

**UiO** : **Department of Informatics**  
University of Oslo

# Threat analysis of mobile banking platforms

Henrik Andre Stene  
Master's Thesis Autumn 2015





# Threat analysis of mobile banking platforms

Henrik Stene - [henriast@ifi.uio.no](mailto:henriast@ifi.uio.no)

August 3, 2015



# Abstract

Online banking solutions have existed for two decades already, and the industry now has a relatively good understanding of security threats and risks against traditional online banking. Mobile banking solutions are more recent, so that the industry has relatively less experience with analysing threats and risks. This Master's thesis focuses on analysing threats against mobile banking. More specifically, in this Master's project we have applied well known threat modelling methodologies. Our goal has been to compare mobile banking applications with the more traditional online banking service. We studied two threat modelling methodologies in order to determine which threat modelling method is best suited to do threat analysis in general and which is best suited for analysing threats against mobile banking in particular. Finally, a threat analysis of mobile banking was undertaken. The main conclusion is that mobile banking has less exposure to threats than traditional mobile banking based in desktop computers.



# Contents

<b>I</b>	<b>Introduction</b>	<b>1</b>
<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Motivation . . . . .	3
1.2	Goal . . . . .	4
1.3	Approach and Research Method . . . . .	4
1.4	Work Done . . . . .	5
1.5	Results . . . . .	5
1.6	Conclusion . . . . .	6
1.7	Outline . . . . .	6
<b>2</b>	<b>Introduction to Threat Modelling</b>	<b>9</b>
2.1	Threat Modelling . . . . .	9
2.2	Threat, Risk and Mitigation . . . . .	10
2.3	The Four-Step Framework . . . . .	10
2.3.1	What are you building? . . . . .	11
2.3.2	What can go wrong with it once it's built? . . . . .	11
2.3.3	What should you do about those things that can go wrong? . . . . .	11
2.3.4	Did you do a decent job of analysis? . . . . .	12
2.4	Discussion . . . . .	12
2.5	Summary . . . . .	13
<b>II</b>	<b>Background</b>	<b>15</b>
<b>3</b>	<b>Methods for threat modeling</b>	<b>17</b>
3.1	STRIDE . . . . .	17
3.1.1	Violation of Authentication . . . . .	19
3.1.2	Violation of Integrity . . . . .	19
3.1.3	Violation of Non-repudiation . . . . .	20
3.1.4	Violation of Confidentiality . . . . .	20
3.1.5	Violation of Availability . . . . .	21
3.1.6	Violation of Authorization . . . . .	21

3.1.7	Using STRIDE on Data Flow Diagrams . . . . .	21
3.1.8	STRIDE-per-element . . . . .	22
3.1.9	STRIDE-per-interaction . . . . .	23
3.1.10	Summary . . . . .	23
3.2	Attack Libraries . . . . .	23
3.2.1	Level Of Detail in Attack Libraries . . . . .	24
3.2.2	Summary . . . . .	25
3.3	Attack Trees . . . . .	25
3.3.1	Attack Tree Components . . . . .	26
3.3.2	Creating new Attack Trees . . . . .	27
3.3.3	Summary . . . . .	28
3.4	Discussion . . . . .	28
3.5	Summary . . . . .	29
<b>4</b>	<b>Mobile Operating Systems</b>	<b>31</b>
4.1	Introduction . . . . .	31
4.2	Security Features . . . . .	31
4.2.1	Secure Boot . . . . .	32
4.2.2	Encryption and Data Access . . . . .	32
4.2.3	Application Security . . . . .	33
4.2.4	Device Security . . . . .	33
4.3	Android . . . . .	33
4.3.1	Secure Boot . . . . .	34
4.3.2	Encryption and Data Access . . . . .	34
4.3.3	Application Security . . . . .	35
4.3.4	Device Security . . . . .	36
4.4	Windows Phone . . . . .	37
4.4.1	Secure Boot . . . . .	37
4.4.2	Encryption and Data Access . . . . .	37
4.4.3	Application Security . . . . .	38
4.4.4	Device Security . . . . .	38
4.5	iOS . . . . .	38
4.5.1	Secure Boot . . . . .	38
4.5.2	Encryption and Data Access . . . . .	39
4.5.3	Application Security . . . . .	39
4.5.4	Device Security . . . . .	40
4.6	Discussion . . . . .	40
4.7	Summary . . . . .	44
<b>5</b>	<b>BankID</b>	<b>47</b>
5.1	Introduction . . . . .	47
5.2	Technology . . . . .	47
5.2.1	PKI . . . . .	48
5.3	Infrastructure . . . . .	48



5.3.1	Felles Operasjonell Infrastruktur . . . . .	49
5.3.2	Bank RA and OTP . . . . .	49
5.3.3	BankID Server . . . . .	50
5.3.4	BankID Client . . . . .	50
5.4	Certificates . . . . .	51
5.4.1	Certificate Format and Security . . . . .	52
5.5	Encryption . . . . .	52
5.6	Process . . . . .	52
5.6.1	Bank Stored BankID . . . . .	53
5.6.2	BankID on Mobile . . . . .	53
5.7	Discussion . . . . .	53
5.8	Summary . . . . .	54
<b>6</b>	<b>Mobile Banking Software</b>	<b>57</b>
6.1	Banks . . . . .	57
6.2	Technology . . . . .	59
6.2.1	Native Applications vs Web Applications . . . . .	59
6.2.2	Mobile Banking Software Technology . . . . .	59
6.2.3	Authentication Service . . . . .	60
6.3	Discussion . . . . .	60
6.4	Summary . . . . .	61
<b>III</b>	<b>Threat Model</b>	<b>63</b>
<b>7</b>	<b>Threat Modelling using STRIDE</b>	<b>65</b>
7.1	Data Flow Diagrams . . . . .	65
7.1.1	Platform . . . . .	65
7.1.2	BankID . . . . .	66
7.1.3	Application . . . . .	66
7.2	Platforms . . . . .	66
7.2.1	STRIDE-per-element . . . . .	66
7.3	BankID . . . . .	69
7.3.1	STRIDE-per-element . . . . .	70
7.4	Application . . . . .	73
7.4.1	STRIDE-per-element . . . . .	73
7.5	Discussion . . . . .	74
7.6	Summary . . . . .	75
<b>8</b>	<b>Threat Modelling using CAPEC</b>	<b>79</b>
8.1	Attack Patterns . . . . .	79
8.2	Scoring . . . . .	80
8.3	Current Threats . . . . .	80
8.3.1	JSON Hijacking (aka JavaScript Hijacking) . . . . .	80

8.3.2	Probe Application Screenshots . . . . .	81
8.3.3	Probe Application Error Reporting . . . . .	81
8.3.4	Probe Application Queries . . . . .	82
8.3.5	Malware-Directed Internal Reconnaissance . . . . .	82
8.3.6	OS Fingerprinting . . . . .	82
8.3.7	Application Fingerprinting . . . . .	83
8.3.8	Social Information Gathering via Research . . . . .	83
8.3.9	HTTP DoS . . . . .	84
8.3.10	Checksum Spoofing . . . . .	84
8.3.11	Intent Spoof . . . . .	85
8.3.12	Principal Spoof . . . . .	85
8.3.13	Signature Spoof . . . . .	86
8.3.14	Pharming . . . . .	86
8.3.15	Phishing . . . . .	87
8.3.16	Clickjacking . . . . .	87
8.3.17	Manipulating User State . . . . .	88
8.3.18	Inducing Account Lockout . . . . .	88
8.3.19	Authentication Abuse . . . . .	89
8.3.20	Create Malicious Client . . . . .	89
8.3.21	Man-in-the-Middle Attack . . . . .	90
8.3.22	Abuse of Transaction Data Structure . . . . .	90
8.3.23	Contaminate Resource . . . . .	91
8.3.24	Infrastructure Manipulation . . . . .	91
8.3.25	Protocol Reverse Engineering . . . . .	92
8.3.26	Lifting Sensitive Data from the Client . . . . .	92
8.3.27	Reverse Engineer an Executable to Expose Assumed Hidden Functionality or Content . . . . .	93
8.3.28	Physical Theft . . . . .	93
8.3.29	Bypassing Electronic Locks and Access Controls . . . . .	94
8.3.30	Bypassing Physical Locks . . . . .	94
8.3.31	Physical Destruction of Device or Component . . . . .	94
8.3.32	Malicious Software Download . . . . .	95
8.3.33	Malicious Software Update . . . . .	95
8.3.34	Target Influence via Social Engineering . . . . .	95
8.4	Discussion . . . . .	96
8.5	Summary . . . . .	97

## **IV Conclusion 99**

### **9 Conclusion 101**

9.1	Discussion . . . . .	101
9.1.1	Platforms . . . . .	101
9.1.2	Threat Modelling Methods . . . . .	102

9.1.3	Current Threats . . . . .	103
9.2	Conclusion . . . . .	104
9.3	Contributions . . . . .	104
<b>Appendices</b>		<b>105</b>
<b>A</b>	<b>CAPEC list</b>	<b>107</b>
A.1	Gathering Information . . . . .	107
A.1.1	Excavation . . . . .	107
A.1.2	Interception . . . . .	108
A.1.3	Footprinting . . . . .	109
A.1.4	Fingerprinting . . . . .	109
A.1.5	Social Information Gathering Attacks . . . . .	110
A.2	Deplete Resources . . . . .	111
A.2.1	Excessive Allocation . . . . .	111
A.2.2	Resource Leak Exposure . . . . .	112
A.2.3	Sustained Client Engagement . . . . .	113
A.2.4	Amplification . . . . .	113
A.3	Injection . . . . .	113
A.3.1	Parameter Injection . . . . .	113
A.3.2	Code Inclusion . . . . .	114
A.3.3	Resource Injection . . . . .	114
A.3.4	Code Injection . . . . .	114
A.3.5	Command Injection . . . . .	115
A.4	Deceptive Interactions . . . . .	116
A.4.1	Path Traversal . . . . .	116
A.4.2	Content Spoofing . . . . .	117
A.4.3	Identity Spoofing . . . . .	117
A.4.4	Resource Location Spoofing . . . . .	118
A.4.5	Action Spoofing . . . . .	118
A.5	Manipulate Timing and State . . . . .	119
A.6	Abuse of Functionality . . . . .	119
A.6.1	Functionality Misuse . . . . .	121
A.6.2	( . . . . .	121
A.7	Probabilistic Techniques . . . . .	122
A.7.1	Brute Force . . . . .	122
A.8	Exploitation of Authentication . . . . .	122
A.9	Privilege Escalation . . . . .	123
A.10	Exploitation of Authorization . . . . .	124
A.10.1	Privilege Abuse . . . . .	124
A.10.2	Exploiting Trust in Client (aka Make the Client In- visible) . . . . .	125
A.11	Manipulate Data Structures . . . . .	126
A.11.1	Buffer Manipulation . . . . .	126

A.12 Manipulate Resources . . . . .	127
A.12.1 File Manipulation . . . . .	127
A.12.2 Configuration/Environment manipulation . . . . .	128
A.12.3 Audit Log Manipulation . . . . .	129
A.12.4 Schema Poisoning . . . . .	129
A.12.5 Protocol Manipulation . . . . .	129
A.12.6 Web Services Protocol Manipulation . . . . .	130
A.13 Analyze Target . . . . .	130
A.13.1 Reverse Engineering . . . . .	130
A.13.2 Software Reverse Engineering . . . . .	130
A.13.3 Cryptanalysis . . . . .	131
A.14 Gain Physical Access . . . . .	131
A.14.1 Bypassing Physical Security . . . . .	131
A.15 Malicious Code Execution . . . . .	131
A.16 Alter System Components . . . . .	131
A.16.1 Software Integrity Attacks . . . . .	132
A.16.2 Hacking Hardware Devices or Components . . . . .	132
A.16.3 Malicious Logic Inserted Into to Product . . . . .	132
A.17 Manipulate System Users . . . . .	133

# List of Figures

2.1	The four questions used in the four-step framework . . . . .	11
2.2	The four-step framework . . . . .	12
3.1	List of the 16 categories found in CAPEC[11] . . . . .	25
3.2	A theoretical attack tree using OR-nodes . . . . .	26
3.3	A theoretical attack tree using AND-nodes . . . . .	27
3.4	A simple attack tree on a customer database . . . . .	27
4.1	An example of the <i>Chain-of-trust</i> mechanism (dm-verity) in Android.[5] . . . . .	34
5.1	Inter bank agreement . . . . .	48
5.2	Functionality provided by FOI . . . . .	49
5.3	Certificate issuing tree . . . . .	51
5.4	BankID Bank Stored Data Flow Diagram . . . . .	55
5.5	BankID on Mobile Data Flow Diagram . . . . .	56
7.1	Platform data flow diagram . . . . .	76
7.2	BankID data flow diagram . . . . .	77



# List of Tables

3.1	The STRIDE Threats, with examples by Adam Shostack.[34]	18
3.2	STRIDE-per-Element[38]	22
4.1	Distribution of Android devices that accessed the Google Play Store.[4]	41
4.2	A visual comparison between systems	44
6.1	Distributions of banks market share per 2014	57
6.2	Mobile banking software functionality for each major bank.	58
6.3	Mobile banking software solution per bank	60
7.1	STRIDE-per-element table over mobile operating systems	67
7.2	STRIDE-per-element table over BankID	70
7.3	STRIDE-per-element table over mobile banking applications	74

**Part I**  
**Introduction**





# Chapter 1

## Introduction

The motivation behind this project is to see how well the banking industry has made the transition from traditional online banking using desktop computer systems and technology, to more modern computer systems like smartphones and tablets. These devices use newer and simpler operating systems than used in traditional computers and it is therefore important to analyze and investigate if there are security defects that might put customers or financial institutions at risk. We consider the three major mobile operating systems, BankID and six different mobile banking applications when creating this threat model.

### 1.1 Motivation

Online banking has existed in Norway since 1996, when *Sparebanken Hedmark*[27] released the first online banking service. Since then the online banking industry has grown into more than 3,7 million users nationwide.[15] This makes online banking a very important service with regard to availability at all times. Online banking is shifting towards the use of mobile devices, together with the rest of the computer industry. *Finance Norway* states in an article that mobile banking is becoming more and more popular, and that there are more than 1,9 million users accessing their bank accounts through mobile banking services.

The seemingly rapid growth of mobile banking services is concerning *The Financial Supervisory Authority of Norway*.[30] This statement makes it important to take a closer look at the security of mobile banking services.

Another important aspect is that we keep more and more information located in these mobile devices, and mobile devices has become the primary tool for accessing the Internet. We keep personal information, biometric data, passwords, credit card information, private emails, and we access bank accounts using these devices. This means that if these

devices were to be controlled or accessed by an unauthorized person, that person could do anything from accessing online user accounts, to access bank accounts or steal credit card information stored on the device.

It is the combination of rapid growth in usage of mobile devices, and that we store so much private and sensitive information on these devices that is the foundation of this thesis.

## 1.2 Goal

The goal of this project is to assess the security assurance of mobile banking software, and to identify the most relevant threats to its software and infrastructure. Our research questions are as follows:

1. What is the level of security of mobile operating systems compared to desktop operating systems?
2. What is the best fit between threat analysis methodology and the type of system to be analyzed?
3. Which threat analysis method is best to use while analyzing mobile banking security?
4. What are currently the most relevant threats against mobile banking in the Norwegian market?

## 1.3 Approach and Research Method

In this project we use existing threat analysis methods and frameworks in order to analyze the security and build a threat model. First we go through the most common threat analysis methods, and then decide upon which method or methods we want to use to analyze mobile banking systems. We answer our research questions by performing four different operations.

1. Analyze the security of the mobile platform, mobile authentication service and mobile banking application, and see how it compares to the security of the online banking service on desktop operating systems.
2. Analyze threat modelling methods in order to find the best fit between threat analysis methodology and the type of system to be analyzed.

3. Use the results from the analysis of threat modelling methods to decide which threat modelling method is best suited for analyzing mobile banking security.
4. Apply one or more threat modelling methodologies to identify the most relevant threats against the mobile banking infrastructure in the Norwegian market.

This is done by performing a comparative study between three different threat analysis methods. We compare differences between them, and take a look at the tradeoffs. Then we do a pilot case, where we apply two different threat analysis methods to the case of mobile banking, and create a threat model using the results from each method. At the end of the thesis, we apply the same comparative methods used to compare threat analysis methodologies, to compare the results from each threat model and we present relevant threats.

## 1.4 Work Done

We have done research on the three major mobile operating systems using available documentation and other data that is public information. We have also done similar research on BankID, which is the primary authentication service used by both online banking and mobile banking services. We have used this information, in addition to gathering public information about six major mobile banking platforms, to create a threat model using two different threat modelling methods.

## 1.5 Results

Platform security on mobile devices are ahead of desktop operating systems, much due to the fact that mobile systems are restricted and that they were initially designed with focus on different important security features.

Looking at the results of using STRIDE, it is clear that it does not produce or discover any concrete threats, unless the threat modeller is experienced and already knows about them. The result of using STRIDE did not provide any attacks or threats that could be used as a foundation for further threat modelling. Therefore, we had to start from scratch using CAPEC.

CAPEC does provide a thorough threat model due to the fact that the attack library is very extensive. The result from using CAPEC as the threat modelling tool shows 34 different threats that public documentation does not account for. This does not prove that all of these

threats are relevant, but it does prove that either the documentation is flawed, or that these attack patterns have not been mitigated.

There are currently some relevant threats against mobile banking applications in Norway. Most attack patterns from CAPEC seems to be mitigated, but some are still possible threats.

## 1.6 Conclusion

Mobile banking application software is in no way less secure than traditional online banking services using desktop computers. However, there are pros and cons with both services. The risk of theft is much higher with smaller mobile devices that end users brings around wherever they go, than with the larger desktop computer that usually resides at the end users home. Mobile operating systems provides much stricter security than their older siblings running on desktop computers. Many mobile devices are strongly protected against malware or other malicious code, due to strict data management systems and application controls provided by the different operating systems. Mobile banking services does also utilize the same user authentication services as traditional online banking. This gives us no reason to believe a priori that mobile banking services are any less secure than traditional online banking.

The different threat analysis methodologies provides different features that can be utilized by the threat modeller. An experienced threat modeller might be able to utilize both abstract methodologies like STRIDE or attack trees, and more detailed tools like an attack library. A beginner will most likely not find many relevant threats using abstract methodologies, but utilizing CAPEC should produce many relevant threats even when used by a novice. While creating a threat model of mobile banking software, we found that an attack library (CAPEC) did produce more relevant threats. Since mobile banking applications handles sensitive data, it is recommended to use an attack library to reach an acceptable level of completeness for the threat model.

## 1.7 Outline

The thesis is divided into chapters and at the end there is an appendix. The first six chapters provides more background we should know for this thesis, and in the last three chapters we create two different threat models using two different threat analysis methods and discuss our findings.

**Chapter 1: Introduction** - Gives a brief introduction of what we have done.

**Chapter 2: Introduction to Threat Modelling** - Gives an introduction to threat modelling, and the steps that we perform in this thesis.

**Chapter 3: Methods for threat modeling** - This chapter gives an introduction to some of the most popular threat modelling methods, and the methods used in this thesis.

**Chapter 4: Mobile Operating Systems** - In this chapter, we take a brief look at the three major mobile operating systems, and the most important security features implemented by these operating systems.

**Chapter 5: BankID** - This chapter gives an introduction to BankID, which is the primary authentication service used by financial institutions in Norway. We also take a look at how BankID authenticates entities and the technology used to make sure that BankID can provide trusted communication between two parties without a current relationship.

**Chapter 6: Mobile Banking Software** - This chapter gives a brief introduction to mobile banking applications and the technologies used by the 6 biggest banks in Norway.

**Chapter 7: Threat Modelling using STRIDE** - In this chapter we try to create a threat model using the threat modelling tool called STRIDE.

**Chapter 8: Threat Modelling using CAPEC** - In this chapter we try to create a threat model using the threat modelling tool called CAPEC.

**Chapter 9: Conclusion** - This Chapter is the conclusion of the project.



## Chapter 2

# Introduction to Threat Modelling

This chapter describes threat modelling and defines important terminology that is needed to perform threat modelling accurately. We briefly discuss risk analysis and risk assessment to clarify what is and what is not included in a threat model, and we discuss the four-step framework for threat modelling.

### 2.1 Threat Modelling

Threat modelling is a really important aspect of software engineering. It is a convenient way of finding security bugs and security errors in a program or a system and, if applied correctly, it will help you to design and implement programs and systems that are more secure from attacks. Every major computer program or computer system needs a threat model in order to make sure that it meets the specified security requirements.

It is a natural thing for humans to create threat models. Subconsciously we threat model every time we make a decision and therefore one might say that we already do know how to threat model. When we leave the house in the morning we address the threat of getting all of our possessions stolen and most of us decides to lock the door before leaving. This is a basic threat model on our belongings and our house. Some consider the threat of things being stolen might so big in this neighborhood that they decide to lock the door even while being at home. Since the door now is locked all the time, we might consider the risk of this attack being executed as mitigated.

In information security, threat modelling is a systematic approach used to discover threats and understand the security requirements of a computer program or computer system. By threat modelling, the devel-



opers may understand how different threats could be realized and then how to avoid these kinds of attacks.

## 2.2 Threat, Risk and Mitigation

In order to introduce the reader to threat modelling, we also have to define what a specified *threat* is. A *threat* is a: *potential cause of an unwanted incident, which may result in harm to a system or organization*. [18]. To emphasize, an attack is only a threat if the cause of that attack may result in harm on the system. From the example above we may say that someone *stealing our possessions* is a threat that leads to the unwanted incident of no longer having our possessions. Using this definition of the word threat makes it possible for us to define the word *mitigation* which is stopping the possibility of a threat being executed. We carefully lock the door to the house before leaving and this represent a mitigation technique against this threat.

The last definition we need to know about is of the word *risk*. We will not be doing a risk analysis in this thesis, but it is important to understand that this would be the next step. Risk is the *effect of uncertainty on objectives* [19] and the level of risk is measured as a component between the consequence of a threat being executed and the likelihood of the threat being executed.

Threat modelling will help us mitigate threats, removing the possibility of an attack being executed on the system, by lowering the likelihood of the threat being executed.

## 2.3 The Four-Step Framework

In the recent book *Threat Modelling - Designing for security* [22], the author *Adam Shostack* introduces an idea that threat modellers should look at threat modelling as four steps that each accomplishes subgoals and that by completing all four steps should make sure that a system is adequately safe. This is of course with the assumption that the threat modeller has done his job carefully.

The four-step framework consists of four questions that the threat modellers and developers need to ask themselves, and these questions can be seen in Figure 2.1 Shostack does suggest that by answering all these questions, step by step, the threat modelling team should be able to create a solid threat model of their system

1. What are you building?
2. What can go wrong with it once it's built?
3. What should you do about those things that can go wrong?
4. Did you do a decent job of analysis?

Figure 2.1: The four questions used in the four-step framework [22]

### 2.3.1 What are you building?

All software development project consists of different types of documents and specifications. Some will be heavily modelled, some will mostly consist of a written description of system requirements and customer wishes. In the first step in the four-step framework, the threat modeller should get to know what they are building. One of the easiest ways of getting an overview is by creating data flow diagrams, or other visual models of the system.<sup>1</sup> By looking at such diagrams, the threat modelling team should be able to get a grasp of how extensive the system really is.

### 2.3.2 What can go wrong with it once it's built?

Step two in the four-step framework is the creation of the threat model. By looking at different models or diagrams of the system, the threat modeller or threat modelling team should be able to find one or more possible attack patterns that may be threats against the system. For this step, it is recommended to utilize one or more threat modelling methods to find threats.

