Hack.lu

Mobile application testing

Who are we?

- Arthur Donkers
  - arthur@1secure.nl

- Frank Spierings
  - frank@warpnet.nl

What’s this all about?

Sharing our experiences, tips and tricks on testing mobile applications

(preferably on non-jailbroken and non-rooted devices)

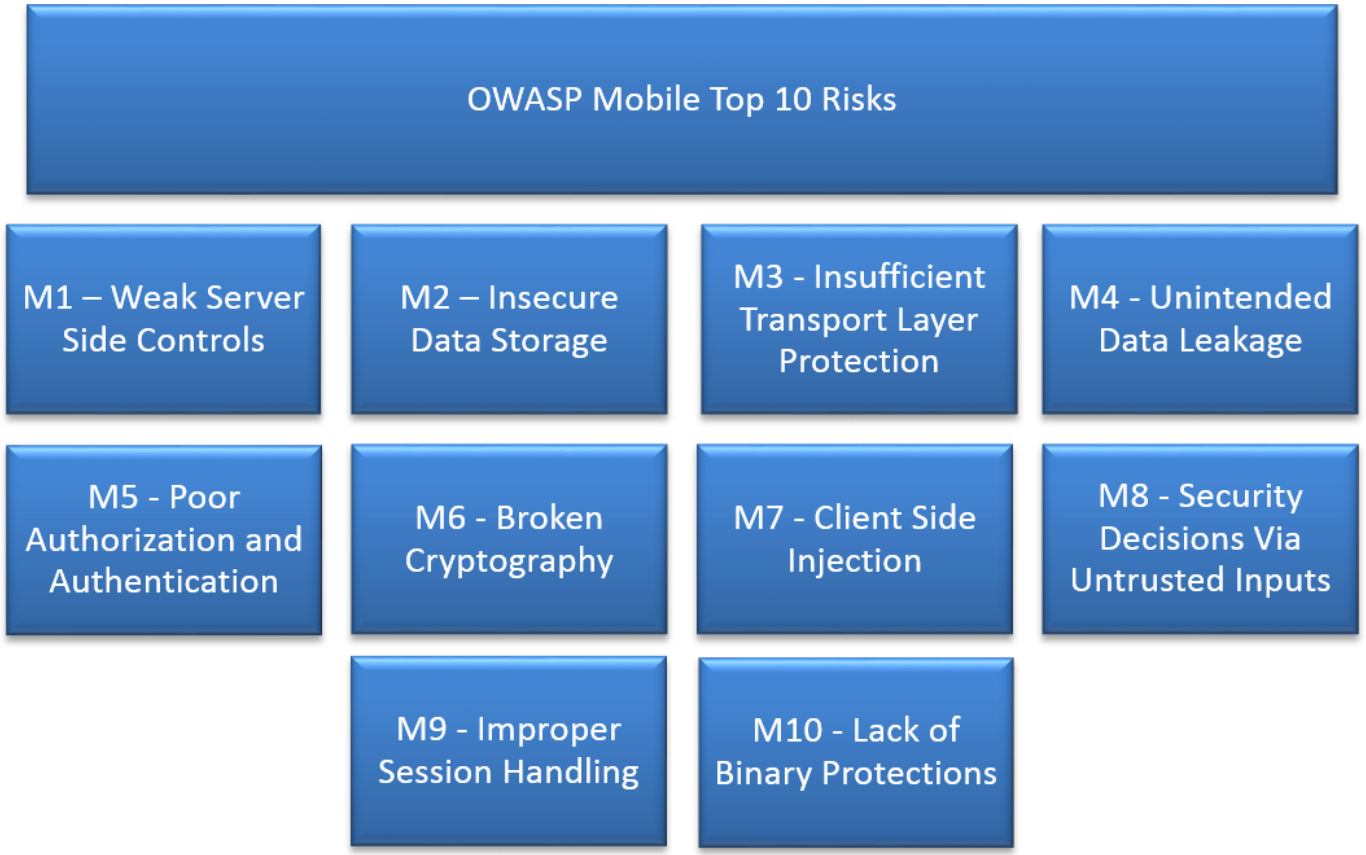
Where are the goodies?

https://github.com/theart42/hack.lu

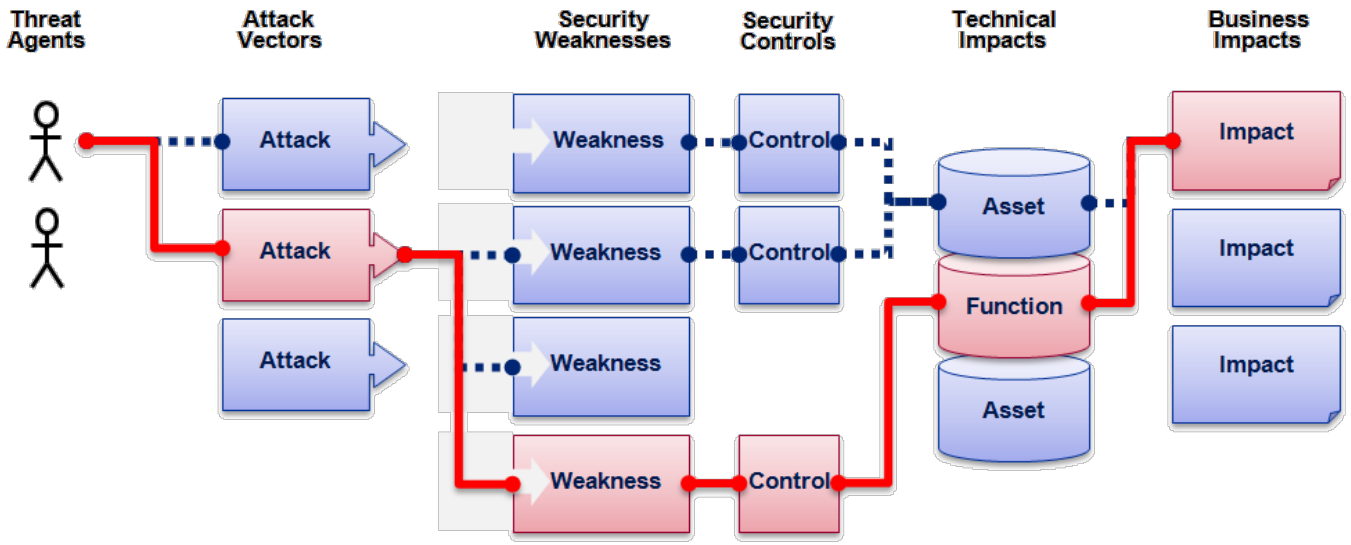
https://github.com/theart42/omed.ios

https://github.com/theart42/omed.android

Web Application hack techniques



OWASP assessment model



6

M1 – Weak server side controls, risk

- In order for this vulnerability to be exploited, the organization must expose a web service or API call that is consumed by the mobile app. The exposed service or API call is implemented using insecure coding techniques that produce an OWASP Top Ten vulnerability within the server.
- Through the mobile interface, an adversary is able to feed malicious inputs or unexpected sequences of events to the vulnerable endpoint. Hence, the adversary realizes the original OWASP Top Ten vulnerability on the server.

M1 – prevention

- Secure coding and configuration practices must be used on server-side of the mobile application. For specific vulnerability information, refer to the OWASP Web Top Ten or Cloud Top Ten projects

M2 – Insecure data storage, risk

- Insecure data storage vulnerabilities occur when development teams assume that users or malware will not have access to a mobile device's filesystem and subsequent sensitive information in data-stores on the device
  - (Jailbreak/rooting!)
- Filesystems are easily accessible. Organizations should expect a malicious user or malware to inspect sensitive data stores. Rooting or jailbreaking a mobile device circumvents any encryption protections. When data is not protected properly, specialized tools are all that is needed to view application data.

M2 – prevention

- The cardinal rule of mobile apps is to not store data unless absolutely necessary
- You also have to consider the implications of losing mobile users' data to a silent jailbreak or root exploit
- The lesson here is to know what data is being stored and protect it appropriately

M3 – Insufficient Transport protection, risk

- Mobile applications frequently do not protect network traffic. They may use SSL/TLS during authentication but not elsewhere. This inconsistency leads to the risk of exposing data and session IDs to interception.
- The use of transport security does not mean the app has implemented it correctly
- To detect basic flaws, observe the phone's network traffic. More subtle flaws require inspecting the design of the application and the applications configuration.

M3 – prevention

- Assume that the network layer is not secure and is susceptible to eavesdropping
- Apply SSL/TLS to transport channels that the mobile app will use to transmit sensitive information, session tokens, or other sensitive data to a backend API or web service
- Account for outside entities like third-party analytics companies, social networks, etc. by using their SSL versions when an application runs a routine via the browser/webkit. Avoid mixed SSL sessions as they may expose the user's session ID.
- Use strong, industry standard cipher suites with appropriate key lengths
- Use certificates signed by a trusted CA provider

M3 – prevention 2

- Never allow self-signed certificates, and consider certificate pinning for security conscious applications
- Always require SSL chain verification
- Only establish a secure connection after verifying the identity of the endpoint server using trusted certificates in the key chain
- Alert users through the UI if the mobile app detects an invalid certificate.
- Do not send sensitive data over alternate channels (e.g, SMS, MMS, or notifications)
- If possible, apply a separate layer of encryption to any sensitive data before it is given to the SSL channel. In the event that future vulnerabilities are discovered in the SSL implementation, the encrypted data will provide a secondary defense against confidentiality violation.

M4 – Unintended data leakage, risk

- Unintended data leakage occurs when a developer inadvertently places sensitive information or data in a location on the mobile device that is easily accessible by other apps on the device
- First, a developer's code processes sensitive information supplied by the user or the backend. During that processing, a side-effect (that is unknown to the developer) results in that information being placed into an insecure location on the mobile device that other apps on the device may have open access to.
- Typically, these side-effects originate from the underlying mobile device's operating system (OS)

M4 – prevention

- It is important to threat model your OS, platforms, and frameworks, to see how they handle the following types of features:
  - URL Caching (Both request and response)
  - Keyboard Press Caching
  - Copy/Paste buffer Caching
  - Application backgrounding
  - Logging
  - HTML5 data storage
  - Browser cookie objects
  - Analytics data sent to 3rd parties

M5 – Poor authentication authorization, risk

- Poor or missing authentication schemes allow an adversary to anonymously execute functionality within the mobile app or backend server used by the mobile app
- Weaker authentication for mobile apps is fairly prevalent due to a mobile device's input form factor. The form factor highly encourages short passwords that are often purely based on 4-digit PINs.
- To test for poor authorization schemes, testers can perform binary attacks against the mobile app and try to execute privileged functionality that should only be executable with a user of higher privilege while the mobile app is in 'offline' mode

M5 – prevention

- Developers should assume all client-side authorization and authentication controls can be bypassed by malicious users. Authorization and authentication controls must be re-enforced on the server-side whenever possible.
- Due to offline usage requirements, mobile apps may be required to perform local authentication or authorization checks within the mobile app's code. If this is the case, developers should instrument local integrity checks within their code to detect any unauthorized code changes. See M10 for more information about detecting and reacting to binary attacks

M6 – Broken cryptography, risk

- In order to exploit this weakness, an adversary must successfully return encrypted code or sensitive data to its original unencrypted form due to weak encryption algorithms or flaws within the encryption process

M6 – prevention

- Use reliable libraries
- Skilled programmers
- Proper key management (Keychain on iOS, Keystore on Android)

M7 – Client side injection, risk

- Client-side injection results in the execution of malicious code on the mobile device via the mobile app
- Typically, this malicious code is provided in the form of data that the threat agent inputs to the mobile app through a number of different means
- The data is malformed and is processed (like all other data) by the underlying frameworks supporting the mobile app as executable code
- The code is malicious in nature and executed by the app

M7 – prevention

- In general, protecting you application from client side injection requires looking at all the areas your application can receive data from and applying some sort of input validation
- In certain cases this is simple but for others it is more complex

M8 – Security decisions via insecure data, risk

- Developers generally use hidden fields and values or any hidden functionality to distinguish higher level users from lower level users. An attacker can intercept the calls (IPC or web service calls) and temper with such sensitive parameters. Weak implementation of such functionalities leads to improper behavior of an app and even granting higher level permissions to an attacker.This can easily be exploited through hooking functionality.

M8 – prevention

- Scrutenize input
- Validate before use
- Only accept if it is completely screened
- Screen ALL input
- Use secure development techniques

M9 – Improper session handling, risk

- In order to facilitate a stateful transaction between a user and a mobile app's backend servers, mobile apps use session tokens to maintain state over stateless protocols like HTTP or SOAP. To maintain state, the mobile app must use a cookie, and add this cookie to all future service transactions between the mobile app and the server.
- Improper session handling occurs when the session token is unintentionally shared with the adversary during a subsequent transaction between the mobile app and the backend servers

M9 – prevention

- To handle sessions properly, ensure that mobile app code creates, maintains, and destroys session tokens properly over the life-cycle of a user's mobile app session

M10 – Lack of binary protections, risk

- A lack of binary protections within a mobile app exposes the application and it's owner to a large variety of technical and business risks if the underlying application is insecure or exposes sensitive intellectual property. A lack of binary protections results in a mobile app that can be analyzed, reverse-engineered, and modified by an adversary in rapid fashion.

M10 – prevention

- First, the application must follow secure coding techniques for the following security components within the mobile app:
  - Jailbreak Detection Controls
  - Checksum Controls
  - Certificate Pinning Controls
  - Debugger Detection Controls

M10 – prevention 2

- Next, the app must adequately mitigate two different technical risks that the above controls are exposed to:
  - The organization building the app must adequately prevent an adversary from analyzing and reverse engineering the app using static or dynamic analysis techniques
  - The mobile app must be able to detect at runtime that code has been added or changed from what it knows about its integrity at compile time. The app must be able to react appropriately at runtime to a code integrity violation

2016

- M1 Improper Platform usage:
    - Not using platform provided security features (not using Android permissions, iOS Touch ID and keychain etc)
    - Not following coding guidelines etc

2016

- M2 Improper data storage:
    - Combination of M2 and M4 from previous top 10

2016

- M3 Insecure communications:
    - Not using SSL/TLS or not properly using SSL/TLS (or no encryption at all!)

2016

- M4 Insecure authentication:
    - Failing to properly identify the user (relying on lock screen of device?)
    - Weak session management (reusing sessions)
    - Failing to maintain identity of user
    - Insecure storage of long term cookies on device

2016

- M5 Insufficient cryptography:
    - Failing to properly use cryptography for local storage
    - Inventing your own cryptography
    - Poor protection of keys

2016

- M6 Insecure authorization:
    - Authorization done on client side
    - Not linking logon identity to data access
    - Enrollment weaknesses

2016

- M7 Client code quality:
  - ‘Old’ M8
  - Poor checks on user inputs
  - Insecure programming techniques

2016

- M8 Code tampering:
    - Unable to detect code manipulation on device
    - Lack of runtime protections

2016

- M9 Reverse engineering:
    - Storing sensitive information in application (without proper protection)

2016

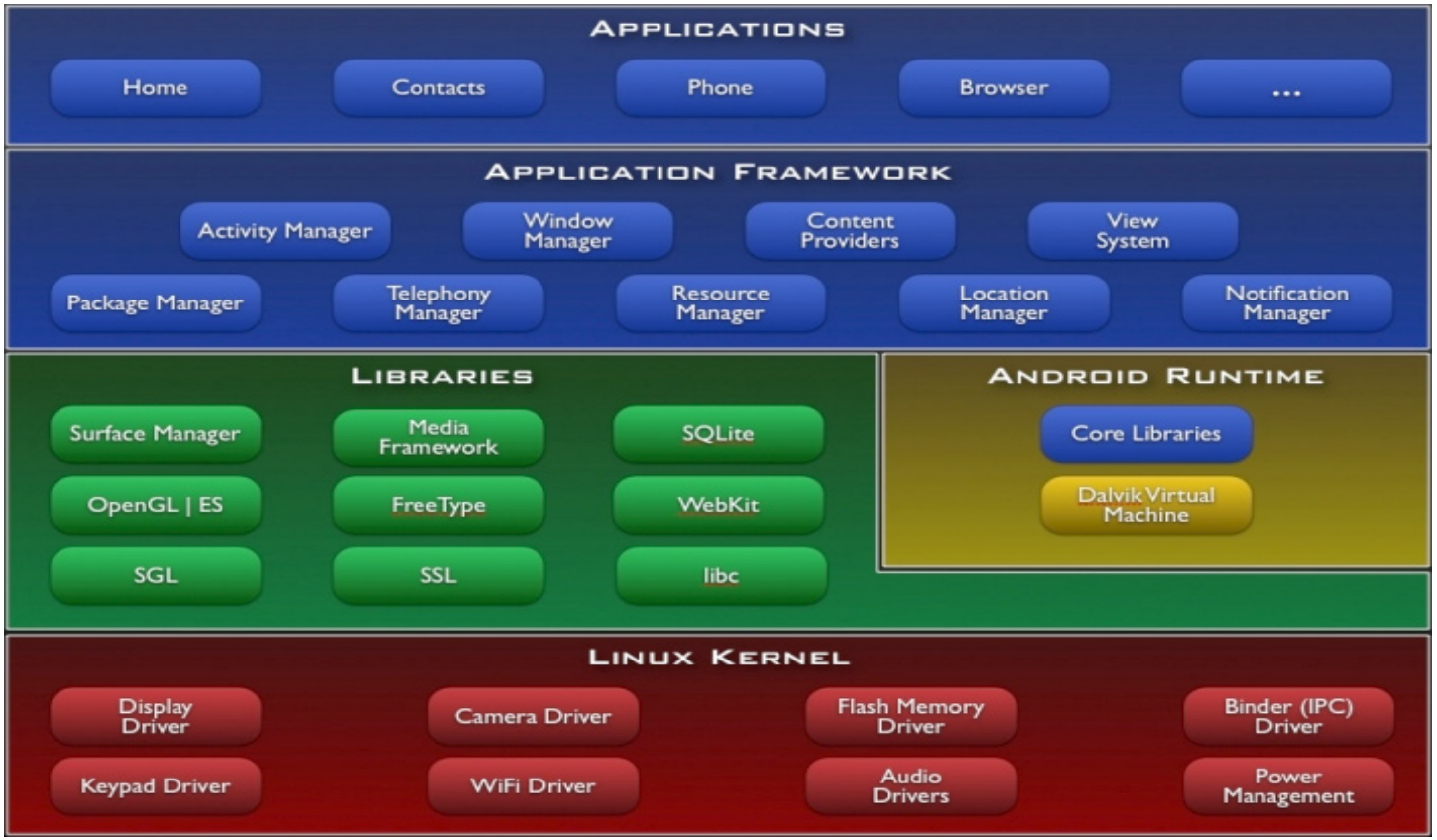
- M10 Extraneous functionality:
    - Backdoors
    - Debugging code in production

Android application testing

Android application testing

- If you want to do it yourself:
  - Android SDK: http://developer.android.com/sdk/index.html#download
  - APKTool: https://code.google.com/p/android-apktool/downloads/list
  - JD-GUI: http://jd.benow.ca/
  - Dex2Jar: https://code.google.com/p/dex2jar/downloads/list
  - Burp Proxy: http://portswigger.net/burp/download.html
  - Andriller: http://android.saz.lt/cgi-bin/download.py
  - Python 3.0: http://python.org/download/releases/3.0/
  - AFLogical: https://github.com/viaforensics/android-forensics
  - SQLite Browser: http://sourceforge.net/projects/sqlitebrowser/
  - Drozer: https://www.mwrinfosecurity.com/products/drozer/community-edition/
  - Lobotomy: https://github.com/LifeForm-Labs/lobotomy

Android application testing



Android application testing

- But first….
- Prepare your device:
  - Enable USB debugging
  - Unlock bootloader
  - Root it

Android application testing

- Rooting your device means installing linux software that grants normal users root privileges
  - Allows you to bypass security
  - Is different for each device
  - May break your device and will void your warranty!

