JSX offers a security benefit by protecting the code from injections through escaping any values embedded inside JSX before rendering. This means that all content is converted to be interpreted as a string before rendering, which helps prevent cross-site scripting attacks.

2.4.2 Redux

Many web applications need to store data which can be conveniently shared between different parts of the application. Redux [20] is a state container which can be added on top of an existing UI layer, such as React, providing a convenient data store which helps applications through a global state data store. When using Redux components can connect to this global data store and retrieve the data they need without having to worry about where this data comes from or which component originally fetched the data. Redux simplifies data sharing by providing centralized data storage inside web applications.

2.4.3 Semantic UI

Responsiveness and good looking design are expected of web applications developed today. Semantic UI [21] is a tool for helping developers design beautiful websites quickly. By using pre-designed components for the user interface such as containers, forms, lists, buttons,
2.5 Web Server

Many different technologies for creating a back-end API server exists, in this section some of the alternatives considered for this thesis are presented.

2.5.1 Node

Node [23] is a runtime environment used to execute a program written in JavaScript outside the browser. This allows developers to use JavaScript for building outside browser programs such as web servers. Node makes use of an event-driven non-blocking I/O model, which makes it easy to use and efficient. Since Node enables JavaScript server side, the developer can reuse the knowledge of using JavaScript on the client side of applications. This means that the full stack can be written in the same language, which is very convenient. Node means fast development but may be insufficient for programs relying on CPU-intensive operations.

Express

Express [24] supply developers with all the tooling they need to build a web server using Node. Express is responsible for handling incoming HTTP-requests and sending back responses to users. Furthermore, Express also makes it easy to work with request parsing, routing and applying middlewares. Middlewares are tiny functions meant to intercept or modify requests when they come through an express server. Express speeds up development and makes life easier for developers looking to create a web server.

2.5.2 Java

Java [25] is a general-purpose programming language that can be used for building applications that may run on multiple platforms. For example, Java can be used for creating a REST back-end API. A benefit of using Java is that it has been around for a long time and has a large community with lots of experience of using the language. A disadvantage of Java is that it is quite complex, and since it can be used for many different things, it is not tailored for use in a specific setting.

DropWizard

DropWizard [26] is a sort of framework for Java that includes all of the different packages one need for building a Java REST back-end API in a single service. DropWizard does all the work of having to find what packages are suitable for building a web server using Java.

2.5.3 Kotlin

Kotlin [27] is a general purpose programming language which is often used as an alternative to Java for Android development. Some proposed key benefits of using Kotlin instead of Java are that Kotlin code is concise and also perceived as safer. This claim is often based on features such as the elimination of null reference errors, enabled extension functions, and support for functional programming and coroutines which are all included in Kotlin. Kotlin is also inter-operable with Java which makes it very easy to start using it inside an existing Java project or by replacing existing Java code.
2.6. JavaScript and Security

Ktor

Ktor [28] is a framework used with Kotlin for building asynchronous servers and clients. The purpose of using Ktor is to make it easier to get started building clients and servers using Kotlin for web and mobile applications. One reason for choosing Ktor over an alternative, such as Spring for Java, is that Ktor is much more lightweight than traditional frameworks, which makes it easier to get started.

2.6 JavaScript and Security

JavaScript is a scripting language that can be used to create content and functionality for websites. A problem with allowing users to submit JavaScript code on websites is that there exist many ways to inject malicious code using JavaScript. Extensive research has been conducted on how to protect against these types of attacks, which can be divided into three main different approaches, changes to browsers and standards, restricting scripts to a safe subset of JavaScript, and code transformation and runtime monitoring approaches [29], [30], [31], [32]. Unfortunately, the solutions proposed are complex, which results in that they are difficult to understand, implement and maintain as a developer. Another thing to keep in mind is that an attacker only has to guess right once on a vulnerability to hack the system, whereas the developer must be right in every possible scenario in order to keep the system secure.

2.6.1 Changes to Browsers and Standards

Erlingsson et al. [30] states that in order to achieve end-to-end security, the web clients (browsers) must be enhanced. By adding Mutation-event transforms to browsers, the authors argue that fine-grained, application-specific security policies can be enforced client-side on not trusted code. Supposedly, this could increase security since the browser would be able to enforce application security policies. However, this is not something developers can change on their own. It is in the hands of the browser companies to decide whether they want to implement enhanced security inside their browsers.

2.6.2 Restricting Scripts to a Safe Subset of JavaScript

One example of the restricting scripts to a safe subset method is GATEKEEPER, which is introduced by Livshits and Guarnieri [31]. GATEKEEPER is presented as a static sound policy enforcement tool, and this tool can be used to inspect JavaScript code that is being supplied by third parties. GATEKEEPER uses a policy that restricts allowed input to a safe subset of Javascript, which it feels confident enough to analyze and determine whether the code is safe or not. However, this tool can only be seen as a proof of concept, not a commercially viable option. The authors state that they hope centrally-hosted software repositories such as App Store or Android Market will take inspiration from GATEKEEPER to ensure security and quality of third-party software contributions. It does not seem like the concept is suitable for smaller actors since implementing a tool like GATEKEEPER would require significant work and testing.

2.6.3 Code Transformation and Runtime Monitoring Approaches

In their paper Phung et al. [32] describes a method to control JavaScript execution. Their method makes use of policies that should be defined by the developers who are supposed to know what functionality is required for the particular case at hand. Every function and API-call are processed to try and identify malicious requests to reject them, based on the policy of what is allowed. The problem with this approach is that these policies must be carefully constructed to allow precisely enough and correct functionality. It is important that the developers are aware of how attacks work in order to not allow vulnerable instructions.
Even though the policy is white list based, meaning only allowing functions and URLs that are explicitly chosen, writing these policies can be difficult and time-consuming.

2.7 Cross-Site Scripting

The most popular form of malicious JavaScript code injection attacks is known as Cross-Site Scripting (XSS) \[29\], \[33\]. On websites today, it is common that users are allowed to post some form of content, such as comments or files. In some cases, it would be convenient if users could enhance this content by using JavaScript to add some functionality to it. However, this can be dangerous since the browser cannot tell if the code is safe or malicious. The preferred way of mitigating these attacks is to sanitize the user input before it is used. The problem with this is that one then might destroy the purpose of allowing users to add their wanted functionality; it is not easy to tell which code is malicious and which is not.

2.7.1 Stored

In a stored Cross-Site Scripting attack \[29\], an attacker will post a malicious script which is then stored inside a database on the web server. When another user visits a part of this website, which includes this post, the malicious script will be executed in their browser. This could result in the attacker gaining access to the user’s cookies, containing information about the user’s authenticated session. This information could then be used by the attacker to impersonate the real user on the website, executing actions on the original user’s behalf.

2.7.2 Reflective

Another version is the reflective Cross-Site Scripting attack \[29\]. In this attack, the malicious script is supplied directly to the victim. It can be included in a message or a mail containing a link which has hidden malicious code inside of it. When the user presses this legitimate looking link, one of the parameters supplied to the website could contain the malicious script. This script will then execute inside the victim’s browser when the website tries to make use of the parameter in some way.

2.8 Cross-Site Request Forgery

Another common website attack is known as Cross-Site Request Forgery (CSRF) \[34\], \[35\]. The attack is executed by sending a link to a user which when clicked takes advantage of the fact that browsers often store session keys. The attacker makes use of this behavior by having the link issue a request to an application and provide the victim’s legitimate session key stored in their browser. This will potentially trick the application into thinking that this user willingly is trying to interact with the application and granting access to perform the requested action based on the fact that it includes a valid session key.

2.9 NoSQL injection

NoSQL injection \[36\] is the equivalent of SQL injection for the NoSQL world. The purpose of the attack is to inject malicious code which will execute when the attack string is evaluated. An exploit could result in attackers gaining access to sensitive data inside the database. A solution to preventing NoSQL injections is to sanitize the inputs before using them. For MongoDB databases, this can be done by defining schemas with the helper framework Mongoose. Mongoose will convert user input into the data type specified in their schema before using them. If an attacker tries to pass in a malicious object, but the defined type in the schema is string, then the object will be converted into a string resulting in no harm being done.
2.10 CSS Injection

Problems with CSS injection can arise when user input is embedded into CSS without escaping dangerous characters. The impact may vary, but it could lead to Cross-Site Scripting vulnerabilities or extraction of sensitive data. It is important not to forget this possibility when allowing users to supply styling props. In React, the style attribute of an element can be used to add dynamical styling when rendering the element. The style attribute will only allow JavaScript objects with camel cased properties and not CSS strings, which prevents Cross-Site Scripting and is more efficient.

2.11 Authentication

APIs usually want to know who is using them in order to accept or deny requests based on this knowledge. This is where authentication enters the picture. Authentication is used in systems to determine who a user is. Users could be a person or another application. Several authentication methods exist such as a username along with a password, cookies, and JSON Web Tokens (JWTs).

2.11.1 Username and Password

Identifying a user through the parameters username and password is the traditional way of authenticating users on the web. Users are required to sign up to the application by registering a username and a password which the server will save in a database. Passwords should never be stored in plaintext but rather have a randomized piece of information (known as salt) added to them before being sent through a hash function such as SHA-256. In case of data-leakage, this will require the attacker to use the individual salt of each user and then perform some form of dictionary or brute force technique to produce the same hash value to be able to identify the password. To prevent this, the hashing algorithm used must be designed to take a considerable amount of time. One example of such a hashing algorithm is Bcrypt. By using a deliberately slow hashing algorithm, it requires much effort from attackers to identify passwords even if they manage to get a hold of the data. Whenever a user tries to log in to the application, the server will take the proposed password, add the user-specific salt to it and then run it through the chosen hash function and only if the hashes match the user will be allowed access. Once a user has proven their identity through an initial sign-in process, they will then be authenticated through another more convenient process as long as they remain signed in to the application. Username with password authentication is therefore often combined with cookies or JWTs for further authentication of requests when a user is signed in.

2.11.2 Cookies

A cookie is an object which can be used for storing information that is passed between a client and a server during communication. As a means of authentication, the cookie may be used to hold a session key. The server places the session key inside the cookie when the user sign-in and the cookie is then also stored locally at the user’s computer. This cookie can then be passed along with requests and used by the server to identify which user issued the request. Cookies are included with HTTP requests by default in web browsers. An issue with this default behavior of web browsers is that it makes the user vulnerable to CSRF-attacks since their session key will automatically be included in requests. This means that the attacker does not need to know the session key in order to execute the attack.

