Reversing Android Apps

Hacking and cracking Android apps is easy
Agenda

• Issues (in the past)
• Android security / code concept
• Techniques for pentesters / reverse engineers
• My experiences and the general quality of apps
My approach

• Bought HTC Desire/Bravo with Android 2.0 (now 2.2.0) in 2010
• Finding security related issues
Issues (in the past?)
Losing phones

I've lost mobile phone. Please call if found.
07910 984 996
Circumventing lock screen
Circumventing lock screen

• Poor lock screen implementation
  – Home button mashing, not all brands <= 2.2
  – Back button during call, not all brands <= 2.0
  – Plug into car dock, unknown
  – Gmail address & password „null“, unknown
• Lock screen not activated
• USB debug on (adb shell)
• Associated Google account
• OpenRecovery, Milestone <= 2.1
• Aquire physical memory (forensic tools)
Google Market – a feel free environment

"what are you fucking to do ~? " + paramString + " is not exsits in this Activity !";
Bring malware to the mobile

• XSS on Google Market website
• Convince users (aka put on market)
• App without permissions installs apps with permissions
  – Angry Birds extra level malware, fixed
  – Browser vulnerability (cookie stealing), < 2.3.5
  – New technique going to be released in November
    • Oberheide/Lanie, Source Barcelona
Other issues

- Facebook-App V. 1.6 is able to read/write/edit SMS/MMS
- Plain authentication tokens, fixed
- SMS receiver incorrect, fixed
- Htclogger, HTC only
- App reversing
- Many more
Nuclear chain of command...

![Diagram](xkcd.com)
... is similar to the Android chain of security
My situation

- Bought HTC Desire in 2010
- Still on Android 2.2.0, means:
  - Screen lock circumvention (button mashing)
  - Vulnerable to DroidDream malware
  - Browser vulnerability
    - Cookie stealing / XSS
    - Can be used to install apps
Android security / code concept
Security Concept

- Dalvik Java VM is no security layer
- Permission system
- Android is a Linux
- One app = one Linux user
Android code

• Write app in Java and HTML/Javascript (Android SDK)
  – The obvious approach
  – Most apps from the Google Market
  – Easy to decompile/disassemble/reassemble
• Write app in ARM native code (Android NDK)
  – Together with Java code
  – ARM Assembler Reverse Engineering and JNI
• Use a framework/generator
  – appmakr.com
  – PhoneGap
  – Others?
Techniques for pentesters / reverse engineers
1. Getting hundreds of Android Apps (apk files)
Obvious download approach

- Open market app on mobile
- Click app and install
- SCP apk file from phone
  → Too slow, not enough space on mobile, etc
How to download all Android apps

• Connect mobile to laptop Wifi with airbase-ng / dnsmasq
• Use iptables to redirect to local Burp
  – thx Android for not having a proxy option
• BurpExtender to save responses with apk files
• Send mobile a HTTP 404 not found
Install all apps?

• One HTTPS request to market.android.com
• Change the app name
  – com.google.android.youtube
• Modified w3af spider / regex plugin
  – Search for terms A ... ZZ on market.android.com
  – No restrictions (e.g. Captcha) as in Google search
• Wrote script that sends HTTPS requests with app name
Download environment
Metadata

- About 300’000 apps in market
- Crawled about 10’000 app names
- Successfully downloaded and decompiled about 3’500 apps (about 15 GB)
  - Took about 3 days to download all these apps
2. Decompile/disassemble
The apktool disassembled structure

- Apk unzipped
  - assets
  - res
    - drawable
      - icon.png
    - layout
      - main.xml
  - values
    - strings.xml
  - META-INF
  - AndroidManifest.xml
  - classes.dex

→ apktool disassembled

+assets
+res
  + drawable
    - icon.png
  + layout
    - main.xml
+values
  - strings.xml
+META-INF
  - AndroidManifest.xml
+smali
  + com
    + ...
  - apktool.yml
Two approaches

• Disassembling to smali
  – Similar to Jasmin syntax (Java assembler code)
  – Apktool
    • Correct smali code
    • Didn’t use dexdump/dedexer

• Decompiling to Java
  – Dex2Jar + Java-Decompiler
    • Sometimes incorrect Java code
Disassembling howto

- Apktool

```bash
me$ java -jar apktool.jar d app.apk output-folder
```
Disassembled example

.method public isAuthenticated()Z
  .locals 1
  .prologue
  .line 635
  iget-object v0, p0, Lcom/dropbox/client2/DropboxAPI;->mClient:Lcom/dropbox/client2/DropboxClient;
  if-eqz v0, :cond_0
  const/4 v0, 0x1
  :goto_0
  return v0
  :cond_0
  const/4 v0, 0x0
  goto :goto_0
.end method
Reasssembling howto

• Apktool

```bash
me$ echo "change something"
change something
me$ java -jar apktool.jar b output-folder/ fake-app.apk
[…]
me$ keytool -genkey -alias someone -validity 100000 -keystore someone.keystore
[…]
me$ jarsigner -keystore someone.keystore fake.apk someone
me$ adb install fake-app.apk
```
3. Other techniques for pentesters
Heap dump

me$ su
me# ps | grep kee
   949 10082  183m S   com.android.keepass
   960 0    1964 S    grep kee
me# kill -10 949
me# grep password /data/misc/heap-dump-tm1312268434-pid949.hprof
thisisasecretpassword

• In Android > 2.3
  – Button in DDMS tool or call android.os.Debug.dumpHprofData(fileName)
Invoking Activities

• Activities are basically user interfaces
  — „one screen“

```bash
me$ dumpsys package > packages.txt
me$ am start -n com.android.keepass/com.keepassdroid.PasswordActivity
```