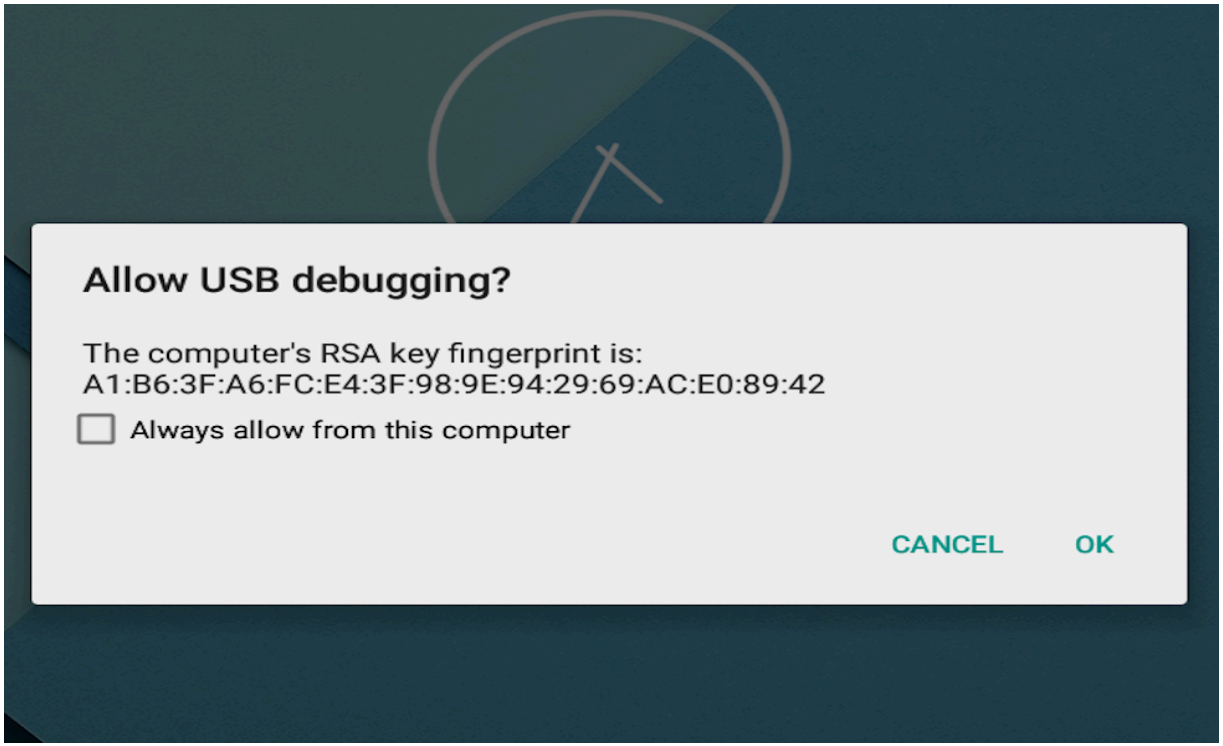
Android application testing

- ADB (Android Debugging Bridge) is one of the first tools to use
- Is part of the platformtools part of the SDK

```
$ adb devices
List of devices attached
emulator-5554   device
```

- If you connect with ADB for the first time, you need to accept the certificate of the computer on the device

Android application testing



Android application testing

Android boot process:

- The boot process is essential in securing Android.
- If you can intervene you can inject and manipulate the Android Platform.
- *(This is actually necessary when rooting an Android device)*.
- For Nexus 7 (2012): http://www.ibtimes.co.uk/how-root- nexus-7-2012-wi-fi-nexus-10-stock-android-5-0-lollipop-via-cf-auto-root-1474771

Android application testing

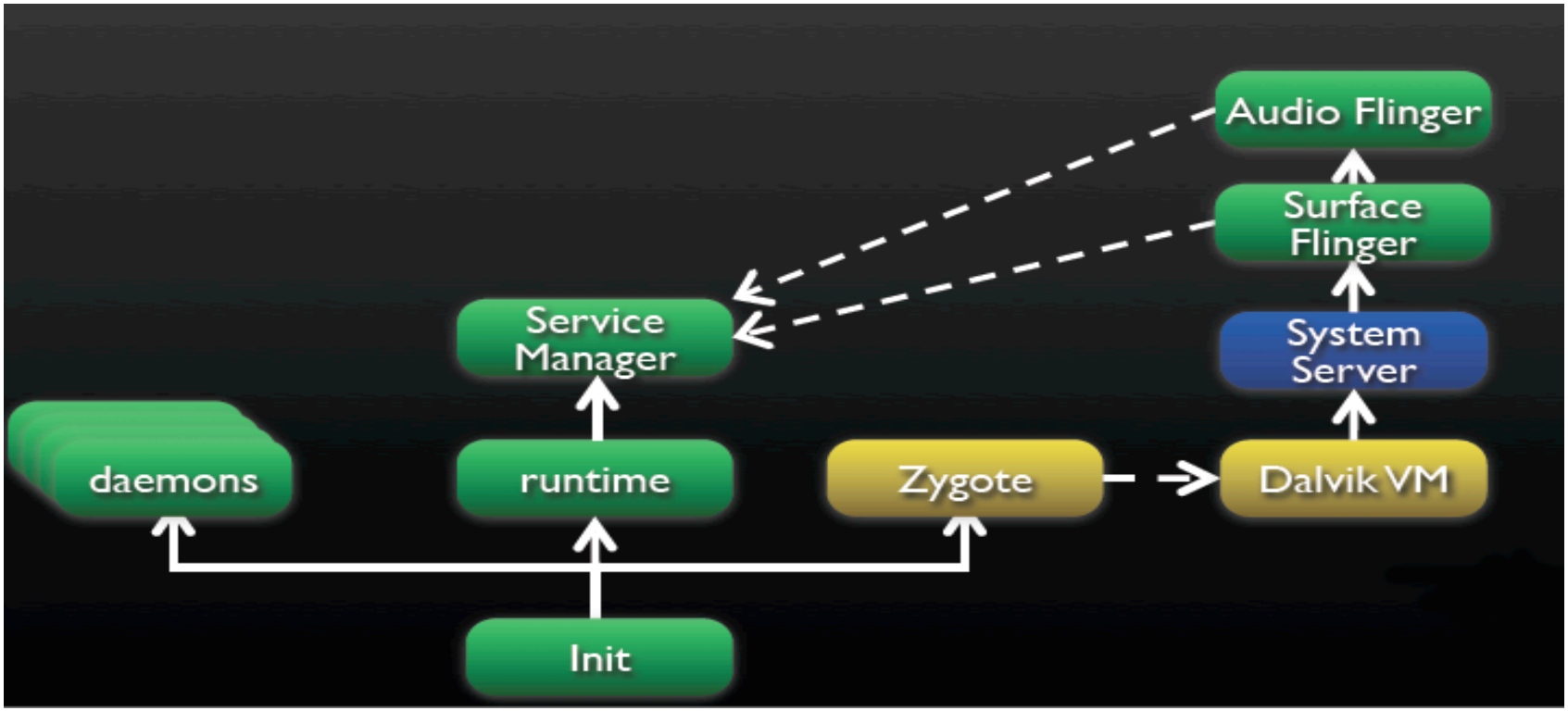
Android boot process:

1. BIOS starts the device (hardware).
2. The BIOS loads the kernel.
3. The kernel creates the first process:
    - Init
    - Mount directories, reads init.rc and other startup files
4. init will start other processes (adbd, vold etc.)
    - In /system there is a build.prop file that also contains additional startup and device information

Android application testing

- Android boot process
  - Once all system processes are started, init will startup Zygote
  - Zygote is responsible for creating Dalvik Virtual machines and loading libraries etc.
  - Applications are spawned from Zygote

Android application testing



Android application testing

- To create your testing environment (examples on Kali):
  - Install the android-sdk (make sure to use the proper java version!):
    - apt-get install openjdk-7-jdk
    - apt-get install android-sdk
- If you're on a 64 bit machine (older kali):
  - apt-get install lib32ncurses5 lib32stdc++6

Android application testing

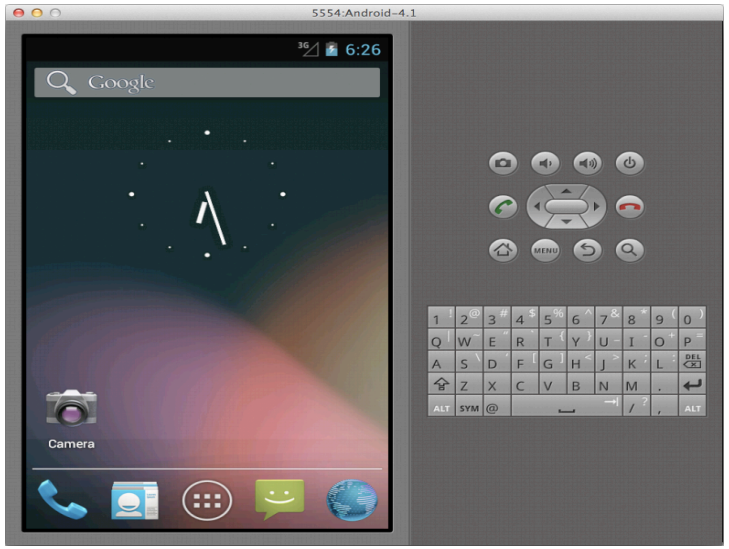
- Use android manager to install platform-tools and the SDK at least

- On Kali this is all installed in /usr/share/android-sdk
- Important subdirectories: tools and platform-tools
- Add them to you PATH so you can find all the commands

- Don’t forget to select the proper java if you have more installed:
  - `root@kali:~# update-java-alternatives -l`
  - `java-1.6.0-openjdk-amd64 1061 /usr/lib/jvm/java-1.6.0-openjdk-amd64`
  - `java-1.7.0-openjdk-amd64 1051 /usr/lib/jvm/java-1.7.0-openjdk-amd64`

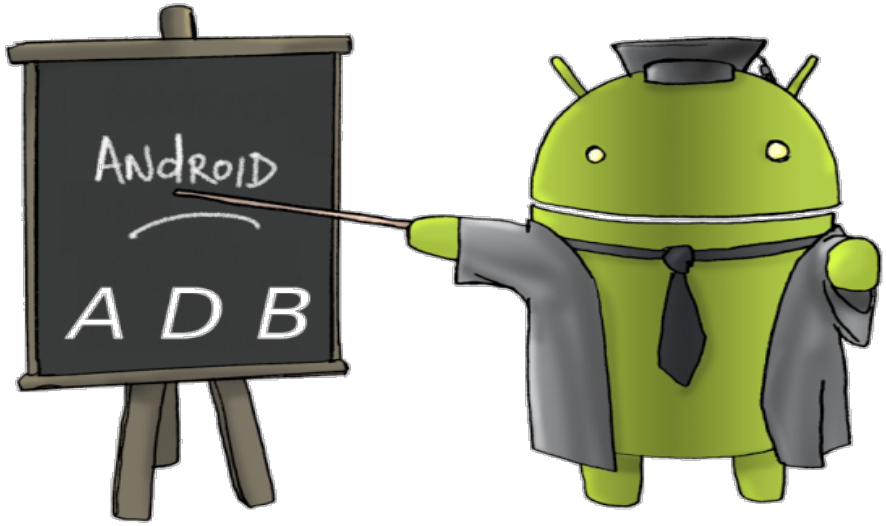
Android application testing

- You can also install Android emulators and virtual devices.
- In my experience this does not provide:
  - a reliable test environment
  - better to buy a Nexus phone and use that as a testbed.



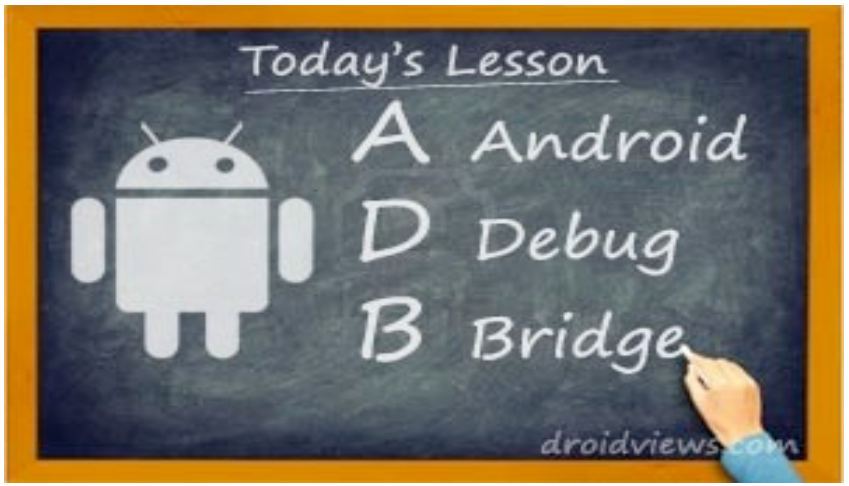
Android application testing

- adb really is your friend….

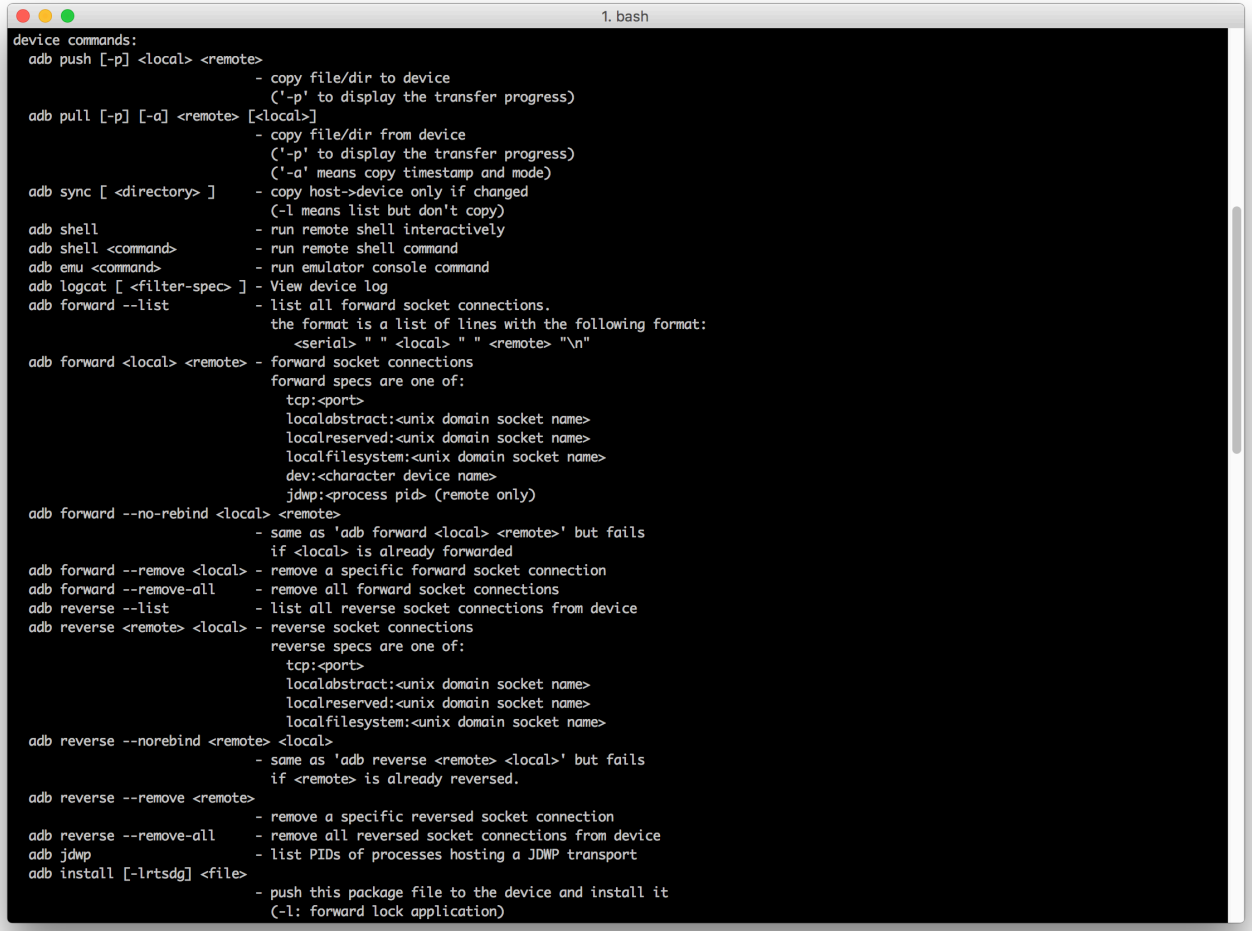


Android application testing

- adb push and pull allow you to put and get files
- adb install allows you to install apk (package) files
- adb backup allows you to make backups
- Etc.