9https://reactjs.org/docs/dom-elements.html#style
2.13.3 JSON Web Tokens

A JSON Web Token (JWT) is an encrypted JSON object which can be used by a server to identify a user. When a user signs in to an application, the JWT is created by taking the user id and encrypting it with a secret string along with the current time. This JWT is then sent to the user and stored locally. Whenever the user then issues a request they will include this JWT which the server can decode to retrieve the user id of the user this token belongs to, hence identifying the user. Compared to cookies, J_TOKENs offers the security advantage of requiring the client to attach the authentication token with each request manually. This means that attackers wanting to execute a CSRF-attack must somehow also provide the correct token with the forged request. At the same time, this also means that developers must implement the addition of the authentication token to legitimate requests manually compared to the seemingly convenient automatic inclusion for cookies.

2.12 Authorization

Authorization is used in systems to regulate access to valuable resources. Many systems contain data that only authorized users are meant to be able to access. To enforce this behavior, the system must make use of an authorization method. Often these valuable resources are available by requesting them from an API which means the API needs to validate the user issuing the request and decide whether they are allowed access before sharing resources in order to protect them.

2.13 Encryption

Encryption is the main tool available for implementing security mechanisms in digital systems. Encryption is used to transform readable data (plain text) into something unreadable (ciphertext). This enables the transmission and storage of data in a form that is not readable. Decryption is used to transform ciphertext back into readable plain text.

2.13.1 Symmetric Encryption

Symmetric encryption uses a secret password to create a unique key. This unique key is then used to encrypt messages and can only be decrypted using the same key. The security increases with the length of the encryption key, but at the same time, it takes longer to encrypt and decrypt. However, encrypting and decrypting using symmetric encryption is generally much faster than asymmetric encryption.

2.13.2 Asymmetric Encryption

Asymmetric encryption uses two keys, known as the public and private key. The public key is supposed to be known by everybody, whereas the private key should always be kept secret. The public and private key are generated at the same time and are mathematically related. If the public key is used for encryption, then only the private key can decrypt it, which ensures confidentiality. If the private key is used for encryption, then only the public key can decrypt it, which ensures authentication.

2.14 Hash Functions

A hash function is used to take a plain text message and transform it into a fixed length hash value regardless of the message size. An important feature of hash functions is that it is very difficult to find the original input from the resulting hash value. Furthermore, a specific hash function will always produce the same resulting hash value when given the same input.
Hash functions can be viewed as one-way functions that require no keys and are used to provide integrity.

### 2.15 Digital Signatures

Digital signatures\[42\] are created by taking the fixed length result of a hash function and encrypting it with the sender’s private key. In a practical scenario, a sender takes the original message used to produce the hash value and sends it along with the digital signature. A recipient can then decrypt the digital signature by using the sender’s public key and can also create a new hash from the original message using the same hash function as the sender. If the two hashes match when comparing the decrypted hash with the newly created hash the recipient has verified both that the message has not been tampered with and that it was sent from the genuine sender. A digital signature ensures authentication, non-repudiation, and integrity.

### 2.16 Transport Layer Security

Transport Layer Security (TLS)\[42\] is important for internet security and privacy because it is the most used method of encrypting data being transmitted over the internet. TLS is a cryptographic protocol that supports confidentiality/privacy through encryption, authentication through an initially negotiated secret, as well as integrity through message integrity checks. When an application communicates with a remote endpoint using TLS, the traffic is guaranteed to use end to end encryption, which means that only the recipient and the sender can read the contents. TLS is also known as the successor of the previously used Secure Sockets Layer (SSL) protocol.

### 2.17 Related Work

Adaptive Cards\[43\] is a concept developed by Microsoft. The idea is to have users write a description of the content they want to share in different applications as a JSON object. This object can then be rendered natively on different host applications, which means that the content will be the same in these different applications, but the look and feel will be specific for each host. This way, users can focus on their content in one simple schema which then can be used for multiple applications, increasing efficiency as well as simplicity. The potential downside of using these cards instead of creating your own is that users cannot fully modify the look and feel. However, as long as users are content with the design provided for different hosts, then they can skip the designing for each platform and focus solely on the content they want to share.

Office Add-ins\[44\] is another concept developed by Microsoft, which builds on the idea to let users build solutions that extends existing products such as Word, Excel, PowerPoint and more. The idea is that the solution can run across multiple platforms and add new functionality. To create add-ins, users must supply an HTML-file, a JavaScript file, a CSS file as well as a manifest file.

Qian et al.\[45\] highlight the need for security and defensive requirements early on in the software development life cycle (SDLC) in order to increase the security of apps. A claimed benefit of addressing security early is that security holes will then be fixed before attacks can take place opposed to a reactionary fix to an existing or already performed attack. The authors also provide a guided security requirements analysis based on the OWASP Mobile top 10 security risk recommendations targeting Android apps. They conclude that increased security can be achieved by including security requirements early on in the SDLC and that
performing an OWASP-based analysis on these requirements can be used to verify them resulting in apps with a higher level of security.

Acharaya et al. [46] investigated how to improve mobile security in health care systems. They decided to create a checklist based on the OWASP mobile top 10 risks that could guide developers trying to secure mobile applications. To test the checklist the authors audited two medical applications and, guided by the checklist, identified a vulnerability in both of them.

Argudo et al. [47] were interested in how privacy vulnerabilities could be analyzed in Android applications. For this purpose, the authors decided to create a methodology combining the OWASP mobile security project and Open Android Security Assessment Methodology (OASAM) controls along with best practices. The methodology was tested by performing privacy vulnerability analysis on multiple mobile applications for public institutions of Ecuador. The vulnerability analysis found several vulnerabilities thereby indicating that mobile application privacy security could be improved by performing security tests of Android mobile applications using the presented methodology.

Sai et al. [48] performed a privacy and security analysis targeting cryptocurrency wallet mobile applications where the analysis was inspired by the OWASP mobile top 10 risks. For comparison, they also analyzed common banking and trading applications. The results show that traditional banking applications tend to have fewer security vulnerabilities and privacy issues. The authors also argue that static code analysis might not always be the best approach since a majority of the reported threats turned out not being real security or privacy issues.

Phumkaew and Visoottiviseth [49] conducted security assessments on three hospitals and five stock-and-trade Android mobile applications. The assessments were guided by the OWASP mobile top 10 risks and made use of static analysis as well as dynamic analysis. The results showed that all of the eight analyzed applications contained vulnerabilities. The authors request that Android app developers improve their security awareness, especially when sensitive information is part of the equation.

Meng et al. [50] identified a need for improved secure mobile software development. The authors suggest that first of all secure software development should be addressed earlier on in the SDLC. Another problem they identified was that many professional developers do not have the necessary security knowledge required. Meng et al. present a static security analysis approach through their plugin FindSecurityBugs for Android Studio constructed to help Android developers locate vulnerabilities in their applications. The FindSecurityBugs security flaw detection includes detectors based on the OWASP mobile top 10 risks to help with the vulnerability coverage but also allow users to design and add their own detection software. The authors hope that the FindSecurityBugs plugin can help developers use Android vulnerability detectors to increase the security level of Android apps.

Jain and Shanbhag [51] were concerned with the security and privacy risks in mobile applications. Given the rapid growth of mobile application usage, the security must keep up in order to protect the users. The authors propose that information security should be included early on in the SDLC which inherently reduce the cost compared to fixing issues at a later stage. Furthermore, they advocate that a thorough security assessment should be performed on all mobile applications before they are allowed to be released into production.

Zhang et al. [52] investigated the virtualization feature within the Android operating system and its security threat to users. They came to the conclusion that most of the virtualization frameworks used today are vulnerable to attacks which can be exploited to spread malware. Furthermore, they provide attack examples along with mitigation recommendations for the frameworks and for app developers.
Wu and Chang [53] performed a cross-platform analysis of the problem with indirect file leaks in mobile applications. They claim that both Android and iOS can be exploited to leak private files from popular apps. Several mitigation methods are proposed such as disabling JavaScript execution in local schemes as well as restricting commands and network requests that can access private file zones.

Mobile applications commonly make use of WebView functionality to allow users to interact with an embedded web browser containing some form of content. Tuncay et al. [54] identified severe vulnerabilities and security issues that can be exploited in applications using WebView. They concluded that popular Android apps are vulnerable and propose a solution called Draco providing a new access control mechanism for WebView that can enforce access policies during runtime. Davidson et al. [55] also investigated issues with web-embedding applications and came to the conclusion that many attack vectors still are exploitable. They propose a solution through the usage of their third-party app WIREFRAME that can provide the web-embedding functionality while at the same time enforcing security policies on every interaction between the Android app and the web content. Furthermore, Li et al. [56] saw a demand for convenient cross-WebView communication which lacked proper security on the channel. They proposed building protection through the OS-level and presented a technique using this method for controlling the channel. Yang et al. [57] also investigated vulnerabilities in apps that make use of WebView. They developed an automatic tool called EOEDroid that scans the web events occurring inside the WebView by using static analysis and selective symbolic execution. Whenever a vulnerability is found the tool also generate exploit code which can be used to verify the vulnerability.
In order to implement a product which will satisfy customer wishes, it is important to understand what the requirements are and what the goal of the project is. Once this understanding is achieved a system architecture and plan for meeting the requirements can be constructed to help guide the implementation and its associated decisions. During the system design phase two main activities were conducted, requirements elicitation and creation of a system architecture both of which are described in the following sections.

3.1 Requirements Elicitation

To achieve an understanding of what type of product was desired, requirements were elicited in meetings with a developer of the existing product who was knowledgeable about what customers wanted. In these meetings, the developer also shared conceptual pictures and ideas that customers had previously shared. It is important to note that a customer could be an entire company with many individual end users. In this thesis, the scenario where a customer is multiple individual end users is treated as if they all shared the same account.

1. It should be possible for customers to add customized content to their mobile application.
2. Customers should be able to add static content to the application that can be updated manually.
3. Customers should be able to add dynamic content which can receive updates both from user input and automatically given that customers implement their own logic.
4. Personal end-user data should always be kept secret and never sent to a customer server.
5. The customer server should be able to verify whether a request is genuine and originating from a trusted source.
Given the requirements and feedback from customers and their end users, a set of use case scenarios were created to facilitate the system design phase as well as increase clarity of what the customers wanted.

