

Smartphone App Security: Vulnerabilities and Implementations

by

Linxi Zhang

**A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Computer and Information Science)
in the University of Michigan-Dearborn
2018**

Master's Thesis Committee:

Associate Professor Di Ma, Chair

Associate Professor Jinhua Guo

Associate Professor Shengquan Wang

Dedication

Every challenging work not only need self-motivation but also the guidance of elders especially those who are very close to my heart. I would like to dedicate my humble effort to my sweet and loving parents who support over the years to help me finish this dissertation.

Acknowledgements

I would like to take this opportunity to thank my advisor Dr. Di Ma for her invaluable guidance, consistent support, and encouragement. I thank her for not only the professional research training but also teaching me every essential, including ethics, rules, and principals to be knowledgeable and scientific in my future life and research. I would also like to thank the members of my master committee, Dr. Di Ma, Dr. Jinhua Guo and Dr. Shengquan Wang.

Furthermore, I thank all a team member in UM-Dearborn CIS department—SAFE lab. I appreciate the time and opportunities for brainstorming and discussions with the most talented co-workers, Haoyu Li, JiaFa Liu, Huaxin Li, Shuang Yu, etc.

I would also like to thank Xing Jin, who implemented HTML5 experiments and kindly gave me suggestions and discussions about my research problems and exchange ideas. And I appreciate UM-Dearborn CIS students, Brandon Falk, Shuang Yu, etc. They took their precious time to take my lab experiment surveys and give me a lot of feedbacks and improvement inputs.

Finally, I would like to thank my husband, Xuke Yan. Without his company and support over the years and encouragement for continuing my research work, I could not complete my master thesis that fast and precise.

Table of Contents

Dedication	ii
Acknowledgements	iii
List of Tables	vii
List of Figures	viii
List of Appendices	x
Abstract	xii
Chapter 1 Introduction	1
Chapter 2 SSL Implementation Vulnerability	4
2.1 Overview	4
2.2 Background.....	5
2.2.1 Android and Secure Socket Layer	5
2.2.2 HTTP and HTTPS	10
2.2.3 Man-in-the-Middle Attack.....	12
2.3 Validation Experiment Design	13
2.3.1 Motivation	13

2.3.2	Lab Design.....	14
2.3.3	Tools for the Experiment.....	15
2.3.4	Experiment Implementation	18
Chapter 3 WebView Attacks		23
3.1	Overview	23
3.2	Background.....	23
3.2.1	WebView.....	24
3.2.2	Trusted Computing Base and Sandbox Protection.....	24
3.2.3	Two Type of Attacks.....	25
3.2.4	JavaScript Injection Attack.....	28
3.3	Validation Experiments Design.....	28
3.3.1	Motivation	28
3.3.2	Lab Design.....	29
3.3.3	Tools for the experiment	30
3.3.4	Experiment Implementation	31
Chapter 4 HTML5-Based Application.....		35
4.1	Overview	35
4.2	Background.....	36

4.2.1	HTML5 and Hybrid App.....	36
4.2.2	PhoneGap	39
4.2.3	The Code Injection Attack.....	41
4.3	Validation Experiment Design	45
4.3.1	Motivation	45
4.3.2	Lab Design.....	46
4.3.3	Tools for the experiment	46
4.3.4	Experiment Implementation	47
	Chapter 5 Conclusion and future work	50
	Reference	52

Lists of Tables

Table 2-1 Configuration of Lab Tools	16
Table 3-1 Configuration of Lab Tools	30
Table 4-1 Configuration of Lab Tools	47

Lists of Figures

Figure 1 2016-2017 Worldwide Smartphone OS Market Sharing from IDC [32].....	6
Figure 2 SSL Handshake [2].....	7
Figure 3 Generation/Verification of Digital Signature	9
Figure 4 HTTP vs HTTPS	11
Figure 5 Man-in-the-middle Attack	12
Figure 6 Man-in-the-middle Attack-2.....	13
Figure 7 Lab Architecture	15
Figure 8 Process of Explicit HTTP	17
Figure 9 Lubuntu Platform Setting	18
Figure 10 Setting up Proxy-1	19
Figure 11 Setting up Proxy-2	19
Figure 12 Setting up Proxy-3	20
Figure 13 Android Emulator	21

Figure 14 Account and Password Caught by mitmproxy	22
Figure 15 Java-to-JavaScript Attack.....	27
Figure 16 JavaScript-to-Java Attack.....	28
Figure 17 JavaScript Injection Attack.....	30
Figure 18 Interface of the Malicious App.....	32
Figure 19 Filling in User’s Information.....	33
Figure 20 User’s Information Caught by Attacker	34
Figure 21 Naive App.....	38
Figure 22 Web App.....	38
Figure 23 Naive App.....	39
Figure 24 The Usage of PhoneGap.....	40
Figure 25 Structure of PhoneGap	41
Figure 26 XSS attack and Code Injection Attack	43
Figure 27 Code Injection Attack for HTML5-based App	46
Figure 28 MP3 File Properties	48

Figure 29 Example Contact Information 48

Figure 31 Example MP3 Player Interface..... 49

Figure 30 Attack Result 49

List of Appendices

Appendix A: SSL implantation Vulnerability Lab	56
Appendix B: WebView Attack Lab	70
Appendix C: Code Injection Attacks on HTML5-based Mobile Apps Lab	79

Abstract

Due to the high occupancy volume of smartphones in modern society, more and more developers join the smartphone app market and develop various mobile applications that could benefit our life in many ways. However, smartphone apps are often blamed for insecurities due to smartphone technologies as well as inexperienced app developers.

In this thesis work, we study smartphone app security vulnerabilities due to either improper implementations or improper use of smartphone technologies. More specifically, we study potential security vulnerabilities in three categories of apps: apps which use the secure socket layer(SSL) protocol for secure communication, apps which use the WebView technology, and apps which are HTML5-based. For each category of apps, we analyze the underlying technologies to show the cause of vulnerabilities, and develop instruction materials for each of the three validation attacks we have implemented and turn them into security teaching labs. These security teaching labs aim to help students to understand the theoretical attack concepts in an accurate and understandable way and cultivate the hacking mindset.

Chapter 1 Introduction

Nowadays smart phones play crucial roles in modern society where over a third of the world's population is projected to own a smartphone by 2018[31]. Not only do the smartphone users extensively rely on the fundamental functions of their phones, but also, they download vast quantities of mobile applications to make their daily life more efficient and enjoyable. According to the May 2017 IDC Quarterly Mobile Phone Tracker [32] report about Smartphone OS Market Share in 2017 Q1, Android and iOS are the most popular operating systems for smart phones, capturing 99.7% worldwide smartphone volume, where more than 85% new shipped devices pre-installed with Android. Based on the above background, the growing numbers of people begin to pay attention to this huge volume market and be aware that both platforms should be secured. However, vulnerable applications running on these platforms are making smart phones insecure. This insecurity is largely caused by the 1) relatively open and unrestricted software development markets that allow anyone be a mobile software developer who may lack sense of security or ignore the security vulnerabilities in the mobile development technology; 2) lacking clear mobile security standards on mobile development technologies.

Security issues in Android are always a big concern [33], especially for secure communication and the application security. To provide secure communication between an application and the app server, SSL technology is utilized to protect data through encryption, digital certificate, and other related technologies. However, a number of previous work demonstrate multiple security vulnerabilities due to improper SSL implementation. The work of [2] shows a number of SSL implementation vulnerabilities in None-Browser software, such as Amazon's EC2 Java library and ZenCart, etc. Although it does not specifically focus on the Android platform, it suggests similar issues may exist in the mobile platforms. The work of [3] focuses on SSL implementation vulnerabilities on the Android applications. In order to break the limitation of native app, developers tend to use WebView, middleware and web technologies to develop applications. However, the inappropriate usage of WebView is still a major cause for application security [4][5]. New security vulnerabilities are also found in HTML-5 based apps [6][7]. The work of [4][5]

illustrates several potential attacks on WebView and explains the reasons for those attacks (fortunately there are no real attacks found in the Android application market yet). The work of [6][7] states systematic studies on the security risks in HTML5-based mobile apps, and the authors claim a new form of code injection attack for HTML5-based apps.

All the research mentioned above contributes to the Android security fields significantly. However, there is an issue that these research works might be too complicated to be understood by most students. Especially, students who do not have the related background must take much time and effort to absorb the complex information. Also, those research papers cannot provide enough guidance for students, to implement and validate the attacks. It is reasonable that those excellent research results should be disseminated into students' fundamental education in cybersecurity filed.

In this thesis work, the main contributions of this thesis are three attack experiments. Those experiments designed to explore security vulnerabilities in three common technologies — SSL, WebView and HTML5-based application. These experiments along with instructions are used to help students learn smartphone application security concepts quickly, comprehensively and precisely, understand the security vulnerabilities in three common technologies, as well as educate them to acquaint security concerns and security vulnerabilities in practical applications through hands-on labs. These specially designed experiments are also expected to help those future developers equipped with sense of security and technology to deal with more tough problems. We implement these validation experiments for each category on the smartphone and on virtual machines, and we explain these three validation experiments step by step in Appendices section.

In Section 2, a systematic analysis of SSL implementation vulnerability is provided. In normal situation, communication between the applications and servers is secure once SSL is applied. However, previous research has shown that a significant number of apps lacks the proper implementation of the SSL protocol, mostly due to the flawed public key certificate validation procedure [2][3].

The WebView attack is instructed in Section 3. As an “embedded browser” for applications, WebView helps the mobile applications to gain more flexible and user-friendly functions, in the

meantime, weakens the Web's security infrastructure on the Trusted Computing Base (TCB) at the client side, and the sandbox protection on the browser side. Consequently, many attacks could be launched by attacking WebView [4][5].

Security vulnerabilities in HTML5-based mobile applications is discussed in Section 4. HTML5-based mobile applications, by using middleware (such as, PhoneGap), take the portability advantage to obtain the convenience on cross-platform development conditions. However, there is a dangerous feature which allows data and code to be mixed together in HTML5-based applications [6][7].

In this thesis work, we study smartphone app security vulnerabilities from three categories caused by improper implementations or improper smartphone technologies utilization. Three categories of apps are apps using the secure socket layer(SSL) protocol for secure communication, apps using the WebView technology, and HTML5-based apps. We analyze the underlying technologies to show the cause of vulnerabilities, and develop three validation attacks instruction materials that we have implemented and turned them into security teaching labs. These security teaching labs aim to help students to understand the theoretical attack concepts in and accurate and understandable way and cultivate the hacking mindset.

Chapter 2 SSL Implementation Vulnerability

2.1 Overview

Secure Socket Layer (SSL) is developed by Netscape Communications originally as an accepted standard for Web security. It allows secure access of a website from a client browser in a secure way. It provides data confidentiality, message integrity, and user authentication through the use of basic cryptographic elements such as encryption, digital signatures, and certificates.

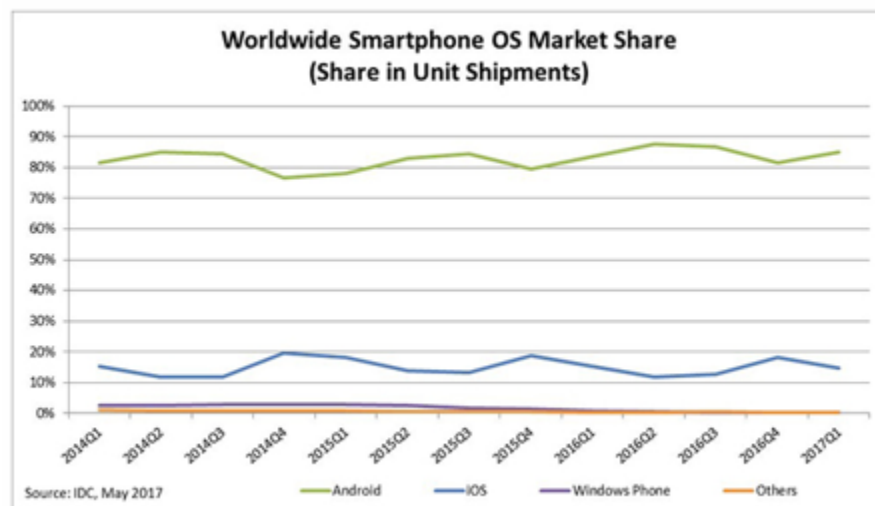
The service of a smart phone app is mainly about communication with outside servers. As a standard for Web security, SSL is necessary to apply on Android apps. By the utilization of SSL Android platform provides a security interaction to connect with server. However, if developers use SSL in an incorrect or improper way, the app applied SSL may be subject to the Man-in-the-Middle (MITM) attack caused by trusting all certificates, allowing all hostnames, trusting many CAs and mixed-mode or no SSL [3].

This section focuses on SSL implementation vulnerability on Android platform, and it is composed of two parts: background and validation experiment. In 2.1, we provide a background of SSL and other related concepts. In 2.2, we conduct a validation experiment that is designed to explore SSL implementation vulnerability. More specifically, we build a lab experiment model, which run in two different environments: on a smartphone or on a smartphone emulator. The advantage of using an emulator instead of a real phone is reducing the budget and limitation of setting up the experiment environment. In this lab, we utilize a Man-in-the-Middle attack (MITM) proxy server to test the attack process. The conversation data between user and server are collectable and analyzable during this attack. This lab aims to help students have a better understanding of SSL implementation vulnerability. At the same time, it helps students to improve their self-learning and independent thinking ability and acquire the critical thinking ability for their future study.

2.2 Background

2.2.1 Android and Secure Socket Layer

Android is the name of the mobile operating system owned by American company-Google. As a mobile operating system, Android is based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android help users to look for information on the web, watch videos, send emails and navigate by using the smartphone or other smart devices. As a highly customizable operating system, Android could be customized to suit customer's needs. For example, various kinds of applications are downloaded and installed on smartphone to help and improve users' daily life. The users manage their bank account, social media account, payment and address account through mobile applications. As a user-friendly and powerful mobile operating system, Android plays a critical role in the mobile market. According to a research form IDC (Figure 2-1), Android is the most popular mobile platform in contemporary smartphone OS market. Because of the Android's high proportion in mobile operating system market, the number of Android applications is huge comparing with other mobile platforms. Google Play market also provides a more freedom and open situation than Apple's Apple Store. Since the Google Play market provides developers and customers a flexible atmosphere, the potential risks become more dangerous.



Period	Android	iOS	Windows Phone	Others
2016Q1	83.4%	15.4%	0.8%	0.4%
2016Q2	87.6%	11.7%	0.4%	0.3%
2016Q3	86.8%	12.5%	0.3%	0.4%
2016Q4	81.4%	18.2%	0.2%	0.2%
2017Q1	85.0%	14.7%	0.1%	0.1%

Source: IDC, May 2017

Figure 1 2016-2017 Worldwide Smartphone OS Market Sharing from IDC [32]

SSL or TLS (Transport Layer Security) is a common building block for encrypted communications between clients and servers. TLS is an updated, more secure version of SSL. Generally, both of them are referred to SSL because it is a more commonly used term. SSL is the de facto standard for secure communication. SSL protocol is an optional choice for the development of app. For security propose, Android applications take advantage of SSL for secure communication. Especially, the protection of sensitive data is very critical for some special mobile applications, including financial or banking app, email service app, and shopping app, etc. SSL helps client and server to establish a security “tunnel” through the unsecure network. Namely, SSL protects the communication from tampering and eavesdropping.

To setting up a secure connection between user and server, SSL offers an identification process called handshake shown as Figure 2-2. When the client and the server both agree to start communicating by using SSL, a SSL handshake occurs. The first step is that SSL client sends a requests message to SSL server; the second step is that SSL server sends back to SSL client a response with server’s certificate; then the client verifies the server’s certificate; the forth step is that server send “cipher” to the client. If the handshake process successfully, a safe communication channel is established. The SSL handshake process includes various parameters which are needed to be verified on both sides. That is, after SSL handshake, the client and the server both identify each other and confirm the encryption algorithm and several parameters. In common scenario, SSL protocols include a handshake with an asymmetric cipher to establish cipher settings and a shared key for a session. For the rest of the communication parts, a symmetric cipher and the session key

will be used for encryption. It should be pointed out that SSL operates over some reliable transport layer. That means SSL is placed as application layer protocols in the TCP/IP reference model and as presentation layer protocols in the OSI model. To clarify the SSL implementation vulnerability, cryptographic encryption, digital signatures and certificates, as three important components of SSL, are discussed in the following.

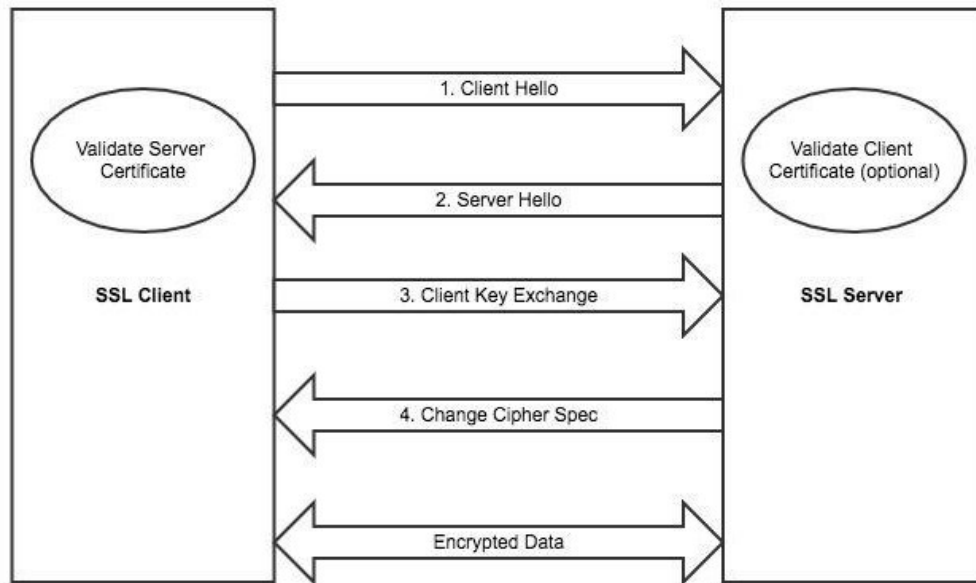


Figure 2 SSL Handshake [2]

Cryptographic encryption

Cryptographic encryption is a critical part in SSL. This process guarantees that information can only be read after decrypting. Namely, the encoding messages or information could only be read by the authorized party who has the decryption key of the encryption method. Encryption prevents interception and decoding by an attacker. Generally, unauthorized users are excluded easily by encryption, because they do not have the required decryption keys. Usually, encryption is categorized into two types: symmetric key encryption or asymmetric key encryption. In symmetric key encryption schemes, both the encryption and decryption process use the same key. Thus, the

two communication parties (for example, the client and the server) must have the same key before they can send each other information in a secure way. In SSL, the symmetric key encryption is used to encrypt data after the handshake process. Asymmetric key encryption is also called public key encryption. In this kind of encryption, the encryption key is published for anyone to encrypt messages. It is the reason why this encryption key is called as public key. By using public key encryption, the decryption key is only accessed by the authenticated user who can decrypt the message. Thus, the decryption key is also called as private key. In SSL, public key encryption is used in the handshake process.

Digital signatures

A digital signature is a mathematical scheme for demonstrating the authenticity of a digital message or documents. A valid digital signature gives a recipient reason to believe that the message was created by the known sender. Since the sender cannot deny having sent the message which is not able to alter in transit process, authentication, non-repudiation and integrity are guaranteed. Therefore, digital signature is a standard element of cryptographic protocol suites and is commonly used for cases where it is important to detect forgery or tampering, such as software distribution, financial transactions, contract management software, etc.

Digital signatures are based on public key cryptography which is able to generate two keys, such as RSA. They are mathematically linked: one private and one public. As shown in Figure 2-3, to create a digital signature, signing software creates a one-way hash of the electronic data to be signed. The private key is used to encrypt the hash. The encrypted hash along with other information, such as the hashing algorithm, is the digital signature. The reason for encrypting the hash instead of the entire message or document is that a hash function converts an arbitrary input into a fixed length value, which is usually much shorter. The value of the hash is unique to the hashed data. Any change in the data, even changing or deleting a single character, results in a different value. When others use the signer's public key to decrypt the hash, the data integrity is validated.

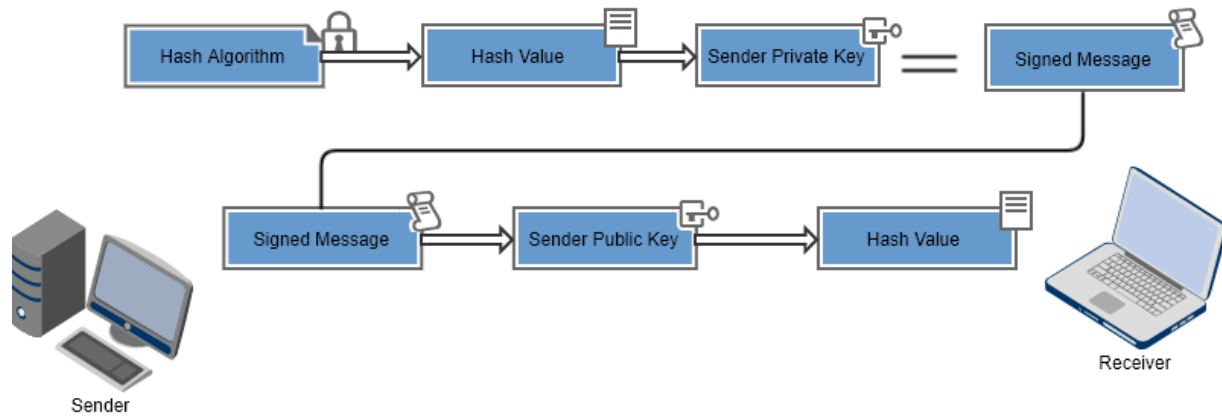


Figure 3 Generation/Verification of Digital Signature

Certificates

A digital certificate is an electronic document that helps entities to exchange information securely over the Internet by using the public key infrastructure. A digital certificate may also be referred to as a public key certificate. In general, the digital certificate contains two main components: a form of ID; and a public encryption key. The ID part is comparable to a passport or driver's license. The public encryption key guarantees the encrypted data can only be decrypted by the owner of the certificate. That is, the digital certificate serves two purposes: identify the entity's identity and secure the communications. A scheme in digital certificate system supports that one certificate can be used to sign other certificates. For example, there is an entity who is trustworthy. Therefore, any other certificates signed by this entity are also trustworthy. In this situation, this trusted entity is referred as a "certificate authority" (CA) which is a trusted third-party organization or a trusted certificate authority. CA issues the digital certificate that provides identifying information is forgery-resistant and can be verified because CA is a trusted authority. CA is also responsible for management digital certificate that contains the name of the certificate holder, a serial number, expiration dates, and a copy of the certificate holder's public key and the digital signature of the certificate-issuing authority. The recipient can use all the information listed above to check the certificate's validity. In SSL handshake process, clients need to identify server's CA, signature information.

The implementation of SSL in Android is not complicated. However, according to an analysis from Sascha Fahl [3], 13,500 popular free apps downloaded from Google's Play Market are vulnerable to Man-in-the-Middle (MITM) attack. The primary reason for those apps' vulnerability is the improper implementation of SSL. Since the developers' background are various, the incorrect implementation of the SSL/TLS protocol led to SSL implementation vulnerabilities. There are some misuse cases [3] can break the secure SSL channel and cause the SSL implementation vulnerabilities. They are 1) Trusting all certificates. By the misusing of TrustManager interface is easily implemented to accept and trust all certificates. 2) Allowing all hostname. The checks of certificate hostname are missing or skipping in some situation. 3) Trusting many or unknown Cas. Even this is not a real flaw, but it may bring poetical risk for SSL process 4) Mixed-mode or no SSL. Some apps are developed to working for both secure and insecure connection. This mixture mode is convenient for attacker to execute attacks based on SSL implementation. In the experiment implementation validation part, we conduct one of the SSL implementation vulnerability attacks.

