TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Siim-Alexander Kütt 20484IVCM

STRIDE-BASED BLACK BOX PENETRATION TESTING METHODOLOGY FOR MICRO-MOBILITY SERVICES

Master's Thesis

Supervisor: Shaymaa Mamdouh Khalil
MSc

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Siim-Alexander Kütt 20484IVCM

STRIDE OHUMUDELIL PÕHINEV PIMETESTIMISE METOODIKA MIKROMOBIILSUSE TEENUSTELE

Magistritöö

Juhendaja: Shaymaa Mamdouh Khalil
MSc

Tallinn 2023

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Siim-Alexander Kütt

04.01.2023

Abstract

Micro-mobility services, such as electric scooters and bikes have become popular as a convenient and environmentally-friendly mode of transportation in large cities. Micro-mobility services are cyber-physical systems and implementation of security practices is fundamental. In addition to the potential financial and reputational impacts of cyber attacks on micro-mobility systems, there also exists a potential impact on human life. Cyber-physical systems are complex and include multiple interconnected components, making it challenging to conduct security testing. This research aims to assist micro-mobility service providers with securing and regularly testing their services. As such, we propose a systematic black-box penetration testing methodology for micro-mobility services based on the STRIDE threat model. The proposed methodology consists of four stages and leverages threat modeling to derive potential threats to the components of a micro-mobility service. We decompose the micro-mobility service into abstract components and propose a contextual data-flow diagram for the micro-mobility service. For each component of the data-flow diagram, we derive potential threats based on STRIDE. We then analyze the attacks we could execute to realize the threat for the components. We describe our approach to selecting relevant attacks and propose threat trees to attack the micro-mobility service during the penetration test. We validate our methodology by conducting a case study on a real micro-mobility service provider in Europe. The methodology produced several findings, which we report to the service provider. However, we omit specific technical details at the request of the service provider. This study contributes to the existing and future research body of penetration testing and threat modeling for cyber-physical systems.

Keywords: *STRIDE, threat modeling, black-box, penetration testing, cyber-physical system, micro-mobility, threat tree, data-flow diagram*

The thesis is written in English and is 76 pages long, including 7 chapters, 28 figures, and 3 tables.

Annotatsioon

Mikromobiilsuse teenused, näiteks renditavad elektrilised tõuke- ja jalgrattad, on muutunud mugavaks ja keskkonnasõbralikuks transpordiviisiks suurlinnades. Mikromobiilsuse teenused on küberfüüsikalised süsteemid ning küberturbe meetodite rakendamine on selliste süsteemide puhul oluline, kuna lisaks võimalikule finants- ja mainemõjule võib potentsiaalselt olla mõju ka inimelule. Küberfüüsikalisi süsteeme on keeruline testida, kuna taolised süsteemid sisaldavad mitmeid omavahel seotud komponente ning see muudab turvatestimise keeruliseks. Käesoleva uurimistöö eesmärk on aidata mikromobiilsuse teenusepakkujatel oma teenuseid turvata ning korrapäraselt testida. Seetõttu pakume välja süstemaatilise pimetestimise metoodika mikromobiilsuse teenustele, mis põhineb STRIDE ohumudelil. Kavandatav metoodika koosneb neljast etapist ja kasutab STRIDE ohumudelit, et tuletada võimalikke küberohte mikromobiilsusteenuste komponentidele. Me jagame mikromobiilsuse teenuse abstraktseteks komponentideks ja pakume välja teenusel põhineva kontekstuaalse andmevoo diagrammi. Iga andmevoo diagrammi elemendi puhul tuletame STRIDE ohumudeli alusel potentsiaalsed küberohud. Seejärel analüüsime rünnakuid, mida oleks võimalik kasutada, et realiseerida komponentide jaoks tuletatud oht. Kirjeldame oma lähenemisviisi asjakohaste rünnakute valimisel ja pakume välja ründepuud, mida saaks mikromobiilsusteenuste pimetestimisel kasutada. Valideerime oma metoodika, viies läbi juhtumiuuringu ühel reaalsel Euroopas põhineval mikromobiilsuse teenusel. Metoodika rakendamine andis mitmeid tulemusi, millest me teenusepakkujat teavitasime. Teenusepakkuja palvel jätsime konkreetsed tehnilised üksikasjad välja. Käesolev uuring aitab kaasa olemasolevatele ja tulevastele teadusuuringutele, mis käsitlevad küberfüüsikaliste süsteemide läbistustestimist ja ohumudelite modelleerimist.

Märksõnad: *STRIDE, ohtude modelleerimine, pimetestimine, läbistustestimine, küberfüüsikaline süsteem, mikromobiilsus, ründepuu, andmevoo diagramm*

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 76 leheküljel, 7 peatükki, 28 joonist, 3 tabelit.

List of Abbreviations and Terms

API	Application Programming Interface
APK	Android Package
CIA	Confidentiality, Integrity, Availability
DFD	Data-flow Diagram
FSTG	Firmware Security Testing Guide
IOT	Internet of Things
MASVS	Mobile Application Security Verification Standard
MSTG	Mobile Security Testing Guide
OTP	One Time Password
OWASP	Open Web Application Security Project
REST	Representational State Transfer
WSTG	Web Security Testing Guide

Table of Contents

List of Figures

List of Tables

1. Introduction

As the world becomes increasingly digitized, the amount of digital services in our society is rapidly increasing. In urban environments, it has become convenient to commute short distances using a micro-mobility service to save both time and money. Such services are operating in several cities in Estonia and even more worldwide [1]. It is estimated, that in Europe alone there are over 360 000 operational micro-mobility devices spread across 600 fleets [2]. These services are built using digital technologies and as such are prone to cyber security and privacy risks. Researchers from a Dallas-based mobile security company Zimperium were able to bypass the authentication mechanism of a Xiaomi M365 electric scooter to allow for remote command injection [3]. The vulnerability allowed the attacker to accelerate scooters at will, lock them remotely and tamper with firmware. Thus, the impacts of a security incident could range from financial loss for the service provider to physical harm for the users.

Penetration testing is a widely used practice within the cyber security domain, however, it is generally done in an unsystematic way. A systematic penetration testing approach would be more efficient due to organized steps and offer more accurate results. Additionally, a systematic approach provides better consistency between tests, which can be easily documented and later referenced. The growing trend of existing micro-mobility services highlights the need for a systematic approach to help security practitioners and researchers analyze the security of such systems. Additionally, the service providers could benefit from an overview of threats to validate or improve their services. Organizational cyber security frameworks, such as ISO 27001 [4], help guide organizations towards better cyber security risk management via a collection of guidelines and best practices. Micro-mobility service providers could use to adopt similar frameworks. However, to the best of our knowledge, there does not exist a systematic methodology for penetration testing micro-mobility services, which could help the service providers test their systems.

This thesis proposes a security assessment methodology consisting of four stages to help derive a black-box penetration testing strategy. The methodology is based on Microsoft's STRIDE [5] threat model and enriched by consolidating approaches from existing STRIDE-related security research and our own applied experiments on a micro-mobility service provider. We validate the methodology by collaborating with a micro-mobility service provider to analyze the security of their service using our methodology. The results of the assessment are provided to the service provider to validate the approach and improve on any findings both from research and service perspectives.

1.1 Research problem & Questions & Objectives

Even now, the micro-mobility sector is rapidly growing and evolving. With this change, the sector will inherently inherit the problematic aspects of information technology, such as cyber security. Numerous studies, such as [6] and [7], have highlighted the importance of security and privacy within the micro-mobility domain. As the sector is fairly new as a whole, mature threat models, security frameworks, or analysis methodologies do not explicitly exist. Additionally, cyber-physical systems are often complex and include a multitude of interconnected components, making it challenging to conduct security testing. A methodology dedicated to this problem would benefit the micro-mobility service providers by helping validate and harden existing and upcoming services against cyber attacks. As such, the thesis aims to research the cyber security threats faced by modern micro-mobility services and propose a methodology to conduct security testing on a micro-mobility service. For this topic we propose the following research questions;

1. What are the common system components of micro-mobility services?
2. How can we take advantage of STRIDE-based threat modeling in order to develop a penetration testing methodology for an existing micro-mobility service?

Ultimately, the thesis seeks to improve the cyber security posture of existing and upcoming micro-mobility services by providing security practitioners, researchers, and service providers with a systematic security analysis methodology applicable to micro-mobility services.

1.2 Novelty & Contribution

The literature review presents several studies [8] [9], where STRIDE has been used to model threats for specific cyber-physical systems. However, there exists no uniform approach on how to apply STRIDE on existing cyber-physical systems. STRIDE is generally used in the literature to analyze threats during the design phase. The importance and originality of this study are that it explores applying STRIDE in a black-box fashion on an already existing micro-mobility service, which is a cyber-physical system. We use the outcomes of the threat modeling to choose more relevant attacks for the penetration testing process. We present a systematic methodology on how to create a penetration testing strategy for a micro-mobility service using STRIDE-based threat modeling. The methodology may further be generalized to include other cyber-physical systems.

There exists literature regarding threat modeling for critical infrastructures, such as microgrids [8]. This study addresses threat modeling combined with deriving a penetration-testing methodology for micro-mobility services, for which little published data exists. This study aims to contribute to this growing area of research by exploring attacks against micro-mobility services. The study provides a data-flow diagram representation of a micro-mobility service. Additionally, we provide an overview of elicited cyber threats against the common micro-mobility service components. We contribute threat trees for attacking components of a micro-mobility service. We provide, when validating the methodology, an overview of the performed penetration test, which current and upcoming service providers may use our research to better design or improve their services.

1.3 Scope & Limitations & Ethics

As this research is limited to a single case study, future research on different service providers might consider other technologies for the scope of analysis. For example, we do not consider Bluetooth technologies as the service provider analyzed in this study does not use Bluetooth technologies. The components and definition of in-scope micro-mobility services will be discussed in later chapters. For threat modeling, we used a STRIDE-per-element approach. The scope for the actual penetration test was further restricted by the service provider and is described in Chapter 5. Organizational cyber security, such as phishing and insider threats are out of the scope of our research as we focus only on the application perspective.

The validation is done as a black-box approach, which allows the penetration tester to only use the public interfaces [10]. The results of the test may be subjective as they are somewhat dependent on the skills of the penetration tester. The validation does not include certain power attacks, such as brute-force attacks, due to legal limitations. GPS-related attacks were not conducted due to technical and resource limitations. The penetration test will only involve the Android application, the cloud service, and a constrained scope for micro-mobility devices. Findings for the iOS version may differ. We do not cover privacy aspects in the research with the exception of analyzing the required device permissions requested by the user interface of the micro-mobility service.

To conduct the penetration test, consent from the service provider was received in written form. As we conduct the penetration test on a real service provider, we become aware of the architectural details of the service. Any sensitive details discovered during the penetration test will be communicated to the service provider separately. At the request of the service provider, we do not reveal any specific technical details about the service in our research. Technical details such as URLs and contents of any data flows remain discussed with a level of abstraction.

2. Background

The term *micro-mobility* has recently become widespread as the number of public rental services in urban environments has rapidly increased. Micro-mobility as a concept has existed since the invention of bicycles and scooters. The most common micro-mobility device, the bicycle, was first invented in 19th century Germany by Karl von Drais [11]. The bicycle was initially pedal-less, and thus could also be considered a scooter. Figure 1 depicts Karl Von Drais on his original *Laufmaschine* (running machine).

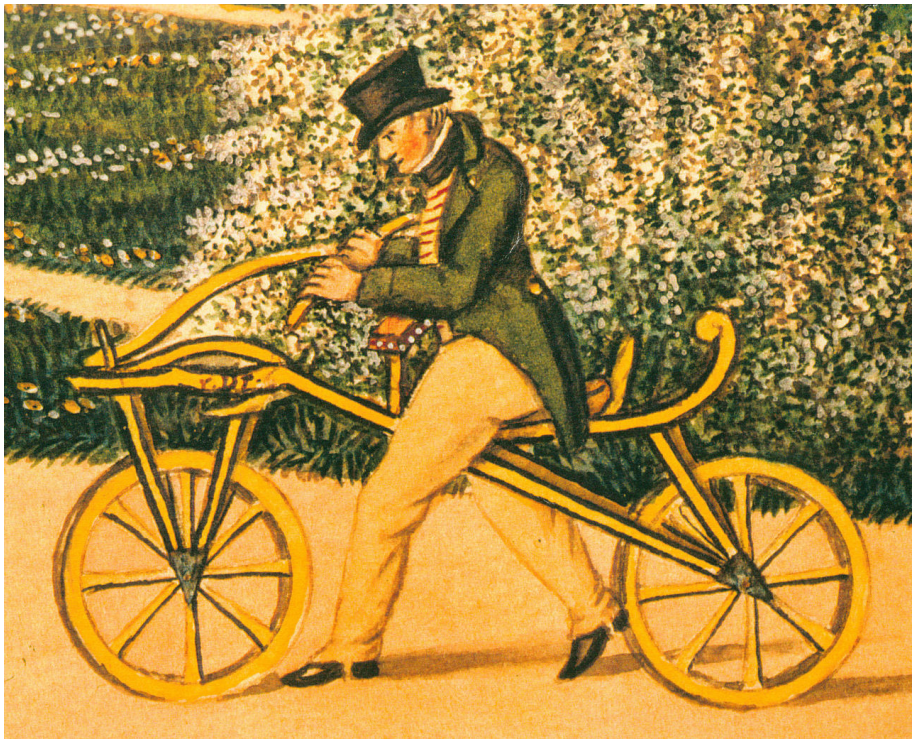

Figure 1. *Illustration of Karl Von Drais on his original Laufmaschine*[1].

Since then, various designs overtook the initial bicycle, slowly evolving towards the modern-day bicycles and scooters we know and use today. Thus, human-powered micro-mobility devices have long been available for the general public to purchase. Currently, micro-mobility devices refer to a set of lightweight vehicles that operate at speeds up to 45 km/h [12]. These speeds and definitions were reached by making the micro-mobility devices electric-powered. The first electric scooters were built in 2003 by Razor USA [13]. Electrification reduced both the time and energy needed to reach your destination. Additionally, electrification reduced carbon emissions when compared to alternative modes of transport, such as cars.

[1]Karl Von Drais - https://www.karldrais.de/index9ed2.html?lang=en (15.04.2022)

The real growth of adoption for modern micro-mobility devices arrived, when private companies started to offer dedicated and digitalized fleets of micro-mobility devices as an on-demand service. One of the first global-scale service providers was Bird[2], with their e-scooter fleet. Providing e-scooters on-demand and for a small one-time fee saw huge potential for growth in large urban environments [14]. Figure 2 depicts an e-scooter fleet of the service provider Bird in the city of Santa Monica, USA.

Figure 2. *Bird electric scooter fleet in Santa Monica, USA*[3].

Micro-mobility devices are built using various technological components. Modern micro-mobility devices are equipped with GPS and GMS sensors, connecting them to the internet via a cellular service and providing other smart services such as location, speed, and other real-time hardware data. Similar to other IoT and cyber-physical systems, micro-mobility services are also vulnerable to cyber-attacks.

2.1 STRIDE

STRIDE is a threat model initially developed by Microsofts Security Task Force to be used by Microsoft product teams to secure their products [15]. The STRIDE taxonomy stands for six mnemonics [5], which depict the following cyber threat types:

[2]Bird - https://www.bird.co/ (11.09.2022)
[3]Santa Monica Daily Press - https://smdp.com/2019/09/20/city-rescinds-reduction-in-bird-fleet/ (11.09.2022)

1. **Spoofing:** Allowing an attacker to identify as someone else.
2. **Tampering:** Malicious modification of code or data.
3. **Repudiation:** An attacker's ability to reject having carried out an action.
4. **Information Disclosure:** Exposure of data to unauthorized parties.
5. **Denial of Service:** Disruption of service availability.
6. **Elevation of Privilege:** An attackers ability to gain elevated capabilities.

While threat modeling is usually done at the development stage, there does not explicitly exist a standard for applying STRIDE. The STRIDE mnemonics have been used for threat modeling cyber-physical systems in several research papers in the literature [8] [16]. We are attempting to model a micro-mobility service and use STRIDE to derive threats for the contextual model. STRIDE was chosen because: (i) it considers threats from an attackers perspective, (ii) it is more complete than the regular CIA triad by also addressing authorization, authentication, and repudiation, (iii) it allows to consider threats against individual components and interactions [8]. Howard and Lipner [5] apply the mnemonics to data flow diagrams (DFD) derived during the design phase of an application or a system. Khan et al [8] apply the STRIDE mnemonics to cyber-physical systems in a similar fashion, analyzing the threats between system interactions.

2.2 Related work

The network and systems engineering cyber security lab at the KTH Royal Institute of Technology in Stockholm have conducted research on IoT penetration testing. The NSE lab has created PatrIoT [17], an IoT penetration testing methodology. The methodology is built on the four stages of planning, modeling, exploiting, and reporting. The planning stage focuses on gathering information about the target of the penetration test, discerning the scope, and enumerating any available resources. The modeling stage includes decomposing the system into smaller components, leveraging a simplified version of STRIDE and DREAD for deriving threats and scoring risks. They improve the threat model as the penetration test progresses. The exploitation stage focuses on testing the system for known vulnerabilities and developing custom exploit payloads. The final stage of reporting describes a method of structured reporting and disclosure. They evaluate their methodology by applying it to conduct a penetration test on seven different IoT devices and describe the results.

Some research has also been conducted on the cyber challenges faced by micro-mobility services and the components of micro-mobility systems. Nisha Vinayaga-Sureshkanth et al. [18] illustrate several key challenges faced by micro-mobility service providers, such as physical attacks, eavesdropping, spoofing, and fuzzing. For each of the threats, they

detail their observations regarding the system and the consequences of successful attacks. Additionally, potential countermeasures for remediation are analyzed. They provide an overview of collected user data, which was sampled from a privacy-policy analysis of several micro-mobility service providers such as Lime, Lyft, and Bird.

As online services often rely on mobile applications for the user interface, the security and privacy implications of mobile applications have to be considered. Enck et al. [19] analyzed over 1100 Android applications based on popularity and discovered different privacy and security implications with advertisement libraries and leaking log information. This illustrates the need to consider the analysis of the smartphone applications in both standalone and multi-component services, as most micro-mobility service providers, including the one analyzed in this study, rely on smartphone applications to provide the user interface of the service.

A study analyzing 20 different ride-hailing applications [20] discovered, that the services focus mostly on the privacy of the consumers, however, little had been done to protect the privacy of the drivers. They discovered it was possible to extract driving behaviors and frequently visited addresses from a feature that provided data to the application about nearby cars. This study included Taxify (now Bolt) in the city of Paris.

Privacy and security requirements for software ecosystems, which include many different types of software and hardware components are challenging to model due to numerous different complex components [6]. This may introduce security and privacy errors into the software ecosystem, which can go unnoticed during development. Using modeling methods enhanced for security and privacy requirements in the development process can help remediate potential vulnerabilities early as demonstrated by Mai et al. [6] in their case study of applying the methods and conducting an after-survey. The outcome of their study illustrates, that a dedicated methodology modeled for micro-mobility services could potentially help improve the security posture of the domain.

Data from several studies suggest that STRIDE could be successfully applied to cyber-physical systems. As a micro-mobility service is a cyber-physical system we discuss research related to the applicability of STRIDE to cyber-physical systems. Khalil et al. [16] have systematically applied STRIDE for threat modeling of cyber-physical systems. They identified research gaps regarding the interactions of cyber and physical spaces, proposing a systematic methodology for modeling threats for cyber-physical and industrial control systems. The methodology would help cyber security experts and practitioners to model threats for cyber-physical systems, such as the Microgrid. Their methodology consists of a nine-stage approach, which includes asset identification, modeling the assets

as a data-flow diagram, and threat elicitation. The final stages of the methodology introduce threat prioritization and requirements selection. A case study was conducted for an energy management software platform in a Microgrid system by composing a team of five experts from relevant fields and validating the proposed methodology in practice.

Haider et al. [21] modeled cyber threats for wireless attacks against advanced metering solutions. After creating a data-flow diagram for a component of the advanced metering infrastructure they applied the STRIDE threat model to derive threats. They chose to use STRIDE due to its maturity in the field. Five threats were derived and DREAD was used for risk modeling. According to the DREAD threat prioritization, a distributed denial of service attack was found to have the highest risk.

Li et al. [9] created a security-usability threat model for industrial control systems. They created a data-flow diagram illustrating a use case, where a control engineer is establishing a secure connection to a programmable logic controller. They applied the STRIDE threat model to the data-flow diagram with a STRIDE-per-element approach to derive threats and task scenarios for the components of the use case. They combined the results with prior research on usability factors. They found that several responsibilities are placed upon the engineer, causing the lack of usability to raise potential threats for spoofing and tampering.

Aside from STRIDE, several other threat modeling methods exist in the literature. Shevchenko [22] et. al discuss twelve threat modeling methods, which target aspects such as risk, security, or privacy. Asset-centered modeling methods such as OCTAVE and Trike focus on a perspective, where the assets of the system can confidently be enumerated and evaluated for organizational risk. Both methods involve a wide range of stakeholders from different departments and are exhaustive and time-consuming. Privacy-centered methods such as LINDDUN focus on the privacy aspects of a system. The aim is to enumerate all components of the system using data-flow diagrams and link the elements to LINDDUN privacy mnemonics. PASTA is a risk-centered modeling approach, which consists of seven stages for combining business objectives with technical requirements. PASTA also requires input from key stakeholders from different departments of the business.

Based on the literature review we conclude that several other threat modeling methods exist, however, they require inputs from key stakeholders or asset information we do not have access to. Some of the discussed modeling methods aim to achieve objectives such as privacy, which we do not focus on in this research. We found STRIDE to be the most mature and commonly used modeling method for cyber-physical systems. Data-flow diagrams combined with the STRIDE threat model have been used successfully to elicit cyber security threats for cyber-physical systems. However, STRIDE has mainly been

applied during the design phase. No studies have employed STRIDE-based threat modeling to develop a black-box penetration testing methodology for a micro-mobility service. The PatrIoT [17] methodology is similar, however, we apply STRIDE to the components in a more exhaustive manner.

2.3 Existing Information Security Assessment Guides

Information security assessment or security analysis is a method of assessing system behavior when subjected to a set of security-oriented tests and examinations [23]. The NIST [23] security assessment guide describes how a structured security assessment approach can provide consistent results with minimal risk when conducting security tests. A structured methodology seeks to lessen the learning curve for new assessment staff and minimize resource-related constraints, thus illustrating the general benefits of having a methodology. This section will describe some of the existing security assessment guides. We provide an overview of the purpose, target audience, techniques, and reporting metrics for the guides. Furthermore, we discuss and explain our reasoning for developing a new approach instead of using one of the existing guides for the case study.

2.3.1 The Technical Guide to Information Security Testing and Assessment

The *Technical Guide to Information Security Testing and Assessment* [23] by NIST was initially developed for use by US federal agencies. However, the methodology is available online for free and can be adopted by nongovernmental organizations. The purpose of the guide is to help organizations plan and conduct security assessments. Furthermore, the guide highlights strategies for reporting and vulnerability mitigation. The target audience is considered to be computer security staff. The guide proposes several techniques to assist organizations with security assessments. The analysis techniques highlighted by the guide are provided below.

- Review Techniques
- Target Identification and Analysis Techniques
- Target Vulnerability Validation Techniques

The review techniques propose to review organizational documentation, logs, rulesets, and system configurations, which can help discover gaps in the security controls. Reviewing logs provides a historical view of system usage and configuration. The ruleset review assesses router access control lists, firewalls, and IDS/IPS rulesets for any discrepancies.