1. **Building**
   Users want access to information about the building they are located in. This information could be at what times the building is open, at what times the alarm is turned on, and a link to the evacuation procedure manual for the building.

2. **Links**
   Users want easy access to important links that often are used. The mobile application should be able to display a number of links and navigate users to desired web addresses.

3. **Lunch**
   Users want to see today’s offered menu of the closest restaurants. The mobile application should be able to display part of the menu and provide a link to the complete version.

4. **News**
   Relevant news should be easily shared using the mobile application. The news could be messages or announcements.

5. **Notes**
   Users want to be able to write down notes directly to the phone instead of physically on whiteboards. The mobile application should be able to receive information and update itself.

6. **Parking**
   Users want to know the status of free parking spaces close to the office. Should be updated on a regular basis to provide accurate information.

7. **Service desk**
   Issues around the office should easily be documented and reported through the mobile application.

8. **Smart office controls**
   Information about the office environment should be available. The information could be temperature status, window shades status as well as lights status.

After creating use cases, another decision was made to create an initial test application inspired by one of the scenarios. The goal of this initial application was to get a feel for what problem the end product was supposed to solve compared to the existing product and generate a better understanding of how the system architecture should be designed. Based on the notes use case scenario an initial test application was implemented using React Native together with Expo which made the development process quick. The main question to be answered was what the differences were going to be between an ordinary application entirely controlled by a supplying company compared to a dynamic application which could be manipulated and interacted with by customers. Based on this initial test application, the main difference was identified. In a regular mobile application, all content inside is predefined by the developers. Users may not modify the content or attach code of their own. In the case where users should be able to manipulate what content is available, they must be able to somehow decide the composition of code running inside the application. The next question then appeared, how can users be allowed to modify or add content on their own? Perhaps the
most natural way would be allowing users to supply their own JavaScript code. This way, users would be given total control of how their content function. However, there are many security issues related to malicious JavaScript code in applications, such as cross-site scripting attacks. Users could potentially supply malicious code trying to extract information or crash the application if they were allowed to provide JavaScript. It is possible to try controlling and validating the JavaScript code users enter, but it is complicated. Since complexity is the enemy of security, a decision was made not to allow users to enter their own JavaScript code directly into the application. Instead, it was decided users should be able to choose between predefined JavaScript components which are known to be safe. It is essential that users are able to interact with these components such as deciding text, layout, size,... but in a strictly safe manner. Inspired by the adaptive cards approach, a strategy which seemed appropriate could be implemented by allowing users to supply a JSON file which contains information about the components they want and how they want to customize them. The app could then parse this JSON file and understand what the users want, rendering the desired content inside the mobile application using different predefined components depending on the given configuration. This way, users would only supply JSON data instead of actual code which is easier to control from a security perspective.

3.2 System Architecture

The requirements, use cases as well as the lessons learned from the early test app were heavily considered during the system architecture creation process alongside relevant theory and related work. The system architecture overviews were created using the tool Draw.io. Due to the requirements, two different strategies were decided. A simpler version for customers wanting the ability to add static content to their app as well as a more advanced version for customers wanting the possibility of handling their own logic and updates for their content. In the following sections the company server refers to the server belonging to the company supplying the mobile application and the customer server belongs to a customer using the mobile application.

3.2.1 Static Content

Customers that require static content inside their application can easily add this content inside a convenient user interface as illustrated in Figure 3.1. Once saved, this configuration will be available to the mobile application and can also be changed in the future through the same process. However, the content will not be able to update itself but will require manual work from the customer through the user interface.

3.2.2 Dynamic Content

Inside the overview that can be seen in Figure 3.2 Customer 1 wants to add content inside the service application that can be updated without human interaction. To achieve this, the customer first creates a JSON file containing an array of elements which will be rendered as notes according to the documentation for using this dynamic content service. When the JSON file is complete, the customer then stores it inside their own server. To apply the configuration to the application, the customer needs to notify the company server that it should retrieve a new configuration for this customer. The company server will only request this new configuration if it is given a white-listed URL that it knows. By storing the JSON file on a customer server instead of sending it directly, the company server only has to trust specified customer servers and not individuals. The configuration can then be updated using the same URL, and this is done by merely repeating the final step in Figure 3.2 either when being notified of a change.

1https://about.draw.io/
3.2. System Architecture

Figure 3.1: Static configuration architecture

or by a regular period of time. Furthermore, since the customer server is supposed to receive requests from the company server and then share information, it would be useful from a privacy perspective if the customer server could first verify whether a request is genuine. If the request is deemed not genuine the customer server should deny the request before sharing any data. A possible approach to solving this would be to include signed digital signatures with each request which would allow the customer server to verify the signature before processing the request. This could be done by having the company server take the message to be sent and transform it into a hash followed by signing it with a secret private key (critically only known by the company server) to create a signed message signature. The signature is then included beside the original message within the request. When the request reaches the customer server, the signature can be verified by first decrypting the signature using the corresponding public key resulting in the correct originally signed hash only if the signature is signed with the legitimate private key. The customer server can then create a new hash from the received message using the same hash function as the company server and compare the two hashes. If the two is a match, it means that the signed message signature and its contents are genuine. One reason for using a fixed length hash instead of the original message directly is that cryptographic signing functions cannot be passed too large values. By transforming the message into a hash before signing it, the message used can be of arbitrary length since the resulting hash will always be of suitable length. Another reason is that if an adversary tampers with the request during transit, changing some part of the message in any way, then the new hash created at the customer server will not match the original hash even if the signature is valid. This means that the hashing process also guarantees that the message itself is genuine.

3.2.3 Rendering Content

After successfully storing the configuration using either the static or dynamic flow the customer will want to have this content rendered inside the mobile application. The flow of rendering user-customized content inside the mobile application is illustrated in Figure 3.3. When a customer user starts the application, it will request configuration information from the company server. Inside this server, every customer will have their configurations stored as an array of elements tied to their customer id if they used the static approach. For customers using the dynamic version, the server will have to reach out to the correct customer server and retrieve the configuration from there. Basically, when a customer requests their configuration, the company server will retrieve the correct configuration and send it back to the requesting customer application. Once retrieved, this customized content represented as a JSON-object will be parsed and rendered inside the mobile application.
3.2. System Architecture

Figure 3.2: Dynamic configuration architecture

Figure 3.3: Configuration usage
In this chapter the implementation of each part of the system is presented including all of the important decisions associated with the implementation.

4.1 Mobile Application

At the start of the implementation, it was decided to begin with the mobile application to verify that the proposed parsing method could work. The mobile application was implemented using the frameworks React Native and Expo for swift development and testing. The third-party library React Native Elements was included to enhance the styling and appearance of the application. To more easily handle data sharing and data storage third-party library Redux was introduced into the application. Furthermore, the third party library redux-thunk was also included to facilitate the usage of Redux alongside asynchronous HTTP requests. Testing during development was done on a physical Android device, and occasional testing was also done on a physical iOS device.

4.1.1 Components

Given the requirement that users were supposed to be able to choose from a set of predefined components, it was important to decide which components were necessary and how users should be able to customize and interact with these. Based on the notes scenario presented earlier, an initial set of basic components were constructed: CardSection, Headerblock, TextBlock, List, ListItem, Action, and Form. This set of components was later improved by including IconBlock, ColumnSet and the Link action-type. Many of these components were constructed to allow receiving customized styling through receiving specific arguments as props. Props are data objects that are passed into the components.

CardSection

The CardSection component is used to wrap other components with styling which makes them look like they belong together inside a card. The background color can also be configured through the background color prop.
4.1. Mobile Application

**HeaderBlock**

The HeaderBlock component renders a header inside the app consisting of an icon and a title. All icons labeled “FontAwesome” from expo vector icons[1] are available. Optionally additional text or an icon can be provided to the “right” prop which will add this to the right-end side of the header block.

**TextBlock**

TextBlock component is used for displaying text. It can be customized through the props `font-size`, `font-weight`, and `color`.

**IconBlock**

IconBlock can be used to include icons and optionally a text next to the icon. Props available for customization are `icon size`, `text` and `text size`.

**ColumnSet**

ColumnSet is used for controlling the layout of content. It requires an array of columns where each column must contain one or more items. The layout of these columns can be controlled through props `width`, `horizontal` and `vertical`.

**List**

Renders a list which iterates over each item inside the provided items array and sends them to the ListItem component. Two types of lists are available through the `type` prop, card section list and bullet list.

**ListItem**

The ListItem component handles every item inside a list and accepts text or a new array. In the case of a new array, ListItem will call the Parser function again recursively to render the correct component. In the case of a card section type list, each list item can also be configured with a title prop.

**Action**

Actions are meant to be ways for the user to add some kind of interaction with the generated content, such as adding a form which can receive inputs or a link which can be clicked. The kind of action that will be rendered is decided through the action type prop. The way in which this action is displayed can be configured through the props `text` and `icon`.

**Form**

In the case of a user wanting to add some kind of form, they will use the Form action-type. This Form component will display a form inside of a modal, where each input field use a sub-component. An example of such a sub-component is the TextInput component. The TextInput component displays a text input field which requires a title and a placeholder. As the name suggests, users may only enter text into this specific input.

4.1. Mobile Application

**Link**

Users wanting the ability to link to an external web page can do this by using the Link action-type and providing a URL. This clickable link can be presented as text, icon or both. Further customization can be added through the props text size, text style, icon size, and color.

**Parser**

To be able to produce any mobile application content at all from the JSON file, some form of parsing to interpret what code components to use had to be introduced. This parsing was implemented by traversing through every element in the JSON-file, one at a time, and sending them into a switch statement. Inside this switch statement, which component type to render was decided by the type property of the current element.

4.1.2 Authentication

Since users should only be able to see their own stored content, an authentication process was needed. It was therefore decided to divide the application into two screens, the initial auth screen for signing in to the application and the configuration screen showing the user-customized content available to the user after signing in. To more easily create this flow in the application the third party library react-navigation was introduced to the project. The result of this screen implementation is shown in Figure A.2a and Figure A.2b inside Appendix A. One important security aspect of mobile security is where sensitive information such as access tokens are stored. After some research, it was decided to save the access token using Expo’s Secure store feature which in turn uses the underlying current best practices for secure storage on Android and iOS respectively.