### 2.3.3 What should you do about those things that can go wrong?

After the threat modellers have found every possible threat they could think of, then it is time for step three. Step three consists of deciding what to do with every threat, and how to possibly mitigate the different threats. Many software development projects might have limited resources, time or money, and it is in this step that the threat modellers need to decide which attacks to mitigate, and which attacks are so obscure, hard to execute, or just not that damaging to the system if

---

<sup>1</sup>More about data flow diagrams later

executed, that they can be ignored. This part of the four-step framework is where we need to address threats.

### 2.3.4 Did you do a decent job of analysis?

After all possible threats have been removed or been considered as not damaging, it is time to re-evaluate the system design and implementation. Threat modelling is considered to be an iterative process, and if the validation of the system fails, then the threat modelling process needs to jump back to step one or step two.



Figure 2.2: The four-step framework

## 2.4 Discussion

Threat modelling is a much needed exercise to perform while developing software. This is however something that often is being neglected by not assigning enough resources to be able to really dive deep into a system to discover threats. There is however no doubt that threat modelling is important, especially in applications using sensitive information like mobile banking services.

In his book, *Adam Shostack* postulates a four-step framework for threat modelling. This framework should help most threat modellers to make solid threat models of their system.

In this thesis, we look at three different systems needed to use banking services on a mobile platform. Since we are analyzing these systems from an outsider's point-of-view, we will not be able to perform step three (*Address Threats*) or step four (*Validate*). We will therefore focus on step one, by analyzing the different systems, and step two, by finding threats using different threat modelling methods.

## 2.5 Summary

Threat modelling is an important part of software development. It can be a great tool to make sure that the system is safe to use for the end user, and that the system is not susceptible to common attacks. Threats are described as *the cause of* an unwanted incident, that can result in harm to a system or organization.

The four-step framework is a great way to divide the complete task of threat modelling into four smaller tasks. The four tasks are *model system*, *find threats*, *address threats*, and *validate*. By making four smaller steps, it makes the iterative process of threat modelling much easier as one could always go one or more steps back in the process.



**Part II**

**Background**



## Chapter 3

# Methods for threat modeling

In this chapter we discuss different approaches to threat modeling. We discuss the use of Microsofts STRIDE method and how to execute the task of creating a good threat model using this mnemonic. We also take a look at two alternative ways of modeling threats. One using attack trees to visualize and display threats, and one using attack libraries to be able to control check that it covers all well known threats. We will also compare the three different methods and try to define strengths and weaknesses.

### 3.1 STRIDE

STRIDE is an approach to threat modeling that was introduced in 1999 by Loren Kohnfelder and Praerit Garg.[22] It is a framework that is developed to help information security personnel model their systems and discover threats. Information security personnel may include software developers or any other person that is involved in the development process of software and computer systems.

The mnemonic STRIDE stands for Spoofing, Tampering (of data), Repudiation, Information Disclosure, Denial of Service and Elevation of privilege. All of these words account for a whole group, or category, of threats. This categorization of threats makes it easier to discover threats against certain aspects of a computer system.

To understand STRIDE as an approach to discovering threats we need to see what each letter in the mnemonic is an example of and what it is supposed to represent in our software. First of all, every part of the STRIDE mnemonic is an attribute that we *do not* want our program or system to have. As seen in Table 3.1 we can see that *Spoofing* is a breach in the authentication of users and this is not something that



Threat	Property	Definition	Example
Spoofing	Authentication	Impersonating something or someone else.	Pretending to be any of billg, microsoft.com or nt-dll.dll
Tampering	Integrity	Modifying data or code.	Modifying a DLL on disk or DVD, or a packet as it traverses the LAN.
Repudiation	Non-repudiation	Claiming to have not performed an action.	"I didn't send that email," "I didn't modify that file," "I certainly didn't visit that web site, dear!"
Information Disclosure	Confidentiality	Exposing information to someone not authorized to see it.	Allowing someone to read the Windows source code; publishing a list of customers to a web site.
Denial of Service	Availability	Deny or degrade service to users.	Crashing Windows or a web site, sending a packet and absorbing seconds of CPU time, or routing packets into a black hole.
Elevation of Privilege	Authorization	Gain capabilities without proper authorization.	Allowing a remote internet user to run commands is the classic example, but going from a limited user to admin is also EoP.

Table 3.1: The STRIDE Threats, with examples by Adam Shostack.[34]

we would like to have in our programs or systems. Each element in STRIDE is actually presenting a possible breach or a property that is being violated. The other properties that might be violated is: Integrity, Non-repudiation, Confidentiality, Availability and Authorization.

Now that we know what kind of properties we need to handle in our threat model, we can look at very basic example. A financial institution has a database containing all necessary information about customers.

This data is highly private and a security breach of a database will have serious consequences for the financial institution. Some example threats against a database might be:

- Someone pretends to be a customer service representative, to gain access to the database.
- A disgruntled employee with malicious intent decides to change all the phone numbers to the financial institutions customers in the database.
- The same disgruntled employee denies having changed phone numbers to all customers.
- Some undisclosed files containing the institutions future employee termination plan is made available to all employees.
- The database is made unavailable due to high load.
- Customers are given rights to read documents not meant for them.

### 3.1.1 Violation of Authentication

It might seem obvious to many that a violation of authentication might be a person using another colleague's username and password to access files or information that he or she is not eligible to access, but authentication problems might be much more than just this. Spoofing in general is when someone is claiming to be someone they are not. This might range from accessing files using your coworkers password, to claiming you are the lawyer of a recently deceased member of british nobility with no heir. In both cases someone is trying to hide their true identity, while performing, or claiming to perform, tasks that they are not really authenticated to do.

### 3.1.2 Violation of Integrity

It is common knowledge that you should never trust what you read and that you should always check sources before accepting something as a fact. This is similar to the property of integrity. The question: "*Is this file the real file, or has it been modified, changed or placed there for me to find?*" is a valid question when it comes to information security. Tampering of data is a serious problem for all kinds of businesses. If our disgruntled employee successfully modifies all phone numbers registered to all the customers in the financial institutions database, then the whole database is useless since it contains false data and we can no longer trust this database anymore.

Usually, tampering is modifying data residing on disk, but it is also possible to have modifying of data on the network or in memory. While tampering of data on disk might be achieved by editing an already existing file, it is also possible to maliciously create files on the disk. This is a typical tampering attack where an attacker creates a file that a program is supposed to create by itself, just to enter false data in it before the program can put a lock on the file. In either case, the goal is to falsify data to make them lose value and usability.

### 3.1.3 Violation of Non-repudiation

*“I didn’t do it”* is a common phrase to hear when something has gone wrong. *Non-repudiation*, or accountability, is a state where it is not possible to dispute an act, like claiming it was not you who tampered with all the phone numbers in the customer database. This is a crucial part of information security, to be able to ensure accountability in a system. For all users, administrators or other people with access, it should be possible for them to prove they did or did not do something. The easiest way to ensure that it is possible to have non-repudiation in a system is to have, retain and analyze logs for every action. As with the disgruntled employee who denies having changed all the phone numbers in the database, by logging who did actually commit the changes in the database logs, the financial institution should have enough proof to disprove the statement made by the employee.

### 3.1.4 Violation of Confidentiality

In most cases, information disclosure is perceived as person A disclosing something to person B, that he or she should not know or does not have the authorization to know, but it is also possible to unintentionally disclose information to a person without the correct authorizations. An example of a common mistake is by logging too much. Even though we need to log things happening in our system we also have to carefully select what we need to log and what data we can omit from each log entry. Information disclosure might happen by accidentally writing secret data to log files containing lower authorization levels than the original file. If the boss of the financial institution tries to save a file to his backup server named *Termination Letter for Alice.docx*[22] but it fails and a log entry is placed in the logs saying *ERROR: Could not save “Termination Letter for Alice.docx”, the disk is full*, then there is an information leak. The file name itself is data that was supposed to be sensitive and only accessible from the boss’ computer or user account, but it is now accessible to anyone who has permissions to see the backup disks error logs.

These examples might seem pretty farfetched, but this is actually common problems that might not have been discovered while implementing the system or during the development of the computer program.

### 3.1.5 Violation of Availability

Businesses can only make money while their services are operational. If the customer database is unavailable, customers will not be able to buy services from the website. To financial institutions this might be services such as loan applications, opening accounts and selling insurance policies. Denial of service is a very serious threat to businesses, and might happen both intentionally and unintentionally. It is therefore important that we implement safeguards that will protect the system against these types of threats. One of the more common threats are *Distributed Denial of Service* (DDOS) attacks and they can be prevented by using a load balancer system.

### 3.1.6 Violation of Authorization

This is the most important threat that we need to discover and mitigate. Elevation of Privilege is allowing someone to do something that they are not authorized to do. This might be a user running code on a computer as an administrator, or someone paying bills from an account they do not own. There are two different types of privilege related violations. It can be achieved by corrupting a process on a computer, but this will only work presuming that a potential attacker already has some control of the computer or the system. The second type of elevation of privilege threats is because the system has buggy access control checks or it may not have any access control checks all together.

### 3.1.7 Using STRIDE on Data Flow Diagrams

STRIDE is a threat modeling tool that is excellent to use on Data Flow Diagrams (DFDs). DFDs are diagrams portraying which parts of a system or a computer program that communicates with each other and in which directions the communication is going. In many cases there are only one way communications from parts requiring lower privilege levels to parts requiring higher. Data is then passed between parts and might be subject to threats, but in this case it is not important since the data is passed on to parts of the system which have higher privilege.<sup>1</sup>

---

<sup>1</sup>Authorization levels might not be as simple as in this example.

The basic principle here is to decompose the system or the computer program into parts and check that each part is not susceptible to relevant threats.

A normal DFD uses four different elements used in modeling a system: data flows, data stores, processes, and interactors. When we are supposed to threat model a system we will have to add another element called a trust boundary. Trust boundaries are represented with dotted lines covering a certain part, or parts, of a system. Trust boundaries represent boundaries between trusted and untrusted parts of the system. Untrusted parts might be something with a lower privilege level than another part, or might also be parts that are completely public. For financial institutions this might be that a customer should not be able to gain access to their account balance directly from the institutions mobile banking application, but should have to authorize themselves before accessing private data. Displaying trust boundaries in a DFD makes it easier to isolate fragile parts of a given system.

### 3.1.8 STRIDE-per-element

STRIDE is a very easy tool to use while looking for threats, and there are different variants of STRIDE that may be important supplements to the traditional STRIDE model. One of these can we see in Table 3.2. STRIDE-per-element is a tool that is supposed to make STRIDE more effective when it comes to finding threats. It does so by acknowledging that certain elements are more likely to be vulnerable to certain types of threats than others. For example, a data store is unlikely to spoof another data store (although running code can be confused as to which data store it is accessing.)[22]

Element	Spoofing	Tampering	Reputation	Information Disclosure	Denial of Service	Elevation of Privilege
Data Flows		X		X	X	
Data Stores		X		X	X	
Processes	X	X	X	X	X	X
Interactors <sup>2</sup>	X		X			

Table 3.2: STRIDE-per-Element[38]

---

<sup>2</sup>External entities

When using the STRIDE-per-element approach we use the table that we see in Table 3.2. We choose one of the parts from the DFD and see which category it fits into. If we are threat modeling a customer database we know that it fits into both the *Data Flow* and the *Data Stores* category. Using the table we can limit our analysis to threats by Tampering, Information disclosure and Denial of Service. STRIDE-per-element to focus on a specific element at a time while looking for threats will make it easier to find threats.

### 3.1.9 STRIDE-per-interaction

STRIDE-per-interaction is a similar variant of the STRIDE model which is a simplified approach that will make it easier to understand how to identify threats. The STRIDE-per-interaction method is centered around each interaction that can happen to a specified part within the computer system. STRIDE-per-interaction will most likely produce the same results as STRIDE-per-element, but the benefit of STRIDE-per-interaction is that it might be easier to understand each threat with this approach.

### 3.1.10 Summary

STRIDE is a useful set of approaches for finding threats against computer systems or computer programs. Using STRIDE, or one of its variants, should make it easy to categorize and discover threats that normally would not see. STRIDE-per-element makes it easier to choose which types of threats to prioritize an certain parts of a computer system or computer program, while STRIDE-per-interaction helps us analyse each possible interaction on a specified part of a system.

Even though STRIDE is a useful tool, the problem of actually discovering a threat is still present. A tool like STRIDE is not very useful if the person doing the threat modeling is not familiar with the system in question.

## 3.2 Attack Libraries

STRIDE as a threat modeling tool may be seen as too high level and that it should be replaced with something more detailed.[22] An example could be a detailed list of common threats to computer systems or other thing that might go wrong. Such detailed lists of threats are called reference tables or *Attack Libraries*. Having lists of attacks or threats can be a very useful tool during threat modeling as it gives concrete examples of the most common problems and might make the threat model more complete. We discuss different types of attack libraries and try

to highlight strengths and weaknesses with different attack library approaches.

### 3.2.1 Level Of Detail in Attack Libraries

Critics of STRIDE has argued that STRIDE is too abstract and that it might make it an inefficient tool to use while threat modeling. This is measured over a scale between abstract, or maybe too abstract, and fully detailed. A fully detailed list is, in this case, a theoretical list containing every possible threat to every possible information system present and in the future. That list would not in any way be practical as it would contain an unlimited number of possible threats, however there are existing attack libraries that are closing in on a “*practical limit*” on how user friendly and effective it is.

While creating an attack library it is very important to both consider the scope of the list, and its audience. In short, do not create a list where you explain every little detail if your audience are security experts, and do not create a list containing hardware security threats if the scope is communications threats.

One of the problems with attack libraries is that they are very time consuming to create. “*Developing a new library requires a very large time investment, which is probably part of why there are so few of them*”[22] There does however exist one very extensive community created list called *Common Attack Pattern Enumeration and Classification*. (Hereinafter referred to as CAPEC) This is a very detailed and highly structured list that seems to close in on the “*practical limit*” for how large and extensive an attack library can get and still be efficient.

#### Common Attack Pattern Enumeration and Classification

CAPEC is an attack library created by MITRE (<http://www.mitre.org>) which is an not-for-profit organization working on different aspects of security.[23] This list is, as of this writing, composed of 463 different *Attack Patterns* which is organized in 16 different *Attack Categories*.

Adam Shostack makes an interesting note about CAPEC on how extensive this list actually is: “*Reveiwng [a system part] against the individual entries is a large task, however; if a reviewer averages five minutes for each of the 475 entries, that’s a full 40 hours of work.*”<sup>3</sup> It is in most situations, unfortunately, not possible to create such extensive threat models every time since computer programs and computer systems consists of multiple parts which each would need roughly 40 hours of work each.

---

<sup>3</sup>CAPEC has reduced the number of threats from 475 to 463, but the quote does still illustrate the amount of work necessary to complete a full CAPEC analysis.

- Gather Information
- Deplete Resources
- Injection
- Deceptive Interactions
- Manipulate Timing and State
- Abuse of Functionality
- Probabilistic Techniques
- Exploitation of Authentication
- Exploitation of Authorization
- Manipulate Data Structures
- Manipulate Resources
- Analyze Target
- Gain Physical Access
- Malicious Code Execution
- Alter System Components
- Manipulate System Users

Figure 3.1: List of the 16 categories found in CAPEC[11]

### 3.2.2 Summary

Even though attack libraries is not the blueprint on how to threat model, it gives a whole other perspective to threat modeling compared to STRIDE. Using abstract approaches like STRIDE, we need to think about and discover each threat ourselves and this might be very time consuming. Detailed attack libraries makes it much easier to find threats by listing them one by one, and using a tool like CAPEC which is a very structured and comprehensive list, makes it easy even for beginners to the art of threat modeling. However, using CAPEC we see that his approach is also very time consuming due to the level of detail in the list.

## 3.3 Attack Trees

Both STRIDE and attack libraries are theoretical approaches on how to discover threats. Attack trees is a graphical way of discovering threats. Using attack trees we get a visual representation of a threat and ways of executing the threat on a given computer system. It is also possible to create general attack tree patterns that can be applied to multiple



systems or system parts. Attack trees can be used to threat model in different ways. The two most common ways of using attack trees are either by using already existing attack trees on your system, or creating specific trees to a system. It is usually recommended to use already existing trees and perhaps build upon them to make them fit your computer system.

The general way of using attack trees are similar to that of STRIDE. We take a look at the DFD of a computer system and isolating specific parts of the system. Then we iterate over each node in the attack tree and see if that node is a threat against your system and then repeat this for all parts of the system and all the trees relevant to your system.

### 3.3.1 Attack Tree Components

Bruce Schneier wrote in 1999 an article about attack trees where he explains them in an easy way. *“Attack trees provide a formal, methodical way of describing the security of systems, based on varying attacks. Basically, you represent attacks against a system in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes.”*[31]

For each threat, or goal as Schneier puts it, we create a root node in a new tree. This root node represent the threat we are trying to execute and for each child in the tree it represents a subthreat or a way of executing the threat. In an attack tree we have two types of nodes. We have *AND-nodes* and *OR-nodes*. When a node is an OR-node we have the possibility to choose either one of them to achieve our goal or subgoal. However when we have AND-nodes we have to accomplish all of them together for achieve our goal or subgoal. If we return to the database example we might consider the threat that *“The database gets deleted”* we can create a simple attack tree to model this threat.

As we can see in Figure 3.4 we have two ways of achieving our goal. We can either *Break into the data center* or we can *Use remote access*. Both of these are just as good to execute our threat, but as we can see on the right side of the tree it might be a longer way doing it from a remote location. We can also see the difference between OR-children and AND-

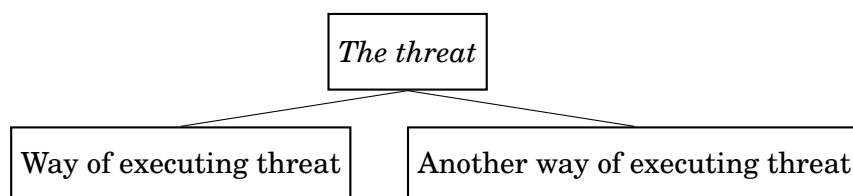


Figure 3.2: A theoretical attack tree using OR-nodes

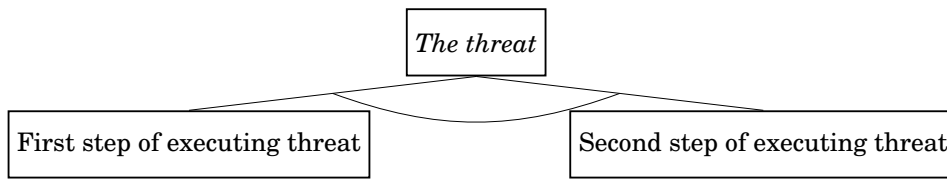


Figure 3.3: A theoretical attack tree using AND-nodes

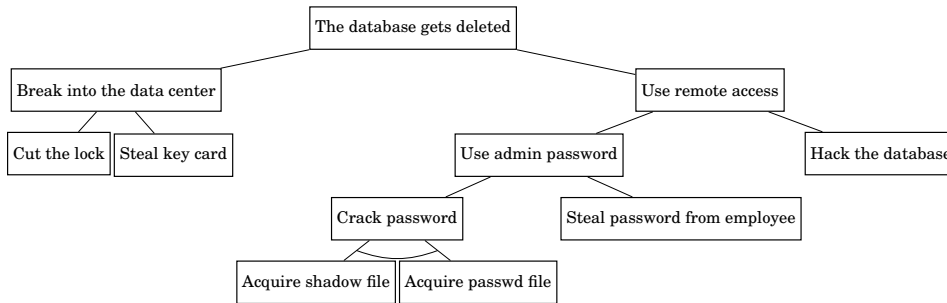


Figure 3.4: A simple attack tree on a customer database

children in this example. From the node *Crack password* we see that we need to accomplish both *Acquire shadow file* and *Acquire passwd file* in order to be able to crack the admin password.

Using an attack tree like this we can easily discover if our computer program or system has any security flaws against this threat. The problem with attack trees is however that creating new trees may be a tedious and time consuming task.

### 3.3.2 Creating new Attack Trees

While creating new attack trees we need to consider what kind of tree it is supposed to be. We can create AND-trees or OR-trees. The type of tree decides its *representation*. In most cases we will create OR-trees. Figure 3.4 is an OR-tree because the root node is an OR-node with two children that both is a possible way to achieve the goal and that they do not rely on each other. The theoretical tree that we can see in Figure 3.3 is an AND-tree because both children to the root node has to be completed in order to achieve the goal. Most attack trees will be OR-trees since it often is possible to execute a threat in many different and independent ways.

After we have decided on a representation we need to choose a *root node*. The root node will contain the threat that we want to model. In Figure 3.4 the root node contains the threat “*The database gets deleted*”. In this example we have a database that we are afraid that might get deleted. We then continue to add subnodes for each way that we can

execute the threat.

One of the harder things with creating attack trees is the same problem that we have with attack libraries. When creating an attack library we need to keep it short enough so that it will be practical to use, but long enough so that it covers every probable threat. It is very important to keep the attack tree within a practical length, while also covering the threat. This problem is called completeness and there is no blueprint on when an attack tree is complete.

### 3.3.3 Summary

Attack trees are useful to visualize threats. By choosing the right representation and the right root node, it should be easy to find weaknesses in a system using this approach. Creating new attack trees is a very time consuming task if we try to keep every attack tree close to the “*practical limit*” of usability.

## 3.4 Discussion

There are many different tools available to use while threat modeling. The three methods I have chosen here each represent a different way of visualizing and approaching the difficult task of creating a threat model for a computer system. STRIDE seems like a very straight forward tool to use. It consists of 6 categories and you try to find one or more threats that fits into each category. It has two alternate versions which simplifies things even more and this makes it a great tool for categorizing and listing threats.

There is however a great problem for beginners to threat modeling. It is not always very intuitive what types of threats we need to look for in each category. Luckily we have other approaches to threat modeling which helps us with this task. Using an attack library helps us to check for the most important threats in each category. When using attack libraries we should consider both the scope and the audience of the threat model and we should consider using an attack library that is not too abstract nor should it be too detailed.

When looking at both STRIDE and attack libraries we see that they are tools with flaws that if we combine them, eliminate each others flaws and should make for a very strong and useful tool. STRIDE is a very abstract tool that makes looking for threats a task with infinite solutions, while an attack library might limit the number of threats. An attack library like CAPEC might be too detailed and contain too many entries that it is not possible, time wise or money wise, to be able to consider them all. The combination of using STRIDE-per-element (Ta-

ble 3.2) and a detailed attack library like CAPEC could make threat modeling easier as it solves the problem of having to check every part of the system for all the entries in the CAPEC library.

Attack trees might be a great tool for presenting threats, it does not present itself as the most efficient approach when trying to discover new possible threats. Attack trees suffers from the same problem as STRIDE does since it is a very abstract tool. We could, like with STRIDE, use it in combination with an attack library but it takes a long time to create the trees. Attack trees appear to be a tool that is more suited to help during the threat mitigation process than discoring them. An attack tree is very suitable to present threats to an uneducated audience and should therefore be considered as supplementary tool.