2.2.2 HTTP and HTTPS

The Hypertext Transfer Protocol (HTTP) is the foundation of data communication for World Wide Web. The original design of HTTP is aiming to provide a method to send and receive HTML pages. In contemporary information society, HTTP is the most used communications protocol. In client-server model with HTTP, a client sends a HTTP request to a server's port which is assigned by using a web tools, such as web browser. Then, the server responses to the client with several resources, such as HTML files or images.

By adding the SSL technology to HTTP, a "secure" version of HTTP is created. That called Hypertext Transfer Protocol Secure (HTTPS, also called HTTP over TLS, HTTP over SSL, and HTTP Secure) provides the encrypted and secure connection channel between the server and the client. Figure 2-4 shows the structures of HTTP and HTTPS. HTTPS is a protocol which provides secure communication over the untrusted network situation. The main intentions of this protocol are: authentication of the visited website; protection of the integrity and the privacy of the exchanged data. In the HTTPS communication, HTTPS takes advantage of HTTP to communicate

and SSL to encrypt data exchanged between the client and server. The digital certificate signed by trusted third-party CA is used in HTTPS. Once the server's certificate is verified and trusted, the secure communication between the client and the server is established. HTTPS protects the connection from Man-in-the-Middle attack and eavesdropping attack.

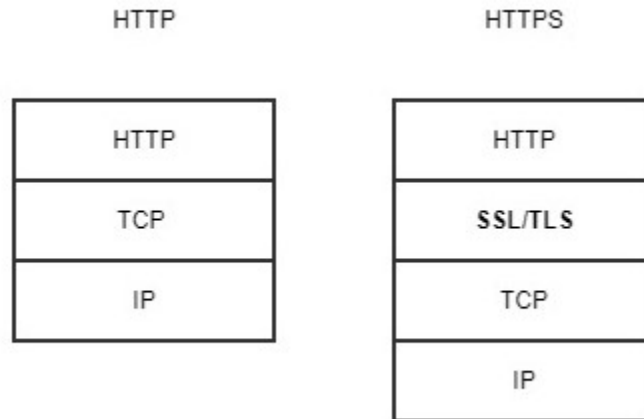


Figure 4 HTTP vs HTTPS

On the one hand, HTTPS is almost identical to HTTP, because HTTPS follows the same basic protocol. The HTTP and HTTPS clients, such as a browser, could establish a connection to the server on a standard port. And when a server receives a request, it returns status and message which contain the requested information or error information if there is an unexpected problem occurred. In addition, because both HTTP and HTTPS use Uniform Resource Identifier (URI) scheme, the resources information can be identified universally. For security purposes, HTTPS is used for an encrypted connection in a URI scheme rather than HTTP. On the other hand, the primary differences between HTTP and HTTPS are distinct. HTTPS is a protocol which encrypts the data between client and the server. Therefore, the data could only be read by authorized entity. In other words, the information cannot be accessed by any other third parties except the client and the server. However, all information is plain text in HTTP. That is, any other entities or third party are able to access the data in the client and the server conversation. It is a dangerous feature for sensitive content. That is the reason why HTTPS protocol are chosen to protect applications' communication security.

2.2.3 Man-in-the-Middle Attack

A man-in-the-middle (MITM) attack is a form of eavesdropping where communication between the server and client is monitored and may be modified by an unauthorized party. The MITM attacker is in a position to intercept messages sent between communication entities. There are two types of MITM attacks: passive MITM attack; active MITM attack. In the passive MITM mode, the only thing attackers can do is eavesdropping the communication information. In the active MITM attack, attackers are able to eavesdrop and modify the communication. MITM attacks cause serious threats because they allow the attackers to have the ability to capture or tamper sensitive information in real-time. Generally, the attacker actively eavesdrops by intercepting a public key message exchange and retransmits the message while replacing the requested key with his own. For two original users, they have no opportunity to recognize their connection under attack or not, they just send and receive the messages which may be modified by attackers as shown in Figure 2-6. And Figure 2-5 describes a whole communication that is total controlled by the attacker. Since mobile devices are usually used in changing and untrusted environments, MITM attack against mobile devices are easier to execute comparing with the traditional devices, including laptop or desktop. In our following validation experiments, we choose the active MITM attack for the experiment part. The MITM attackers have the capability to eavesdrop and generate a fraud certificate signed by themselves to execute this attack.

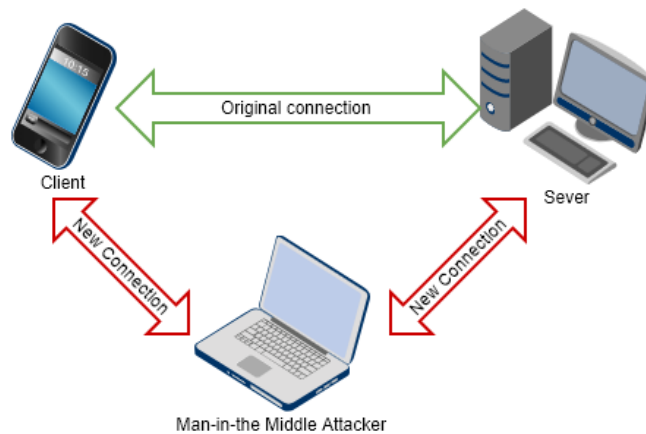


Figure 5 Man-in-the-middle Attack

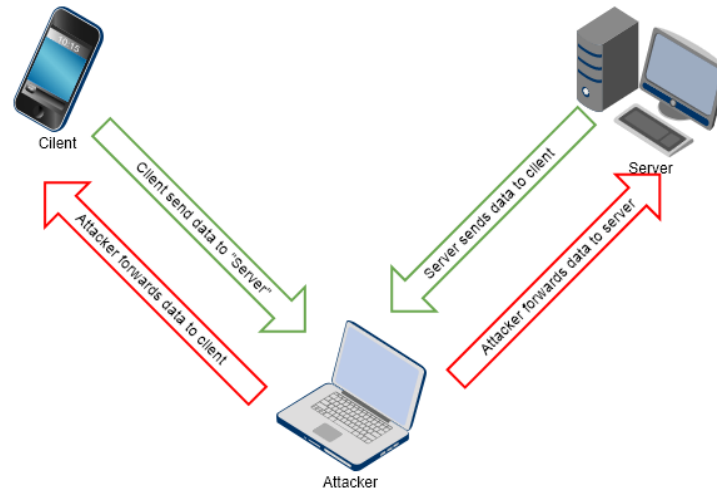


Figure 6 Man-in-the-middle Attack-2

2.3 Validation Experiment Design

2.3.1 Motivation

SSL is a standard for secure Internet communications. However, SSL certificate validation has been shown completely broken in many security-applications and libraries. Flawed certificate validation renders this software vulnerable to man-in-the-middle attack. In the smartphone application market, applications applied SSL are suspected to be flawed in certificate validation, because those applications' developers have various level of security knowledge and the there is no clear standard guidance for the usage of this technology. For education propose, this lab aim to help students have hands-on experience on the Android SSL vulnerability caused by improper usage of SSL. In addition, this lab experiment can not only help students to learn the contents of SSL vulnerability but also inspire them use the knowledge to real project. In this experiment, we expect students conduct a serial of experiments to find flawed apps and analyze the causes. We list the example apps that is subject to MITM attack in experiment example result part. This experiment can also help students who learn from a lecture or by themselves understand the SSL implementation vulnerability. Furthermore, students can have a better understanding on SSL implementation vulnerability attack by doing this experiment. There are three design objectives: 1) Enhancing the basic principles and concepts through learning the process, such as what is the SSL implementation vulnerability, methods to launch MITM attack, etc.; 2) Helping students to

gain first-hand experience of SSL implementation vulnerability by putting what they have learned about the vulnerability from class into real attack; 3) Inspiring students to explore more SSL security problems by guiding them implement this lab experiment step-by-step.

2.3.2 Lab Design

The lab design is establishing a MITM attack model and executing this attack. Three elements are involved: the client; the server; and the proxy. Since the server is the remote server identified by the victim client, we only need to modify the client and proxy side. Figure 2-7 shows the architecture of this lab. The proxy executes the attack as an attacker. And CN means common name or hostname, SAN means subject alternative name. SAN filed in a certificate is optional that allows an arbitrary number of alternative domains to be specified. If the attacker extract both CN and SAN, the attacker is capable to generate a fake certificate. SNI means server name indication which allows the client specify the server' name at the beginning of SSL handshake. When a MITM attack launched, a normal SSL handshake changes to the following process: 1) The victim client sends a connection request to mitmproxy by using HTTP; 2) mitmproxy sends back a "200 Connection Established" response; 3) The client trusts the proxy as the remote server and initiates the SSL connection with SNI; 4) The proxy sets up a connection between the server by using the client's SNI; 5)The server responses with the corresponding certificate containing CN and SAN information; 6) MITM generates a fraud certificate and sends back to client; 7) The attacker initiates the MITM attack successfully. The client communicates with the attack over the established SSL connection. In this experiment, the server is the mobile app's server that is no need to modify.

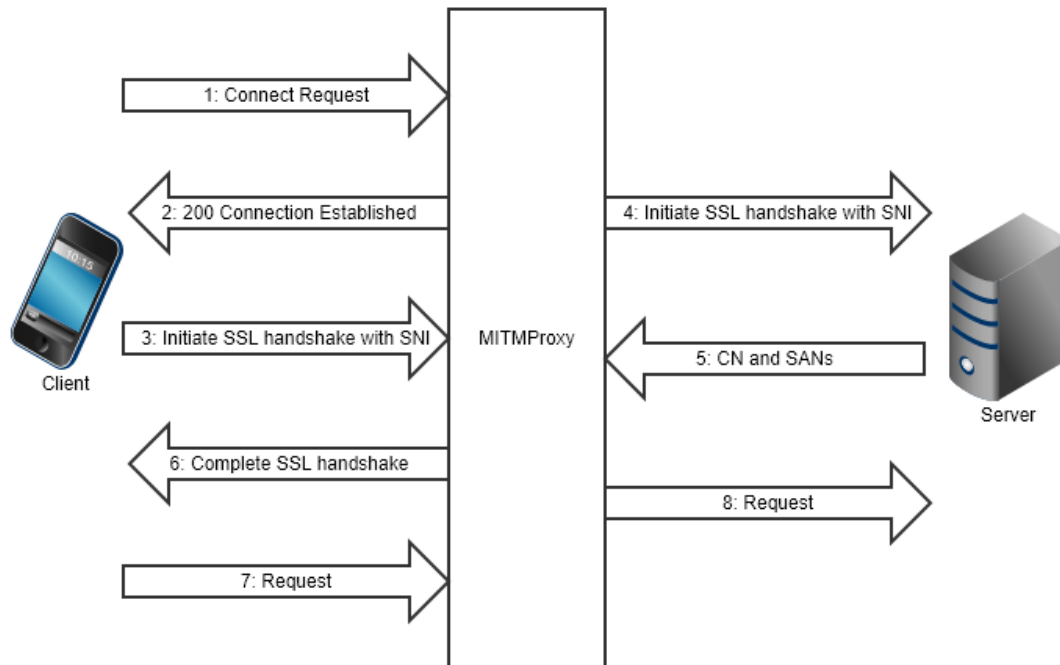


Figure 7 Lab Architecture

2.3.3 Tools for the Experiment

2.3.3.1 Client Side

In this lab, the client side could work on a real mobile device or a virtual machine. Both can provide the same result. The Android virtual machine option reduces the device limitation for this experiment and provides more flexibility for the students. Table 2-1 illustrates the detail version information of Android device and android virtual machine.

2.3.3.2 Proxy server

In MITM attack, the attacker inserts in the communication between the client and the server. To setting up a proxy as a MITM attacker, mitmproxy is chosen for this lab. mitmproxy is an open source proxy software which developed by Python and C. As a man-in-the-middle proxy, mitmproxy allows user to intercept HTTP and HTTPS traffic - the latter by forging SSL certificates. It also can be used to intercept, modify, replay and save HTTP/HTTPS connection. This is incredibly useful for debugging pyWPikibot network issues, especially because tools such

as ethereal are incapable of sniffing the HTTPS traffic. In addition, mitmproxy allows tampering with the traffic, allowing user to fake network errors.

PROXY	
Mode	Virtual Box Machine
Operating system	Lubuntu
Version	16.04.3 LTS
Network mode	Bridged and wired
Mitmproxy version	3.0.4
ANDROID DEVICE	
Operating system	Android
Version	7.0
Application market	Google Play
Proxy	mitmproxy
ANDROID VIRTUAL MACHINE	
Operating system	Android
Version	7.1.1
Application market	Google Play
Proxy	mitmproxy
OTHER SOFTWARE	
Name	Wireshark
Version	2.2.10
Description	Traffic monitoring

Table 2-1 Configuration of Lab Tools

In terms of the usage and the objective, four mitmproxy's operating modes are provided. They are Explicit HTTP, Explicit HTTPS, Transparent HTTP and Transparent HTTPS. According to the lab design and the motivation, only Explicit HTTP and Explicit HTTPS are discussed in the following paragraph. The reference [34] shows how to use the other modes. And the version information of mitmproxy is listed in Table 2-1.

Explicit HTTP

The process of an explicit HTTP connection is the easiest and most reliable method to eavesdrop the combination of MITM attacker. Because this proxy uses the same protocol as the

client and server, the mitmproxy is capable to insert the conversion between the client and the server directly. The following image shows the process of explicit HTTP.

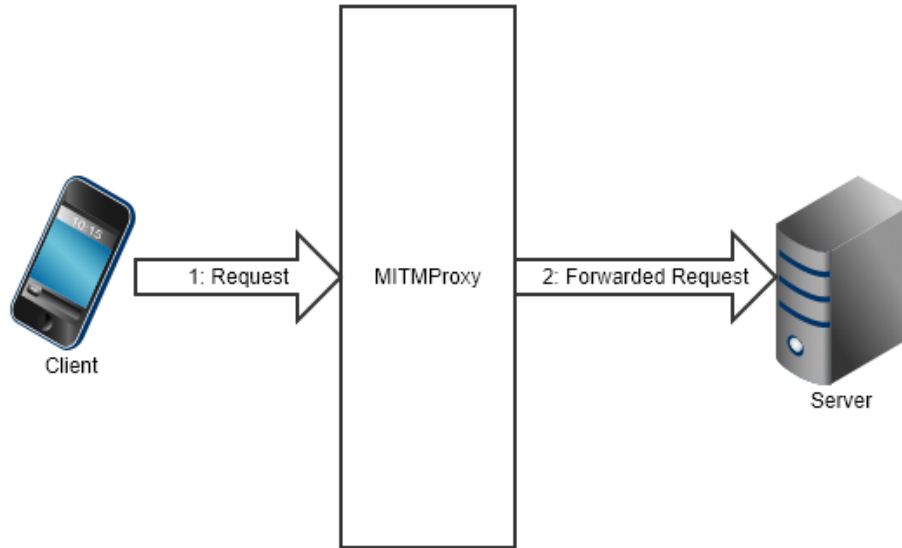


Figure 8 Process of Explicit HTTP

Explicit HTTPS

Since the HTTPS adds the security parts--Certificate Authority system, the process of an explicit HTTPS connection (Figure 2-7) is difficult comparing with explicit HTTP. As a Man-in-the-Middle proxy, mitmproxy pretends to be the client to the server, pretends to be the server to the client. However, the basic design idea for the CA is that protect the connection from this MITM attack. The CA system allows a trusted third-party to cryptographically sign an SSL certificate for verification. When the signature from an untrusted party or do not match, the secure connection between client and server will be closed. However, there are still many disadvantages which will help MITM attack launch successfully. In this case, as a MITM attacker, mitmproxy receives the server's certificate after the client's request, then the proxy sends back its own certificate instead of server's certificate. Generally, an application without SSL implementation vulnerabilities will reject the fake certificate and end the connection immediately. Otherwise, if an application accepts

the certificate from the proxy, that means the application has the SSL vulnerabilities, at the same time, it may leak sensitive and confidential information to the attacker.

2.3.4 Experiment Implementation

Experiment Setup

- Set up the malicious proxy server

Because mitmproxy run on Linux platforms usually, Windows operating system user is required to create and run a Lubuntu Linux virtual machine (VM) on VirtualBox (or other virtual machine tools) on Windows. After creating a new virtual machine, change the network settings for this VM to Bridged Adapter (shows in Figure 2-10). Then, the installation of mitmproxy is done on this well-built Lubuntu platform.

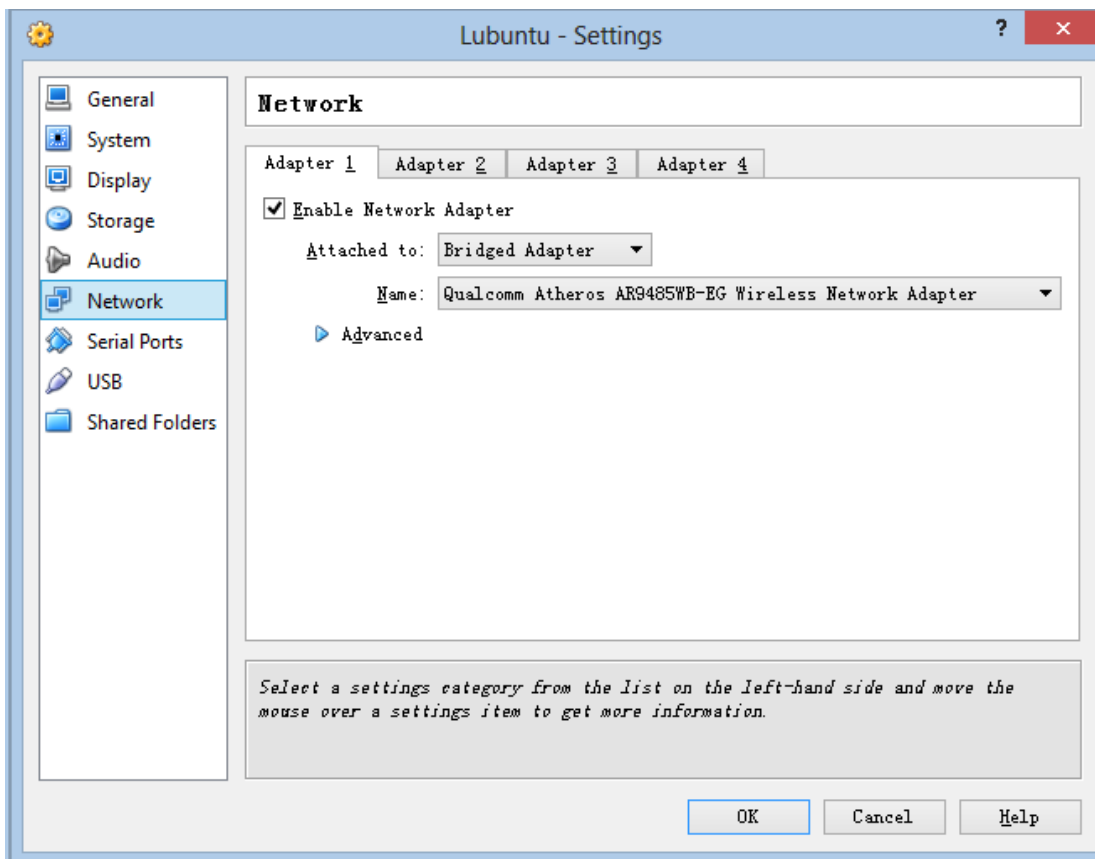


Figure 9 Lubuntu Platform Setting

In order to establish a connection between the Android and mitmproxy, the following steps are needed for the Android side. Figure 2-11 shows the first step that long-pressing on connected network on the Wi-Fi setting section. Figure 2-12 demonstrates that next step is checking the “Modify network config”. Figure 2-13 presents the last step, choosing “Show advanced options”, then “Manual”, filling in the IP address which is the Lubuntu’s IP address in the Proxy hostname and 8080 for Proxy port.

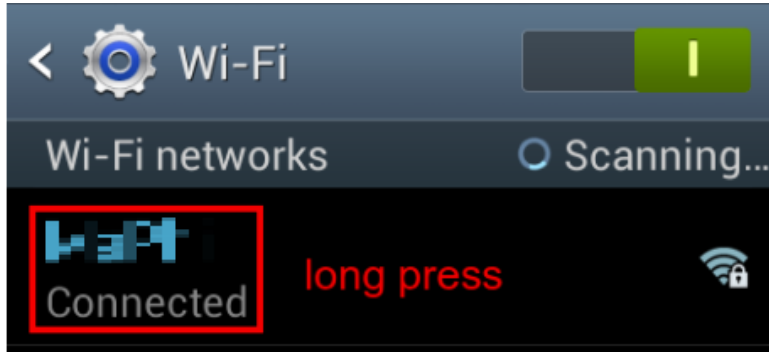


Figure 10 Setting up Proxy-1

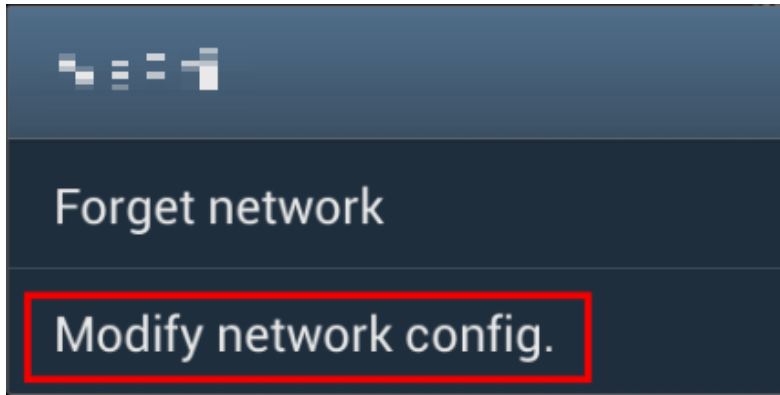


Figure 11 Setting up Proxy-2

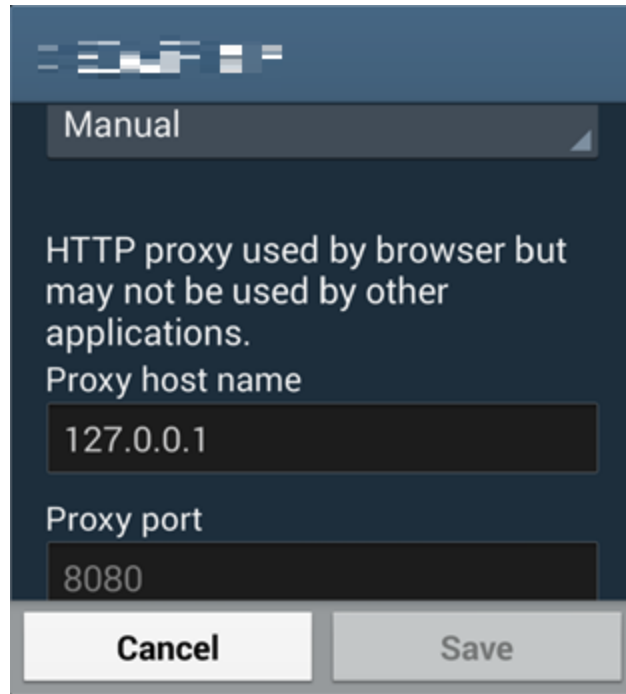


Figure 12 Setting up Proxy-3

- Set up the Android device

Since a real smartphone or an Android emulator can work as same as each other, the Android device setting up process is also same. For both of method, the preparation work is installing the targeted test application on Android. We describe this process on Android emulator as an example. The Android emulator requests Eclipse (or other tools, such as Android Studio) to lunch the Android emulator. The Figure 2-14 shows the interface of the emulator powered by Eclipse. The details are provided in the Appendices section with a whole instruction for this SSL implementation vulnerabilities validation experiment.