4.1.3 Mobile Application - Evaluation

By specifying some of the available constructed components in a JSON file, an initial test of producing notes into an application was conducted. Since the JSON file quickly becomes large if many components are included, the test only used a few components to make it easier to show the concept in figures. The aim of the test was to render a header block on the screen along with a list containing two list items. The resulting rendered application from the test can be seen in Figure 4.1. The test was successful, and parsing of JSON files was now available, but the configuration was currently stored locally within the app itself. The next step was to store these configurations inside a web server. An example configuration JSON file and additional examples of rendered use cases can be seen in Appendix A.

![Figure 4.1: Content rendered from parsed JSON file.](image-url)
4.2 Web Server

The web server was supposed to store customer configurations and supply them to the mobile application when needed. This seemed like a typical scenario where a REST-API could be used to handle and respond to HTTP requests from clients such as web browsers and mobile phones.

The requirements on the API were to store JSON-configurations and tie them to a user ID. When an authenticated user asks for their configuration, they should be able to retrieve it. There exist many different languages and frameworks for building web servers, and primarily three different options were investigated: Node with the framework Express, Java with the framework DropWizard, and Kotlin with the framework Ktor. After some basic initial testing of all three alternatives, a decision was made to use Node with Express with the motivation being fast development which results in more time available for better functionality and security testing and in turn a more exciting thesis. The alternatives were deemed too time-consuming for the thesis because of lacking prior knowledge of these techniques.

An initial architecture for the company web server was created based on the requirements, as shown in Figure 4.2. The client side such as a mobile app or a website communicates with the server by sending HTTP requests to routes which the server listens to. When a user wants to share some arbitrary data with the server they will send a POST-request, the third party library BodyParser is used to help Express easily get a hold of this data inside a route handler. Two additional middlewares, named helmet and cors, were also included but omitted in the figure. For security reasons the third-party middleware package helmet was applied which helps protect Express apps through various security best-practices. A security feature of browsers known as cross-origin resource sharing (CORS) also needs to be configured inside web servers to allow requests from browsers to be accepted. This was done by using the third-party middleware package cors. Another convenient package which was used during development was nodemon which automatically restarts the development server whenever a code change is saved.

4.2.1 Routing

Inside the server there exist two sets of routes, authentication routes (auth routes) and configuration routes. These are reasonably easy to understand, auth routes are responsible for handling user sign up and sign in whereas the configuration routes handle user requests for creating, fetching and updating configurations. The configuration routes are protected by a middleware which requires users to be authorized before they may access configuration resources. Third party package Axios was included to make the implementation of HTTP requests easier which was used in the routes that requested data from customer servers for dynamic configurations. To address the requirement of customers being able to verify whether a request was genuine, it was decided to include a signed digital signature in customer server requests. This was done by first taking the message to be sent and transforming it into a fixed length hash. The hash is then signed with a secret private key which creates a signed message signature that can be sent along with the message. The customer server is then able to verify the origin by using the corresponding public key.

4.2.2 Authentication

Given the requirement that configurations should only be available to the owning customer, there must exist some form of authorization process of requests to prevent data leakage. For easier management of authentication strategies as well as authorization the third party library passport was included which can be thought of as a sort of gatekeeper for the server. Every time a request wants to interact with a protected route inside the server passport will inspect
the request and determine whether it should be accepted or denied. When a user signs up for the application passport will require them to supply an email and a password for setting up the local strategy authentication. When received inside the server this password is then salted and hashed by using the hashing third-party library bcrypt-nodejs before it is stored inside the database. Whenever a user visits the mobile application, they are requested to sign in by using their signed up email and password. This was implemented by using the third-party package passport-local. After a user has successfully signed in, they still need to be authenticated on each request concerning valuable resources. For this purpose, JWT authentication was chosen based on its scalability, flexibility and easier CSRF-security compared to the common alternative cookies. Implementation of JWT authentication was done by using third-party packages jwt-simple, passport-jwt, and moment for setting the expiration time of the token. Every time a user sign in to the application they are provided with a new JWT which can then be attached to each request allowing passport to verify which signed in user issued the particular request. By using this strategy, only users that are signed in will be allowed to access anything related to configurations and users can only access their own configurations. To clarify, users sign in to the application by using their email and password and when signed in their requests are authenticated using the JWT they received when they signed in.
4.2.3 Database

The decision to use MongoDB as the database was made because it is easy to set up and integrate with a Node back-end. Another reason was that the database as a service company Mlab offers developers to create MongoDB sandbox databases hosted on their cloud for free, which seemed useful for development. Since validation and easy interaction with the database was a priority, it was also decided to include the object data modeling third-party library Mongoose. Mongoose introduces the concept of schemas to the application which basically means a blueprint of which objects and what properties are allowed to be stored inside the database. This offers a benefit in the form of protection against NoSQL injection since Mongoose will not accept users to store something that has not been specifically allowed. A sandbox database was set up on Mlab and connected to from the server.

4.2.4 Web Server - Evaluation

After implementing the server, it was time to test if a user could use the mobile application and interact with the server. To perform this test the hosting platform Heroku was used to deploy the server allowing the mobile application to make requests to it. Heroku was chosen because of prior knowledge using this platform and because it offers free sandbox hosting. Having successfully deployed the server another test was done to verify that users could sign in to the mobile application and fetch only their own configuration from the server. After creating two users and giving them unique configurations the test was easily verified as successful since both users were able to sign in and only access the content tied to their id. Figures 4.3a and Figure 4.3b illustrate how the application renders different content depending on how the users have configured their own application. Furthermore, retrieving configuration resources without providing a valid JWT was tested and resulted in an error message politely telling the user that they were unauthorized to make the request.

![Figure 4.3a](image1.png) ![Figure 4.3b](image2.png)

(a) Content for a user. (b) Content for another user.

Figure 4.3: Figure illustrating customized content for two users.

4.3 User Interface

The user interface had requirements of supporting authentication and static as well as dynamic configuration possibilities for users.
4.3. Authentication

The implementation of the user interface was initialized by creating a sign up as well as a sign in page where users could register and sign in. These two pages are showcased in Figure B.1 and Figure B.2 inside Appendix B. Using the same technique as for the mobile application, authentication was implemented having users sign in by entering email and password and then storing the returned JWT locally. This JWT could then be attached to all future requests and allow the server to identify which user issued the request. Since the same server is used by both the user interface and the mobile application the same user account is used on both platforms. Customers are only allowed to register for the application through the user interface.

4.3.2 Static Configuration

When customers were able to sign up and sign in to the application successfully, the next task was the static configuration page. By using the library react-json-view functionality of viewing and manipulating a static configuration was swiftly implemented along with a button allowing users to save their configuration to the server. Figure B.3 inside Appendix B illustrates the default static content configuration page before the user has changed or saved their configuration. Once the user saves a configuration, it will immediately be available the next time an end user sign in to the mobile application.

4.3.3 Dynamic Configuration

The user interface also needed to allow customers setting up a reference to their own server from which the required configuration JSON-file could be retrieved. Another tab was implemented inside the user interface which allowed users to add cards to their accounts. A card was supposed to represent a piece of configuration grouped together into a functioning component of its own including all of the functionality required to fulfill a use case scenario. A user could then have several cards being retrieved and used inside their application. This approach allowed users to more easily split up the added content instead of having to create a giant JSON-file containing all the functionality in the same file. Cards could be added through a form requiring a name, type, and a URL. The name was used as a unique identifier for each card to help the user remember which cards they had set up. The type could be used for grouping cards into different categories, such as locations or specific end-user groups. The URL was supposed to be the link to the customer API which would be used for retrieving the configuration of each card when needed. This needed some additional thought since the given URL potentially could be malicious or faulty. By requiring the company to explicitly allow a known base-URL for each customer only one domain would be available for them to use. This way, not trusted URLs for a customer would not be accepted and therefore not used for requesting data hence providing some protection. Furthermore, two buttons were added to each row representing a configured card, a preview button which opens a modal where the received configuration can be inspected and a deletion button used for removing the card.

4.3.4 User Interface - Evaluation

To test the static configuration functionality, a user was signed up and then configured using the user interface shown in Figure 4.4. After saving the configuration and logging in to the mobile application the correct content successfully rendered inside the application which can be seen in Figure 4.5. This content could also be manually manipulated by changing the configuration inside the user interface, saving it and then retrieving this updated content in the mobile application. The dynamic configuration flow could not yet be tested since it required a customer server providing the configuration for each card.
4.4 Customer Server

A basic customer server example which could serve cards to the company server in the form of configuration snippets (cards) was needed to verify the behavior of the system. To speed up development, much of the company server structure was reused for the customer server.
4.4. Customer Server

Authentication and user interface was skipped since it did not seem necessary to prove the concept. The framework Express along with third-party libraries Axios and BodyParser was used for routing. Cloud service provider Mlab was used for hosting a MongoDB instance where cards could be stored and third-party package Mongoose to interact with the database. Since there was no user interface, the API development environment Postman was used to push data to routes directly. The initial card configurations were added to the database this way. At this stage, a JSON configuration snippet for each of the eighth test case scenarios was created and placed inside the customer server database to be available when the company server requested them. Once again, the hosting platform Heroku was used to deploy the customer server to create a more realistic scenario where both servers were running on a somewhat real production level. As a final touch, the customer server was supposed to verify the message signature of incoming requests before processing them. This was done by extracting the received signature and decrypting it into the supposedly original hash using the company server public key. After that, using the same hash function as the company server, a new hash can be created from the received message and compared with the original hash and if they match the request is considered to be genuine. By using this technique, only requests providing a signature signed using the correct secret private key and a message that has not been tampered with are able to attain information from the customer server.

4.4.1 User Interface - Further Evaluation

Eighth different cards were now available for retrieval at the customer server, and it was thereby possible to test the dynamic flow of the user interface. After signing in a user and switching to the dynamic tab, four cards were added to the user. The retrieved configurations seemed correct which was verified by using the configuration preview function. However, verification that the mobile application worked properly with this new flow was also needed. After signing in to the mobile application with the same user, the correct cards rendered and everything seemed to be working. Users could now choose which cards they wanted to render inside their mobile application through the user interface, and the evaluation procedure can be seen divided into figures inside Appendix B. However, user interaction with the mobile application such as adding new content through forms did not yet trigger any update or change.