### 3.5 Summary

In this chapter we have given a brief overview of three very different approaches to threat modeling. Two abstract methods and one detailed. STRIDE is an abstract method of threat modeling using a table with 6 headlines, or categories, which should help the threat modeling team to start finding threats against a certain part of the system. STRIDE have also two simplified modifications that will help removing unnecessary work from the threat modeling process. STRIDE-per-element is a modification where the type of system or type of part of a system decides which of the six STRIDE categories we should consider. This will narrow the search for threats down and make the task easier. STRIDE-per-interaction uses STRIDE-per-element to narrow down the set of possible threats, but instead of using it on the system part, it does so on the different interaction that part does. This makes STRIDE-per-interaction a little more time consuming than STRIDE-per-element, but makes it easier for the beginner to understand by also listing all the different interaction a system and this might prevent mistakes.

Attack libraries are lists or a set of common threats that one should consider. There is no blueprint of how an attack library should be, so the variety ranges from short lists to lists containing several hundreds of threats. Shorter lists are defined as more abstract since they contain little specific information about threats while longer lists are defined as detailed. CAPEC is an open source attack library that contains 463 different threats categorized in 16 different categories. This is a very detailed list and is a great place to start. Critics of this attack library has stated that it is too detailed, making it impossible to consider each and every threat to every part of a computer system.

An attack tree is another abstract way of doing a threat analysis. Attack trees consists of a root node containing a specific threat and sub-

odes containing ways of executing the threat. There are two types of attack trees: *AND-trees* and *OR-trees*. In an *OR-tree* ( Figure 3.2 ) we can achieve to execute the threat with only successfully achieving one of the subnodes, while in an *AND-tree* ( Figure 3.3 ) we need to complete all of the tasks in the subnodes.

## Chapter 4

# Mobile Operating Systems

In this chapter we discuss the current state of mobile operating systems, how they work and how they differ from traditional desktop operating systems. This chapter is supposed to be a brief introduction to the major mobile operating systems and we will present the three most common mobile operating systems on the market: *Android*, *Windows Phone* and *iOS*. We also do a comparison of them to see if there are any differences in the security design.

### 4.1 Introduction

Mobile operating systems are a relatively new branch of operating systems. Real smartphones have only existed for 7 years and after the release of Apple's iPhone, several other mobile operating system projects were established. This makes mobile operating systems relatively young compared to traditional desktop operating systems like *Windows*.

Since the initial release of the *iPhone*, smartphones and tablets have really taken over the role of the traditional computer, and 80 % of the Norwegian population had access to a smartphone in 2014.[36] This makes smartphones one of the major platforms used to access the internet and it is also one of the major platforms for users to access their bank accounts.[15]

### 4.2 Security Features

To be able to analyze and compare the different mobile operating systems we need to take a look at the different parts of the smartphone. We will separate operating system security into four different categories:

- Secure Boot.
- Encryption and Data Access.

- Application Security.
- Device Security.

### 4.2.1 Secure Boot

The first step to ensure that a mobile operating system is secure, is by having a booting mechanism that is secure. It is important to ensure the integrity of the operating system between every reboot, so that we know that no one has tampered with the operating system while it was shut down. One type of threats that can attack an operating system in this way is “*root kits*”. A root kit is a tool that gives a user or a process an unauthorized level of access to system resources. It is important that the device boot mechanism prevents that unauthorized code gets loaded before or during the operating system boot.

In Figure 4.1 we can see an example of a mechanism for secure boot. This kind of secure boot is called *chain-of-trust*. Chain-of-trust is when there is a hierarchy of code signing while booting the device. The booting mechanism starts with the device hardware being coded to only start code that is signed with the correct certificate. This feature prevents the hardware from loading unauthorized code. After the hardware boots valid code a chain of signed code is started with each level checking the next making this a chain-of-trust that prevents any unauthorized code from being executed while the operating system is loading.

### 4.2.2 Encryption and Data Access

After the operating system is done booting it needs to be able to make sure that the user has secure access to its files. This is achieved using encryption on user data and other system files on the device.

While encrypting files sound like an easy task, it is also important that the operating system can guarantee for three properties: *Confidentiality*, *Integrity* and *Availability*. (Hereafter referred to as CIA.) Confidentiality is the property of keeping private data private. This is not only limited to unauthorized users reading private data, but it is more generally referred to as unauthorized users learning sensitive information. Integrity is the property of keeping data valid. If a file containing the password hash for a user has been tampered with it is no longer valid and there is a data integrity breach. The last property of CIA is to make sure that system resources and user files are available when the user needs it. In ISO 7498-2[16]: “**Availability** - *The property of being accessible and usable upon demand by an aauthorized entity*”. You (the entity) having the key to your apartment (the resource) door is not helpful if the door is stuck (not accessible) due to a flaw in the door

design. It is important that the operating system maintains full access to resources and data to keep the property of availability.

All of these properties makes an argument for having data encryption built in to the operating system. By having all files encrypted by the operating system it should be possible to guarantee that user data is kept confidential to unauthorized users, data integrity is ensured since the data hash could reveal if the file has been tampered with and the encryption locks the file to prevent tampering from happening and availability is ensured since unauthorized users should not be able to put a lock on unauthorized files.

### 4.2.3 Application Security

Almost every smartphone is running, or is able to run, third-party applications today. These third-party applications are applications that the user has downloaded through an integrated downloading service or by downloading it directly from the internet. These applications creates some insecurities if they have access to critical parts of the operating system or system services. third-party applications that creates insecurities like this are called “*Malware*”. Malware is a computer program that either infects systems and uses them maliciously or a program that tries to breach one of the CIA properties and the operating system should implement protection mechanisms against such applications.

### 4.2.4 Device Security

Every device is in danger of being lost or getting stolen. This poses a threat to the device and every device should have device security functionality. Normal security functions could be PIN codes or password protection. Device passwords or PIN codes should be manageable by a *Mobile Device Manager* (MDM) and the MDM should be able to enforce different security policies such as enforcing a password length of minimum 8 digits or forcing the user to use characters and digits in the password.

## 4.3 Android

Android has the largest install base of any mobile operating system to date. It has been on the market since 2008 and has since then had 8 major releases.[4]. The developers behind Android have focused on strengthening Android security features during the last major updates to make sure that this platform is one of the more secure mobile operating systems available on the market.



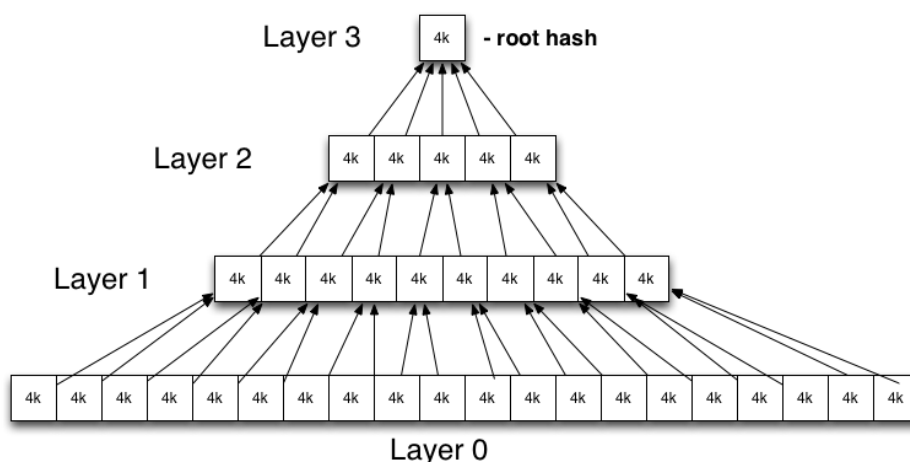


Figure 4.1: An example of the *Chain-of-trust* mechanism (dm-verity) in Android.[5]

### 4.3.1 Secure Boot

Later releases of Android have secure boot as a built-in user optional security feature. This service is called *device-mapper-verity* (dm-verity) and is implemented as a measure to prevent *root kits* from being able to take advantage of vulnerabilities during boot. This security feature is implemented in the kernel that Android uses, and if malware is successfully loaded before the kernel starts its boot process, it will still be able to pose a threat to system integrity.

### 4.3.2 Encryption and Data Access

Android uses the Linux-kernel as the fundament for the operating system. This means that Android can use all the security features that the Linux-kernel offers. The Linux-kernel is used by millions of desktop computers worldwide. Due to the kernel's age and long service life, it is believable that the it is stable and secure to use.

The Linux-kernel is open source. This implies that the kernel is subject to research, that it gets tested and attacked, and that vulnerabilities get mitigated by anyone that show interest in Linux security. Since there are no secrets to the code, this also guarantees that anyone fixing vulnerabilities can not intentionally hide any possible vulnerabilities without people being able to review his code. The Linux-kernel is still under development and all major vulnerabilities that gets discovered is handled and the kernel is getting updated with new releases of the Android operating system.

Since Android is using the Linux-kernel it can benefit from the security functionality that is already built in. One of these features is the user administration functionality. The Linux-kernel is designed so it can manage to have multiple users on the same device. One of the fundamental security features is to isolate different users resources from one another. It does this by:

- Preventing user A from reading user B's files
- Ensuring that user A does not exhaust user B's memory
- Ensuring that user A does not exhaust user B's CPU resources
- Ensuring that user A does not exhaust user B's devices (e.g. telephony, GPS, bluetooth)

All encryption on Android is done using 128-bits AES with cipher-block chaining (CBC) and ESSIV:SHA256. All new devices need to have a built inn security module to be eligible to ship with Android 5.0 or newer preinstalled.

### 4.3.3 Application Security

Android utilizes the user administration security functionality when it installs and runs both standard applications and third-party applications. Each new application is assigned its own unique user id (UID) and each application runs as a new process with this UID. This process of isolating each application is near perfect since it forces each application to run in its own little *sandbox*. We can see in subsection 4.3.2 that two different users are not able to access each others memory space, CPU resources or data files. The application runs as if there are no other applications present on the device.<sup>1</sup> By running applications in its own sandbox we also potentially prevent memory errors from compromising the system. Other applications or the operating system cannot use the physical memory that is allocated for the application sandbox. By using this kind of isolation we can assume that the application security in Android is more than sturdy enough.

One of the weaknesses on how Android manages third-party applications is how the application can request access to sensitive APIs. By default, Android will protect its users by limiting access to a range of APIs. An application will not have access to the GPS sensor without the direct consent by the user. The user will have to explicitly grant access to the API in question and this is usually done during the installation

---

<sup>1</sup>There are possibilites of having applications knowing about each other, but we will not discuss them here. This is merely a simplification to clarify how it works.

process. When installing the application, the user is presented with a list of APIs that it will have to grant access to in order for the application to work properly. If the user denies the application access to one or more of the APIs, then the installation will be cancelled. This puts the user in charge of what type of data each application can access, but there are some weaknesses here as well.

Some applications can share data through *Inter Process Communication* (IPC). Malware can through this kind of process communication send sensitive data to each other. Application A requests access to the GPS sensor, while application B requests access to the network. Now these two applications poses a threat to the system since one application can acquire the users position and the other can send that through the network.

The Android marketplace is the web shop where users can buy and download third-party applications for their device. A developer must digitally sign its code for the application and the developer must pay a standard fee of \$ 25.[5] This fee is put there to prevent developers from publishing malicious applications to their web shop, *Google Play*. There is however some weaknesses to this security method. The only requirement to pay the fee is a credit card, by using anonymous credit cards like *Visa Spendon*[40], a developer can anonymously pay the registration fee. There is also a lack of source code control before the application gets launched to the web shop. There is also a weakness to the code signing since anyone can generate the certificates needed to sign code.

Android has since Jelly Bean 4.2 had an optional service called *Verify Apps*. This services helps the user make smart decisions when installing third-party applications on their system. On Android, users can manually download *Android Application Packages* (APKs) and install them without having to use the Google Play store. This means that developers can create applications and distribute them by forged emails or a fraudulent website. Verify apps can be a great countermeasure to these kind of threats but this service is optional and is deactivated by default. Since the user needs to manually activate the service, it can not count as a core security functionality.

#### 4.3.4 Device Security

Android's latest release, offers 4 types of device security. Default/Open, PIN, Password or Pattern. The operating system does encrypt all files on the device using the PIN, password or pattern provided as a key. This ensures that data files can not be accessed without having the key generated by the type of security on the device. If the user chooses not to use the default security option, all files are still encrypted using a standard password.

## 4.4 Windows Phone

Windows Phone is the mobile operating system developed by Microsoft. It was initially released in 2010 under the name *f*Windows Phone 7. Windows Phone is the youngest of the three mobile operating systems that we will be investigating in this chapter. Windows Phone has been developed with focus on meeting business requirements, and security has therefore been a primary focus during the development. This includes the integration of the Exchange protocol which is used by many businesses today.

### 4.4.1 Secure Boot

The latest release of Windows Phone uses the well known Windows NT-kernel. Windows Phone uses the same kernel as is used in the desktop versions of Windows and Windows Phone can therefore utilize the same security functions. Earlier versions of Windows Phone, namely 7.0 and 7.5, did use the simpler Windows CE-kernel which limits operating system functionality. Microsoft switched to the NT-kernel from Windows Phone 8 and the future Windows 10 will also use the NT-kernel.

Windows Phone uses UEFI instead of BIOS, and UEFI has built in secure boot functionality. UEFI boots the hardware boot code which in turn will try to load the Windows Phone bootloader. UEFI also validates the boot loader which then should guarantee that the system integrity is maintained. This functionality is called *Secure Boot*.

After UEFI has initialized the boot process and validated the Windows bootloader, Windows starts its own secure booting mechanism. This mechanism is called *Trusted Boot* and is executing the same chain-of-trust-measures which verifies the code on each new step. This is similar to Android's service, *dm-verity*, that can be seen in Figure 4.1. Trusted Boot verifies each segment in the booting process and ensures that the system has not been tampered with while being switched off.

### 4.4.2 Encryption and Data Access

Windows Phone protects its users against information theft by encrypting the whole device using 128-bit AES encryption. The encryption key is managed by a dedicated encryption module called a *Trusted Platform Module* (TPM) which is a standardized encryption engine which helps the operating system to securely store keys and encrypt user data.

To ensure that native applications and third-party applications can not access each others data files, memory, or important system files, each application runs in its own sandbox. This sandboxing is named an *AppContainer*. Each AppContainer has its own dedicated memory space

which is protected with Address Space Layout Randomization (ASLR) and individual rights and access to system resources like the GPS sensor.

### 4.4.3 Application Security

All applications that can run on a Windows Phone need to be signed with a Microsoft issued certificate. If any part of an application, or a whole application does not comply with these restrictions it will not be able to run on the device. This kind of security measures is an important part of reducing the possibility of malicious code to run on the device. To get approved, any application code must be signed by the developer and then signed by a Microsoft employee. This means that all applications that are available through the *Windows Phone Store* is pre-approved and controlled by a Microsoft employee and should be considered secure. This creates an extra layer of security for a malicious developer to breach. A developer can not simply hide malicious code in his binaries, since he needs to disclose his code to a Microsoft employee.

### 4.4.4 Device Security

Windows Phone supports the ability to limit device access by the usage of passcodes. Passcodes might be simple four-digit PIN codes or they might be more complex alphanumeric passwords. Using mobile device managers like *Microsoft Exchange*, Windows Phone supports the addition of password security policies on the device. Enforcing the use of stronger passwords make the data on the device secure.

## 4.5 iOS

Apple announced and released its first smartphone in 2007. The iPhone became the first phone in a whole new generation of smartphones. With the release of the first iPhone OS<sup>2</sup>, Apple also changed how phones were designed since it used a modified version of the OS X desktop kernel.

### 4.5.1 Secure Boot

The foundation for iOS is the OS X kernel which is also used in Apple's desktop operating system. The first OS X operating system was released in 2001 and this implies that the kernel has been used and tested for almost 15 years. Apple have made the kernel code publicly available to ensure that there will be security through transparency. If anyone

---

<sup>2</sup>iPhone OS is now renamed to iOS since it supports iPods, iPads and iPhones

wants to discover vulnerabilities in the kernel, they are allowed and able to do so.

iOS implements a secure booting mechanism. For each part of the boot chain the system checks if the next code module is signed by Apple, and that it has not been tampered with. This is done to ensure system integrity even if the system has been switched off. If any code in the boot mechanism has been tampered with, the device will not start since the chain-of-trust is then broken. Elements that are included in the chain-of-trust are bootloaders, kernel, kernel extensions and firmware.

The booting mechanism in iOS starts off with a hardware *root-of-trust*. This is a low level bootloader on a read-only memory (ROM) chip. It is assumed that the boot code on this chip is secure from tampering since it is a ROM-chip that is installed during production of the device. From this root-of-trust the next code in the boot mechanism is validated and then and only then executed.

## 4.5.2 Encryption and Data Access

All new iOS devices have a built-in encryption engine. This is a dedicated 256 bit AES engine which encrypts all files on disk. iOS uses two main keys. One hardware key which encrypts file meta data and one separate key which together with the hardware key encrypts the file content. The hardware key is generated when the operating system is installed on the device, and to access data this key is needed. The MDM service from Apple is called *iCloud*, and iCloud gives the user the possibility of remotely deleting the hardware key from the device. This will result in all data files on the device being made inaccessible.

Every application installed on the device runs in total isolation from one another. Every application is allocated its own *home folder* on disk and its own memory space. The memory space is protected using ASLR and each home folder is locked to a specific UID like in Android. Because of this sandboxing every application is secure from other applications running on the same device. *The majority of iOS runs as the non-privileged user “mobile”, as do all third-party apps.*[20] This makes third-party applications not able to access data files owned by root.

## 4.5.3 Application Security

Apple have focused on application security while designing the iOS operating system. They have implemented a range of different barriers which a developer must pass to be able to develop and distribute applications.

To make an application runnable on an iOS device the developer

must submit its code for review by an Apple employee.<sup>3</sup> The code must be signed by both the developer using a private certificate and it needs to be signed by the Apple employee. This actually extends the chain-of-trust from the operating system all the way to any application running on the device.[20] Developers who wish to create and sell applications through Apple's web store, *App Store*, have to register themselves with the iOS Developer program and they need to pay a yearly fee of \$ 99. This creates a payment barrier which helps identifying developers who deliberately try to add malware to the App Store. When the application is complete the developer submits code for review and if it gets approved the application is made available to the end user through App Store.

#### 4.5.4 Device Security

iOS offers the possibility of password protecting devices. The operating system prompts the user during installation asking the user to set a four-digit PIN code. The user can explicitly request that the passcode should be a longer alphanumeric password instead of a four-digit PIN. iOS does offer password policies through an MDM, which makes it possible to enforce stronger passwords. Together with the device security that the operating system offers while installed, iOS also offers an *Activation Lock* feature. The device itself gets linked to an iCloud account and it is not possible to reinstall the operating system unless the device is unlinked from the iCloud account. This makes iOS devices less desirable targets for theft since the original owner will have to unlock the device.

### 4.6 Discussion

All the different mobile operating systems that we have presented in this chapter started out very differently. iOS was the first operating system and was released in 2007, followed by Android in 2008 and Windows Phone in 2010. The starting points for these systems were quite different, but they have during the last few years been adopting technology from each other, which makes all platforms look quite secure on paper.

All of the mobile operating systems have the possibility to restrict access to the device by using passcodes. Passcodes are on all the different operating systems an optional feature to prevent unauthorized user from accessing the device. All operating systems also provide the possibility of enforcing password policies on a device using an MDM.

---

<sup>3</sup>This is similar to the approach Microsoft is using, but it is worth mentioning that this was first done by Apple.

An example policy is to require the user to use an alphanumeric password instead of a four-digit pin. The possibility of enforcing the use of more complex passcodes does however not apply for private consumers as they are usually not a part of a business controlled MDM.

All of the different operating systems support one form of secure booting mechanism. iOS is however the only system which has supported this feature since the first version. Windows Phone did not have secure boot in Windows Phone 7 or 7.5/7.8 but from 8.0 this is an obligatory feature. Windows Phone is not the same operating system between 7.X versions and 8.X versions since they changed the kernel between these major releases. This makes devices running 7.X unsecure due to the lack of secure boot functionality.

Android has since version 4.4 (KitKat) had secure boot as an optional function. The user had to manually activate it and it is not guaranteed that all users running KitKat have activated this functionality. If we take a look at Table 4.1 we can see that there is more than 50 % of the Android user base using versions older than KitKat. This infers that more than 50 % of all Android devices are not protected against root kits.

Version	Codename	Distribution
2.2	Froyo	0.4%
2.3.3 - 2.3.7	Gingerbread	7.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	6.4%
4.1.x	Jelly Bean	18.4%
4.2.x		19.8%
4.3		6.3%
4.4	KitKat	39.7%
5.0	Lollipop	1.6%

Table 4.1: Distribution of Android devices that accessed the Google Play Store.[4]

There is also one minor detail about dm-verity that is worth noticing. dm-verity is optional for users running KitKat, and it is activated by default for users running version 5.0 or newer. But if the user had 4.4 installed on the device with dm-verity disabled and then upgraded to 5.0, the user is still susceptible to root kits since if an attacker manages to downgrade the Android version back to KitKat, dm-verity will be disabled. This is a minor security breach that is worth further investigation.

Android, Windows Phone (8.0 and newer) and iOS all use renown



and well tested kernels. There are only two security aspects that we need to highlight. The first is that devices running Windows Phone 7 are using the simpler Windows CE kernel. Newer versions do however use the more widespread Windows NT kernel which provide more security features than Windows CE. The other aspect that needs attention is that older versions of Android not necessarily uses *Security Enhanced Linux* (SELinux). SELinux is the kernel used by Android Version 4.3 (Jelly Bean) or newer. This means that more than 50 % of all Android devices in use are not protected by the enhanced SELinux kernel. “*Unfortunately, existing mainstream operating systems lack the critical security feature required for enforcing separation: mandatory access control. As a consequence, application security mechanisms are vulnerable to tampering and bypass, and malicious or flawed applications can easily cause failures in system security.*”[3]

One of the biggest differences between the more common desktop operating systems and mobile operating systems is how applications run in their own isolated sandboxes. All of the operating systems isolate each application by giving them private memory space and a private home folder on disk. iOS and Android does this by creating a new user on the operating system for each user which limits what the application can do outside its own disk and memory space. subsection 4.3.2 shows how assigning a unique UID per application can restrict application access to the system. Windows Phone does not describe how this feature is implemented, but it is probable that it uses a similar kind of user restriction feature.

All of the mobile operating systems also use ASLR to protect each applications memory space and they all use the *Execute Never* (XN) flag that is accessible on every ARM processor. ASLR is a memory protection mechanism which randomizes the physical addresses used by different components of a process. This should technique makes it harder for an attacker to exploit specific memory vulnerabilities.[14]. XN, is a flag that may be used upon memory regions to tag that this region does not contain executable code. Using the XN flag makes it harder for an attacker to execute code using memory vulnerabilities. This makes it difficult to attack an applications by hacking memory segments and this does prevent someone from injecting code to memory and executing it. All memory segments that are writeable can not execute any code.

iOS and Windows Phone have chosen a security model where every third-party application which is sold through each platforms application store must be signed by both the developer and Apple/Microsoft. After the developer has signed the code it will have to be submitted for review before being able to sell it in the application store. Android does deviate from this security model. Android let users download and install application binaries from anywhere on the internet. Windows Phone and iOS

does primarily let the user install applications bought, or downloaded from free, from each platform's application store. Android does not pre-check applications that are being sold via the Google Play marketplace. Since there is no approval needed to sell applications on Google Play it is most likely multiple applications available on Google Play that would fall into the category of malware. Android does from version 4.2 (Jelly Bean) provide an optional function called Verify Apps which in turn will verify that the application is secure before letting the user install it on their system. This functionality is comparable to the pre-approval strategy from Apple and Microsoft, but it is only user optional and it is in no way a preventive security measure like pre-approving applications.