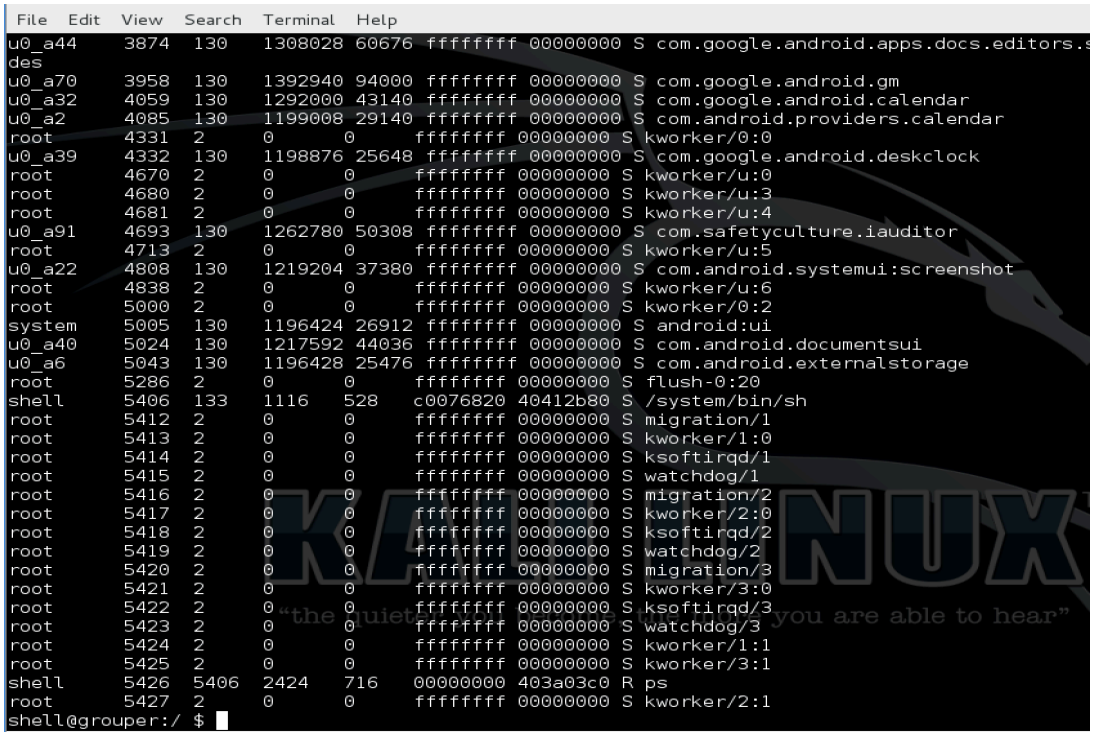


Android application testing

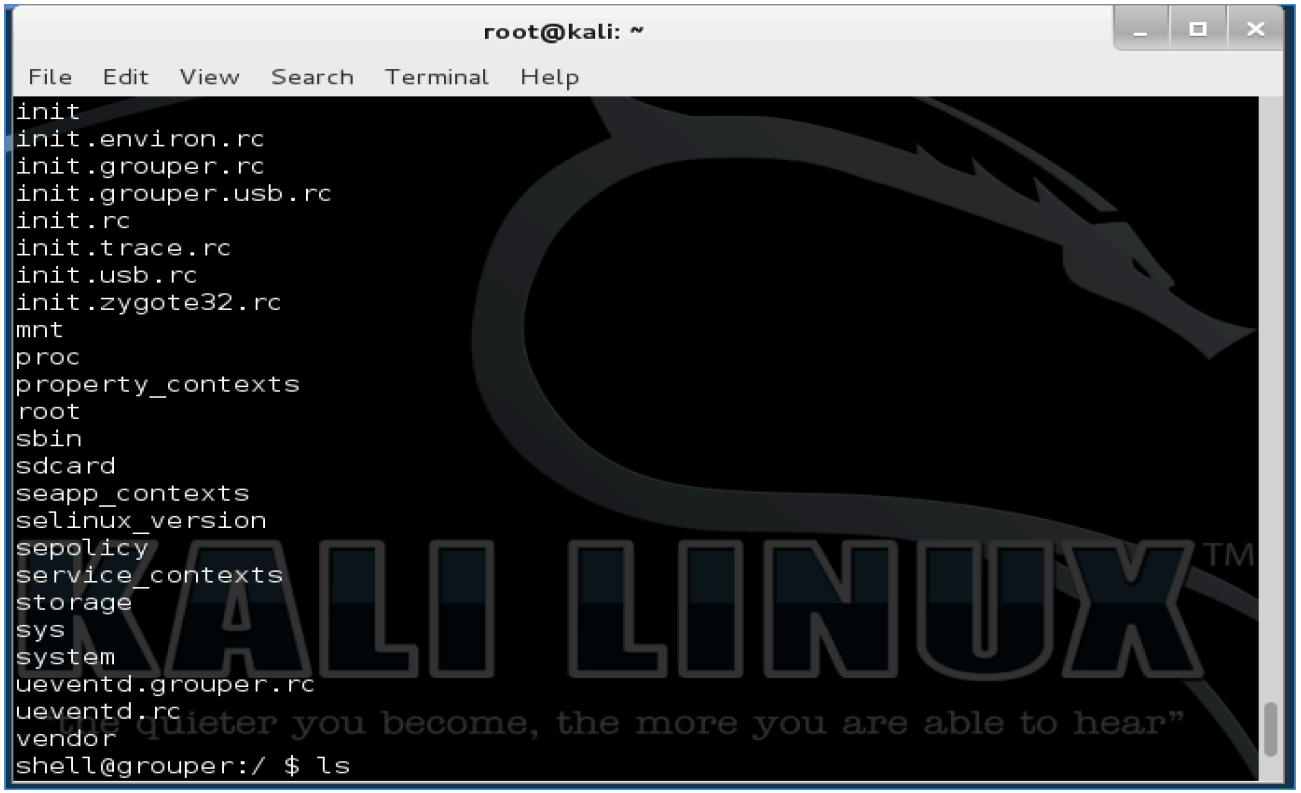


Android application testing

- adb shell command allows you to access the device and browse around

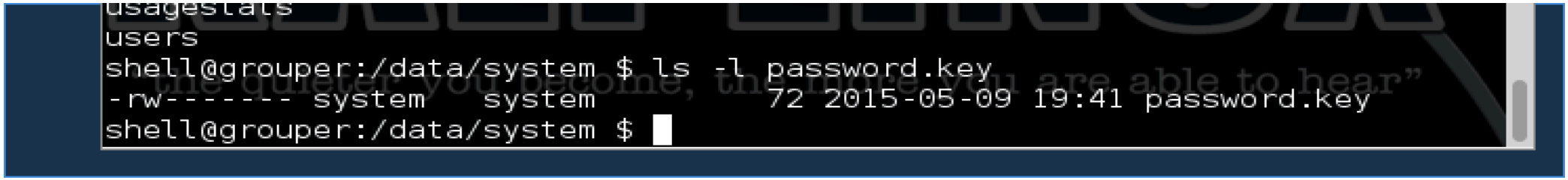


Android application testing



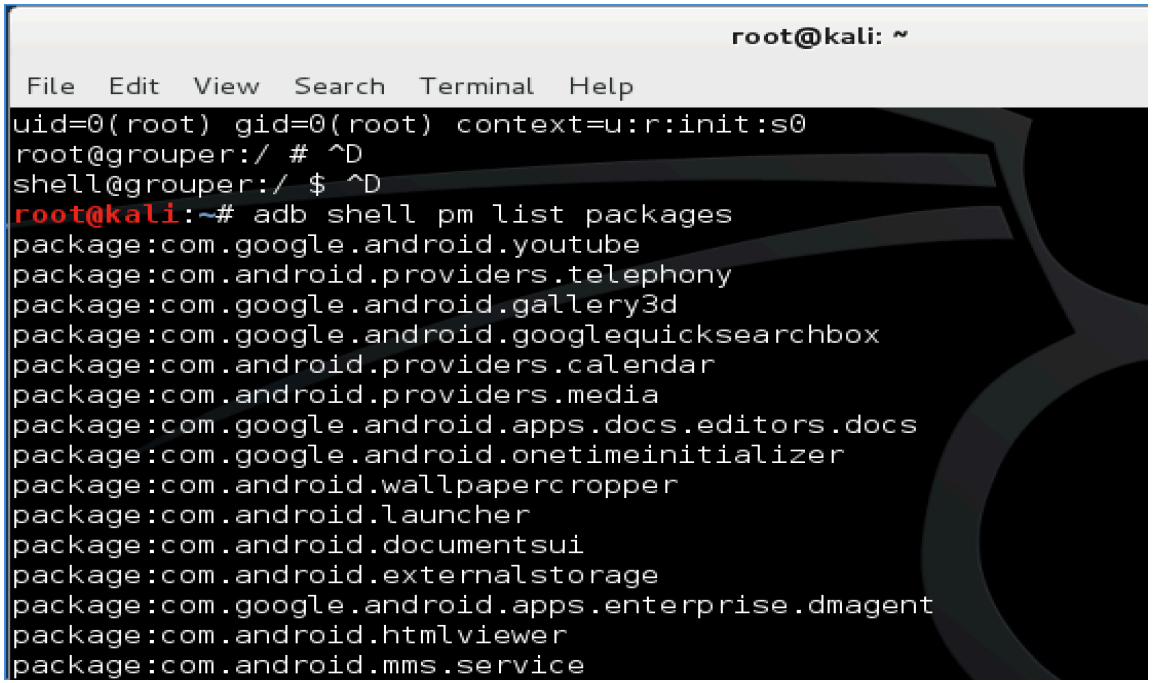
Android application testing

- Once USB debugging is activated, and the phone is rooted, interesting things may happen…
- Passwords and gestures are stored in /data/system



Android application testing

- List installed packages: adb shell pm list packages



Android application testing

- Look at the logfiles
  - adb logcat –d –f /mnt/sdcard/logcats.log

```
shell@grouper:/ $ ^D
root@kali:~# adb logcat -d -f /mnt/sdcard/logcats.log
root@kali:~# adb shell
shell@grouper:/ $ cd /mnt/sdcard
shell@grouper:/mnt/sdcard $ ls -l
drwxrwx--- root     sdcard_r          2015-05-09 17:21 Alarms
drwxrwx--x root     sdcard_r          2015-06-13 17:16 Android
drwxrwx--- root     sdcard_r          2015-05-09 19:18 DCIM
drwxrwx--- root     sdcard_r          2015-06-13 16:26 Download
drwxrwx--- root     sdcard_r          2015-05-10 17:23 MifareClassicTool
drwxrwx--- root     sdcard_r          2015-05-09 17:21 Movies
drwxrwx--- root     sdcard_r          2015-05-09 17:21 Music
drwxrwx--- root     sdcard_r          2015-05-09 19:06 Notifications
drwxrwx--- root     sdcard_r          2015-05-09 19:14 Pictures
drwxrwx--- root     sdcard_r          2015-05-09 17:21 Podcasts
drwxrwx--- root     sdcard_r          2015-05-09 17:21 Ringtones
-rw-rw---- root     sdcard_r    487863 2015-06-13 18:14 logcats.log
shell@grouper:/mnt/sdcard $
```

Android application testing

- Modifying the app to be able to test it, but you run into some obstacles:
  - Runtime signature verification
  - Runtime rooting detection
  - Runtime debuggable detection
  - Certificate pinning

Android application testing

- To intercept HTTP(S) traffic, use burpsuite, similar to iOS…
- However…
  - On Android it is not as easy to set a system wide proxy, better use the transparent proxy, Access Point and redirection technique.

Android application testing

- Another great tool you need is apktool
  - This tool is used to manage Android install packages (apk files)
  - Current version on Kali is 1.5.2-kali or 2.0.0
  - New 2.0 version is available at https://ibotpeaches.github.io/Apktool/.
  - This version also support Android 5 (lollipop) and higher

Android application testing

```
root@kali:~# java -jar apktool2.jar
Apktool v2.0.0 - a tool for reengineering Android apk files
with smali v2.0.5 and baksmali v2.0.5
Copyright 2014 Ryszard Wiśniewski <brut.alll@gmail.com>
Updated by Connor Tumbleson <connor.tumbleson@gmail.com>

usage: apktool
 -advance,--advanced   prints advance information.
 -version,--version    prints the version then exits
usage: apktool if|install-framework [options] <framework.apk>
 -p,--frame-path <dir>   Stores framework files into <dir>.
 -t,--tag <tag>          Tag frameworks using <tag>.
usage: apktool d[ecode] [options] <file_apk>
 -f,--force              Force delete destination directory.
 -o,--output <dir>       The name of folder that gets written. Default is apk.out
 -p,--frame-path <dir>   Uses framework files located in <dir>.
 -r,--no-res             Do not decode resources.
 -s,--no-src             Do not decode sources.
 -t,--frame-tag <tag>    Uses framework files tagged by <tag>.
usage: apktool b[uild] [options] <app_path>
 -f,--force-all          Skip changes detection and build all files.
 -o,--output <dir>       The name of apk that gets written. Default is dist/name.apk
 -p,--frame-path <dir>   Uses framework files located in <dir>.

For additional info, see: http://ibotpeaches.github.io/Apktool/
For smali/baksmali info, see: http://code.google.com/p/smali/
root@kali:~#
```

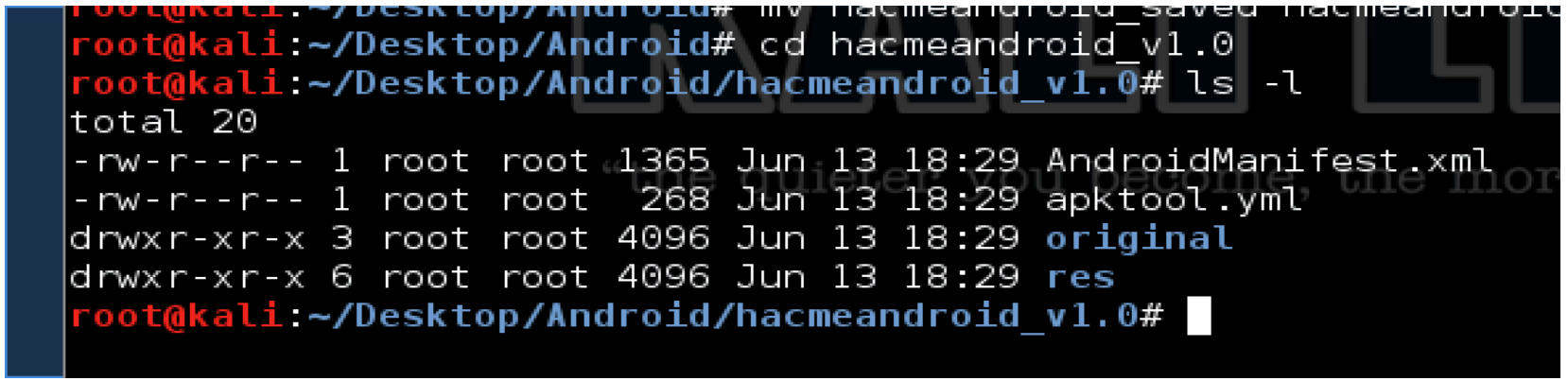
Android application testing

- Handles apk files, unzips them and converts dex files to smali

```
root@kali:~/Desktop/Android# java -jar apktool2.jar d hacmeandroid_v1.0.apk
I: Using Apktool 2.0.0 on hacmeandroid_v1.0.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /root/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...
```