• Fortunately this example doesn’t work
Tons of other tools

- Androguard
- Apkinspecton
  - GUI combining apktool, dex2jar, a Java decompiler, byte code, etc.
- DED
- androidAuditTools
- Smartphonesdumbapps
- Taintdroid (Privacy issues)
- Android Forensic Toolkit
- viaExtract
- More
Experiences when decompiling/disassembling 3‘500 apps

Finding security related issues
Metadata

• About 3’500 apps
  – 2’300 unique email addresses
  – 1’000 «fuck»
  – Several twitter / facebook / flickr / geocaching API keys
Low hanging fruits
Hashing and encryption – a short best practices refresh

• Secure algorithms/implementations
• Random, long salts/keys
• Hashing
  – Separate salt for every hash
  – Several hashing rounds
    • E.g. hash(hash( ... hash(pwd+salt)+salt ... ))
• Encryption
  – Keep the key secret
Key: MSB always 0

Used for sending passwords in HTTPS

```java
byte[] array0fByte1 = { 110, 72, 113, 80, 114, 89, 52, 52, 68, 115, 55, 71, 104, 98, 72, 71 };
sKey = new SecretKeySpec(array0fByte1, "AES");
sKeySize = 16;
sIvBytes = new byte[16];
byte[] array0fByte2 = sIvBytes;
sIvSpec = new IvParameterSpec(array0fByte2);
sPaddingChar = 32;
```
Used to signalise the server that in-game goods were purchased
private String passphrase = "********P4SSw0rD";
String str1 = "pLe@sEDØn'TcRackME";
Who calls this „ah“ constructor?
Obfuscated code

• 4 greps later...
• c.f includes the key
  – c.f calls a.bs(key)
    • a.bs calls a.ah(key)
      – a.ah uses the key and locale variables for encryption
• We know all the input data for the encryption routine
• It’s symmetric crypto
• We can decrypt „it“ (whatever it might be)
TestXXXXX.java

- Yeah, let’s copy/paste a test email!
TestXXXXX2.java

• And credentials for the test server...
Some apps I looked at more closely

(it’s getting worse)
App 1 - banking app

• Who really wants banking on the mobile?
• A lot of banking apps! Yay!
• App 1
  – No obfuscation + can easily be recompiled
  – App simply shows the website
  – Hides the URL and SSL cert/lock from the user
  – Can only be used with mTAN
App 2

- Server had self-signed SSL certificate
- SSL MITM Dump:

```
/username=B1436A 13E85D20 F2428D6E 232C2B93FE....password=2C30F3866 016E6C59 52655C06
400BCC6. imei=405 23204606 E450...
```

Wow, it’s encrypted... Don’t we need a key for that?
App 2

• AES key

```java
public byte[] cryptKey42 = {-31, -21, 4, 24, -21, 54, -63, -40, -38, 61, -47, -115, -95, -36, -142, 64, 53, 120, -85, -96, -69, 85, 81, 16, -36, 80, -102, 95, -20, 110, 36, -11};
```
private boolean deviceRoot()
{
    try{
        Runtime.getRuntime().exec("su");
        return true;
    }
    catch (IOException localIOException){
        return false;
    }
}

App 3 – Circumventing root detection

- Not necessary
App 4 – Another root detection

```java
public static boolean isDeviceRooted()
{
    File f = new File("/system/sbin/su")
    return f.exists()
}
```
App 4 - Removing root detection

```bash
me$ java -jar apktool.jar d app.apk source
[...]
me$ sed -i """" 's/system\(/sbin\)/su/system\(/sbin\)/CEW1PFSLK/g' source/smali/net/example/checks.smali
me$ java -jar apktool.jar b source/ fake.apk
[...]
me$ keytool -genkey -alias someone -validity 100000 -keystore someone.keystore
[...]
me$ jarsigner -keystore someone.keystore fake.apk someone
me$ adb install fake.apk
```
App 4 – Was that a good method to remove the root detection?

• Altering the app
  – No updates

• We only want to fail that simple check
App 4 - Prevent root detection

root stays root!

me$ adb shell
$ su
# cd /system/bin/; mount -o remount,rw -o rootfs rootfs /;
mount -o remount,rw -o yaffs2 /dev/block/mtdblock3 /system
# echo $PATH
/sbin:/system/sbin:/system/bin:/system/xbin
# mv /system/sbin/su /system/xbin/
A special secret key

• 445 apps use the same AES key
  – byte[] a = { 10, 55, -112, -47, -6, 7, 11, 75, -7, -121, 121, 69, 80, -61, 15, 5 }
Google Ads

- Encrypt last known location
  - All location providers (GPS, Wifi, ...)
- Send via the „uule“ JSON parameter
- Notified Google on the 23th of June
  - No response yet
- To be honest I haven‘t seen the „uule“ parameter in my network yet
Google Ads

• Why didn’t they use asymmetric crypto?
Countermeasures

• Use asymmetric crypto instead of symmetric when transferring data to a server
• Store hashes/session tokens instead of passwords
• Good obfuscation is Security Through Obscurity
• Pentest your apps
• Know the limitations
  – root stays root
References

- http://designora.com/graphics/android-logo/
- http://www.slideshare.net/bsideslondon/bsideslondon-spo#text-version
- https://theassurer.com/p/756.html
- http://thomascannon.net/blog/2011/02/android-lock-screen-bypass/
- http://www.xkcd.com/898
- http://thomascannon.net/projects/android-reversing/
Thx!

DREAMLAB TECHNOLOGIES

- Twitter: floyd_ch
- http://floyd.ch