System configuration analysis helps reveal any missing gaps in the systems concerning hardening policies. The target identification and analysis techniques include technical techniques such as network discovery, port and service identification, vulnerability scanning, and several different types of wireless scanning. The aim is to identify active devices within the scope of the assessment and analyze them for any possibility of exploitation. Furthermore, it is explained how vulnerability scanners can discover and analyze devices without human intervention. The vulnerability validation techniques involve password cracking, penetration testing, and social engineering. The aim is to confirm the potential vulnerability, which was discovered in the identification phase, and demonstrate the possibility of exposure. It is worth mentioning that while some social engineering aspects are covered by the guide, organizational cyber security remains out of scope for our methodology.

Additionally, the guide proposes several chapters to assist with planning and performing the security assessment. The chapters cover policy and logistics constraints, as well as legal considerations to be taken into account while performing the penetration test. Furthermore, the guide details how the test is to be executed, taking into account human coordination and data handling constraints. Finally, the guide provides steps regarding post-testing activities, such as mitigation, reporting, and remediation.

The guide highlights, how different security assessment techniques can be performed but does not emphasize which techniques should be performed. The guide focuses a lot on organizational scope, while we are seeking to understand the threats to components and data flows of a single system. In the guide, no threat modeling is performed to understand the most likely threats to the system under test.

2.3.2 The Open Source Security Testing Methodology Manual

The *Open Source Security Testing Methodology Manual* [24] is an open-source security testing project, that is being maintained by the Institute for Security and Open Methodologies (ISECOM). The guide is freely available online for any organization to adopt. The OSSTMM guide aims to assist with conducting security testing on any kind of system. The framework can be adapted to different types of audits, such as penetration tests, and is designed so that the results would be quantifiable and verifiable. Correctly following all of the guidelines presented by the guide allows the security assessor to perform an *OSSTMM audit*. The guide is not explicitly aimed toward any type of organization or activity domain and is scoped to cover several operational security channels, including physical, human, wireless, and wired channels.

The techniques covered by the guide are divided into several operational security channels, which we describe below.

- Techniques related to the human security channel require interaction with key personnel responsible for the assets under test. The channel aims to verify that key personnel are security aware and following corresponding company security policies or other internal regulations. Furthermore, steps for assessing the security of several aspects of the human channel, such as the review of internal policies and organizational culture.
- Guidelines related to the physical channel aim to test the material boundaries of the organization and require interaction with gatekeeping barriers and personnel. The aim is to test out the security of physical controls such as doors, locks, and other means of entry into areas of restricted access.
- Guidelines related to the wireless channel aim to test assets from within proximity via the interception and analysis of electromagnetic radiation. The analysis will assess the age of the systems and software, and security controls regarding monitoring and tampering on wireless devices, including modification of signals.
- Guidelines related to wired security can be considered the network security part of the guide. Instructions are provided to assess the monitoring and filtering of incoming and outgoing network data, conduct internal network reconnaissance, discover active services and ports, evaluate authentication controls and verify any discovered exploitation opportunities.

Feedback about the results of the security assessment is provided via the *Security Test Audit Report* (STAR). The STAR serves as a summary of the attack surface within the scope of the assessment, that can be provided to organizational stakeholders as an overview. The STAR report also highlights systems, which were not tested.

The OSSTMM guide is a big manual, which focuses also on several organizational aspects. The guide describes testing many different channels, even human-related. This scope takes great effort to employ and coordinate between different stakeholders. Threat modeling is not discussed and it is difficult and time-consuming to derive the channels applicable to micro-mobility services. It is even more difficult to choose the actual attacks, which might be relevant. Our methodology is faster as the key threat tree patterns have already been derived from another study.

2.3.3 The Penetration Testing Framework

The *Penetration Testing Framework* by Kevin Orrey [25] defines a very hands-on approach to penetration testing. The guide presents topics around network reconnaissance, where several tools and techniques are presented, which would help the tester to conduct both active and passive reconnaissance. This chapter is followed by the discovery and probing techniques, which would help fingerprint remote hosts. Any active, and possibly vulnerable, services could then be exploited. Additionally, recommendations for vulnerability scanners, such as Nessus[4], are provided to test discovered hosts for vulnerabilities. Techniques regarding network security, wireless penetration, and physical security are presented. Finally, steps for auditing IBMs AS/400 server systems, as well as various techniques regarding vendor-specific attacks against service providers such as Cisco and Citrix are presented.

Two reporting templates are provided to assist with conducting the penetration test. The pre-inspection template aims to insure the compliance and legitimacy of the test and provides a checklist for key points to verify with stakeholders. The key points include the scope of the test, relevant documents signed by an authority, and other necessary clearances. The final reporting template provides key points to highlight regarding the test, such as the date and the testing team. This is followed by all the technical details related to the findings and proposed remediations. The presented techniques are very hands-on and allow an experienced tester to get an overview of attacks against specific technologies. However, no supporting structure or entry point is presented. There is little description and structure regarding the scope of the guide. Non-technical stakeholders would not be able to assess the usefulness of the guide for their organization. There exists no systematic structure and an entry point to the framework. There exist no guidelines on how to elicit threats specific to components. While the actual attacks are provided, the testers will have to find a different way to choose and structure the relevant attacks for their system. As the guide is more like a collection of possible attacks against different technologies, we can leverage it for our case study. Based on our threat trees, we can adopt some of the relevant attacks into our penetration test.

2.3.4 The OWASP Testing Guides

The OWASP Foundation[5] is a well-known nonprofit security organization in the industry. They have developed several guidelines for testing web applications, mobile applications, and even IoT devices. Aiming to improve the overall security posture of software, they lead

[4]Nessus Vulnerability Scanner - https://www.tenable.com/products/nessus (15.09.2022)
[5]OWASP Foundation - https://owasp.org/ (16.09.2022)

open-source security projects and conferences all over the world. The OWASP Foundation has developed several guides that are of interest to us. The Web Security Testing Guide (WSTG) [26] provides guidelines for testing the security of web-based applications. The guide is of interest to us because micro-mobility services often include a web application for business logic. The guide was developed to help web application developers and application security experts write better software and discover security vulnerabilities. The WSTG contains a collection of the most common attack vectors in modern web applications, which can be used as a guideline to conduct security assessments against web applications. Additionally, the WSTG helps to construct business cases for testing, highlights best practices for reporting, and provides instructions on how to integrate the guide into the software development lifecycle. For reporting and security measuring, the WSTG recommends considering the number of vulnerabilities detected combined with their risk rating. It is worth mentioning, that the OWASP WSTG does describe threat modeling to an extent. However, it is proposed as a proactive measure to be used during the development stage and is not discussed in the context of helping a penetration test for an existing system.

The second relevant guide from OWASP is the Mobile Security Testing Guide (MSTG) [27]. The MSTG provides a comprehensive guide to test the security of smartphone applications both on Android and iOS operating systems. The guide is of interest to us because the micro-mobility services in our scope include a smartphone application. The guide is directed toward mobile security testers and includes topics such as static and dynamic testing, reverse engineering, source-code obfuscation, and analyzing internal platform functionality. Mobile software developers and architects may choose to adopt the OWASP Mobile Application Security Verification Standard[6] (MASVS) as a standard when developing smartphone applications. The MSTG is centered around techniques for verifying the MASVS standard. The MSTG does not create or outline any reporting metrics but provides general recommendations on the topics to include in the assessment report.

Finally, there exists a guide from OWASP relevant for IoT testing. The Firmware Security Testing Methodology (FSTG) [28] was created to allow developers and security practitioners to better understand the security aspects of IoT devices. The FSTG presents nine stages to analyze the security of a firmware image. The stages cover information gathering and reconnaissance on the target device and its firmware. Additionally, the FSTG describes how the firmware of a device might be acquired, extracted, and analyzed. Finally, the FSTG describes exploiting techniques to create a proof-of-concept exploit, should any vulnerability be discovered within the firmware image. Conclusively, the FSTG enables

[6]OWASP MASVS Standard - https://mobile-security.gitbook.io/masvs/ (16.09.2022)

security researchers and developers to conduct security assessments on device firmware images.

The micro-mobility services we are considering generally consist of three key components, such as a web application, a smartphone application, and a micro-mobility device. This is further described in Chapter 4. The OWASP guides all have some relevance regarding the components of our case study. However, the guides are standalone and do not present a single systematic methodology. There is no guidance on how to model existing components and choose attacks relevant to our system. Furthermore, the FSTG considers only the firmware image and not other embedded components, which a cyber-physical system might possibly have. Similarly to the Penetration Testing Framework [25], the OWASP guides provide us with a list of possible attacks we can choose between.

Conclusively, none of the guides consider specifically modeling existing systems via a mature threat model on a level we are proposing and provide a systematic way of eliciting threats relevant to a cyber-physical system. We seek to understand the threats a micro-mobility service generally faces and attack the system from these angles. For a micro-mobility service, adherence to cyber security requirements is of particular concern due to the potential impact on human life. Thus, there is reason to focus on a more specific methodology for this domain. The existing guides are general and not specific to the cyber challenges a constrained system may face. Instead, the guides provide us with a list of possible attacks we can choose between.

2.4 Tools

In order to conduct the analysis we employed several free online and open-source tools. We used the online service APKPure to acquire the latest version of the Android application provided by the service provider as of October 2022. Additionally, we acquired an older version of the Android application to compare and assist with static analysis. Initial static analysis for the latest version was done by utilizing the MobSF online static analysis tool. Additionally, we used 7Zip and JadX to extract files from the user interface component. This allowed us to extract the source code of the user interface. The latest version of the user interface was obfuscated. This required us to use tools such as HBCDump to disassemble the Hermes Javascript bytecode. The source code was analyzed manually via command line and text editing tools such as *grep*, *find*, and Notepad++. For information gathering on the domain of the service provider, we used tools such as DNSRecon, DNSEnum, and Fierce. We used OWASP Dirbuster for directory enumeration.

Dynamic analysis was conducted using a rooted Huawei Honor 8 Android phone running the Android version 7 Nougat operating system. We used the Android Debug Bridge (ADB) with superuser privileges for local storage analysis. For bypassing device integrity checks, we used the Magisk software suite[7] with relevant modules. The Objection mobile exploration toolkit was used to enumerate the process environment. In order to analyze the network traffic between the user interface and the back-end service we used MITMProxy[8] and Burp Suite Community Edition[9] to execute a man-in-the-middle attack.

[7]Magisk - https://github.com/topjohnwu/Magisk (15.10.2022)
[8]MITMProxy - https://mitmproxy.org/ (20.10.2022)
[9]Burp Suite - https://portswigger.net/burp (20.10.2022)

3. Methodology

This chapter describes a systematic methodology for using the STRIDE threat model, data-flow diagrams and threat trees by Howard and Lipner [5] to create a penetration testing strategy for a micro-mobility service. The methodology used in the research can be divided into four different stages. The threat modeling stage draws inspiration from the research by Khan et al. [8]. Figure 3 depicts the four different stages of the proposed methodology.

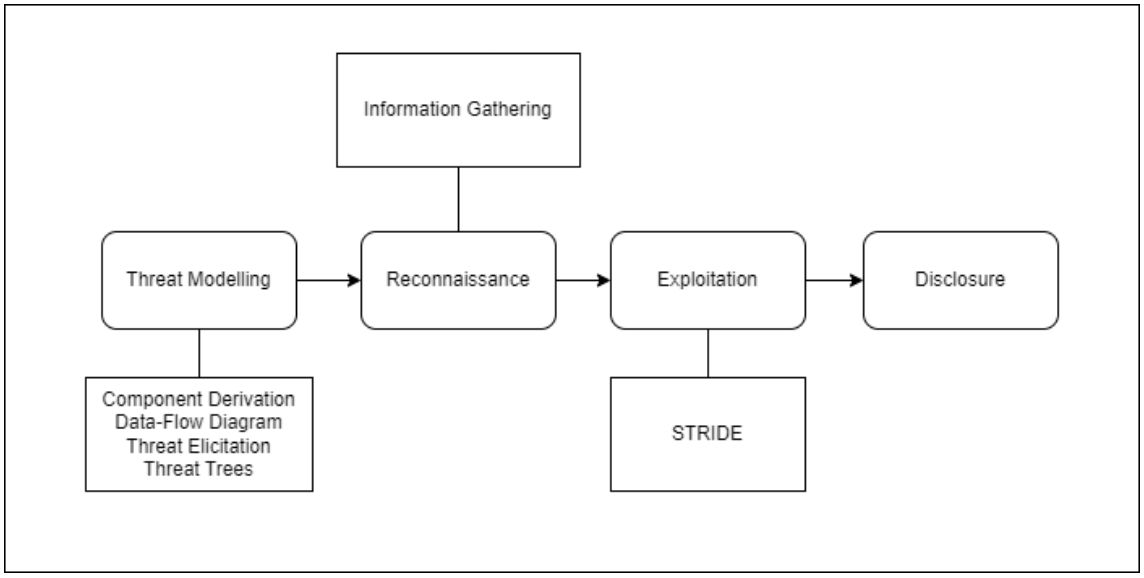

Figure 3. *The four stages of the methodology.*

3.1 Threat modeling

Threat modeling is an important step in the methodology as it allows us to identify the system assets and focus on the potential cyber threats faced by the assets in a systematic way. This results in greater efficiency and accuracy. Additionally, threat modeling can guide the code review and penetration testing processes [5]. If threat modeling was not used there would be no overview of the existing assets and we could not consider their susceptibility to common cyber-attacks in an organized fashion. The modeling stage consists of four parts: (i) deriving the main system components, (ii) plotting the micro-mobility service components as a data-flow diagram, (iii) eliciting threats faced by the components using STRIDE, and (iv) creating threat trees using research by Howard and Lipner [5].

3.1.1 Component Derivation

Deriving system components such as processes, data stores, data flows, and external entities is necessary as each may be vulnerable to threats from the STRIDE mnemonics. To assess all threats, the components must be defined. However, it is challenging to define components for an existing system, which is a black box to the tester. This causes the model to often reflect only partial reality and is an issue for black-box security testing. In this research, we derive abstract system components by observing the system from the user's perspective and interacting with the different functionality available to the user. Providing inputs to the system and observing the outputs can help an expert predict data flows. It may be helpful to consider, which components are necessary to produce the observed outputs. For example, if a user interface of a micro-mobility service displays the location of all available micro-mobility devices, there must exist an outgoing data flow from the devices, which provides this data. Additionally, it is helpful to analyze the documentation provided by the service provider, such as privacy policy and technical blogs. Furthermore, the functionality necessary to administer the service could be considered. This step acts as an entry point to the threat modeling stage as it allows us to enumerate the system components.

We may be able to observe the functionality of the service for any potential discrepancies, which might be helpful to explore during the exploitation phase. The tester might observe conditions for any audio or visual feedback emitted under expected conditions, such as beeping. It can be argued, that this counts as active reconnaissance and leaves a digital footprint. However, we provide no systematic steps to execute in order to derive the components. The tester should analyze potentially privileged activities and the parts involved in granting such privileges. This will allow us to draw trust boundaries. It is useful to graph all the components for a visual overview. We present a graphical overview of our system in Chapter 4. Some consideration must be given to the scope of the penetration test. The data-flow diagram discussed in Chapter 4 contains several components, which remain out of scope for our actual penetration test due to the application-focused scope and restrictions from the service provider.

3.1.2 Data-flow Diagram

After deriving the components of the system they can be plotted as a data-flow diagram. Plotting a data-flow diagram is useful as it allows for visualization of the data flows between the system components. The data-flow diagram notation was chosen as the modeling language because it is simple to understand, fast to create, and easily compatible

with STRIDE. Alternatives, such as the Unified Modeling Language (UML) notation require more effort to create and are more complex to understand due to using a greater number of symbols [29, Ch. 2]. For the level of detail within the data-flow diagrams, we remain at the contextual level. The contextual level is a general overview of the whole system, which is easily understood by a wider audience [30]. From the external perspective and without deeper analysis we are not able to draw in greater detail. The data-flow diagram notation consists of several elements to aid the modeling process. We define the elements of a data-flow diagram in Figure 4.

Figure 4. *Elements of a data-flow diagram [29, Ch. 2].*

The data-flow diagrams distinguish between four main elements and the trust boundary notation. There are slightly different descriptions for the elements in the literature, however, the underlying concepts remain the same. We adopted the DFD element definitions and visual designs from Shostack [29]. Additionally, we chose to assign numeric values to some of the components to illustrate the scope of the methodology. We acknowledge that organizational threats, such as malicious insiders may exist, however, they are out of the scope of this research. We do not number these components and keep them only to provide context.

Data flow diagram elements include external entities, which are depicted using a square rectangle. External entities are defined as uncontrollable actors or code outside the system. Examples of external entities are legitimate users of the system or malicious actors running a script. Rounded rectangles represent *processes*, which can be defined as data processors, such as any code or application running in the system. Processes can consume the data, modify it and return new data. Examples of *processes* are web applications, REST interfaces, and other software logic. Data flows are depicted as arrow lines with a direction. Data flows describe the flow of data between external entities, processes, and data stores. Examples include HTTP requests between processes and external entities. Data stores are depicted as text between two parallel lines. Data stores are defined as anything, which can store data. Examples include databases and text files.

Trust boundaries are section boundaries within the data-flow diagram, where data moves from a lower privileged section to a higher privileged section. Data moving from a low-privileged section to a highly privileged section should be validated for correctness as it

can allow unintended behavior, such as escalation of privilege. Inversely, data moving from a highly privileged section to a low-privileged section must not disclose any sensitive information [5]. Example scenarios include overly revealing error messages or describing data around the configuration and architecture of a component in more detail than necessary.

3.1.3 Threat Elicitation

Having modeled the data-flow diagram, we can apply the STRIDE mnemonics to derive threats for each of the data-flow diagram elements. We will apply STRIDE in a STRIDE-per-element fashion as we are modeling abstract components, where we are not aware of all the possible different interactions. The method allows the discovery of the most likely threats for each of the components as opposed to the STRIDE-per-interaction approach, which enumerates threats for each interaction and is less applicable to abstract models [29]. Each of the STRIDE mnemonics applies to one or more data-flow diagram elements. We have already defined STRIDE mnemonics in Chapter 2 and now present a general mapping of STRIDE mnemonics to the data-flow diagram element types in Figure 4. Table 1 displays the STRIDE to DFD mapping as defined by Howard and Lipner [5].

Mapping of STRIDE to DFD Elements						
DFD Element	S	T	R	I	D	E
External Entity	X		X			
Data Flow		X		X	X	
Data Store		X	†	X	X	
Process	X	X	X	X	X	X

Table 1. The Mapping of STRIDE to DFD Elements[5, Ch. 9].

We can observe that not all mnemonics are applicable to each element. The presented mappings are only proposed as a general guideline and can be modified depending on the analyzed system [29]. Thus, the actual mappings may differ from the proposed mappings in different systems. For example, tampering with an external entity may make sense in our context as we can modify the user interface running on the client side. The process element is generally prone to all of the STRIDE threats. The data store element has a special symbol † for the repudiation threat. Repudiation can be a threat to data stores if they contain audit or logging data, in which case an attacker may be able to cover their tracks [5, Ch. 9]. The derived threats for the data-flow diagram elements allow us to choose more relevant attacks and generate threat trees, which help guide the penetration testing process.

3.1.4 Threat Trees

To illustrate the testing process in a more systematic way we chose to graph the threats in a tree format. We chose to call them threat trees because we keep the root nodes of the trees as STRIDE threats. Our threat trees describe the attacks a tester should execute to test if a system component is vulnerable to the given STRIDE threat. This can happen if any of the security preconditions proposed by Howard and Lipner [5, Ch. 22] are violated. Howard and Lipner [5, Ch. 22] have created threat trees for all of the data-flow diagram element types, which describe the relevant security preconditions for each element. The threat trees consist mostly of OR operations unless an AND operation is explicitly specified via a label. We have included all of the threat trees created by them in Appendix 2. Their threat trees highlight possible security preconditions for each of the STRIDE threats on the generally applicable DFD element types discussed in Table 1. If the precondition for an attack is true, the element is vulnerable to the respective STRIDE threat. Using the previously produced data-flow diagram and the threat-to-element mapping provided in Table 1, it is possible to enumerate all of the applicable security preconditions for a component. We can then graph the attacks required to test the preconditions as trees, which we can follow during the penetration testing process. We use only OR branches within the threat trees as the leaf nodes are attacks and not security preconditions. We aim to make the attack paths separate to make the tree easier to follow. If an attack depends on another threat, the trees can be cascaded. We did not graph such cascaded variants to keep the threat trees simple, however, we acknowledge that one threat may cascade to another.

We chose to use this approach due to the simplicity and flexibility of creating such trees. The threat trees are simple to create and understand while remaining flexible with their usage. We discourage creating a single threat tree, which contains all of the STRIDE threats, system components, and their respective attacks. The graph will likely become overly big and hard to follow. We keep the root node of the tree as the name of the threat. The attacks necessary to test the security preconditions are described as leaf nodes of the tree. The child nodes are used to organize the components of the system within the attack path. The branching continues until a leaf node requires the tester to conduct an actual attack on a component. The attack tests for the existence of a security precondition proposed by Howard and Lipner [5, Ch. 22]. However, the actual attacks (leaf nodes) must be chosen by the tester based on the security precondition. We used existing guides such as the OWASP WASTG [26], OWASP MASTG [27] and the Penetration Testing Framework [25] to derive the attacks for our case study. Additionally, Howard and Lipner [5, Ch. 22] provide attack techniques for each of the security preconditions. While this approach should gather most of the relevant attacks, the tester is free to add additional attacks and tests for completeness. We provide examples of our threat trees in Chapter 5.

3.2 Reconnaissance

Reconnaissance is the act of gathering information about the target system via both active and passive methods. Active methods involve interacting with the system components and may leave a digital footprint, while passive methods do not interact with the components and employ publicly available data on the Internet [31]. Reconnaissance is the second stage of the methodology because the results of the reconnaissance stage are a prerequisite for executing some of the attacks derived in the modeling stage. The reconnaissance stage will provide attackable uniform resource locators (URLs) and an improved understanding of the service architecture on a technical level. Additionally, we may discover development artifacts and potentially sensitive data from the publicly accessible components.

During reconnaissance, we aim to identify the used technologies, defenses, and internal architecture of the system. Additionally, we identify the authentication and authorization schema between the system components. Similarly to STRIDE threat trees, we graph all of the reconnaissance activities in a tree-like format to make the stage more systematic. We name the root node of the tree as information gathering because it is the objective of this stage. While this is not a STRIDE threat, it still illustrates the objective of the tree, which the attacks (leaf nodes) aim to achieve. We organize the tree using child nodes for each derived component. We graph the leaf nodes as information-gathering attacks relevant to each component. This tree is described in Chapter 5.1. To derive the attacks for the reconnaissance stage, we chose relevant attacks from existing guides based on the nature of the component. Guides such as the OWASP MASTG [27], the OWASP WASTG [26] and Penetration Testing Framework [25] provide a multitude of techniques. There may exist some overlap between attacks chosen for the information gathering and the attacks required to test for the STRIDE threats. For example, conducting a man-in-the-middle attack on a data flow can both be a reconnaissance activity and an attack on the information disclosure preconditions for a data flow. Furthermore, the STRIDE threat trees derived during the modeling stage may be updated after the reconnaissance stage with more specific attacks against the technologies used by the system.