Android application testing

- Expanded directory looks like this



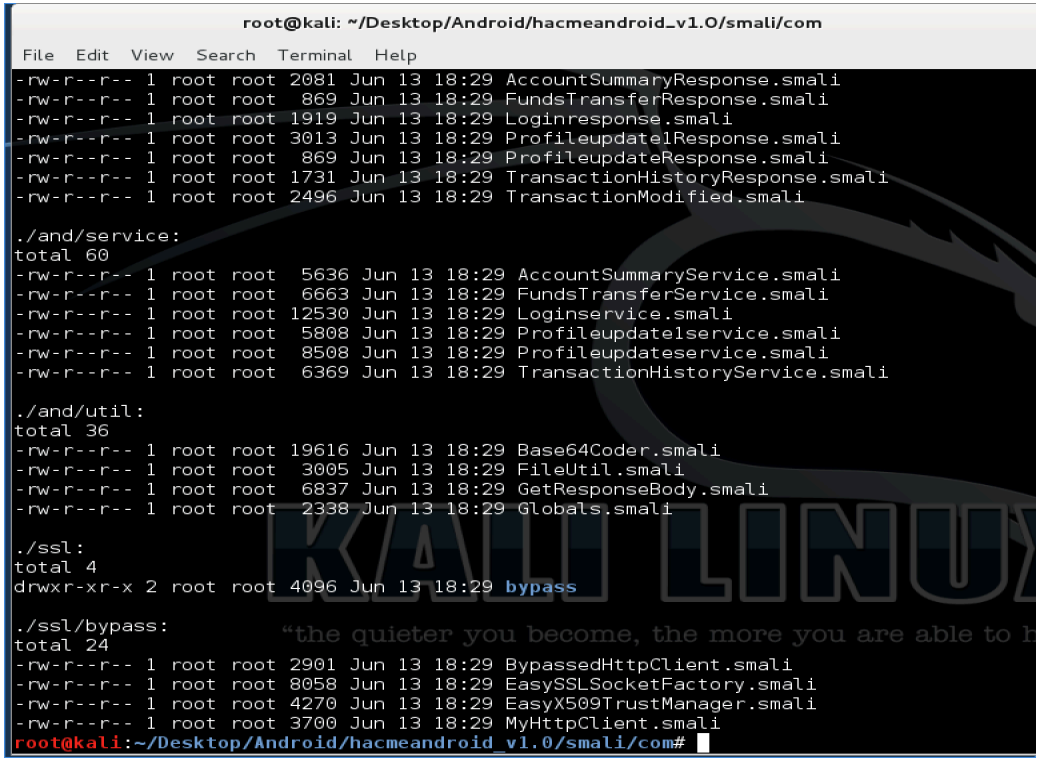
Android application testing

- AndroidManifest.xml

```
drwxr-xr-x 6 root root 4096 Jun 13 18:29 res
root@kali:~/Desktop/Android/hacmeandroid_v1.0# cat AndroidManifest.xml
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.and">
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:label="@string/app_name" android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:label="Funds Transfer" android:name=".FundsTransferActivity"/>
        <activity android:label="Transaction History" android:name=".TransactionHistoryactivity
"/>
        <activity android:label="Profile update" android:name=".profileupdateactivity"/>
        <activity android:label="Account Summary" android:name=".accountsummaryactivity"/>
        <activity android:label="login screen1" android:name=".loginactivity"/>
        <activity android:label="Remember me" android:name=".loginactivity2"/>
        <activity android:label="Home" android:name=".homeActivity"/>
        <activity android:label="Proxy Setting" android:name=".Proxysetting"/>
    </application>
</manifest>
root@kali:~/Desktop/Android/hacmeandroid_v1.0#
```

Android application testing

- Smali files

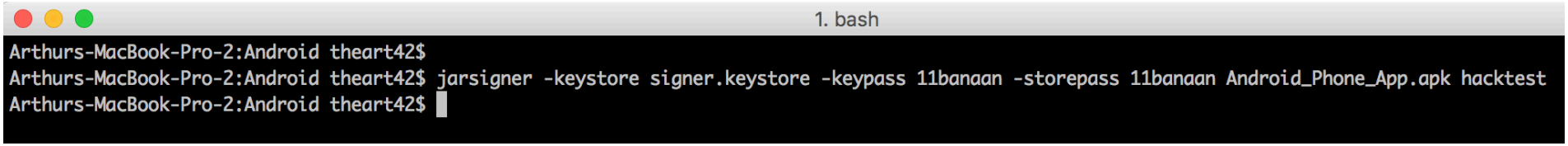


Android application testing

- Smali files can easily be changes, recompiled and repackaged using apktool
  - This creates a new .apk (modified) and this can be installed and used
- 32 bit vs 64 bit problems will be encountered in your Java environment!!

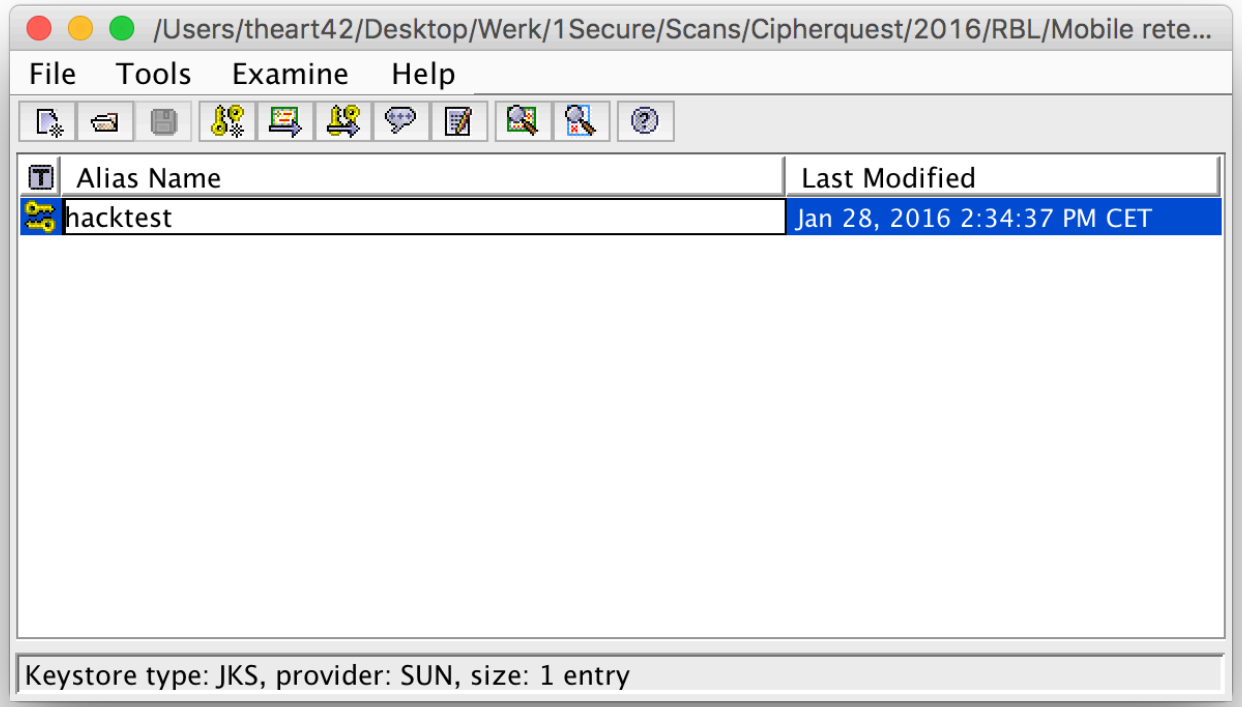
Android application testing

- Application signing
  - Any application developer can create his or her own (self-signed) certificate to sign a package file.
  - Keep this certificate as you need the same certificate for all your applications.



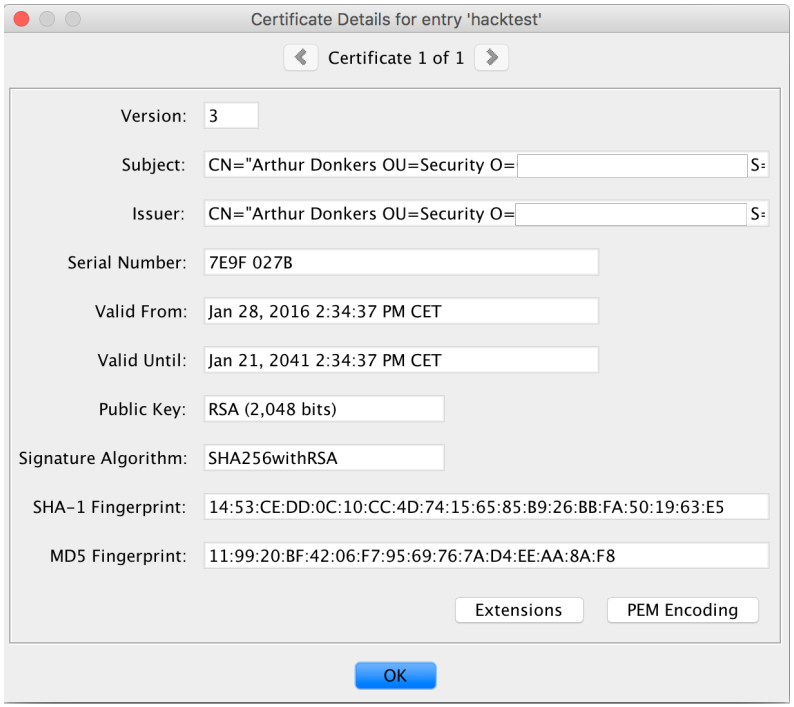
Android application testing

- You can create keystores with different programs (e.g. Portecle)



Android application testing

- You can create keystores with different programs



Android application testing

- Application signing
  - If you want to check the certificate of an installed apk:

```
$ unzip testing.apk
$ cd META-INF
$ openssl pkcs7 -in CERT.RSA -print_certs -inform DER -out out.cer
$ cat out.cer
```

Android application testing

```
.method public constructor <init>()V
    .locals 1

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

    const/4 v0, 0x1

    iput-boolean v0, p0, Lsrc/com/rbl/controller/SplashActivity;->j:Z

    const-string v0, "SHA-512"

    iput-object v0, p0, Lsrc/com/rbl/controller/SplashActivity;->s:Ljava/lang/String;

    sget-object v0, Lcom/infosys/core/a;->p:Ljava/lang/String;

    iput-object v0, p0, Lsrc/com/rbl/controller/SplashActivity;->h:Ljava/lang/String;

    const-string v0, "5b4240613935323763323387f2decaf80ec8dab5e1bbf50ded7fe1060ba280e0511c825c77df7448a0e1d0ce853738fa187d22b37f482b9db8206646ef79e863225c101fa
1314a236d361aa"

    iput-object v0, p0, Lsrc/com/rbl/controller/SplashActivity;->t:Ljava/lang/String;

    return-void
.end method
```

Android application testing

```
    invoke-virtual {v0, v1, v2}, Landroid/content/pm/PackageManager;->getPackageInfo(Ljava/lang/String;I)Landroid/content/pm/PackageInfo;

    move-result-object v0

    iget-object v0, v0, Landroid/content/pm/PackageInfo;->signatures:[Landroid/content/pm/Signature;

    array-length v1, v0

    if-eqz v1, :cond_1

    const/4 v1, 0x0

    aget-object v0, v0, v1

    const-string v1, "SHA-512"

    invoke-static {v1}, Ljava/security/MessageDigest;->getInstance(Ljava/lang/String;)Ljava/security/MessageDigest;

    move-result-object v1

    invoke-virtual {v0}, Landroid/content/pm/Signature;->toByteArray()[B

    move-result-object v0

    invoke-virtual {v1, v0}, Ljava/security/MessageDigest;->update([B)V

    invoke-virtual {v1}, Ljava/security/MessageDigest;->digest()[B

    move-result-object v0

    const/4 v1, 0x0

    invoke-static {v0, v1}, Landroid/util/Base64;->encodeToString([BI)Ljava/lang/String;

    move-result-object v0

    iget-object v1, p0, Lsrc/com/rbl/controller/SplashActivity;->t:Ljava/lang/String;

    invoke-static {v0, v1}, Lsrc/com/rbl/controller/SplashActivity;->c(Ljava/lang/String;Ljava/lang/String;)Z

    move-result v0

#    if-nez v0, :cond_1
```

Android application testing

- Detection of debug flag

```
.method public a(Ljava/lang/String;Ljava/util/List;)Ljava/util/List;
    .locals 11

    iget-object v0, p0, Lsrc/com/rbl/uicontroller/ek;->a:Ljava/lang/String;

    const-string v1, "here0"

    invoke-static {v0, v1}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

    iget-object v0, p0, Lsrc/com/rbl/uicontroller/ek;->a:Ljava/lang/String;

    invoke-static {v0, p1}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I

    sget-object v0, Lcom/infosys/core/a;->ch:Landroid/app/Activity;

    invoke-virtual {v0}, Landroid/app/Activity;->getApplicationInfo()Landroid/content/pm/ApplicationInfo;

    move-result-object v0

    iget v0, v0, Landroid/content/pm/ApplicationInfo;->flags:I

    and-int/lit8 v0, v0, 0x2

#    if-eqz v0, :cond_0
    goto :cond_0
```

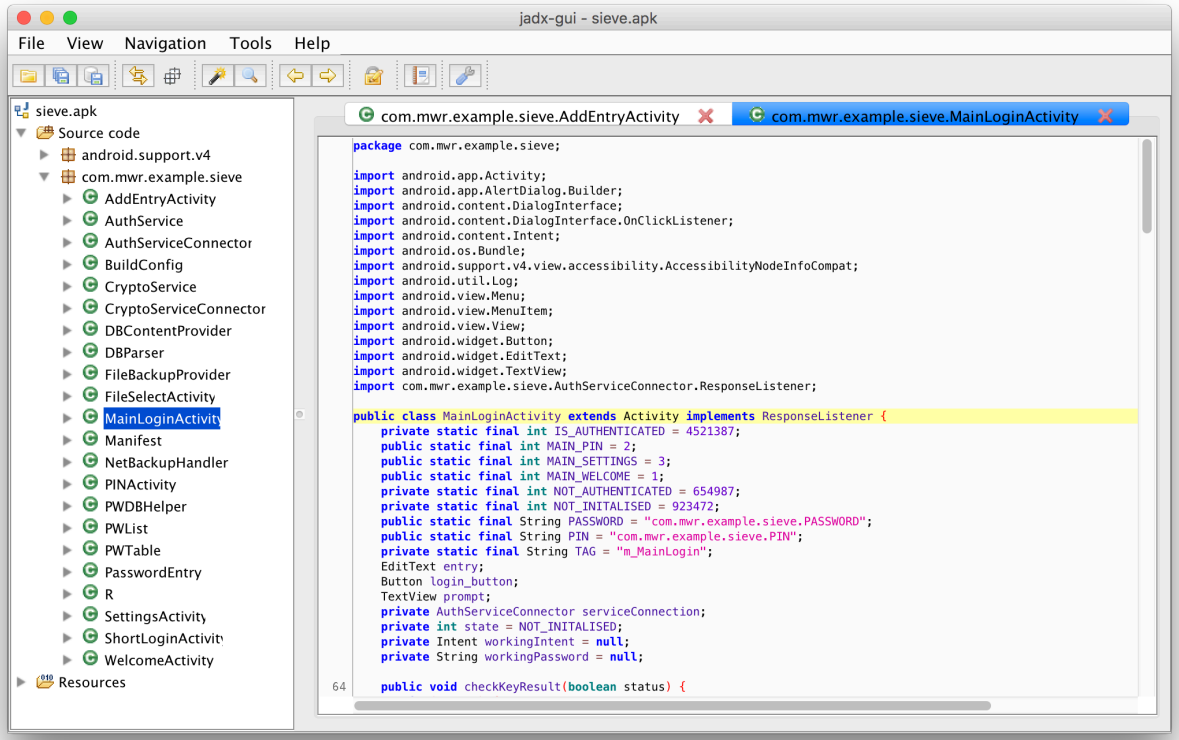
Android application testing

- Another option is dex2jar and jd-gui to decompile to Java code

```
root@kali:~/Desktop/Android# dex2jar classes.dex
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar classes.dex -> classes_dex2jar.jar
Done.
root@kali:~/Desktop/Android# ls -l
total 36156
-rw-r--r-- 1 root root  6344307 Jun 13 18:24 apktool2.jar
-rw-r--r-- 1 root root    62492 Mar 29  2012 classes.dex
-rw-r--r-- 1 root root    68085 Jun 13 19:00 classes_dex2jar.jar
drwxr-xr-x 6 root root     4096 Jun 13 18:51 hacmeandroid_v1.0
-rw-r--r-- 1 root root   255117 Jun 13 18:50 hacmeandroid_v1.0.apk
drwxr-xr-x 6 root root     4096 Apr  5  2012 HacmeBank - Android v1.0
-rw-r--r-- 1 root root 30271149 Jun 13 18:27 hacmebank_android.zip
drwxr-xr-x 2 root root     4096 Jun 13 18:41 save
root@kali:~/Desktop/Android#
```

