OWASP German Chapter
Stammtisch Initiative/Ruhrpott

Android App Pentest Workshop 102
About

• What we try to cover in the second session:
  – MITM attacks
  – Understanding smali code
  – Removing smali code
  – Adding smali code
  – JNI (Java Native Interface)
Setup

• You will need the following:
  – A laptop or any hardware that can run a VM
  – VM: Ubuntu 16.10 Yakkety (64bit).vdi
  – Android VM: Android-x86.5.1 rc1.vdi
  – Virtualbox (recommended)
  – Internet connection to google up things

DL: https://drive.google.com/drive/folders/0BwhtuArcTcxMWlhvTW5SYkFsbWc
Attended last session?

Follow the instructions at

Recap - Session 1

- Setup of a Mobile Application Pentest Environment
- Basics of Mobile Application Pentests
- Common issues in Mobile Applications
Man In the Middle Attacks
How to MITM

- There are many ways to redirect traffic
How to MITM

• There are many ways to redirect traffic
  – Layer2 attacks
  – Redirect via iptables on a router/switch
  – Android proxy settings (not working in VM)
  – etc.
Task – Basic HTTP Request

- You will find iptables rules in /sdcard/Download/proxy.iptables
  
  ```
  iptables -t nat -A OUTPUT -o eth0 -p tcp --dport 80 -j DNAT --to 10.13.13.102:8080
  ```

- Start HTTP server: sudo python ~/Resources/http/http.py

[10 min]
Basic HTTP Request – Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>iptables -t nat -A OUTPUT -o eth0 -p tcp --dport 80 -j DNAT --to 10.13.13.102:8080</td>
<td>Traffic redirection</td>
</tr>
<tr>
<td>Start Burp with a proxy listening on Port 8080</td>
<td></td>
</tr>
<tr>
<td>Open android application and submit HTTP request</td>
<td></td>
</tr>
<tr>
<td>View Burp Logs</td>
<td></td>
</tr>
</tbody>
</table>
Task - Basic HTTPS Request

• This time the application uses a SSL/TLS secured connection
• Modify your iptables
• Start https server: sudo python ~/Resources/https/https.py

Can you still intercept the traffic?

[10 min]
## Basic HTTPS Request – Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iptables -t nat -A OUTPUT -o eth0 -p tcp --dport 443 -j DNAT --to 10.13.13.102:8080</code></td>
<td>Traffic redirection</td>
</tr>
<tr>
<td>Start Burp with a proxy listening on Port 8080</td>
<td></td>
</tr>
<tr>
<td>Import Burp CA into android VM</td>
<td></td>
</tr>
<tr>
<td>Open android application and submit HTTPS request</td>
<td></td>
</tr>
<tr>
<td>View Burp Logs</td>
<td></td>
</tr>
</tbody>
</table>
Task – SSL Pinning

“Pinning is the process of associating a host with their expected X509 certificate or public key. Once a certificate or public key is known or seen for a host, the certificate or public key is associated or 'pinned' to the host.”

Hint: You do not have to modify the source code!

[10 min]
## SSL Pinning – Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iptables -t nat -A OUTPUT -o eth0 -p tcp --dport 443 -j DNAT --to 10.13.13.102:8080</code></td>
<td>Replace assets/owasp.crt with Burp certificate</td>
</tr>
<tr>
<td>Import Burp CA on android VM</td>
<td></td>
</tr>
<tr>
<td>Open android application and submit HTTPS request</td>
<td></td>
</tr>
<tr>
<td>View Burp Logs</td>
<td></td>
</tr>
</tbody>
</table>
Introduction to smali code

• We will not discuss all kind of syntax elements
• But we will discuss enough to get you prepared for the next tasks
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

// Method declaration for the constructor
public MainActivity() {
    super();
    setContentView(R.layout.activity_main);
}

// Method declaration for the oncreate method
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}

// Method implementation for the oncreate method
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}

// Method implementation for the setcontentview method
public void setContentView(int layoutId) {
    super.setContentView(layoutId);
}

// Method implementation for the onactivitycreated method
@Override
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    setContentView(R.layout.activity_main);
}
class public Lcom/example/context/testproject/MainActivity;
.super Landroid/support/v7/app/AppCompatActivity;
.source "MainActivity.java"

# direct methods
.method public constructor <init>()V
   .locals 0
      .prologue
      .line 12
      invoke-direct {p0}, Landroid/support/v7/app/AppCompatActivity;-><init>()V
      return-void
.end method

# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
   .locals 1
      .param p1, "savedInstanceState"  # Landroid/os/Bundle;
      .prologue
      .line 16
      invoke-super {p0, p1}, Landroid/support/v7/app/AppCompatActivity;->onCreate(Landroid/os/Bundle;)V
      .line 17
      const v0, 0x7f04001a
      invoke-virtual {p0, v0}, Lcom/example/context/testproject/MainActivity;->setContentView(I)V
      .line 19
      return-void
.end method
any of static, private, or constructor
none of static, private, or constructor
any of static, private, or constructor

invoke-direct {p0}, Landroid/support/v7/app/AppCompatActivity;-><init>()V

return-void
.end method

one of static, private, or constructor

invoke-virtual {p0, v0}, Lcom/example/context/testproject/MainActivity;->setContentView(I)V
Registers are used for storing. Starts with 0, can be up to 15 [some "move" instructions can use other, too]
Registers are used for storing. Starts with 0, can be up to 15 [some "move" instructions can use other, too]
Some more about registers