4.4.2 Updating Content

To be able to update the card configuration retrieved from the customer server the mobile application needed to take data as well as information about what element that was used for submitting. The mobile application could then pass this information to the company server which could forward this information to the customer server. In this process, at the company server level, the user device information containing potential personal information would not be recorded or forwarded to the customer server. When the customer server receives input from the company server some form of logic must be implemented that will handle the data and react to it accordingly. Two different handlers were implemented inside the customer server to illustrate example behavior. Initially, a handler for the notes scenario was implemented which would take new notes submitted from users and add them inside the notes card configuration. Next, a handler for the service desk scenario was implemented which would similarly take new issue reports from users and add them to the service desk card configuration. The mobile application was set up to fetch the new configuration after receiving a response that the submission was successful.
4.4.3 External Data

Some of the scenarios, such as displaying daily restaurant menus or free parking spaces, require data that changes over time. This means that the customers must come up with a solution to retrieve and provide updated data to be able to use it inside the mobile application. To showcase a possible solution an example was implemented for displaying the number of current free parking spaces of three local parking garages. The implementation made use of an open API which the local municipality offers where anyone can retrieve the current number of free parking spaces in a parking garage. Whenever the company server requested the customer server for the parking card configuration the customer server would reach out to the open API and retrieve the correct current number. When receiving this number for each of the three parking garages the parking card configuration was updated accordingly before sending the parking card back to the company server which passes it on to the mobile application.

4.4.4 Customer Server - Evaluation

The customer server was supposed to serve the company server with cards. It should also be possible to update these cards based on user interaction as well as insert appropriate external data when needed. To test that cards could be updated by user interaction the mobile application was configured (through the user interface) to use the two cards which had handlers implemented for them inside the customer server. These two cards were the notes and service desk cards. Having signed in to the mobile application, a note was initially added and thereafter an issue was reported. In both cases, the actions were quickly reflected in the mobile application, and the content could definitely be interacted with and updated by the users. To test usage of external data the parking card was added to the user inside the user interface. After once again signing in to the application it was possible to receive updates for the current number of parking spaces available inside the three local parking garages. Retrieving updated data was initially done manually by pressing a refresh button triggering a new get request to the customer server through the company server which resulted in new data being fetched and served. Later on, the user interface instead included a check-box when adding cards that when checked resulted in the mobile application automatically updating itself with new data periodically when using the card. The system now supported updates through user interaction as well as usage of external data along with automatic periodic updates.
To evaluate the security of the mobile application it was decided to conduct a case study. The plan of the case study was to identify suitable mobile application security testing frameworks and apply them to the implemented system. The goal was to gain further understanding of how cross-platform mobile applications can be security tested and evaluated in the current technological landscape as well as an understanding of the security level of the implemented system. The following sections describe the methodology used and the results of this case study.

5.1 Method

After searching for cross-platform security evaluation resources for a while it seemed like the mobile application security field lacked good alternatives for cross-platform applications compared to the large scope and thorough explanations of the OWASP Mobile Security Testing Guide (MSTG) framework. Unfortunately, the MSTG did not target cross-platform applications but rather native applications developed for Android or iOS. At the same time, much of the information inside the MSTG seemed valuable for testing a cross-platform application and given that cross-platform applications are a fairly new technology finding security frameworks targeting them proved difficult. Therefore, a decision was made to try using the MSTG to evaluate the security of the mobile application part of the system to see if it could be useful for cross-platform applications despite its primary focus being native mobile applications. The Mobile Application Security Verification Standard (MASVS) consists of seven primary sections where each section has defined requirements that are expected of a mobile application to confine to level 1 or level 2 of security for that particular area. There also exists the eighth section regarding resilience against reverse engineering and special client attacks which can be used as an additional layer of security concerning more advanced attacks. The eighth section was deemed out of scope for this thesis and is therefore not included in the case study. The remaining seven areas included were:

1. Architecture, Design and Threat Modeling
2. Data Storage and Privacy
5.1. Method

3. Cryptography
4. Authentication and Session Management
5. Network Communication
6. Platform Interaction
7. Code Quality and Build Setting

The system implemented in this thesis was categorized as a normal application without particularly sensitive information (such as payment information or medical records). Because of this, only level 1 (L1) of the security verification requirements were analyzed which, according to the MASVS, is suitable for typical mobile applications. The evaluation of each area was done by first looking at the verification requirements for each of the sections in the MASVS and then reading the related parts in the MSTG for both Android and iOS. After reading these parts the way of verifying the requirements in a cross-platform system was determined through a reasoning process. The results of this reasoning and its subsequent methodology actions for each area are described in the following sections.

5.1.1 Architecture, Design and Threat Modeling

The first part of the evaluation was well suited for testing a cross-platform application since requirements on architecture and design are not specific for Android or iOS and should be the same no matter the platform. The requirements could be decided as fulfilled or not through static analysis by reasoning about the system architecture and general implementation decisions.

5.1.2 Data Storage and Privacy

Data storage and privacy seemed tricky to evaluate in a cross-platform system since Android and iOS have different ways of storing data and handling processes that need to communicate with each other. Furthermore, both platforms offer several storage options each. In the mobile application, the data considered sensitive were the stored access token, the password being submitted during sign in, and personal identifying data. Because of this, the requirements pertaining to stored sensitive data in this case only had to consider how the access token was handled in the system since it was the only sensitive data being stored in the mobile application. The requirements were verified through static analysis of the implementation and a touch of dynamic analysis.

5.1.3 Cryptography

Evaluating cryptography for the mobile application was not obvious. When the mobile application communicates with the web server the communication protocol used depends on the mobile device operating system version and the web server configuration. Since Heroku configured the company server in the implemented system, it was not obvious how the mobile application and the company server communicated with each other. After some research, it was verified that Heroku automatically set up TLS on their hosted servers, and TLS uses both asymmetric and symmetric encryption. Through further dynamic analysis of the company server by accessing it inside a web browser it was verified from the browser security tab that it used the TLS 1.2 version protocol, as shown in Figure 5.1. Therefore, whenever the mobile application communicates with the company server, it uses cryptography since the company server demands it (as long as the mobile operating system supports TLS 1.2 which is true for Android 4.4.2 and iOS 5.1.1 versions and above). Furthermore, the algorithms (cipher suite) which TLS 1.2 in this case used were also identified. Further confirmation on the TLS
implementation of the company web server was verified through the SSL Labs server test\footnote{https://www.ssllabs.com/ssltest/}. Beside the cryptography concerning communication, the mobile application also relied on cryptography inside the company server for signing and verifying access tokens through the usage of the HMAC SHA256 algorithm. The requirements were verified using this gathered information generated from both static and dynamic analysis.

![Figure 5.1: Security tab for the company server inside Firefox.](image.png)

### 5.1.4 Authentication and Session Management

Authentication and session management requirements are aimed at the remote endpoint of the mobile application which conveniently means it is as easy to evaluate cross-platform systems as platform-specific systems. In the implemented system the company server is the remote endpoint being inspected. The requirements could be determined through static analysis reasoning about the authentication architecture and implementation details of both the mobile application and the company server.

### 5.1.5 Network Communication

After carefully reading the instructions in the MSTG a decision was made to use dynamic analysis and conduct an experiment to inspect the network communication of the mobile application. Since the experiment needed to be set up differently depending on the target platform a decision was made to target Android because of previous experience using Android Studio and emulated Android devices. First Android Studio was installed and used to create an emulated Pixel 3 Android device running Android Version 7.0 (API level 24). Then the software Android Debug Bridge (ADB) was used to gain root access on the device. Having root ADB was again used to install the software tool tcpdump on the device which can be used to capture TCP and IP packets and also to store this information inside a file. To be able to, later on, analyze this stored packet information another tool called Wireshark was installed on the main computer. Wireshark is a packet analyzer which is very useful for analyzing network traffic. Using these tools the experiment was carried out by having tcpdump record all packets in a file while performing all available actions inside the mobile application. After recording network packets while using the application the file containing the information was transferred to the computer from the emulated device. The network traffic could then be analyzed using Wireshark to determine what protocols were used during communication.
5.2. Results

5.1.6 Platform Interaction

Platform interaction is also tricky for cross-platform systems since it is not obvious to know which underlying processes that Android and iOS use when the code is written at a higher abstraction level targeting both platforms. With an advanced application, the analysis could be difficult to conduct since the person analyzing must know which underlying processes the abstraction layer uses to perform the actions. Regarding the implemented system used in this thesis, it is somewhat easy to analyze because it is a simple prototype application not using any special platform mechanisms such as internal communication between apps. The requirements could be decided as fulfilled or not through static analysis by reasoning about the system architecture and general implementation decisions.

5.1.7 Code Quality and Build Setting

Since the implemented system was meant as a prototype, it was never really set up to be used in a real production build setting. Because of this the implemented system of course immediately fails many of the requirements pertaining to build setting. The requirements could once again be decided as fulfilled or not through static analysis by reasoning about the system architecture and general implementation decisions.

5.2 Results

Here the results of applying level 1 (L1) of the OWASP Mobile Application Security framework to the system are presented. For each area being evaluated a table will show the requirements and whether they are considered to be fulfilled. Each requirement has a unique number (#) which are referred to when explaining why the requirement was considered fulfilled or not. Furthermore, each requirement has a description and two columns named Expo and React Native (RN). The Expo column illustrates whether the implemented system in its current form is considered to fulfill the requirement. In the Expo column, a check mark (✓) means that the requirement is considered fulfilled, a cross (✗) means that the requirement is not fulfilled, and a dash (-) means that the requirement is not considered applicable for the analyzed system. The RN column represents whether the requirement is believed to be achievable in a pure React Native version of the system with available native code additions. In the RN column a check mark (✓) means that the requirement is considered achievable, a cross (✗) means that the requirement is not thought to be achievable, and a dash (-) means that the requirement is not considered applicable for the analyzed system. The RN column is hypothetical and its markings have not been properly tested or verified.

5.2.1 Architecture, Design and Threat Modeling

The results of evaluating level 1 architecture, design and threat modeling requirements are shown inside Table 5.1. Starting with #1.1, given that the mobile application is rather small, all mobile application components were quickly identified and verified to be needed. As for #1.2, in the implemented system the remote endpoint is the company web server where every route handler carefully inspects the payload and denies the request if there is something wrong which satisfies the requirement. A high-level architecture is illustrated in the thesis and security has been addressed which satisfies #1.3. Finally, addressing requirement #1.4, data considered sensitive in the system are personal identifying data, passwords and access tokens which are all identified.
5.2. Results

Table 5.1: Architecture, design and threat modeling requirements.