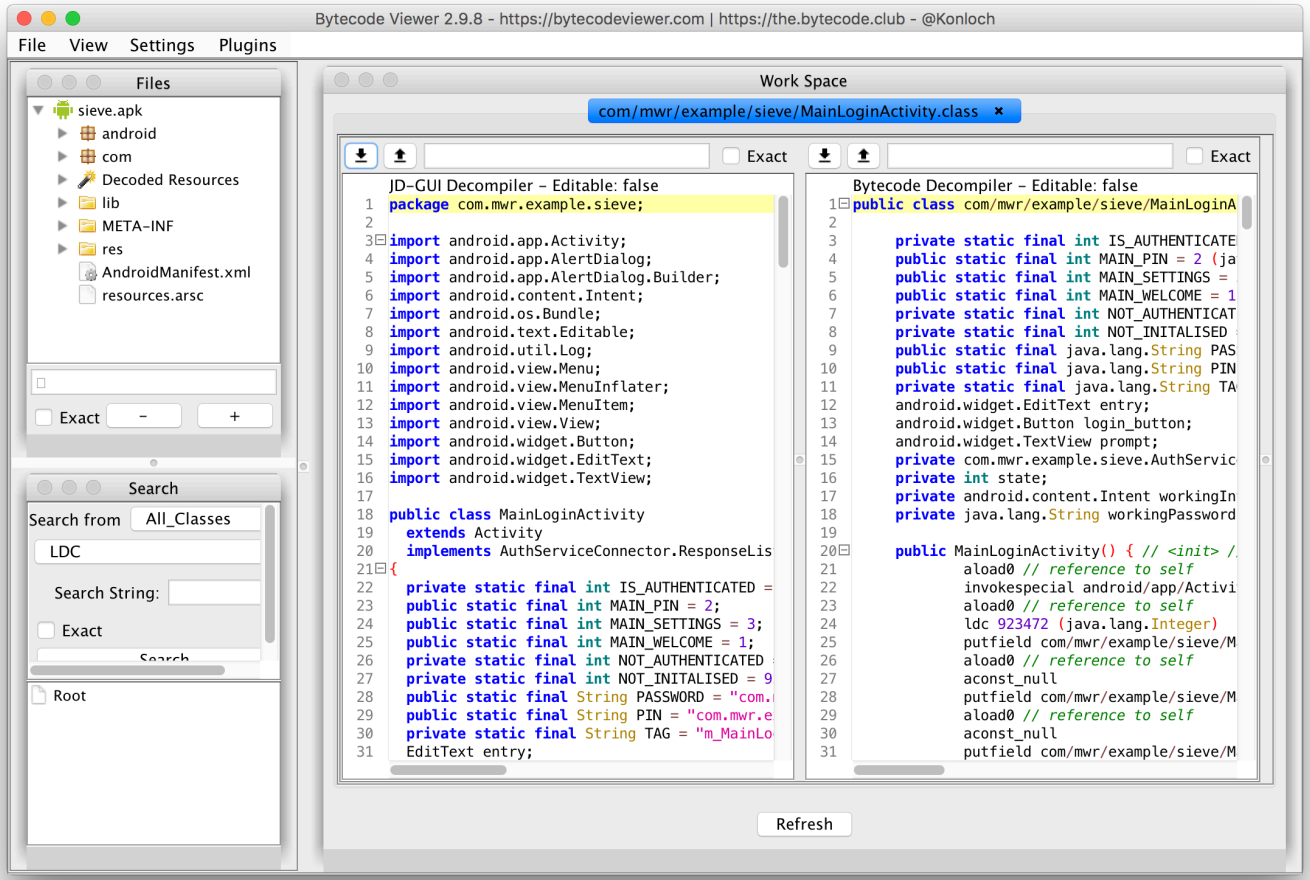
Android application testing

- JADX is an integrated tool environment



Android application testing

- Or ByteCodeViewer



Android application testing

- Runtime rooting detection
  - This almost exclusively is done by looking for the su binary in a number of places on the device
  - Look for the string ”su” to see where the existence of this file is checked
  - Sometimes the developer checks for other files as well
  - You have to disable all of these checks, may take a while

Android application testing

- Once the code has been decompiled, look in the AndroidManifest.xml file for content providers. They will share application data!
- grep –R ‘content://’ AndroidManifest.xml.
- Each content provider is a potential dataleak point.
- adb shell content query --uri <url> will provide information on content.

Android application testing

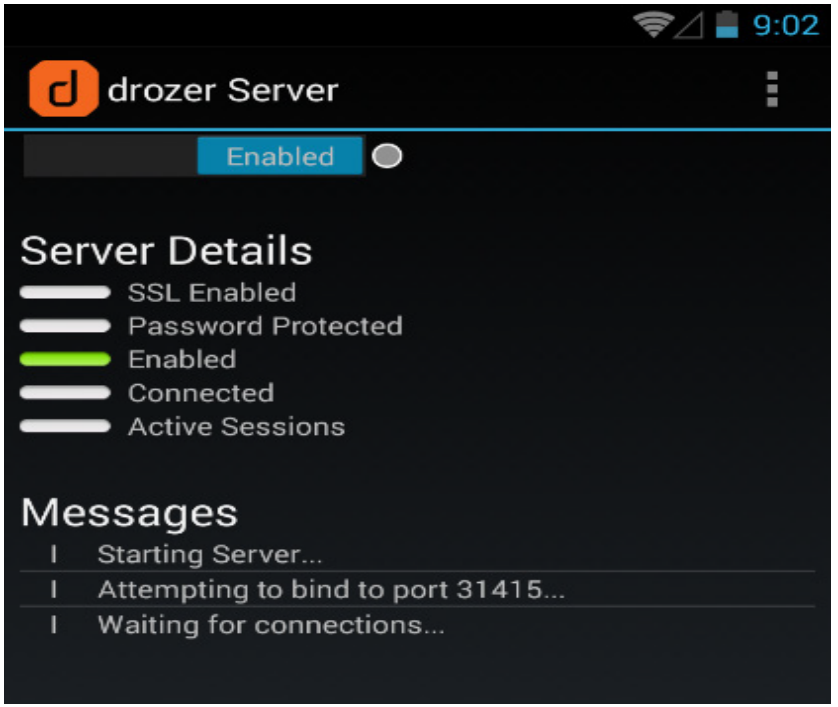
```
$ grep -R 'content://' .
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/media"
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/media_api_id"
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/media_silent_delete"
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/parent_note_of_media"
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/media_with_owner"
./NotePad$Media.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/images_for_note"
./NotePad$MediaNotes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/media_notes"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/topnotes"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_show_deleted"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_silent"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_silent_delete"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_nodeid"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_nodeid_silent"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_nodeid_silent_delete"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/notes_with_images"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/add_media_for_note"
./NotePad$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/add_media_for_note_silent"
./NotePad.smali:.field public static final SCHEME:Ljava/lang/String; = "content://"
./NotePad.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePad/wipe"
./NotePadPending$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePadPending/notes"
./NotePadPending$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePadPending/notes_nodeid"
./NotePadPending$Notes.smali:    const-string v0, "content://com.threebanana.notes.provider.NotePadPending/notes_nodeid_parent"
```

Android application testing

- We can do this manually, or use Drozer:
  - https://www.mwrinfosecurity.com/products/drozer/
  - https://github.com/mwrlabs/drozer
- Drozer is a client server product that automates a lot of the testing.
- Uses an agent on the mobile device:
  - adb install agent.apk

Android application testing

- Use adb to forward port 31415:
  - adb forward tcp:31415 tcp:31415
- Start application/agent on device
- Start drozer on system:
  - drozer console connect



Android application testing

- Find content URI's

```
dz> run app.provider.finduri com.threebanana.notes
Scanning com.threebanana.notes…
content://com.threebanana.notes.provider.NotePad/notes
content://com.threebanana.notes.provider.NotePadPending/notes/
content://com.threebanana.notes.provider.NotePad/media
content://com.threebanana.notes.provider.NotePad/topnotes/
content://com.threebanana.notes.provider.NotePad/media_with_owner/
content://com.threebanana.notes.provider.NotePad/add_media_for_note
content://com.threebanana.notes.provider.NotePad/notes_show_deleted
content://com.threebanana.notes.provider.NotePad/notes_with_images/
```

Android application testing

- Query content URI's

```
dz> run app.provider.query content://com.threebanana.notes.provider.NotePad/notes --vertical
          reminder  0
         hashcodes
       child_count  0
          altitude  0
          sharekey  0
        source_url  null
            nodeid  -1
         timestamp  1386319989407
               _id  1
           created  1386319989407
         longitude  0
     parent_nodeid  -1
    api_pending_op  1
         parent_id  -1
 accuracy_altitude  0
         publicURL
              text  This is a secret note
```

Android application testing

- Other application issues:
  - Insecure file storage (wrong permissions)
  - File inclusion attacks
  - Client side injection attacks

- See OWASP top 10 mobile security

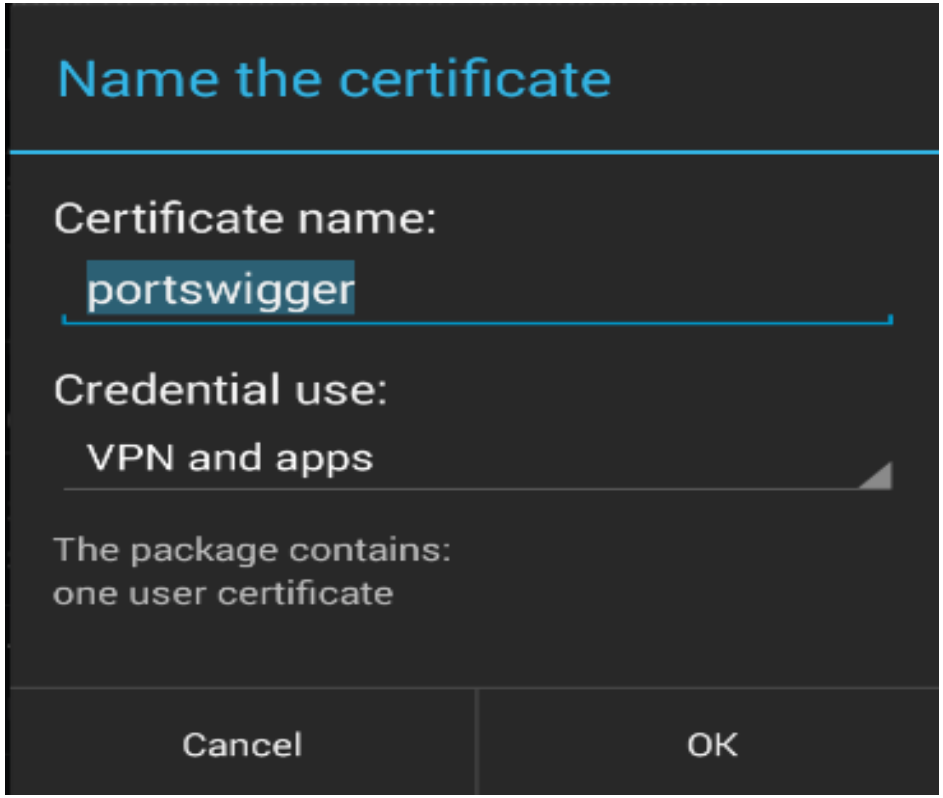
Android application testing

- Traffic interception
  - Passive, using tcpdump-arm
  - http://omappedia.org/wiki/USB_Sniffing_with_tcpdump

Android application testing

- Traffic interception
  - Active, using a malicious AP, Linux interception etc…
  - Intercepting HTTPS traffic requires installation of proxy CA.
  - adb push portswiggerca.crt /mnt/sdcard/portswiggerca.crt.
  - On device go to Settings -> Personal -> Security and install certificate from sdcard.

Android application testing



Android application testing

- Alternative ways to install certificates:
  - Pull the cacerts.bks file from /system/etc/security
  - Add certificate (using Bouncy Castle for instance)
  - Remount /system partition read/write
  - Restore cacerts.bks file

Android application testing

- What about certificate pinning on Android?
  - The most common way involves a number of steps:
  - Read the server certificate from a file
  - Store it in a keystore object in you app
  - Initialize a TrustManager with this keystore
  - With this TrustManager you create an SSLContext and SSLSocketFactory
  - Use this SSLSocketFactory to make the connection to your server
  - The TrustManager verifies the server certificate against the keystore

Android application testing

```
InputStream in = resources.openRawResource(certificateRawResource);

keyStore = KeyStore.getInstance("BKS");
keyStore.load(resourceStream, password);
```

Android application testing

```
HttpParams httpParams = new BasicHttpParams();

SchemeRegistry schemeRegistry = new SchemeRegistry();
schemeRegistry.register(new Scheme("https", new SSLSocketFactory(keyStore), 443));

ThreadSafeClientConnManager clientMan = new ThreadSafeClientConnManager(httpParams, schemeRegistry);

httpClient = new DefaultHttpClient(clientMan, httpParams);
```

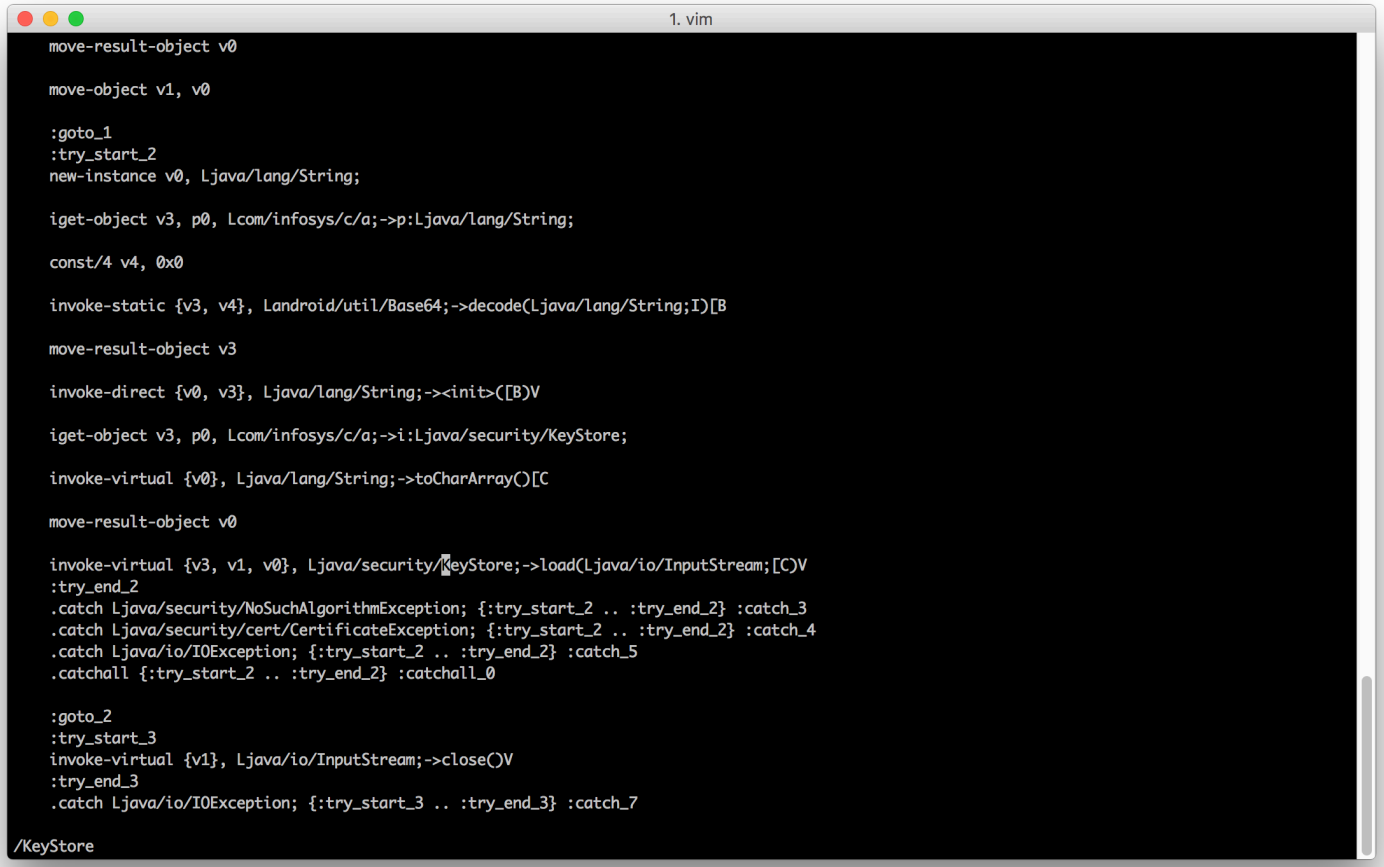
Android application testing

- How to bypass?
  - Sometimes the code itself verifies the certificates, this check can be disabled in the smali file(s)
  - Often you need to replace the proper BKS keystore with one that contains your man-in-the-middle certificate
  - Challenge is to find the password for opening/modifying the keystore
    - Sometimes this password is stored in plain text in the code (DUMB!)
    - More often this password is encrypted/obfuscated
    - Use the Java Debugger to obtain the password

Android application testing

- To get the password
  - Look through the code to find the initialization of the KeyStore object
  - Look for the KeyStore.load method and determine the parameters
  - This indicates which parameter contains the password
  - Make a note of the module, method and the parameter for later