3.3 Exploitation

During the exploitation stage, we follow the threat trees for each of the STRIDE threats and execute all the attacks graphed as leaf nodes. The attacks for the exploitation stage will generally differ from the reconnaissance stage as many of the attacks executed during this stage are more interactive, such as sending malformed input. Attacks during the reconnaissance stage pose a more read-only approach. During the modeling stage, we

logically organized attacks regarding the system components to create more structured attack trees. The threat trees created for our case study can be found in Section 5.2 of Chapter 5. With consent from the service provider, we execute the graphed attacks using publicly available methods and tools. We then report back to the service provider with the results of the reconnaissance and exploitation stages.

3.4 Reporting

This research does not aim to develop nor propose a novel reporting template. There already exist several reporting templates, which can be utilized for reporting the findings of a penetration test. For reporting our findings to the service provider, we chose to use a template developed by OWASP [26]. Additionally, there exists a reporting template created by Emre Süren for the PatrIoT methodology [17] and reporting guidelines provided by NIST [23]. The tester could also choose another reporting template or create their own. As highlighted by the PatrIoT methodology, the use of the same systematic reporting template allows for the generation of statistics and leverages previous reports during the testing process [17]. For this reason, we recommend the use of the same template if several case studies are conducted using the same methodology.

4. Threat Modeling of Micro-Mobility Services

This chapter will introduce the service provider of the case study. Additionally, we propose abstract components general to micro-mobility services in our scope. We present a data flow diagram for the proposed service components and derive cyber security threats from the data flows using the STRIDE threat model with a STRIDE-per-element approach.

4.1 Introduction of the Service Provider for the Case Study

The term "service provider" refers to the micro-mobility service provider analyzed in this study. At their request, we do not mention the actual name of the company and technical details related to their services. The service provider provides on-demand electric scooters, which operate in several different cities and countries in Europe. The service provider has several years of expertise within the micro-mobility domain and has developed solutions for customers around the globe. Additionally, they have developed smart scooters for private use. The company currently manages thousands of electric scooters and provides services to over 100 000 users.

The scooters are equipped with long-lasting electric batteries and different smart technologies such as GPS sensors and internet connectivity. The scooters record the rides of the users and provide a detailed overview of the ride history via a smartphone application. The users can use the scooters for up to 90 minutes per session and are charged by the minute, making it convenient for small commutes inside the city. The micro-mobility service consists of a smartphone application, a Google Firebase cloud service, and physical electric scooters operating within predefined geographical regions of the service areas.

The service can be used by interacting with the user interface. The user interface of the service allows users to register using their mobile phone numbers. After verifying the ownership of the phone number, the user must enter their payment details, such as credit card info to pay for the service. Once the payment details are added, the user has to find the nearest available scooter in the service area and scan the unique QR code to begin the ride. The available scooters are found via a map in the user interface. The map highlights all available scooters within the service area.

The user interface presents all of the scooters available for rent, including their location and estimated range. This data is requested from the back-end service. The user must then

choose a suitable scooter and click a button to start the ride. The scooters can be reserved in advance if a user wishes to use a specific scooter. The requirement to start a ride is to have a valid payment method in the wallet section of the user interface. After starting the ride, the vehicle status changes and the scooter becomes electric-powered. The user can begin the ride to their destination. After starting the ride, the back-end service ties the user's identity to the bike and starts tracking the ride session. Each vehicle is stored in the cloud storage under a specific resource identifier document. This resource is public as long as the vehicle is not being used. If the scooter is in use, the data can only be viewed by the user using the scooter. This is a good practice because otherwise the service would allow tracking of users and there would be an impact on the user's privacy. After reaching the destination, the user has to end the ride by clicking the relevant button. The session ends with the ride details being presented to the user and saved into ride history.

There are two payment plans for using the service. A user may choose to link their credit or debit card to the service and pay using a per-ride basis. Additionally, it is possible to pay a fixed price for a period ticket, which lasts a fixed amount of time. Period tickets are available with separate pricing options. The website of the service provider is promotional and offers no functionality related to the service. The website provides contacts for communication and customer support. The website includes an online shop for buying helmets, which could be used while using the service. The helmets are not mandatory as per the service agreement.

Micro-mobility services are active within some predefined region of operations. Everything outside of the area is considered a no-parking zone. The service incurs a monetary penalty on the user, should they end their ride session in a no-parking zone. The operating areas are stored in the service provider's cloud storage as chained polygonal lines. These area mappings are publicly accessible. Inside the regular service areas, numerous no-parking zones such as cemeteries and parks are defined, where scooter presence is not wanted.

4.2 Components of a Modern Micro-Mobility Service

To better understand how cyber risks would manifest, it is important to investigate the components of a micro-mobility service, as the system consists of more than just the micro-mobility device. The first step to deriving the components would be to use the service for its intended purposes, without running any specific tests. However, we provide no specific steps or methods to execute. The tester should observe, how the service reacts to the inputs of the user under regular circumstances. This is useful for component analysis and data-flow mapping. There are several merits to using the service before executing any attacks. It allows us to discover the initial web components to be tested. It may save time

by allowing us to observe the functionality of the service for any potential discrepancies, which might be useful to verify during the attacking stages. Any conditions for audio or visual feedback emitted by the micro-mobility device under regular and unexpected conditions, such as beeping, could be considered.

The general pattern of using a micro-mobility service starts by first interacting with the service via the *user interface*. Commonly, the technical solution for this is provided as a smartphone application. The user interface is then used to provision a rental scooter or another *micro-mobility device*. Service functionality such as authentication is provided by the *back-end service*. The user and service data itself must be kept in some sort of *data storage*. As such, we define the three main components of a micro-mobility service as illustrated by Figure 5. These components were derived after observing and using the micro-mobility service as a regular user. Additionally, we observed a few other micro-mobility service providers from the user perspective. The threat models discussed in the following chapters may partially apply to micro-mobility services, which include other components.

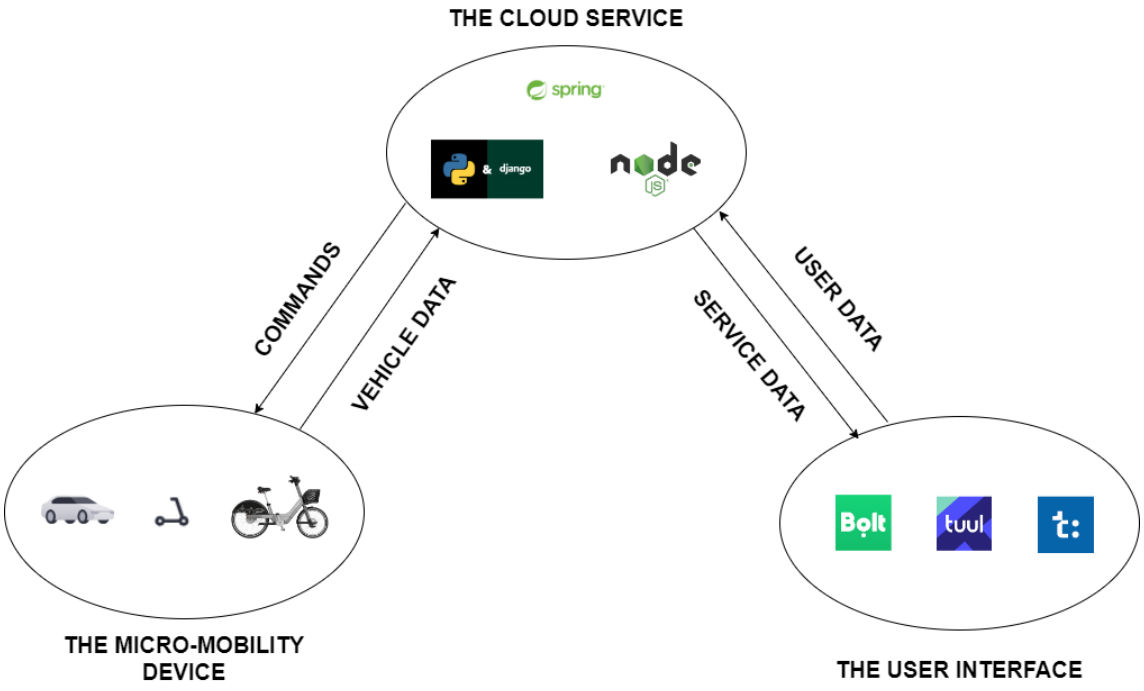

Figure 5. *Components of a modern micro-mobility ecosystem.*

The three abstract components will be the main focus for modeling the data flow diagram. The components communicate with each other as displayed in Figure 5. The user interface, most commonly a smartphone application, is used by end-users to interact with the micro-mobility service. The user interface sends user data such as names, credentials, geographic location, and payment details to the back-end service. The smartphone applications are usually available for download on Google Play and iOS App stores.

The back-end service is a cloud-based web application developed and maintained by the service provider. It communicates with all the user interfaces, sending them service data related to authentication, device provisioning, ride and payment histories, and other service functionality. Additionally, the back-end service interacts with the micro-mobility devices by issuing commands. Furthermore, the back-end service allows the service provider to administrate and maintain the micro-mobility device fleet and its users.

The micro-mobility devices are objects located in the physical world. Examples include scooters or bikes, both electric and pedal-powered. These devices are special due to the fact that they communicate with the back-end service over a communication channel. This channel is most commonly a mobile broadband service such as 3G or 4G. They send vehicle data, such as timestamps, current location, battery level, health data, version info and other relevant logs to the back-end service. A user with sufficient permissions within the service can interact with and use the devices. For example, an authenticated user may use their smartphone application to rent an electric scooter for commuting around the city.

4.3 Data-Flow Diagram

There are cyber threats associated with each of the components and the system as a whole. It is useful to graph the flow of data between the system components and external entities. This allows us to derive cyber threats. Having defined the data-flow diagram elements and described the notation, we now provide a data-flow diagram for the three components derived in Figure 5. Figure 6 describes a micro-mobility service as a data-flow diagram.

Data flows into the back-end service component via user interfaces. User interfaces are used by regular users of the service. Thus, we can consider the user interface an external entity to the service, because it may send arbitrary data directly to the back-end service. The management data sent by the service admins or technicians differs from the user data as it contains configurations. Inversely, the operational data is different than regular service data as it can contain privileged data about logs, configurations and operational fleet status. The back-end service processes the data sent by the user interfaces by either passing on relevant commands to the micro-mobility devices or sending service data back to the user interfaces. Additionally, the back-end service interacts with a data store. Cloud storage is used to store all of the data used by the service. The back-end service process will store and retrieve this data.

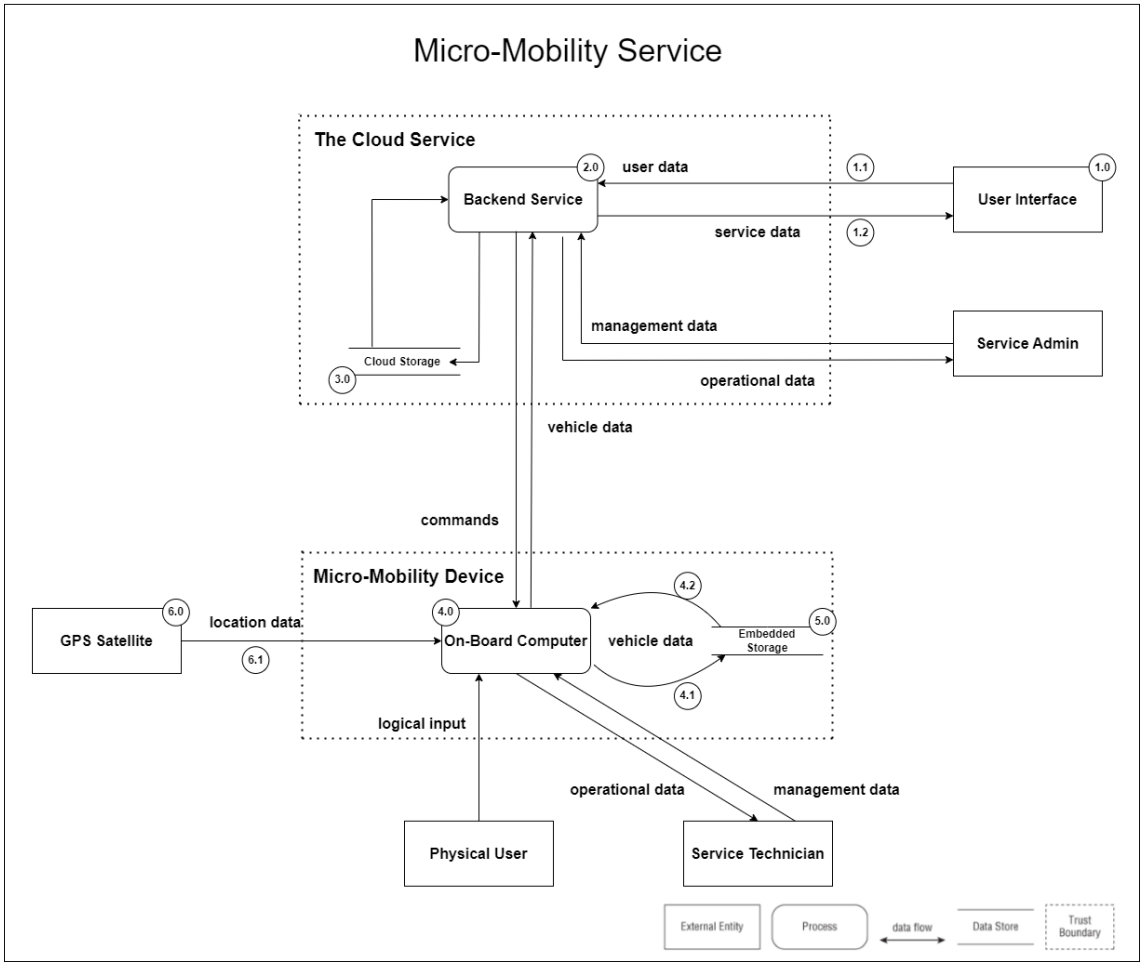

Figure 6. *A micro-mobility service as a data-flow diagram.*

The back-end service process interacts with the micro-mobility devices over a wireless medium by communicating with their on-board computers. As a micro-mobility device is a cyber-physical system and its on-board computer is running software, we consider it a process within the data-flow diagram. The on-board computer accepts commands from the back-end service and returns vehicle data to the back-end service. The micro-mobility devices include an internal data store, the embedded storage, which stores vehicle data and other configurations locally. Data is stored and retrieved from this data store by the on-board computer process. This data is provided to the back-end service when required. Figure 7 illustrates an on-board computer module used by a micro-mobility service.

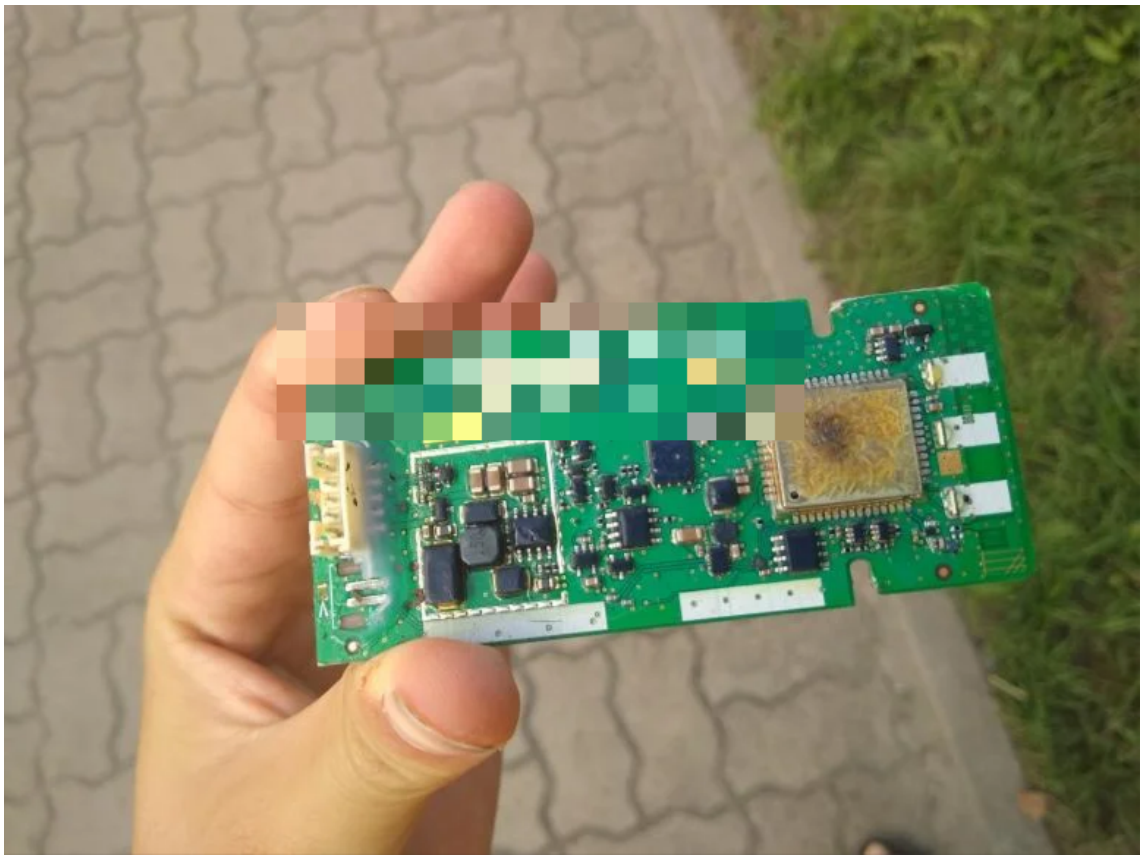

Figure 7. *An on-board computer of a micro-mobility service*[1].

As service providers need to track the location of their micro-mobility devices, there exists a unidirectional data flow from GPS satellites to the GPS module of the on-board computer. The data flow provides location data to the respective sensors we consider a part of the on-board computer process. Additionally, we note that all of the service elements reside within a trust boundary. The trust boundaries were derived by considering the transit of data from less privileged sections to higher privileged sections. We consider any data sent by external entities to be possibly untrusted and thus they reside outside of the trust boundaries. The data flow between the on-board computer and the back-end service is trusted within our scope. However, it is possible to subject the micro-mobility device to a number of attacks, which might result in a potential integrity violation. This may cause the data sent by the micro-mobility device to become untrusted. Thus, we have separated the trust boundaries for the entire cloud service and the micro-mobility device.

The data-flow diagram shown in Figure 6 contains additional external entities and data flows, which are not numbered. These external entities and the corresponding data flows are out of the scope of our research, as we are focusing on the perspective of a regular user using the service. We added them to acknowledge their presence. For example, a malicious

[1]Reddit Gallery - https://www.reddit.com/gallery/wyh6ez (20.10.2022)

service admin or technician may abuse their status as an insider to tamper with data flows. We do not consider attacking such external entities in our work. We did not apply STRIDE to data-flows, where such out-of-scope attacker presence is required. Examples include attacks against the organization by compromising the personal computer of an employee. The physical user is drawn to illustrate the presence of the user on a scooter and the logical inputs they give physically. For example, attempting to move a locked device or increasing the speed of the device by pressing the corresponding handle. We do not apply STRIDE to this external entity and data flow, however, some business logic scenarios could still be tested. Additionally, we consider the potential threat actor to be a technically competent hacktivist or a cybercriminal capable of executing the discussed attacks. However, we acknowledge that even a technically less skilled malicious user may be able to lead the system to an erroneous state.

4.4 Identifying threats to the system

Figure 6 illustrates in-scope elements of our model. We can use the diagram to generate a table for a better overview of all the elements and their respective element types. Table 2 displays all the data-flow diagram elements of the system in accordance with their element type.

DFD Element Type	**DFD Item Number**
External Entities	User Interface (1.0) GPS Satellite (6.0)
Processes	Backend Service (2.0) On-Board Computer (4.0)
Data Stores	Cloud Storage (3.0) Embedded Storage (5.0)
Data Flows	UI and Back-end Service (1.1, 1.2) OBC and Embedded Storage (4.1, 4.2) GPS Satellite and OBC (6.1)

Table 2. All in-scope data-flow diagram elements and their respective types.

We discovered during our modeling and threat elicitation phase, that we cannot directly apply the table proposed by Microsoft as we are viewing the system quite abstractly and as a black-box. For example, the information disclosure threat for the GPS satellite data flow is not applicable as the data flow does not contain sensitive data. Combining Table 2 and Table 1 allows us to map the data-flow diagram elements and STRIDE memnonics to highlight the threats faced by the data-flow diagram elements. Table 3 presents a combined table highlighting the threats faced by each of the in-scope data-flow diagram elements.

DFD Element	S	T	R	I	D	E
Process						
Back-end Service	X	X	X	X	X	X
On-board Computer	X	X	X	X	X	X
External Entities						
User Interface	X	X	X	X		
GPS Satellite	X		X			
Data Store						
Cloud Storage		X	X	X	X	
Embedded Storage		X	X	X	X	
Data Flow						
User Interface and Back-end (1.1 -> 2.0 -> 1.2)		X		X	~~X~~	
OBC and Embedded Storage (4.1 -> 5.0 -> 4.2)		X		X	X	
GPS Satellite and OBC (6.1)		X		~~X~~	X	

Table 3. DFD elements mapped to STRIDE threats.

In Table 3, we have applied a process called reduction [5, Ch. 9], which allows for reducing the number of data flows, if they use the same communication technology and share the same external entity and process. Additionally, we have applied the reduction process to data flows inside the trust boundaries, where data is of the same nature and over the same communication technology. This allows us to reduce the number of entities to analyze.

Having outlined all of the threats that the elements face in Table 3, we can now analyze potential attacks against the components. Applying the threat tree patterns produced by Howard and Lipner in Appendix 2 to all of the data-flow diagram elements allows us to generate threat trees. The proposed threat tree patterns help guide our focus during the penetration test of a micro-mobility service. We omitted attacks, which did not apply to our case study.

4.4.1 Spoofing

The external entity and process elements are vulnerable to spoofing as displayed by Figure 17 in Appendix 2. An attacker can obtain legitimate credentials from a weak client-side interface or server-side storage and use them to spoof the back-end service. Additionally, an attacker may be able to leverage insufficient authentication to conduct spoofing. An on-board computer process with weak or no authentication may be spoofed if an adversary has physical access. They may potentially assume the role of another device by installing modified firmware. Legitimate credentials may also be guessed by brute-force attacks.

External entities are vulnerable to spoofing as a malicious actor may pose as another user or even an administrator by falsifying credentials or leveraging hard-coded credentials. Additionally, an attacker could run a brute-force attack with common usernames and passwords to achieve spoofing capabilities. GPS satellites may be spoofed as attackers might falsify the location data received by the on-board computer [32] and back-end service process.

4.4.2 Tampering

The process, data flow, and data store elements are vulnerable to the tampering threat as displayed by Figures 18, 19, and 20 in Appendix 2. In our system, the user interface can also be tampered with, as the adversary may statically patch and repack the user interface. Attackers may also hook and modify certain functionality of the user interface dynamically during runtime. The back-end service may be tampered with by providing false credentials or sending malformed inputs and observing the result. Additionally, physical access to the memory of the on-board computer could allow an attacker to lead the process to a corrupt state by connecting to the on-board computer or modifying circuitry.

A data store may be tampered with if there does not exist a sufficient protection scheme and an attacker can write data. Additionally, an attacker might attempt to access the data store directly, without following regular access logic. A data store may be vulnerable to over-capacity failures as an attacker might attempt to upload large amounts of data. As the data store becomes full, any additional data may be discarded or overwritten and service providers should be prepared for the scenario. This tampering threat may further cascade to allow repudiation threats on the data store. Cloud providers often provide automatic scaling for the data stores and thus such attacks could cause additional costs for the service provider.