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Expo</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>All app components are identified and known to be needed.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1.2</td>
<td>Security controls are never enforced only on the client side, but on the</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>respective remote endpoints.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>A high-level architecture for the mobile app and all connected remote</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>services has been defined and security has been addressed in that architecture.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>Data considered sensitive in the context of the mobile app is clearly</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>identified.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.2.2 Data Storage and Privacy

The results of evaluating level 1 data storage and privacy requirements are shown inside Table 5.2. Starting with #2.1 and #2.2, the access token stored inside the mobile application is stored by using Expo’s Secure Store API. This, in turn, means that on iOS the token is stored using the keychain services\(^2\) and on Android the token is stored inside SharedPreferences encrypted with Android’s keystore system\(^3\). These procedures are considered the current best practices for storing sensitive information on the respective platform and are both described in the MSTG. Furthermore, only a few error messages are logged in the mobile application, never sensitive data which satisfies #2.3. There is also no data being sent to third parties, only to the company server meaning #2.4 is satisfied. Asserting that the keyboard cache is disabled for the password input inside the mobile application was done through a dynamic analysis where the application was used to sign in on both an Android device as well as an iOS device. In both cases, the email field provided cached suggestions, but the password field did not, which according to the MSTG means that the keyboard cache has been successfully disabled for the input field and in turn that #2.5 is satisfied. Also, the system does not communicate with any other process on the mobile device (Inter Process Communication) and therefore #2.6 is not applicable for the analyzed system. It was also verified that no sensitive data ever is exposed in the user interface since the password and access token never are shown inside the application, satisfying #2.7.

Table 5.2: Data storage and privacy requirements.

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Expo</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>System credential storage facilities are used appropriately to store sensitive data, such as PII, user credentials or cryptographic keys.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2.2</td>
<td>No sensitive data should be stored outside of the app container or system credential storage facilities.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2.3</td>
<td>No sensitive data is written to application logs.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2.4</td>
<td>No sensitive data is shared with third parties unless it is a necessary part of the architecture.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2.5</td>
<td>The keyboard cache is disabled on text inputs that process sensitive data.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2.6</td>
<td>No sensitive data is exposed via IPC mechanisms.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2.7</td>
<td>No sensitive data, such as password or pins, is exposed through the user interface.</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

\(^2\)https://developer.apple.com/documentation/security/keychain_services
\(^3\)https://developer.android.com/training/articles/keystore
5.2.3 Cryptography

The results of evaluating level 1 cryptography requirements are shown inside Table 5.3. The mobile application makes use of TLS 1.2 when communicating with the company server. TLS is a hybrid cryptographic system that uses asymmetric cryptography to exchange a securely randomized key which then can be used for symmetric encryption in further communication meaning that #3.1 is satisfied. TLS 1.2 is a proven protocol which is currently recommended for communication (together with TLS 1.3), satisfying #3.2. TLS 1.0 is deprecated, and TLS 1.1 is planned to be marked as deprecated in 2020 whereas TLS 1.2 will not be deprecated in the foreseeable future, meaning that #3.4 is satisfied. The algorithms used by TLS in the implemented system were also analyzed. During the initial handshake process of TLS asymmetric encryption with the ECDHE RSA algorithm is used for exchanging a key that can be used for symmetric encryption. After exchanging the key, symmetric encryption with the AES128 GCM algorithm is used for further communication. The SHA256 hashing algorithm is also used during communication to provide integrity checks. This cipher suite is known as ECDHE-RSA-AES128-GCM-SHA256 and is one of the recommended cipher suites for current services, providing the highest security which in turn satisfies #3.3. The company server also received the highest possible grade of A+ on a scale A-F in the SSL Labs server test. The cryptographic keys are not reused for multiple purposes, satisfying #3.5. Furthermore, the HMAC SHA256 algorithm used when signing and verifying access tokens is cryptographically secure since it uses SHA-256 which is considered a secure hashing algorithm thereby also satisfying #3.2, #3.3 and #3.4. At the time of writing Expo does not offer a sufficiently secure crypto API for generating random numbers which means that #3.6 cannot be satisfied, but there does exist a feature request for Expo addressing the need to solve this. However, the randomization should not be a problem in the implemented system since the mobile application’s random numbers are not used for any cryptographic reasons. If the system used pure React Native without Expo, it could use existing third-party libraries offering sufficiently secure crypto APIs for generating random numbers which would satisfy #3.6.

Table 5.3: Cryptography requirements.

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Expo</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3.2</td>
<td>The app uses proven implementations of cryptographic primitives.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3.3</td>
<td>The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3.4</td>
<td>The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3.5</td>
<td>The app does not re-use the same cryptographic key for multiple purposes.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3.6</td>
<td>All random values are generated using a sufficiently secure random number generator.</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

4https://arstechnica.com/gadgets/2018/10/browser-vendors-unite-to-end-support-for-20-year-old-tls-1-0/
5https://wiki.mozilla.org/Security/Server_Side_TLS
6https://expo.canny.io/feature-requests/p/crypto-api
5.2.4 Authentication and Session Management

The results of evaluating level 1 authentication and session management requirements are shown inside Table 5.4. Users are required to sign in to the application using a username and a password to gain access to the system which means that #4.1 is satisfied. After a user has signed in, stateless token-based JWT authentication is used. All JWTs are signed and verified using the HMAC SHA256 algorithm which is known as a secure algorithm, satisfying #4.3. Since token-based authentication is used no session exists which means that #4.2 and #4.4 are not applicable for the system. Requirement #4.5 is satisfied since a password policy requiring passwords to be longer than four characters is enforced at the company server. There does not exist an implemented mechanism to protect against excessive credential submission which means that #4.6 is not fulfilled. However, this mechanism could be implemented regardless of using Expo or React Native by using third-party package node-rate-limiter-flexible which is recommended by the Express project as a security best practice for Express servers to prevent brute force attacks. Access tokens expire 60 minutes after creation which means that they will thereafter no longer be accepted by the server and that #4.7 is satisfied. This invalidation mechanism was verified by lowering the expiration time to 30 seconds and trying to access resources with the access token after 30 seconds, which was denied correctly. Users can obtain new access tokens by signing in to the application.

Table 5.4: Authentication and session management requirements.

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Expo</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>If the app provides users access to a remote service, some form of</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>authentication, such as username/password authentication, is performed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at the remote endpoint.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>If stateful session management is used, the remote endpoint uses</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>randomly generated session identifiers to authenticate client requests</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>without sending the user’s credentials.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>If stateless token-based authentication is used, the server provides a</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>token that has been signed using a secure algorithm.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>The remote endpoint terminates the existing session when the user logs out.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4.5</td>
<td>A password policy exists and is enforced at the remote endpoint.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>4.6</td>
<td>The remote endpoint implements a mechanism to protect against the</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>submission of credentials an excessive number of times.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.7</td>
<td>Sessions are invalidated at the remote endpoint after a predefined period</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>of inactivity and access tokens expire.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.2.5 Network Communication

The results of evaluating level 1 network communication requirements are shown inside Table 5.5. Data sent and retrieved for the mobile application was inspected using the packet analyzer Wireshark. During the inspection, it was verified that all sent and received application data was encrypted and sent over the network using the TLS 1.2 protocol, hence #5.1 is satisfied. This is illustrated in Figure 5.2a where the traffic has been filtered to only display TLS packets. Looking at the plain HTTP traffic in Figure 5.2b, only a private message being repeated between the emulated device and the host machine is recorded (the cause of this message was that Expo was not updated to the latest version), no real communication used HTTP. A sample of the raw, unfiltered data is illustrated in Figure C.1 inside Appendix C.

7https://github.com/animir/node-rate-limiter-flexible
5.2. Results

(a) Network traffic filtered on TLS packets.

(b) Network traffic filtered on HTTP packets.

Figure 5.2: Figure illustrating the network traffic for the mobile application during usage.

As stated in the cryptography section the TLS 1.2 protocol is recommended and considered one of the current best practices for communication meaning that #5.2 is also satisfied. However, the mobile application does not verify the X.509 certificate resulting in #5.3 not being fulfilled. The MSTG provides explanations on how to do this for Android and iOS respectively, but no solution for doing this using Expo was found. There does exist a feature request \(^8\) for adding SSL pinning to Expo and in the future it could become possible to satisfy #5.3. In a pure React Native project, it should be possible to implement verification of the X.509 certificate for Android and iOS respectively by using native code inside each platform’s independent code section.

\(^8\)https://expo.canny.io/feature-requests/p/ssl-pinning
Table 5.5: Network communication requirements.

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Expo</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>5.2</td>
<td>The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>5.3</td>
<td>The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.</td>
<td>✗</td>
<td>✔</td>
</tr>
</tbody>
</table>

5.2.6 Platform Interaction

The results of evaluating level 1 platform interaction requirements are shown inside Table 5.6. Since the mobile application does not request any permissions #6.1 is satisfied. Data that users input through the UI is validated where necessary. However, the dynamic configuration retrieved over the network from the customer server is only validated at evaluation to contain known tags and types, and malicious customers could potentially try to provide faulty configurations. But given the system design, this should not be a problem since the configuration is only allowed to decide component styling and text content inside the mobile application which means that a faulty configuration should only result in content being rendered malformed or not rendered at all. The one exception is the links component that does allow customers to provide any URL they want which will be opened whenever clicked. No sanitation of such URLs is performed which could be a potential attack vector allowing users to provide malicious URLs and because of this #6.2 is not fulfilled. However, this URL will only be presented to the specific customer’s end users. The link component uses the Expo component WebBrowser which in turn use ChromeCustomTabs for Android and SFSafariViewController for iOS, both of which implement their own security mechanisms to protect users from malicious sites and thereby offers some protection even if the URL itself would be malicious. The application does not export any functionality through custom URL schemes or IPC facilities resulting in #6.3 and #6.4 not being applicable. WebViews are also not used inside the mobile application in favor of the WebBrowser component which means that #6.5, #6.6, and #6.7 are not applicable for the system. Finally, object deserialization is also not used in the system resulting in #6.8 not being applicable.

5.2.7 Code Quality and Build Setting

The results of evaluating level 1 code quality and build setting requirements are shown inside Table 5.7. The mobile application does not have a signed finalized release build, has not been built in release mode and have not removed debugging symbols which means that #7.1, #7.2, and #7.3 are not fulfilled. It is possible to satisfy #7.1, #7.2, and #7.3 during the build process using both Expo and React Native but it was not considered important for this project knowing that the implemented system is a prototype. Since the application run in a development environment, it does log verbose errors and debugging messages as intended, not fulfilling #7.4, but this could also be fixed using both Expo and React Native. However, all third-party components used by the mobile app are identified and checked for known

9https://developer.chrome.com/multidevice/android/customtabs
5.2. Results

Table 5.6: Platform interaction requirements.