All developers who wish to sell applications using the built-in application store on Windows Phone or iOS have to register their personal information with their applications. By registering this information and having a registration fee it is a way of filtering out developers that have malicious intentions. Android does have a registration fee, but they do not require personal information. It is possible that there might be a correlation between the security model used by iOS and Windows Phone (register, pay, sign, approval, publish), the security model used on Android (pay, sign, publish) and the number of malicious applications available for each system.

Isolation of applications in Android, Windows Phone and iOS is usually very strong. There is however an exception where applications signed by the same developer often can share resources. This might pose a threat if application A has access to GPS sensor data and application B has access to the network. This isolation technique used in all the different mobile operating systems seems to be a very efficient way of protecting the system.

All of the operating systems supports the usage of a dedicated encryption module. All systems encrypts all data files on disk, either by including the passcode as an encryption key or using a standard password. iOS has by default the strongest encryption with a 256-bit key, Windows Phone uses 128-bit and Android uses 128-bit by default but has the option to use 256-bit if the user request it. There is however not a big practical difference between the 128-bit or 256-bit and all systems are deemed to be secure by today's standard.

As we can see in Table 4.2, there are no major differences between any of the major mobile operating systems. The only real difference is during application development. Android does not have any pre-approval of applications before they are available for the end-user.

---

<sup>4</sup>No pre-approval of applications available

<sup>5</sup>Equivalent functionality

<sup>6</sup>Android 5.0 and newer

<sup>7</sup>Windows Phone 8.0 and newer

	<b>Android</b>	<b>iOS</b>	<b>Windows Phone</b>
Secure Boot	Yes <sup>4</sup>	Yes	Yes
Network Security	-	-	-
SSL	Yes	Yes	Yes
TLS	Yes	Yes	Yes
VPN	Yes	Yes	Yes
Data Encryption	128-bit	256-bit	128-bit
Device Passcode	Yes <sup>4</sup>	Yes <sup>4</sup>	Yes <sup>4</sup>
Application Security	-	-	-
Application Process Isolation	Yes	Yes	Yes
ASLR	No <sup>5</sup>	Yes	Yes
Resource Management	Yes	No	Yes
TPM	Yes <sup>6</sup>	Yes	Yes <sup>7</sup>
Application Verification	No <sup>4</sup>	Yes	Yes

Table 4.2: A visual comparison between systems

If Android would implement this feature, then there are no differences between these systems.

As long as the chain-of-trust is not breached, every system can be deemed secure. We know that it is possible to breach the chain-of-trust in all systems. Both Microsoft and Google have included the possibility to “root” devices for developers to get test applications, and there exist similar, but unofficial, rooting kits for iOS called *jailbreaks*. Systems which are rooted or jailbroken can not be considered as secure due to the fact that the chain-of-trust is broken.

## 4.7 Summary

There are three major mobile operating systems available on the market today. Android, Windows Phone and iOS. Even though modern mobile operating systems represent a young branch of operating systems, the technology has already come far. Every system uses a desktop quality kernel with all the security features those include. Older versions of Windows Phone OS have less security due to the Windows CE kernel, but all new phones uses the Windows NT kernel.

All mobile operating systems are designed with focus on security, and the adaptation of the application isolation feature should be considered a success. By isolating every application from one another, the system can ensure that there is little-to-no unauthorized communication between third-party applications and between applications and the system. All systems use industry standard protection of memory and

encryption technologies. All operating systems are by default secure, but devices that are rooted or jailbroken could pose a security threat for sensitive applications.

There is no major difference between each system except for the developing process. Apple and Microsoft have strict guidelines for how an application should be created and very strict policies that must be followed to make an application runnable on a *non-rooted* device. The code signing and code reviewing practices from Apple and Microsoft makes their systems a bit more secure than Android.



# Chapter 5

## BankID

This chapter is a brief introduction to the history, technology and adaptation of BankID, the prime authentication service used by Norwegian financial institutions.

### 5.1 Introduction

BankID is a collaborative project to create a common system for user authentication between all financial institutions in Norway. This project is a collaboration between Norwegian banks, *Bankenes Standardiseringskontor* (BSK), *BankID Norge* and *Finansnæringens Fellesorganisasjon* (FNO). Each of these institutions have a certain role in the research and development of BankID. The BankID project was started back in 2000 and the first customers were using BankID in 2004. Since then, the user base has grown to more than 3,1 million customers. This makes BankID the leading authentication service in Norway and it is used by both private institutions and the Norwegian government.

### 5.2 Technology

The technology used in BankID authentication is based on Public Key Infrastructure (PKI), which is an ISO framework using public key cryptography and the X.509 standard.[14] PKI is based on asymmetrical cryptography which makes it possible for two parties which does not have an existing relationship to communicate securely using the open internet. *Public Key Infrastructure consists of programs, data formats, procedures, communication protocols, security policies, and public key cryptography mechanisms working in a comprehensive manner to enable a wide range of dispersed people to communicate in a secure and predictable fashion.*[14] In short: PKI establishes a level of trust for BankID customers to communicate securely using the internet.

### 5.2.1 PKI

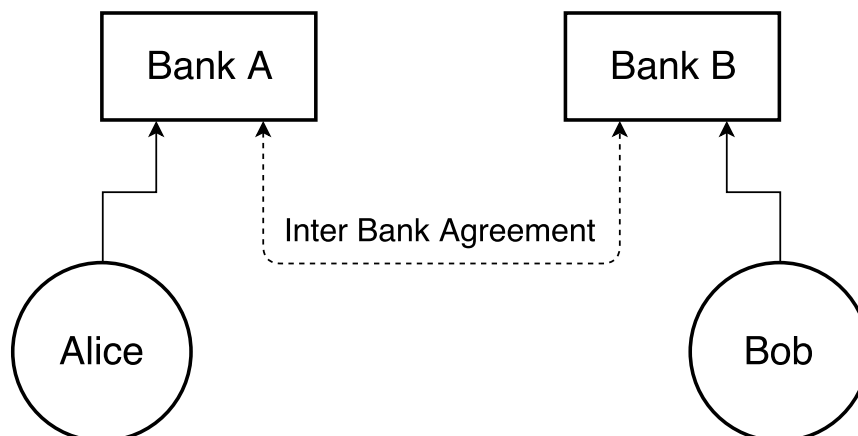


Figure 5.1: Inter bank agreement

PKI enables customers to trust each other by letting each party of the communication have a trusted relationship with a mutual third-party. This third-party is also known as the *Certificate Authority*. (Hereinafter known as CA.) In BankID there are two types of third parties. Usually the third-party would be the two original parties' local bank but all customers do not share the same bank. In that case it exists an *inter bank agreement* to create a chain of trust between banks as well. These *inter bank agreements* are regulated by BankID. Each entity that wants to take part of a PKI needs to get a digital certificate from one of the CAs. To be able to get a certificate from the CA each entity will need request a certificate from a *Registration Authority*. (RA) The RA confirms the entity's identity and initiates the certification process with a CA on behalf of the entity.

PKI supplies security services such as: Confidentiality, Integrity, Authentication, Nonrepudiation and Access control.[14]

## 5.3 Infrastructure

BankID consists of several major components:

- Felles Operasjonell Intrastruktur
- Bank RA
- Bank OTP
- BankID Server
- BankID Client

- BankID on Mobile Client

*Felles Operasjonell Intrastruktur* is a centralized package of multiple subcomponents which binds all other components together. All other components are either hosted by the bank itself or is at the hand of the customer.

### 5.3.1 Felles Operasjonell Intrastruktur

*Felles Operasjonell Intrastruktur* (Hereafter referred to as FOI.) is the back end part of the BankID system that is being operated by a service provider. FOI consists of APIs which are necessary for the rest of the BankID components to communicate with each other. FOI provides a range of functional components that can be seen in Figure 5.2.

- Ordering Certificates
- Issuing Certificates
- Distribution of information of the current state for certificates to the bank RA
- Revoking Certificates
- Suspending Certificates
- Reinstating Certificates
- Renewing Certificates
- Validating Certificates
- OTP<sup>1</sup> routing for validation using the banks own system
- Central storage and usage of private keys
- Central storage and distribution of certificates and public keys
- All cryptographic operations on behalf of the customer
- Carrier communications to activate and use BankID on Mobile
- Certificate and information administration

Figure 5.2: Functionality provided by FOI

The central infrastructure (FOI) is developed, administrated and hosted by Nets Norway (Nets).

### 5.3.2 Bank RA and OTP

Two of the main components of BankID are services that are hosted by the financial institutions. These components are the registration authority and the *One Time Password* (OTP) mechanism. BankID requires



that the RA confirms the identity of a customer by having the customer show up in person and provides valid identification. An example of valid identification is a passport. The RA then request a certificate from the CA on behalf of the customer.

Each bank may chose their own OTP mechanism as long as it is approved by *Bankenes Standardiseringskontor* (BSK). *“The bank will administrate the one time password mechanism and will have functionality which approves the code. Central servers for Bank stored BankID will recieve the code from the BankID Client and send the code by a private network to the banks service for approval of one time passwords.”*[9]

### 5.3.3 BankID Server

The BankID server is the software that is implemented at a merchant with an agreement with its bank to accept end users BankID certificates used for identification or signing.

### 5.3.4 BankID Client

The BankID client is either a Java-applet or a Javascript and HTML5 application<sup>2</sup> that is automatically downloaded to the end users local computer whenever the end user wants to authenticate him- or herself or sign a document. The client is downloaded from the central infrastructure (FOI) every time the user wants to perform an action, and the client is unique every time. This means that the end user has to re-download the client application every time.

### BankID on Mobile Client

The BankID on Mobile client is a mobile application that is provided for all users with a compatible phone, compatible carrier and compatible sim card. Every carrier that wants to offer this kind of service needs to hook themself up to BankID FOI. BankID on Mobile differs from normal BankID authentication by storing private keys on the sim card, instead of being centrally stored at FOI.

## 5.4 Certificates

Since BankID is based on the PKI framework, all entities using the service are being issued certificates. Issuing of certificates starts with a

---

<sup>2</sup>There is also an iOS application that the user can download to his own iPhone, iPod or iPad, and an Android application for Android phones and tablets. However, since the release of the Javascript and HTML5 application, there is no longer any need to use these native applications when wanting to authenticate or sign something.

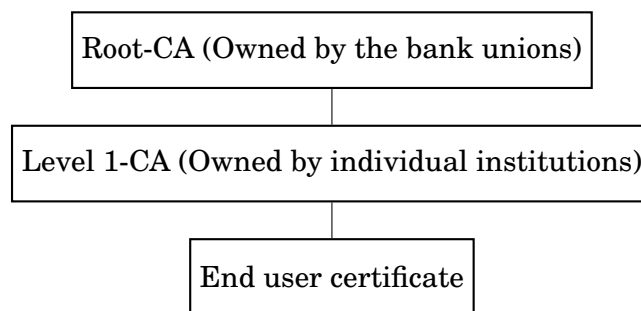


Figure 5.3: Certificate issuing tree

Root-CA<sup>3</sup> which is owned and hosted in collaboration by the two bank unions in Norway, *Sparebankens Servicekontor* and *Finansnæringens Servicekontor*. This CA is used to issue one or more Level 1-CA's. A Level 1-CA is owned by each bank, financial institution or other institution which are supposed to be *Registration Authorities* (RA). A Level 1-CA provides three key functions:

- Issuing Certificates
- Revoking or suspending Certificates
- Generation and distribution of CRLs<sup>4</sup>

When an institution has their Level 1-CA they can act as an RA to issue BankID certificates to customers. There are four different types of end user certificates in use by BankID today. Person-BankID, Employee-BankID, Mobile-BankID, Merchant-BankID. Person- and Mobile-BankID are used by private citizens, and these two differ only in the matter of where private keys are stored. In Person-BankID the private keys are stored by the issuing institution and for Mobile-BankID the private keys are stored on the SIM-card. Merchant-BankID is used by websites/companies and the Employee-BankID is used by individuals whom are acting on behalf of an organization or business. The certificate tree can be seen in Figure 5.3.

#### 5.4.1 Certificate Format and Security

All BankID end user certificates are formatted to the X.509 standard. X.509 is a PKI certificate standard published by the ITU-T.[21] The X.509 specification profiles the format and semantics of certificates. It

---

<sup>3</sup>Certificate Authority

<sup>4</sup>Certificate Revocation List

also profiles the CRLs for the Internet PKI. It contains procedures for processing certification paths in the internet environment.

To protect communication between the RA of the institution holding a Level-1 CA and the Certificate Ordering (also known as: ODS) system in FOI, there is issued an SSL client certificate on the CA that the institution is issuing certificates from, and there is issued a signing certificate on the same CA. This protects the communication between the institutions systems and the systems that are a part of FOI by making the communication secured with two-way SSL over a closed network. It also protects data integrity using the signing certificate since all messages sent from the institution's RA are signed with this certificate and all responses from the ODS is signed in the same way.

## 5.5 Encryption

Key encryption is one important aspect of BankID. BankID differs from X.509 PKI by using a third-party to perform all cryptographic operations. Since all private keys (except for Mobile-BankID) are stored in a central storage, the user is required to request access to its own private keys from this entity.[25] When a customer wants to perform a cryptographic operation it will be done by the central infrastructure called FOI.

All keys used in FOI are protected using strong encryption technologies. For all private keys stored for the customer at FOI the cryptographic key size is no less than 1024 bits for both signing keys and identification keys. All private keys 2048 bits large and hashed using SHA256 algorithm. BankID on Mobile uses a smaller, undefined key length, but BankID is currently working to upgrade this key length to the same as for bank stored keys.

All keys are stored using a *Hardware Security Module* (HSM) which is an external physical device attached to the system. This HSM module is the one doing all cryptographic operations for customers together with storing private keys.

## 5.6 Process

In this section we will describe the different processes of identification using BankID.

### 5.6.1 Bank Stored BankID

The signing process using bank stored BankID is, as shown in Figure 5.4, very complicated. First, the client verifies if the merchant is

legitimate, then both the client and the merchant challenges each other and the client sends the challenge to the central infrastructure. FOI then validate the merchant certificate, validate the OTP, gets the private key and returns a signed challenge to the client. The client forwards the signed challenge to the merchant, which then validates the client certificate using the BankID VA. This process is used on what is described as a secured connection.

### 5.6.2 BankID on Mobile

The signing process using BankID on Mobile is similar to the signing process of bank stored BankID. However, instead of accessing private keys stored with FOI the challenge is transmitted to the device and the private key is accessed from the sim card. Then the signed challenge is sent through FOI back to the merchant which validates the signature with the CA. This can be seen in Figure 5.5.

## 5.7 Discussion

BankID is providing state of the art encryption and signing services using PKI infrastructure. All the different parts of the infrastructure communicates using the open internet, but with secured connections. The data that is being sent over these secure connections are encrypted using SSL session keys which in turn makes the communication secure. There is however something with the design, that makes BankID differ from a traditional PKI. When using bank stored BankID, the private key is not held directly by the user. The private key is stored in centralized servers, which makes this an artificial form of PKI. The Norwegian government have security guidelines regarding secure communication over the internet. These guidelines divides security into four levels. Level four, which is the highest, demands that there is made use of a PKI. BankID does not directly fulfill these security requirements, because the private key is not directly stored at the customers location, and it was not until 2012 that BankID became an accepted method of authentication while using public services.

The different BankID keys, public key and private key, are stored using very high key sizes. All bank stored private keys using 2048 bit key size and all signing keys and identification keys are using 1024 bit key size. This key size is the NIST recommended size.[12] There is however a weakness to BankID on Mobile, because of the limited key size. BankID are working on getting this key size bigger, but until then this might be a security threat. All certificates issued with BankID are following the X.509 standard.

The BankID central infrastructure handles all signing requests. This makes the central infrastructure a very important component, and if this component should fail, then all BankID services will be unavailable. This makes FOI a great target for potential attackers that would like to harm the banking industry.

## 5.8 Summary

BankID is using technology that is considered to be state of the art. Public key cryptography and PKI is a great solution to make parties that do not already have an existing relationship to trust each other. Using PKI, BankID makes it possible for merchants and clients/customers to trust each other using their bank as a trusted source. BankID does differ from traditional PKI because the two parties do not necessarily share the same bank. The *inter bank agreement* is a solution to this problem, making banks trust each other as well. All keys and certificates follows industry standards and recommendations by both NIST and ISO.

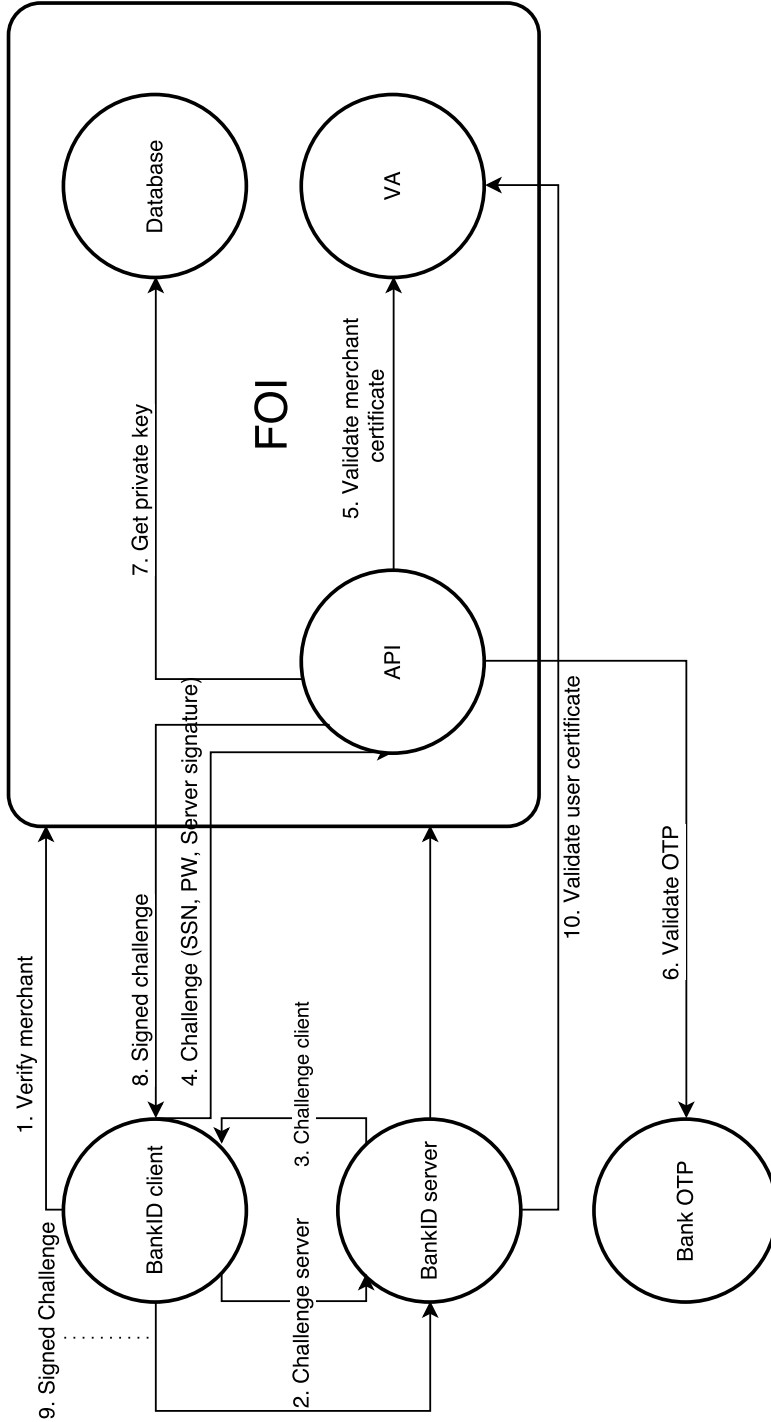


Figure 5.4: BankID Bank Stored Data Flow Diagram

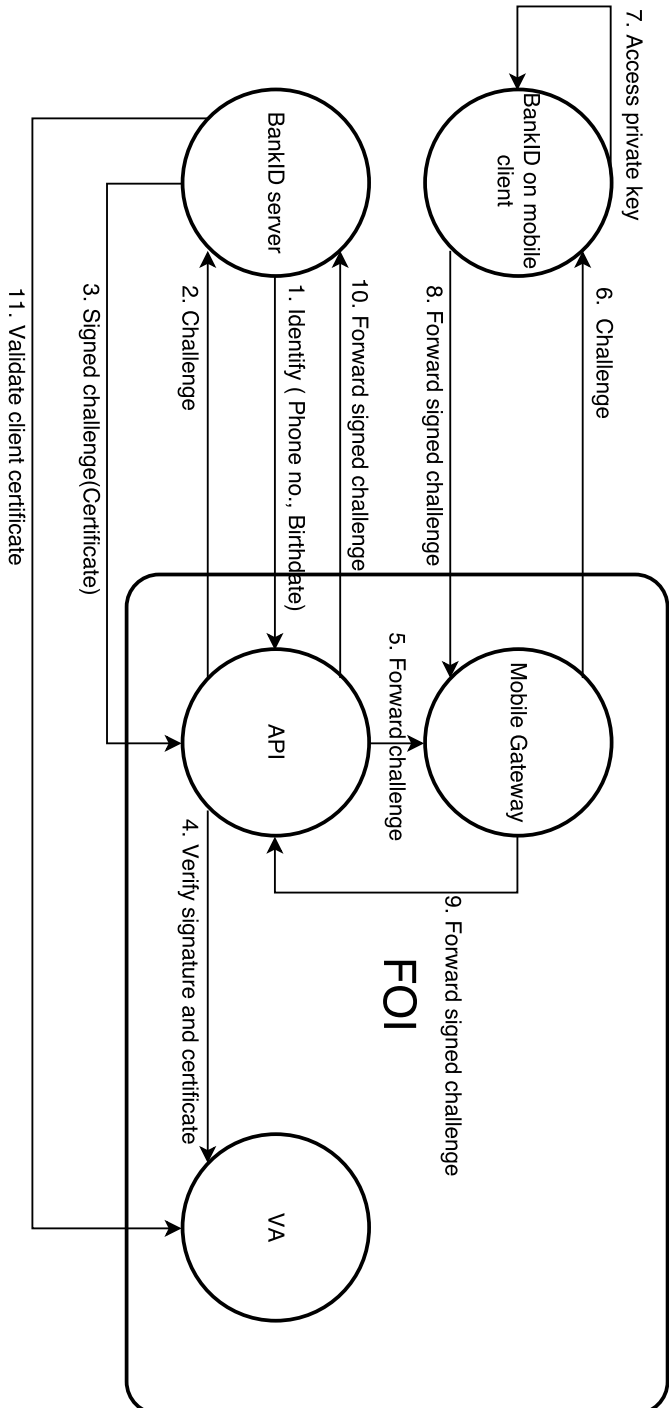


Figure 5.5: BankID on Mobile Data Flow Diagram

## Chapter 6

# Mobile Banking Software

Mobile banking software has grown to be one of the more popular banking services of all time. No banking service has ever had such growth as mobile banking.[28] This makes mobile banking software an interesting piece of software to analyze. In this chapter we give a brief introduction into different mobile banking software available for end users.

### 6.1 Banks

Mobile banking software is usually not standard off-the-shelf software. Most banks have made special software to let customers access their bank accounts and other banking services. There are currently 5 major Norwegian banks that control almost 70 % of the market. These banks are: *Den norske bank* (DNB), *Sparebank 1* (SB1), *Nordea*, *Danske Bank* (DB), and *Skandiabanken*. The market shares for these banks can be seen in Table 6.1.