An attacker could tamper with the data flow by executing a man-in-the-middle attack. This may allow to compromise the integrity of the channel and messages. Attackers may be able to modify requests in transit. Weak message integrity may allow for replay attacks or message modification (false data injection). The GPS data flow is vulnerable to tampering as attackers may use special hardware and software to manipulate GPS signals [18] [32]. Additionally, an attacker might tamper with the data flow between the on-board computer and the embedded storage using special tools.

4.4.3 Repudiation

The repudiation threat is faced by the process and external entity elements as depicted by Figure 21 in Appendix 2. For this threat is important to assess, whether we could influence the logs or the logging application for the processes. Non-trusted access to logs may allow an attacker to cover their activities by modifying the logs. As a penetration tester, it is not possible to assess the granularity of logs from the service provider side. If the logs were available, the granularity and authentication level of the logs could be analyzed, as only trusted data should be logged. The level of logging should be sufficient to repudiate any malicious claims. We consider the user interface an external entity and can analyze the logs stored on the local device instead. Additionally, we can analyze the possibility of replaying client-side HTTP requests. For the GPS satellite, attackers could attempt to replay GPS signals [33].

Data stores may be vulnerable to the repudiation threat if they contain data related to auditing or service logs. Due to our abstract and contextual diagram, we do not differentiate between the different data stores the service might have and consider a single cloud and embedded storage. On an abstract level, they likely contain audit and logging data and thus are potentially vulnerable to the repudiation threat. Tampering with the cloud storage component might allow attackers to modify such logs. Alternatively, an attacker might directly access and attempt to modify the logs within the embedded storage.

4.4.4 Information Disclosure

A process, data flow, and a data store can be vulnerable to the information disclosure threat as depicted by Figures 22, 23 and 24 in Appendix 2. Input validation failure errors and physical memory access may disclose privileged information from a process. Additionally, a process may simply return sensitive data to the client due to a development oversight. An attacker can use fuzzing software against the service endpoints to observe the returned service data for any sensitive data. A potential adversary could access the on-board computer over a wired medium and attempt to disclose sensitive data. In our example, the user interface is vulnerable to information disclosure as the source code or memory of the user interface may disclose sensitive data.

A data store must be protected with access controls as an adversary may be able to disclose sensitive data by abusing weak permissions and performing arbitrary read actions. The same is possible if there exist no access controls at all. If the data store offers data deletion functionality, such as account deletion, the data must be correctly removed. Incorrectly

removed data may allow an attacker to disclose and reuse the old data, should they find a way to access the data store. Additionally, an attacker might attempt to access the data store directly, without following regular access logic.

A data flow containing privileged information may be read by an attacker executing a man-in-the-middle attack. The service should be designed so that overly sensitive data could not be observed by man-in-the-middle attacks. The data flow between the on-board computer process and the embedded storage could be potentially read by an adversary using tools such as a logic analyzer. However, this threat does not apply to the location data flow from GPS satellites as the data is public and not sensitive.

4.4.5 Denial of Service

The process, data flow and data store elements are vulnerable to denial of service attacks as described by Figures 25, 26 and 27 in Appendix 2. For processes, attackers may abuse existing service functionality to consume application-specific resources by writing a script or using a tool, which constantly calls for resource-consuming functionality. This scenario also applies to data flows and data stores as most such attacks are aimed at the process serving the data store or flow [5, Ch. 9]. The attackers might tamper with the on-board computer process via physical access. This can potentially result in a denial of service vector.

Denial of service against a data store could be achieved by corrupting the data in the data store or finding a way to deny access to the store by tampering with configurations. Denial of service against a data flow involves corrupting the message within the data flow or incapacitating the channel used for the data flow. For example, it is possible to incapacitate the data flow from a GPS satellite to the on-board computer of a micro-mobility device. This could happen to data-flow 6.1 in Figure 6 as an adversary may utilize special hardware to conduct GPS jamming [32] or take the micro-mobility device to a location, where GPS signal is weak. An attacker could incapacitate the channel between the data flow of the on-board computer and the embedded storage via physical tampering. The data flow between the user interface and back-end service we do not consider for denial of service as it is under attacker control. Attacking this data flow for other users is out of scope.

4.4.6 Elevation of Privilege

The elevation of privilege threat is relevant only for the process element and the corresponding threat tree is depicted by Figure 28 in Appendix 2. Privilege escalation can be

directly caused by failed input validation and weak or insufficient authorization. Spoofing and tampering attacks may also lead to privilege escalation. An adversary may connect to the on-board computer of a micro-mobility device over a wired medium and elevate their privileges if there exists no authentication logic. Additionally, an attacker might tamper with a data store, such as the user database, and assign themselves the admin role directly in a static manner. If the schema used for authorization is weak and can be tampered with by an adversary, the attacker might be able to elevate their privileges. For example, an attacker could modify the contents of an authentication token to gain higher privileges.

5. Penetration Test of a Micro-Mobility Service

This chapter will describe the reconnaissance and exploitation stages of the penetration testing methodology, which were conducted on a real micro-mobility service provider. We provide an overview of the process by following our elicited threat trees for penetration testing and discussing the discovered results. We note that in Chapter 4, the numbered elements of the data flow diagram were discussed to be our scope for the research. Due to restrictions from the service provider, the scope of the penetration test is reduced to only include the user interface, back-end service, and cloud storage components. This includes the data flows between the user interface and the back-end service. The on-board computer and the embedded storage remain out of scope for the penetration test at the service provider's request as they did not wish to disclose their intellectual property and risk physical damage. However, we were allowed to use the scooters for tests, which did not require disassembling any components.

5.1 Reconnaissance

The reconnaissance stage focuses on exploring publicly available resources, such as the service provider's website and the user interface component. The user interface is a critical component of the service, which the users interact with directly and thus holds a vast amount of technical data regarding the service. Thus, one part of the reconnaissance stage will focus on Android application analysis. Additionally, parts of the back-end service or cloud storage may contain publicly browsable directories or hidden subdomains and files. Even job advertisements present on the service provider's website can reveal clues about the used technologies[31]. We have graphed the applicable information-gathering attacks, which allow us to discover technical details regarding the service in Figure 8.

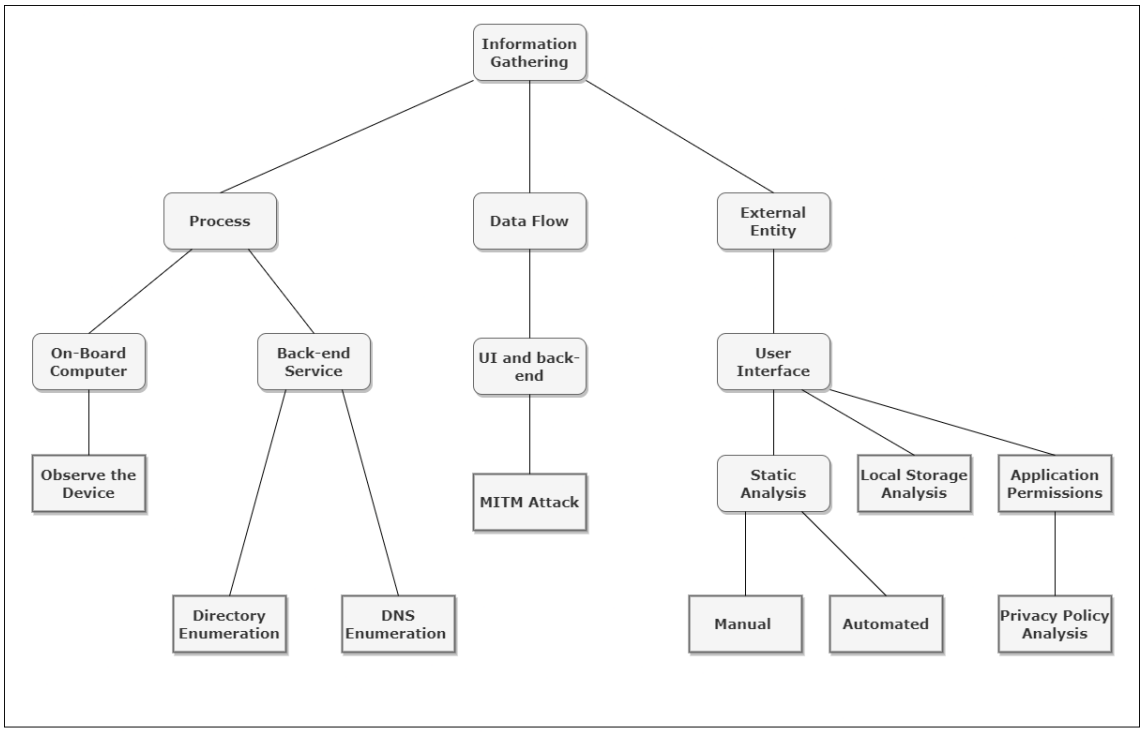

Figure 8. *The reconnaissance stage as a threat tree.*

We categorized reconnaissance activities by components. For example, we can conduct reconnaissance on the user interface component by conducting manual and automated static analysis on the Android application. Additionally, our reconnaissance stage already contains a MITM attack and thus, there is an overlap with the information disclosure threat tree, where this attack is also prevalent. We have kept the duplicate attacks in both trees for better visual illustration. The proposed information-gathering attacks were adopted from guides relevant to system components. As the back-end service is a web application and the user interface is a mobile application, we chose attacks from the OWASP Web Application Security Testing Guide [26] and the OWASP Mobile Application Security Testing Guide [27]. The DNS reconnaissance techniques were adopted from the Penetration Testing Framework [25].

5.1.1 Observing the device

Simply observing the micro-mobility device visually may potentially reveal some technical data regarding the micro-mobility service. We observed the service provider's scooters visually in an attempt to evaluate if any serial numbers or sensitive QR codes were present on the frame of the scooter. We fully examined all sides of the scooter but found no sensitive or technical data on the frame of the scooter. The handlebars contained a QR code, which describes the unique shortcode of the device, however, the data is not sensitive.

Additionally, the frame had instructions on how to use the service and where the user interface could be acquired from. The lower frame of the scooter handlebars had GPS-related markings, indicating that the scooters are tracked.

The frame of the bike was firm and no loose parts were observed. Any disassembly would require tools such as screwdrivers and hex keys. There were several screws within the base of the scooter, where we assumed the electric battery to exist. No obvious markings were observed to deduce the location of the on-board computer of the scooters within the frame. We used the beeping functionality of the scooter, in an attempt to discover the location of the on-board computer. The beeping functionality emits a small and continuous sound from the scooter to distinguish it from other nearby scooters. This functionality can be triggered from the user interface. The beeping sound came from near the base of the scooter as opposed to near the handlebars. As we never actually disassembled the scooter, we did not locate the on-board computer. However, a malicious adversary could employ similar tactics and tools to disassemble and interact with the on-board computer physically.

5.1.2 Enumeration

As the back-end service component is vulnerable to information disclosure, we can enumerate it for files, directories, and other possible subdomains. There may exist directories, which are not protected by access controls and allow an attacker to disclose sensitive data. In order to broaden the attack surface, we first used DNS enumeration techniques in an attempt to discover related virtual hosts and subdomains. We used the results of the DNS enumeration during a directory enumeration attack, where the existence of commonly known resource locator paths was tested via a word list. The techniques used to perform the tests are adopted from the OWASP WASTG [26] and the Penetration Testing Framework [25].

The main domain of the service provider we already know from having used the service for its intended purposes. We used the DNSRecon enumeration script[1] to discover the actual IP of the website. The DNS mail records revealed that the service provider is using Google services for emails. Additionally, we found a subdomain for their merchandise shop, which is created using Squarespace. DNSRecon revealed three DNS name servers, for which we attempted a DNS zone transfer attack. A zone transfer attack attempts to replicate the contents of a zone file from a DNS server. The transfer requests were correctly refused and thus nothing was disclosed. We also used a DNS enumeration script called DNSEnum[2], which was able to discover several subdomains via a word list-based approach. DNSEnum

[1]DNSRecon - https://github.com/darkoperator/dnsrecon (09.11.2022)
[2]DNSEnum - https://www.kali.org/tools/dnsenum/ (09.11.2022)

discovered the administrator portal of the service provider. We compared the enumeration results with the NMAP DNS enumeration script and discovered no additional domains. Finally, the Fierce DNS enumeration tool[3] discovered a new subdomain, which the other tools had missed. The subdomain was a self-service portal, where privately sold scooters could be registered for personal use. This is a separate business venture of the service provider, however, the domain name coincides with the public service. Conclusively, DNS enumeration allowed us to discover several subdomains, thus broadening our attack surface for further reconnaissance and exploitation.

Having discovered several subdomains related to the service, we used the OWASP Dirbuster tool[4] to enumerate the files and directories present on these web domains. We used an out-of-the-box wordlist provided by OWASP, which contained 87000 common directories and file names discovered on at least three different hosts in the past. We enumerated the different subdomains, which we discovered during DNS reconnaissance. We limited the number of requests sent by the tool in order to reduce the load on the service. We used the default scanning options for directories and followed up with file extension enumerations on potentially sensitive directories. We used the file extensions TXT, PHP, and PDF. We discovered that the service provider's informative website is built using WordPress and uses a custom WordPress theme. Inside the custom theme folder, we found a redundant Javascript file used for development, however, the contents were not sensitive. We also ran a WordPress security scan using WPScan, which revealed four outdated plugins. We recommended the service provider update the plugins and remove the old development file. After analyzing other directories revealed by the initial enumeration, we found no sensitive data being disclosed by the different subdomains of the service provider. The limitation of our approach is the use of a single and known word list. Although common, the keywords may not have discovered all directories and files and thus sensitive data may still exist in a public form. However, we were not able to disclose any sensitive data using this word list on the discovered subdomains of the service provider.

5.1.3 Static Analysis

During static analysis, we seek to better understand the architecture of the service and search the source code of the user interface for any resource identifiers and secrets, which may allow for information disclosure. Additionally, we analyze the source code related to the service functionality. For example, we analyze the unlock functionality to discover how the service requests unlocking for micro-mobility devices. We search for checks, which are done on the client side, and evaluate their abuse potential.

[3]Fierce - https://github.com/mschwager/fierce (09.11.2022)
[4]OWASP Dirbuster - https://www.kali.org/tools/dirbuster/ (13.11.2022)

The user interface of the service provider is a smartphone application available for Android and iOS devices via either Google or iOS app stores. In order to analyze the smartphone application we must first acquire the binary of the application. We chose to analyze the Android application due to its greater market share [34], robust documentation, and availability of analysis tools. We acquired the latest Android APK version of the service provider as of October 2022 from APKPure[5] and used the MobSF online analyzer[6] to conduct initial automated static analysis. MobSF automatically analyzes several core components of the application along with metadata [35]. This allowed us to gain an overview of the common security issues the user interface might have and acts as an entry point for further manual analysis. Figure 9 displays the security scoreboard of automated static analysis for the Android application of the service provider.

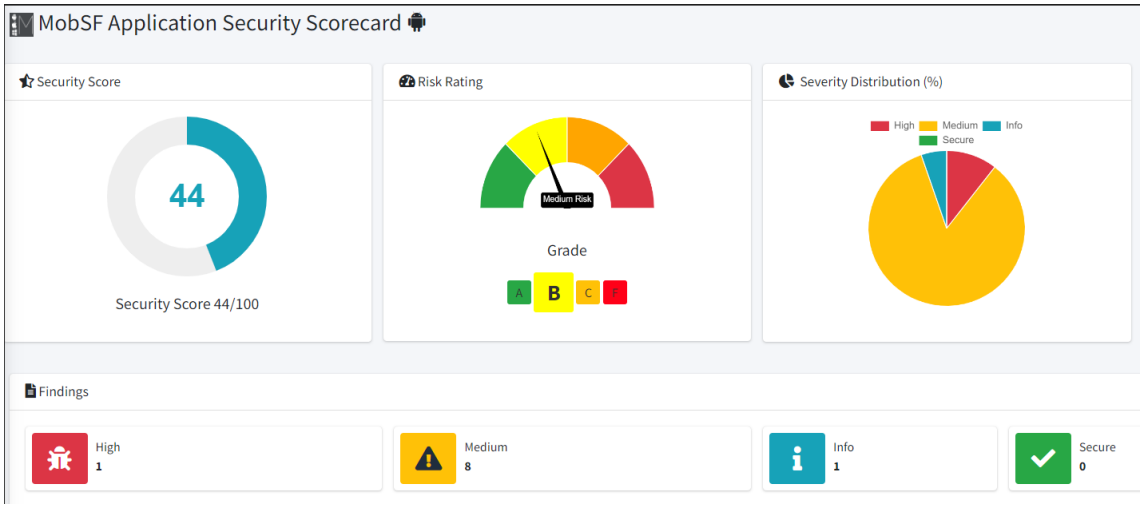

Figure 9. *Security scorecard for the Android application of the service provider.*

The medium security score assigned by the MobSF analyzer does not immediately mean that the service is at risk. Rather, the results should be used as a point of reference for manual analysis as the highlighted risks may not necessarily manifest. The most notable discoveries from the automated static analysis were as follows.

1. Evidence regarding service architecture (Payment provider, Cloud service).
2. Several uniform resource locators and public secrets.
3. Android permissions requested by the user interface.

The automated analysis revealed a lot of libraries and classes containing the keyword *React*. This indicates, that the smartphone application was created using the React Native[7]

[5]APKPure - https://apkpure.com/ (10.11.2022)
[6]Mobile Security Framework Online Analyzer - https://mobsf.live/ (11.11.2022)
[7]React Native - https://reactnative.dev/ (13.11.2022)

development framework. React Native allows an application to be developed with a single framework and deployed cross-platform. We further used 7Zip and JadX[8] to unpack and analyze the contents of the Android APK files. We manually confirmed that the main activity class of the Android application extends another class called *ReactActivity*. Thus, the underlying code base is the same for both Android and iOS operating systems. The programming language for React Native applications is Javascript.

Usage of React Native means, that we are not able to analyze the source code of the user interface trivially via existing tools. The Javascript source code of React Native applications will reside in the file *index.android.js*. The source code will be minified and bundled into the file *index.android.bundle* during the build process [36]. We extracted the Javascript source code of the smartphone application and discovered that the service provider is using the Hermes Javascript engine. Hermes[9] is a Javascript engine developed by Meta for React Native. Hermes will cause the Javascript to be compiled into Hermes Bytecode and allow for improved start-up times, reduced memory, and storage usage [37]. Hermes compiles the otherwise plain text source code of the application to Hermes bytecode. The source code is unintelligible and Listing 5.1 displays the result of the Linux command *file* on the bundled source code.

```
index.android.bundle: Hermes Javascript bytecode, version 84
```

Listing 5.1. Linux *file* analysis of the bundled source code.

To analyze the source code of the application, we are left with the following options:

1. Use tools from the community or literature to disassemble the bytecode.
2. Analyze the inner workings of the Hermes engine and create a disassembler.
3. Extract the source code from earlier versions of the application, where Hermes was not used.

Analyzing the inner workings of the Hermes engine and creating a disassembler is out of the scope of our thesis. It would be infeasible due to the time, skill and effort required. We are left with the options of searching for existing tools or attempting to find an earlier version from when the Javascript was still bundled in plain text. In both cases, we will not be able to exactly reproduce the original source code as earlier versions portray earlier source code and no tools exist, which would allow recreating the source code to the extent as Java or .NET applications allow.

[8]JADX - https://github.com/skylot/jadx (13.11.2022)
[9]Hermes Javascript Engine -https://hermesengine.dev/ (13.11.2022)

Online tools, such as HBCTool[10] provide disassembling and assembling capabilities for the Hermes bytecode versions 59, 62, 74, and 76. The developers of Hermes have created a tool called HBCDump[11], which provides disassembling capabilities for all Hermes bytecode versions. The bytecode version of the latest release is higher than supported by HBCTool. Thus, we used HBCDump to disassemble the source code. Additionally, we tracked down the earliest release of the service provider's Android application from before Hermes was implemented. We refer to this version of the user interface as the older version.

The earliest version not implementing Hermes on APKPure was released in July 2021. The *index.android.bundle* file of the older release contained stripped and minified Javascript source code. We were able to beautify the source code so it would be easier to analyze. We did not find included a source mapping, which would allow us to remap the minified identifiers. Thus, we used both the plain-text Javascript source code from the older version and the disassembled source code of the latest version during the static analysis. This gave us the ability to manually analyze for further artifacts regarding service architecture, functionality, and resource locators.

We discovered some public keys of the payment provider, alongside with indicate usage of platform-specific payment providers, such as Google and Apple Pay. We also discovered that the Google Firebase cloud service is being used. As the service provider is using Google Firebase, we can get a good understanding of the architecture from the extensive documentation that Google provides [38]. Additionally, usage of Firebase indicates the possibility to search for identifiers related to Google Firebase application configuration options [39]. We searched for relevant keywords to determine the back-end resource locators, field validation logic, and Firebase configuration options. The used keywords can be found in Listing 5.2.

```
"apiKey", "database", "bucket", "command", "lock", "unlock", "reserve",
"api", "appEngine", "cloud", "structuredquery", "http", "validate"
```

Listing 5.2. Keywords used for string search analysis.

The MobSF automatic analysis revealed the Firebase database URL and Google API key from the Android string resource. Combined with manual analysis, we were able to extract the following resource locators and secrets from the source code of the user interface.

[10]HBCTool - https://github.com/bongtrop/hbctool (14.11.2022)
[11]HBCDump - https://github.com/facebook/hermes/tree/main/tools/hbcdump (14.11.2022)

- The Firebase Database URL.
- The Google Cloud Functions and Application Engine URLs.
- The Back-end Service REST Interface URL.
- The Public Secrets of the Payment Provider.
- The Google project API key and Application Identifier.

This data allowed us to later validate, whether these resource locators are actually sensitive and protected by access controls. Storing resource locators on the user interface is not a security risk as long as the back-end service implements access controls. No hard-coded secrets, which would indicate a bad service design or a development oversight, were discovered within the user interface of the service provider. This includes development phone numbers, which are often used for testing Firebase workflows.

Although the source code is not easily interpretable, we were still able to analyze parts of the service functionality by cross-referencing code from both versions. We discovered that the user interface is implementing Google Cloud Functions, which allows the service to run specific code on-demand in response to events. This functionality is commonly used for serverless IoT and mobile back-ends [40]. We found that cloud functions are used to invoke commands on micro-mobility devices. The flow of commands follows the structure described in Figure 5, where the user interface sends an HTTP request to the back-end service. A cloud function is then invoked to send out a command to the scooter. Cloud functions are used for functionality such as beeping, scooter unlocking, and even user deletion. We were also able to discover the resource locators of these cloud functions and use them during the exploitation stage. Additionally, we were able to discover several other resource locators related to back-end service. The source code revealed structured queries used to query the Firebase cloud storage for available scooters. A sample query included in the source code allowed us to understand the database schema and helped with further exploitation attempts on the cloud storage.

Conclusively, we discovered no sensitive information disclosure from the user interface component. Static analysis allowed us to better understand the general architecture and technologies used by the service provider. The discovered technologies were the React Native development framework and the Google Firebase back-end architecture services. Additionally, there exists a Google and Apple Pay integration. We discovered that the service provider is using the Hermes Javascript engine, which by nature implements analysis protections during the build process. As such, the source code of the user interface is not presented in plain text. We discovered several resource locators related to service components, functionality, and Firebase configuration options, such as the location of the back-end service and cloud function interfaces.