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Expo</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>The app only requests the minimum set of permissions necessary.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>6.2</td>
<td>All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>6.3</td>
<td>The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6.4</td>
<td>The app does not export sensitive functionality through IPC facilities, unless these mechanisms are properly protected.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6.5</td>
<td>JavaScript is disabled in WebViews unless explicitly required.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6.6</td>
<td>Webviews are configured to allow only the minimum set of protocol handlers required (ideally, only https is supported). Potentially dangerous handlers, such as file, tel and app-id, are disabled.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6.7</td>
<td>If native methods of the app are exposed to a WebView, verify that the WebView only renders JavaScript contained within the app package.</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6.8</td>
<td>Object deserialization, if any, is implemented using safe serialization APIs.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

vulnerabilities by using the command line tool snyk\(^{11}\) fulfilling #7.5. Snyk found five vulnerabilities inside Expo which unfortunately cannot be handled since Expo is the core of the entire application and cannot be replaced. However, Expo is continuously evolving and updated which means that the security issues will probably be fixed rather soon. The application also catches and handles possible exceptions which fulfill #7.6. Furthermore, error handling logic always denies access by default and JavaScript allocates and frees memory automatically through garbage collection resulting in #7.7 and #7.8 being fulfilled. No free security features are activated which means that #7.9 also fails. These free security features do not seem available for Expo apps but do seem available when using React Native.

Table 5.7: Code quality and build setting requirements.

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Expo</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>The app is signed and provisioned with a valid certificate, of which the private key is properly protected.</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>7.2</td>
<td>The app has been built in release mode, with setting appropriate for a release build.</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>7.3</td>
<td>Debugging symbols have been removed from native binaries.</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>7.4</td>
<td>Debugging code has been removed, and the app does not log verbose errors or debugging messages.</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>7.5</td>
<td>All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>7.6</td>
<td>The app catches and handles possible exceptions.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>7.7</td>
<td>Error handling logic in security controls denies access by default.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>7.8</td>
<td>In unmanaged code, memory is allocated, freed and used securely.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>7.9</td>
<td>Free security features offered by the toolchain, such as byte-code minimization, stack protection, PIE support and automatic reference counting, are activated.</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>

\(^{11}\)https://snyk.io/
6 Discussion

This chapter contains discussion regarding results, method, future development and the work in a wider context.

6.1 Results

In this section, the results of the system design, implementation, and case study will be discussed.

6.1.1 System Design

Having implemented the designed system, it was verified to be successful in meeting the set requirements stated at the beginning of the design phase. However, after conducting the security evaluation through the case study, it seems like it would have been beneficial to read about the security requirements before designing the system. Security was addressed in the system design but not deeply enough to fulfill all of the level 1 requirements of the MASVS. By knowing what the security requirements were the system design could have been improved in many of the case study areas and thereby achieved a better score for the MASVS. A keynote to take away from this project is to include the security requirements in the system design phase for assuring that level 1 security is achieved if a project wants to adhere to the MSTG requirements. Nonetheless, the design proved very useful when implementing the system and seems to have done its main job in guiding the implementation to achieve the desired system. Furthermore, many of the security requirements were deemed as fulfilled which inclines that the design nonetheless provided some of the key security parts.

6.1.2 Implementation

Guided by the system design, the implementation generated good results. Evaluation of the mobile application proved that it was able to render all of the eighth different use-case scenarios which were constructed during the design phase. The mobile application also worked seamlessly well on both Android and iOS without having to perform any platform specific coding except for a small amount of targeted styling of the available components. Using
6.1. Results

Expo resulted in rapid prototype development but at the cost of production level value. The mobile application did not do that well in the production-oriented security requirements of the case study which possibly could have been improved by using pure React Native. At the same time, the system might only have been partly finished or failed given the limited time frame if Expo had not been used. To summarize, the resulting prototype does generate value and insight as to if the general solution concept could work but lacks some of the production level parts. One could argue that the main focus of a prototype system is to test and verify the main concepts and ideas. Producing a production ready system is typically not expected of a prototype system. From the results, the decision to use Expo for the project seems reasonable and a viable choice considering the aim of the thesis.

The web server acted as the core of the system by handling all communication between the mobile application, UI web page, and the customer server. It was responsible for many of the core features of the service such as user authentication and authorization, storing and updating local configurations, as well as storing references to dynamic configurations and requesting them from customer servers. The implementation was done in several steps but did, in the end, provide all of the required features and kind of glued the whole system together. The evaluation of the web server showcased that authentication and authorization, as well as requested functionality for both static and dynamic configuration usage, was sufficient. Given that the web server was a large and critical part of the system the decision to use Node and Express for fast development and making use of previous knowledge seems like it was the right choice because of the time frame.

Regarding the user interface it does provide basic functionality for users to set up both static and dynamic configurations. However, it does not provide more functionality than what is needed, and there does not exist much guidance except for some small text instructions. Many enhancements could and probably should be made to help users use the feature in a real production setting. One example would be to provide users with proper guidance instructions by showcasing what components are available and how they can be used. Another example would be introducing the ability to upload a complete JSON-file instead of having to use the static configuration tool to get started. Nonetheless, the user interface does handle signing up and signing in alongside the requested configuration functionality in a convenient way.

Continuing with the customer server, it is important to remember that the customer server implemented in the prototype system is just one example of how the concept can be used. The system provides customers with the ability to create a wide variety of cards based on their own ideas which they can update and interact with. However, the implemented customer server does showcase implementation examples of essential requested functions such as handling end-user inputs, automatic updates, and integrity signature verification. Even if the customers prefer some other kind of API structure or techniques, the concept of how to handle this commonly requested functionality can be quite helpful.

6.1.3 Case Study

The purpose of the case study was to investigate how cross-platform mobile applications can be evaluated with regards to security and get a feel for how secure the implemented system was. The results present evaluation information about all possible parts of a mobile application which makes it possible to gain an understanding of the general security level of the application as a whole. The Expo application does meet many of the requirements, and it looks like it could pass all requirements for level 1 if the project was migrated to a pure React Native version with some security additions to the native parts. At the same time, it currently does not seem possible to achieve true MSTG level 1 security in a mobile application.
6.2 Method

without the possibility of including native code. The reason for this is that some features such as certificate pinning and cryptographically secure randomization algorithms are not (yet) available through the JavaScript abstraction layer. Even though the MSTG was aimed at native applications, it proved very useful for identifying which parts of the application that needed to be inspected for security reasons and provided guidance on what kind of testing approaches that was suitable. Furthermore, the specified requirements for each area seemed just as appropriate for cross-platform applications as for native ones. The MSTG, therefore, proved to be useful for security testing mobile applications in general, not just for native applications, thanks to the in-depth testing descriptions alongside verifiable requirements and the analysis provided by the authors.

The Expo application does pass most of the requirements, and the security level of the application does not seem too bad. In the future, as cross-platform techniques mature, it could perhaps become possible to achieve reliable security for regular mobile applications even without any native code. However, it should be noted that this analysis is based on a limited prototype system which does not include things like IPC mechanisms, WebViews and build setting tasks. There could be many more hidden problems to be discovered in these areas if they were to be inspected more thoroughly.

6.2 Method

In this section, the method for creating the system design and implementation as well as the procedure for the case study will be discussed.

6.2.1 Design

The design was created from initial requirements elicitation, followed by use case scenarios, as well as an initial test application and related work studies. In hindsight, the most critical thing missing in this process was that security requirements research was not receiving enough attention. If the MSTG had been identified and used during the early design stages, the resulting product would have done better in the security evaluation. However, the design did a good job in guiding the implementation phase, and it was never a problem knowing in what order things should be done. The design was naturally split up into different subsystems. This allowed focused work on one part at a time, instead of having to jump between parts which probably would have resulted in a slower development process. Every part of the design phase proved to be useful for creating a good design that could guide the implementation. The initial requirements elicitation guided the design on a high abstraction level whereas use case scenarios were beneficial for knowing which specific components the system needed to support. The initial test application helped gain an understanding of how components could be implemented and interacted with as well as what was possible. Furthermore, the related work studies inspired the core concept of the system in the form of the JSON file parsing technique.

6.2.2 Implementation

Implementation was separated into phases based on the subsystems identified during the system design. This allowed focused work on one part at a time which made for a smooth development process flow and easy prioritizing of the ordering. The decision to start with the mobile application followed by the company server was motivated by them being the core of the system. If the implementation for some reason stopped then at least the minimal viable product addressing the problem description would be in place. Luckily the implementation went smoothly, and the whole visualized system was implemented, but it felt safe having the most critical parts of the system in place first to have something to fall back on. The
available configuration components presented in this thesis are only examples of what could be included. If the system was implemented for production level usage, it could include additional components offering customers even more customization and value. Furthermore, the system was constructed by the author alone, and there have not been any external quality assurance processes conducted, which means the implementation should not in any way be considered externally quality assured.

6.2.3 Case Study

The initial plan for the case study was to scan the internet using Google Scholar, IEEE Xplore, and the Linköping University library to find several scientific articles addressing security evaluations for cross-platform mobile applications and compare them. Some articles addressing security evaluation of one specific area for one of the platforms were found, but no articles covering both platforms or one platform in great detail was found. After some additional research, the OWASP community was identified as a possible resource. They offered a recently published framework called the Mobile Security Testing Guide (MSTG) for security evaluation of native mobile applications, meaning it targeted both Android and iOS. The MSTG is written collectively by the OWASP community through the open source platform GitHub. This allows anyone willing to contribute to the guide by creating a pull request containing their additions. However, these additions must be approved by the OWASP people responsible for the project before actually being included in the guide. Since the MSTG is written using open source anyone interested can create issues and send suggestions for improvements on GitHub, which means that the project receives a great deal of feedback and help. The MSTG might not have been inspected and accepted by an acknowledged scientific conference jury, but it is constantly judged and also improved by security interested people on the internet. It would have been an interesting addition for the thesis to apply two different frameworks on the implemented system for comparison. Hopefully, more security evaluation related resources for cross-platform mobile applications will become available in the future. However, having used the MSTG, it does seem like a thorough and viable framework for security evaluations of mobile applications in general. The MSTG is also a continuous project which is constantly updated and released in new versions which should make it useful even as the technological landscape changes over the years. Another thing to keep in mind is that the methods used for verifying the MASVS requirements in this thesis are adaptions made by the author to try and verify them for both platforms simultaneously. This means that the verification methods used in this thesis in many cases do not follow the platform targeted procedures as instructed in the MSTG.