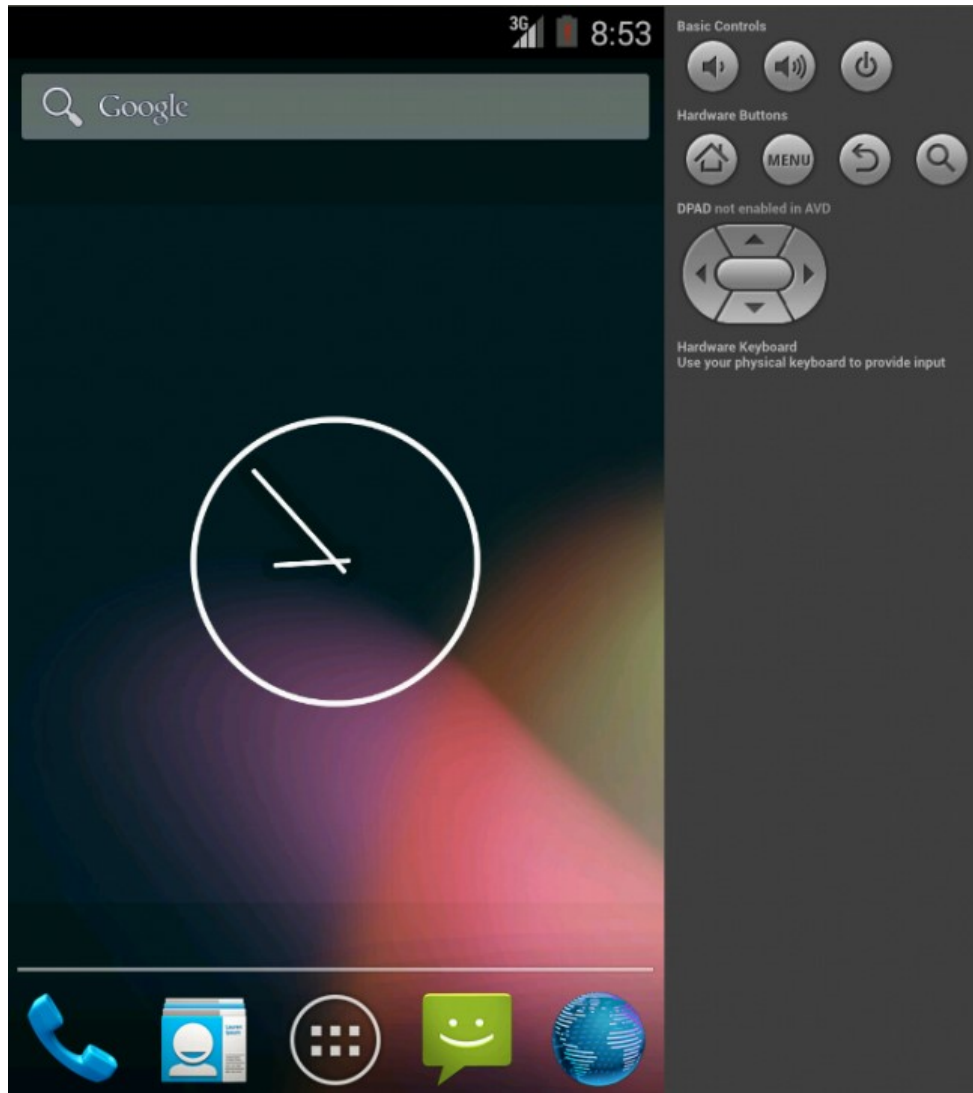


Figure 13 Android Emulator

Experiment Result

The Figure 2-15 shows the example result by testing an application called Skout which is a popular social media application in the Android application market. According to our validation experiment, Skout applies SSL to get secure communication. However, Skout do not check the source of certificate and accept the fraud certificate signed by mitmproxy. This result shows the Skout application is compromised by the MITM attack. As the attacker, the mitmproxy is able to get the secretive customer information directly. The red circled area presented in Figure 2-15

shows the login information (account name and the password). This experiment validates one of the SSL implementation vulnerabilities claimed in the previous research work [2][3].

```
<param0 i:type="d:string">test@126.com</param0>  
<param1 i:type="d:string">20142014</param1>  
<devicebrand i:type="d:string">generic</devicebrand>  
[1/72] [l:~b test] ?:help q:back [*:8080]
```

Figure 14 Account and Password Caught by mitmproxy

Chapter 3 WebView Attacks

3.1 Overview

WebView is a system component for the mobile operating system that allows apps to display content from the web directly inside an application. There are two ways to view web content on a device: through a traditional web browser or through an application includes WebView in the layout. If an application is designed to have browser functionality powered by the WebView library, an instance of a WebView class is created to run the browser function. After this step, a browser is embedded within the app. As a browser, it supports the application for loading web pages and executing JavaScript, etc. Moreover, WebView allows the app to interact with web pages and other web apps. As a powerful technology, WebView is used for two most popular smartphone platforms: Android and iOS. It should be pointed out that WebView is called UIWebView in iOS and WebView in Android. And they have totally same features. As [5] mentioned, the similar attacks and risks are achieved successfully on iOS. For simplification, we focus on the discussion about the Android WebView.

This section is composed of two parts: background and validation experiment. In 3.2, we explain WebView's security issues and attacks caused by vulnerabilities of WebView. In 3.3, we implement a validation experiment for the JavaScript injection attack. We conduct the lab experiment model for two different environments: smartphone or smartphone emulator. Those two options offer more flexibility for students. In this lab, we develop a third-party Facebook app with WebView to execute the JavaScript code injection attack. During this attack, the victim's Facebook account and password are collectable directly. This lab aims to help students have a better understanding of SSL implementation vulnerability, improve their self-learning and independent thinking ability, and acquire the critical thinking ability for their future study.

3.2 Background

3.2.1 WebView

WebView is a component that allows applications to render web content. As a part of the mobile platform, WebView is able to display web pages directly inside the application as an “embedded browser” that shows the information written in Web technology, including HTML5, JavaScript and CSS. As an essential component in the mobile system, it provides a simple but powerful browser to mobile or tablet applications. WebView was designed to display or process web contents on native apps. The web-browsing functionalities are packaged into a class, and they can be embedded into an app. By using this class, the app can have its own web browser. Due to the unknown safety issue of web content from an external source, WebView has a sandbox which was implemented inside like a traditional browser. That is, JavaScript code inside can only run in an isolated environment, and JavaScript code has no permission to access the devices sensors, cameras, files and another system resources, etc. However, the usage of WebView weaken web’s security infrastructure: trusted computing base and sandbox protect. For instance, one of the WebView’s API – `addJavaScriptInterface` enables web application’s JavaScript code to invoke Android application’s Java code; `loadUrl` allows Java code invoke JavaScript code. By using WebView, the application has the capability to interact with web content through WebView’s APIs in two ways: from app to web page, from web page to app.

3.2.2 Trusted Computing Base and Sandbox Protection

The trusted computing base (TCB) plays a critical role in a mobile system. In general, the trusted computing base means everything that can provide a secure environment, such as the operating system and security mechanism, hardware, network hardware, and software, etc. In this section, to simplification, we use this concept to clarify the potential risk when WebView is used as an embedded browser. Because the “embedded browser” is customized for mobile applications, those applications are able to provide more user-friendly and powerful experience than the normal web browser. For example, the Facebook, Twitter, Instagram and Chase Bank and so on. Thanks to WebView, many apps have significant advantages comparing with the web page. For instance, the customers of certain bank can deposit checks through a corresponding mobile application by taking the photos checks through mobile device’s camera. WebView provides many APIs to create

the bridge between the apps and the inside browser. However, the utility of WebView makes the TCB on the user side weak or even vulnerable. In a common situation, a typical web browser is a TCB for the client, such as Chrome, Firefox, Opera, etc. The reason why those browsers are TCBs is that they are tested comprehensively for a long time before they release, especially on the security side. Nevertheless, the embedded browser powered by WebView could not be concerned as a TCB. Because the developer may use WebView in an inappropriate or incorrect way when the developer has not enough time to take a security test or has incomprehensive knowledge background of mobile security. Therefore, TCB of the Web infrastructure weakened by WebView that could cause some security problems. The inappropriate utilization of WebView transform the interaction between web application and installed app into the interaction between web application and malicious browser.

Another security issue is sandbox protection model that applied by all browsers. It mitigates the security vulnerabilities which caused by running unknown risky JavaScript code in browsers. The sandbox model contains two parts: isolating webpages from system and enhancing Same-Origin Policy (SOP). The isolation guarantees that the web pages isolate from the system resources; SOP means that a web browser ensures only scripts in a first web page from same origin condition. That is, SOP aims to isolate one origins webpage from other origins webpages. However, WebView changes the security condition. For example, an API called `addjavascriptInterface` make it possible for web pages form one origin to affect other origins webpages. By this API, an interface become a global interface and creates holes for the sandboxes. Because it permits JavaScript code to access system resource by any pages loaded in the WebView. Above all, the app's developer who uses WebView in an inappropriate way may case critical vulnerability. Attackers take advantage of WebView vulnerabilities and design many attacks.

3.2.3 Two Type of Attacks

The attacks caused by WebView vulnerabilities are categorized into two types. The first one is the attacks from malicious Web pages. This attack relies on the WebView's mechanism that JavaScript codes invoke Android application's native java code (JavaScript-to-Java interaction). Namely, a Java object could be registered by WebView API, and then the JavaScript code can

invoke all the public functions or methods in this Java object in an Android application with WebView. For example, the attacker launches an attack by using a malicious web page and steals the victims' sensitive information in their mobile device, including contact information, SMS information, photos, GPS information, and files, etc. Another type of attacks is triggered by malicious applications. These attacks executed by invoking JavaScript from Java (Java-to-JavaScript interaction). One of this attack's mechanisms is supported by the loadUrl API. In common scenario, the apps using this feature provide a more plentiful experience than a normal web application. The validation experiment design for WebView attack topic aims to realize the second type attack in Section 3.3. It is should be pointed out that SQL injection attack, cross-site forgery attack, and cross-site scripting attack do not belong to either of two attacks shows in the figure. Because those attacks also happen in other situations. And the usage of WebView do not aggravate or mitigate those attacks' conditions. Therefore, those attacks are not discussed in this section.

Java-to-JavaScript Attacks

The Figure 3-4 shows how malicious apps attack webpage. In the Java-to-JavaScript attack, the attacker executes the malicious native Java code to the victim webpage directly. A prerequisite for launching attack is that the attacker must develop or control a malicious application designed for a specific web application, such as Facebook, Twitter and so on. Namely, Java-to-JavaScript attack could only be launched when the victim users using third-party mobile applications. According to the statement of [4], there are a number of third-party applications for an intended web application, such as Facebook, Twitter and Instagram, etc. in the Android applications market. For instance, there is a great third-party app for Facebook called FriendCaster developed by Handmark. Even though there is no malicious third-party app be found in the application market yet, it is possible that attackers still are working on launching this type attack in the near future.

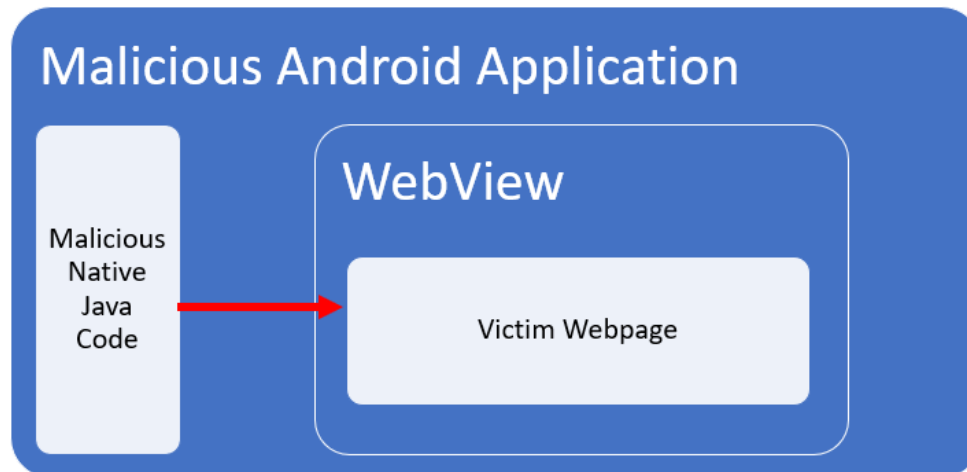


Figure 15 Java-to-JavaScript Attack

JavaScript-to-Java Attacks

The Figure 3-5 shows malicious web pages attacks Android applications. The assumption for this attack is that the mobile apps are designed to serve a specific web application without any malicious purpose. The goal of the attack is compromising the apps and their intended web application. Generally, the victim applications are developed from the owner of the intended web applications or some independent entity. Namely, the first-party apps and the third-party apps may under the attack from malicious web pages. The first step to realize this attack is that allure the victim users run the malicious web code into the applications by some methods, such as social networks, advertisements, contact information and so on.

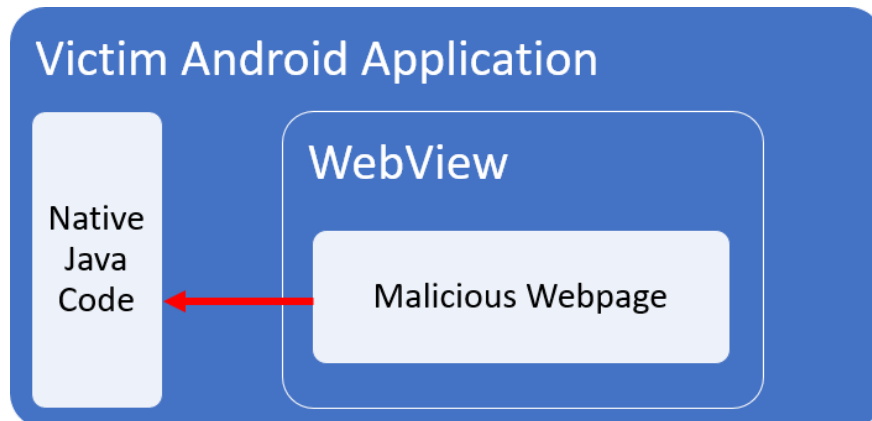


Figure 16 JavaScript-to-Java Attack

3.2.4 JavaScript Injection Attack

JavaScript injection attack is a typical Java-to-JavaScript attack on WebView. Through this attack, the attacker injects malicious JavaScript code to any targeted web application or websites rendered by WebView. One of the WebView's API—*loadUrl* plays an important role in JavaScript Injection attack. *loadUrl* are used to load specified URL in WebView component. The specific URL can be any string defined by the attacker. Since those certain URL and target website have the same privilege, those certain JavaScript code can access and operate the webpage's DOM tree, cookies, etc. The following example code shows a JavaScript injection attack. The contents in the *sb* string are injected into WebView as JavaScript injection code.

```

1 public void onPageFinished(WebView view, String url)
2     {
3         super.onPageFinished(view, url);
4         StringBuilder sb = new StringBuilder();
5         sb.append("document.getElementsByTagName('form')[0].onsubmit =
6             function () {");
7         sb.append("var objPWD, objAccount;var str = '';");
8         sb.append("var inputs = document.getElementsByTagName('input');");
9         sb.append(".....");
10        view.loadUrl("javascript:" + sb.toString());
11    }

```

3.3 Validation Experiments Design

3.3.1 Motivation

WebView is a subclass of View, and it is used to display web pages. WebView enables smartphone and tablet (both in Android & iOS) apps to embed a simple but powerful browser inside them. It allows apps to interact with the Web content through its APIs by two directions: from apps to web pages-invoke Java from JavaScript; from web pages to apps-invoke JavaScript from Java. Since WebView weakens Web's security infrastructures, the improper usage of WebView by the app's developer can cause many vulnerabilities. In order to demo and validate the WebView vulnerabilities, we design and develop this experiment. This experiment can also help students who learn from a lecture or by themselves understand the WebView vulnerability. Furthermore, by doing this experiment, students can have a better understanding on WebView attack. There are three design objectives: 1) Enhancing the basic principles and concepts through learning the process, such as WebView's API that causes vulnerabilities, methods to launch attack from malicious apps, etc.; 2) Helping students to gain first-hand experience of WebView vulnerability by putting what they have learned about the vulnerability from class into real attack; 3) Inspiring students to explore more WebView related security problems by guiding them implement this lab experiment step-by-step.

3.3.2 Lab Design

The object of this section is the validation experiment for JavaScript injection attack in detail. We conduct this lab to implement one of the Java-to-JavaScript attack – JavaScript code injection attack. In order to implement this experiment, we develop a malicious Android application which inject malicious JavaScript code to target victim website and extract information. The Figure 3-6 shows our experiment structure and components. Since the malicious Android application contains malicious Java code, the attacker who owns this malicious Android application can execute attack to extract victim's sensitive information through the WebView's vulnerability. Because of the mixture of code and data in WebView, the injected malicious code has the capability to be loaded in WebView. After executing the malicious code, the attacker launch attack successfully and get the victim's sensitive information. Specifically, we design the malicious apps as a third-party Facebook app, attacker is able to extract the user's account and password information.

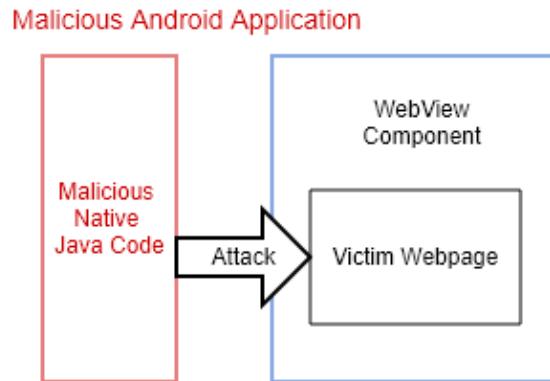


Figure 17 JavaScript Injection Attack

3.3.3 Tools for the experiment

In this lab, an Android platform and an Android development software are needed. Since this experiment can work on both Android device and Android simulator, there is no solid device requirement that provides more flexibility to implement this validation experiment. Both provide the same result. Table 3-1 illustrates the detail version information of Android device, Android virtual machine and Android development software. Note that the Android development software information listed in Table 3-1 is a reference. Any other tools for Android development are acceptable.

ANDROID DEVICE	
Operating system	Android
Version	8.0
ANDROID VIRTUAL MACHINE	
Operating system	Android
Version	8.0
DEVELOPMENT SOFTWARE	
Name	Eclipse
Version	4.7
Description	Develop and debug the Android application; Provide the Android Virtual Machine

Table 3-1 Configuration of Lab Tools

3.3.4 Experiment Implementation

Experiment Setup

This lab is divided into two parts. 1) The first part is that development of a normal WebView application which can log in to Facebook like a common third-party application. 2) The second part is filling the malicious code to the application that developed in the first step to execute JavaScript code injection attack. After importing project into Eclipse, the code this application is readable and changeable. The following example code showed below is the malicious code which lunches the injection attack by using WebView's API *loadUrl*. The detailed experiment instruction is attached in the Appendix section.

```
@Override
public void onPageFinished(WebView view, String url) {
    super.onPageFinished(view, url);

    StringBuilder sb = new StringBuilder();
    sb.append("document.getElementsByTagName('form')[0].onsubmit = function () {");
    sb.append("var objPWD, objAccount;var str = '';");
    sb.append("var inputs = document.getElementsByTagName('input');");
    sb.append("for (var i = 0; i < inputs.length; i++) {");
    sb.append("if (inputs[i].type.toLowerCase() === 'password') {objPWD = inputs[i];}");
    sb.append("else if (inputs[i].name.toLowerCase() === 'email') {objAccount = inputs[i];}");
    sb.append("}");
    sb.append("if (objAccount != null) {str += 'Username: '+objAccount.value;}");
    sb.append("if (objPWD != null) { str += ' , Password: ' + objPWD.value;}");
    sb.append("window.MYOBJECT.processHTML(str);");
    sb.append("return true;");
    sb.append("};");

    view.loadUrl("javascript:" + sb.toString());
}
```

In the above example code, the *sb* strings is used to execute attacks on the targeted web application as JavaScript code within the web page. In the meantime, those codes are capable to extract the user's sensitive information on the targeted web application, such as user's Facebook account and password information.

Experiment Result

The following figures show that the malicious app launching an attack to extract user's Facebook account and password information. It is should be pointed out that we did not let the

malicious app send that information to anyone but show a warning window for education purpose. In a real attack, it may be sent to the attacker or used other ways.