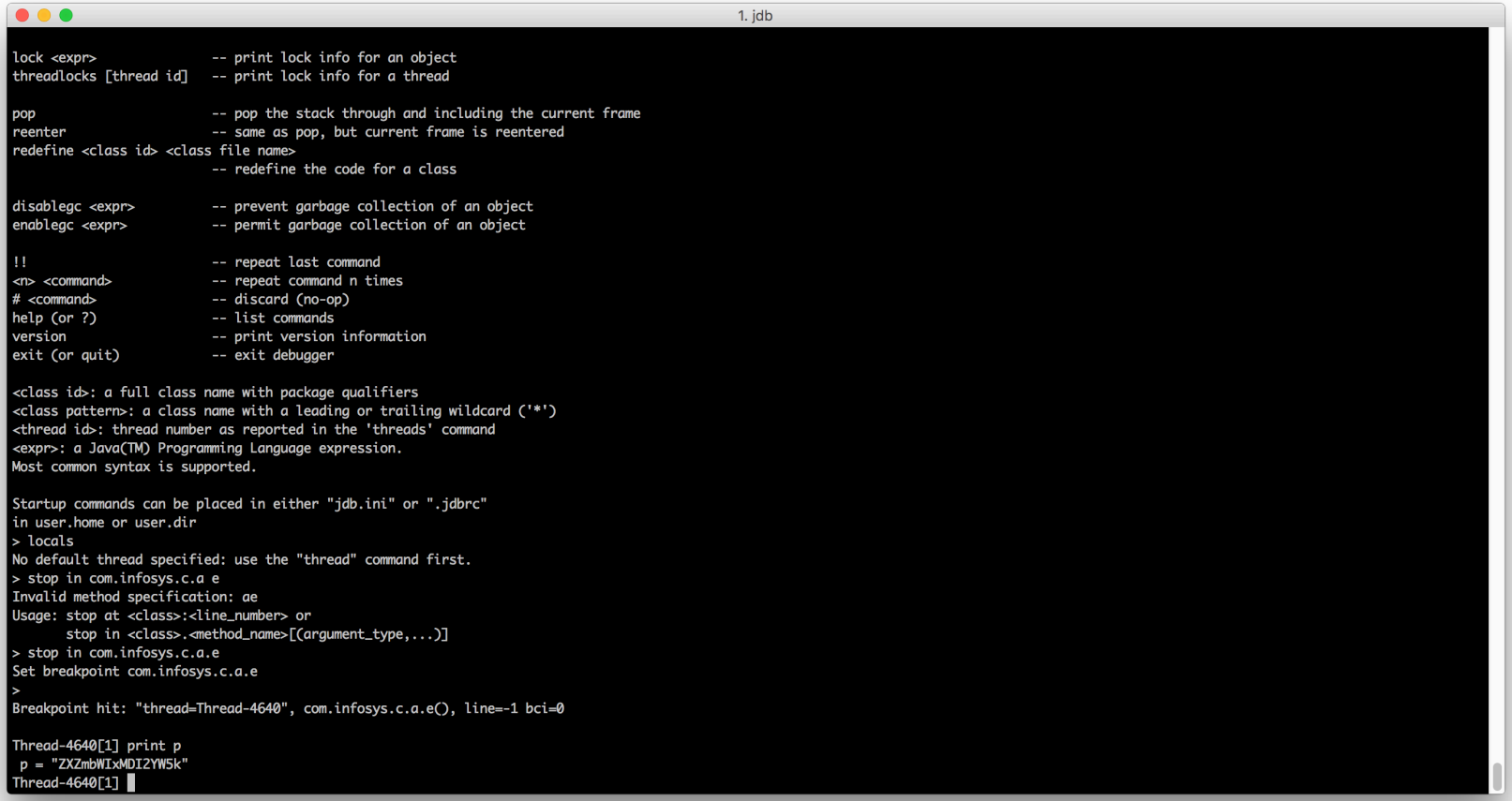
Android application testing



Android application testing

- To get the password
  - Modify the AndroidManifest.XML file to enable debugging
  - Rebuild the APK, sign and install
  - Run the application
  - Use adb jdwp to find the portnumber where the app is listening for JDB
  - Forward this port: adb forward tcp:8000 tcp:<jdwp port>
  - JDB –attach localhost:8000
  - Set a breakpoint on the method in the module
  - Once it breaks, print the contents of the variable

Android application testing

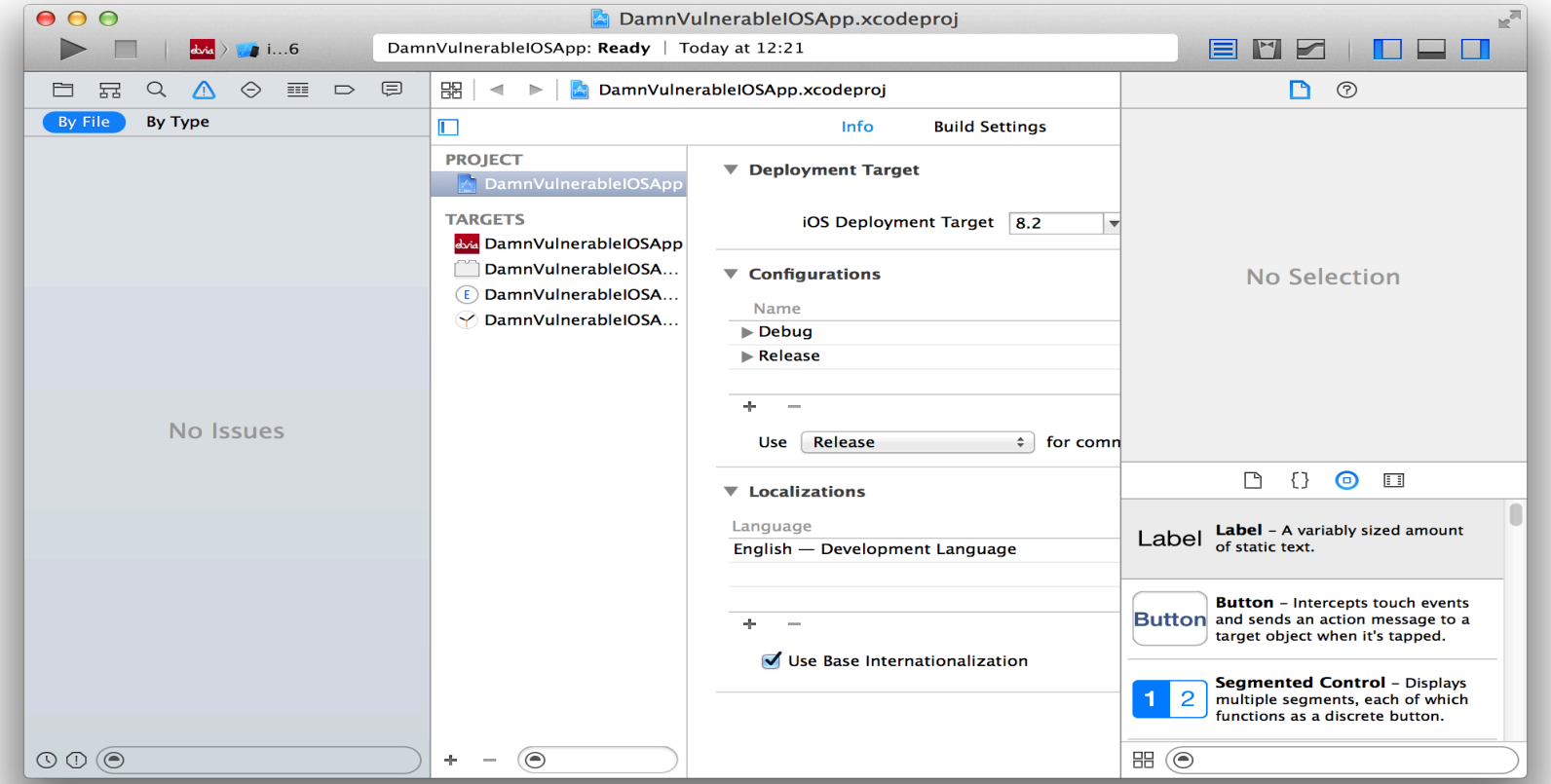


iOS application testing

iOS application testing

- Background information
- iOS has its roots in OS X, based on the Darwin OS, sources are available!
- Apps are written in Objective-C or Swift
- Xcode is the development environment, runs on OS X and provides a simulator
- http://damnvulnerableiosapp.com/ Damn Vulnerable iOS application

iOS application testing



iOS application testing

- Objective-C is object oriented enviroment
- Applications are built on a collection of objects
- Objects exchange messages (as their way of method calling)
- Libraries are often called frameworks

iOS application testing

- As objects exchange messages, all objects are active once an application has started

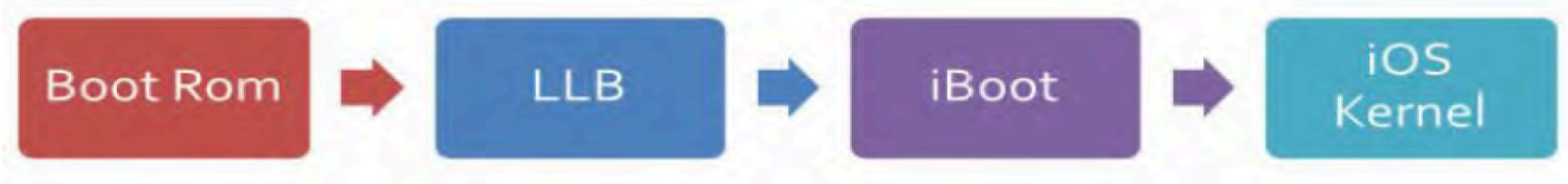
- If you can send a message to an object, you can bypass the regular flow

```
@implementation Classname
@synthesize blorg; // generates set/get methods
@synthesize gurgle;

Instance *myInstance = [[Instance alloc] init];

[myInstance setGurgle:@"foo"]; // infix notation
myInstance.gurgle = @"foo";    // dot notation
```

iOS application testing

- First security control is the secure boot

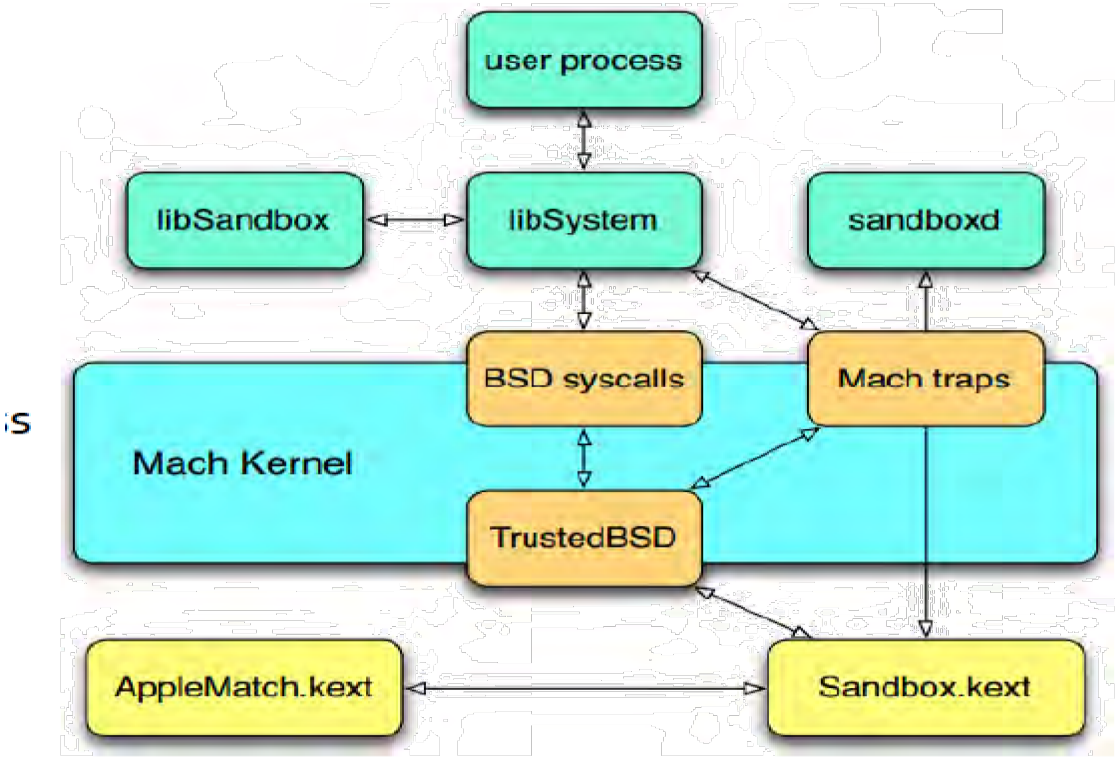
- Each step ensures that the proper software is running:



iOS application testing

- Once system is running, application sandbox (seatbelt) takes over
- Entitlements control access to user information and system-wide resources
- All apps run under 1 user: mobile
- Apps are digitally signed by Apple, an Enterprise or Developer certificate
- Signature is checked before starting an application

iOS application testing



iOS application testing

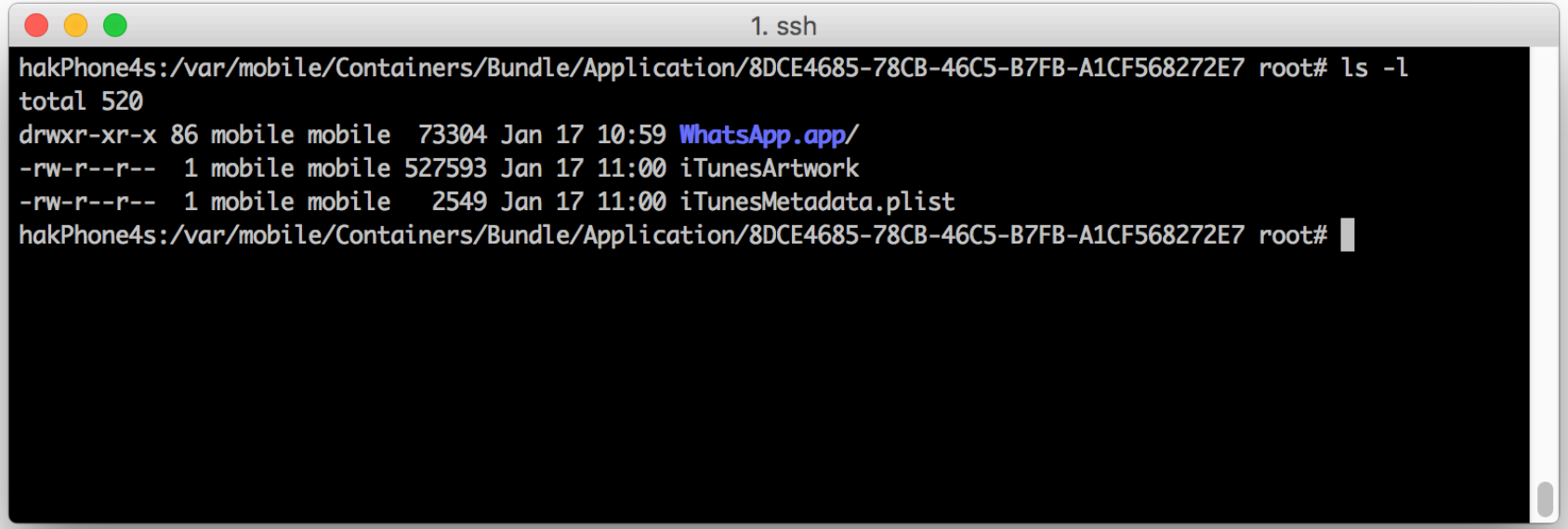
- Device has full disk encryption (FDE) by default, protects against hardware attacks and allows for quick erase
- Each file is encrypted with its own key, to implement access control
- Data and apps are not protected against:
  - Jailbreaking
  - Backup attacks
  - Software vulnerabilities

iOS application testing

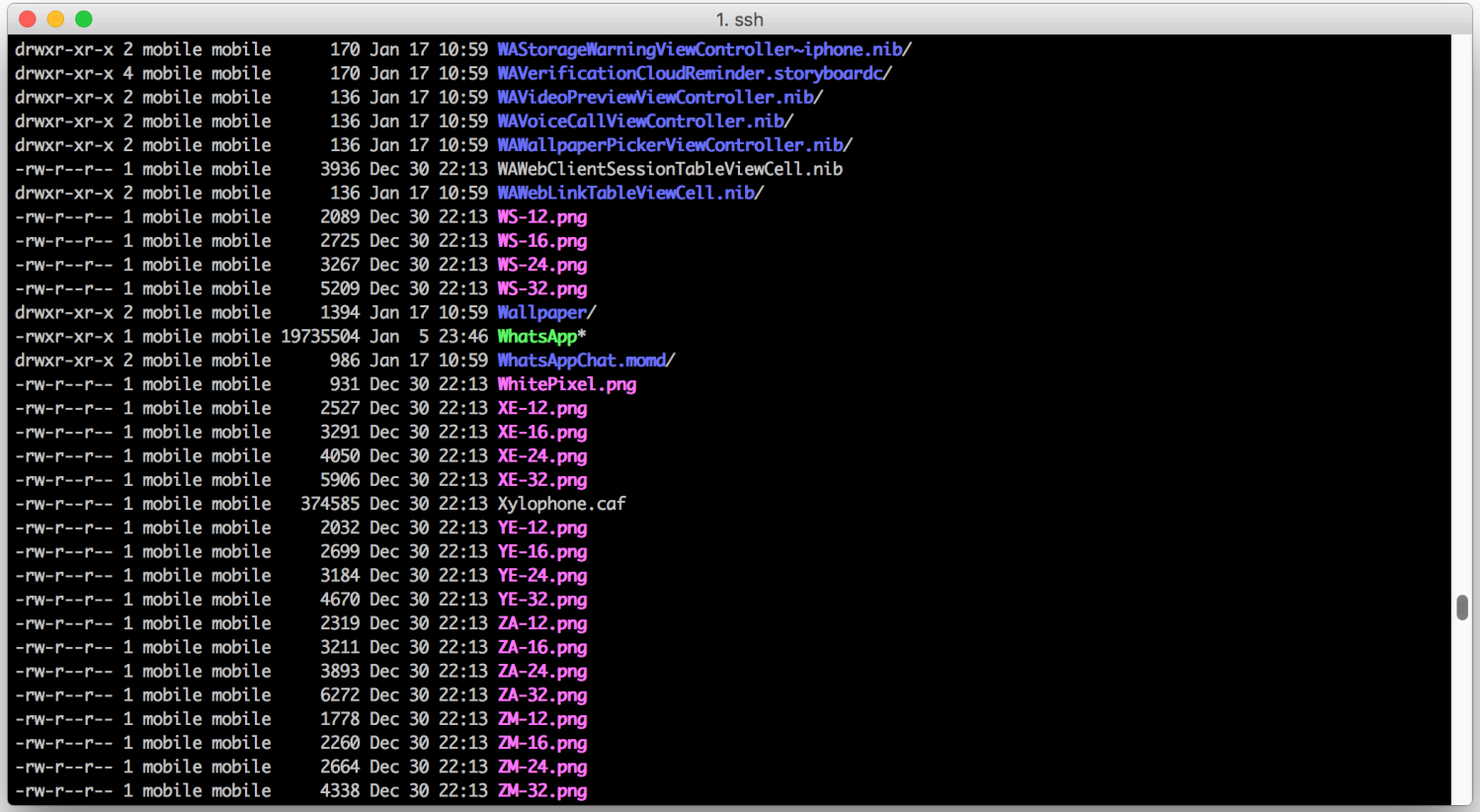
- Each app has a unique UUID and is stored in:
  - /var/mobile/Containers/Bundle (code)
  - /var/mobile/Containers/Data (data)
- Distributed as a (signed) .ipa file (ZIP file)
- Native apps are stored in /Applications