6.3 Future Development

There are many things which can be improved in the implemented system, which is not too surprising since it is a prototype mainly aiming to prove a possible solution to the problem. The first thing that could be done would be to add a proper guide for how to use the system in the UI, showcasing some example components and complete cards. This would help users understand how to use the system and what the resulting benefit could be. Another improvement could be to add the possibility for customers to upload their own JSON-file directly instead of having to use the tool on the website for static configurations. Furthermore, a functionality allowing users to set an active field for cards providing them the ability to turn one or many cards off temporarily could be convenient. A priority field on cards which decides the ordering inside the mobile application would also be an enhancement. Providing data updates through Server-Sent Events instead of periodic requests could provide better efficiency through issuing network requests only when needed. Another optimization idea is to cache configuration responses for each user resulting in faster response times, and clearing the cache whenever the configuration is modified. Also, if the server responds to a request
6.4 The Work in a Wider Context

with status code 401 (unauthorized) which means that the access token has expired, then the system should make sure that the user auth state is cleared and navigate the user to the sign in page forcing a new sign in. This would provide a better user experience compared to having requests being silently denied when the access token expires.

Regarding the security level of the system, some requirements were considered not fulfilled in the case study. To successfully reach level 1 in all areas of the MSTG security verification requirements the not fulfilled requirement needs be addressed and fixed. First, the requirements which possibly can be fixed without converting to a React Native project are discussed followed by the ones believed to require a migration.

Starting with #4.6, this requirement targets the remote endpoint which means that it does not matter whether Expo or React Native is used for the mobile application. In the case of an express server, a solution would be to use the third-party package node-rate-limiter-flexible which offers functionality to prevent excessive amounts of submissions. Addressing #6.2, a potential solution to the links component problem could be to completely exclude the component from the system entirely. If the trust in ChromeCustomTabs and SFSafariViewController is not enough and the risks are deemed not worth it compared to the value provided by the Link component. Another solution could be to somehow implement validation and sanitation of provided URLs but this seems difficult since it would require recognizing whether a URL is malicious or not. Both Expo and React Native offer sufficient build functionality which means that #7.1, #7.2, #7.3, and #7.4 could be fulfilled using either of the two.

Continuing with the believed migration requirements. Starting with #3.6, since Expo currently does not support sufficiently secure crypto APIs for generating random numbers this problem cannot be solved using Expo. After a migration to React Native there are several third-party packages available which offer secure random number generation. Regarding #5.3 Expo does not provide functionality for verifying or pinning certificates yet. Using React Native, X.509 certification verification supposedly could be implemented in the native code sections for Android and iOS. As for #7.9, no free security features compatible with Expo were found, but there are alternatives available for React Native applications.

Furthermore, it would have been interesting to conduct another security evaluation case study on the same application, using a different security evaluation framework, and compare its coverage and results to the MSTG one. Analyzing the differences for the highest possible security level achievable between applications including native code compared to true cross-platform ones by using the MSTG framework would also be interesting.

6.4 The Work in a Wider Context

In a wider context, the design and the implemented system does illustrate an interesting approach of allowing end users to determine content in mobile applications. This could be interesting in many different scenarios, and even though the implementation source code itself is not provided with the thesis, the ideas and lessons learned can be useful. If a company is successful in implementing a production level version of the system their customers will gain the ability to determine content (or parts of the content) in their mobile application. This could be a key selling point for the company product and be useful both to the company but also for all of their end users which can receive requested customized functionality.

The case study of mobile security evaluation targeting a cross-platform mobile application could be used as guidance or inspiration by anyone trying to evaluate the security of their cross-platform application. But it is important to remember that the application studied is missing advanced platform interactions and build setting. Furthermore, the case study could
be used as a resource when trying to create a framework or testing guide for cross-platform security evaluations in the future. Digital security is an increasingly important topic and mobile applications usage seems to increase which means proper security evaluation methodologies and verification are essential not only for the responsible engineers but by extension for every user of these services. Cross-platform techniques are on the rise and will probably become increasingly popular given its ability to share a code base for multiple platforms reducing the maintenance struggles and need for application specialists. Given helpful resources, developers will have an easier time securing their applications and verify its behavior which is becoming increasingly important as the usage grows every day. Therefore security resources addressing not only natively developed applications but also cross-platform techniques would be an important benefit both to mobile application developers and their users.
Conclusion

This master thesis aimed to design and implement a system to examine the possibility of allowing customers to define modules that could be rendered in a mobile application. Furthermore, a method for evaluating the security of the system should be identified and used to present a result of the general security level of the system. To guide the aim of the thesis, the following research question was formulated: How can user-customized content in a mobile application be realized with considerations to security and how can the security level of such an application be evaluated?

In this thesis, design and implementation of a system that allows customers to control the content rendered inside a mobile application are presented. This was realized through implementing a parsing technique of JSON files in which customers could organize their own modules by providing nested component tag and property names, choosing from predefined components suiting their needs. The answer to the first part of the research question is therefore that user-customized content on a fundamental level can be realized using the design and implementation presented in this thesis.

Regarding the security evaluation of the system, a case study was conducted on the implemented system. To answer the rest of the research question, one example of how the security level of such an application can be evaluated is by using the OWASP Mobile Security Testing Guide (MSTG) framework. The MSTG provides comprehensive approaches for testing and verifying the security of mobile applications divided into seven areas. The case study was done using the MSTG framework and the results showed that the system passed most of the requirements for achieving level 1 security, which is suitable for typical mobile applications without especially sensitive data according to the MSTG, but not all of the requirements. Among the failed requirements were the usage of sufficiently secure random number generators and certificate pinning which could not be fixed because of the system using the Expo framework alongside React Native. Furthermore, the system failed the build setting requirements because the application was implemented as a prototype system and never meant for immediate production usage. Hence, the case study shows that the system does have a reasonable level of security but not enough for production level usage according to the MSTG framework. However, it is believed an acceptable security level could be achieved through
implementing the system using pure React Native and ditching Expo. This is thought to be possible since React Native is believed to support secure random number generators, certificate pinning, and have access to additional free security features as discussed in Section 6.3.
Bibliography


A Mobile Application

A.1 Notes Scenario

Users want to be able to write down notes directly to the phone instead of physically on whiteboards.

Figure A.1: Display notes for a room in an office.
A.1. Notes Scenario

Listing 1: JSON file example of configuration for displaying notes.

```
"body": [
{
    "type": "HeaderBlock",
    "text": "NOTES",
    "icon": "edit"
},
{
    "type": "List",
    "items": [
        {
            "title": "Computer TV Channel",
            "description": "The HDMI cable is connected to HDMI1"
        },
        {
            "title": "Telco Room Code",
            "description": "2564"
        },
        {
            "title": "Cleaning day",
            "description": "This room gets cleaned every Tuesday"
        }
    ]
},
{
    "type": "CardSection",
    "content": [
        {
            "type": "Action",
            "text": "Add Note",
            "icon": "plus",
            "actionType": "Form",
            "form": {
                "title": "Add Note",
                "icon": "plus",
                "formItems": [
                    {
                        "type": "textInput",
                        "title": "Title",
                        "placeholder": "Enter title"
                    },
                    {
                        "type": "textInput",
                        "title": "Description",
                        "placeholder": "Enter description"
                    }
                ]
            }
        }
    ]
}...
```
A.2 Screens

The two screens implemented in the prototype mobile application.

(a) Auth screen

(b) Configuration screen

A.3 Building Scenario

Users wants access to information about the building they are located inside.

Figure A.3: Display building information.
A.4 Links Scenario

Users want easy access to important links that often are used.

![Image of a webpage with links](image-url)  

Figure A.4: Provide links to frequently used websites.

A.5 Lunch Scenario

Users want to see today's offered menu of the closest restaurants.

![Image of a webpage with lunch menu](image-url)  

Figure A.5: Display lunch menu of local restaurants.
A.6 News Scenario

Relevant news should be easily shared using the mobile application.

![News Scenario](image1)

Figure A.6: Provide the latest news for the company.

A.7 Parking Scenario

Users want to know the status of free parking spaces close to the office.

![Parking Scenario](image2)

Figure A.7: Display available parking slots.
A.8 Smart Office Control Scenario

Information about the office environment should be available.

![Smart Office Control Scenario](image)

Figure A.8: Display office control information.

A.9 Service Desk Scenario

Issues around the office should easily be documented and reported.

![Service Desk Scenario](image)

Figure A.9: Display all known issues.

Figure A.10: Report a new issue.
A.10  Multiple Cards

Users can configure and setup multiple cards suiting all of their needs.

Figure A.11: Multiple configured cards.  Figure A.12: Multiple configured cards.
A.11 Updating Content - Input

Users can interact with the content through actions such as inputting data.

Figure A.13: Add a new note inside the mobile application.

Figure A.14: The customer server directly updates the configuration.
A.12 Updating Content - Periodic

The content can be configured to update itself periodically. The mobile application will ask the customer server continuously for an updated configuration.

Figure A.15: Current available parking slots.

Figure A.16: Available parking slots a moment later, automatically updated.
B.1  Sign In

The sign in page of the user interface.

![Welcome to Unique Content!](image)

Figure B.1: Sign in page.
B.2 Sign Up

Before users can sign in they naturally need to sign up for the service through the sign up page.

Figure B.2: Sign up page.

B.3 Static Content

The static content configuration page where customers can configure unique static content for their mobile application.

Figure B.3: Default static content page.
B.4 Dynamic Content

The dynamic content configuration page where customers can configure unique dynamic content, organized into entities called cards, for their mobile application.

Figure B.4: Default dynamic content page.
B.5 Dynamic Content - Example

Example of a user having configured four cards that will be rendered inside the application.

Figure B.5: Configuration using four cards.
The configuration retrieved from the customer server for each card can be inspected by pressing the eye symbol.

Figure B.6: Received configuration inspection tool.
When the user signs in to the mobile application the four cards are rendered in the same order as they appear inside the user interface.

Figure B.7: First part of configured content.

Figure B.8: Second part of configured content.
C.1 Network Traffic

The figure illustrates a sample of the raw unfiltered network traffic for the mobile application during usage.

Figure C.1: Sample of the raw unfiltered network traffic.