5.1.4 Analysis of Permissions

As per the recommended guidelines for working with permissions from Google, a use case should be fulfilled without using any permissions, whenever possible [41]. Dangerous permissions grant the application access to restricted data or actions, which may affect the system and other services running on the system [41]. The purpose and need of the aforementioned dangerous Android permissions should be analyzed. According to our static analysis, the latest version of the user interface requires the following dangerous Android permissions from the client device:

1. ACCESS_FINE_LOCATION
2. CAMERA
3. READ_EXTERNAL_STORAGE
4. WRITE_EXTERNAL_STORAGE
5. RECORD_AUDIO

The Android permission manifest[12] defines all the possible Android permissions and their purpose. According to the manifest, the ACCESS_FINE_LOCATION permission is legitimate as the service must keep track of the user's location in order to provide the navigation map and user location. The CAMERA permission is required in order to scan the QR code of the scooters or add graphic attachments to fault reports. It remains unclear as to why the permission RECORD_AUDIO is required. Acting as regular users of the application, we did not encounter a situation, where this would have been necessary. The functionality to call service support makes use of Android's native phone application and the permission is never requested. We saw traces of requesting audio permissions in the source code of the older interface version in relation to video recording. This permission appears unused and we did not find a way to make the application request the permission from us. Similarly, we were able to use all of the functionality without ever granting the user interface STORAGE-related permissions. Additionally, the user interface defines a permission called SYSTEM_ALERT_WINDOW, which allows an application to display alert windows for any functionality purposes. As long as an application does not have malicious intent, the permission is not of concern. However, compromise from the service provider side may allow attackers to display over the screen of the devices. We conclude, that we discovered some potentially redundant permissions during static analysis as the service never requested the RECORD_AUDIO and STORAGE-related permissions. Thus, the Android manifest may not need to define them.

[12]Android Permission Manifest - https://developer.android.com/reference/android/Manifest.permission (18.11.2022)

The privacy policy of the service provider is valid as of January 2022 and defines their approach to collecting and handling user data. It is important to compare the privacy policy to the observed reality as the user interface may disclose more data to the service provider than discussed in the privacy policy. The service provider claims to use data only in ways necessary to provide the business and services. The privacy policy details the collection of data during visits to the service provider's website and other service-related interfaces. This includes the camera and location functionality of smartphones. Additionally, location and vehicle data from the scooters is being regularly collected. The service provider defines the collected data in four different parts.

- Commercial Data
- Transactional Data
- Usage Data
- Profile Data

Commercial data includes user data such as phone numbers, email addresses, names, payment details, and passwords. The service provider claims to store passwords in an encrypted format and not in plain text. This data is processed for contractual reasons related to delivering the service and identifying the user as a person. In case of traffic offenses or vehicular damage, this data may be transferred to insurance companies or other authorities, such as the police. The data retention period is not specified and remains active until deletion is requested by the user. Transactional data includes data regarding payments and invoices created when using the service. This data is also processed by a third-party payment provider used by the service provider. This data is retained until limitation periods for claims expire. Service usage data contains user location data, login details, behavioral data, vehicular data, and identifiers regarding the devices used to access the service. The data is used to provide the service and analyze customer behavior. This data is retained until deletion is requested by the user. Profile data includes marketing and communication preferences. Data shared with the service provider regarding any feedback or surveys is also retained under this section. This data is used for statistical purposes and potential customer profiling. The data is retained until user consent is withdrawn or the contract has been terminated.

The user is required to agree to this privacy policy upon registering. Based on what we have observed, the service is not collecting from the users more than what they explain within the privacy policy. While the user interface is defining the RECORD_AUDIO permission within the manifest, it is never actually requested from the user. The data retention periods are defined and the users are equipped with the possibility to request that their data be deleted.

5.1.5 Local Storage Analysis

The local storage of a client device should not contain any sensitive assets or personally identifiable information as it can be easily analyzed and disclosed by an adversary. Insecure data storage ranks as the second most common mobile application security risk in the OWASP Mobile Top 10 [42]. Within our methodology, this falls under the information disclosure threat from the user interface component. The impacts of the disclosed sensitive data can range from identity theft of a single user to material loss for the business [42].

We used a rooted Huawei Honor 8 phone with the Android operating system to access the otherwise protected data storage of the user interface. The capability was provided to us by the Android Debug Bridge, which allowed us to browse the local file system of the phone. We analyzed the files and folders present on the local storage and executed several storage related test cases from the OWASP MASTG [27]. We analyzed the usage of the SharedPreferences API, which allows the user interface to store small collections of key-value pairs locally on the client device [43]. We found, that the service is caching the following data using the SharedPreferences API.

- The refresh token.
- The Google AppCheck token.
- The latest authentication token.
- The unique user identifier.
- The phone number of the user.

Although the authentication and refresh tokens are sensitive, they are commonly cached on the local storage for improved service usability. Still, the identity of a user can be compromised by accessing the storage of a lost or stolen device. We found no other sensitive data within the directories used to store the shared preferences.

The Android operating system provides functionality to use SQLite databases as a way to store data locally on the device. Although the local storage of the user interface contains a folder for databases, the contents and databases stored inside the folder were not found to contain any sensitive service or user data. As highlighted by the IoT Penetration Testing Cookbook [44, Ch. 5], the contents of the locally stored cookies and web data folders were analyzed. The HTTP cache of the user interface and local files inside the storage did not contain any sensitive data. The permissions to access the files inside the local storage were only assigned to the local service user and group. We discovered, that the user interface is logging various data via Firebase analytics directly to the back-end service.

Conclusively, we found no sensitive or personally identifiable data within the local storage of the user interface. Sensitive details, such as credit card information, ride history, and user data are fetched remotely from the back-end service and appear to be stored in memory.

5.1.6 Dynamic Analysis

As highlighted by Figure 6, there exists a data flow between the user interface and the back-end service (elements 1.1 and 1.2 on Figure 6). For our test case, this translates to the Android application communicating with the Firebase service. It is possible to analyze the contents of this data flow by executing a man-in-the-middle attack on the data flow. Tools such as MITMProxy or Burpsuite provide the capability to route communication through a special proxy. This allows an attacker to observe, capture and tamper with the data moving between the user interface and the back-end service.

To execute a man-in-the-middle attack, we needed to utilize several functionalities of a rooted Android phone. We added the self-signed certificate authority of the proxy to the operating system's trusted certificate list. For this, we utilized a Magisk module[13] to trust user-added certificates on the system level. We also discovered, that the service is implementing Firebase App Check [45], which helps to protect remote resources against unauthorized requests by attesting to device and application integrity. We modified the Android device fingerprint of our phone via another Magisk module[14] to pass the required basic integrity checks.

After setting up a proxy between the data flows 1.1 and 1.2 displayed in Figure 6, we discovered that user and service data is not completely passed through the proxy. The main functionality of the user interface was unresponsive. Only some of the network requests for user profile data, ride state, and geographic area data were being proxied. Data regarding payment details, available scooters, ride history, and cloud functions was not proxied. We searched the source code of the user interface for SSL pinning but were unable to discover any custom pinning. We tampered with the user interface component using the Objection analysis library[15] and hooked relevant network functionality during execution to disable basic SSL pinning. However, this did not enable us to fully listen to every request. Many of the requests we were not able to proxy due to the protections implemented by the user interface and the Firebase App Check service. Conclusively, dynamic analysis alarmed us of some existing protections of service components and revealed some new resource locators related to the authentication flow. In Section 5.2.2 we further used dynamic analysis to tamper with some of the requests we were able to proxy.

[13]Magisk Trust User Certs - https://github.com/NVISOsecurity/MagiskTrustUserCerts (15.10.2022)
[14]MagiskHide Props Config - https://github.com/Magisk-Modules-Repo/MagiskHidePropsConf (15.10.2022)
[15]Objection - Runtime Mobile Exploration - https://github.com/sensepost/objection (15.11.2022)

5.2 Exploitation

During the exploitation stage, we leveraged the results of the reconnaissance stage and followed the threat trees to conduct further tests, which attack the security preconditions related to all STRIDE threats. From the reconnaissance stage, we discovered that the service provider uses Google Firebase as their back-end interface. Google Firebase is a product provided by Google to their clients and is something the service provider uses. This means that we will not be analyzing the security of Google Firebase products directly, rather we analyzed how the service provider uses Firebase and whether best practices are followed.

As mentioned in Chapter 4, not all elements are vulnerable to all STRIDE threats. The attacks in the exploitation stage focus on security-related preconditions for all applicable STRIDE threat and component combinations. We have categorized the attacks by components to create more structure for the threat tree. For each STRIDE threat, we included a threat tree in the respective section. We separated the threat trees to keep the trees within a reasonable size and not create an overly big threat tree, which would be harder to follow. It is important to note that the threat trees do not contain all possible attacks and components due to restricted scope. Several attacks, such as directly accessing memory and other physical tampering are out of scope and thus not graphed. As we did not graph cascading threats, some threat trees may only consist of a single attack. A real attacker, however, would be able to attack all of the possible security preconditions highlighted in Appendix 2. Thus, our exploitation stage does not consider all possible attacks for all possible components of the entire micro-mobility service.

5.2.1 Spoofing Attacks

The user interface and back-end service components are vulnerable to spoofing. There are several security preconditions, which can allow for spoofing as highlighted by Figure 17 in Appendix 2. For our threat trees, we chose attacks based on the preconditions. For example, tampering with the authentication process, guessing legitimate credentials, or finding legitimate credentials in client-side storage are some of the preconditions, which can lead to spoofing. As we are dealing with a web application and an Android application, we can choose attacks from the OWASP WASTG [26] and MASTG [27] or incorporate the tests proposed by Howard and Lipner [5, Ch. 22]. From these sources, we chose various attacks such as modifying our authentication token, analyzing the client-side storage for legitimate credentials, and executing a brute-force attack. We follow the same approach for each of the STRIDE threats and thus the derived threat trees contain attacks, which are based on the security preconditions highlighted by Howard and Lipner [5].

Figure 10 illustrates attacks, which test for spoofing-related security preconditions on the components of the micro-mobility service. We note that, while brute-force attacks were out of scope for this study, they were included as a general attack vector for an otherwise in-scope component.

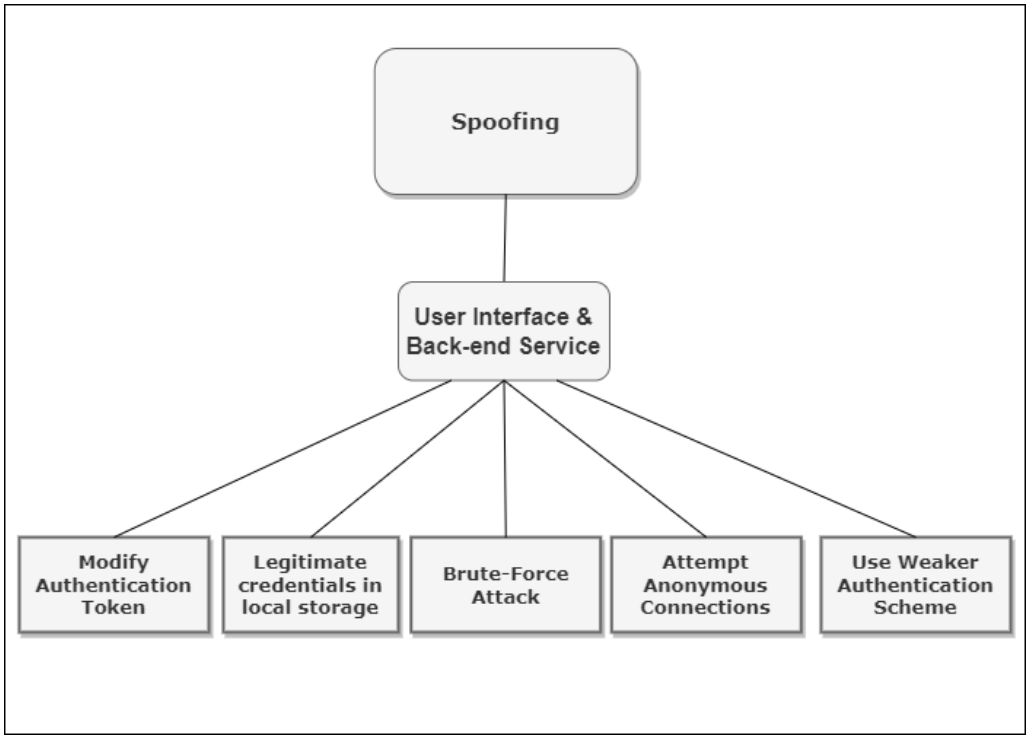

Figure 10. *The spoofing threat tree.*

Authenticating to the micro-mobility service requires the user to enter their phone number. There is no conventional email and password authentication and having access to the relevant SMS messages of the phone number will act as proof of identity. Figure 11 displays the authentication view as seen from the user's perspective.

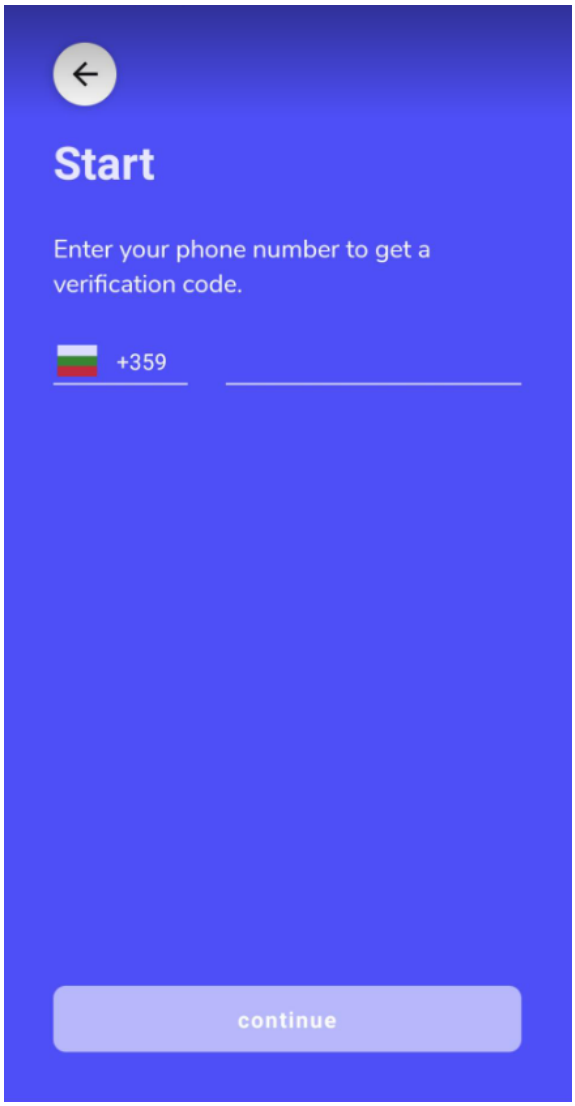

Figure 11. *The authentication interface.*

After entering the phone number and pressing the login button an SMS from the sender CloudOTP is received. The user must then enter a 6-digit code received via the SMS and the service will then provide an authentication token to the user. Several real examples of the received verification codes are provided below. They revealed no obvious repetitive patterns.

- 586495
- 096711
- 577617
- 615659

Google Firebase provides several different methods of authentication and CloudOTP [46] refers to the service of providing one-time passwords via a cloud service. While

authentication via the phone number is convenient, it is less secure than other methods due to the easy potential transfer of a phone number and usage restrictions on multi-profile devices [46]. The authentication request is sent to the Google Identity Toolkit[16], which Firebase uses to request a JSON Web Token (JWT).

JSON Web Tokens are a means of representing claims between two different parties [47]. The JWT acts as an authentication token for HTTP requests between the user interface and the back-end service. A JWT contains three base64 encoded parts separated by dots, which are the header, the payload, and the signature. The claims inside the payload of the JSON Web Token are encoded as a JSON object and can be used to implement access controls. The JWT is signed using a secret of the service provider and thus cannot be modified. During the dynamic analysis, we captured our JWT and analyzed the claims within. The lifetime of the issued tokens was one hour. The identity was tied to the phone number of the user and the audience is restricted to the respective Firebase database of the service provider. The JWT claims contain our unique user identification code, which the service provider could use for authorizing the usage of resource locators within the back-end service.

The usage of OTP-based authentication makes guessing legitimate credentials harder as the service is not using conventional email and password authentication. A conventional brute-force attack cannot be executed to guess any credentials. There is no way for the password policy to be weak as only one-time passwords are used. We were not able to modify the claims inside the JSON Web Token issued to us because it is signed. We would be able to spoof another user if we were able to acquire a JWT with another unique ID. However, any attempts to reconstruct the token with modified claims will invalidate the signature and thus the token itself. We attempted to downgrade authentication by tampering with the signature generation algorithm of the token and changing it to none. We observed, that responses of the back-end service for our requests with tampered JSON Web Tokens were rejected. Using Postman, we attempted various anonymous connections to the back-end service, which did not contain any authentication token. We attempted to read and modify our own data, but all of the anonymous requests were denied. Additionally, we were not able to find any service credentials within the client-side storage and source code of the user interface, as described in Sections 5.1.5 and 5.1.3. Thus, we were also unable to spoof the back-end service process.

To compromise the identity of another service user, an adversary would need to know the mobile phone number of an existing user and be able to steal the one-time password sent to the phone number via SMS. Alternatively, the user's JWT would have to be stolen

[16]Google Identity Toolkit - https://cloud.google.com/identity-platform/docs/reference/rest (06.06.2022)

via physical means such as extracting it from a stolen device. It is worth noting that the service provider allows multiple devices to authenticate under the same identity and does not provide any security warnings regarding authentication from new devices. The service provider does not offer any forms of local authentication such as screen lock PIN, password, or biometrics.

We conclude that while phone number authentication is not the securest when compared to alternatives [46], we found no fault with the implementation. The core authentication logic is provided by the Google Identity Toolkit and no implementation concerns were discovered during the analysis. Our attempts to spoof users by modifying the authentication tokens failed. Legitimate credentials cannot be guessed because the service is only using one-time passwords. Additionally, we found no service accounts or other legitimate service credentials within the client-side storage, which we could have used for spoofing the back-end service.

5.2.2 Tampering Attacks

Tampering is a threat prevalent in all of the components. Figure 12 illustrates attacks, which test for tampering-related security preconditions on the components of the micro-mobility service.

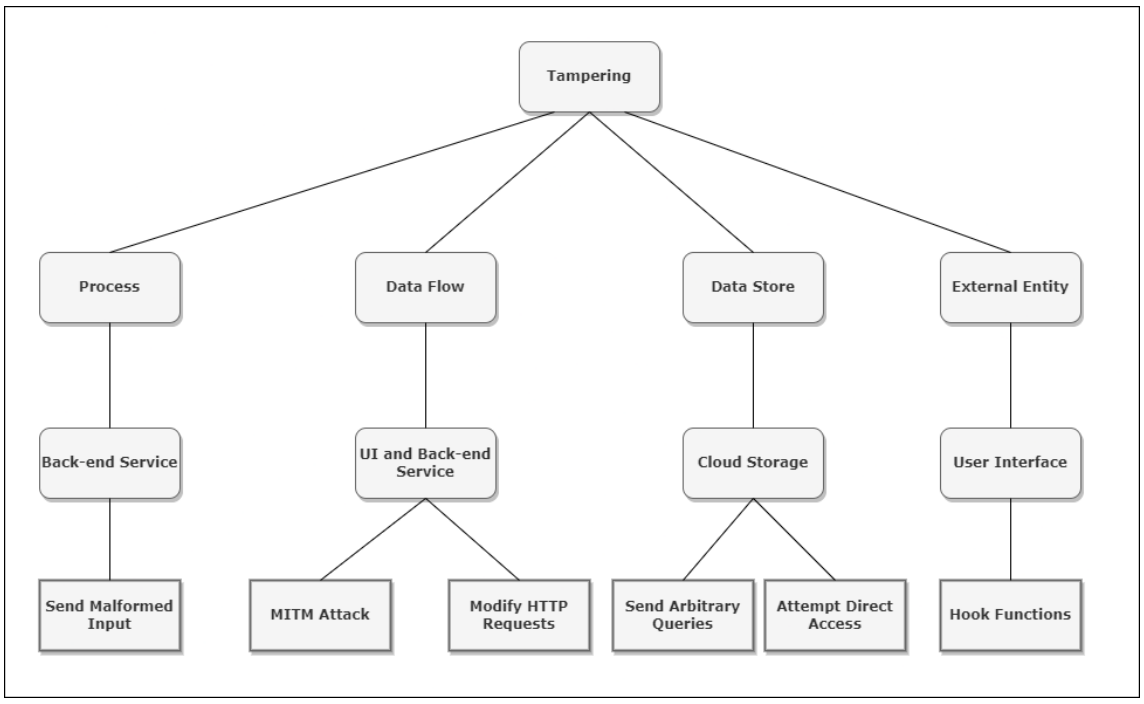

Figure 12. *The tampering threat tree.*

We cannot tamper with the back-end service process directly as we do not have any access to memory nor is such access part of our testing scope. We can attempt to lead the process

to a corrupt state by submitting invalid input and observing if it is verified for correctness. During static analysis, we observed that some application logic was being used to verify input correctness on the client side. We observed client-side input validations for the email field, the first and last name fields, and the discount code input field. The email field was validated via a large regular expression. The first and last names have to be less than 50 characters long. The discount code must be between 2 and 20 characters. The client-side input fields for credit card numbers are set to numeric. We were able to enter arbitrary data into the cardholder's name field, which resulted in a silent error. The erroneous details were not submitted or stored. We could not validate if the client-side restrictions for user data were also enforced on the server side as these requests did not move through our proxy during the man-in-the-middle attack. This was because of the protections described in Section 5.1.6. Due to this, we were unable to observe the structure of the requests and tamper with them. Similarly, our attempts of sending malformed inputs directly by HTTP requests via Postman were refused by the back-end service.

We can tamper with the data flow between the user interface and the back-end service by violating the integrity of the channel. This data flow is depicted as element 1.1 in the data-flow diagram presented in Chapter 4. We captured the request for user data using a man-in-the-middle attack and changed the target user to another test user we owned. No data was returned and we received a permission denied error. It is also possible to capture and tamper with the structured database queries that the service uses to search for the ride history of the user or available scooters in an area. We modified the filters in an attempt to access the Firebase database in an unintended matter, but as described in Section 5.2.4, the requests were refused. Thus, even though we were able to capture and tamper with some of the requests while they were in transit, the back-end service process refused the altered requests and no vulnerabilities were discovered. It is worth mentioning that we constructed the same requests in Postman and modified the content of the requests there, however, tampering with the requests during transit may make the request appear more legitimate as all of the expected request headers are present.

We can tamper with a data store if it has weak protection. Several of our previous tests have already determined the cloud data store component of the service provider to employ expected security controls in the form of Firebase Security Rules. In Section 5.2.4 we attempted to access the database directly via an extra access channel, the web browser. Our attempts to access and tamper with the data store directly failed. Requests with modified parameters sent to the database were denied as we were not able to bypass the data store protection scheme. We cannot test the suggested different failure modes of a data store, such as discard and wraparound scenarios when the data stores become full. While this may allow for tampering in some scenarios, we cannot test this using a black-box approach on a cloud provider's database. Thus we did not include such attacks within the threat tree.

External entities are not generally vulnerable to tampering threats. However, in our system we consider the user interface an external entity. In Section 5.1.6 we used the Objection toolkit to hook and modify certain functions of the user interface in an attempt to disable SSL pinning. This constitutes to tampering with the component, as we have mutated existing code logic. Similarly, attackers could patch certain sections of the user interface and reconstruct a tampered version of the user interface. Such attacks might allow an adversary to bypass client-sided checks, such as field input validation, location checks, and SSL pinning.

5.2.3 Repudiation Attacks

Repudiation is a threat that is hard to integrate into a penetration testing strategy. From the attacker's perspective, there exists no overview of the granularity of the logs for the back-end service component. Figure 13 illustrates an attack, which could lead to a repudiation threat for the user interface component.