- On older (< 9) iOS:
  - /private/var/mobile/Library/Caches/com.apple.mobile.installation.plist contains list of all installed applications

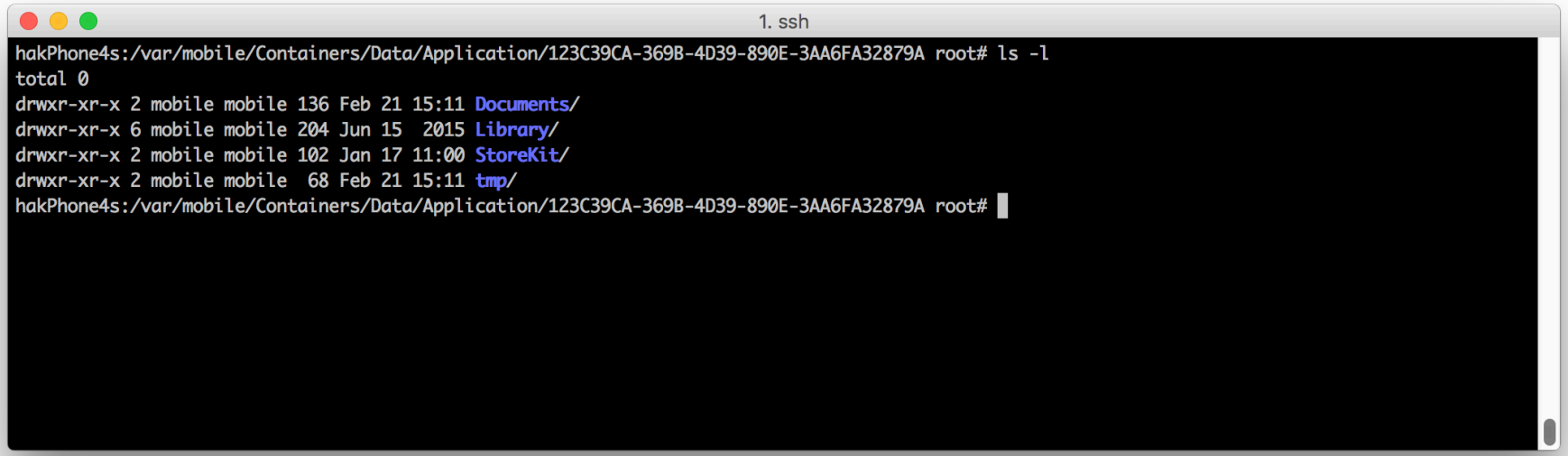
iOS application testing



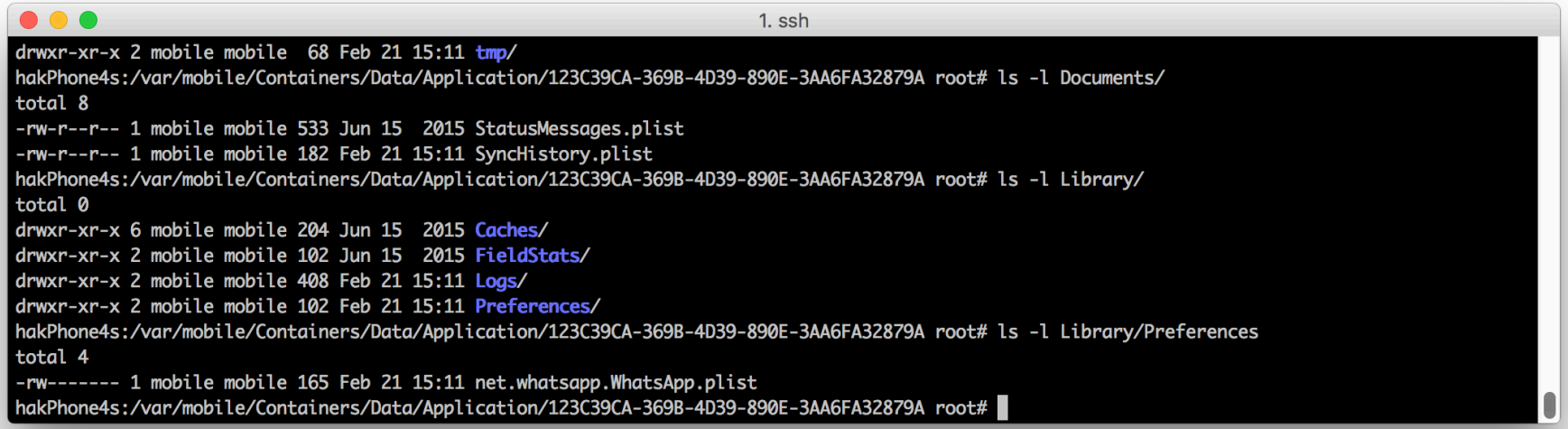
iOS application testing



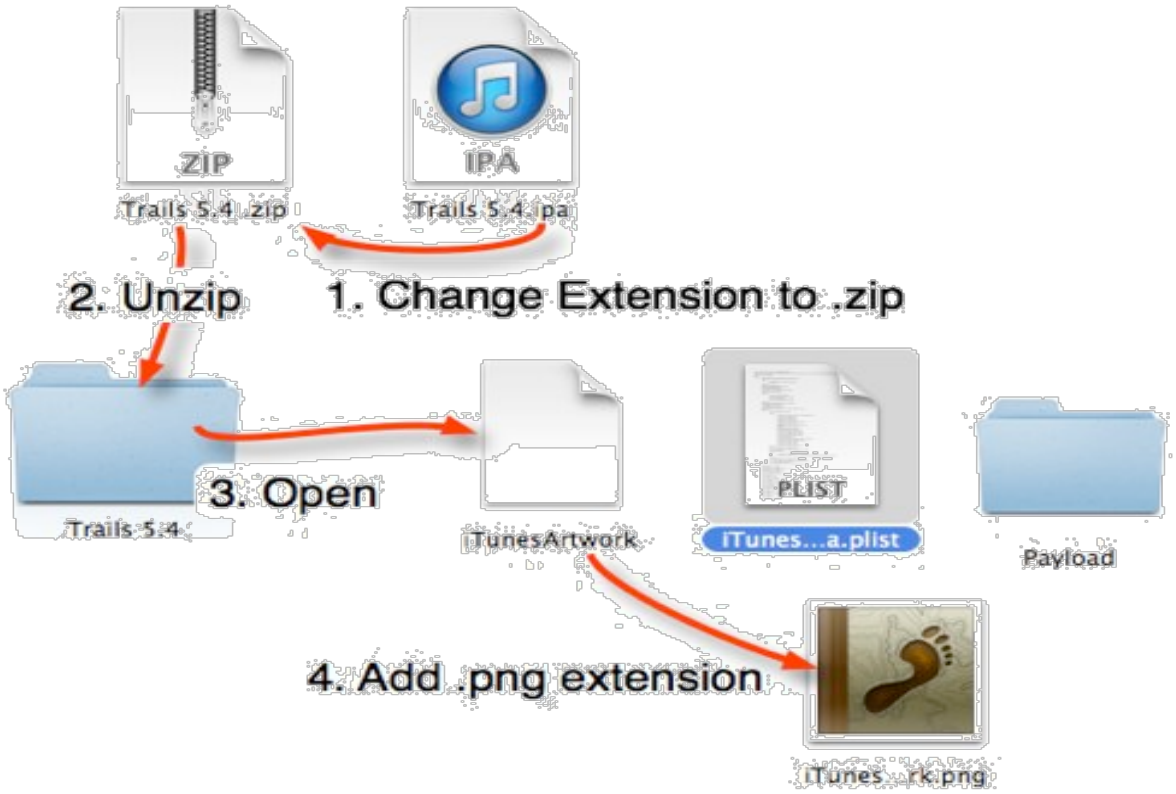
iOS application testing



iOS application testing

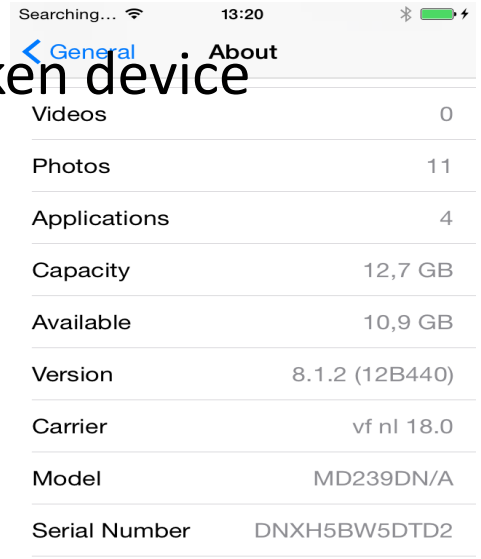


iOS application testing



iOS application testing

- To be able to REALLY test an app, you need a Jailbroken device
- Jailbreaking allows you to:
  - Run any app;
  - Install your own tools;
  - Alternative app stores;
  - Unsigned binaries;
  - Disable sandbox;
  - SSH access to device.
- Jailbreaks become increasingly rare



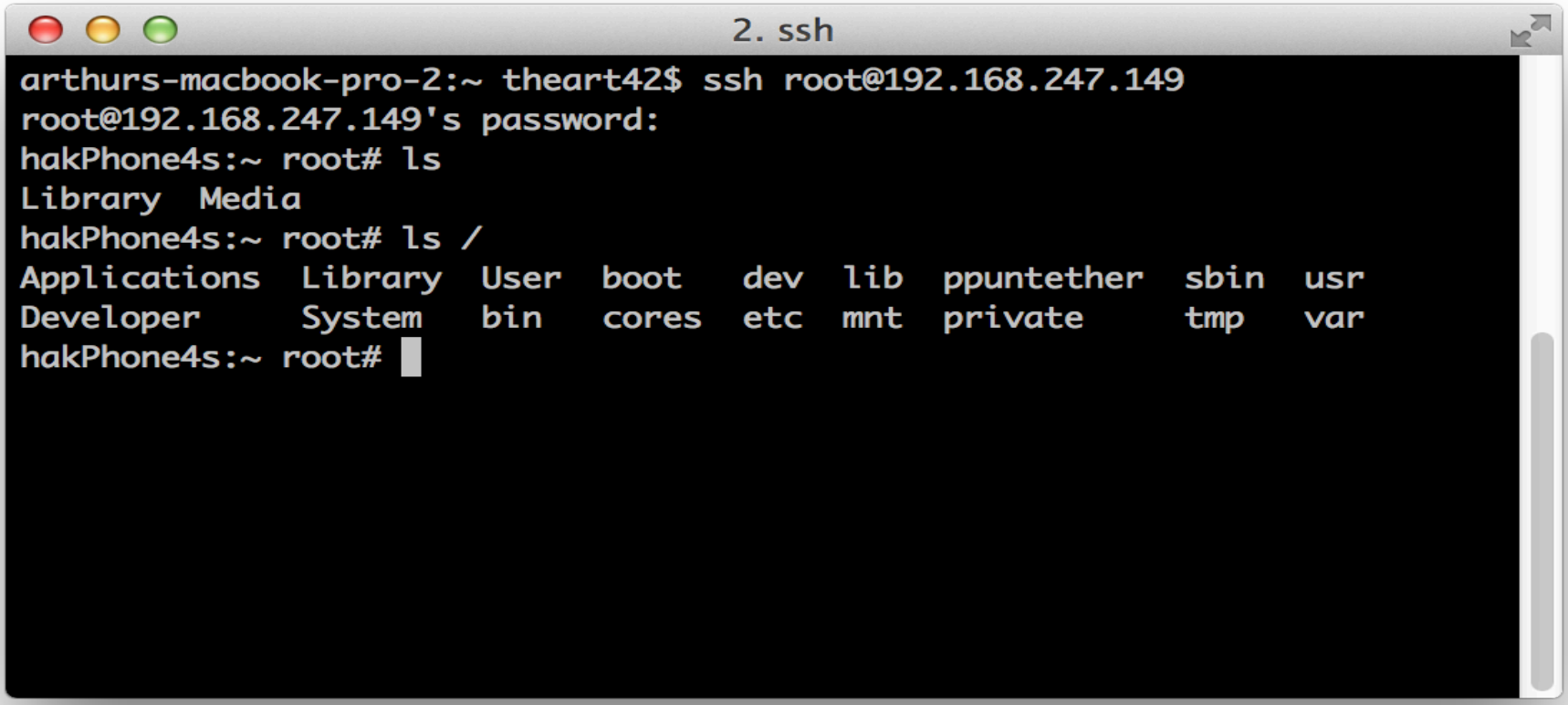
iOS application testing

- After jailbreak, install Cydia app store:
  - Repositories with additional tools
  - Packaged as .deb files (Linux)
  - Install APT packages
  - Install OpenSSH

iOS application testing

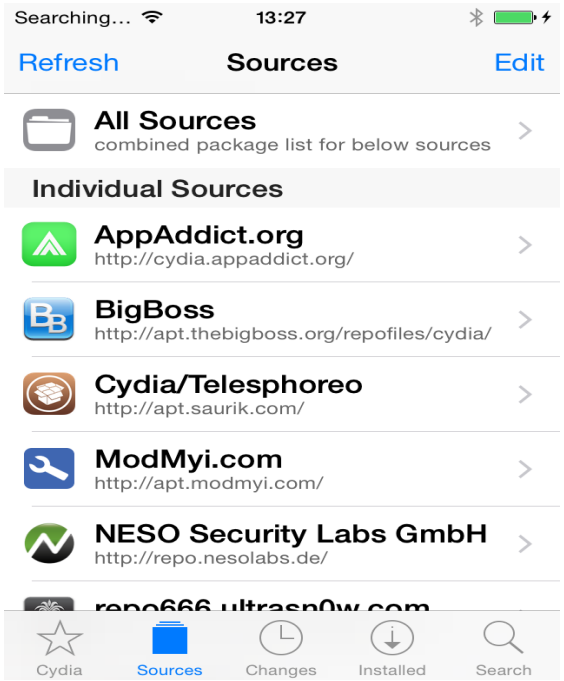
- Jailbreak security issues:
  - Default password for root and mobile user is alpine.
  - Change this after jailbreaking and installing OpenSSH!!

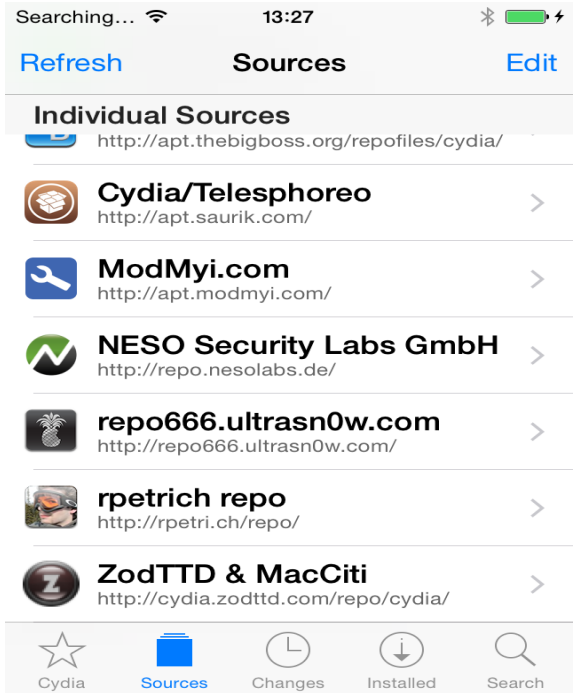
iOS application testing



iOS application testing

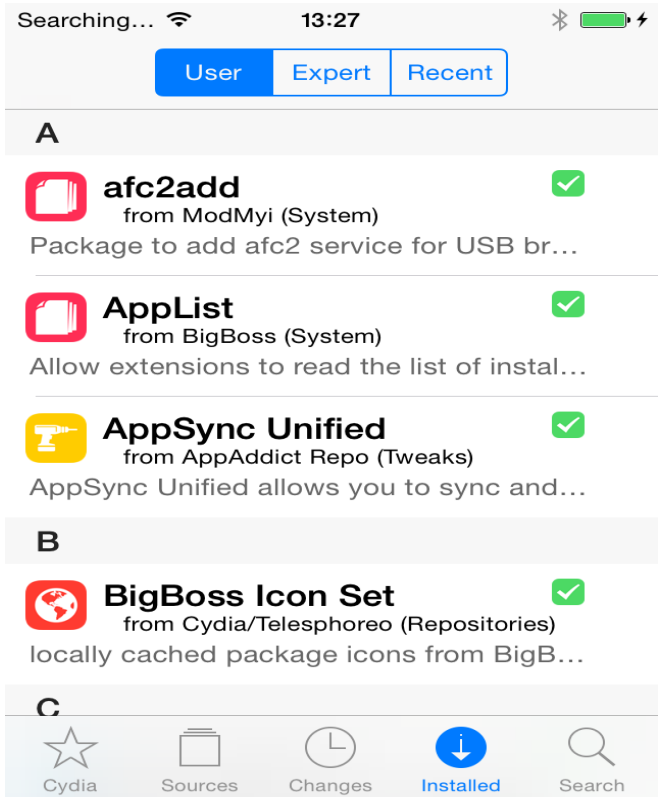
- Cydia repositories





iOS application testing

- Packages



iOS application testing

- Packages install most useful tools in one go:

```
iPhone:~ root# apt-get install adv-cmds com.sull.clutchpatched
curl cycript odcctools developer-cmds dpkg
com.ericasadun.utilities file file-cmds findutils gawk git grep
inetutils com.autopear.installipa ldid less lsof mobilesubstrate
com.saurik.substrate.safemode mobileterminal-applesdk nano netcat
network-cmds python sed shell-cmds sqlite3 syslogd system-cmds
tcpdump top uikittools unrar unzip vim wget whois zip
```