Bank name	Marketshare
DNB	33 %
SB1	18 %
Nordea	8 %
Skandiabanken	7 %
Danske Bank	3 %
Local savings banks	13 %
Other banks	17 %

Table 6.1: Distributions of banks market share per 2014

As we can see in Table 6.1, 30 % of the market is split between smaller national and international banks. A portion of these are grouped together under the name *Eika Gruppen AS* which will be considered in



this threat analysis. The rest of the banks will not be a part of this threat analysis.

At the time of writing, all of the major banks offers mobile banking software for iOS and Android. *DNB* offers a mobile application that is designed for iPhone, Android Phones and Windows Phone. *Sparebank 1* offers a mobile banking applicaton that is designed for the iPhone and for Android phones, and they support both Windows Phone 7.5 and Windows Phone 8 and newer. *Nordea* offers mobile banking services to customers owning an iPhone or Android Phone but it does not support any versions of the Windows Phone operating system. *Danske Bank* provides its customers with mobile banking software that is designed for iPhone, Android Phone and Windows Phone. It is also providing a specially design application for iPad, Android Tablets and Windows Tablets, and DB is currently the only bank to offer a native application designed for tablets. *Skandiabanken* offers both an applications for iPhone and Android Phones and iPads and Android Tablets, but it does not offer any native applications for Windows Phone users. Skandia-banken does however claim that their online banking services offered through a web browser is compatible with all platforms. (PC, Tablets and Smartphones). A list of all functionality can be seen in Table 6.2. Local savings banks that are affiliated with *Eika Gruppen AS* do support iPhone and Android phones and phones using Windows Phone can access the online bank through the mobile web browser. It is noteworthy that some banks provide applications for Windows Phone 7.5.

Bank name	iOS	Android	Windows	Browser
DNB	Phone	Phone	Phone <sup>1</sup>	Full
SB1	Phone	Phone	Phone <sup>1</sup>	Full <sup>2</sup>
Nordea	Phone	Phone	No <sup>3</sup>	Full
Skandiabanken	Phone	Phone	No <sup>3</sup>	Full
Danske Bank	Phone, Tablet	Phone, Tablet	Phone, Tablet <sup>1</sup>	Full
Eika	Phone	Phone	No <sup>3</sup>	Full

Table 6.2: Mobile banking software functionality for each major bank.

## 6.2 Technology

In this section, we discuss some of the technological aspects of mobile applicaton development. Unfortunately, we will not get many facts

<sup>1</sup>Version 7.5, 8.0 and 8.1.

<sup>2</sup>The normal online banking page

<sup>3</sup>Online banking services are supported through the web browser.

about the development process, technologies used during development or potential weaknesses that the banks are aware of. I made several inquiries to all of the major banks regarding information about their application development process, but no one responded, or wanted to give useful responses to my inquiries. This is unfortunate, but very understandable as such information is highly sensitive. We will make assumptions based on observations from each application and from the few responses obtained from inquiries to customer service.

### 6.2.1 Native Applications vs Web Applications

Every platform supports the development of both native applications and embedded web applications. The latter is more of a web site disguised as a native application running on the device. There is a debate on whether or not to develop web apps, that can be embedded into applications for each platform using a *web view* or if one should create native applications. We will not go in depth in this discussions, but there are some points that we need to discuss. First of all, a native application can access all available APIs on each platform. This could make for both faster, but also more secure applications. However, web applications makes it easier to quickly deploy apps across different platforms. Since a web application is only a native app containing a web view that displays a website, it is very easy to create a simple native app for each platform and make that app display the website.

A web application brings both advantages and disadvantages, due to the fact that security breaches discovered on one platform possibly could be utilized on all platforms, but the discovery of such breach will let developers create one patch that works across all platforms.

There is also a third option, called hybrid applications where parts of the application is made with native tools, and parts of the application is using a web view to display a web application.

### 6.2.2 Mobile Banking Software Technology

All major banks' mobile banking software applications are developed using hybrid applications or web applications. This implicates that all web related threats needs to be considered for these kinds of applications. The different solutions per bank can be seen in Table 6.3.

Even though there are differences in how the different banks chooses to display their mobile banking software, there is no actual difference between using a dedicated web view and displaying the mobile bank in the default mobile browser application.

---

<sup>4</sup>Not confirmed

Bank name	Application Technology
DNB	The application loads a website in the browser
SB1	The application uses a dedicated web view
Nordea	The application uses native code
Danske Bank	The application uses a dedicated web view <sup>4</sup>
Skandiabanken	The application uses a dedicated web view
Eika Gruppen AS	The application uses native code <sup>4</sup>

Table 6.3: Mobile banking software solution per bank

### 6.2.3 Authentication Service

All major banks are utilizing the BankID authentication service, but all banks accept lesser secure authentication services if the customer wishes to only do non-critical operations. There are different authentications services used, but most utilize a form of access control using PIN codes. Critical operations are considered to be when moving money to another customer. Transferring money between accounts within the same customer or getting account balance statements are considered to be non-critical.

## 6.3 Discussion

There are only five (5) major banks in Norway, which in turn control 70 % of the user base. This leaves 30 % of the market divided on the rest of the banks. 71 of these smaller banks, are grouped under the name *Eika Gruppen AS* and they collaborate on giving their customers services like online and mobile banking.[35] The five major banks and Eika have all developed their own mobile banking service to offer their customers. Nationwide, there are approximately 1.9 million users of mobile banking services, which is about half of all online banking users.[15] This is again a testimony to how popular mobile banking services have become in a very short time frame. This means that developers have not have much time to develop these services relative to the traditional online banking service. This might therefore pose as a security threat.

All major banks provides applications to iOS and Android devices, and some of the banks provide an application for Windows Phone. The fact that not all major banks offers an application for the Windows Phone platform is not that alarming, since all of the banks' online banking service is supported by mobile browsers.

The technology used in all applications are web technologies. The reason for this might be to more easily be able to integrate a mobile banking service to existing APIs since all banks already have existing

online banking services for desktops that uses web technology. This does however make all mobile banking applications susceptible to browser-related and similar attacks. This must be considered in the threat model.

All mobile banking services utilize the BankID authentication service, but also provide the possibility of using less secure authentication methods for non-critical operations. Skandiabanken is the only one that does not offer the possibility of authentication through BankID every time you want to log into the banking service. They do however let you authenticate yourself with BankID once and then keeps you authenticated as long as you provide a PIN code. Since all critical operations need to be authenticated with BankID we can deem this feature to be secure.

## **6.4 Summary**

All major banks offers both online and mobile banking services that are compatible with smartphones and tablets. Four of six banks discussed in this chapter disguise their mobile banking applications as native apps that contain a web view with the actually banking service. All major platforms are accepted and all applications utilizes one of the most secure authentication methods available in Norway, BankID.



**Part III**

**Threat Model**



## Chapter 7

# Threat Modelling using STRIDE

In this chapter, we create a threat model using the abstract method STRIDE. To create this threat model, we utilize the STRIDE variant STRIDE-per-element. Every threat is assigned on of the three following levels.

1. **Not applicable** - This threat is either completely mitigated or is not possible due to other factors.
2. **Possibly mitigated** - This threat is either very hard to execute, or not possible to execute, and can be deemed harmless.
3. **Current threat** - This threat is most likely possible to execute.

### 7.1 Data Flow Diagrams

To be able to perform a valid threat model of the different systems, we need to analyze each system into a data flow diagram (DFD). In this section we create DFDs for the platform, for the different parts of BankID and for the application.

#### 7.1.1 Platform

As we can see in Figure 7.1 on 76, there is data flowing everywhere. The kernel is the communication-center between applications, storage mediums, memory and communication components. If the application wants to send data to an external entity then the data need to cross three trust boundaries. The first trust boundary is within the application source code, the next is within local storage media, and the last is



when communication with external entities.<sup>1</sup>

### 7.1.2 BankID

As we can see, the BankID DFD is a little bit more complicated. The reason for this is because we have four different agents. We have components hosted by the bank, hosted by a merchant, running on customer device or hosted by Nets. There is data flowing from all entities to FOI, and we see that there are separate trust boundaries between what is hosted by the bank, merchant, customer or Nets. This is because the data is flowing through open networks. This DFD can be seen on 77.

### 7.1.3 Application

It is difficult to create a data flow diagram to model application behaviour. What we do know is that data is flowing from the application to the hardware components, and that the application might transfer data over communication components. Therefore we can just use the same diagram as we see used for the platform.

## 7.2 Platforms

In chapter 4 we discussed how the different mobile operating systems handles different security measures. We concluded that there is no major difference between each operating system except for the pre-approval process for applications in the official application store on each platform. Microsoft and Apple both have a pre-approval process. Android does not offer such service, but rather an automated service to verify application source code after it has been downloaded. Now, we need to divide mobile operating systems in general and see which parts of the STRIDE mnemonic that applies to each element in the operating system.

### 7.2.1 STRIDE-per-element

Using STRIDE-per-element on a whole system, might be difficult due to the extensiveness of a whole system. There are millions of lines of code to analyze and several different parts to analyze. In section 4.2 we listed 4 major security features that is important for both the system and data integrity. These features were kernel security through secure boot, data security through encryption and data Access, the general security of

---

<sup>1</sup>One could argue that the trust boundary around the kernel/disk/memory is not needed. This boundary separates different storage media and communication components and applications. It is added to make it perfectly clear that data is being made available to a communication protocols or to an application

both native and third-party applications, and the general security of the device itself. In this threat model, we will therefore focus on these four key aspects of the mobile operating system.

Element	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privilege
Kernel security		X				
Data security		X		X	X	
Application security	X	X		X	X	X
Device security		X		X	X	

Table 7.1: STRIDE-per-element table over mobile operating systems

As we can see in Table 7.1, none of the four components activates the *Repudiation*-part of the STRIDE mnemonic. This is done per-choice because none of the three mobile operating systems in questions provide log files that are accessible to the end user. Without any log files, there is no possibility for the end user to confirm or deny any action using the mobile operating system. “Again, if you don’t have logs, don’t retain logs, or can’t analyze logs, repudiation actions are hard to dispute.”[22]

If we compare Table 7.1 to Figure 7.1 we see that the DFD includes Kernel security, Data Security and Application security. Device security is not a part of the DFD, but we will still analyse this security aspect.

## Kernel Security

The kernel is only susceptible to tampering. If the kernel is tampered with, this could then lead to Spoofing, Information Disclosure, Denial of Service and Elevation of Privilege, but these are all secondary threats because of this dependency.

Tampering of the kernel is a threat which will be classified as *Possibly mitigated*. Both Android and iOS uses open source kernels, which have been developed, tested and patched for about a decade each. Windows Phone uses the NT-kernel which have been through the same development process and has been used for over a decade.

The Kernel has data flow between all components, but trust boundaries are only breached when giving data to an application or when sending or receiving data from an external entity through communication components.

### Data security

The data security aspect of the operating system is susceptible to tampering, information disclosure and denial of service. Tampering is most likely not a problem with any of the major mobile operating systems. All systems uses automatic encryption when saving data to storage media. There is one known exception and that is data files stored on a removable storage medium on Android. There is also process specific authorization levels provided by each operating system. Application A cannot access Application B's files due to the fact that they do not share the same user identification number. This makes all operating systems protected from data tampering. The threat status of data tampering is therefore possibly mitigated.

Since each application can only access its own files on disk, there is no imminent threat against information disclosure between application with different authorization levels.<sup>2</sup> The threat status of information disclosure is therefore possibly mitigated.

Denial of service is unfortunately very tricky to analyze. Data corruption, which would be the same as tampering with data, is most likely not a problem with most mobile operating systems. However, there might be ways of destroying keys used to access the data. If the hardware key (iOS) or another similar key is destroyed, then file access is denied. Denial of service is therefore a current threat.

### Application security

Application security is one of the more important aspects on all platforms. Third-party applications are the main source of unauthorized code on a system, and it is therefore very important to have strict application control. Fortunately, this has been one of the primary features in all mobile operating systems. Since all systems run applications in separate sandboxes, the threat of anyone accessing data belonging to another application is virtually non-existent. Application security triggers all aspects of STRIDE, except for repudiation. Spoofing between applications are only a threat if there is inter-process communication present on the system. As far as we know, its only Android that offers inter-process communication by utilizing *intents* or *remote process calls*. This makes information disclosure possible during inter-process communication, but only for Android systems. This threat is therefore possibly mitigated for iOS and Windows Phone, but a current threat on Android.

---

<sup>2</sup>We specify the fact that applications can have different authorization levels, since there would not be any reason to hide data between them if they were authorized to see the same data

Denial of service using applications is most likely only possible by having applications that executes resource depletion by flooding the system with function calls. This can be done by accessing the disk or other components too frequent, and therefore blocking calls needed for normal operation. There is only one system that accepts unvalidated third-party applications, and that is Android. Due to the lack of pre-approval routines before applications are made available on the Android Marketplace, and the fact that Android phones can install applications downloaded from other websites or application stores, this problem will only be able to affect Android phones. However, there is no reason to believe that the process scheduler would allow processes to flood the system. The threat status of denial of service attacks is possibly mitigated.

All applications run with the same privilege level on all systems. This implies that elevation of privilege threats are not applicable. Data tampering and information disclosure has already been analyzed in earlier sections.

### **Device security**

Device security deals with the security of the physical device. The physical device is in most cases protected by a passcode. All systems provides this feature, but is not necessarily activated by default on all devices. As long as the user activates this feature, then the physical device is protected against tampering and immediate information disclosure. Tampering threats are possibly mitigated.

Denial of service would be if someone lost the device, or it gets stolen. This is not something that can be mitigated and is therefore a current threat on all systems.

Information disclosure could happen even if someone have not activated passcode protection mechanisms. The passcode on the system protects files by encrypting data on disk. If the device is stolen, an attacker can perform an offline brute force attack to test every key possible and break open the system. This threat can be mitigated by using an MDM. Using MDM software to delete another key from the system will make it unbreakable during a realistic time frame. It is therefore recommended that all devices connects to an MDM service to mitigate this threat.

## **7.3 BankID**

In this section we try to split the different parts of the BankID authentication service that we read about in chapter 5. BankID is a very complex system composed of many different components. Central infrastructure,

distribute infrastructure, and client side applications or functionality.

### 7.3.1 STRIDE-per-element

BankID can be divided into four different parts. Central infrastructure, Bank provided distributed infrastructure, Merchant provided distributed infrastructure, and client side infrastructure. Bank provided distributed infrastructure can further be divided into Bank RA and Bank provided OTP mechanism. The Bank RA is hosted by the bank, but the Bank OTP is given to the customer. Central infrastructure is what we referred to as FOI in section 7.3.1. FOI contains all systems hosted by Nets and is where the customers private keys are stored. There are three types of distributed infrastructure. Bank provided infrastructure is the RA and the OTP mechanism. Merchant provided infrastructure is the BankID server that communicates with FOI to authenticate an end users certificate. The last type of distributed infrastructure are the BankID clients. There are two types of clients: BankID client and BankID on mobile client. We can see these different parts in the DFD in Figure 7.2.

Element	Spoofing	Tampering	Reputation	Information Disclosure	Denial of Service	Elevation of Privilege
FOI <sup>3</sup>	X	X	X	X	X	X
Bank RA	X	X	X	X	X	X
Bank OTP		X			X	
BankID Server	X	X	X	X	X	X
BankID clients <sup>4</sup>	X	X	X	X	X	X

Table 7.2: STRIDE-per-element table over BankID

As we can see from Table 7.2, using STRIDE-per-element does not really help us to eliminate many types of threats. FOI is the term used for many different services as we can see in Figure 5.2. FOI consists of data flows, data stores, processes, and interactors. Bank provided services utilizes data flows, processes and interactors. BankID server is a process, and BankID clients are processes. This means that every key aspect of the BankID infrastructure is susceptible for all kinds of attack suggested by the STRIDE mnemonic.

<sup>3</sup>FOI consists of many services

<sup>4</sup>BankID client and BankID on mobile client

## FOI

To be able to analyze FOI we need to break FOI into its key components. It is composed of: Data servers that store keys, certificates, CAs and more. API interfaces for BankID clients, BankID servers and Bank RAs/OTPs to communicate with, and processes that provides functionality to APIs. This means that we have both data stores, data flows and processes that operates within this single entity. All of these are susceptible to different attacks that needs to be mitigated.

Data stores are susceptible to all types of threats that we can see in Table 3.2. Data stores might be tampered with to corrupt data, data might be disclosed to unauthorized entities or the data might be made unreadable which would deny service to authorized entities. We can presume that the data centers hosted by Nets have sufficient physical protection layers and that they have implemented redundancy for all stored files. These types of threat are therefore possibly mitigated.<sup>5</sup>

Data flows are vulnerable to the same type of threats as data stores. Tampering of data flows should not be possible due to the fact that all inbound and outbound data flows are encrypted using SSL. Regarding the fact that no encryption is totally safe we scale this threat to be possibly mitigated.

Processes are the hardest part to analyze. There is no documentation that tells us what types of processes that are running, which privilege levels they are running with and so on. It is therefore not possible to give a detailed list of threats, but still we need to address some threats. We need to assume that the whole central infrastructure is properly physically guarded. As we talked about with data stores, we presume that the data center has sufficient access control mechanisms, locks, and other protection layers needed. We also need to presume that they have strong firewalls implemented to stop unauthorized communication from external entities. The last, but also the most important part, will be how the servers are programmed. Assuming they are using normal server operating systems, we need to make sure that processes with different authorization does not share information through leakages, that they do not tamper with files by accident, that they do not share information between processes, that they do not exhaust the system denying other processes access to system resources, and that they do not provide access to privileged functionality to unauthorized entities. There is not sufficient information to determine the likelihood of these threats being mitigated.

There are no unknown interactors that communicates with FOI. All communication with FOI comes from either the Bank RA, BankID

---

<sup>5</sup>But these kinds of threats are almost impossible to completely mitigate as a strong enough bomb or other extreme measure still would be able to knock out a system.

clients or BankID servers.[9] This type of threat is therefore not applicable.

### **Bank RA**

The Bank *Registration Authority* is the system used by each individual bank to register new customers and request certificates to this customer. This functionality is usually integrated in the customer administration system used by the bank.[9] This means that it is up to each bank to ensure the security of its RA from all possible threats. We can assume that the security is adequate and that these threats are possibly mitigated.

### **Bank OTP**

Each bank chooses its own one time password generation method, as long as it is accepted by BSK. The threat of tampering with these devices/mechanisms are possibly mitigated, but it is in most cases not protected from theft or misplacement. Denial of service is therefore a current threat.

### **BankID Server**

The BankID server handles all necessary communications between the merchant, BankID central infrastructure and the customer, to be able to perform a transaction. The server uses data flows and processes. All data flow between the server and BankID central infrastructure is encrypted. This is done using PKI and we can assume that the threat of tampering with data, and information disclosure is possibly mitigated. Denial of service is however a threat if an attacker manages to re-route data to a fake host, or by depleteing the server of resources. The threat of denial of service is possibly mitigated.

The server runs on a process on the BankID server computer. This makes all threats from the process row in Table 3.2 applicable. Other processes on the server may try to spoof the server process, tamper with server process data, disclose information, deny the process access to system resources or gain privelege. The process security is dependent on the server which it is installed on, and since there are no guidelines on how to secure the server itself, we must consider this a current threat.

### **BankID Client**

The BankID client, or the BankID on Mobile client handles all necessary communication from the customer to BankID central infrastructure and from the customer to the merchant. The client uses data flows and a process on the client computer. All data flow between the client

and central infrastructure is encrypted. This is done using PKI and we can assume that the threat of tampering of data is possibly mitigated.

The BankID client runs on a process on the customers computer which in most cases are open to install third-party applications from any source. This makes customer computers a great target for attacking BankID, since there are a wide range of operating systems and lots of malware attacking these systems. We must therefore consider this as a current threat.

## 7.4 Application

In this section we try to analyze the different components of mobile banking applications of the six major bank groups. There are three types of applications used by these banks. DNB, which has the largest market share, uses a browser level application which runs through the mobile web browser on the system.<sup>6</sup> Sparebank 1, Skandiabanken and Danske Bank uses a hybrid application. Some critical elements run in native code, but some runs as a web application inside a web view in the application. The applications used by Eika and Nordea are written using native code. Eika provide mobile banking applications for iOS and Android, but supports Windows Phone through the normal online banking website. Nordea also provide applications for iOS and Android, but not Windows Phone. Windows users must access their accounts through the web browser. Since there are no major differences between using browser applications, embedded web applications or native applications, we do not differentiate between them.

### 7.4.1 STRIDE-per-element

We divide the mobile banking application into the same four parts used in Table 3.2. We use the DFD on 76, but we focus on the application communication with the device/kernel.

#### Data flow

Data flow from the application and to the system kernel could become offer for tampering, disclosure and denial of service. The kernel does protect application data while residing on disk and in memory, but there is no documentation for how it protects data while reading and writing. With that being said, it is not likely that any application will be able

---

<sup>6</sup>DNB has an app available for all platforms, but when clicking the link to start the mobile bank the user gets transferred to the mobile web browser.



Element	Spoofing	Tampering	Reputation	Information Disclosure	Denial of Service	Elevation of Privilege
Data flow		X		X	X	
Data store		X		X	X	
Process	X	X		X	X	X
Interactors	X		X			

Table 7.3: STRIDE-per-element table over mobile banking applications

to intercept the data while it is being transferred and this threat is therefore possibly mitigated.

### Data store

Storing of data is handled by the platform. Since all data is encrypted on disk, applications are limited to using their own private memory space and disk space, and since physical memory is protected, we can assume that most threats are mitigated. For more information, see section 7.2.1.

### Process

Process management is handled by the platform. It is possible to install malicious applications on Android that could potentially be a threat to mobile banking applications running on the device. For more information, see section 7.2.1.

### Interactors

An interactor is the entity which is using the application. This can potentially be an unauthorized entity which has gained access to the device and is using the application. This can only be mitigated using physical security and passcodes and this threat is therefore possibly mitigated. See section 7.2.1 for more about device security.

## 7.5 Discussion

STRIDE as a tool for threat modelling, is not the best tool for beginners. The threat model that was produced in this chapter using STRIDE-per-element is far from a complete threat model. We have barely scraped

the top of the ice-berg of possible attacks on these three system components. The main weakness here would be time and experience. While using STRIDE-per-element, it seems that one would need an extreme amount of time to be able to create an adequate threat model. When using STRIDE, experience would be a key factor. An experienced threat modeller should be able to add possible threats to the threat model, due to his knowledge about the subject. This however, implies that the experienced threat modeller uses his own *attack library* that resides in his own brain. After using STRIDE-per-element on three different systems, I would recommend using this tool as a group exercise to try and discover new threats that an experienced threat modeller has not already discovered. It is also very important to mention that this threat model was done without the possibility to test any attacks to see if they pose a threat to one of the systems. All threats are based on information that is publicly available through documentation from every platform, from BankID and from simple application testing. We also had to create DFDs from the information provided by documentation, there is no guarantee that these DFDs are 100 % correct since much documentation is missing or not publicly available.<sup>7</sup> STRIDE could possibly help to find more threats if we had a test environment to test all the parts of the different systems.

## 7.6 Summary

Using STRIDE to discover threats is not very efficient if the threat modeller is a novice to the art of threat modelling. In this chapter we created simple data flow diagrams and used STRIDE-per-element on these diagrams to see if we could discover any threats. By using STRIDE we did not manage to produce many relevant threats, and a threat model done purely by DFDs and STRIDE will either take a very long time to complete, or not be satisfactory.

---

<sup>7</sup>There are obviously no data flow model that is 100 % accurate. Therefore, it is never a guarantee that the DFD will be relevant.