- Further information: https://github.com/JesusFreke/smali/wiki/Registers
class public Lcom/example/context/testproject/MainActivity;
  .super Landroid/support/v7/app/AppCompatActivity;
  .source "MainActivity.java"

# direct methods
.method public constructor <init>()V
  .locals 0
   .prologue
   .line 12
   invoke-direct {p0}, Landroid/support/v7/app/AppCompatActivity;-><init>()V

   return-void
   .end method

# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
  .locals 1
   .param p1, "savedInstanceState"   # Landroid/os/Bundle;
    .prologue
    .line 16
    invoke-super {p0, p1}, Landroid/support/v7/app/AppCompatActivity;->onCreate(Landroid/os/Bundle;)V
    .line 17
    const v0, 0x7f04001a
    invoke-virtual {p0, v0}, Lcom/example/context/testproject/MainActivity;->setContentView(I)V
    .line 19
    return-void
   .end method
# Primitive Types

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>void - can only be used for return types</td>
</tr>
<tr>
<td>Z</td>
<td>boolean</td>
</tr>
<tr>
<td>B</td>
<td>byte</td>
</tr>
<tr>
<td>S</td>
<td>short</td>
</tr>
<tr>
<td>C</td>
<td>char</td>
</tr>
<tr>
<td>I</td>
<td>int</td>
</tr>
<tr>
<td>J</td>
<td>long (64 bits)</td>
</tr>
<tr>
<td>F</td>
<td>float</td>
</tr>
<tr>
<td>D</td>
<td>double (64 bits)</td>
</tr>
</tbody>
</table>
Some more about types

• Further information:
  https://github.com/JesusFreke/smali/wiki/TypesMethodsAndFields
Even more links

- [https://github.com/JesusFreke/smali/wiki](https://github.com/JesusFreke/smali/wiki)
- [http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html](http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html)
Task – Root Detection Bypass

```java
package com.example.context.testproject;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import java.io.File;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        File file = new File("/system/xbin/su");
        if (file.exists()) {
            TextView text = (TextView) findViewById(R.id.root_detection);
            text.setText("Rooted device identified"辖区);
        } else {
            TextView text = (TextView) findViewById(R.id.root_detection);
            text.setText("You got a clean device");
        }
    }
}
```
### Root Detection Bypass – Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>java -jar apktool_2.2.0.jar d &lt;AppName&gt;</code></td>
<td>Decode apk</td>
</tr>
<tr>
<td>Browse to decompiled code and open .smali file</td>
<td>Open file with your favourite editor</td>
</tr>
<tr>
<td>Change e.g. file path, delete check or change if statement</td>
<td>Modify smali code</td>
</tr>
<tr>
<td><code>java -jar apktool_2.2.0.jar b ./app-debug</code></td>
<td>Rebuild</td>
</tr>
<tr>
<td><code>keytool -genkey -alias mystore -keyalg RSA -keystore KeyStore.jks -keysize 2048</code></td>
<td>Create keystore (needs to be done only once)</td>
</tr>
<tr>
<td><code>jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore &lt;name&gt; &lt;apk&gt; &lt;alias&gt; -storepass &lt;pw&gt;</code></td>
<td>Sign apk - otherwise you wouldn't be allowed to install</td>
</tr>
<tr>
<td><code>adb uninstall &lt;app&gt;</code></td>
<td>Uninstall old version</td>
</tr>
<tr>
<td><code>adb install &lt;apk&gt;</code></td>
<td>Install new version</td>
</tr>
</tbody>
</table>
What about new code?

• Sometimes you want to extend the code with your own, e.g. logcat messages

```java
const-string v0, "owasplog"
const-string v1, "Hello"
invoke-static {v0, v1},
Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;)I
```
The “simple” way

- We want to avoid writing our own smali code
- We can use Java2Smali ([https://github.com/ollide/intellij-java2smali](https://github.com/ollide/intellij-java2smali)) which is an Android Studio plugin
Task - Secure Encryption

• Add your own log messages to the application
• Understand the encryption/decryption process
• Break it!
Secure Encryption

- Cipher text `AxoY2sHi1VVSE5MEdbJG0LQVA+PTZATGw/NIdTPw (base64)`
- Based on `android.os.Build.ID`
- Cipher text XOR `android.os.Build.ID`
Java Native Interface

“The Java Native Interface (JNI) is a programming framework that enables Java code running in a Java Virtual Machine (JVM) to call and be called by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages such as C, C++ and assembly.”  
https://en.wikipedia.org/wiki/Java_Native_Interface
Risks that come with JNI

• Vulnerabilities that do not occur within JAVA such as Buffer Overflows

=> Android itself has protections/mitigations against those
Task - Library Call

Your task is to extract the library and build your own project which can use the library!
## Library Call – Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adb pull /data/app/ruhrpott.owasp.com.vuln_app_1.</code></td>
<td>Download apk + library folder</td>
</tr>
<tr>
<td>`strings ./lib/x86/liblibcall.so</td>
<td>grep -i java`</td>
</tr>
<tr>
<td>Create new Android Studio project with same namespace as apk</td>
<td></td>
</tr>
<tr>
<td>Create same class and function name</td>
<td></td>
</tr>
<tr>
<td>Load Library</td>
<td></td>
</tr>
</tbody>
</table>
END OF SESSION 2
Other CTF Challenges

https://drive.google.com/drive/folders/0B_nKK17ymHGpUF9uR1BWRWNCbU0?usp=sharing (goo.gl/LKJRPT)