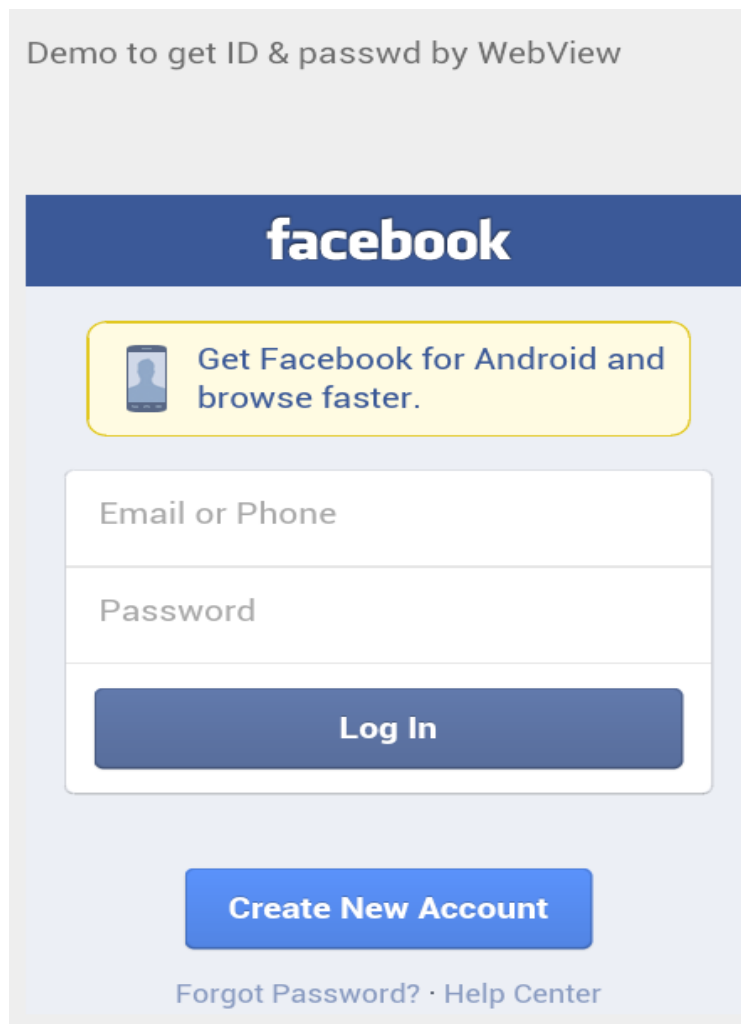


Figure 18 Interface of the Malicious App

- 1) User type in the personal information

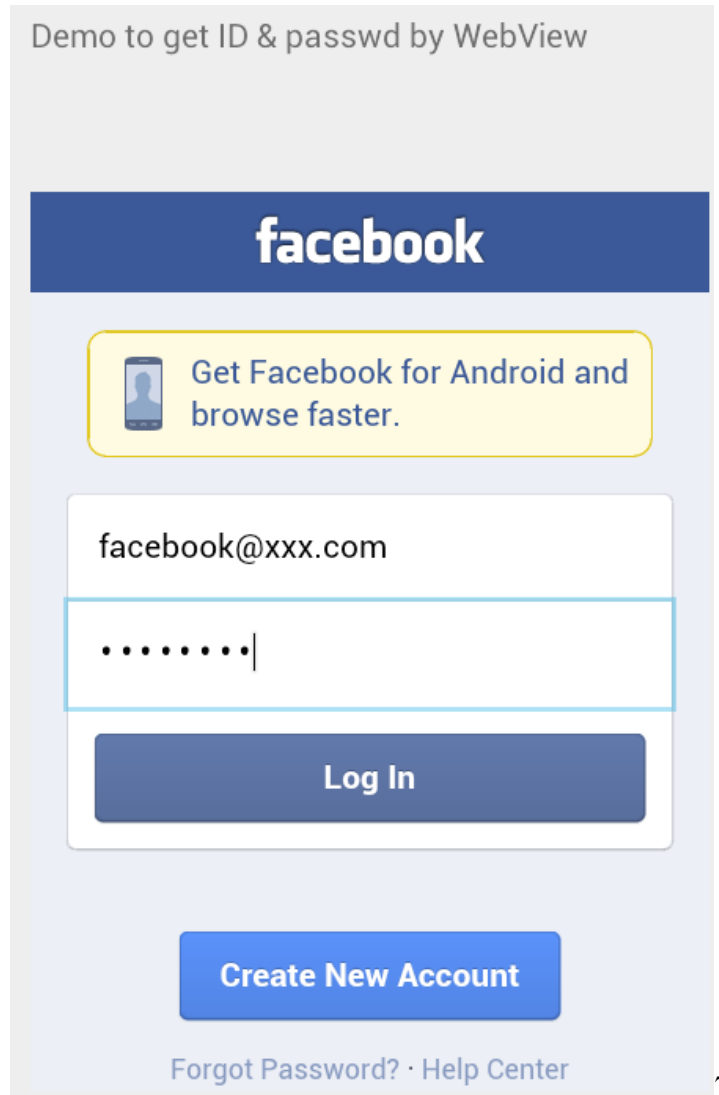


Figure 19 Filling in User's Information

2) Capture the information

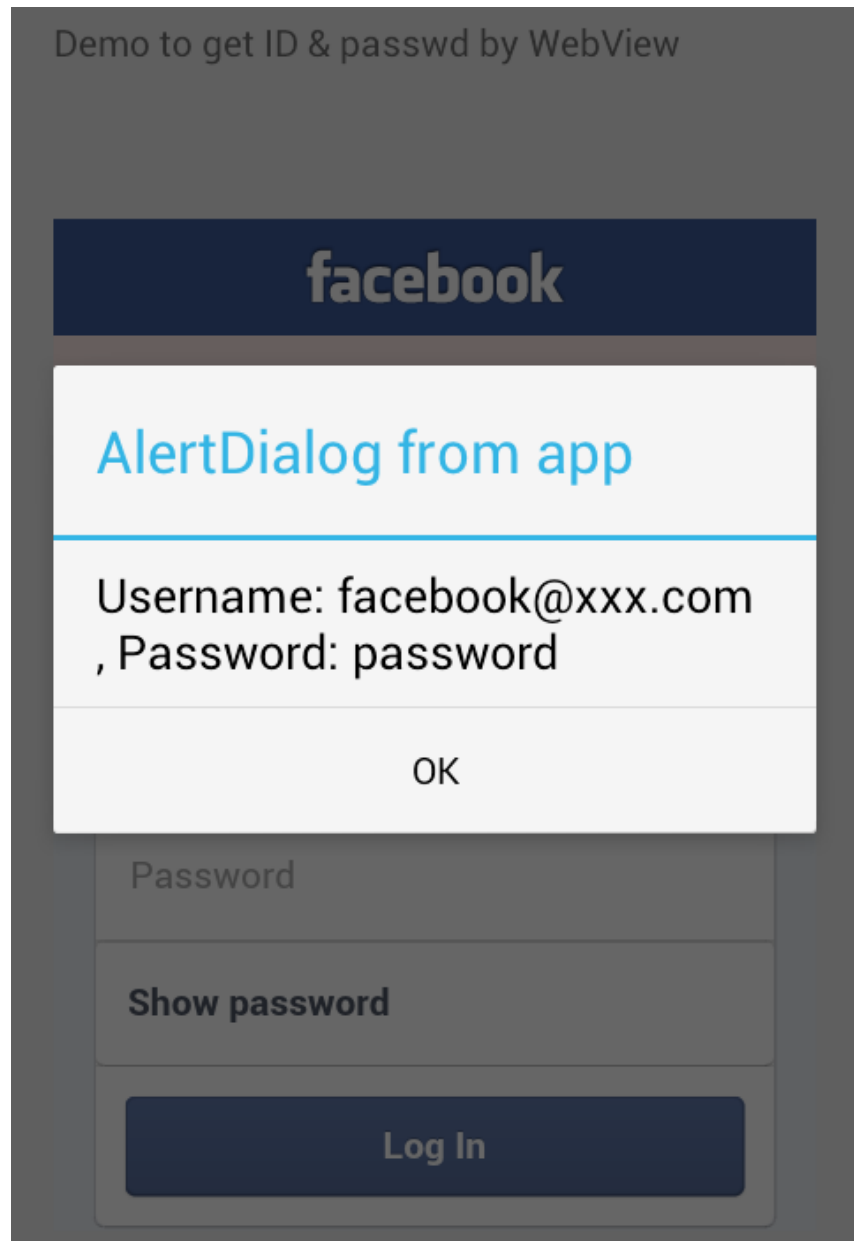


Figure 20 User's Information Caught by Attacker

Chapter 4 HTML5-Based Application

4.1 Overview

It is a noticeable tendency that smart phones have become more and more popular in the worldwide. In the meantime, there are abundant kinds of applications which are implemented to run on the different mobile platforms. As a hybrid app, a HTML5-based mobile app is easily developed for different mobile systems comparing with the native app. HTML5-based mobile app is getting very prevalent. HTML5-based mobile apps are easily developed by using standard web technologies, including HTML5, JavaScript, and CSS, combining with middleware, such as PhoneGap, Appcelerator and RhoMobile. However, HTML5-based apps are suffering from the attacks that are inherited by using web technologies. According to the prior research [48], Cross-Site Scripting(XSS) attack is a typical attack method for the situation using HTML5. Unfortunately, HTML5-based mobile apps inherit this shortage. Namely, due to the usage of Web technologies and the particularity of mobile applications, HTML5-based mobile apps allow data and code to mixed together. It is should be pointed out that there are many other attacks for HTML5-based applications. In this section, we introduce a novel code injection attack. Comparing with traditional XSS attack, this code injection attack can initiate attacks through many injection channels, such as Barcode, MP3, SMS, Contact, etc. That means this code injection attack for HTML5-based app have the possibility to cause more damages and affect all major mobile platforms, including Android, iOS, Windows Phone, etc.

This section is composed of two parts: background and validation experiment. In 4.2, we provide the background and discussion covering from WebView, PhoneGap to the code injection attacks analysis [6][7]. In 4.3, we conduct a validation experiment for the code injection attacks through MP3 attack channel. More specifically, a lab experiment model is built to run in two different environments: smartphone or smartphone emulator. By using an emulator instead of a

real phone, the requirement of building experiment environment is reduced. In this lab, we develop a MP3 HTML5-based application to test the attack process. The attacker is able to achieve victim's contact information during this attack. This lab aims to help students have a better understanding of HTML5-based app vulnerability. Moreover, it helps students to improve their self-learning and independent thinking ability and acquire the critical thinking ability for their future study.

4.2 Background

4.2.1 HTML5 and Hybrid App

Mobile phones, particularly Internet-enabled smartphones, are in widespread use. Mobile phones are prevalent among multiple consumers. A report from [49] in 2016, 87 percent of the U.S. population ages 18 and above owned or had regular access to a mobile phone. And Seventy-seven percent of mobile phones are smartphones(Internet-abled), up from 71 percent in 2015 and 61 percent in 2013. All of those are reasons why there are plenty of developer start to mobile application development. Obviously, different versions of the application may be developed because of the various mobile platforms. For instance, the mobile applications for Android and iOS are developed by Java and Object C respectively. Fortunately, HTML5-based techniques break the limitation of the various system. Because the standard web technology could be supported by all mainstream mobile platforms, it is possible that developers just needs one-time development for HTML5-based applications for all smartphone platforms by using middleware. Essentially, the applications developed by HTML5-based techniques have the advantage to be simply transformed from one platform to another. Since HTML5-based applications built by HTML5, JavaScript, and CSS or another stand web technologies, HTML5-based apps are quite different from native applications. Even though HTML5-based application could overcome the limitation of cross mobile platform problem, it may lead to the code injection attacks through different channels, for example, QR code, SMS, MP3 file and so on. According to a survey by Evans Data shows that among the 1,200 surveyed developers,75% are using HTML5 for application development. [35] This code injection attack for HTML5-based apps are needed to pay more attention.

As HTML5-based app developers, they have two missions: web code part and native code part. The developers need to use JavaScript and other web technologies to develop the web part. For the native code part, developers have two choices. The first one is writing native code by themselves. Another is using middleware provided by the third party. In general, developers choose the second method. Because the third-party middleware provider is responsible for portability which is higher than the native code written by certain developer. By using those middleware, a variety of mobile platforms are supported. Moreover, middleware makes the application turn to a “Hybrid” application. The native app, web app, and hybrid app are three main types of apps in the mobile application market. The hybrid app means that it has both characteristics of the native app and web app.

- Native app is coded in a specific programming language for a certain operating system, such as Objective-C for iOS and Java for Android. Namely, a completed native app could not work cross-platform. Figure 3-1 illustrates the relationship between native app, device and operating system. Since the native app interacts with the mobile device and operating system directly, it provides the fastest, most responsive and most reliable user experience compare with the other two types apps. And the disadvantages of this native app are higher costing during the developing process, poor compatibility ability for different platform.
- Web app relies on the web browser. As a “mobile version” of a specific website, a web app is developed by JavaScript, CSS, HTML5 or other languages even need not access to software development kit (SDK). The development budget is also reduced in this web app condition. Figure 3-2 shows the relationship between web app, web browser, device and operating system. The weakness of web applications is obvious: the user experience poorer than the native apps; they have no ability to connect with the mobile hardware directly.
- Hybrid app is a combination of native app and web app. It combines the advantages of both type apps and discards the shortcomings. For example, a hybrid app is easier to develop and maintain on different platforms than native app; it can interact with mobile device hardware through the native parts of the app, etc. A hybrid app comprises

WebView, native UI code and native hardware access code. Figure 3-3 indicates the relationship between hybrid app, device and operating system. As an essential part of “Hybrid” app, WebView provides great performance for the hybrid app.

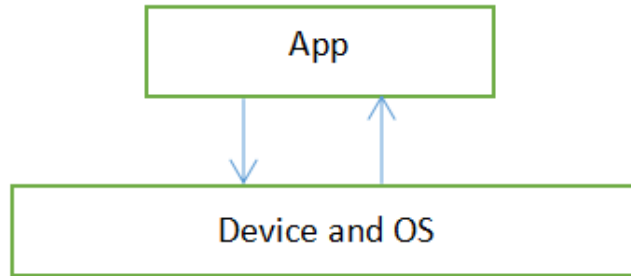


Figure 21 Naive App

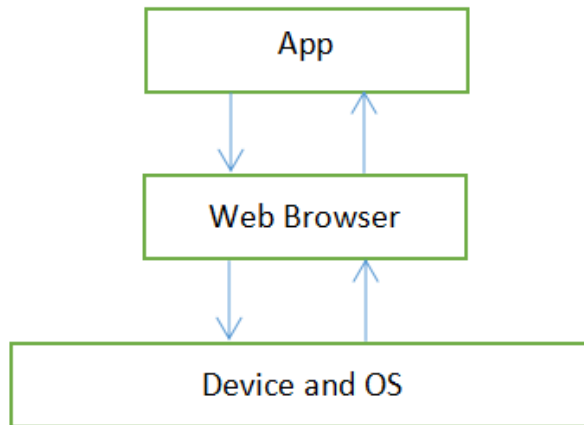


Figure 22 Web App

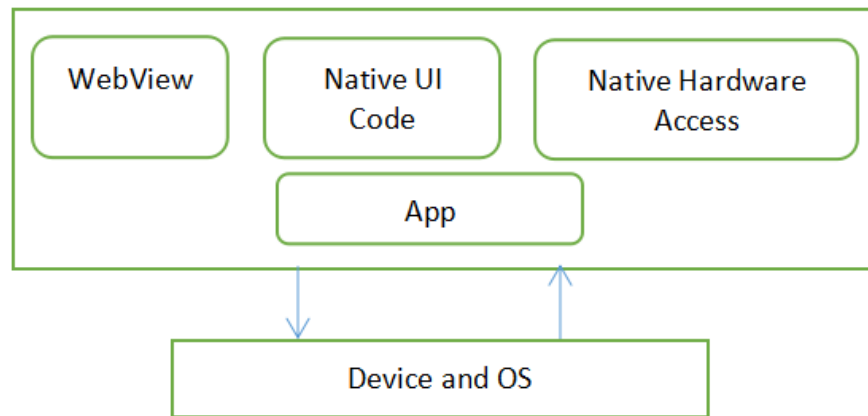


Figure 23 Naive App

For HTML5-based application’s developers, they can use the middleware to reduce the development process instead of development for each platform. There are several third party middleware which are well established, such as PhoneGap Appcelerator and RhoMobile [36] [37] [38]. In this section, we focus on PhoneGap which is the most popular third-party middleware in the current mobile application market.

4.2.2 PhoneGap

PhoneGap is a software development framework for Adobe System that is used to develop mobile applications for various mainstream mobile operating systems. The advantage of PhoneGap is obvious. The developer who use PhoneGap to develop applications does not require to have the knowledge of mobile programming native languages, such as Java and Objective-C, instead of web development languages, including HTML, CSS, and JavaScript. Many well-recognized mobile operating systems are available in the market, regardless it is open-source and proprietary. The market [39] is shared by Android, iOS, BlackBerry, Windows, and others, and the percentage of occupancy is 52%, 40%, 3%, 2% and 3% respectively. Because each mobile platform provides their special tools and environments for developers, once the developers tend to cover all major mobile operating systems for the high reachability for their users, they need to develop their application for different platforms multiple times. Expect the tedious develop process for each mobile operating system, the developer must understand platforms, tools and background

knowledge for each of operating system. As a third party and open source middleware, PhoneGap is a great solution for the problem which mentioned before.

As a middleware framework, PhoneGap can be used to developed mobile applications through web technologies, such as HTML5, JavaScript, and CSS. By using PhoneGap, an application could be easily transformed from one platform to another, as long as the second platform is supported by PhoneGap. By the utilization of PhoneGap, the developer of application only need to use standard web APIs. The other parts of the application will be finished by PhoneGap, such as the interface of the app and portability for various mobile operating systems. There is a WebView instance embedded in the PhoneGap framework by default. And the HTML5 pages or JavaScript code will depend on the WebView to execute.

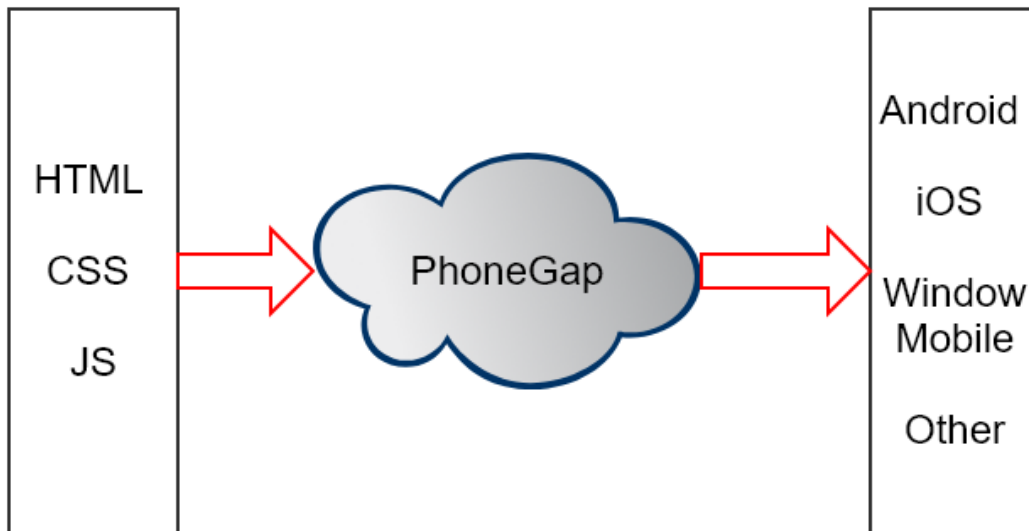


Figure 24 The Usage of PhoneGap

PhoneGap Framework is built in two parts as shown in Figure 4-2. One is the bridge part; another is the plugin part including camera, contact, SMS, media and Wi-Fi, etc. The bridge part is in charge of the connection between JavaScript code and Java code. The function of this bridge part is achieved by “Cordova” which allows the JavaScript code inside WebView to interact with the Java code outside. For the plug-in part, those plugins give the permission to allow HTML5-based applications to access the mobile device’s resources directly, including Contacts, SMS,

Camera, etc. Because those plugins are written in native code, the developer have the option to customize their own plugins depending on demands. It is should be noted that the portability of those plugins is lower than plugins provided by middleware. For example, PhoneGap offers 16 official plugins for applications to use directly [6]. By using those official plugins, the portability issues and other related issues including maintaining are handled by PhoneGap.

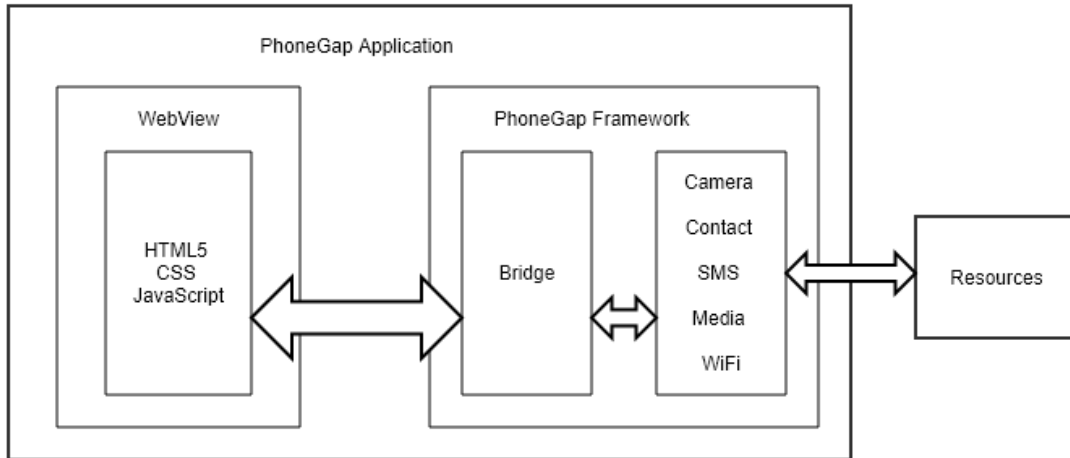


Figure 25 Structure of PhoneGap

4.2.3 The Code Injection Attack

Due to the emerging of the high occupation of a smartphone, there are a large amount of developers for the smartphone’s applications. The more developers lead to more diversity and more possibilities for the application’s market, however, the quality and security of applications are hard to guarantee. Those developers who may only have the programming training or learned by themselves are easy to use this technology careless, then led to explore under risks and malicious attacks. According to the research on resent years [41] [42], a dangerous feature of web theologies is the permission for a mixture of data and code. The designers of this feature aim to build an interaction between the native code and JavaScript code, and then allow JavaScript code adding or embedding to HTML pages. Namely, this feature allows the code processed by the web technologies together with data. In this situation, the code part in the mixture is able to be identified and executed in the JavaScript engine inside of the application. Attackers can take advantage of

this feature to execute attacks, such as code injection attack. It is obvious that the web technologies applied to mobile applications developments lead to many advantages, such as lower budget, high portability on cross-platforms, etc. However, like a coin has two sides, for the HTML5-based applications which based on web technologies, those applications are subjected to a new form of code injection attack through many channels. It should be pointed out that XSS is a typical code injection attack but not new. The OWASP top-ten list [40] provides the information that, XSS is the third most common security risk in mobile applications. For typical XSS attack, the channels are the Internet, Wi-Fi, and Bluetooth, etc. Comparing with those channels, code injection attack for HTML5-based app is a new division method [6]. Since HTML5-based applications must connect with the outside resources, such as users, environment and other devices, the data used to interact with HTML5-based applications can be used to launch the attack. For example, RFID tags, media files, 2D barcode, the ID field of Bluetooth devices and Wi-Fi access points are the new unique channels for this code injection attack. Namely, the attacker has the opportunity to execute an attack by injecting malicious code through those smartphone unique channels. For the HTML5-based apps' users, some normal behavior on the mobile device in daily become risky and dangerous. For instance, listening to music from a MP3 player that displays the name, album and lyric of the songs; scanning a QR code to make a payment; searching and connecting a new Wi-Fi provider, receiving a SMS message, etc. The behaviors mentioned above can be used to launch code injection through new channels provided by middleware, including contacts, camera, Wi-Fi, NFC and media.

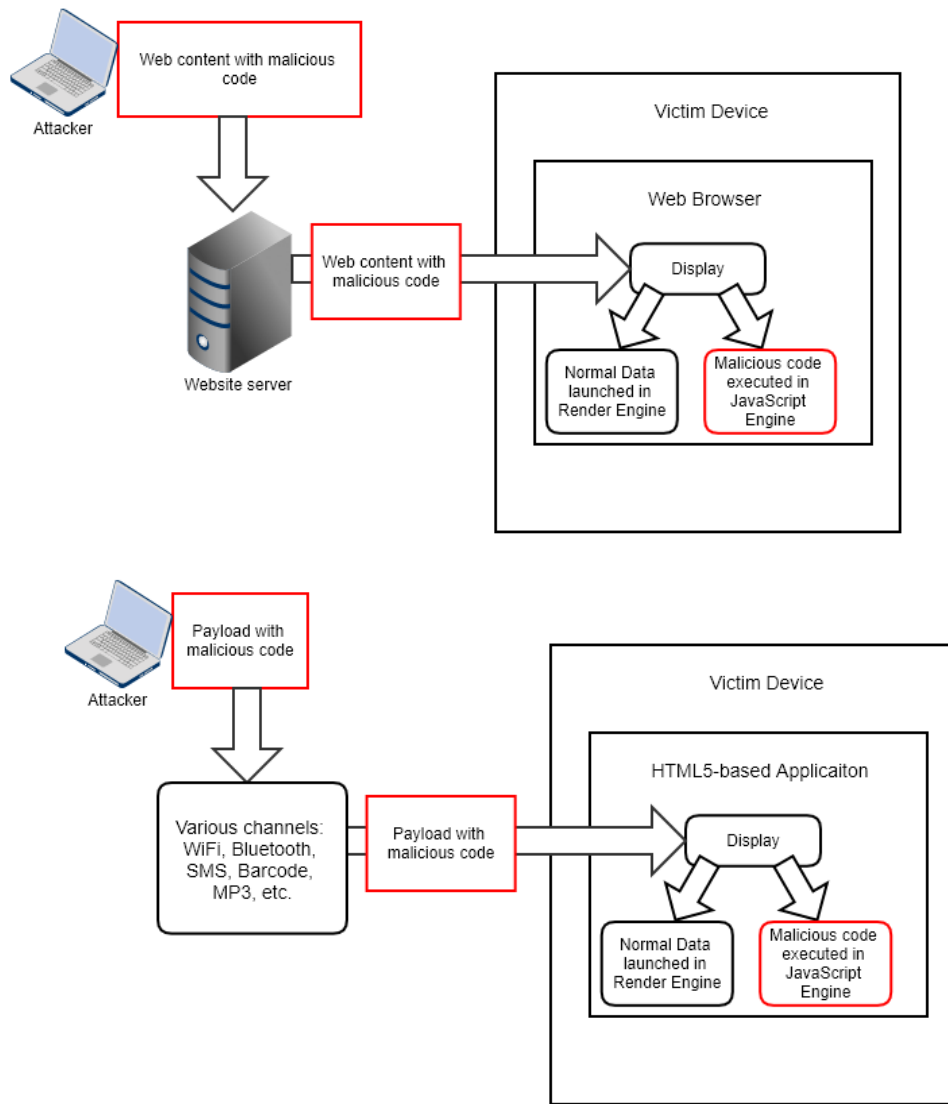


Figure 26 XSS attack and Code Injection Attack

XSS attack needs attacker to inject malicious code to web content and initiate the attack through the web browser channel. As shown in Figure 4-3, after the attack, the victim's web browser executes malicious code in JavaScript engine instead of displaying the normal data launched in render engine. Comparing with XSS, this code injection attack cause more damages for victims through more channels. The attacker has multiple choice provided by PhoneGap to inject malicious code, includes Wi-Fi, Bluetooth, SMS, barcode, MP3 file. And the victim HTML5-based application executes the injected malicious code instead of normal web data in JavaScript engine. Obviously, this code injection attack could not be launched on the native mobile

application which written in any native language, including Java, Objective-C, etc. Because there are no opportunities for the mixture of code and data. Essentially, once the web technologies are utilized in a mobile application, this application may have the vulnerability caused by code injection attack. In the following section, to explain the code injection attack idea clearly, we illustrate the new channels for code injection attack on the HTML5-based application. After that part, the worse damages caused by this code injection attack will be discussed.