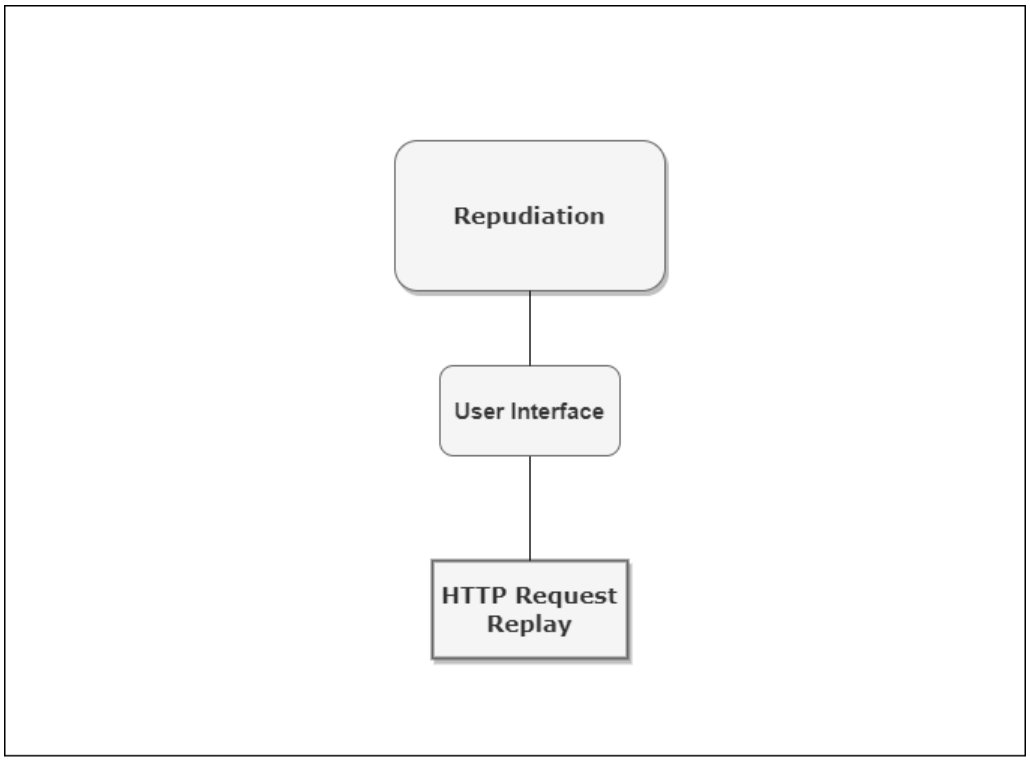

Figure 13. *The repudiation threat tree.*

We discussed in Section 5.2.2, how the authentication token and authorization claims are signed and any messages are tied to the identity within the claims. As the token has a lifetime of one hour there only exists a small time frame in which they can be replayed and possibly repudiated. We were able to replay the requests via a MITM attack for as long as the token remained active. The tokens do not expire once new tokens are issued.

We observed the user interface to log certain events from the client side. If the service provider is keeping little or no logs, they are automatically vulnerable to the repudiation threat as anyone can claim they were not responsible for their actions within the service. If the service is logging weakly authenticated data or the logs themselves can be tampered with directly there exists a repudiation threat. We did not find a way to directly access the log files. For this threat, we worked with the service provider and provided an overview regarding possible repudiation threats. We suggested they evaluate the security preconditions which could lead to repudiation threats for their system. We found no service accounts during the reconnaissance stage and were otherwise unable to spoof our identity to introduce a repudiation threat. The spoofing attacks are described in Section 5.2.1.

5.2.4 Information Disclosure Attacks

According to Table 3, almost all of the elements are vulnerable to information disclosure. We have graphed attacks, which may allow information disclosure. The leaf nodes of the threat tree illustrate these attacks in Figure 14 for our scope.

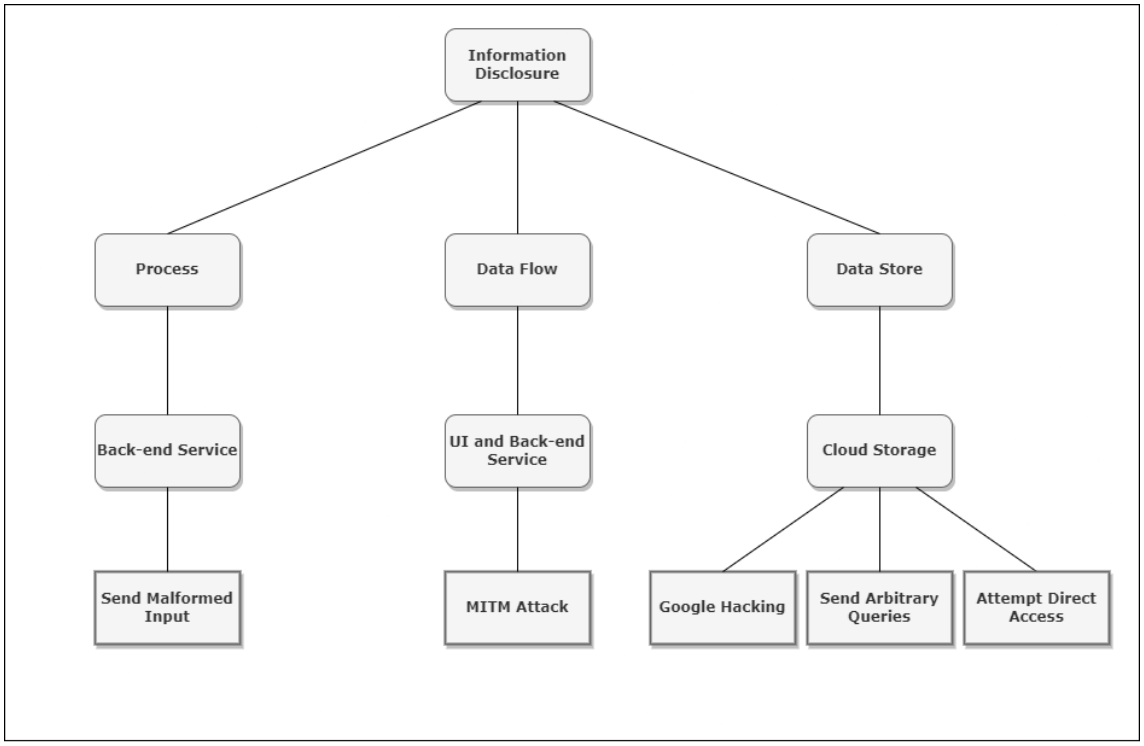

Figure 14. *The information disclosure threat tree.*

In Figure 14, there exists an overlap with the reconnaissance stage as we have already executed the MITM attack in Section 5.1.6. Similarly, sending malformed inputs to the back-end service process to attack input validation was already discussed in Section 5.2.2. Thus this section will describe our attacks against the cloud storage component.

We conducted attacks on the cloud storage component by submitting specially crafted queries to the Google search engine. This technique attempts to discover sensitive data from the domains we discovered during enumeration in a passive approach. Commonly referred to as Google Hacking, this technique can reveal directory listings or files within the back-end service component, which have been indexed by the Google search engine [48, Ch. 3]. These files and directories may contain configurations, internal documents and development versions of the site [26]. We can use the various queries provided by the Google Hacking Database[17].

The total amount of possible combinations in which way an attacker could craft the queries is infeasible to consider. Thus we limited our search queries to common file extensions and directory listing queries observed within online resources, the Google Hacking Database, and a relevant book [48]. The queries we used are presented below.

1. filetype:doc | filetype:pdf | filetype:txt | filetype:ppt | filetype:xls
2. intext:password | intext:"pass word" | intext:pwd | intext:pw
3. inurl:/wp-content/uploads/ | inurl:admin | inurl:backup | inurl:ftp
4. intitle:"index of" | intitle:admin | intitle:backup | intitle:ftp

We used these queries within the Google search engine and constrained them to the five domains discovered during enumeration using the *site* operator. The queries returned some documents regarding investor relations, however, none of them we could consider sensitive. Thus, using the aforementioned queries we were not able to disclose any sensitive data from the back-end service component.

We have discovered several resource identifiers related to the Firebase cloud storage during the reconnaissance stage and can now execute attacks for information disclosure on the component. We will evaluate, whether the service provider's Firebase database is misconfigured and open to the public. Navigating to `https://serviceprovider.firebaseio.com/.json` should display the entire database in JSON format if it were misconfigured. The operation was denied and a permission-related error message was displayed. This means that the service provider's Firebase database is protected via access controls and not available to unauthenticated users. Some parts of the cloud storage are public, such as the geographical area collection, which defines the operating areas of the service. The area database reveals, that there are hidden speed restrictions in certain parts of the areas. Likely due to road conditions, with the intent to avoid user injury and vehicle damage. We can query the Firebase NoSQL database using structured queries. Listing 5.3 illustrates a query to gather all available scooters from the database. The query is sent to

[17]Google Hacking Database - https://www.exploit-db.com/google-hacking-database (15.11.2022)

the database as the body of an HTTP POST request. The database then returns the data in JSON format.

```
{
    "structuredQuery": {
        "from": [
            {
                "collectionId": "scooters"
            }
        ],
        "where": {
            "compositeFilter": {
                "filters": [
                    {
                        "fieldFilter": {
                            "field": {
                                "fieldPath": "isAvailable"
                            },
                            "op": "EQUAL",
                            "value": {
                                "booleanValue": true
                            }
                        }
                    },
                    {
                        "fieldFilter": {
                            "field": {
                                "fieldPath": "area"
                            },
                            "op": "EQUAL",
                            "value": {
                                "stringValue": "zYUgfOe2fkPvabN4t3oq"
                            }
                        }
                    }
                ],
                "op": "AND"
            }
        }
    }
}
```

Listing 5.3. Structured query for gathering available scooters.

This database query was not restricted via access controls and could be executed by anyone. As we are querying the whole database, changing the query parameters might allow an

attacker to disclose sensitive data. Our attempts with several other query parameters returned a permission denied error and were unsuccessful. This means that the service provider is implementing Firebase security rules[18] by conditionally allowing specific queries and denying others. Additionally, we could not use our authentication token to execute arbitrary queries in order to disclose the private data of other users, unavailable vehicles, and other data. While we were not able to test every possible combination of query parameters, we can conclude that basic access controls are implemented and arbitrary data cannot be freely queried.

5.2.5 Denial of Service Attacks

The back-end service and cloud storage components may be vulnerable to the denial of service threat. Denial of service might be caused by consuming application-specific or fundamental resources in large quantities and as a potential side product of failed input validation. For the back-end service process, we attempted entering some malformed inputs into the credit cardholders name field, however, it is hard to evaluate the effect this had on the back-end service. Instead, we focused on our ability to invoke legitimate functionality multiple times to consume resources and observed the restrictions set on us by the service provider. If the service provider is not limiting our ability to consume resources via means such as rate limiting, the service may have a potential denial of service vector. Figure 15 illustrates attacks, which could lead to a denial of service threat.

There exists functionality within the service to make the scooter emit a beep. This functionality is designed, so that a user may find their device within a cluster of devices in a small enough area. This could be the situation in front of a popular shopping mall. The user location is checked on the client side and if the user is near the scooter based on client-provided location data, they are able to click an otherwise disabled button. This button will make the scooter beep. During static analysis, we discovered that the radius for being able to use the beeping functionality is 100 units between the location of the user and the scooter. This functionality is implemented using Google Cloud Functions [40]. As the check is client sided, an adversary can spoof their geographic location using Android apps such as Fake GPS Location[19].

While it is hard to protect against this type of spoofing, the implementation of rate limiting could be considered. Arbitrary invocation of cloud functions increases the consumption of Google Cloud fundamental resources. During our tests, we found no limitations to the number of cloud functions we could invoke. Since the service scales to meet bigger

[18]Firebase Security Rules - https://firebase.google.com/docs/rules (19.11.2022)
[19]Fake GPS Location - https://play.google.com/store/apps/details?id=com.lexa.fakegps (26.11.2022)

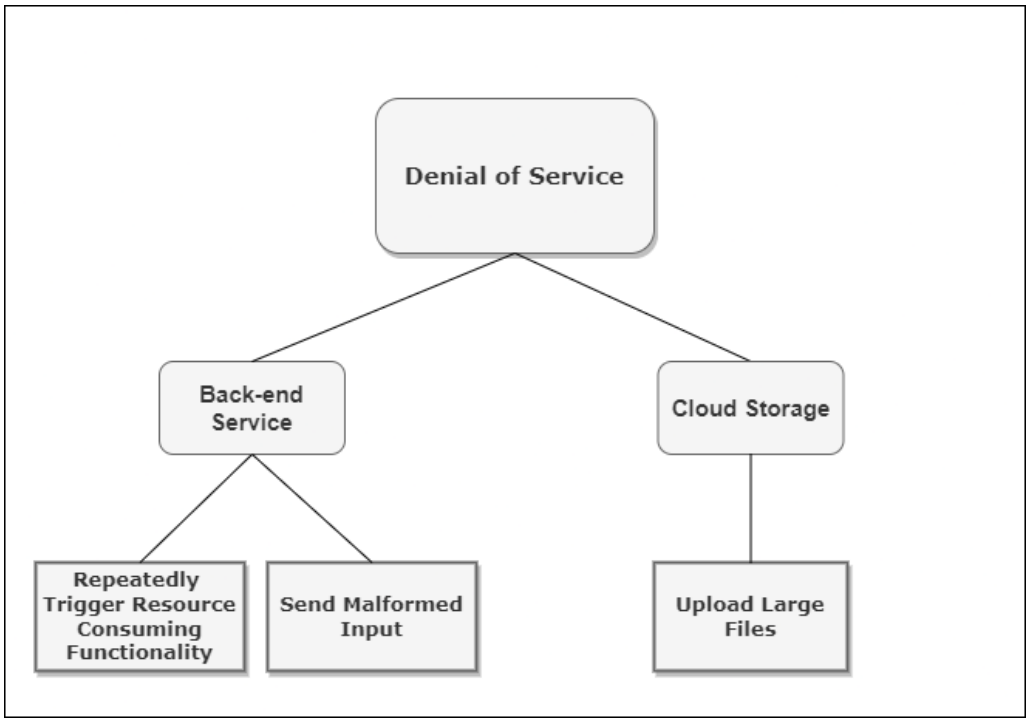

Figure 15. *The denial of service threat tree.*

demand it incurs a bigger bill for the service provider if used maliciously [49]. It is worth mentioning that the functionality could only be invoked when authenticated to the service. However, anyone can still register an account and invoke this functionality without paying for the service. Similarly, we were able to invoke arbitrary amounts of cloud functions on another subdomain of the service provider, which was related to the private scooters described in Section 5.1.2. We reported these findings to the service provider as a potential denial of service vector for the back-end service process.

In Chapter 4, we described several ways to achieve denial of service against a data store. The most relevant of those vectors for our system was resource consumption. Google Cloud scales automatically to meet the demand and this functionality might backfire if attackers seek to intentionally cause such scaling. This type of scaling is usually associated with additional costs to the service provider. During our tests, we attempted to find functionality we could have abused to enter large amounts of data into the database. An attacker could invoke and immediately end large amounts of ride sessions to create entries within the database, however, this will incur a monetary cost to the attacker. Monetary penalty is an effective tactic in such situations as described by a study on service disruption attacks on mobility services [50].

For the cloud storage component, we created an empty JPG file and filled it with junk data up to a size of 100 megabytes. The junk data was acquired from the Unix */dev/urandom* interface. The service provider allowed users to report accidents and other issues via the

application to the operations team. A user could attach up to three pictures together with the report. As the files must be stored within some data storage, if there exist no limitations regarding uploads an attacker may be able to consume resources. We attached a picture file filled with 100 megabytes of junk data and sent a report. No errors or success messages were displayed and we could not confirm the denial of service vector. We provided data regarding the attack to the service provider and recommended they verify that there exists a size limitation on this functionality. We recommended for the service provider consider, whether there exists any mechanism to stop large amounts of data from being uploaded by users and examine the impact this activity could have.

5.2.6 Privilege Escalation Attacks

The back-end service and the on-board computer process may be vulnerable to privilege elevation. As the on-board computer process was out of scope for our tests due to constraints by the service provider, we focused on privilege escalation attacks against the back-end service process. Figure 16 illustrates attacks, which could lead to privilege escalation.

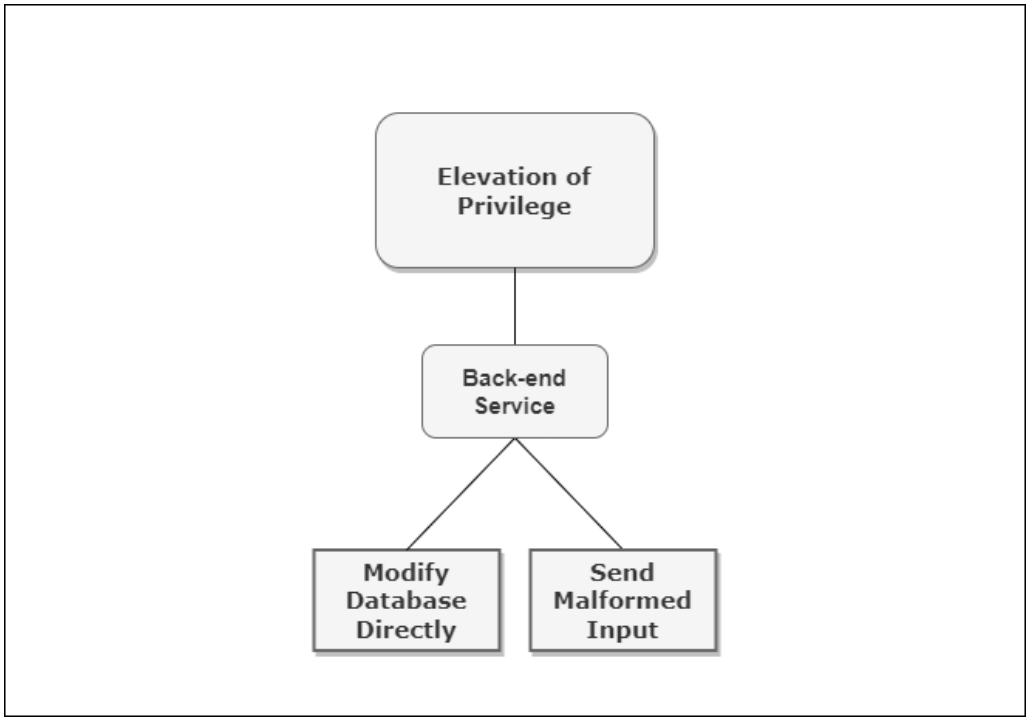

Figure 16. *The privilege escalation threat tree.*

We already discussed input validation in Section 5.2.2 and we did not find a way to elevate our privileges by submitting false data. We discussed tampering with the cloud storage in Section 5.2.4 and Section 5.2.2. We found no way to assign ourselves higher privileges directly. In Section 5.2.1 we discussed the nature of the used authentication token and how it cannot be modified due to a digital signature. However, during DNS enumeration,

we found a subdomain of the service provider used by the service administrators. We attempted to abuse the nature of this portal to elevate our privileges within the service. In our threat trees, this still falls under the direct modification attack.

The admin panel is accessible to all and is protected by a login form. There exists user registration functionality, allowing anyone to create an account and log into the admin portal. Moreover, we discovered that both the administrative application and the service back-end are using the same Firebase database. The users, however, are stored in different database collections. The collection *users* is used for regular users and the collection *platformUsers* is used for the service administrators. The ability to register and log into the admin portal is available for anyone even without email verification. However, there are access controls implemented, so that an account can initially do nothing but log out. New users of the admin portal are presented with an empty user interface. The account must be verified by another service administrator before the functionality of the admin portal can be accessed. While the database is the same, the users of the service and the admin portal are managed separately. We tested this by adding an email to our regular user account and registering under the same email to the admin portal. We were unable to view details related to our regular user using the JWT of our admin user. We were also not able to operate normally using the service administrator token on the live service. We observed the only difference within the tokens to be the *tenant* field as highlighted in Listing 5.4 and 5.5. This indicates, that the main service and the admin portal might exist in different Google tenants[20].

We created a second administrator user and tried to use the authentication token to access the data of our original service administrator user. The attempts failed as we received a permission denied error. However, as an admin profile was now created for us, we tried to use Postman to query the database for all of our service administrator profile details. We noticed that the service administrator account profile included a *role* field, which was set to a null value for us. We assume this role is involved in the authorization process and successful modification of this value could lead to privilege escalation. We attempted to edit our role directly but were unable to do so. We provided this scenario to the service provider and recommended they verify, that the service administrator token and portal could not be abused in the ways described previously.

We conclude, that the functionality of the administrative interface is not open to the public. It remains unclear as to why registration to the admin portal is publicly allowed and without email verification. A potential adversary could employ social engineering techniques and trick an employee into granting administrative privileges to the account of the adversary.

[20]Google Tenants - https://cloud.google.com/talent-solution/job-search/docs/tenants (05.06.2022)

We are unsure of the organizational process behind authorizing users in the admin portal but believe that registration should not be open to the public. The admin accounts registered by the author as part of the penetration test were deleted by the service administrators after three days.


```
{
  "auth_time": 1648300023,
  "user_id": "
     gFFzCtgtyjZ7lpNBt9PTyM4s6X72"
     ,
  "sub": "
     gFFzCtgtyjZ7lpNBt9PTyM4s6X72"
     ,
  "iat": 1648300023,
  "exp": 1648303623,
  "phone_number": "+37255555555",
  "firebase": {
    "identities": {
      "phone": [
        "+37255555555"
      ]
    },
    "sign_in_provider": "phone"
  }
}
```


Listing 5.4. JWT data of regular user.


```
{
  "auth_time": 1648304993,
  "user_id": "
     mT48tRvBF9g58WrXWsYqHaJEjb32"
     ,
  "sub": "
     mT48tRvBF9g58WrXWsYqHaJEjb32"
     ,
  "iat": 1648304993,
  "exp": 1648308593,
  "email": "author@gmail.com",
  "email_verified": false,
  "firebase": {
    "identities": {
      "email": [
        "author@gmail.com"
      ]
    },
    "sign_in_provider": "password"
      ,
    "tenant": "Platform-sdf4af"
  }
}
```


Listing 5.5. JWT data of service administrator.

5.2.7 Application Logic Tests

In addition to all the systematic tests derived from our methodology, we conducted some experiments related to flaws in the logic of the application. We analyzed strategies, where people have led similar micro-mobility services to an erroneous state. For example, we observed some service providers, such as Bird, to not disable the power of the micro-mobility device while it is accelerating[21]. We propose three experiments on the service, which are presented below.

[21] How to hack a Bird scooter - https://www.youtube.com/watch?v=syX_e0sXtnI (26.11.2022)

- Attempting to move and ride a scooter, which we had not rented nor reserved.
- Ending a ride session while the scooter is moving.
- Ending a ride session while the scooter is still but accelerating.

The first experiment highlights the ability of anonymous users to interact with the system. The scooters of the service provider do not have any locking mechanisms to stop them from moving. Instead, the scooter will start to continuously emit a beeping sound. This means anyone can walk up to a scooter and move it without special restrictions. The devices are never really locked, instead, the electric power functionality is only activated during a ride. The scooters do have GPS sensors that allow the service provider to track the devices, however, attackers may employ GPS jamming techniques as discussed in Section 4.4.5.