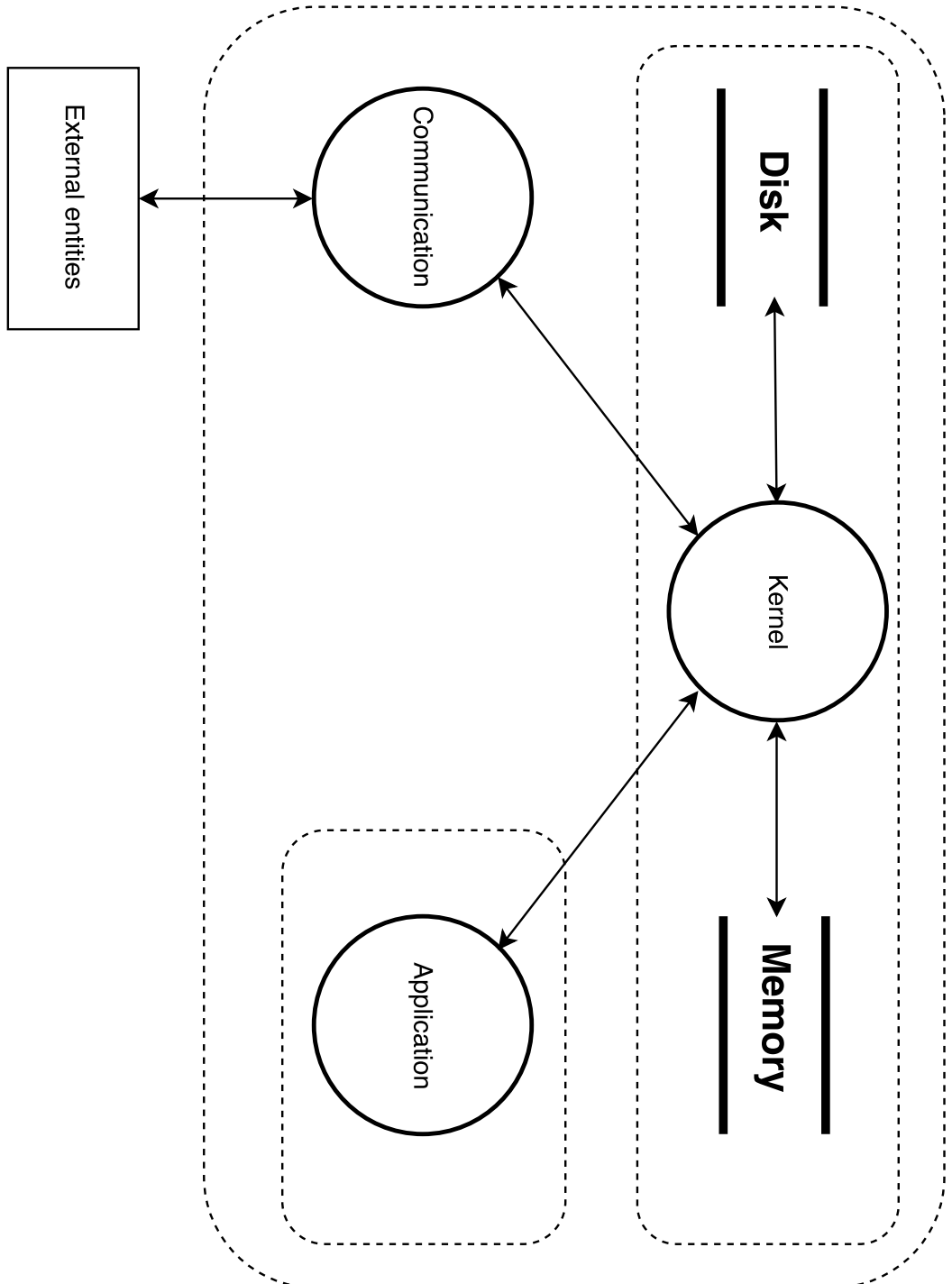


Figure 7.1: Platform data flow diagram

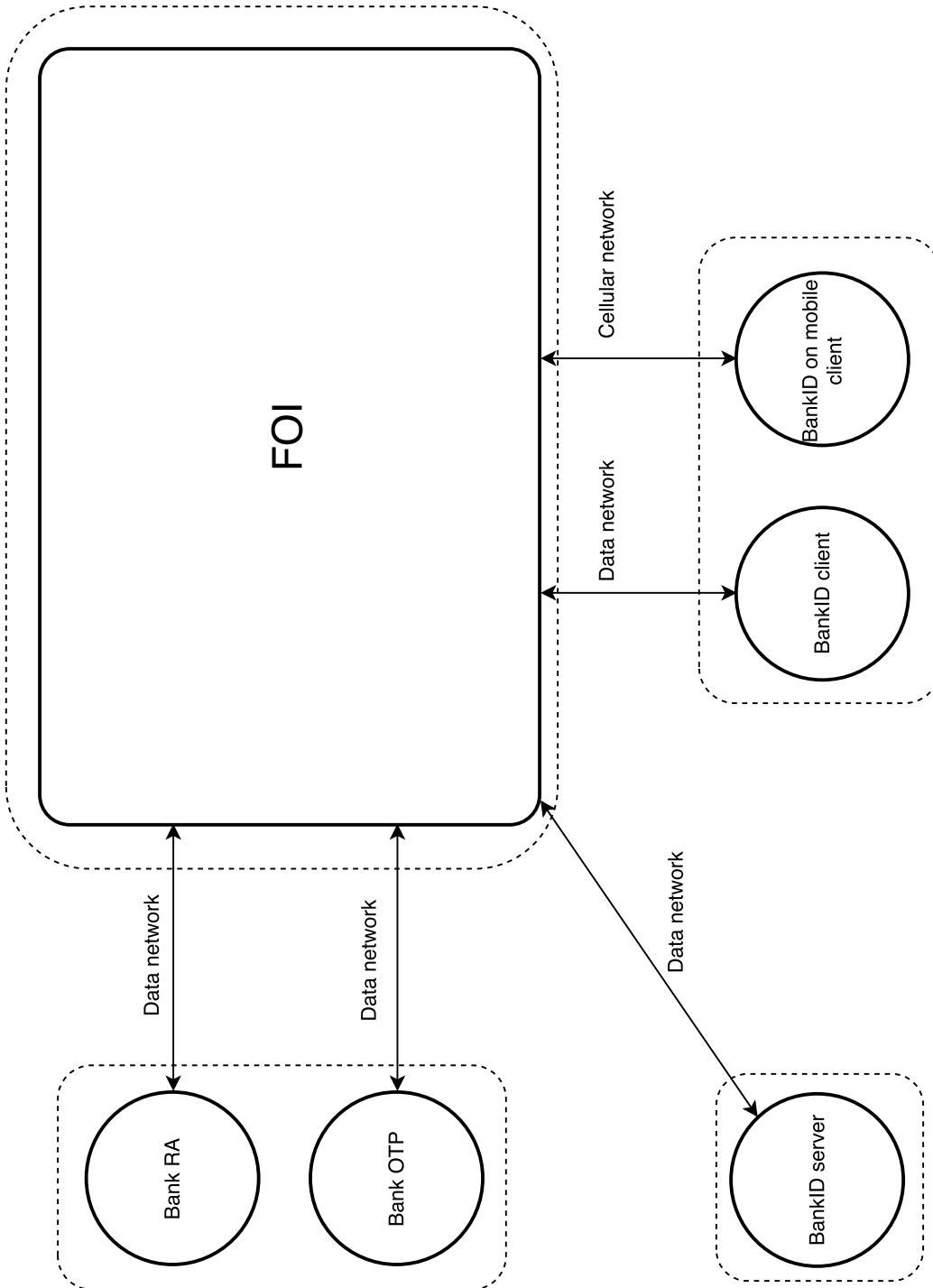


Figure 7.2: BankID data flow diagram



## Chapter 8

# Threat Modelling using CAPEC

In this chapter, we create a threat model using the detailed attack library CAPEC. CAPEC is an attack library that is very close to the limit of practicality, and in this chapter we consider many attack patterns from this library. As we can see in Appendix A, we have processed 169 different attack patterns. In this chapter, we discuss why we chose these patterns, how we scored them and we present an analysis of our findings.

### 8.1 Attack Patterns

We chose to use the CAPEC view that was sorted by mechanism of attack. This is the same view as described in section 3.2.1 and it contains 16 different categories. We included all of the categories in this threat model, but I have omitted some of the attack patterns, due to the level of detail in each pattern.<sup>1</sup> In most cases we included the category and the following attack patterns. This includes all meta, standard and detailed attack patterns. All first generation meta attack patterns are included, but some meta attack patterns have children that was omitted. I have also decided to not include children from a standard attack patterns. Children of standard attack patterns are more detailed versions of the standard attack pattern and there is in very few cases possible for us to decide with certainty if an attack pattern is mitigated or not. Some children of meta attacks were omitted due to the same reasoning as children of standard attack patterns. An example here is *CAPEC-416: Target Influence via Social Engineering* which has 9 children which are

---

<sup>1</sup>Many attack patterns are very similar to one another. In those cases, I have analyzed the *meta attack pattern* which is the parent of these similar attacks.

different ways of persuading the victim. Two examples of children who are very similar are *CAPEC-423: Target Influence via Perception of Liking* and *CAPEC-422: Target Influence via Perception of Commitment and Consistency*.

## 8.2 Scoring

We use the same scoring system as in chapter 7.

1. **Not applicable** - This threat is either completely mitigated or is not possible due to other factors.
2. **Possibly mitigated** - This threat is either very hard to execute, or not possible to execute, and can be deemed harmless.
3. **Current threat** - This threat is most likely possible to execute.

All scoring is done using the knowledge we have from chapters in this thesis and from additional documentation and white papers on these different subjects. We used this scoring system to see if we believe that it is possible to mitigate a threat or if the threat is un-mitigable.

One problem with this kind of general threat analysis is getting help from people in the business. After contacting several of the major banks, only one wanted to give a short statement about their development security details. This is no surprise, as we would assume that information about development processes and security details are company secrets. This means that we have no information that is not already made publicly available and these public documents are the foundation that the resulting threat model is built on.

## 8.3 Current Threats

In this section, we analyze every attack pattern from Appendix A and see why we need to consider that attack pattern as a threat that is still not completely mitigated or not mitigable at all.

### 8.3.1 JSON Hijacking (aka JavaScript Hijacking)

JSON hijacking or JavaScript hijacking is when an attacker utilizes weaknesses or flaws in the parsing of JSON requests given to a server. Systems that uses JavaScript Object Notation (JSON) as a transport mechanism between the target server and a client are susceptible to this attack. Since BankID utilizes JSON objects to transfer certain types of

data between a client/server and the central infrastructure, there is a potential threat in attackers using these kinds of attacks.

Since there is no available documentation for how the BankID system parses JSON objects, we need to consider this as a current threat that will need mitigation.

Possible mitigations include making sure that the server actually can differentiate between legitimate JSON requests and being able to deny forged requests, and making the URLs that the system uses to accept JSON request dynamic so that they are unique for each user session.

### 8.3.2 Probe Application Screenshots

The iOS operating system uses a graphic task manager. This task manager, also called *multitask switcher*, is using screenshots of all running applications to aid the user when switching between applications. These screenshots are stored on the user device and the attacker might either gain access to these screenshots by jailbraking the unit, or by physically accessing the phone and using the multitask switcher. In both scenarios, the attacker need physical access to the device. By utilizing these screenshots, an attacker might get unauthorized access to valuable information.

This attack is a threat both to the mobile operating system (iOS) and the mobile banking application that runs on the system. The operating system should not store these screenshots over a longer period. There are possibilities to mitigate this threat, but most of the major banks have not mitigated this threat.

Possible mitigations include hiding elements that display sensitive data upon changing the active application. The operating system provide two methods `applicationWillResignActive()` and `applicationDidEnterBackground()`, and developers should hide sensitive information in these two.

### 8.3.3 Probe Application Error Reporting

All platforms, BankID and applications can provide error logging services. An attacker might try to access and utilize the fact that applications are reporting errors by repeatedly probing the system and analyzing how the system reacts to errors.

This attack might be a threat to the application. If the application saves log files on the device and the log information is in plain text, then an attacker might use these log files to analyze the system.

This attack can be mitigated by creating coded log files. A coded log file will never display an error in plain text, but rather use an error code



system to protect information. This will however only mitigate threats as long as the *code book* is kept a secret. Other mitigation techniques will be to not log error files on the device, but rather send all errors (preferably encrypted error codes) to the server.

### 8.3.4 Probe Application Queries

This attack is feasible if an attacker has authorization to query an application. By sending modified but authorized request to the system, an attacker might trick the system into returning more information than it is supposed to do.

Both BankID and mobile banking applications are available to the public and this makes this attack a possible threat for both the BankID authentication system and the application.

This attack has no recommended mitigation method from CAPEC, but making sure that servers will not accept modified requests or requests that does not comply with the expected format is recommended.

### 8.3.5 Malware-Directed Internal Reconnaissance

This attack is executed by installing malware on a computer within the organizational perimeter where the attacker want to perform reconnaissance. By having an application on the inside of the system, the attacker can bypass most security measures (firewalls) and makes it harder to detect the attack.

This attack affects all systems that use servers in this threat model. Both BankID and the mobile banking application have server side functionality that is used to communicate with the end user. By installing malware inside the FOI perimeter or at the bank's server center, an attacker might gain valuable information about configuration, composition, and security mechanisms.

This attack has no recommended mitigation method from CAPEC. These types of attacks are very hard to mitigate. All machines that connects to the same network as the application servers, should have limited capabilities.

### 8.3.6 OS Fingerprinting

An attacker might try to gain valuable information about the operating system (OS) running on system servers, by performing fingerprinting attacks. By getting knowledge about the OS that is running on system servers, an attacker can plan more detailed attacks based on the OS version on the system.

This kind of attack affects all systems in this threat model. An attacker may look for devices with older mobile operating systems to try and utilize these devices. This is especially a problem with devices using the Android operating system. In Table 4.1, we see that about 50 % of all Android devices currently in use, run Android version 4.3 or older. These versions are not powered by Security Enhanced Linux and lack other security features provided by newer versions. This makes them more susceptible to some attack patterns. OS fingerprinting may also be done on BankID servers and FOI, and on the mobile banking application servers. By using the knowledge of which operating systems the servers are running, the adversary might find weaknesses and leverage them to gain unauthorized access.

This attack has no recommended mitigation method from CAPEC, and it is very difficult to mask which OS a system is running. Mitigation can be temporarily achieved by keeping all systems patched and updated with the latest security releases.

### **8.3.7 Application Fingerprinting**

An attacker might try to gain valuable information about the application version running on a system. This can be done by performing fingerprinting activities. By getting knowledge about application version installed on a target server the attacker gains valuable knowledge before planning more detailed attacks against the system.

This kind of attack affects BankID systems and mobile banking applications. Even though all mobile operating systems have built in software update mechanisms, not all users utilizes this functionality. If an application gets released with a security flaw and the attacker gains knowledge about this flaw, he can then execute this attack on devices that run this version of the application. This also applies to BankID releases. If either a BankID server, BankID client or a version of the FOI infrastructure is flawed, an attacker might want to get a fingerprint from merchants or clients, or try to fingerprint which version FOI utilizes.

This attack has no recommended mitigation method from CAPEC, but it might be recommended to enforce customers to use the application version with the latest security updates and forcing merchants and FOI to keep BankID servers and BankID clients up-to-date. By doing so, this threat might possible be mitigated.

### **8.3.8 Social Information Gathering via Research**

Research is a very powerful tool. An attacker might try and gather valuable information by employing various methods of research. There exist

no particular way for an attacker to do this kind of research, because the research method is dependant on what the attacker is trying to achieve.

This kind of attack might affect all systems. Since there is no limit to how much time and resources an attacker might use on researching a system, there is no guarantee that any system is safe. This includes all system parts that have public documentation or where the attacker might gain access to system software so that they can create local test environments.

This attack has no recommended mitigation method from CAPEC. However, by making sure to change the system architecture, communication protocols or other parts of the system regularly, the manpower (cost) needed to perform research within the limited time window will be greater than the profits.

### **8.3.9 HTTP DoS**

An attacker performs flooding using HTTP requests on a target server. This is done to bring down a specified web application instead of whole systems. This type of denial of service attack requires less traffic and is therefore harder to detect.

This kind of attack might affect BankID servers/FOI, and it might affect mobile banking applicatons that uses a web view. Since this attack is using the HTTP protocol, all mobile banking applications that uses HTTP request to get information from web servers are possibly vulnerable to these kind of attacks.

This attack may be partially or completely mitigated by configuring web servers to limit the waiting period on HTTP sessions, and by implementing load balancing systems. Load balancing systems does not completely mitigate an infinitely large HTTP DoS attack, but it might mitigate smaller attacks.

### **8.3.10 Checksum Spoofing**

Checksum spoofing is when an attacker spoofs a checksum message sent to a client or a server with the purpose of making a payload seem to be valid. These kinds of checksums are used to verify the integrity of a message between the client and the server. When the attacker intercepts a message, he changes the message body and also changes the corresponding checksum to match that of the message body. By doing this the attacker spoofs the system into believing that the message is valid.

This kind of attack could affect BankID and the mobile banking application. BankID messages are encrypted and protected by PKI technology, we can assume that this threat is mitigated for this system.

However, we have no information on how data from the different mobile banking applications are transmitted between the host bank and the customer. This makes checksum spoofing a current threat for mobile banking applications.

This attack has no recommended mitigation method from CAPEC. This threat might be mitigated by encrypting all data that should be transmitted using encryption technologies like PKI. If the application utilizes HTTPS, then this threat might be mitigated.

### **8.3.11 Intent Spoof**

An attacker may use a previously installed malicious application, issue intents that are directed at a target application's trusted component. The goal of this attack is to achieve different objectives such as modification of data, data injection or information disclosure. Intent spoofing is only an issue on Android device and this makes iOS and Windows Phone users safe from this attack.

This kind of attack will only affect platforms running Android, and it will only affect mobile banking applications that are listening to untrusted intents. Since we do not know if the different mobile banking applications utilizes intents, we need to list this as a current threat.

This attack may be mitigated by avoiding to export components unless they are meant to handle intent requests from untrusted applications. Developers should limit application component exposure to partially or completely mitigate this threat.

### **8.3.12 Principal Spoof**

An adversary might pretend to be some other person than the one he truly is. By pretending to be some other person in an interaction, the attacker can try to trick the threat victim into performing a task or disclosing valuable information to the attacker. These kind of attacks are often used as a part of phishing and pharming attacks. A principal spoof does not rely on falsified, spoofed or stolen authentication credentials, but rather letting the appearance and content of the message make the illusion of authenticity. The attacker is hoping that the message looks so real that the victim gets tricked into disclosing information or perform tasks on behalf of the attacker.

This kind of attack has the possibility of affecting all systems. Phishing attacks are common and both banks, and BankID are susceptible and vulnerable to such attacks. As an example, many BankID users received phishing emails that tried to phish information from customers in June, 2015.[8] This attack may also be targeted against customers from one specific bank.

This attack has no recommended mitigation method from CAPEC. Even though banks, BankID and other institutions have advertised and done public service announcements about never revealing sensitive information over email unless they have contacted the bank directly themselves, this still happens. As long as the user base is still susceptible to believing that they need to provide sensitive information when asked, then these attacks cannot be completely mitigated. By keep doing public service announcements about this subject, this attack pattern might be partially, but never completely mitigated.

### 8.3.13 Signature Spoof

By generating messages or data that fool the recipient to believe that the data block or message was generated and signed cryptographically by a reputable source, the attacker might trick the victim (person, application or system) into executing malicious actions.

This kind of attack can potentially affect both BankID and the mobile banking application. Both of these systems rely on transporting data between a client and a server and if an attacker is able to create signatures that appear to be valid, it will put the whole system in danger.

This attack has no recommended mitigation method from CAPEC, but it is likely that using a PKI with large enough keys should be sufficient to partially mitigate this threat. Since BankID uses very large keys, we may assume that they are sufficiently protected, but this attack pattern is still a potential threat. We do not know how the different mobile banking applications protect data during data transmission, but we must assume that they use some sort of cryptography that is similar in strength as BankID.

### 8.3.14 Pharming

A pharming attack is when a targeted victim is tricked into entering sensitive data into a site that the user believes to be trusted. This is normally done by copying the layout and design from a website, then sending emails with a link to the fake website where the user is instructed to enter sensitive data.

These kind of attacks are a subset of the principal spoof attack pattern. Both the BankID service and the mobile banking application are susceptible to these kind of attacks.

The suggested mitigation technique is to make sure that all sensitive data is transmitted over a secure connection. *End users must ensure that they provide sensitive information only to websites that they trust, over a secure connection with a valid certificate issued by a well-known*

*certificate authority*. [10] A big problem with this mitigation technique is that knowledge of security is not widespread and a typical end user does not know what a *valid certificate issued by a well-known certificate authority* is.

### 8.3.15 Phishing

A phishing attack is when a targeted victim is tricked into revealing confidential information to an attacker who masquerades as a legitimate entity. This is normally done by masquerading as customer service representatives or other authoritative entity from a business or institution that the attacker wants to access.

These kind of attacks are a subset of the principal spoof attack pattern. Both the BankID service and mobile banking applications are susceptible to these kind of attacks.

The suggested mitigation technique is to make sure that the end user knows that it is not supposed to follow any links you receive in emails, give away sensitive information to anyone who calls, or in general provide information to an entity that is initiating contact without getting their details verified. The problem with this mitigation technique is that this is purely a public service announcement problem. There is no way of getting such information to every customer. However, by doing public service announcements one can partially mitigate this problem, but it is not possible to do a complete mitigation.

### 8.3.16 Clickjacking

A clickjacking attack is when an attacker hijacks a users web session by overlaying buttons over the user interface that the user want to interact with. If a user loads the mobile banking application through a web view, and if the attacker manages to put buttons overlaying the buttons on the website, he can then be able to gather sensitive information by tricking the victim into interacting with his malicious site, while the user believes he is visiting a safe site.

These kind of attacks can potentially affect all applications and systems that allow JavaScript, flash or other similar browser extensions. BankID (2.0 and newer) requires the user to have JavaScript activated to be able to authenticate himself, and most mobile banking applications use web views to display a website containing the banking service. This makes both these systems susceptible to this attack pattern.

The suggested mitigation technique consists of deactivating JavaScript, Flash and CSS. It is also recommended that the end user does not use the same browser to navigate to unfamiliar sites while using a sensitive service. This mitigation technique is not possible due to the fact

that JavaScript is required to use BankID.

### 8.3.17 Manipulating User State

An attacker may modify user state information that is maintained by the target site or system. The attacker might be able to access previously inaccessible parts of the site or system by granting himself authorization. Typical state information maintained by a site or application can include names, payment information, history etc.

This attack pattern can target all services that maintains user state information. BankID is an authentication service that maintain state information about which user is logged in. The same happens with most mobile banking applications, since they are websites represented by the mobile web browser or embedded in an application.

The suggested mitigation technique is to never rely on state information that is stored in user-controllable locations. Since BankID relies usage of certificates, we can assume that this threat is potentially mitigated. However, we have no information about how mobile banking web applications stores session information and we need to consider this as a current threat.

### 8.3.18 Inducing Account Lockout

An attacker may try to block legitimate users from accessing their accounts by locking accounts. This denial of service attack may be executed by repeatedly trying to access the account using a wrong password. The interesting thing about these kinds of attacks are that most websites and web services will have account lockout activated as a security feature to make sure that attackers cannot execute attack pattern *CAPEC-49*.<sup>2</sup> The very mitigation technique vs *CAPEC-49* is one of the prerequisites of this attack pattern.