The Code Injection Attack Channels

Even through HTML5-based technology provides high portability on cross-platforms and lower budget for HTML5-based application's developments, the feature of a mixture of data and code together cause vulnerabilities for those apps. It is noticed that this threat caused by the design, namely, the JavaScript code are able to add and execute inside HTML web pages. For example, in the typical cross-site scripting (XSS) attack, the malicious code is able to inject and execute the victim application. The damage of this XSS attack is that the privilege of the victim application can be stolen by the attacker. Moreover, all the malicious (XSS) attack actions are launched in a web browser through web technologies. However, except the web channel, this code injection attack for the HTML5-based application can be executed through many other channels.

Generally, the code injection channels belong to three categories [7][8]- ID channels, data channels unique to mobile devices and metadata channels in media. A number of channels can be used to execute this attack in those categories. For example, ID channels can detail to the Wi-Fi access point, Bluetooth, Barcode, NFC, and Text Extraction belongs to data channels unique to mobile devices category; the metadata channels category contains two main types- one is for MP3, MP4 and images, another is FM Radio. Due to the limitation of this thesis, we only pick one channel for descript and validation.

The Code Injection Attack Damage

Due to the uniqueness of the HTML5-based application, more privileges are given comparing with the web applications, including contact, camera, and Wi-Fi, etc. THE code injection attack

on HTML5-based application cause worse damages. There are three potential damages [7]. The first one is that malicious code is able to attack the device directly by using the “Bridges”. That is, after the installation of the PhoneGap plugins and HTML5 APIs in the application, the “Bridges” is established for JavaScript code interact with device resource directly. The attacker takes the opportunities to add malicious code to plugins to launch an attack. Secondly, the malicious code can be injected into other vulnerable PhoneGap apps on the same device. Thirdly, the injected malicious code can turn the compromised device into an attacking device.

4.3 Validation Experiment Design

4.3.1 Motivation

Many app applications are required to support different mobile platforms. Comparing with the native apps, the HTML5-based mobile apps are developed by the HTML5 technology, which is platform agnostic. It is understandable that all mobile operating systems access the Web by supporting Web technology. Therefore, due to this advantage of HTML5-based mobile apps, HTML5-based mobile applications is getting very prevalent. HTML5-based mobile apps could be easily developed by using the standard web technologies, such as HTML5, JavaScript and CSS, and native code, such as Java. For the better portability of apps, the majority of developers use middleware as native code container to developer HTML5-based apps. A number of middleware have been developed, including PhoneGap, Appcelerator, RhoMobile and so on. By using this web technologies and middleware, the developers can develop an app one time for multiple mobile operating systems. However, the inappropriate usage of middleware leads to vulnerabilities. As the most used one, PhoneGap is discussed and utilized in this lab. Moreover, comparing with XSS attack, the code injection attack for HTML5-based mobile apps are damaged worse through more channels, such as MP3, WIFI, 2D barcode, and SMS, etc. We design a malicious application (MP3 Player) which could steal the contact information which stored in the smartphone, such as, name and phone number by code injection attack. This experiment can guide students who learn from a lecture or by themselves understand the HTML5-based app vulnerability. There are three design objectives: 1) Enhancing the basic principles and concepts through learning the process, such as HTML5-based app vulnerability, methods and channels to launch code injection attack, etc.; 2) Helping students to gain first-hand experience of HTML5-based app vulnerability by putting what

they have learned about the vulnerability from class into real attack; 3) Inspiring students to explore more HTML5-based apps security problems by guiding them implement this lab experiment step-by-step.

4.3.2 Lab Design

The design of this lab is that launching a code injection attack for HTML5-based apps developed by PhoneGap. We develop an example HTML5-based MP3 player app to validate the HTML5-based app's vulnerability through one the attack channels. In a normal situation, the MP3 player is not able to get the contact information when it plays the MP3 files. However, in the case of malicious apps, it can get user's contact information which stored on the mobile device. By injecting malicious code to the MP3 player, the injecting malicious code can execute attack in JavaScript Engine to get victim's contact information or other sensitive information. In this lab, we offer the whole project with code to students. And we require students extract victim's other information based on example, including GPS, other files, and photos, etc.

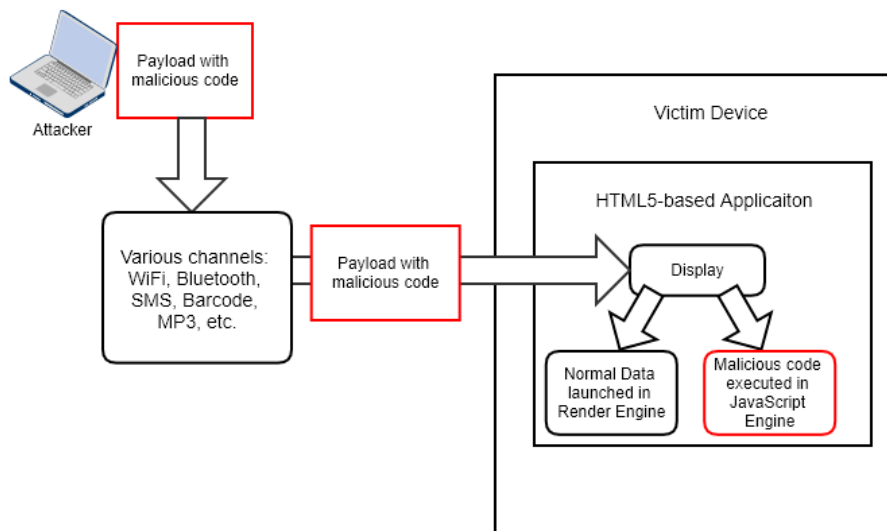


Figure 27 Code Injection Attack for HTML5-based App

4.3.3 Tools for the experiment

In this lab, the requirement of tools are an Android platform and an Android development software. There is no solid device requirement for the Android platform that provides more flexibility to implement this validation experiment. Because this experiment can work on Android device as well as Android simulator. Table 4-1 identifies the detail version information of Android device, Android virtual machine and Android development software. It is noted that the Android development software information listed below is only for reference. Any other tools for Android development are optional.

ANDROID DEVICE	
Operating system	Android
Version	7.0 (or lower)
ANDROID VIRTUAL MACHINE	
Operating system	Android
Version	7.1.1
DEVELOPMENT SOFTWARE	
Name	PhoneGap
Version	5.1.1
Description	Native Container
Name	Eclipse
Version	4.7
Description	Develop and debug the Android application; Provide the Android Virtual Machine

Table 4-1 Configuration of Lab Tools

4.3.4 Experiment Implementation

Experiment Setup

This experiment consists of two parts. 1)The first part is that execute the attack by launching the provided HTML5-based MP3 player. 2) The second part is modifying the malicious code to get more victim's critical information. After installing the example .apk file on Android system, the MP3 player are able to execute attack by playing MP3 file. The Figure 4-6 illustrates the contact information in the test mobile device. And the Figure 4-7 is the detail property information for the MP3 file which will be used to launch an attack in this validation experiment part. The album artist

information is the malicious code which will trigger the code injection attack on HTML5-based mobile application through MP3 file channel.

Experiment Result

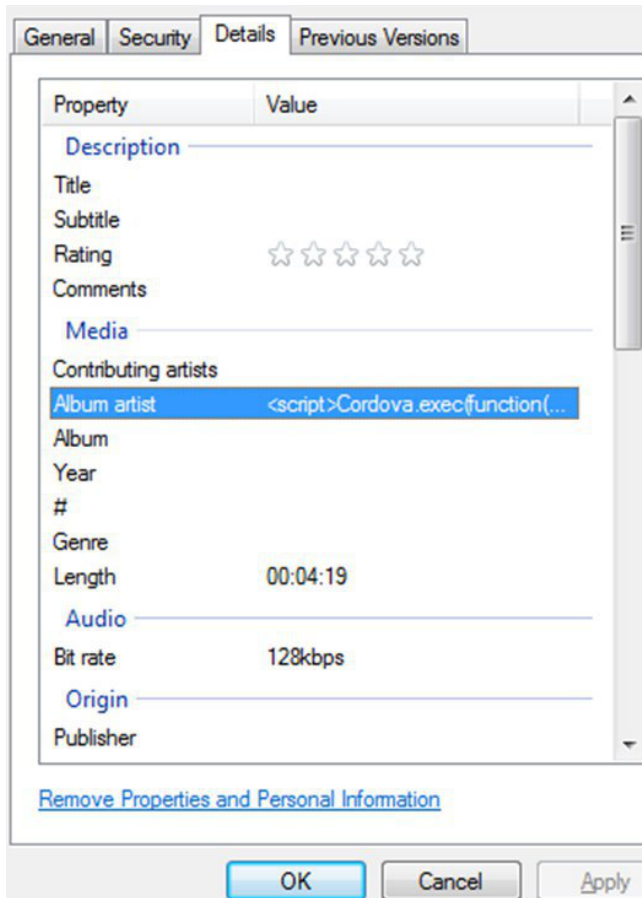


Figure 28 MP3 File Properties

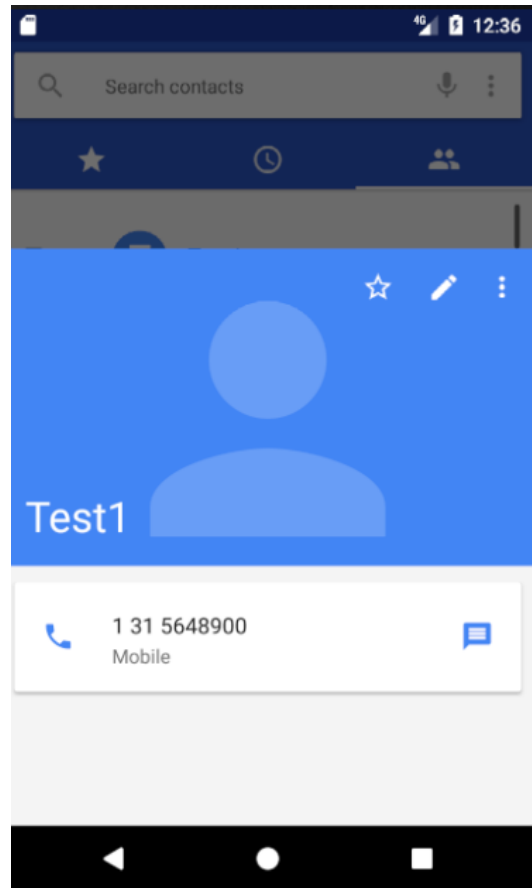


Figure 29 Example Contact Information

The following images demonstrate the malicious application execute an attack by playing an MP3 file to extract victim's contact information in the mobile device. Figure 4-8 shows the attack launched successfully. The victim's contact information is caught by popping an alarm window. It is noted that we did not allow the attacker send or share that sensitive information to

anyone. However, the victim's sensitive information caught by attacker might be used in many ways.

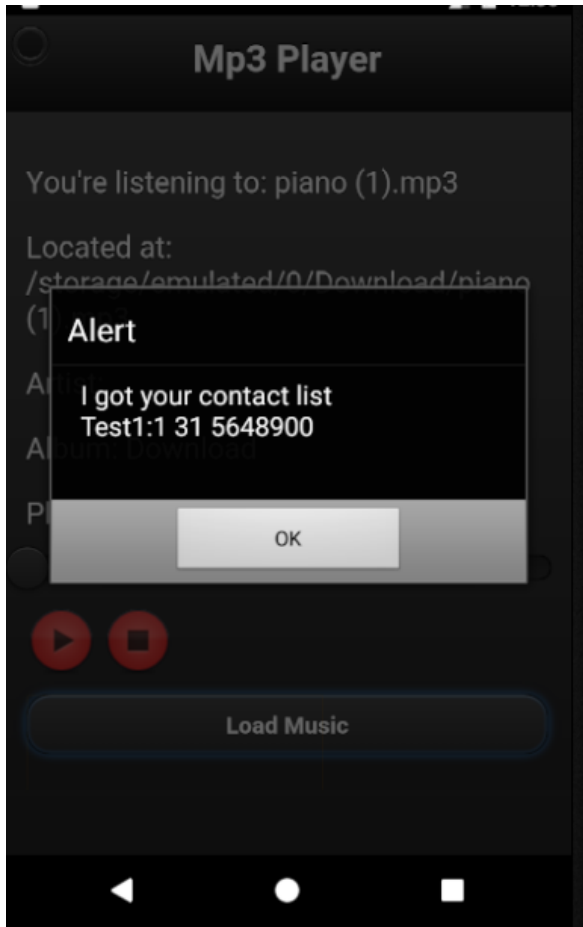


Figure 31 Attack Result

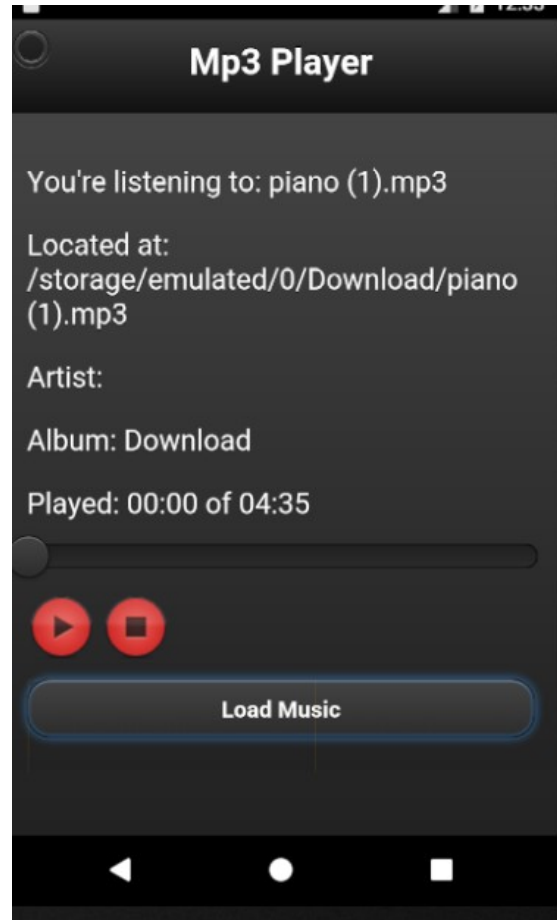


Figure 30 Example MP3 Player Interface

Chapter 5 CONCLUSION AND FUTURE WORK

Smartphone, as handhelds devices which have started a new revolution in software engineering, almost involved in everyone's daily life. The applications which installed on the mobile phone help and improve every day's life and work of the users. For example, the financial bank applications help the user to access bank account through smartphone directly; the third-party shopping applications support the user sharing shopping experience to Facebook accounts. However, security of smartphone has become a significant problem. The attackers focus on smartphone mobile application because of three main reasons: 1) lacking good standards for mobile; 2) developers have limited knowledge on application development in security way; 3) attacker can access more valuable resources stored in mobile phone compare with traditional devices, including GPS geographic location information, SMS messages, contacts, calendars, etc. In this dissertation, we divide three topics: SSL; WebView; and HTML5-based applications to clarify those attacks with three validation experiments, respectively.

We hope the work that presented in this dissertation will not only provide the research effort but also a contribution to computer science and security education. In this thesis, we provide the systematic and comprehensive concepts explanation and related attacks analysis. For the chapter 2, we provide the full background of SSL implementation vulnerabilities with various related concepts, such as HTTP and HTTPS, digital certificate, etc. The analysis of SSL implementation vulnerabilities illustrates that the inappropriate utilization of SSL for the mobile applications can cause a number of risks. Even through SSL is a security technology that provides the encrypted and secured communication, it is not able to guarantee the security of connection. For those require high-security level applications, including bank applications, social applications, developers should pay more attention to the usage of SSL. For the WebView attack in chapter 3, we state the risks caused by the utilization of WebView. And the reasons why those attacks could be launched are discussed in detail. For chapter 4, we illustrate the HTML5-based applications' vulnerabilities.

Because the middleware allows the native code from the application to interact with JavaScript code which runs inside of the WebView, the code injection attack can damage much more than a traditional XSS attack. Besides, we provide three validation experiments to demonstrate the SSL validation attack, WebView code injection attack and HTML5-based application vulnerabilities in an accurate and explicit way.

There are several works could be done to improve our current work. The first one is the proxy for the SSL implementation vulnerabilities validation experiments. We only use mitmproxy's one function – fake a self-signed certificate to the lunch the MITM attack. We can extend and analysis SSL implementation vulnerabilities in a more comprehensive way. The second one is that conducting more types of WebView attacks as the validation implementation experiments. For example, the WebView attack launch from the malicious web page. And the third one is that we consider adding more channels attacks validation experiments for HTML5-based applications' analysis, including camera, SMS, etc. Moreover, the countermeasures for each vulnerability need to be discussed and analyzed in the future.

REFERENCE

- [1] 99.6 percent of new smartphones run Android or iOS
<https://www.theverge.com/2017/2/16/14634656/android-ios-market-share-blackberry-2016>
- [2] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, V. Shmatikov. “The Most Dangerous Code in the World: Validating SSL Certificates in Non- Browser Software”
- [3] S. Fahl, M. Harbach, T. Muders, M. Smith, L Baumgartner, B. Freisleben. “Why Eve and Mallory Love Android: An Analysis of Android SSL (in) Security”
- [4] T. Luo, X. Jin, A. Ananthanarayanan, W. Du. “TouchJacking Attacks on Web in Android, iOS, and Windows Phone”
- [5] T.Luo, H. Hao, W. Du, Y. Wang, H. Yin. “Attacks on WebView in the Android System”
- [6] X.Jin, X. Hu, K. Ying, W. Du, H. Yin, G. Peri. “Code Injection Attacks on HTML5-based Mobile Apps: Characterization, Detection and Mitigation”
- [7] X. Jin, T. Luo, D. Tsui, W. Du. “Code Injection Attacks on HTML5-based Mobile Apps”
- [8] Transport Layer Security. https://en.wikipedia.org/wiki/Transport_Layer_Security
- [9] SSL Protocol explanation. <http://kb.cnblogs.com/page/162080/>
- [10] How to set mitmproxy <https://corte.si/posts/code/mitmproxy/howitworks/index.html>
- [11] How MITM proxy works. <https://github.com/mitmproxy/mitmproxy>
- [12] Brendan Cusak. Secure Socket Layer
<http://searchsecurity.techtarget.com/definition/Secure-Sockets-Layer-SSL>
- [13] G. Morales Buitron-Damas. Https connections over android. In IEEE, 8th international Conference on digital object identifier, 2013.
- [14] Jay Graves. SSL Pinning for Increased App Security
<http://www.doubleencore.com/2013/03/ssl-pinning-for-increased-app-security/>
- [15] PAUL SAWERS. Nielsen: US smartphones. <http://thenextweb.com/#!/pZE8v>
- [16] Android Team. Webview, Android developers
<http://developer.android.com/reference/android/webkit/WebView.html>

- [17] Longteng Xu Wanging You. Comparison of TCP and SSL for mobile security. In IEEE, international Conference on sensor network security, 2013.
- [18] Wikipedia. iOS operating system. <http://en.wikipedia.org/wiki/iOS>, Oct 2017.
- [19] How To: Use mitmproxy to read and modify HTTPS traffic
<http://blog.philippeckel.com/2013/07/01/how-to-use-mitmproxy-to-read-and-modify-https-traffic-of-your-phone/>
- [20] Definition of MITMProxy, <https://mitmproxy.org/doc/index.html>
- [21] Man-in-the-middle attack http://en.wikipedia.org/wiki/Man-in-the-middle_attack
- [22] What is SSL? <https://www.globalsign.eu/ssl-information-center/what-is-ssl.html>
- [23] How MITMProxy works? <https://mitmproxy.org/doc/howmitmproxy.html>
- [24] Upstream Certs <http://mitmproxy.org/doc/features/upstreamcerts.html>
- [25] Understanding WebView and Android security patches
<https://www.androidcentral.com/android-webview-security>
- [26] Attacks on Android WebView
<http://resources.infosecinstitute.com/android-hacking-security-part-7-attacks-android-webviews/#gref>
- [27] Penetration Testing Lab- Android WebView Vulnerabilities
<https://pentestlab.blog/2017/02/12/android-webview-vulnerabilities/>
- [28] Android WebView Exploit Tutorial <https://cyberarms.wordpress.com/2014/02/26/android-webview-exploit-tutorial-70-of-devices-vulnerable/>
- [29] Building Web Apps in WebView
<https://developer.android.com/guide/webapps/webview.html>
- [30] Embedding the WebView <http://docs.phonegap.com/develop/1-embed-webview/android/>
- [31] Smartphones industry: Statistics and Facts
<https://www.statista.com/topics/840/smartphones/>
- [32] Smartphone OS Market Share, 2017 Q1
<https://www.idc.com/promo/smartphone-market-share/os>
- [33] Source, Security, <https://source.android.com/security/>
- [34] How to debug HTTP(s) traffic on Android
<https://medium.com/@rotxed/how-to-debug-http-s-traffic-on-android-7fbe5d2a34>

- [35] 75% of developers using html5: survey
<http://eweek.com/c/a/Application-Development/75-of-Developers-Using-HTML5-Survey-508096>
- [36] PhoneGap, [http:// phonegap.com](http://phonegap.com)
- [37] Rhomobile, <http://rhomobile.com>
- [38] Appcelerator, <http://appcelerator.com>
- [39] PhoneGap Tutorial, <https://www.tutorialspoint.com/phonegap/>
- [40] OWASP Top 10 Application Security Risks- 2017
https://www.owasp.org/index.php/Top_10_2017-Top_10
- [41] B. A B. “Cross-site Scripting Attacks on Android WebView”
- [42] S. Sedol, R. Johari, “Survey of Cross-site Scripting Attack in Android Apps”
- [43] Android Security: Implementation of Self-signed SSL certificate for your App
<https://www.codeproject.com/Articles/826045/Android-security-Implementation-of-Self-signed-SSL>
- [44] L. Onwuzurike, E. Cristofaro. Danger is My Middle Name-Experimenting with SSL Vulnerabilities in Android Apps. In Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks (2015).
- [45] D. Southiraraj, J. Sahs, G. Greenwood, Z. Lin, L. Khan. SMV-HUNTER: Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps. In Network and Distributed System Security Symposium (2014).
- [46] C.Spensky, J. Stewart, A. Yerukhimovich, R. Shay, A. Trachtenberg, R. Housley, and Robert. Couunningham. SoK: Privacy on Mobile Devices-It’s Complicated. In Proceedings in Privacy Enhancing Technologies (2016).
- [47] C. Qian, X. Luo, Y. Le, G. Gu. VulHunter: Toward Discovering Vulnerabilities in Android Applications. IEEE Computer Society (2015)
- [48] Survey of Cross-site Scripting Attack in Android Apps, Stanzein Sedol and Rahul Johari, International Journal of Information & Computation Technology. ISSN 0974-2239 Volume 4, Number 11 (2014), pp. 1079-1084
- [49] Board of Governors of the Federal Reserve System, <https://www.federalreserve.gov/>
- [50] Eclipse official website, <http://www.eclipse.org/> Eclipse official website

- [51] How to create and launch emulator in Eclipse
<http://theopentutorials.com/tutorials/android/how-to-create-android-avd-emulator-in-eclipse/>
- [52] Install Android SDK, Eclipse, and Emulator(AVDS) <http://android.konreu.com/developer-how-to/install-android-sdk-eclipse-and-emulator-avds/>
- [53] PhoneGap official website, <https://phonegap.com/>
- [54] Android Studio, <https://developer.android.com/studio/index.html>.

Appendix A: SSL implantation Vulnerability Lab

1. Objectives

SSL (Secure Sockets Layer) is the de facto standard for secure Internet communications. However, SSL certificate validation has been shown completely broken in many security-applications and libraries. Flawed certificate validation renders software vulnerable to man-in-the-middle attack.