The second and third experiments are associated with the electric power disabling logic of the micro-mobility devices. How is it decided, when the electric-powered acceleration is revoked from the device? The back-end service process and the on-board computer process must work together to decide. Continuously accelerating with a Bird scooter while at a standstill and stopping the ride via the user interface allowed users to continue to use the scooter after their session was officially ended. We tested these scenarios on our service provider and discovered that we were not able to replicate a similar fault. The scooters of the service provider lose power when a ride session is ended regardless of the state of the scooter within the physical world. The second and third experiments produced no unexpected results and we were not able to keep using the scooter after the sessions ended.

5.3 Reporting and Results

This section provides an overview of our findings and the disclosure process with the service provider. We used a modified reporting template by OWASP [26] to report our findings to the service provider. The template first described a disclaimer regarding the test, as we may have not identified all of the security vulnerabilities. We then provided an executive summary for the management and a summary of interesting findings which we present below.

1. We were able to invoke an unrestricted amount of cloud functions within parts of the service, which may represent a denial of service vector. The service allows an authenticated user to potentially consume large amounts of fundamental resources, which may incur a monetary cost on the service provider (Section 5.2.5).
2. We found the user interface to declare unused Android permissions. The user interface declares permissions RECORD_AUDIO and WRITE_EXTERNAL_STORAGE, which we observed to be unused. As they are unused, there is no reason to

declare them in the Android manifest and potentially grant the permissions to the user interface (Section 5.1.4).

3. There exists a public and redundant Javascript file on the website of the service provider. While the file did not disclose any sensitive data, the file appears to have no use. Potentially, the same folder might contain other unnecessary or sensitive files, which we were not able to find. We recommended the service provider evaluate the contents of the discovered directory and remove any redundant files (Section 5.1.2).
4. As a general observation, we discovered that the service allows concurrent authentications from different devices. The service does not inform the users when a sign-in from a new device is detected. While not directly a vulnerability, we recommended the service provider consider implementing this feature (Section 5.2.1).

We provided the report to the service provider in greater detail than we describe in the thesis. We were asked not to reveal details regarding the service components and our findings as part of the testing agreement. We presented the modeling, reconnaissance, and exploitation stage results with technical details to the service provider. We recommended the service provider verify our findings and investigate the potential impact of our elicited threats on the system components, which were not tested.

6. Discussion

This chapter discusses several caveats of the approach and possible directions for further research. Firstly, as we propose no systematic way to derive the system components, the construction of a correct data flow diagram is adamant. This is further enhanced by the fact that we are looking at a black-box perspective. A data flow diagram, which does not correspond to reality will guide the penetration test in the wrong direction. In future research, this problem could be addressed by integrating an existing and systematic approach for deriving system components into the methodology or proposing a novel approach. Additionally, the general mapping of DFD elements to STRIDE threats provided by Howard and Lipner [5] might not apply to all systems. Correctly assigning the mappings has an impact on the penetration test.

Secondly, the ability of the tester to choose reliable attacks, which test the STRIDE security preconditions of the components plays a key role during the penetration test. In the same way, this applies to choosing information-gathering techniques during the reconnaissance stage. We used existing guides to choose the attacks, however, the choices we made can be challenged by others and with possibly valid reasons. Similarly, the approach to executing the attacks can be challenged. For example, the choices made for choosing word lists for enumeration attacks or search queries for Google Hacking. This is a fundamental problem of black-box penetration testing. A future direction to address this problem could be the integration of an attack taxonomy into the methodology.

We do not assign any criticality to the findings as it is challenging without being the system owner. As a selective black-box study, we cannot assess the impact our findings would have if abused. Additionally, our testing scope was constrained and thus any evaluations for determining the riskiest components of the system would be partial. In future research, prioritizing the findings could be considered in a more systematic way. For example, the CVSS score could be calculated, similarly to the quantitative threat modeling method discussed in [22].

The overall sequence of stages for our methodology is similar to the PatrIoT methodology [17]. However, we are applying STRIDE per each element, which they did not do. This results in a more complete list of possible threats. Furthermore, we utilize threat tree patterns created by Howard and Lipner [5] to help choose the attacks. The PatrIoT methodology [17] uses an agile methodology to improve the level of detail for the data

flow diagram as the information gathering stage progresses. We do not reiterate the data flow diagram based on findings from the reconnaissance stage in this research. While it would probably help, a detail level greater than the contextual version will require greater effort. The results from reiterating the data flow diagram after reconnaissance may prove superior as the resulting threat tree could be more complete, however, it would require further research.

Shostack [29] and Howard and Lipner [5] discuss that such threat trees provide only the most common threats but the total set of possible attacks is hard to estimate. The same is true for our approach as there is no guarantee the tester will be able to use generalized threat trees to derive a complete set of attacks on preconditions, which may lead to STRIDE threats. Shostack [29] ends each threat tree with a node called "Other", which illustrates all other possible attacks. The nature of the "Other" node is not described and it is used as a way to complete each threat tree, as such a leaf node encompasses everything. We did not directly address this but allowed the tester to add more attacks for completeness.

The validation carried out in this study is subjective in that it is based on the author's experience from several points. The scope of the actual penetration test was constrained compared to the entire micro-mobility service. This results in validation being weaker as the threats to cyber-physical components were only analyzed theoretically. It can be inherently assumed that an attacker may be seeking to abuse all of the possible security preconditions brought out in Appendix 2, which we were not able to do in our research. Future research might consider this approach in a white-box penetration test as it solves the problem of correct component derivation and allows for an improved level of detail. An improved level of detail returns in a more complete threat tree. While the methodology helped guide our penetration test and produced some actual results, we cannot claim it is superior to existing guides as no comparisons have been made. We can claim it was more systematic compared to the ones analyzed in the study. The effectiveness of the methodology on other cyber-physical systems is subject to further research.

Before starting the research, we did an experiment where the author attempted to derive a penetration testing strategy for a micro-mobility system based on no structured methodology or modeling. The strategy contained ten attacks and information-gathering techniques related to the Android application and back-end service, which were derived from the author's previous experience in testing such services. The author had little experience with physical attacks and thus the physical components were not considered at all in the initial plan. While most of those attacks were also elicited during our threat modeling, we were able to derive a total of 29 combinations of threat-component pairs via systematic STRIDE-per-element threat modeling. The combinations are presented in Figure 3 of

Chapter 4. This granted us a more detailed overview of all the attacks we should be executing. However, the knowledge we had of the system was not much different in both cases. Employing the methodology increased our understanding of the system and increased the number of relevant attacks we could execute based on susceptibility to STRIDE threats. This contributed to a more systematic penetration test and a better threat overview for the service provider. The proposed methodology provides a way of leveraging STRIDE-based threat modeling to develop a penetration testing strategy for an existing micro-mobility service.

7. Summary

This research aimed to investigate the components of modern micro-mobility services and cyber security threats they may be vulnerable to. Additionally, we investigated the applicability of STRIDE-based threat modeling for developing a penetration testing methodology for a micro-mobility service. We have proposed a four-stage methodology based on the STRIDE threat model and we validate the methodology by conducting a penetration test on a real micro-mobility service.

The stages of the methodology include (i) threat modeling, (ii) reconnaissance, (iii) exploitation, and (iv) reporting. During the threat modeling stage, we divided the service into abstract components. Based on the components we graphed a contextual data-flow diagram of the micro-mobility service. We elicited threats on each of the service components on a STRIDE-per-element basis. After threat elicitation, we used existing threat trees by Howard and Lipner [5, Ch. 22] to investigate security preconditions, which could lead to STRIDE threats. Based on the preconditions, we constructed threat trees containing relevant attacks to guide the penetration testing process. During the reconnaissance stage, we followed a threat tree we created to gather information regarding the service architecture and technologies. During the exploitation stage, we followed threat trees for each of the STRIDE mnemonics to attack the relevant security preconditions for the service components. We reported our findings to the service provider using the reporting template from OWASP [26].

The key findings of the research can be summarized as four main points. Firstly, we have analyzed a real micro-mobility service provider and proposed a contextual data-flow diagram of the service components in Section 4.3 of Chapter 4. Secondly, we have derived cyber security threats related to micro-mobility services based on the STRIDE threat model in Section 4.4 of Chapter 4. Furthermore, we have constructed threat trees, which could be used during a penetration test of a micro-mobility service provider in Chapter 5. Based on the validation, we have found that STRIDE-based threat modeling can be applied to existing micro-mobility services to help guide the penetration testing process. We provide actual findings in Section 5.3 of Chapter 5.

These findings can help current and upcoming micro-mobility service providers understand the cyber security threats relevant to their systems. Additionally, the methodology may be used to conduct security testing on micro-mobility services. This is of particular

importance because micro-mobility services include several different and interconnected components, making them hard to test in practice. Cyber security is a primary concern for such systems due to the potential impact on human life.

Our research, however, was limited in several aspects, which had a potential impact on the results. The validation was carried out in a constrained scope at the request of the service provider and we were unable to test some of the components. GPS-related attacks were not conducted due to resource constraints. The validation was carried out only on one service provider introducing a potential result bias. As we conducted a black-box test, the results may be dependent on the skills of the penetration tester.

Further research may decide to improve the methodology by addressing some of the caveats described in Chapter 6, such as deriving correct components for an existing and black-box system. This is important because the results of the methodology are mainly dependent on the threat modeling stages. Furthermore, future research may investigate the applicability of the methodology with a white-box approach. White-box perspective solves the issue of correct component derivation, making the resulting threat trees more complete and reliable. Another direction of future research may be to apply the methodology to other cyber-physical systems, while also assigning criticality to the findings.

References

[1] Bolt. *Cities: Ride Around the World*. [Accessed: 11-12-2022]. 2022. URL: https://bolt.eu/en-ee/cities/.

[2] Oliver O'Brien. *European update: 360,000 e-scooters available across the continent*. [Accessed: 10-11-2022]. June 2021. URL: https://zagdaily.com/trends/european-update-360000-e-scooters-available-across-the-continent/.

[3] Ran Idan. *Xiaomi Scooter Hack Enables Dangerous Accelerations and Stops*. [Accessed: 10-12-2022]. Feb. 2019. URL: https://www.zimperium.com/blog/dont-give-me-a-brake-xiaomi-scooter-hack-enables-dangerous-accelerations-and-stops-for-unsuspecting-riders/.

[4] ISO Central Secretary. *Information technology — Security techniques — Information security management systems — Requirements*. en. Standard ISO/IEC 27001:2013. Geneva, CH: International Organization for Standardization, 2013. URL: https://www.iso.org/standard/54534.html.

[5] Michael Howard and Steve Lipner. *The Security Development Lifecycle*. Redmond, WA, USA: Microsoft Press, 2006.

[6] Phu X. Mai et al. "Modeling Security and Privacy Requirements: a Use Case-Driven Approach". In: *Information and Software Technology* 100 (2018), pp. 165–182. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2018.04.007. URL: https://www.sciencedirect.com/science/article/pii/S0950584918300703.

[7] Arfive Gandhi, Yudho Giri Sucahyo, and Yova Ruldeviyani. "Investigating the Protection of Customers' Personal Data in the Ridesharing Applications: A Desk Research in Indonesia". In: *2018 15th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. 2018, pp. 118–121. DOI: 10.1109/ECTICon.2018.8619912.

[8] Rafiullah Khan et al. "STRIDE-based threat modeling for cyber-physical systems". In: *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. 2017, pp. 1–6. DOI: 10.1109/ISGTEurope.2017.8260283.

[9] Karen Li, Awais Rashid, and Anne Roudaut. “Vision: Security-Usability Threat Modeling for Industrial Control Systems”. In: *European Symposium on Usable Security 2021*. EuroUSEC ’21. Karlsruhe, Germany: Association for Computing Machinery, 2021, pp. 83–88. ISBN: 9781450384230. DOI: 10.1145/3481357.3481527. URL: https://doi.org/10.1145/3481357.3481527.

[10] Michael Felderer et al. “Chapter One - Security Testing: A Survey”. In: ed. by Atif Memon. Vol. 101. Advances in Computers. Elsevier, 2016, pp. 1–51. DOI: https://doi.org/10.1016/bs.adcom.2015.11.003. URL: https://www.sciencedirect.com/science/article/pii/S0065245815000649.

[11] Victoria Hazael. *200 years since the father of the bicycle Baron Karl von Drais invented the ’running machine’*. [Accessed: 14-03-2022]. Feb. 2017. URL: https://www.cyclinguk.org/cycle/draisienne-1817-2017-200-years-cycling-innovation-design.

[12] Institute for Transportation and Development Policy. *The Electric Assist: Leveraging E-bikes and E-scooters for More Livable Cities*. [Accessed: 14-03-2022]. 2019. URL: https://www.itdp.org/wp-content/uploads/2019/12/ITDP_The-Electric-Assist_-Leveraging-E-bikes-and-E-scooters-for-More-Livable-Cities.pdf.

[13] Razor. *Corporate History*. [Accessed: 14-03-2022]. URL: https://global.razor.com/au/about-razor/corporate-history/.

[14] Tom Huddleston. *Lime, Bird, Spin: Why scooter start-ups are suddenly worth billions*. [Accessed: 13-03-2022]. URL: https://www.cnbc.com/2018/07/11/lime-bird-spin-why-scooter-start-ups-are-suddenly-worth-billions.html.

[15] Loren Kohnfelder and Praerit Garg. *The threats to our products*. [Accessed: 01-12-2022]. 1999. URL: https://shostack.org/files/microsoft/The-Threats-To-Our-Products.docx.

[16] Shaymaa Mamdouh Khalil et al. “Threat Modeling of Cyber-Physical Systems - A Case Study of a Microgrid System”. In: *Computers & Security* (2022), p. 102950. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2022.102950. URL: https://www.sciencedirect.com/science/article/pii/S016740482200342X.

[17] Emre Süren et al. “PatrIoT: practical and agile threat research for IoT”. In: *International Journal of Information Security* (Nov. 2022). DOI: 10.1007/s10207-022-00633-3.

[18] Nisha Vinayaga-Sureshkanth et al. “Security and Privacy Challenges in Upcoming Intelligent Urban Micromobility Transportation Systems”. In: *Proceedings of the Second ACM Workshop on Automotive and Aerial Vehicle Security*. AutoSec ’20. New Orleans, LA, USA: Association for Computing Machinery, 2020, pp. 31–35. ISBN: 9781450371131. DOI: 10.1145/3375706.3380559. URL: https://doi.org/10.1145/3375706.3380559.

[19] William Enck et al. “A Study of Android Application Security”. In: SEC’11. San Francisco, CA: USENIX Association, 2011, p. 21.

[20] Qingchuan Zhao et al. “Geo-locating Drivers: A Study of Sensitive Data Leakage in Ride-Hailing Services”. In: *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. ISBN: 1-891562-55-X. URL: https://www.ndss-symposium.org/ndss-paper/geo-locating-drivers-a-study-of-sensitive-data-leakage-in-ride-hailing-services/.

[21] Mohmmad Haseeb Haider et al. “Threat Modeling of Wireless Attacks on Advanced Metering Infrastructure”. In: *2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)*. 2019, pp. 1–6. DOI: 10.1109/MACS48846.2019.9024779.

[22] Nataliya Shevchenko et al. *Threat modeling: a summary of available methods*. Tech. rep. Carnegie Mellon University Software Engineering Institute, 2018.

[23] Karen A. Scarfone et al. *SP 800-115. Technical Guide to Information Security Testing and Assessment*. Tech. rep. Gaithersburg, MD, USA, 2008.

[24] Pete Herzog. “Open-source security testing methodology manual”. In: *Institute for Security and Open Methodologies (ISECOM)* (2010).

[25] Kevin Orrey. *The penetration testing framework*. [Accessed: 05-02-2022]. 2014. URL: http://www.vulnerabilityassessment.co.uk/Penetration%20Test.html.

[26] The OWASP Foundation. *OWASP Web Security Testing Guide*. [Accessed: 20-02-2022]. URL: https://owasp.org/www-project-web-security-testing-guide/stable/.

[27] The OWASP Foundation. *OWASP Mobile Security Testing Guide*. [Accessed: 20-02-2022]. URL: https://owasp.org/www-project-mobile-security-testing-guide/.

[28] The OWASP Foundation. *OWASP Internet of Things*. [Accessed: 29-09-2022]. URL: https://owasp.org/www-project-internet-of-things.

[29] Adam Shostack. *Threat Modeling: Designing for Security*. 1st. Wiley Publishing, 2014. ISBN: 1118809998.

[30] Lucidchart. *What is a Data Flow Diagram*. [Accessed: 29-11-2022]. URL: https://www.lucidchart.com/pages/data-flow-diagram.

[31] Patrick Engebretson. *The basics of hacking and penetration testing: ethical hacking and penetration testing made easy*. Elsevier, 2013.

[32] Ali Jafarnia-Jahromi et al. "GPS vulnerability to spoofing threats and a review of antispoofing techniques". In: *International Journal of Navigation and Observation* (2012).

[33] Malte Lenhart, Marco Spanghero, and Panagiotis Papadimitratos. "Relay/Replay Attacks on GNSS Signals". In: *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '21. Association for Computing Machinery, 2021, pp. 380–382. ISBN: 9781450383493. DOI: 10.1145/3448300.3468256. URL: https://doi.org/10.1145/3448300.3468256.

[34] Statcounter. *Mobile Operating System Market Share Worldwide*. [Accessed: 11-12-2022]. 2022. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[35] Aaron Guzman and Aditya Gupta. *IoT Penetration Testing Cookbook: Identify Vulnerabilities and Secure Your Smart Devices*. Packt Publishing, 2017. ISBN: 1787280578.

[36] Meta. *Integration with an Android Fragment - React Native*. [Accessed: 18-03-2022]. 2022. URL: https://reactnative.dev/docs/integration-with-android-fragment.

[37] Meta. *Hermes - Javascript engine optimized for React Native*. [Accessed: 11-12-2022]. 2022. URL: https://hermesengine.dev/.

[38] Google. *Firebase Documentation*. [Accessed: 21-03-2022]. 2022. URL: https://firebase.google.com/docs.

[39] Google. *Firebase JavaScript API reference*. [Accessed: 13-11-2022]. 2022. URL: https://firebase.google.com/docs/reference/js/v8/firebase/.

[40] Google. *Cloud Functions*. [Accessed: 12-11-2022]. URL: https://cloud.google.com/functions.

[41] Google. *Permissions on Android*. [Accessed: 06-03-2022]. URL: https://developer.android.com/guide/topics/permissions/overview.

[42] OWASP. *Top 10 Mobile Risks - Final List 2016*. [Accessed: 14-11-2022]. URL: https://owasp.org/www-project-mobile-top-10/.

[43] Google. *Save key-value data*. [Accessed: 14-11-2022]. URL: https://developer.android.com/training/data-storage/shared-preferences.

[44] Aaron Guzman and Aditya Gupta. *IoT Penetration Testing Cookbook: Identify Vulnerabilities and Secure Your Smart Devices*. Packt Publishing, 2017. ISBN: 1787280578.

[45] Google. *Firebase App Check*. [Accessed: 14-11-2022]. URL: https://firebase.google.com/docs/app-check.

[46] Google. *Authenticate with Firebase on Android using a Phone Number*. [Accessed: 24-03-2022]. 2022. URL: https://firebase.google.com/docs/auth/android/phone-auth.

[47] Michael Jones, John Bradley, and Nat Sakimura. *Json web token (jwt)*. rfc 7519. RFC Editor, 2015. URL: https://www.rfc-editor.org/rfc/rfc7519.

[48] Bill Gardner, Johnny Long, and Justin Brown. *Google Hacking for Penetration Testers, Third Edition 3rd Edition*. Dec. 2015. ISBN: ISBN-10: 0128029641 ISBN-13: 978-0128029640.

[49] Google. *Firebase Security Checklist*. [Accessed: 15-11-2022]. URL: https://firebase.google.com/support/guides/security-checklist.

[50] Jérôme Thai, Chenyang Yuan, and Alexandre M. Bayen. "Resiliency of Mobility-as-a-Service Systems to Denial-of-Service Attacks". In: *IEEE Transactions on Control of Network Systems* 5.1 (2018), pp. 370–382. DOI: 10.1109/TCNS.2016.2612828.

Appendices

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Siim-Alexander Kütt

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Stride-Based Black Box Penetration Testing Methodology for Micro-Mobility Services", supervised by Shaymaa Mamdouh Khalil
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

04.01.2023

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Microsoft Threat Tree Patterns