This attack pattern can be executed on mobile operating systems and BankID. Every mobile operating system has restrictions on how often one can try to guess the password, which makes locking a user out from its own phone very easy if the attacker has physical access. The same attack is possible on BankID since it is the authentication system used to access the mobile banking application.

The suggested solutions from CAPEC is actually not applicable in this scenario. CAPEC suggests that developers should take the IP address of the entity trying to authenticate itself into account in addition to the log in name/user name. The problem with this is that the IP address of cellular devices changes often. However, a possible partial

---

<sup>2</sup>CAPEC-49: Password Brute Forcing

mitigation technique for BankID would be to include the device id, but this would not solve this problem entirely. If an attacker then gains physical access of a device that is white listed, then the attacker can try an unlimited amount of passwords until he gains access. Account lockout on the mobile operating system is also a problem. There is no suggested mitigation technique for physical devices.

### **8.3.19 Authentication Abuse**

An attacker may gain unauthorized access to an application, device or a system through knowledge of a weakness in the authentication mechanism or by exploiting flaws in the authentication implementation. This attack is executed by tricking the application, device or system to give access by a carefully controlled sequence of events. This attack allows the attacker to be certified as a valid user through illegitimate means.

This attack pattern can be executed on mobile operating systems and BankID. The documentation about the implementation process for BankID authenticated user sessions, is not sufficient. Therefore, we need to consider this as a threat to BankID.

This attack has no recommended mitigation method from CAPEC. Even though we should trust the BankID authentication system, we still need to consider authentication abuse as a possible attack pattern.

### **8.3.20 Create Malicious Client**

An attacker creates an application which is made to interface with a target service. This client is created to violate assumptions that the service makes about clients. For example, servers may make assumptions that clients will send correct messages etc. By creating a malicious client, the adversary might be able to utilize these assumption into making the service perform actions on behalf on the unauthorized user.

This attack pattern can be executed in two different ways. It can either be done by creating a malicious application that the attacker installs on his device using developer tools, so he can try an execute the attack pattern by himself. The other way to execute the attack is by creating a malicious application and tricking legitimate users to perform it. The latter is only possible on Android devices, since iOS and Windows Phone do not allow end users to install applications from other sources than their official application store.

This attack pattern has no recommended mitigation method from CAPEC. This threat is currently partially mitigated due to the fact that iOS and Windows Phone have control over the distribution process of



applications. Android has implemented an application verification system that is supposed to verify source code, but this system does not mitigate this threat.

### **8.3.21 Man-in-the-Middle Attack**

An attacker targets communication between a client and a server by placing a component in the communication channel. This component intercepts all communication between the client and the server and this gives the attacker the opportunity to both read and alter data before it is sent to the real recipients without leaving any traces of data being read or altered. A man-in-the-middle attack gives an attacker the possibility of corrupting data, disclosing information and maybe even gaining elevated privilege.

This kind of attack can be applied to BankID and the mobile banking application. An attacker might try to intercept data sent between the BankID client and the server, intercept authentication challenges or intercept data that is sent to and from the mobile banking application.

This attack needs a special mention, because this pattern has actually been proven to work. In 2007, the NoWires research group at the University of Bergen, performed a man-in-the-middle attack against BankID using a controlled environment. The conclusion was that BankID was vulnerable to man-in-the-middle attacks that enabled a potential attacker to hijack a user session.[2] Since then, BankID has mitigated this exact threat so the man-in-the-middle attack performed by NoWires is no longer possible to perform.

The suggested solutions from CAPEC is to make sure to get public keys signed by a CA, to encrypt communication between the client and server, applications to use strong mutual authentication and to exchange public keys using a secure channel. BankID implements all these things, but they were still susceptible to a man-in-the-middle attack. The same attack is no longer possible, but the threat of other man-in-the-middle attacks is still present. We do not have any information about the security of the communication between the mobile banking application and the banking service. We need to assume that there might exist flaws in the security that can be utilized by an attacker.

### **8.3.22 Abuse of Transaction Data Structure**

An attacker may be able to abuse the transaction data structure used when two components communicate. By abusing weaknesses in the parsing logic, and attacker can manipulate the data structure and make applications behave in an unexpected manner.

This kind of attack can be applied to both BankID and the mobile banking application. An attacker might change the transaction data between the BankID client and a server/FOI. The attacker might also try to change the transaction data between the mobile banking application and the back end servers.

This attack pattern has no recommended mitigation method from CAPEC. Both BankID and mobile banking applications should encrypt all transaction data, and make sure that all data is signed. We do know that BankID signs data, so the threat is possibly mitigated, but we do not know whether or not the mobile banking application does the same and we need to consider this as a current threat.

### **8.3.23 Contaminate Resource**

An attacker may contaminate the resource information system or device by causing them to handle sensitive information in a malicious way. By doing so, the attacker may gain access to information that he is not authorized to see.

This kind of attack can be applied to all systems. An attacker may contaminate the resource of the mobile operating system, either by making sure that the OS image file is contaminated. The attacker can contaminate all the different parts of the BankID infrastructure, or it can contaminate the mobile banking application.

This attack pattern has no recommended mitigation method from CAPEC. This attack is not easily mitigated, because the source of contamination may be a disgruntled employee with malicious intentions, it may be malware that an unaware employee might have installed by a mistake, or an attacker may break into secure areas to perform the attack. With so many different ways of executing an attack, complete mitigation might not be possible, and this is therefore a current threat to all systems.

### **8.3.24 Infrastructure Manipulation**

An attacker may exploit the infrastructure used by a network entity in order to manipulate data. The attacker will do so by gathering network data and either read or modify data in order to control the information flow between system components.

This kind of attack can be applied to BankID and the mobile banking application. BankID consists of two major parts, central infrastructure (FOI) and distributed infrastructure. The latter is all system components that are located outside of the security of Nets' data centers, and these components communicate using the open Internet. All traffic between the central infrastructure and the distributed parts, is encrypted

and should be considered to be reasonably safe. However, we do not know how data flows between the different components of the central infrastructure. Since we also lack information about how data between the mobile banking application and the back end servers is protected, then this attack pattern might be possible to execute on the application as well.

This attack pattern has no recommended mitigation method from CAPEC. All communication should be encrypted, even when it is not using the open Internet. Even if data is only passing through local networks it should be secured in case of anyone listening in on the communication.

### 8.3.25 Protocol Reverse Engineering

An attacker may try to analyze and decode information about the protocol used by network applications. By deciphering the protocol, an attacker may be able to gain an advantage and to access unauthorized information.

This type of attack can be applied to both BankID and mobile banking applications. Most mobile banking applications uses HTTP protocol to transmit data due to the fact that they are using websites to display the banking service. There are however applications that run using native code.<sup>3</sup> These applications might use other protocols that also can be reverse engineered. BankID transmits all data between the client and the server, and central infrastructure using HTTP, but we do not know whether or not they use HTTP or other protocols to communicate between the internal components of the central infrastructure.

This attack pattern has no recommended mitigation method from CAPEC. There is generally very little one can do to stop someone from reverse engineering. By obfuscating the communication protocol and by encrypting data it is possible to lengthen the process, but the attacker will be able to reverse engineer it at some point. By rewriting the protocol and updating security measures regularly, it should be possible to lengthen the reverse engineering process indefinitely.

### 8.3.26 Lifting Sensitive Data from the Client

An attacker may try to gather sensitive data from an application. Sensitive data might be stored in configuration files, temporary files saved by the application or in the application itself.

This attack pattern can be applied to the mobile banking application. By looking at temporary files an attacker might find sensitive information. Both Windows Phone and iOS uses file encryption on disk, which

---

<sup>3</sup>Eika Gruppen AS and Nordea

makes this attack pattern very hard to perform on these systems. Android does not encrypt all files on older versions of the operating system, and these system versions makes this attack a current threat.

This attack pattern has no recommended mitigation method from CAPEC. By making sure that all files, including configuration files and temporary files, are encrypted we can mitigate this attack pattern, but since not all mobile operating system versions do this we have to consider this as a current threat.

### **8.3.27 Reverse Engineer an Executable to Expose Assumed Hidden Functionality or Content**

An attacker may analyze binaries or executables to an application in order to discover data structures, functions and source-code. This can be achieved by utilizing many different techniques, like black box and white box. By reverse engineering the applications and looking at the source code, an attacker may be able to find flaws in the application design that can be utilized to perform unauthorized activity.

This type of attack can be applied to BankID and the mobile banking application. Both the BankID server and client is downloadable, and should be able to be analyzed by reverse engineering. The same applies for the mobile banking application which is downloadable through the application store on each platform.

This attack pattern has no recommended mitigation method from CAPEC. There is generally very little one can do to stop someone from reverse engineering executables, and a we must therefore consider this a current threat.

### **8.3.28 Physical Theft**

An attacker may gain unauthorized physical access to a system in order to perform one or more attacks on the system. This includes information theft, tampering and denial of service.

This type of attack can be applied to the mobile operating system (device) and the BankID infrastructure. By stealing a phone, an attacker may try to perform any attack pattern while also denying the legitimate user access to the service. An attacker may also gain access to either the distributed infrastructure that is hosted by the bank (Bank RA) or the central infrastructure (FOI) and perform a variety of attacks by having physical access.

The suggested mitigation techniques recommended by CAPEC is to have physical security. This includes locks, alarms, access restriction and monitoring. It is easier to physically steal a mobile phone than to steal servers from the data centers that hosts FOI, but we must consider

theft to be a current threat for both parts of the BankID infrastructure and the mobile operating systems (device).

### **8.3.29 Bypassing Electronic Locks and Access Controls**

An attacker might bypass electronic security, by exploiting security assumptions to bypass access control systems. By tricking the access control system, an attacker may gain unauthorized physical access to infrastructure, and by doing so he might be able to perform a variety of other attacks.

This type of attack might be applied to the BankID infrastructure. If Nets utilizes electronic access control systems to be able to give employees access to their data centers, then we need to consider that someone may try to bypass electronic locks and the access control system. We can assume it is likely that they use electronic access control to certain parts of their data centers.

This attack pattern has no recommended mitigation method from CAPEC. It is not possible to be able to completely mitigate this threat, but upgrading the access control systems regularly to prevent attackers from analyzing the system could be a good technique to partially mitigate this attack pattern.

### **8.3.30 Bypassing Physical Locks**

An attacker might bypass physical security in order to access a building or a facility. Physical locks may range from padlocks, traditional lock and key mechanisms and more. Physical locks can be bypassed by bumping the lock, snapping the lock, cutting the lock, or picking the lock. Snapping and picking a lock may destroy the lock, making it obvious that there has been a security breach.

This kind of attack can be applied to the BankID infrastructure. Nets uses physical locks to secure their facilities, and there is no guarantee that these security measures cannot be breached.

This attack pattern has no recommended mitigation method from CAPEC. Using guards to patrol the facilities and watch security cameras, one could partially mitigate this attack pattern.

### **8.3.31 Physical Destruction of Device or Component**

An attacker could physically destroy a device or a component in order to destroy data or deny service.

This kind of attack can be applied to BankID and the different mobile operating systems (device). The device could be stolen and destroyed in a variety of different ways, and there are no good mitiga-

tion techniques. The BankID infrastructure could be attacked by either targeting the bank that hosts the RA, or by attacking the central infrastructure (FOI).

This attack pattern has no recommended mitigation method from CAPEC, and there is always a possibility of physical devices getting destroyed.

### **8.3.32 Malicious Software Download**

An attacker may use deceptive methods to trick a user or a service into downloading and install a malicious application. This application contains dangerous code which enables the attacker to control, tamper with or destroy the device.

This kind of attack can be applied to the BankID infrastructure and to the different mobile operating systems. Mobile operating systems are mostly safe, but Android allows users to download and install applications from unauthorized sources, which exposes the system to such attack patterns. All BankID servers, located at FOI or at a bank, could be susceptible to malicious software downloads.

This attack pattern has no recommended mitigation method from CAPEC. By limiting users from downloading applications from untrusted sources, the operating system or BankID servers would be more secure, but this will only be a partial mitigation and not a complete mitigation.

### **8.3.33 Malicious Software Update**

An attacker may use deceptive methods to trick a user or a service into downloading and installing a malicious software update. The software update contains dangerous code which enables the attacker to control, tamper with or destroy the device.

This kind of attack can be applied to the BankID infrastructure and to the different mobile operating systems. All operating systems have security patches and software updates, and the same applies for BankID servers. An attacker may use several different methods to perform this attack pattern, but all of them require the attacker to deceive the device into downloading code from an untrusted source.

This attack pattern has no recommended mitigation method from CAPEC. This attack might not be completely mitigable, but using code signing might help the automated service or the end user into being sure that the software update comes from the correct source.

### 8.3.34 Target Influence via Social Engineering

An attacker may use social engineering techniques and try to deceive authorized users into sharing information with the attacker or performing malicious actions on behalf of the attacker. This can be done in many different ways. Examples includes: Posing as persons with authority, by manipulating the target, by staying committed to the task, or by being liked.

This kind of attack can be applied to all the different systems. The device may be broken into if the end user willingly, and maybe unknowingly, shares the secret access code needed to use the device, or by unlocking it on request of the attacker. The BankID infrastructure has many employees, and any of these employees are potential targets for such attacks. The mobile banking application cannot be directly attacked by social engineering, but by tricking the device owner into giving away valuable and secret information, an attacker may be able to use this to access the bank accounts without the account owners consent.

This attack pattern has no recommended mitigation method from CAPEC. Social engineering could possible threaten all the different systems in this threat model, and there are no concrete mitigation techniques. Customers, customer service representatives, technicians or other personell may be deceived, and even though proper training might partially mitigate this attack pattern, there is nothing that can completely mitigate it.

## 8.4 Discussion

Attack libraries is an excellent tool to use while threat modelling. In this chapter, we used the CAPEC attack library to identify attack patterns that might be applied to the three different parts of mobile banking services in Norway. Using CAPEC was easy due to the fact that this library is very detailed and includes a variety of tools to help the threat modeller to perform his task.

There are however some flaws with the CAPEC attack library. As we mentioned in chapter 3, attack libraries are closing in on the border of practicality. The complete CAPEC attack library consists of 463 different attack patterns. In this thesis we have analyzed 169 of the available attack patterns, and the remaining 294 attack patterns are all just more detailed attacks than the previous patterns. Analyzing and reflecting upon 169 attacks took about 7 work days for me to complete. This was then done by spending an average of 20 minutes per attack pattern, which in a business setting might not even be enough to complete a full security test. This might stop CAPEC from being the

saviour of all security experts, since most businesses will not pay for the extensive work needed to perform a complete threat model using CAPEC. The 7 work days needed on 169 patterns, is just another example of the extensiveness of the CAPEC attack library.

Since attack libraries need to grow in size in order to be complete enough to trust, they will always be close to the limit of practicality. This makes threat modelling using attack libraries very inefficient and in most cases too time consuming to be completed. In this thesis, we have tried to create a threat model using only public documentation or resources, which even limits testing capabilities. Some of the threats mentioned in this chapter might have already been mitigated, but since no documentation proves it then it is up to the banks themselves to test it.

In most projects, using CAPEC might even be so time consuming that it is not feasible to create a full threat model using CAPEC every time the system changes. In most cases, when a system is built from scratch, there should definitely be made a complete CAPEC threat model. For each major update, all parts that are affected by the update, should be re-evaluated using CAPEC.

## **8.5 Summary**

Out of the 169 CAPEC threats that we analyzed in this threat model, we found that 34 of the threats could be defined as a level 3 threat (Current threat). A level 3 threat are attacks that might not have been mitigated or not even be mitigable at all. CAPEC is a great tool even for beginners in the art of threat modelling to utilize, but it is unfortunately very time consuming. An attack library must necessarily be large in order to reach a certain degree of completeness. There might not be enough available resources to create a complete threat model using CAPEC every time a system gets an update, but all new systems should be modelled using CAPEC and all affected parts of an updated system should be re-evaluated.





**Part IV**

**Conclusion**



# Chapter 9

## Conclusion

### 9.1 Discussion

Mobile banking has had an incredible growth in the market and, at the time of writing, does account for roughly 40 % of all online banking usage. This makes mobile banking security a relevant issue to investigate.

#### 9.1.1 Platforms

Comparing mobile banking software to traditional online banking software show us that there is little to no difference between the web applications and the BankID identification and signing service, on each platform. The major difference is the operating system running the banking application and BankID application. Mobile devices offer a higher level of security out-of-the-box than desktop operating systems. All mobile operating systems uses desktop level kernels, but the utilization of the kernel is more optimized for security than on desktops.

One of the key security features is the way mobile operating systems protects application data. Both data on disk and data residing in memory is protected by different security measures. The use of virtual memory and ASLR protects memory segments from being accessed by unauthorized applications. Data on disk is protected by the user account protection mechanism in the kernel as we can see in section 4.3 and subsection 4.3.2, and all systems encrypt data stored on disk.

The latest versions of all of the three major operating systems includes strict boot security and application verification functionality, which should make sure that there is no malware or other malicious applications running on the system, and that the integrity of mobile banking application is maintained.

The fact that mobile operating systems offer a higher level of security out-of-the-box is very interesting. This might be caused by the

attitude consumers have towards the different devices. The operating system running on iPhones have been restricted since its initial launch in 2007. Apple provides both the hardware and the software, and by doing this they can implement security features that has previously not been possible. This includes secure booting mechanisms and restricting where users can download third-party applications. Doing this on a traditional computer, might not be possible because of end users mentality. Many consumers believe that computers should be open and unrestricted, and adding restrictions to these devices might cause consumers to boycott the manufacturers. This is however not the case with mobile devices. All mobile devices from Apple and Microsoft are completely or partially locked to a certain operating system, and this would not be possible to do with computers due to how consumers perceive these devices. Because of this, mobile operating systems are actually ahead when it comes to platform security.

### 9.1.2 Threat Modelling Methods

The various methods for threat modelling we have presented in this project provide very different ways of analyzing systems. Attack trees are, together with STRIDE, a very abstract method for threat modelling. Attack trees seem to be a good tool to use while working in teams, since it gives a good visual representation of threats and the different attack patterns that might result in this threat. It also looks like a good tool to present attack patterns to individuals who are not skilled in the art of threat modelling. In this project, we used STRIDE instead of creating attack trees. The choice behind this was that without any prior experience, STRIDE did provide a better foundation by categorising different threats. With that being said, STRIDE did not produce many concrete threats that we were not already aware of.

The two abstract threat modelling methods seems to be better suited for more experienced threat modellers, because of the need to know how a system can be attacked in order to be able to consider whether or not the attack pose a threat to the system.

Threat modelling using CAPEC was much easier since this attack library consist several hundred different attack patterns. By first analyzing the composition of every system, we could just consider each attack pattern, and see if it was a threat to one of the systems or not. One of the negative sides of using a very detailed attack library like CAPEC is that it is very time consuming. Not only is it necessary with in depth knowledge of the systems you are analyzing, but you need to consider between 150 and 463 different attack patterns in order to cover the whole range of attacks. In this project, we only considered 169 out of 463 attacks. This was done because there are so many attacks that

were so similar to other attack patterns that we could cover both by analyzing the *parent attack*.

By looking at the two threat modelling methods we utilized in this project, we see that STRIDE did produce few concrete threats, while using CAPEC we found 34. This does however not imply that this will be the case for every system or every person creating a threat model. A more experienced threat modeller will undoubtedly be able to utilize STRIDE more than a beginner, and experienced threat modelling teams might do even better. Still, it seems to be a lot safer to use an attack library while creating a threat model.

Mobile banking applications handles highly sensitive data ranging from account numbers and balance, to private passwords and social security numbers (SSN). During the initial development of such application, the developers need to realize that they are handling sensitive data and take necessary security measures such as doing a really thorough threat analysis of the system. Using STRIDE, the developers might have problems reaching a satisfying level of completeness, since it is very hard to know if the threat model is good enough or not. By using an attack library like CAPEC as a reference to see if every threat has been mitigated, developers might be able to reach a level of completeness which will allow them to ship their product. Since banking applications handles highly sensitive data, it is recommended that a threat model using the complete CAPEC library is created before shipping the applications. Using CAPEC did produce more thorough results than using STRIDE, and it seems to be the better method to use while creating a threat model of such an extensive system like mobile banking applications.

### 9.1.3 Current Threats

It seems that mobile banking applications provides a high level of security against technological attacks. There are however some threats that will never be completely mitigated. Attacks that may deny the end user service are hard to completely mitigate. This includes physical theft or destruction, locking the user out of their account and using flooding attacks like DDoS to choke backend servers from providing service.

We also need to consider social engineering, and phishing or pharming attacks. The latter two are being executed multiple times every year against Norwegian banks, and social engineering is always a threat against systems handling sensitive information. This set of threats is really relevant, because gathering enough information may result in the attacker gaining full control of an account.

Data modification attacks, like man-in-the-middle or transaction data structure abuses, does also pose a threat against these systems, and de-

velopers should make special note of such threats.

## 9.2 Conclusion

Mobile banking software does seem to be secure from an outsiders point-of-view, and mobile operating systems seem to provide higher level of security than desktop operating systems. Most technologies used are state-of-the-art and follow recommendations from both NIST and ISO. This includes all the different components of mobile operating systems, all parts of the BankID infrastructure and the different mobile applications.

CAPEC is a better threat modelling tool for beginners, but more experienced threat modellers could also benefit from using CAPEC instead of more abstract methods like STRIDE or attack trees. Mobile banking applications handles sensitive data, and it is recommended to use an attack library while analysing the system in order to reach an acceptable level of completeness.

The most relevant threats to mobile banking services in Norway are social engineering attacks, denial of service attacks like DDoS and other data modification attacks.

## 9.3 Contributions

Here is what I consider to be my contributions.