On the smartphone app market, apps are developed by developers with various levels of security knowledge, and many of them are suspected to be flawed in certificate validation. In this lab, students are expected to conduct a serial of experiments to find flawed apps and further analyze the cause. They will learn how to set up the proxy, monitor the HTTPS packages, identify the sensitive information in the packages and distinguish malicious applications.

2. Pre-lab Reading

There are two excellent articles on SSL implantation Vulnerabilities on the mobile operating system. “The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software” by Georgiev et al. [1] demonstrated that even the standard SSL libraries such as JSSE, OpenSSL, GnuTLS, etc. which are used in applications might have the certificate validation incorrectly problem. The authors of this paper presented that using SSL in no-browser software is a surprisingly challenging task.

The other one is “Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security” by Fahl et al. [2] focusing on SSL problem in Android platform. The authors demonstrated an investigation of the current usage and the security threats on SSL/TLS in Android.

3. Experiment

This experiment is designed to find whether an app is flawed in certificate verification process. A malicious proxy server is used to issue a fake certificate when an app tries to set an SSL-connection with a legitimate server. In a normal situation, the verification process should fail, and the secure connection cannot be set up. However, in the case of flawed app, a fake certificate can pass a flawed verification process. In either case, a secure channel is established between the app and the malicious proxy server --- hence all further communications between the app and the legitimate server will be monitored by the malicious proxy server.

There are three components in this experiment. They are the applications' server; the malicious proxy server and the client part. This experiment focuses on the proxy server and the client side. And the remote applications' server is not discussed here. It should be pointed out that the mobile device installed app is not the only client type. It can be any device which installed application with SSL. For students' convenience, the client type could be a smartphone (Android), iOS phone, or Android emulator.

In the following, brief instructions are given on:

1. how to set up the malicious proxy server (Mitmproxy)
2. how to set the malicious proxy server as your smartphone's proxy server
 - 1) Android phone
 - 2) iOS phone
 - 3) Android virtual device
3. how to test whether an app is flawed or not.

3.1 Set up the malicious proxy server

A. The malicious proxy server MITMProxy runs in the Linux environment. So, the first step is to set up a virtual Linux environment. (Ignore this step if you use Linux as your primary OS.)

- a) Download VirtualBox from <https://www.virtualbox.org/> and install it.

- b) Download a distribution of Linux. Recommend Lubuntu, which has low hardware requirements. Download Lubuntu from <http://www.lubuntu.net/>.
- c) Create a new Virtual Machine in VirtualBox like this. You can use defaults for all following settings (Figure 1).

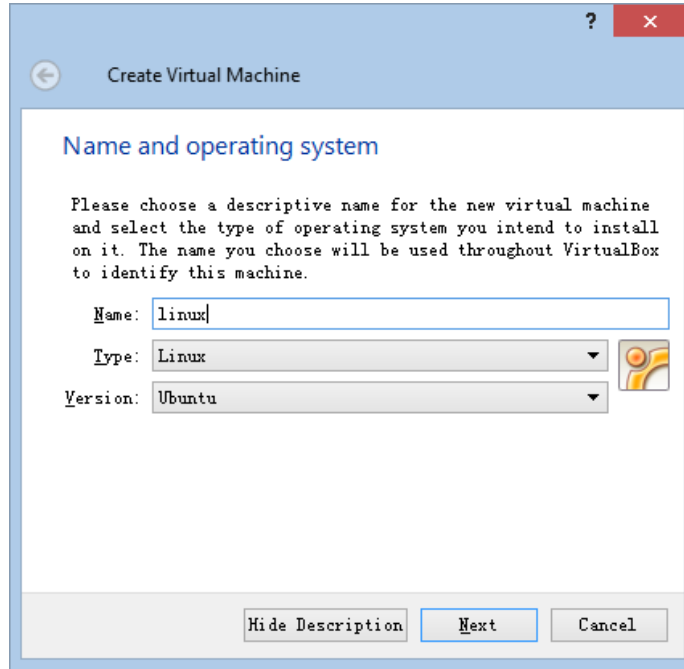


Figure 1. New Virtual Machine

- d) Create a new Virtual Machine in VirtualBox like this. You can use defaults for all following settings (Figure 1).

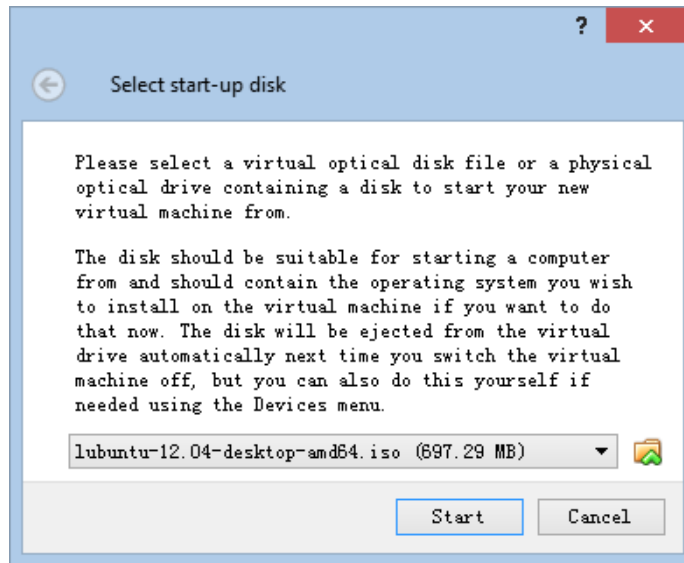


Figure 2. Select start-up disk

- e) Finish Lubuntu installation. You can use defaults for most settings.
- f) Set Network adapter of your VM works on as Bridged Adapter. Make sure your smartphone could reach it (Figure 3).

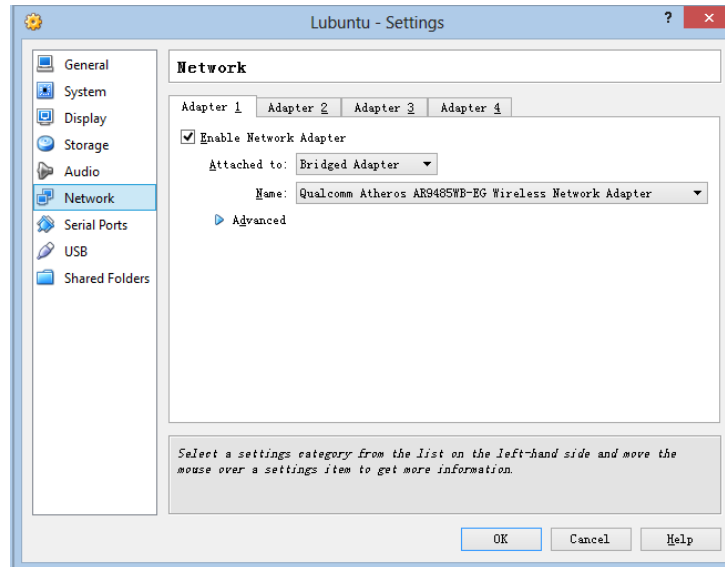


Figure 3. Setting network

B. Install MITMProxy proxy on your virtual Linux machine

- a) First, install pip on your Linux. pip is a tool for installing and managing Python packages. On Debian and Ubuntu, use commands: (as <http://www.pip-installer.org>)

```
$ sudo apt-get install python-pip
```

- b) Install MITMProxy. If pip is installed, use commands: (for detailed instructions, please check <http://mitmproxy.org/>)

```
pip install MITMProxy
```

If an error occurs, please check all packages used by MITMProxy have been installed.

- c) Start mitmproxy in Terminal by commands:

```
$ mitmproxy
```

Detailed usages, please follow <http://mitmproxy.org>. (For testing purpose, DO NOT add the root certificate to trust list on a smartphone.)

- d) Use the following command to get the VM's IP address (or the proxy's IP address) which is to be used by the smartphone

\$ ifconfig

C. Optional malicious proxy (this part is optional if you do not want to use MITMProxy)

Besides MITMProxy, “Charles” and “Fiddler” could also be used as a malicious proxy. Please be aware that different proxies use different ways to generate fake certificates. For more information, please check those on the related reference.

3.2 Setup your Smartphone proxy

A. Android smartphone setup

- a) Setup proxy in the advanced Wi-Fi settings. (If the UMD-Wireless or UMD-Secure cannot work, please do this experiment in other Wi-Fi environments.)
- b) Connect to the desired network (access point) if not already. Select “Modify network” config:

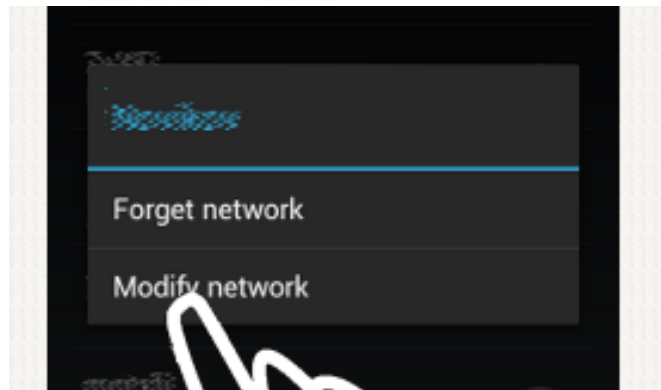


Figure 4. Modify network

- c) In the opened dialog, mark “Show advanced options” checkbox:

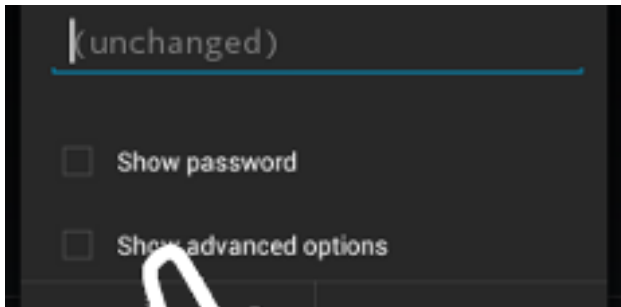


Figure 5. Advanced options

- d) Scroll to proxy settings, select Manual, proxy options will appear. Set Proxy hostname as your VM's IP address. Proxy port: as 8080.

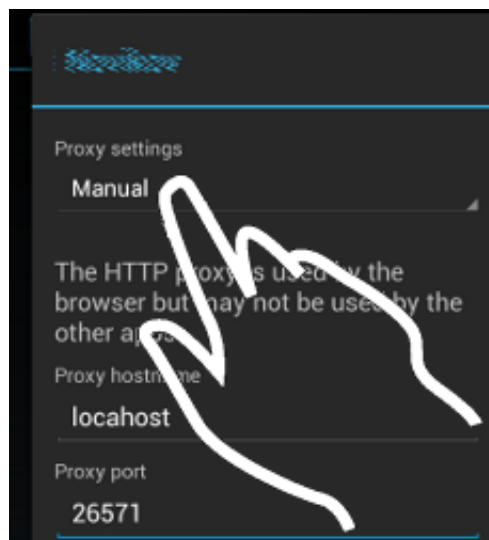


Figure 6. Manual setting

- e) Alternatively, you can set proxy by ProxyDroid. If your smartphone has been 'rooted', you can download an app named ProxyDroid. This app would help you manage your proxy settings.

B. iOS smartphone setup

- a) Setup proxy in the WLAN settings. (If the UMD-Wireless or UMD-Secure cannot work, please do this experiment in the other Wi-Fi environment.)

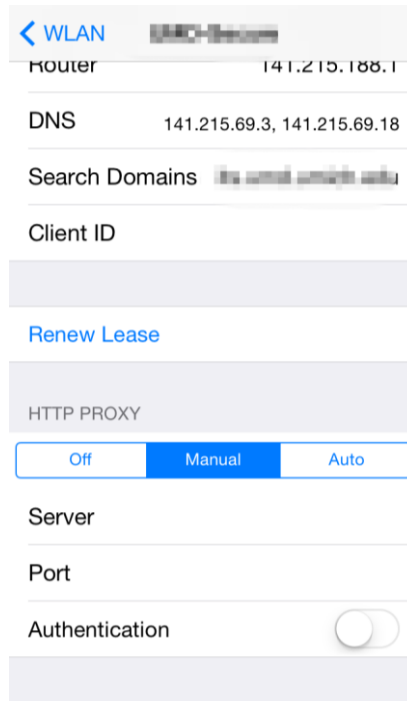


Figure 7: WLAN setting

- b) Scroll to HTTP PROXY settings, select Manual, proxy options will appear. Set Server as your VM's IP address. Proxy port: as 8080.

C. Android emulator

In this lab, the Android emulator is Android Virtual Devices (AVDs) which based on Android Software Development Kit (Android SDK) and Eclipse IDE. The brief steps are described in the following (For more detail, check the reference at the end of this instruction or google online). If eclipse is already installed on your computer, you could skip this part directly. Besides this Android virtual machine, other Android emulators are also can be used, such as Android Studio. For more information, please check those on the related reference.

- a) Download the Android Software Development Kit (SDK)

The SDK starter package is not a completed development environment which just included the core SDK tools. <http://developer.android.com/sdk/index.html>, download the latest version of SDK starter.

- b) Download Eclipse IDE for Java Developers

This link is the Eclipse downloads. <http://www.eclipse.org/downloads/>, Find the Eclipse IDE for Java Developers. Noted: the correct version for your operating system should be guaranteed. After your Eclipse IDE download is complete, unzip and move to a permanent folder

c) Install the Android Development Tools (ADT) plugin

Open Eclipse to install the Android Development Tools (ADT) by using Eclipse's built-in plug-in system. Here are the basic steps.

- i. Choose "Help" > "Install New Software..."
- ii. Click the "Add..." button and create a new entry:
 - Name: "Android ADT" (this space is for your own personal use, so name it whatever you want)
 - Location: "https://dl-ssl.google.com/android/eclipse/" (try just http:// if the https:// does not work)
- iii. Check all the boxes to install all the tools
- iv. Just keep clicking "I agree", "Next", "Yes", etc. until it asks you to restart
- v. Go ahead and restart Eclipse when prompted to

d) Install Android SDK Components

Download the Software Development Kits(SDKs). This step could be done by Eclipse IDE and the Android ADT. Here are the basic steps.

- i. Open Eclipse, click "Window" > "Android SDK and AVD Manager"
- ii. In "Available packages", select the platforms you want to support. You can either choose all, or pick-and-choose what you want to develop for.
- iii. In the "Third party Add-ons", decide what you are interested in. The Google APIs must be installed.
- iv. Choose "Install Selected", then the "Accept All" radio button, then "Install".

e) Setup proxy in the Mobile networks setting. (If the UMD-Wireless or UMD-Secure cannot work, please do this experiment in the other Wi-Fi environment.)

- i. Create New AVD, select Window > AVD Manager > New. The new Android Virtual Device should set the Android Virtual Device (AVD) as shown below. (Noted: the AVD showed just for reference, you could create different if the experiment could run.)

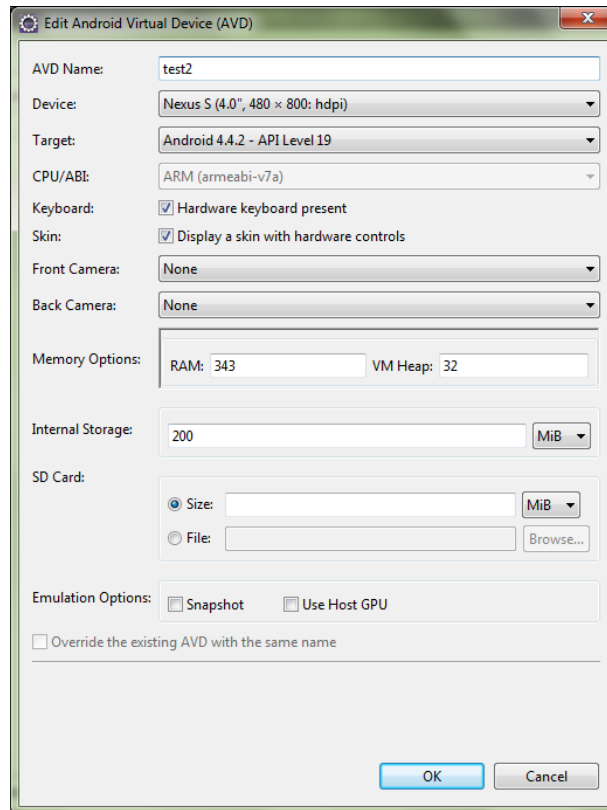


Figure 8. Example AVD setting

ii. Then launch the ADV, the interface shows as below:

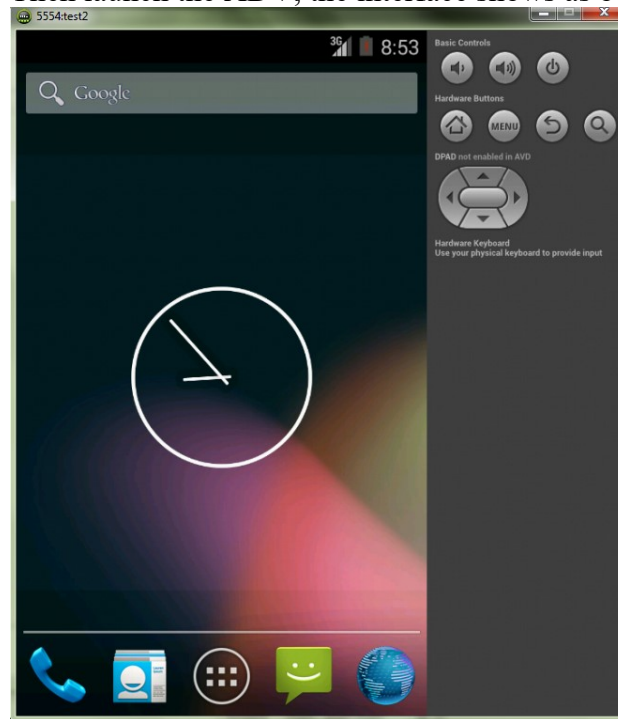


Figure 9. ADV interface

- iii. Choose the setting icon. Choose the More.

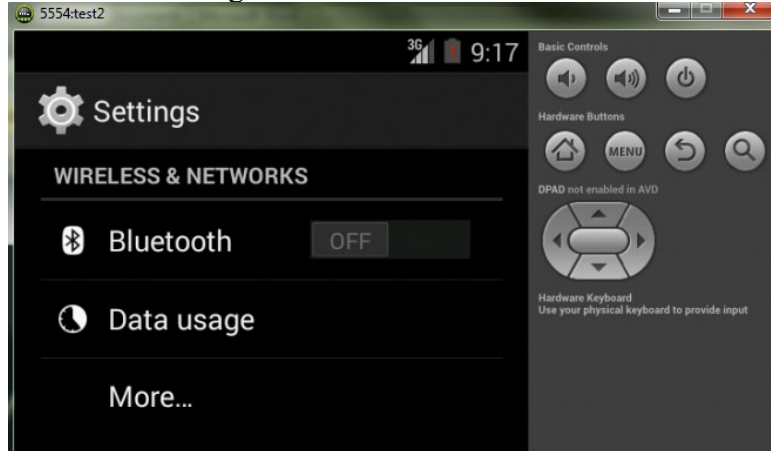


Figure 10. Android setting

- iv. Choose the Mobile networks.

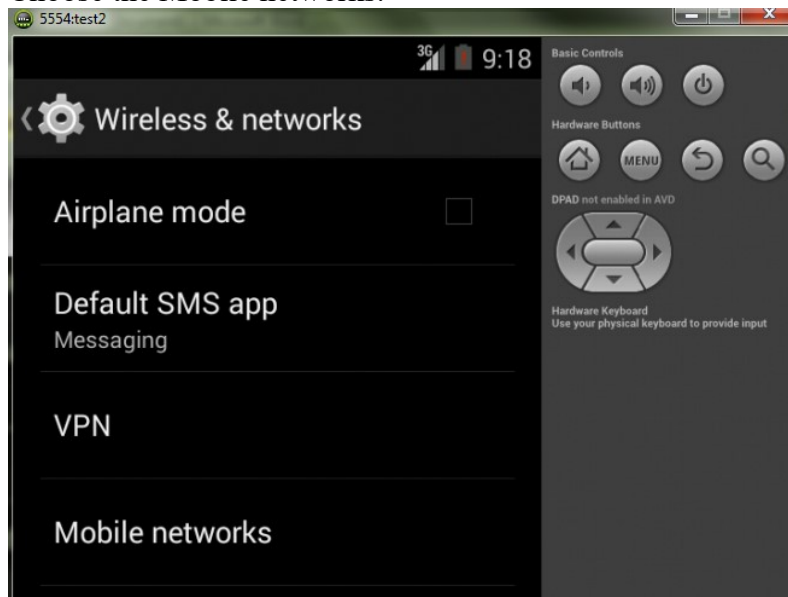


Figure 11. Mobile networks

- v. Choose the Access Point Names

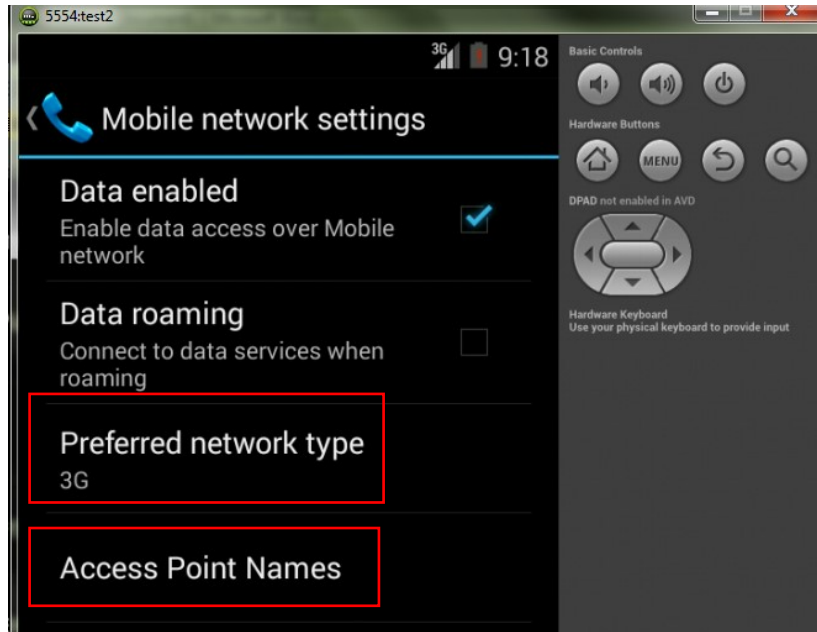


Figure 12. Access point names

- vi. Then change the item T-Mobile US

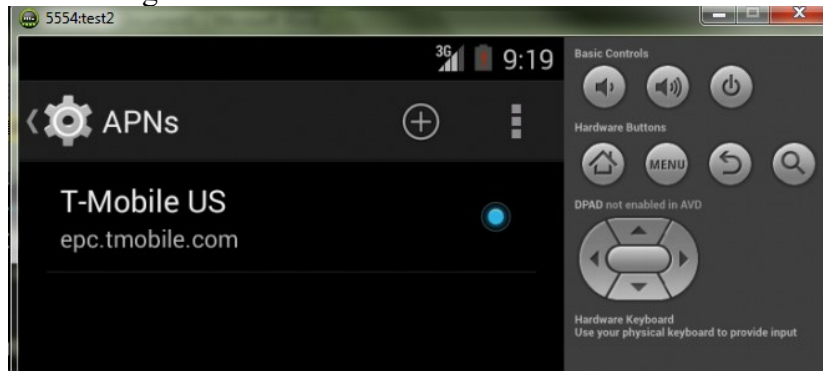


Figure 13. Choose APNs

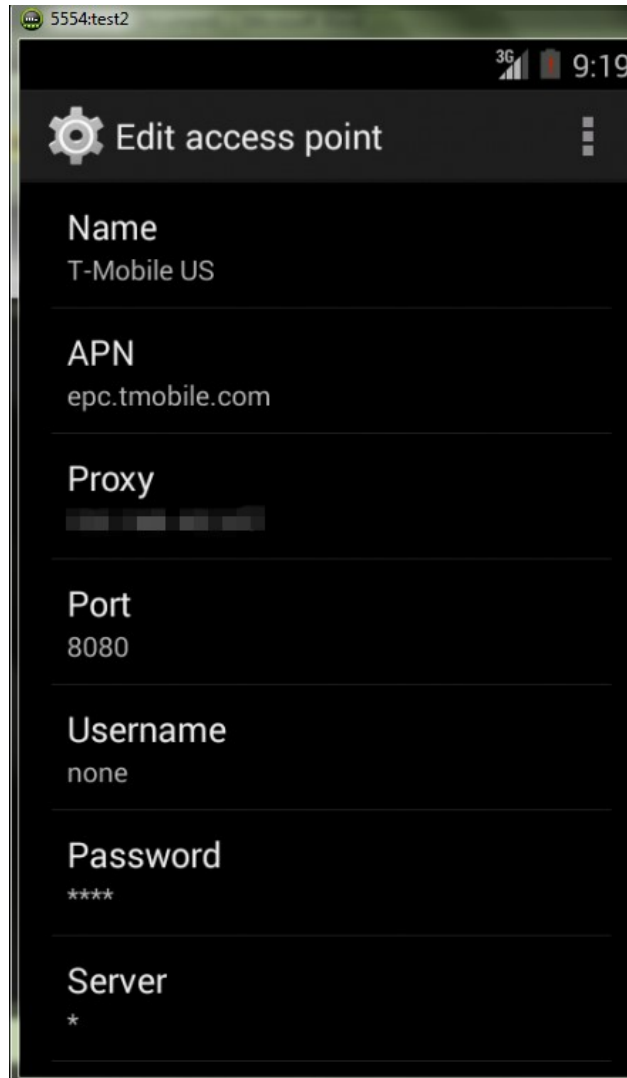


Figure 14. Set proxy and port

- vii. Change the proxy to the proxy server's IP address and change the port number to 8080.
- f) Install the Android application on the emulator.
 - i. Put the .apk files in the platform-tools folder. (the default path: C:\Program Files\Android\android-sdk-windows\platform-tools) and for example, named .apk file is game.apk
 - ii. Install the application.
 - open the cmd windows, enter the command:
cd C:\Program Files\Android\android-sdk-windows\platform-tools
 - After entering the folder, enter the command:
adb install game.apk

```
C:\Documents and Settings\Administrator>cd C:\Program Files\android-sdk-windows\platform-tools
C:\Program Files\android-sdk-windows\platform-tools>adb install game.apk
385 KB/s (1516333 bytes in 3.843s)
  pkg: /data/local/tmp/game.apk
Success
C:\Program Files\android-sdk-windows\platform-tools>
```