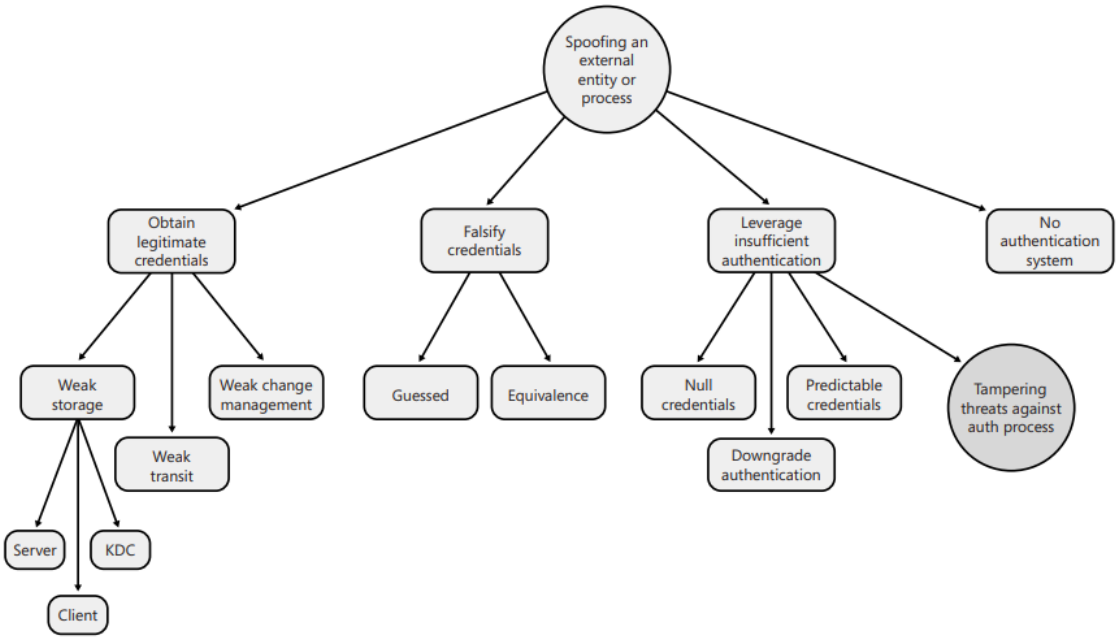

Figure 17. *Spoofing an External Entity or Process [5, Ch. 22].*

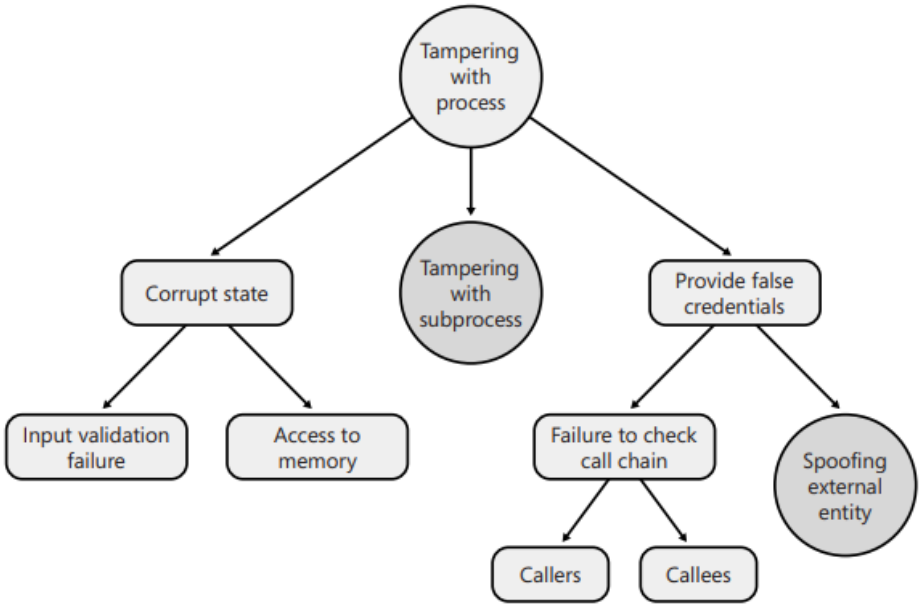

Figure 18. *Tampering with Process [5, Ch. 22].*

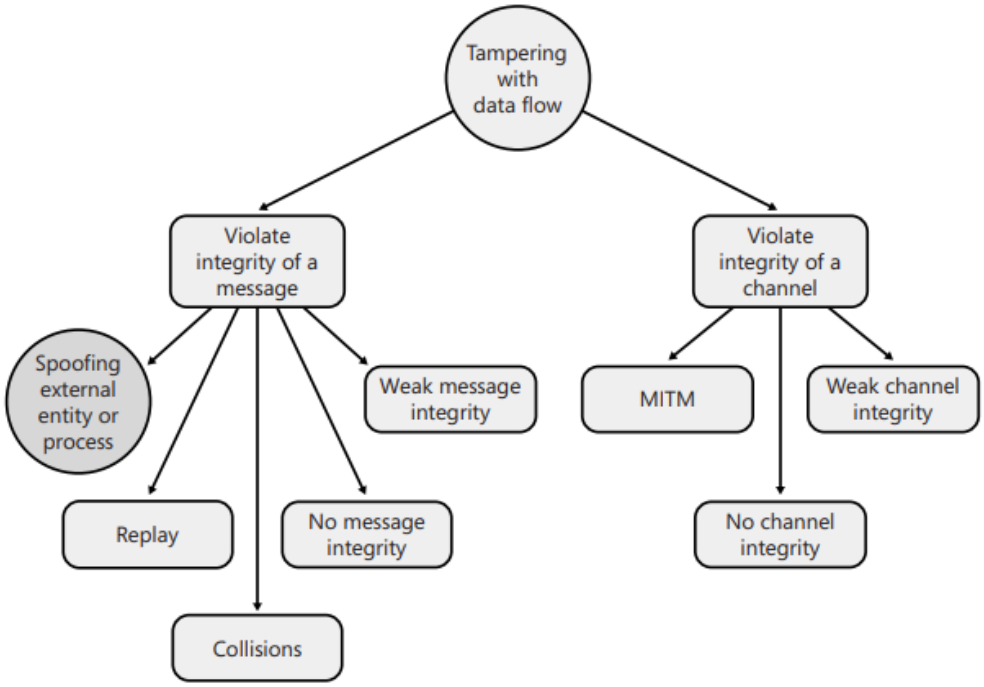

Figure 19. *Tampering with Data Flow [5, Ch. 22].*

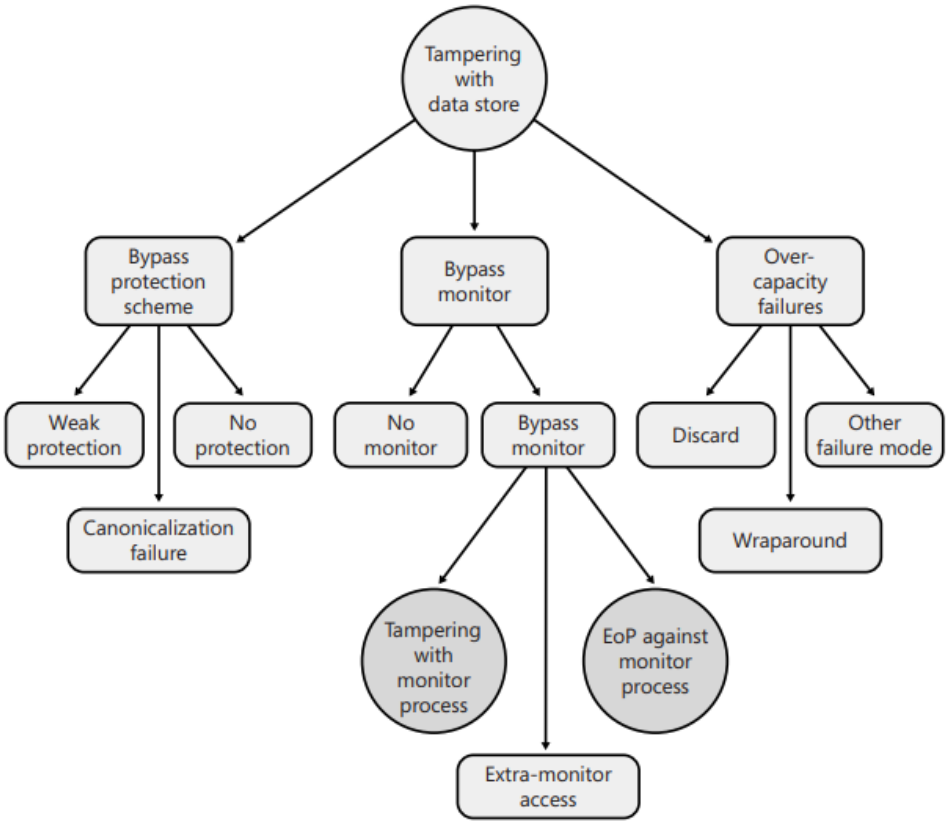

Figure 20. *Tampering with Data Store [5, Ch. 22].*

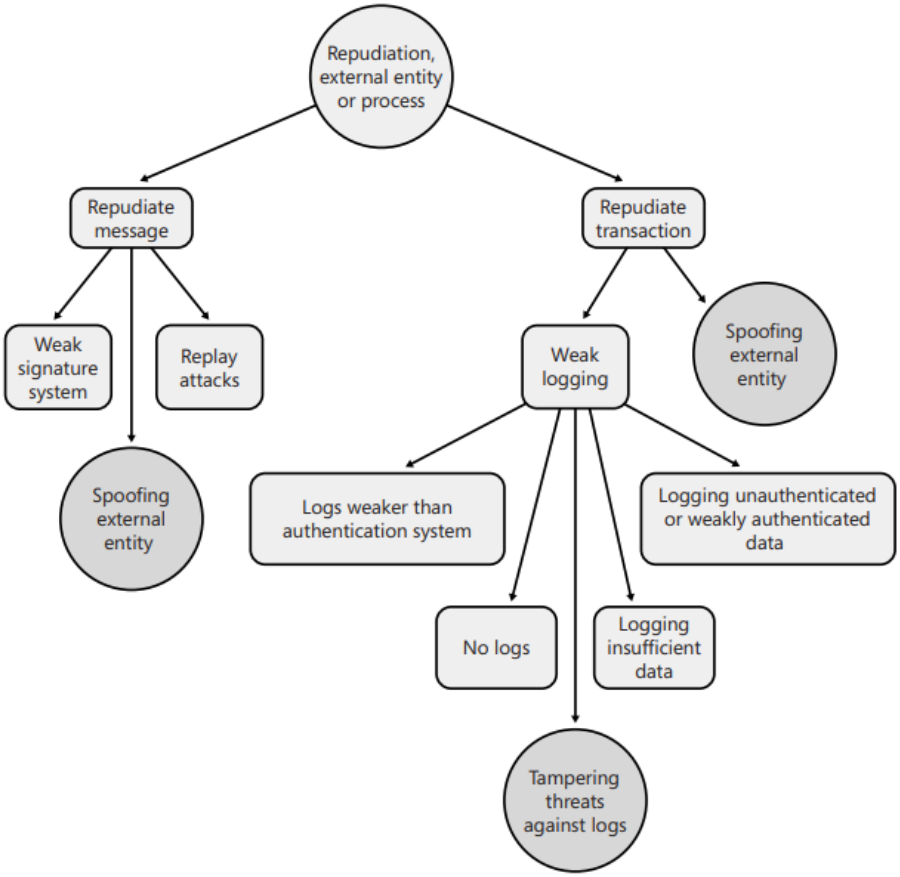

Figure 21. *Repudiation for External Entity or Process [5, Ch. 22].*

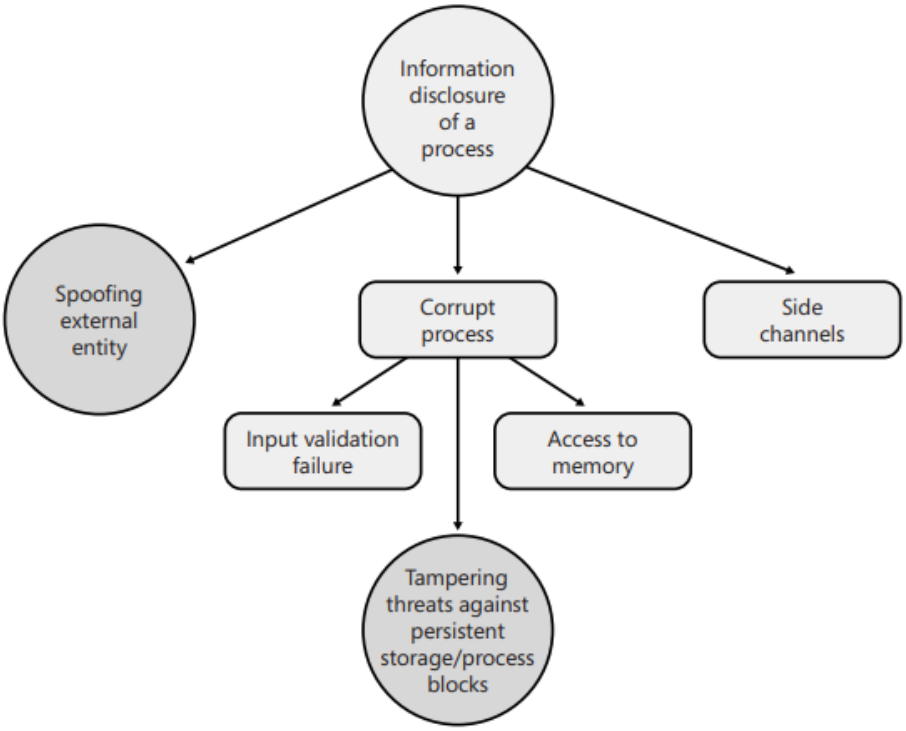

Figure 22. *Information Disclosure for Process [5, Ch. 22].*

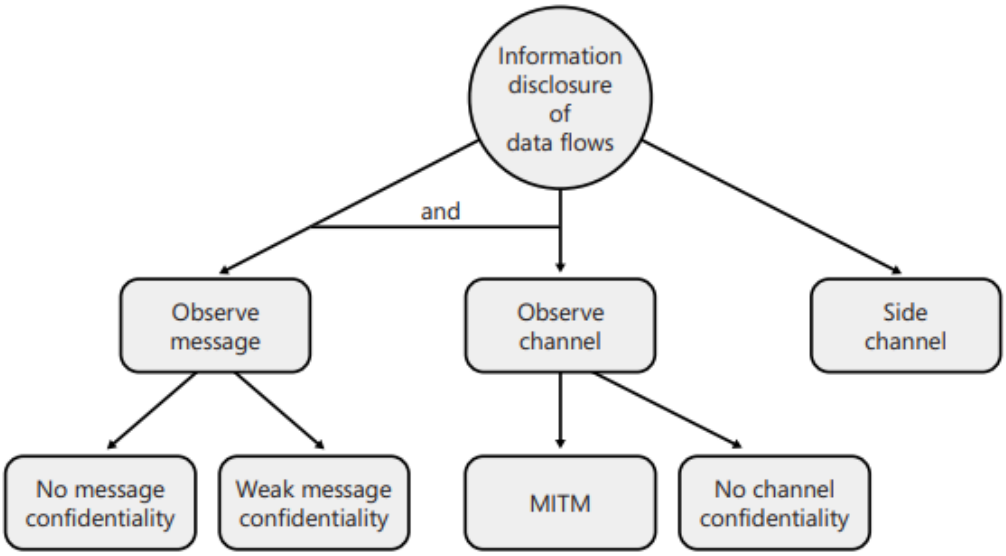

Figure 23. *Information Disclosure for Data Flow [5, Ch. 22].*

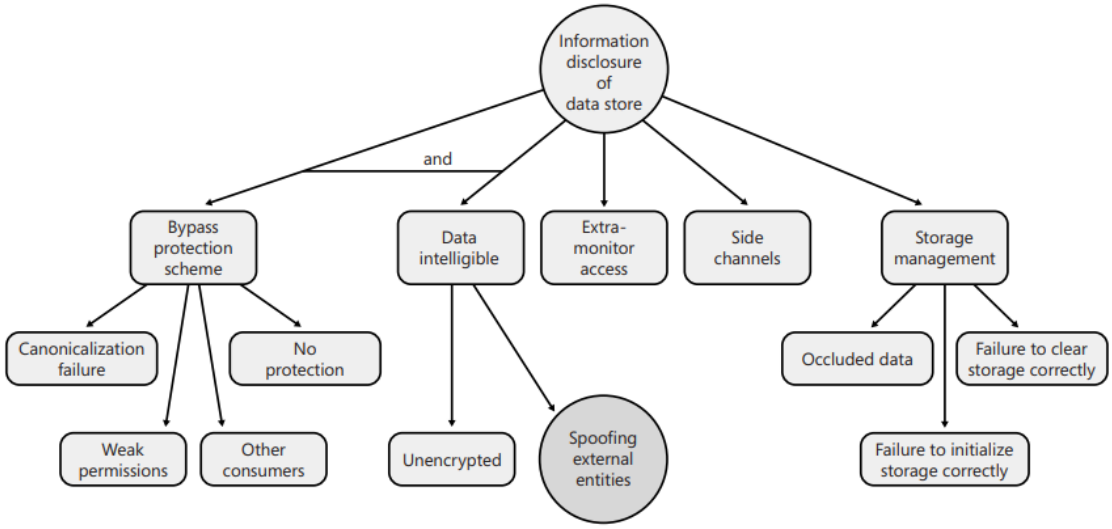

Figure 24. *Information Disclosure for Data Store [5, Ch. 22].*

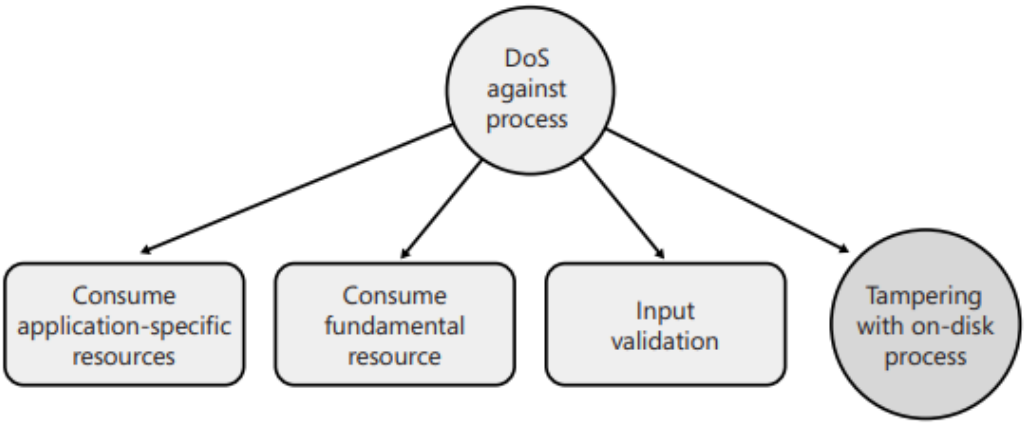

Figure 25. *Denial of Service for Process [5, Ch. 22].*

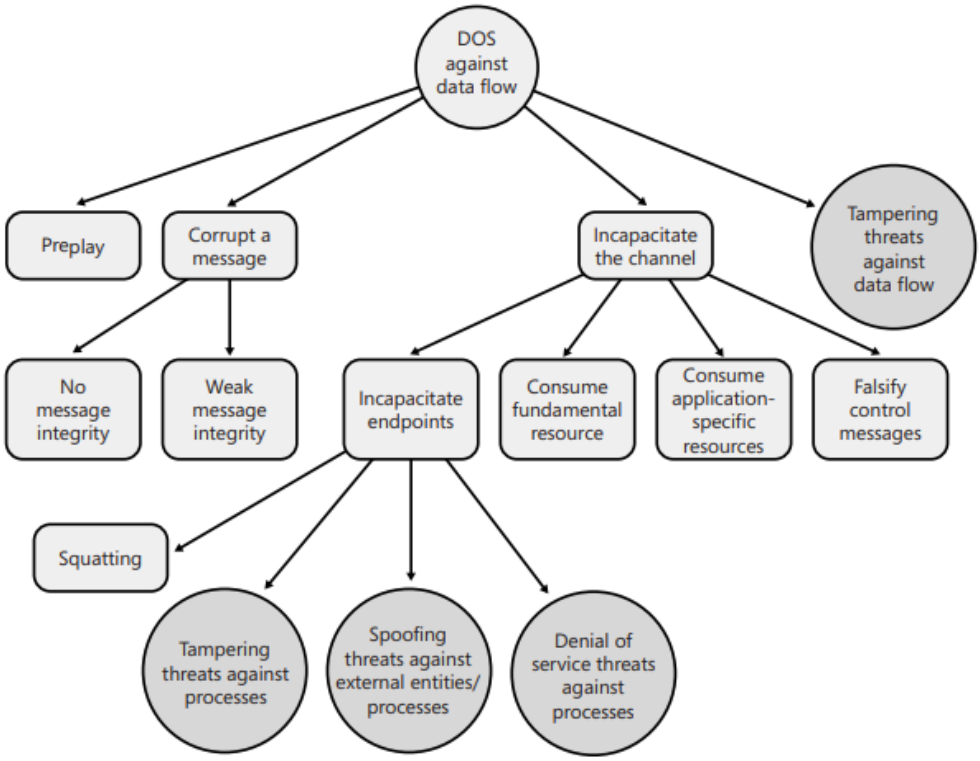

Figure 26. *Denial of Service for Data Flow [5, Ch. 22].*

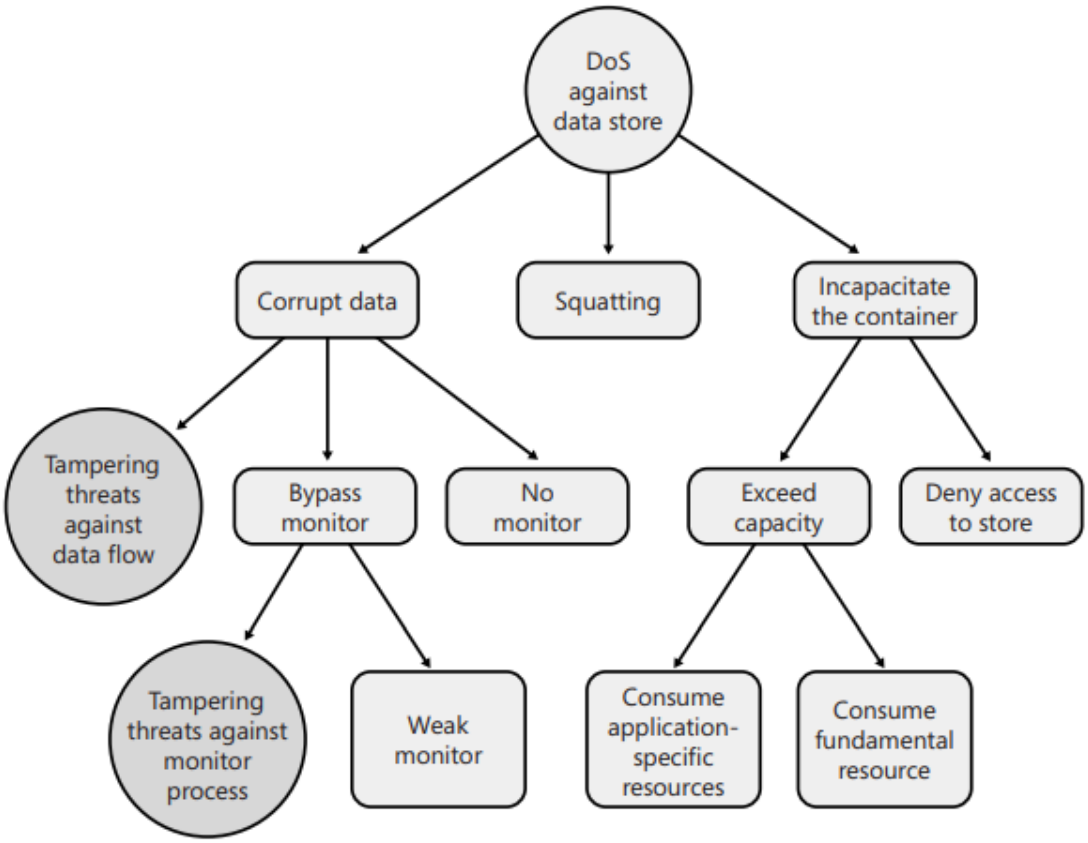

Figure 27. *Denial of Service for Data Store [5, Ch. 22].*

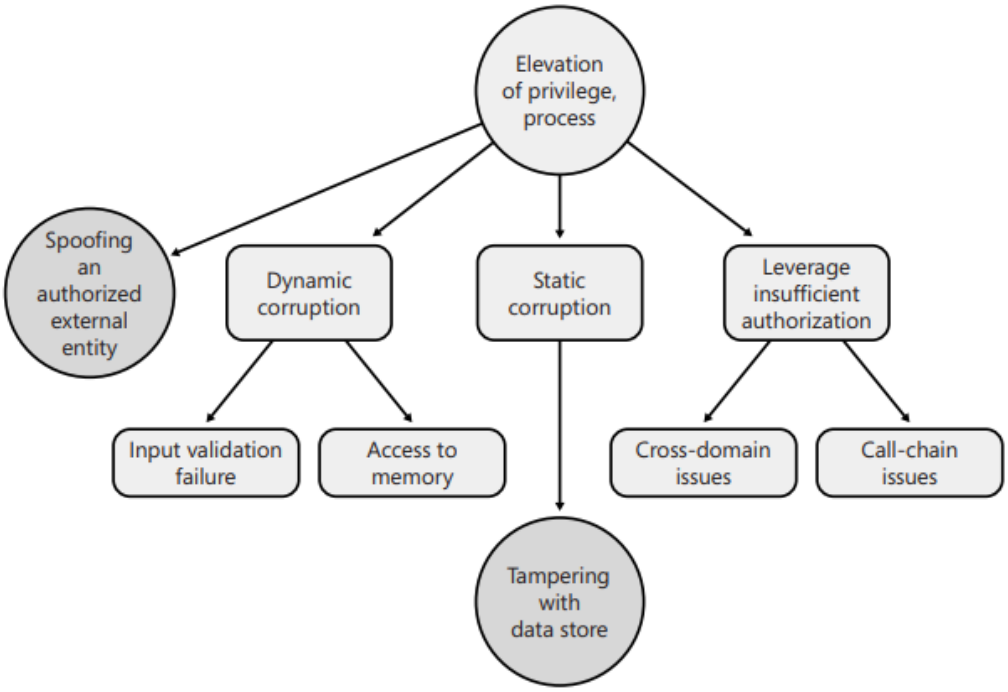

Figure 28. *Escalation of Privilege for Process [5, Ch. 22].*