- Created a threat model using STRIDE
- Created a threat model using CAPEC

# **Appendices**





# Appendix A

## CAPEC list

### A.1 Gathering Information




#### A.1.1 Excavation




<b>CAPEC-111: JSON Hijacking (aka JavaScript Hijacking)</b>		
Platforms	Not applicable	✓
BankID	Current threat	✗
Application	Not applicable	✓




<b>CAPEC-127: Directory Indexing</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Not applicable	✓




<b>CAPEC-150: Common Resource Location Exploration</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Not applicable	✓




<b>CAPEC-462: Cross-Domain Search Timing</b>		
Platforms	Not applicable	✓
BankID	Not applicable	✓
Application	Not applicable	✓

<b>CAPEC-468: Generic Cross-Browser Cross-Domain Theft</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	




<b>CAPEC-498: Probe Application Screenshots</b>		
Platforms	Current threat	
BankID	Not applicable	
Application	Current threat	




<b>CAPEC-54: Probe Application Error Reporting</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Current threat	

<b>CAPEC-545: Probe Application Queries</b>		
Platforms	Not applicable	
BankID	Current threat	
Application	Current threat	

<b>CAPEC-546: Probe Application Memory</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

### A.1.2 Interception

<b>CAPEC-158: Sniffing Network Traffic</b>		
Platforms	Not applicable	
BankID	Possibly mitigated	
Application	Possibly mitigated	

<b>CAPEC-31: Accessing/Intercepting/Modifying HTTP Cookies</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Possibly mitigated	

<b>CAPEC-383: Harvesting Usernames or UserIDs via Application API Event Monitoring</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-499: Intent Intercept</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓

### A.1.3 Footprinting

<b>CAPEC-292: Host Discovery</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Not applicable	✓

<b>CAPEC-300: Port Scanning</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Not applicable	✓

<b>CAPEC-309: Network Topology Mapping</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Not applicable	✓

<b>CAPEC-529: Malware-Directed Internal Reconnaissance</b>		
Platforms	Not applicable	✓
BankID	Current threat	✗
Application	Current threat	✗

### A.1.4 Fingerprinting

<b>CAPEC-311: OS Fingerprinting</b>		
Platforms	Current threat	✗
BankID	Current threat	✗
Application	Current threat	✗

<b>CAPEC-541: Application Fingerprinting</b>		
Platforms	Not applicable	✓
BankID	Current threat	✗
Application	Current threat	✗

### A.1.5 Social Information Gathering Attacks

<b>CAPEC-405: Social Information Gathering via Research</b>		
Platforms	Current threat	✗
BankID	Current threat	✗
Application	Current threat	✗

<b>CAPEC-406: Social Information Gathering via Dumpster Diving</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-407: Social Information Gathering via Pretexting</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Not applicable	✓

<b>CAPEC-408: Information Gathering from Traditional Sources</b>		
Platforms	Not applicable	✓
BankID	Not applicable	✓
Application	Not applicable	✓

<b>CAPEC-409: Information Gathering from Non-Traditional Sources</b>		
Platforms	Not applicable	✓
BankID	Not applicable	✓
Application	Not applicable	✓

## A.2 Deplete Resources

<b>CAPEC-482: TCP Flood</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-486: UDP Flood</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-487: ICMP Flood</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-488: HTTP Flood</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-489: SSL Flood</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-528: XML Flood</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

### A.2.1 Excessive Allocation

<b>CAPEC-230: XML Nested Payloads</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-231: XML Oversized Payloads</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-492: Regular Expression Exponential Blowup</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-493: SOAP Array Blowup</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-494: TCP Fragmentation</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-495: UDP Fragmentation</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-496: ICMP Fragmentation</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

## A.2.2 Resource Leak Exposure

<b>CAPEC-131: Resource Leak Exposure</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

### A.2.3 Sustained Client Engagement

CAPEC-469: HTTP DoS		
Platforms	Not applicable	✓
BankID	Current threat	✗
Application	Current threat	✗

### A.2.4 Amplification

CAPEC-490: Amplification		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

## A.3 Injection

### A.3.1 Parameter Injection




CAPEC-134: Email Injection		
Platforms	Not applicable	✓
BankID	Not applicable	✓
Application	Not applicable	✓




CAPEC-135: Format String Injection		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠




CAPEC-138: Reflection Injection		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

CAPEC-15: Command Delimiters		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠









<b>CAPEC-174: Flash Parameter Injection</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Not applicable	

<b>CAPEC-6: Argument Injection</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Not applicable	




<b>CAPEC-76: Manipulating Input to File System Calls</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	

### A.3.2 Code Inclusion




<b>CAPEC-251: Local Code Inclusion</b>		
Platforms	Not applicable	
BankID	Possibly mitigated	
Application	Possibly mitigated	




<b>CAPEC-253: Remote Code Inclusion</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Possibly mitigated	




### A.3.3 Resource Injection




<b>CAPEC-23: File System Function Injection, Content Based</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

### A.3.4 Code Injection




<b>CAPEC-18: Embedding Scripts in Non-Script Elements</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	




<b>CAPEC-19: Embedding Scripts within Scripts</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	




<b>CAPEC-41: Using Meta-characters in E-mail Headers to Inject Malicious Payloads</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Not applicable	




<b>CAPEC-63: Simple Script Injection</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Possibly mitigated	

### A.3.5 Command Injection

<b>CAPEC-136: LDAP Injection</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-182: Flash Injection</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-183: IMAP/SMTP Command Injection</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-249: Linux Terminal Injection</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-250: XML Injection</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-66: SQL Injection</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Not applicable	✓

<b>CAPEC-88: OS Command Injection</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓

## A.4 Deceptive Interactions

### A.4.1 Path Traversal




<b>CAPEC-139: Relative Path Traversal</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Possibly mitigated	⚠




<b>CAPEC-64: Using Slashes and URL Encoding Combined to Bypass Validation Logic</b>		
Platforms	Not applicable	✓
BankID	Not applicable	✓
Application	Not applicable	✓




<b>CAPEC-78: Using Escaped Slashes in Alternate Encoding</b>		
Platforms	Not applicable	✓
BankID	Not applicable	✓
Application	Possibly mitigated	⚠

<b>CAPEC-79: Using Slashes in Alternate Encoding</b>		
Platforms	Not applicable	✓
BankID	Not applicable	✓
Application	Possibly mitigated	⚠




### A.4.2 Content Spoofing




<b>CAPEC-145: Checksum Spoofing</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Current threat	




<b>CAPEC-218: Spoofing of UDDI/ebXML Messages</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Not applicable	




<b>CAPEC-502: Intent Spoof</b>		
Platforms	Current threat	
BankID	Not applicable	
Application	Current threat	

### A.4.3 Identity Spoofing

<b>CAPEC-194: Fake the Source of Data</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Possibly mitigated	

<b>CAPEC-195: Principal Spoof</b>		
Platforms	Possibly mitigated	
BankID	Current threat	
Application	Current threat	

<b>CAPEC-473: Signature Spoof</b>		
Platforms	Not applicable	
BankID	Current threat	
Application	Current threat	

<b>CAPEC-89: Pharming</b>		
Platforms	Not applicable	
BankID	Current threat	
Application	Current threat	

<b>CAPEC-98: Phishing</b>		
Platforms	Not applicable	✓
BankID	Current threat	✗
Application	Current threat	✗

#### A.4.4 Resource Location Spoofing

<b>CAPEC-159: Redirect Access to Libraries</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Possibly mitigated	⚠

#### A.4.5 Action Spoofing

<b>CAPEC-103: Clickjacking</b>		
Platforms	Not applicable	✓
BankID	Current threat	✗
Application	Possibly mitigated	⚠

<b>CAPEC-501: Activity Hijack</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓

<b>CAPEC-504: Task Impersonation</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓

<b>CAPEC-505: Scheme Squatting</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓

<b>CAPEC-506: Tapjacking</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓

## A.5 Manipulate Timing and State

<b>CAPEC-25: Forced Deadlock</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-26: Leveraging Race Conditions</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Not applicable	✓




<b>CAPEC-29: Leveraging Time-of-Check and Time-of-Use (TOC-TOU) Race Conditions</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Not applicable	✓




<b>CAPEC-462: Cross-Domain Search Timing</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓




<b>CAPEC-74: Manipulating User State</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Current threat	✗




## A.6 Abuse of Functionality




<b>CAPEC-113: API Abuse/Misuse</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠




<b>CAPEC-133: Try All Common Application Switches and Options</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	




<b>CAPEC-141: Cache Poisoning</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	

<b>CAPEC-213: Directory Traversal</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-465: Socket Capable Browser Plugins Result In Transparent Proxy Abuse</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-48: Passing Local Filenames to Functions That Expect a URL</b>		
Platforms	Not applicable	
BankID	Possibly mitigated	
Application	Possibly mitigated	

<b>CAPEC-87: Forceful Browsing</b>		
Platforms	Not applicable	
BankID	Possibly mitigated	
Application	Possibly mitigated	

<b>CAPEC-95: WSDL Scanning</b>		
Platforms	Not applicable	
BankID	Possibly mitigated	
Application	Possibly mitigated	

**A.6.1 Functionality Misuse**

<b>CAPEC-162: Manipulating hidden fields to change the normal flow of transactions (eShoptlifting)</b>		
Platforms	Not applicable	✓
BankID	Not applicable	✓
Application	Not applicable	✓

<b>CAPEC-2: Inducing Account Lockout</b>		
Platforms	Current threat	✗
BankID	Current threat	✗
Application	Not applicable	✓

<b>CAPEC-464: Evercookie</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓

<b>CAPEC-50: Password Recovery Exploitation</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

**A.6.2 (**




Abuse of Communication Channels)




<b>CAPEC-12: Choosing a Message/Channel Identifier on a Public/Multicast Channel</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠




<b>APEC-217: Exploiting Incorrectly Configured SSL Security Levels</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠






## A.7 Probabilistic Techniques




<b>CAPEC-155: Screen Temporary Files for Sensitive Information</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-28: Fuzzing</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	




<b>CAPEC-39: Manipulating Opaque Client-based Data Tokens</b>		
Platforms	Not applicable	
BankID	Possibly mitigated	
Application	Possibly mitigated	




### A.7.1 Brute Force




<b>CAPEC-20: Encryption Brute Forcing</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Not applicable	

<b>CAPEC-49: Password Brute Forcing</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Not applicable	




## A.8 Exploitation of Authentication




<b>CAPEC-114: Authentication Abuse</b>		
Platforms	Current threat	
BankID	Current threat	
Application	Current threat	




<b>CAPEC-115: Authentication Bypass</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	




<b>CAPEC-21: Exploitation of Session Variables, Resource IDs and other Trusted Credentials</b>		
Platforms	Not applicable	
BankID	Possibly mitigated	
Application	Possibly mitigated	

## A.9 Privilege Escalation

<b>CAPEC-104: Cross Zone Scripting</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-17: Accessing, Modifying or Executing Executable Files</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-58: Restful Privilege Elevation</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-75: Manipulating Writeable Configuration Files</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

## A.10 Exploitation of Authorization

<b>CAPEC-234: Hijacking a privileged process</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-236: Catching exception throw/signal from privileged block</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓




<b>CAPEC-30: Hijacking a Privileged Thread of Execution</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓




<b>CAPEC-30: Hijacking a Privileged Thread of Execution</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠




<b>CAPEC-69: Target Programs with Elevated Privileges</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

### A.10.1 Privilege Abuse




<b>CAPEC-1: Accessing Functionality Not Properly Constrained by ACLs</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓




<b>CAPEC-180: Exploiting Incorrectly Configured Access Control Security Levels</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Possibly mitigated	




<b>CAPEC-221: XML External Entities</b>		
Platforms	Not applicable	
BankID	Possibly mitigated	
Application	Possibly mitigated	




<b>CAPEC-503: WebView Exposure</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

### A.10.2 Exploiting Trust in Client (aka Make the Client Invisible)

<b>CAPEC-202: Create Malicious Client</b>		
Platforms	Current threat	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-207: Removing Important Functionality from the Client</b>		
Platforms	Not applicable	
BankID	Possibly mitigated	
Application	Possibly mitigated	

<b>CAPEC-39: Manipulating Opaque Client-based Data Tokens</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	

<b>CAPEC-77: Manipulating User-Controlled Variables</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	

<b>CAPEC-94: Man in the Middle Attack</b>		
Platforms	Not applicable	✓
BankID	Current threat	✗
Application	Current threat	✗

## A.11 Manipulate Data Structures

<b>CAPEC-124: Attack through Shared Data</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-128: Integer Attacks</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-129: Pointer Attack</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

### A.11.1 Buffer Manipulation

<b>CAPEC-100: Overflow Buffers</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-540: Overread Buffers</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

## A.12 Manipulate Resources

<b>CAPEC-257: Abuse of Transaction Data Structure</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Current threat	✗

<b>CAPEC-548: Contaminate Resource</b>		
Platforms	Current threat	✗
BankID	Current threat	✗
Application	Current threat	✗

<b>CAPEC-153: Input Data Manipulation</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠




<b>CAPEC-161: Infrastructure Manipulation</b>		
Platforms	Possibly mitigated	⚠
BankID	Current threat	✗
Application	Current threat	✗

<b>CAPEC-171: Variable Manipulation</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠




### A.12.1 File Manipulation




<b>CAPEC-11: Cause Web Server Misclassification</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠




<b>CAPEC-177: Create files with the same name as files protected with a higher classification</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-263: Force Use of Corrupted Files</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Not applicable	




<b>CAPEC-35: Leverage Executable Code in Non-Executable Files</b>		
Platforms		
BankID		
Application		




<b>CAPEC-35: Leverage Executable Code in Non-Executable Files</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	

<b>CAPEC-44: Overflow Binary Resource File</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	

<b>CAPEC-73: User-Controlled Filename</b>		
Platforms	Not applicable	
BankID	Not applicable	
Application	Not applicable	

### A.12.2 Configuration/Environment manipulation

<b>CAPEC-203: Manipulate Application Registry Values</b>		
Platforms	Not applicable	
BankID	Possibly mitigated	
Application	Possibly mitigated	

<b>CAPEC-96: Block Access to Libraries</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	

**A.12.3 Audit Log Manipulation**

<b>CAPEC-81: Web Logs Tampering</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

<b>CAPEC-93: Log Injection-Tampering-Forging</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠

**A.12.4 Schema Poisoning**

<b>CAPEC-146: XML Schema Poisoning</b>		
Platforms	Not applicable	✓
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠




**A.12.5 Protocol Manipulation**




<b>CAPEC-168: Windows ::DATA Alternate Data Stream</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Not applicable	✓

<b>CAPEC-220: Client-Server Protocol Manipulation</b>		
Platforms	Possibly mitigated	⚠
BankID	Possibly mitigated	⚠
Application	Possibly mitigated	⚠




<b>CAPEC-275: DNS Rebinding</b>		
Platforms	Possibly mitigated	⚠
BankID	Not applicable	✓
Application	Not applicable	✓



<b>CAPEC-276: Inter-component Protocol Manipulation</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	




<b>CAPEC-277: Data Interchange Protocol Manipulation</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	

### A.12.6 Web Services Protocol Manipulation




<b>CAPEC-279: Soap Manipulation</b>		
Platforms	Not applicable	
BankID	Possibly mitigated	
Application	Possibly mitigated	




## A.13 Analyze Target

### A.13.1 Reverse Engineering




<b>CAPEC-192: Protocol Reverse Engineering</b>		
Platforms	Not applicable	
BankID	Current threat	
Application	Current threat	

### A.13.2 Software Reverse Engineering




<b>CAPEC-167: Lifting Sensitive Data from the Client</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Current threat	

<b>CAPEC-190: Reverse Engineer an Executable to Expose Assumed Hidden Functionality or Content</b>		
Platforms	Not applicable	
BankID	Current threat	
Application	Current threat	




### A.13.3 Cryptanalysis




<b>CAPEC-463: Padding Oracle Crypto Attack</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	

## A.14 Gain Physical Access




<b>CAPEC-507: Physical Theft</b>		
Platforms	Current threat	
BankID	Current threat	
Application	Not applicable	

### A.14.1 Bypassing Physical Security




<b>CAPEC-395: Bypassing Electronic Locks and Access Controls</b>		
Platforms	Not applicable	
BankID	Current threat	
Application	Not applicable	

<b>CAPEC-391: Bypassing Physical Locks</b>		
Platforms	Not applicable	
BankID	Current threat	
Application	Not applicable	




## A.15 Malicious Code Execution




<b>CAPEC-542: Targeted Malware</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

## A.16 Alter System Components




<b>CAPEC-547: Physical Destruction of Device or Component</b>		
Platforms	Current threat	
BankID	Current threat	
Application	Not applicable	

**A.16.1 Software Integrity Attacks**




<b>CAPEC-185: Malicious Software Download</b>		
Platforms	Current threat	
BankID	Current threat	
Application	Not applicable	




<b>CAPEC-186: Malicious Software Update</b>		
Platforms	Current threat	
BankID	Current threat	
Application	Not applicable	




**A.16.2 Hacking Hardware Devices or Components**




<b>CAPEC-402: Bypassing ATA Password Security</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

**A.16.3 Malicious Logic Inserted Into to Product**




<b>CAPEC-452: Malicious Logic Insertion into Product Hardware</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Not applicable	

<b>CAPEC-456: Malicious Logic Insertion into Product Memory</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Not applicable	

<b>CAPEC-538: Open Source Libraries Altered</b>		
Platforms	Possibly mitigated	
BankID	Not applicable	
Application	Not applicable	

<b>CAPEC-539: ASIC With Malicious Functionality</b>		
Platforms	Possibly mitigated	
BankID	Possibly mitigated	
Application	Possibly mitigated	

## A.17 Manipulate System Users

<b>CAPEC-416: Target Influence via Social Engineering</b>		
Platforms	Current threat	
BankID	Current threat	
Application	Current threat	



# Bibliography

- [1] *A Brief Guide to Android Security*. URL: [http://www.acumin.co.uk/download%5C\\_files/WhitePaper/android%5C\\_white%5C\\_paper%5C\\_2.pdf](http://www.acumin.co.uk/download%5C_files/WhitePaper/android%5C_white%5C_paper%5C_2.pdf) (visited on 02/10/2015).
- [2] *A Proof of Concept Attack against Norwegian Internet Banking Systems*. Tech. rep. NoWires Research Group Department of Informatics University of Bergen, Norway, 2008.
- [3] National Security Agency. *Security Enhanced Linux*. URL: <https://www.nsa.gov/research/selinux/index.shtml> (visited on 05/28/2015).
- [4] *Android Platform Versions*. URL: <https://developer.android.com/about/dashboards/index.html> (visited on 02/10/2015).
- [5] *Android Security Overview*. URL: <https://source.android.com/devices/tech/security/> (visited on 02/10/2015).
- [6] *App Store Distribution*. URL: <https://developer.apple.com/support/appstore/> (visited on 02/16/2015).
- [7] *Apple Security Updates*. URL: <http://support.apple.com/en-us/HT1222> (visited on 02/10/2015).
- [8] Tone Hoddø Bakås. *Svindel e-post med logo fra BankID*. URL: <https://norsis.no/2015/06/svindel-e-post-med-logo-fra-bankid/> (visited on 07/18/2015).
- [9] Nets Norway BankID Norge and Knowit. *BankID COI White Paper*. Tech. rep. BankID Norge, 2015.
- [10] *CAPEC-89: Pharming*. URL: <http://capec.mitre.org/data/definitions/89.html> (visited on 07/18/2015).
- [11] *Common Attack Pattern Enumeration and Classification*. URL: <http://capec.mitre.org> (visited on 05/20/2015).
- [12] Quynh Dang Elaine Barker. *Recommendation for Key Management Part 3: Application-Specific Key Management Guidance*. Tech. rep. National Institute of Standards and Technology, 2014.
- [13] Dieter Gollmann. *Computer Security*. Third Edition. WILEY, 2011.

- [14] Shon Harris. *Cissp All-In-One Exam Guide*. McGraw-Hill/Osborne Media; 6 Har/Cdr edition (October 18, 2012), 2013.
- [15] Ann Håkonsen. *Mobilbanken - den nye hverdagsbanken?* URL: <https://www.fno.no/aktuelt/sporreundersokelser/dagligbankundersokelsen1dagligbankundersokelsen-2015/mobilbanken--den-nye-hverdagsbanken/> (visited on 05/25/2015).
- [16] *Information processing systems - Open Systems Interconnection - Basic Reference Model - Part 2 Security Architecture*. Tech. rep. International Organization for Standardization, 1989.
- [17] *Information technology - Security techniques - Entity authentication assurance framework*. Tech. rep. International Organization for Standardization, 2012.
- [18] *Information technology - Security techniques - Information security management systems - Overview and vocabulary*. Tech. rep. International Organization for Standardization, 2014.
- [19] *Information technology - Security techniques - Information security risk management*. Tech. rep. International Organization for Standardization, 2010.
- [20] *iOS Security Overview*. URL: [http://images.apple.com/business/docs/iOS%5C\\_Security%5C\\_Guide%5C\\_Oct%5C\\_2014.pdf](http://images.apple.com/business/docs/iOS%5C_Security%5C_Guide%5C_Oct%5C_2014.pdf) (visited on 02/10/2015).
- [21] *ITU Telecommunication Standardization Sector*. URL: <http://www.itu.int/en/ITU-T/Pages/default.aspx> (visited on 07/09/2015).
- [22] Adam Shostack/Carol Long. *Threat Modeling - designing for security*. Wiley, 2014.
- [23] *MITRE*. URL: <http://www.mitre.org> (visited on 05/20/2015).
- [24] *Mobile/Tablet Operating System Market Share*. URL: <https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10%5C&qpcustomd=1> (visited on 02/25/2015).
- [25] Vebjørn Moen. "Risk Assessment of a National Security Infrastructure". In: *IEEE Security & Privacy* (2009).
- [26] Carey Nachenberg. *A Window Into Mobile Device Security. Examining the security approaches employed in Apple's iOS and Google's Android*. Tech. rep. Symantec, 2011.
- [27] *Nettbank*. URL: <https://snl.no/nettbank> (visited on 07/23/2015).
- [28] The Financial Supervisory Authority of Norway. *Risiko- Og Sårbarhetsanalyse (ROS)*. Tech. rep. The Financial Supervisory Authority of Norway, 2013.

- [29] The Financial Supervisory Authority of Norway. *Risiko- Og Sårbarhetsanalyse (ROS)*. Tech. rep. The Financial Supervisory Authority of Norway, 2014.
- [30] Michael Orelid. *Mobilbank bekymrer Finanstilsynet*. URL: <http://www.cw.no/artikkel/sikkerhet/mobilbank-bekymrer-finanstilsynet> (visited on 07/09/2015).
- [31] Bruce Schneier. “Attack Trees”. In: *Dr. Dobb’s Journal* 24.12 (Dec. 1999).
- [32] Bruce Schneier. *Schneier on Security*. URL: <https://www.schneier.com> (visited on 05/20/2015).
- [33] Adam Shostack. *Experiences Threat Modeling at Microsoft*. Tech. rep. Microsoft, 2008.
- [34] Adam Shostack. *Microsoft Cyber Trust Blog*. URL: <http://blogs.microsoft.com/cybertrust/2007/09/11/stride-chart/> (visited on 05/13/2015).
- [35] *Sparebankgrupperinger*. URL: <http://www.sparebankforeningen.no/id/1493> (visited on 07/09/2015).
- [36] SSB. *Media Statistics for 2014*. URL: <https://www.ssb.no/statistikkbanken/SelectVarVal/saveselections.asp> (visited on 05/25/2015).
- [37] James Trew. *Microsoft confirms no upgrade path to Windows Phone 8, unveils 7.8 for legacy devices*. URL: <http://www.engadget.com/2012/06/20/microsoft-unveils-windows-phone-7-8-for-legacy-devices/> (visited on 02/25/2015).
- [38] *Uncover Security Design Flaws Using The STRIDE Approach*. Tech. rep. Microsoft, 2006.
- [39] *Unified Extensible Firmware Interface*. URL: [http://en.wikipedia.org/wiki/Unified%5C\\_Extensible%5C\\_Firmware%5C\\_Interface](http://en.wikipedia.org/wiki/Unified%5C_Extensible%5C_Firmware%5C_Interface) (visited on 02/12/2015).
- [40] *VISA Spendon*. URL: <https://spendon.com/nb-no/> (visited on 05/27/2015).
- [41] *Windows CE*. URL: [http://en.wikipedia.org/wiki/Windows%5C\\_CE](http://en.wikipedia.org/wiki/Windows%5C_CE) (visited on 02/12/2015).
- [42] *Windows NT*. URL: [http://en.wikipedia.org/wiki/Windows%5C\\_NT](http://en.wikipedia.org/wiki/Windows%5C_NT) (visited on 02/12/2015).
- [43] *Windows Phone 7*. URL: [http://en.wikipedia.org/wiki/Windows%5C\\_Phone%5C\\_7](http://en.wikipedia.org/wiki/Windows%5C_Phone%5C_7) (visited on 02/12/2015).
- [44] *Windows Phone 8.0 Security Overview*. URL: <http://www.microsoft.com/en-us/download/details.aspx?id=36173> (visited on 02/10/2015).



- [45] *Windows Phone 8.1 Security Overview*. URL: <http://www.microsoft.com/en-us/download/confirmation.aspx?id=42509> (visited on 02/12/2015).