Figure 15 Install application

1. Open the emulator, the .apk file installs successfully.

3.3 Testing procedure

A. Download apps from Google Play

- a) You should choose the apps which come with potential risks, such as those which have a login page for username and password.
- b) The following are some example categories that could be used in this project (you are allowed to choose a different category other than the list below). Please select two categories and test the top 50~100 apps in each category:
 - SOCIAL
 - COMMUNICATION
 - FINANCE
 - BUSINESS
 - TRAVEL AND LOCAL
 - SHOPPING

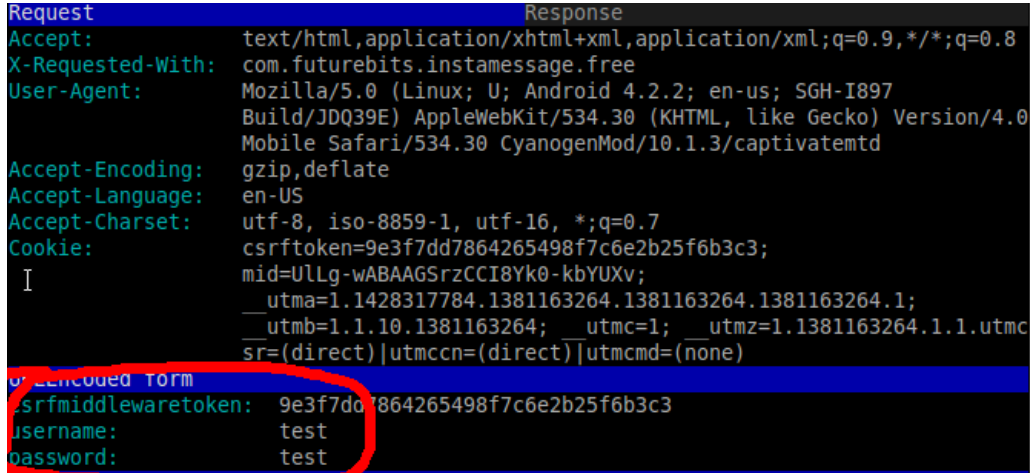
B. Test the app with malicious proxy

After your devices connected to the malicious proxy, you can monitor all traffics transmitted between your phone and the proxy. Figure 16 shows the example attack result.

- a) Perform “login in” in your app.
- b) If the app is secure, if you use ProxyDroid, you will be given a connection error message. If you set up your proxy in the Advanced Wi-Fi Settings, your app will skip the proxy and connect to the legitimate server without the use of the proxy server. In both cases, you wouldn’t find a request with username and password form the monitor.

- c) If the app is insecure, you will find your username and password on the monitor.

You can install packet sniffer software such as Wireshark to monitor the traffic.



```
Request      Response
Accept:      text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
X-Requested-With: com.futurebits.instamessage.free
User-Agent:   Mozilla/5.0 (Linux; U; Android 4.2.2; en-us; SGH-I897
            Build/JDQ39E) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0
            Mobile Safari/534.30 CyanogenMod/10.1.3/captivatemtd
Accept-Encoding: gzip,deflate
Accept-Language: en-US
Accept-Charset: utf-8, iso-8859-1, utf-16, *,q=0.7
Cookie:       csrfmiddlewaretoken=9e3f7dd7864265498f7c6e2b25f6b3c3;
            mid=ULLg-wABAAGSrzcCI8Yk0-kbYUXv;
            __utma=1.1428317784.1381163264.1381163264.1381163264.1;
            __utmb=1.1.10.1381163264; __utmc=1; __utmz=1.1381163264.1.1.utm
            sr=(direct)|utmccn=(direct)|utmcmd=(none)

Content-Type: application/javascript
Content-Length: 1024
Expires: -1
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache

[Encoded Form]
csrfmiddlewaretoken: 9e3f7dd7864265498f7c6e2b25f6b3c3
username: test
password: test
```

Figure 16. Example attack

Appendix B: WebView Attack Lab

1. Objectives

Web browser provides the World Wide Web information for users. It plays an important role in daily life. Smartphone users can use applications to get web information. WebView is the part of the smartphone OS responsible for rendering web pages in most mobile applications. WebView helps developers integrate browser's function easily, such as web page navigation, and JavaScript execution. However, apps are developed by developers with various level of security knowledge. The applications developed by careless developers not only have some potential risks but also can be used to wiretap sensitive information.

This lab designed for students who have the computer science background and want to explore computer security issues. After the lab, students will have a better understanding of WebView attack. The JavaScript Code Injection is realized in this experiment. We provide a malicious app called Capture.apk and the corresponding project package to students. It should be pointed out that this project package is not completed version. Students need to add the malicious code by themselves and let the app run the same function as the example Capture.apk. They could learn how to execute the JavaScript Code injection attack and add the malicious code to the application.

2. Pre-lab Reading

There are two excellent articles on WebView security problems in mobile operating system. "Touchjacking Attacks on Web in Android, iOS, and Windows Phone" by Luo et al. [4] demonstrates that the APIs for WebView are secured, but WebView is still dangerous. The authors of this paper describe several attacks and show that attackers can compromise the integrity and confidentiality of the web contents inside WebView.

Another is “Attacks on WebView in the Android System” by Luo et al. [5] illustrates a number of attacks on WebView. In those attacks, two attacks types are discussed in detail. One is the attack from malicious apps; another is the attack from web pages. This lab design is based on the first one. This paper’s section 2 (Short Tutorial on WebView) indicates the knowledge which students need to learn to finish this lab. The examples in this lab are derived from this paper.

3. Experiment

The basic experiment is designed to launch a malicious application which could steal the Facebook user’s name and password by WebView attack. Normally, the application could not get the user’s information which is loaded in WebView. However, in the case of malicious apps, the attacker can get user’s information or even change the user’s account information.

In the following, brief instructions are given on

1. how to execute the attack;
2. how to set up project into Eclipse;
3. add malicious code to the “uncompleted” project;
4. how to test an attack success or not.

3.1 Execute the attack

There are two choices in this part: Using Android virtual machine which is provided by Eclipse; using a real mobile device. Two options do not affect the results. Students could choose either one depending on the environment. Besides this Android virtual machine, other Android emulators could also be used, such as Android Studio. For more information, please check those on the related reference.

A. Android virtual machine

The Android emulator is Android Virtual Devices (AVDs) which is based on Android Software Development Kit (Android SDK) and Eclipse IDE. The brief steps are described in the following (For more detail, check the reference at the end of this

instruction or google online). If the computer is installed eclipse in advance, please skip this part directly.

- a. Download the Android Software Development Kit (SDK)

The SDK starter package which only includes the core SDK tools is not a completed development environment. <http://developer.android.com/sdk/index.html>, download the latest version of SDK starter.

- b. Download Eclipse IDE for Java Developers

This link is the Eclipse downloads. <http://www.eclipse.org/downloads/>, find the Eclipse IDE for Java Developers. Noted: the correct version for your operating system should be guaranteed. After your Eclipse IDE download is complete, unzip and move to a permanent folder

- c. Install the Android Development Tools (ADT) plugin

Open Eclipse to install the Android Development Tools (ADT) by using Eclipse's built-in plug-in system. Here are the brief steps.

- i. Choose "Help" > "Install New Software..."
- ii. Click the "Add..." button and create a new entry:
 - Name: "Android ADT" (this space is for your own personal use, so name it whatever you want)
 - Location: "<https://dl-ssl.google.com/android/eclipse/>" (try just <http://> if the <https://> does not work)
- iii. Check all the boxes to install all the tools
- iv. Just keep clicking "I agree", "Next", "Yes", etc. until it asks you to restart
- v. Go ahead and restart Eclipse

- d. Install Android SDK Components

Download the Software Development Kits(SDKs). This step could be done by Eclipse IDE and the Android ADT. Here are the basic steps.

- i. Open Eclipse, click "Window" > "Android SDK and AVD Manager".

- ii. In “Available packages”, select the platforms you want to support. You can either choose all, or pick-and-choose what you want to develop for.
 - iii. In the “Third party Add-ons”, decide what you are interested in. The Google APIs must be installed.
 - iv. Choose “Install Selected”, then the “Accept All” radio button, then “Install”.
- e. Setup proxy in the Mobile networks setting. (If the UMD-Wireless or UMD-Secure cannot work, please do this experiment in the other Wi-Fi environment.)

Create New AVD, select Window > AVD Manager > New. The new Android Virtual Device should set the Android Virtual Device (AVD) as shown below. (Noted: the AVD showed just for reference, you can create different ones as long as the experiment run successfully.)

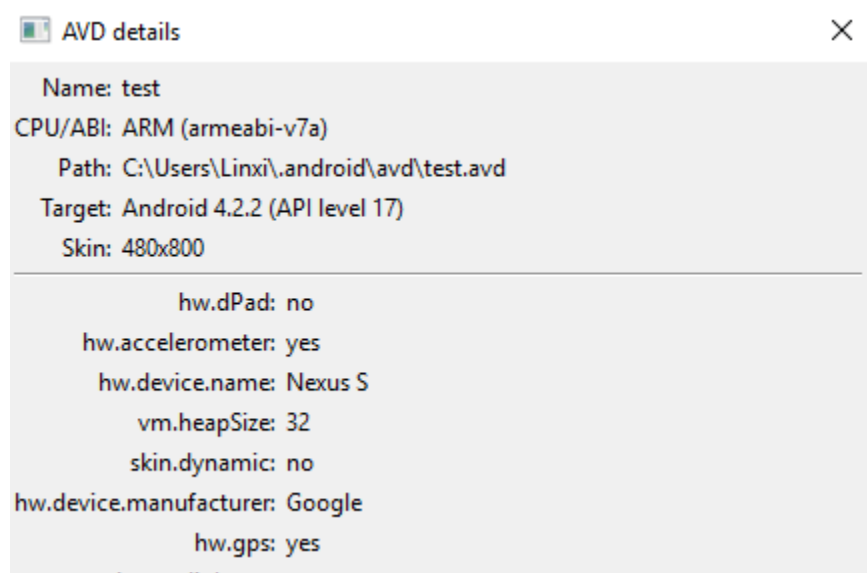


Figure 1. AVD setting details

B. Use real mobile device

- a. Find the file Capture.apk in bin folder

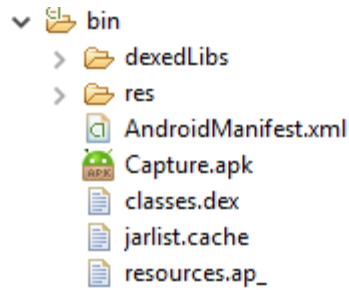


Figure 2. Capture.apk location

- b. Copy this. apk file to the Android device, for example, pause in Downloads folder. Then open Downloads, tap on the APK file, and tap Yes when prompted. The app will begin installing on device.

3.2 Set up project into Eclipse

Import the project package to Eclipse (Figure 3).
Open Eclipse>Choose File>Imported

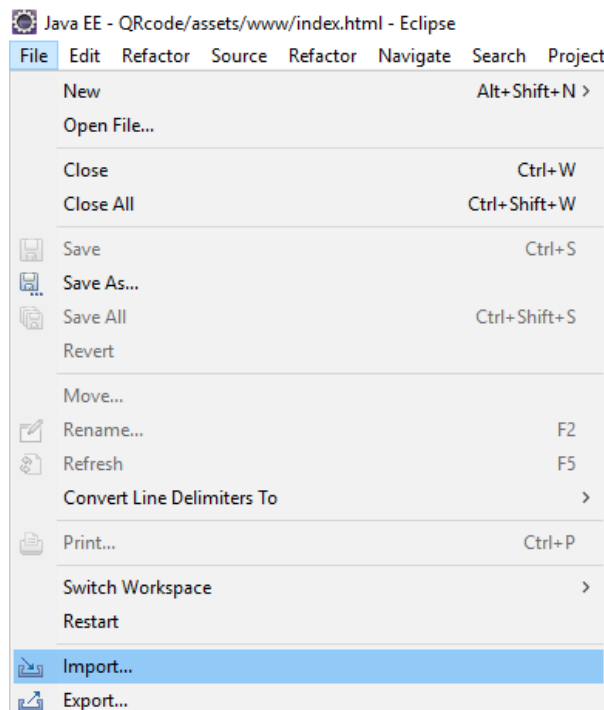


Figure 3. Import projects

- 1) Select import the project package, choose “Existing Android Code Into Workspace”.

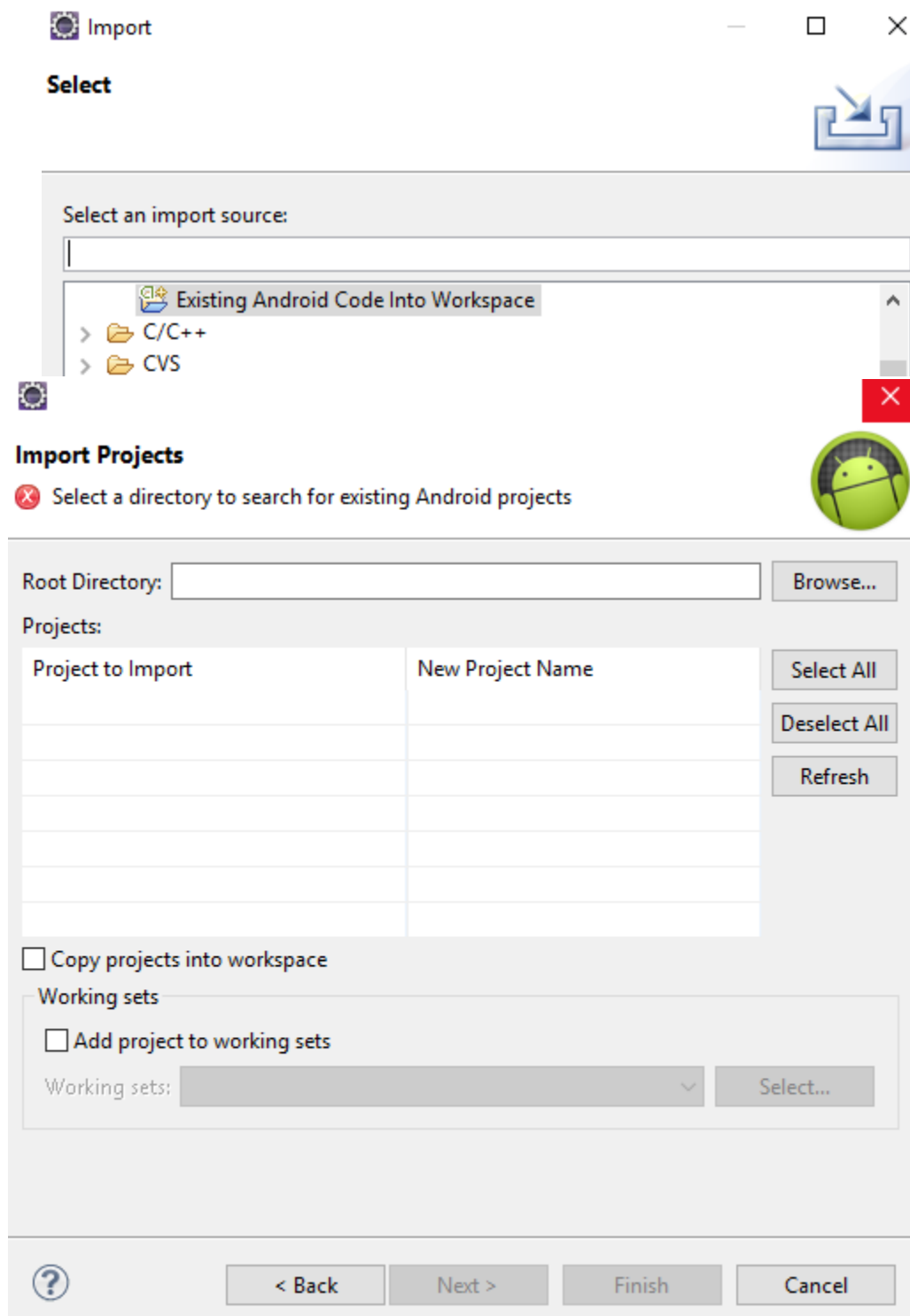


Figure 4. Import projects setting

Choose Browse>Select the project package which provided>Finish
 Success import in the Android projects.

If the Android project is imported successful, the Project should have those files.

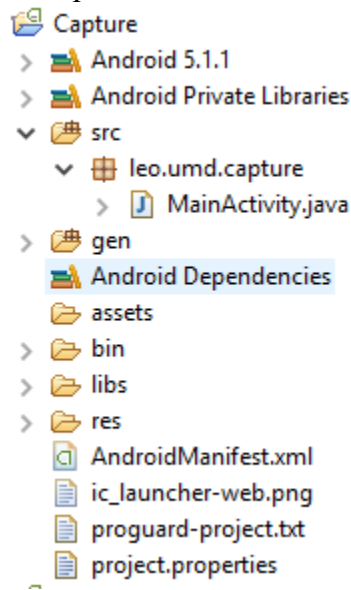


Figure 5. Project files

3.3 Add malicious code

The following “uncompleted” code in **MainActivity.java** shows the malicious part. Students should add the rest malicious code in the blank area. The `sb.append()` is the key part. Those three blue lines showed below are the hints for students to easy to find the direction of this part.

```
@Override
public void onPageFinished(WebView view, String url) {
    super.onPageFinished(view, url);

    StringBuilder sb = new StringBuilder();
    sb.append("document.getElementsByTagName('form')[0].onsubmit = function () {");
    sb.append("var objPWD, objAccount;var str = '';");
    sb.append("var inputs = document.getElementsByTagName('input');");

        sb.append(....)

        sb.append(....)

        ...

        ...

    view.loadUrl("javascript:" + sb.toString());
}
```


3.4 Test an attack success or not

After the part 3.3, students should test whether their attack success or not. The following is one successful attack examples for reference.

a. The Login interfaces

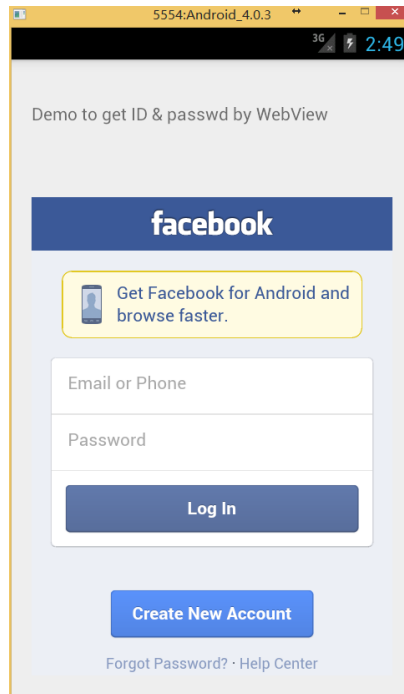


Figure 6. App Interface

b. User type in the personal information

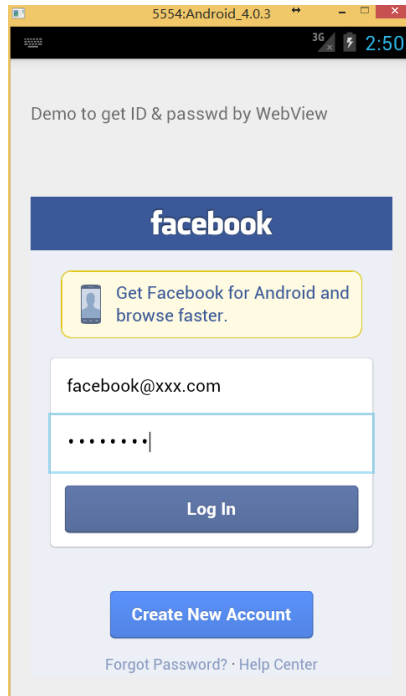


Figure 7. Fill in account and password

3) Capture the information

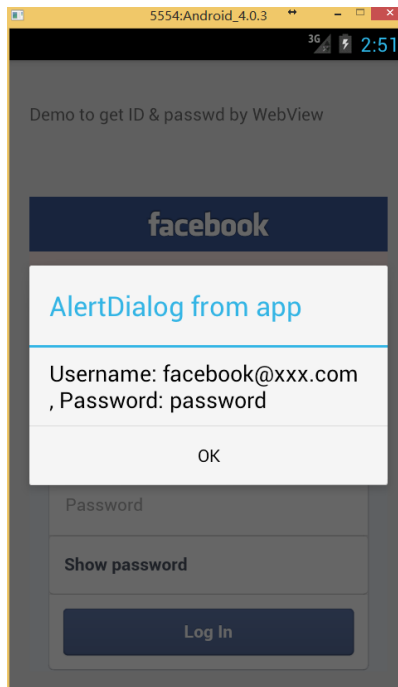


Figure 8. App result

Noted: For education purpose, we did not let the malicious app sends that information to anyone but shows alarm. In a real attack, it may be sent back to attacker.

Appendix C: Code Injection Attacks on HTML5-based Mobile Apps Lab

1. Objectives

Smartphones have become more and more popular in the worldwide. In the meantime, abundant of applications are implemented to run on the mobile platforms. Compare with the native apps which developed by native language; the HTML5-based mobile apps take advantage of web technologies. HTML5-based mobile apps are easily developed by using the standard web technologies, such as HTML5, JavaScript and CSS, and middleware, such as PhoneGap, Appcelerator, RhoMobile, etc. The middleware is in charge of connection between the web content and mobile device resources, including camera, GPS, etc. It is understandable that all mobile operating systems access the Web by supporting web technology. Therefore, the HTML5-based mobile applications are used cross-platform and getting prevalent. However, Web technology has a dangerous feature that allows the mixture of data and code. For example, a traditional web application has the XSS vulnerability. Because HTML5-based mobile apps built upon the same technology as web applications, it inherited the XSS vulnerability. Moreover, due to the usage of middleware, HTML5-based mobile apps have a much broader attack surface than that for web applications. For example, MP3 files, WIFI, 2D barcode, and SMS, could be attack channels for HTML5-based application.

This lab designed for students who have the computer science background and want to explore this computer security problem. After the lab, students should have a better understanding of code injection attacks on HTML5-based mobile apps. The Code Injection attack through MP3 file is realized in this experiment. We provide a malicious app's .apk file which called example.apk and the corresponding project package for students. After this example attack, the contact information is extracted by the attacker. We provide this project

package code only for reference. Students should launch an attack by using this MP3 file method to get victim's other credentials information, including GPS, files, etc.

2. Pre-lab Reading

There are two excellent articles on code injection attacks on HTML5-based apps. One is "Injection Attacks on HTML5-based Mobile Apps" by Jin et al. [7] demonstrated how HTML5-based apps become vulnerable, and how attackers exploit their vulnerabilities through a variety of channels, and what damage can be achieved by the attackers. The examples in this lab are derived from this paper.

Another one is "Code Injection Attacks on HTML5-based Mobile Apps: Characterization, Detection, and Mitigation" by Jin et al. [6] discussed a systematic study on the risk in HTML5-based mobile apps. And the authors develop a vulnerability detection tool to analyze 15,510 PhoneGap Apps from Google Play. Those attacks are divided into two types: attack from malicious apps; attack from web pages. This lab design based on the first one.

3. Experiment

This experiment is designed to launch a malicious application (MP3 Player) which could steal the contact information and other information that stored in the smartphone by code injection attack. An example HTML5-based application developed by using PhoneGap is provided. Normally, this type of application is not able to get the contact information when it plays the MP3 file. However, in the case of malicious apps, it can get user's contact information by launching this code injection attack. Students are required to execute the example code injection attack and extract the other information.

In the following, brief instructions are given on

1. How execute the attack;
2. How to set up project into Eclipse;
3. Add malicious code to the example project;
4. Test the attack success or not.

3.1 Execute the attack

The lab provides an example app to demonstrate the attack. You can install the app on either an Android virtual machine (if you do not have an Android device) or a real Android device. Both could get the same result. Students could choose either one depends on the situation. Besides the Android virtual machine suggested in this document, other Android emulators could also be used, such as Android Studio [7]. For more information, please check those on the related reference.

A. Setup the environment

Choose either

(a) using Android virtual machine or (b) using an Android phone:

(a) Using Android virtual machine

The Android emulator is Android Virtual Devices (AVDs) which based on Android Software Development Kit (Android SDK) and Eclipse IDE. The brief steps are provided in the following (For more detail, check the reference at the end of this instruction or google online). If your computer installed eclipse before, you could skip this part directly.

i. Download the Android Software Development Kit (SDK)

The SDK starter package is not a completed development environment which only included the core SDK tools. <http://developer.android.com/sdk/index.html>, download the latest version of SDK starter.

ii. Download Eclipse IDE for Java Developers

This link is the Eclipse downloads. <http://www.eclipse.org/downloads/>, Fine the Eclipse IDE for Java Developers. Noted: You should guarantee the correct version for your operating system. After your Eclipse IDE download is complete, unzip and move to a permanent folder

iii. Install the Android Development Tools (ADT) plugin

Open Eclipse to install the Android Development Tools (ADT) by using Eclipse's built-in plug-in system. Here are the basic steps.

Choose "Help" > "Install New Software...."

Click the "Add..." button and create a new entry:

- Name: "Android ADT" (this space is for your personal use, so name it whatever you want)
- Location: "https://dl-ssl.google.com/android/eclipse/" (try just http:// if the https:// does not work)

Check all the boxes to install all the tools

Just keep clicking "I agree", "Next", "Yes", etc. until it asks you to restart

Choose "Install Selected", then the "Accept All" radio button, then "Install".

iv. Install Android SDK Components

Download the Software Development Kits(SDKs). This step could be done by Eclipse IDE and the Android ADT. Here are the basic steps.

Open Eclipse, click "Window" > "Android SDK and AVD Manager"

In "Available packages", select the platforms you want to support. You can either choose all or pick-and-choose what you want to develop for.

In the "Third party Add-ons", decide what you are interested in. The Google APIs must be installed.

Choose "Install Selected", then the "Accept All" radio button, then "Install".

- #### v. Setup proxy in the Mobile networks setting. (If the UMD-Wireless or UMD-Secure cannot work, please do this experiment in the other Wi-Fi environment.)

Create New AVD, select Window > AVD Manager > New. The new Android Virtual Device should set the Android Virtual Device (AVD) as shown below.

(Noted: the AVD showed below only for reference, you could create different as long as the experiment could be run.)

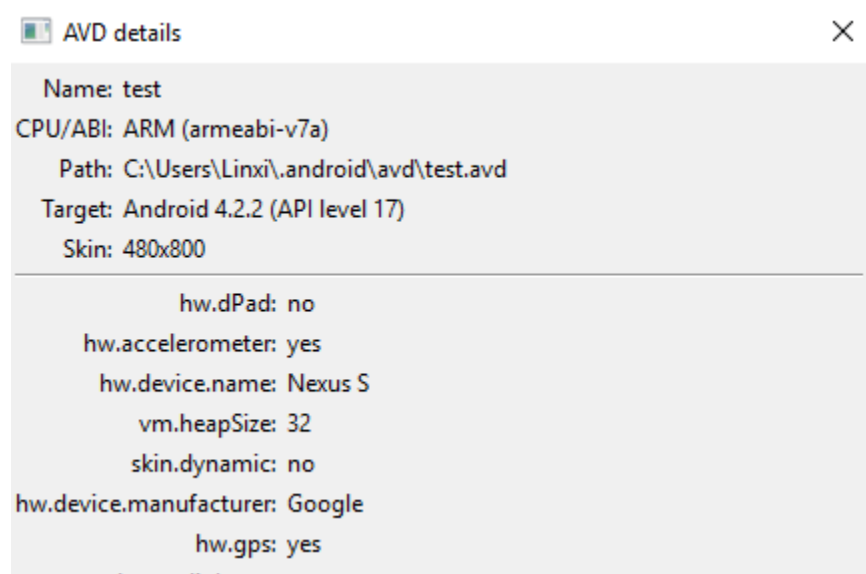


Figure 1. AVD setting details

vi. Find the file example.apk in bin folder.

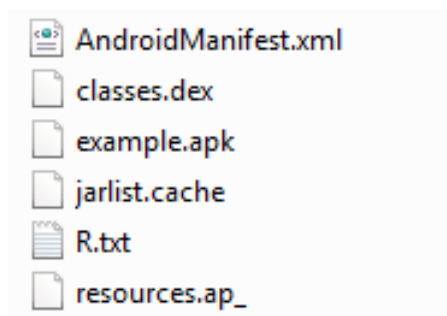


Figure 2. example.apk location

vii. Install example.apk on the emulator.

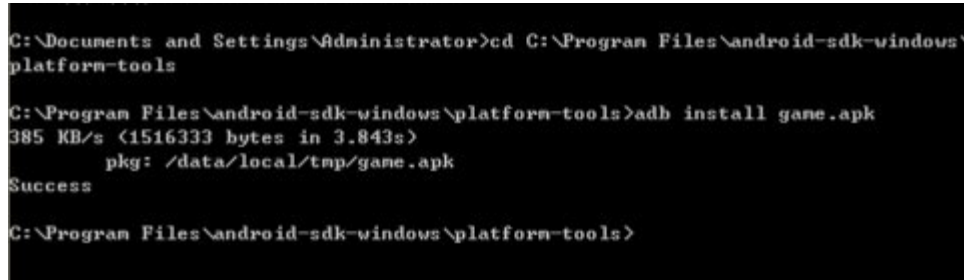
1. Put the .apk files in the platform-tools folder. (the default path: C:\Program Files\Android\android-sdk-windows\platform-tools) and for example, named .apk file is game.apk
2. Install the application.

open the cmd windows, enter the command:

```
cd C:\Program Files\Android\android-sdk-windows\platform-  
tools
```

After entering the folder, enter the command:

```
adb install game.apk
```



```
C:\Documents and Settings\Administrator>cd C:\Program Files\android-sdk-windows\  
platform-tools  
C:\Program Files\android-sdk-windows\platform-tools>adb install game.apk  
385 KB/s (1516333 bytes in 3.843s)  
  pkg: /data/local/tmp/game.apk  
Success  
C:\Program Files\android-sdk-windows\platform-tools>
```

Figure 3. Install apk file

- viii. 3. Open the emulator, the installed application will be shown on the interface.

Use real mobile device

- i. Find the file example.apk in bin folder (Figure 2)
- ii. Copy this. apk file to the Android device, for example, pause in the *Downloads* folder. Then open *Downloads*, tap on the APK file and tap *Yes* when prompted. The app will begin installing on the device.

B. Execute the attack

- 1) Import the malicious MP3 file piano.mp3 to the virtual machine or the mobile device.
 - For the virtual machine, (for other emulators, the processes may different)
 - Go to Window->AVD Manager-> (select AVD)->Edit (in Right panel) ->New (in Hardware) ->SD Card support ->Edit AVD
 - Through DDMS put mp3 in a folder (if not create) in sdcard folder of mnt folder
 - relaunch your AVD and play mp3 file

- For the mobile device, copy this MP3 file to the Android device, for example, store in the Downloads folder.
- 2) Try to run the example.apk with the piano.mp3 file. Before running the example app, you should save at least one contact information if there is no contact information on your Android device. The following figure (Figure 4) shows the saved contact information.

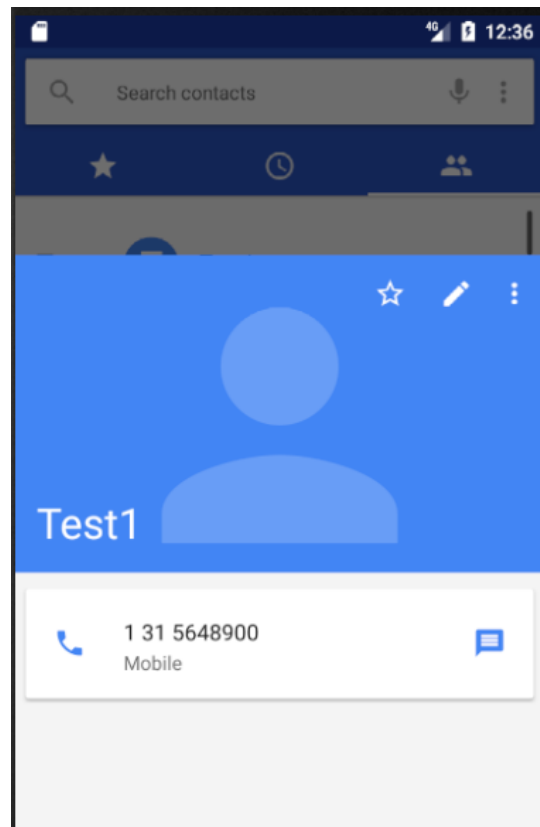


Figure 4. Example Contact information

Then, open the installed example.apk app (Figure 5).

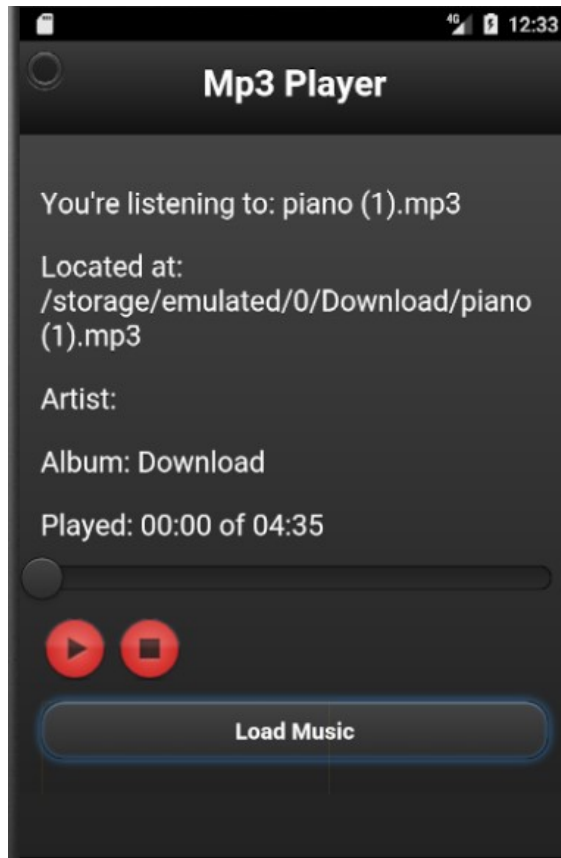


Figure 5. example.apk app interface

Next, click “Load Music” button, the attack will launch. The attacks through the MP3 file can get contact information. Figure 6 shows the successful result for reference.

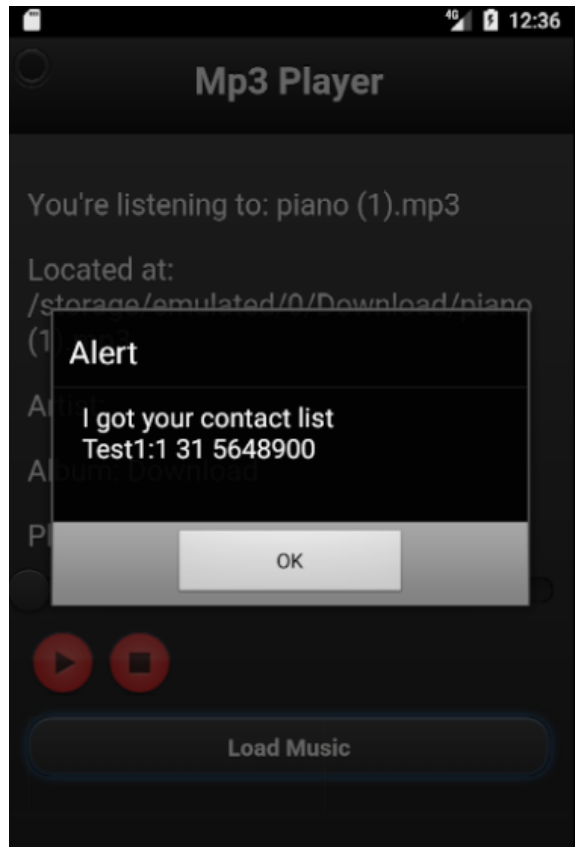


Figure 6. Attack result

3.2 Set up project into Eclipse

Import the project package to Eclipse. (Figure 7)

Open Eclipse>Choose File>Imported

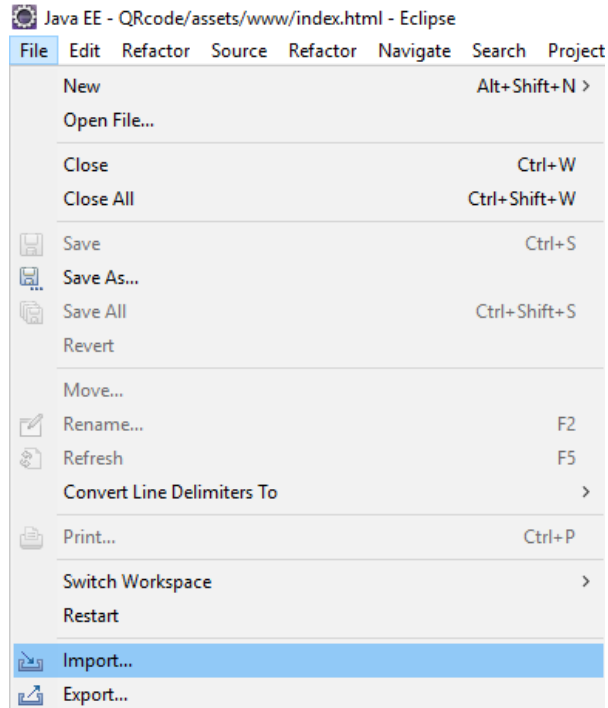
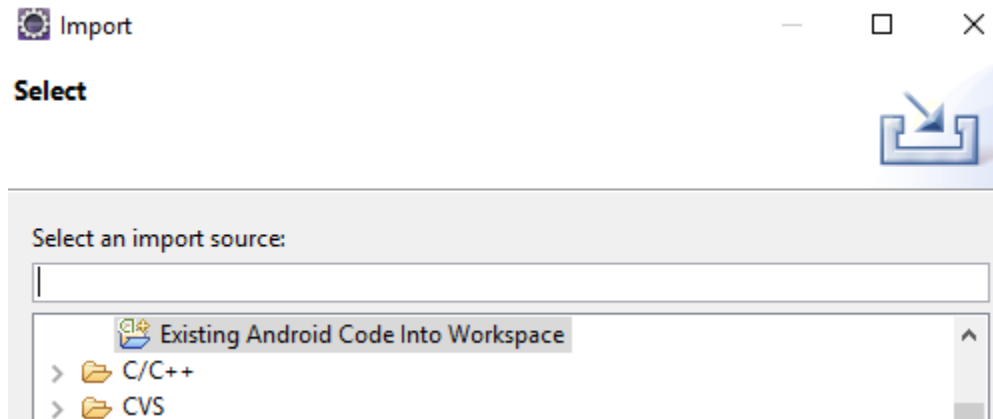


Figure 7. Import a project

- 2) Setting for import the project package, choose “Existing Android Code Into Workspace”



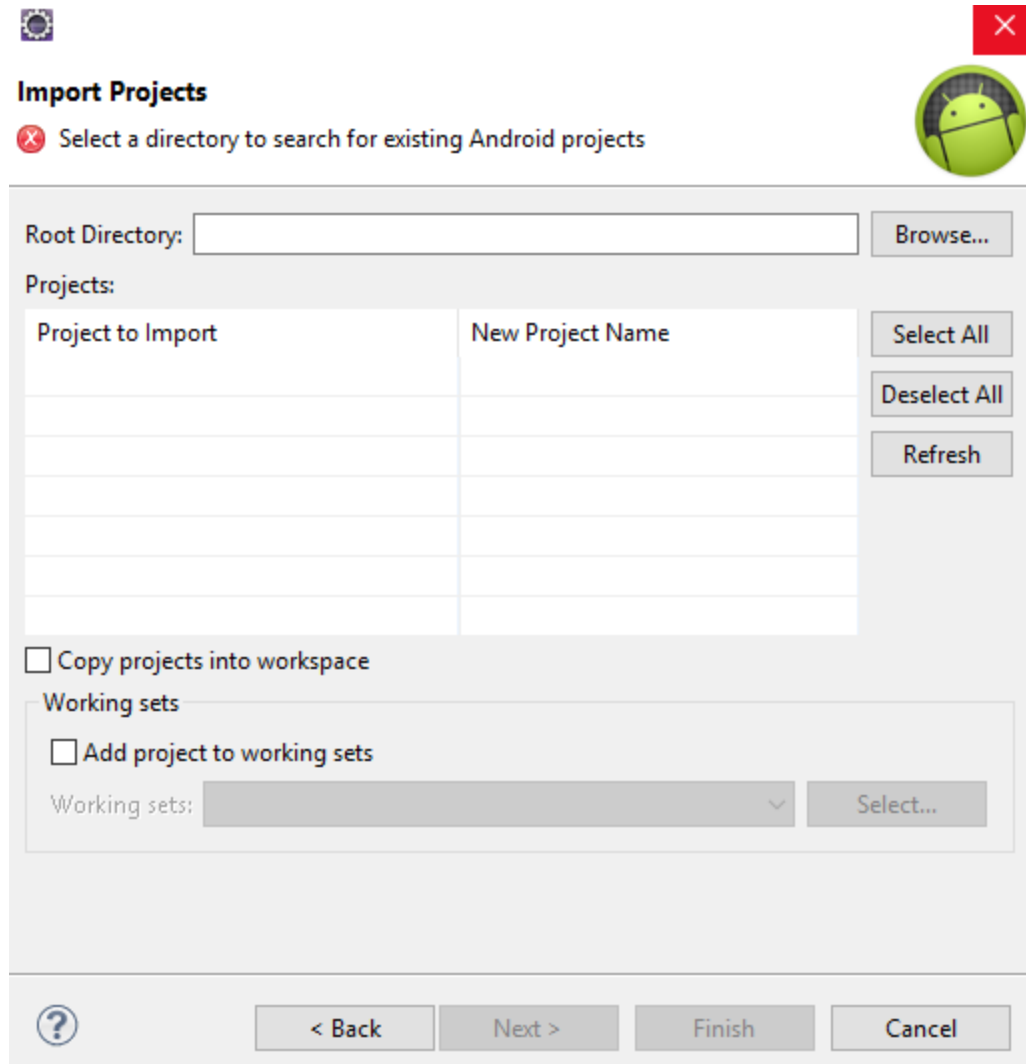


Figure 8. Import projects interface

Choose Browse>Select the project package which provided>Finish
 3) Review the project's code, and try to find the malicious part.

3.3 Add malicious code to the project

The following image shows the provided MP3 file's properties (Figure 9). The artist name and album name are filled with JavaScript codes. When the MP3 player runs and plays this MP3 file, the code and data will mix up. That is the reason why the attack could be launched. Students should find the vulnerable code in project package provided. After the seeking process, try to edit the vulnerable code to get other sensitive information.

- (Noted: check the Album artist part of this MP3 file.)

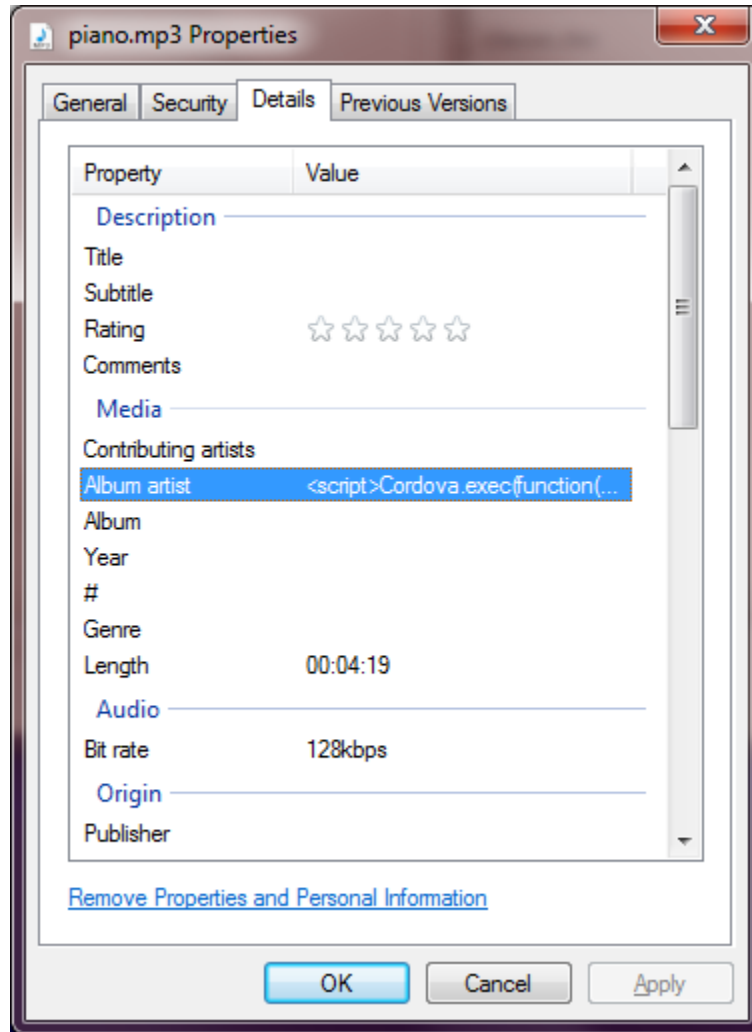


Figure 9. Piano.mp3 details

3.4 Test the attack success or not

After adding the malicious part of example app, Students should try to extract other credentials information, such as storage, camera, and location information, etc.

Noted: For education purpose, we did not let the malicious app send that information to anyone but show it on screen. In a real attack, that information may be sent to the attacker.