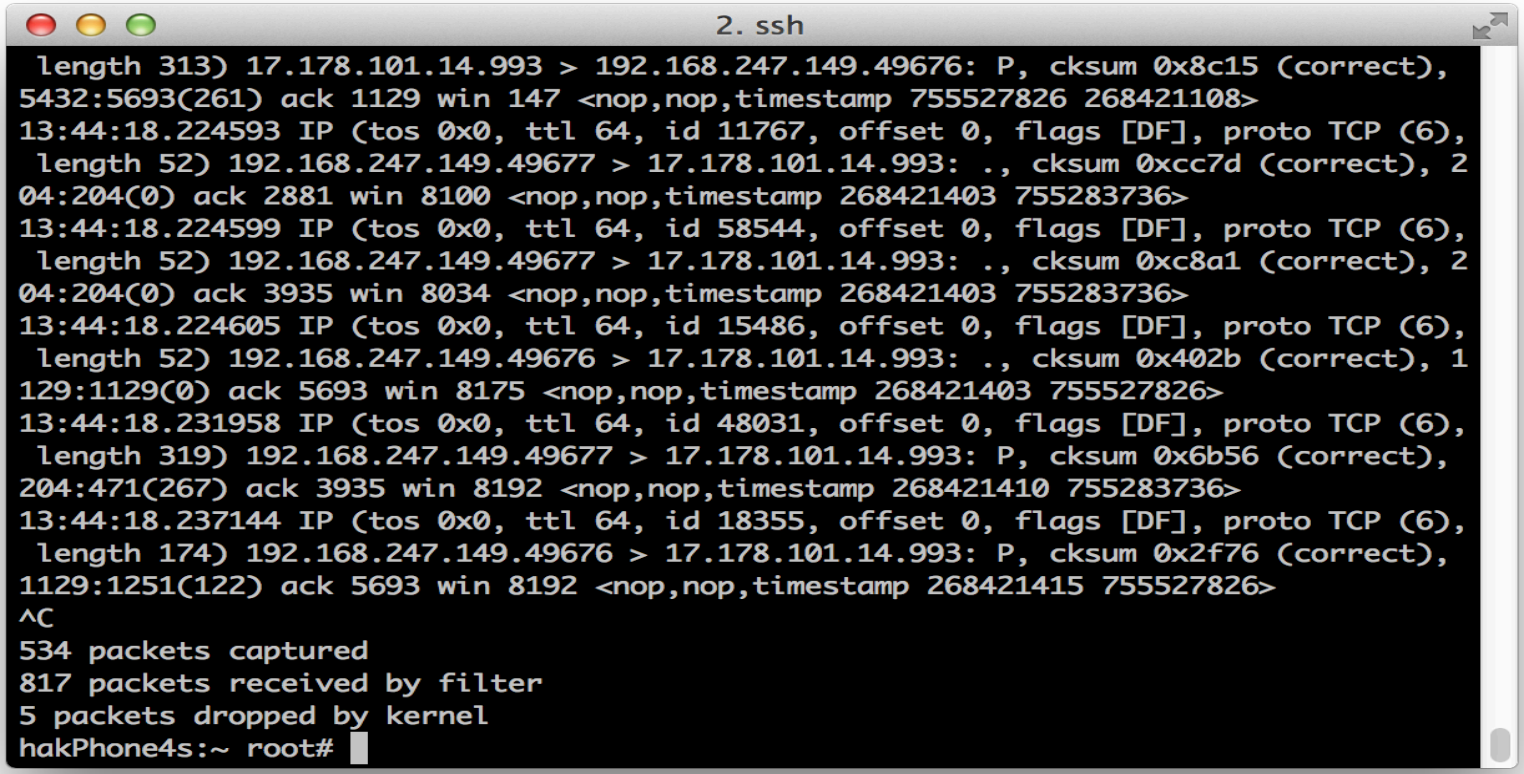
iOS application testing

- In Xcode you can:
  - Disable code signing.
  - Build an application.
  - Make a distribution file (.ipa).
  - Copy the package to the device (scp).
  - Deploy using IPA Installer Console or use iPhone configuration utility.

iOS black box testing

- Traffic analysis:
  - Install tcpdump on device and ‘sniff’ traffic
    - tcpdump –i en0 –n –vv –s 0 –U not port 22

iOS black box testing



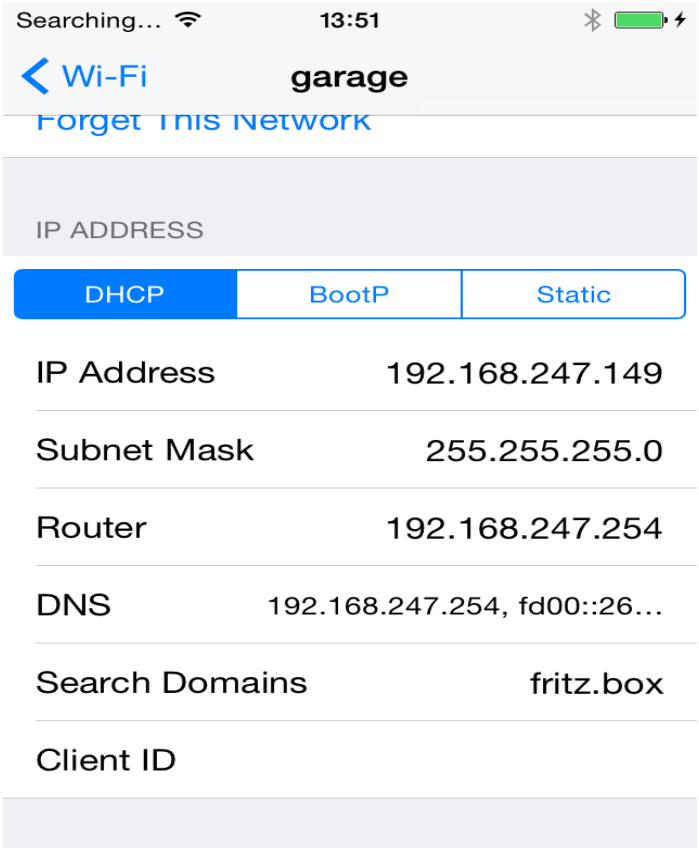
iOS black box testing

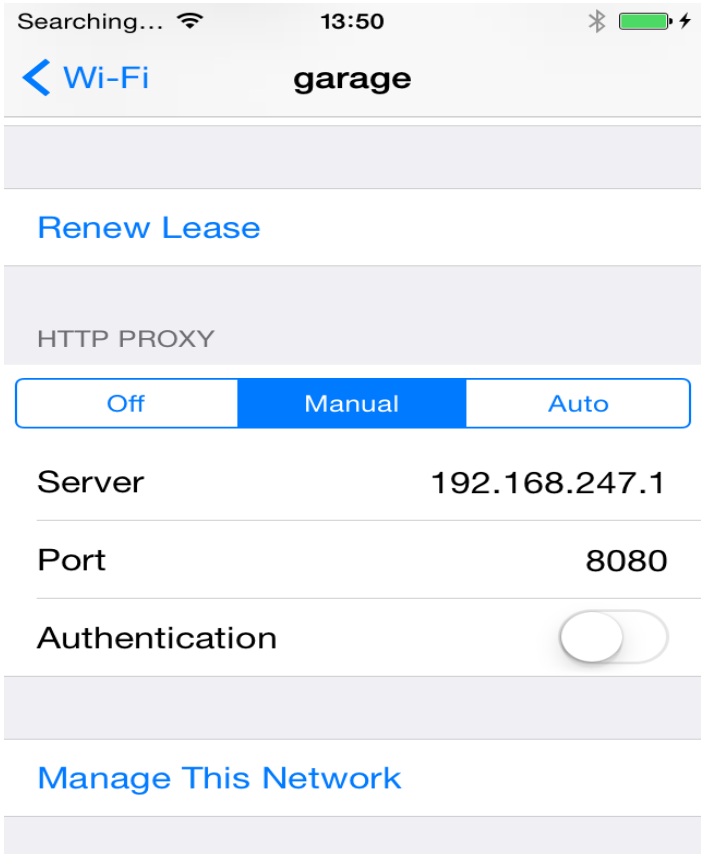
- Or:
  - Build an Access Point with Kali or Pineapple
  - Analyse traffic with Wireshark or tcpdump
  - Don’t forget to disable firewall on your access point
  - And enable ip_forwarding
- Much more powerful and more tools at your disposal (especially if you are using Kali)
- Use a wifi dongle that has monitoring capabilities!

iOS black box testing

- Use proxy software on Kali to intercept Web traffic
  - Burpsuite
  - ZAP
- Set proxy in wifi profile on device,
- or use iPhone Configuration Utility,
- or use transparent redirection on Kali

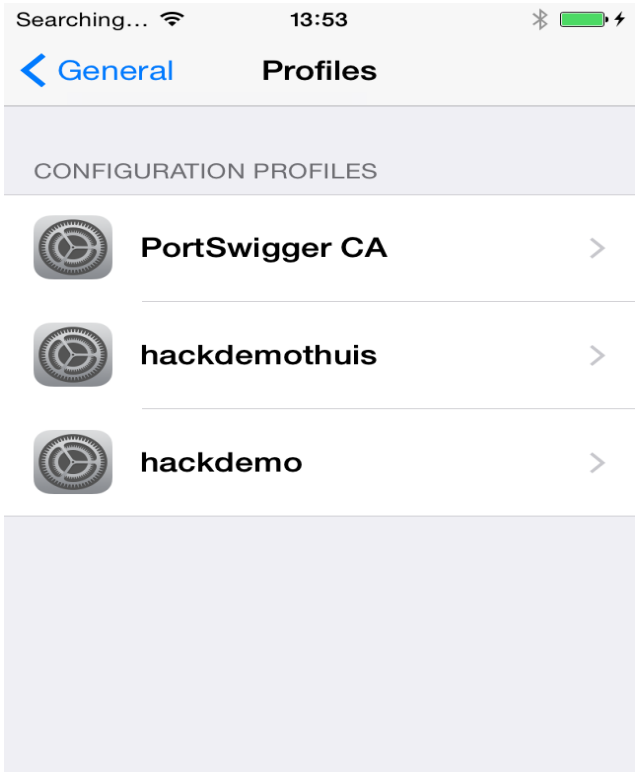
iOS black box testing



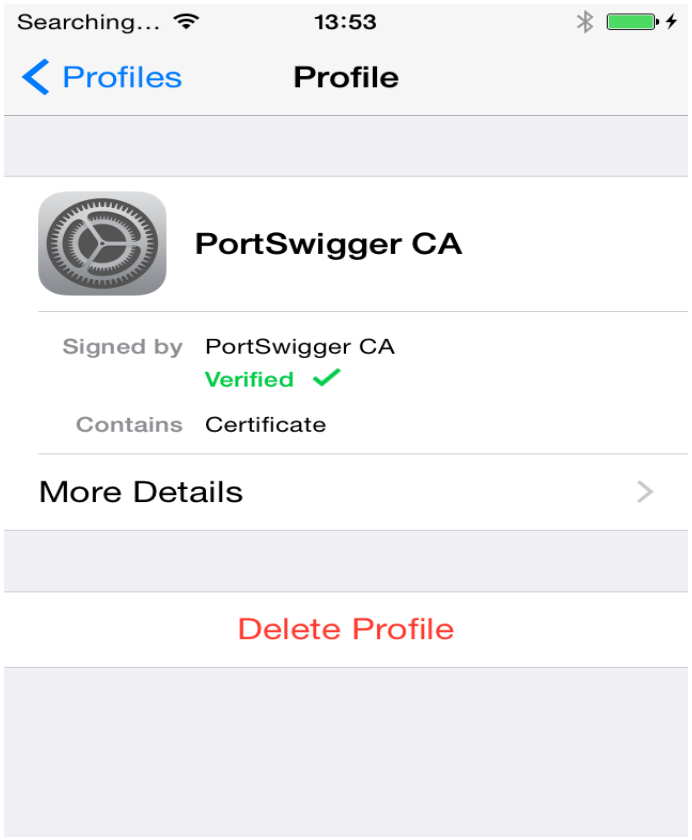


iOS black box testing

- Install the burpsuite CA:
  - Send .crt file as mail attachment and install
  - Use the iPhone Configuration Utility
- Settings -> General -> Profiles



iOS black box testing



iOS black box testing

- If an application does not honor the HTTP proxy settings, what can you do?
  - Intercept DNS traffic and return the IP address of your test machine as the target IP. Don’t forget to forward the traffic to the actual destination
  - Perform transparent redirection on your Linux machine using iptables.
- In both cases, your proxy must support invisible/transparent proxying

iOS black box testing

- An example:

```
iptables -t nat -F PREROUTING
iptables -t nat -F POSTROUTING
iptables -t nat -A PREROUTING -i eth2 -s 172.16.42.0/24 -d 1.2.3.4 -p tcp --dport 443 -j DNAT --to
172.16.42.42:443
iptables -t nat -A POSTROUTING -o eth0 -d 1.2.3.4 -j SNAT --to 172.16.98.222
iptables -t nat -A POSTROUTING -s 172.16.42.0/24 -d 0.0.0.0/0 -j MASQUERADE
echo 1 > /proc/sys/net/ipv4/ip_forward
```

iOS black box testing

- For other traffic (non-HTTP) you can use Mallory as a generic proxy or write your own scripts…. Both are complicated.
- Use iptables for redirection.
- Luckily most (> 90%) of client / server traffic is web based:
  - Screen scraping
  - Web services

iOS black box testing

- What about certificate pinning?
  - Hack the application (replace certificate files with your own).
  - Use ios-ssl-killswitch or trustme.

- They overrule or patch the SSL framework to always accept Certificate.
- If done properly, pinning is hard to bypass (actual verifications in application code).

iOS black box testing

- In client/server communication, check for:
  - Proper certificate handling (does the app accept self-signed certificates)
  - Is sensitive information cached somewhere?
  - Where are the (persistent) cookies?
  - Any Javascript in use?
  - XML injection
  - Be careful with Unique Device ID (can be spoofed)

iOS black box testing

- In client / server communication, to protect your customers, check for:
  - Access to address book, photos
  - Geolocation
  - Access to phone etc…

iOS black box testing

- Local data storage:
  - User preferences
  - Media files
  - Attachments
  - Config settings

iOS black box testing

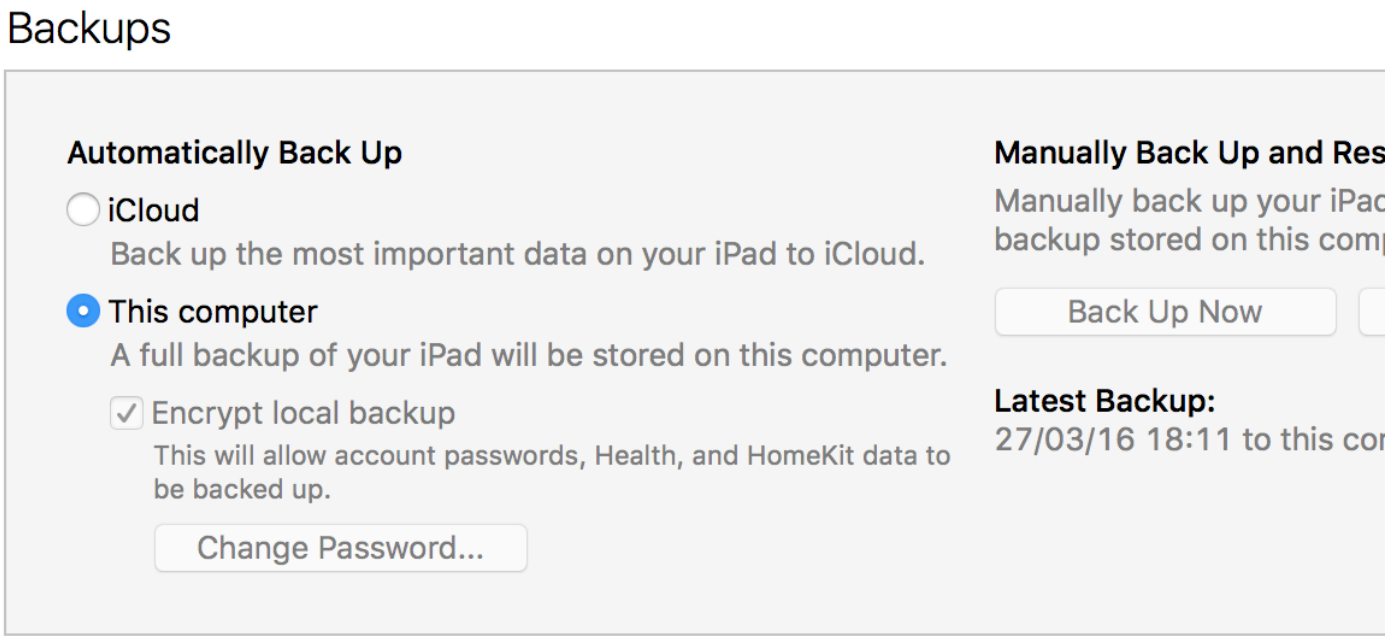
- Local data stored in:
  - plain files
  - .plist files
  - SQLite database
- You can often find credentials, urls, password, tokens, cookies etc.

iOS black box testing

- You can use the SSH access to browse the file system
- Or use iExplorer to browse device
- Don’t forget that these files are stored in your backup as well, so force encrypted backups
- Use the data protection classes to properly protect data on the device

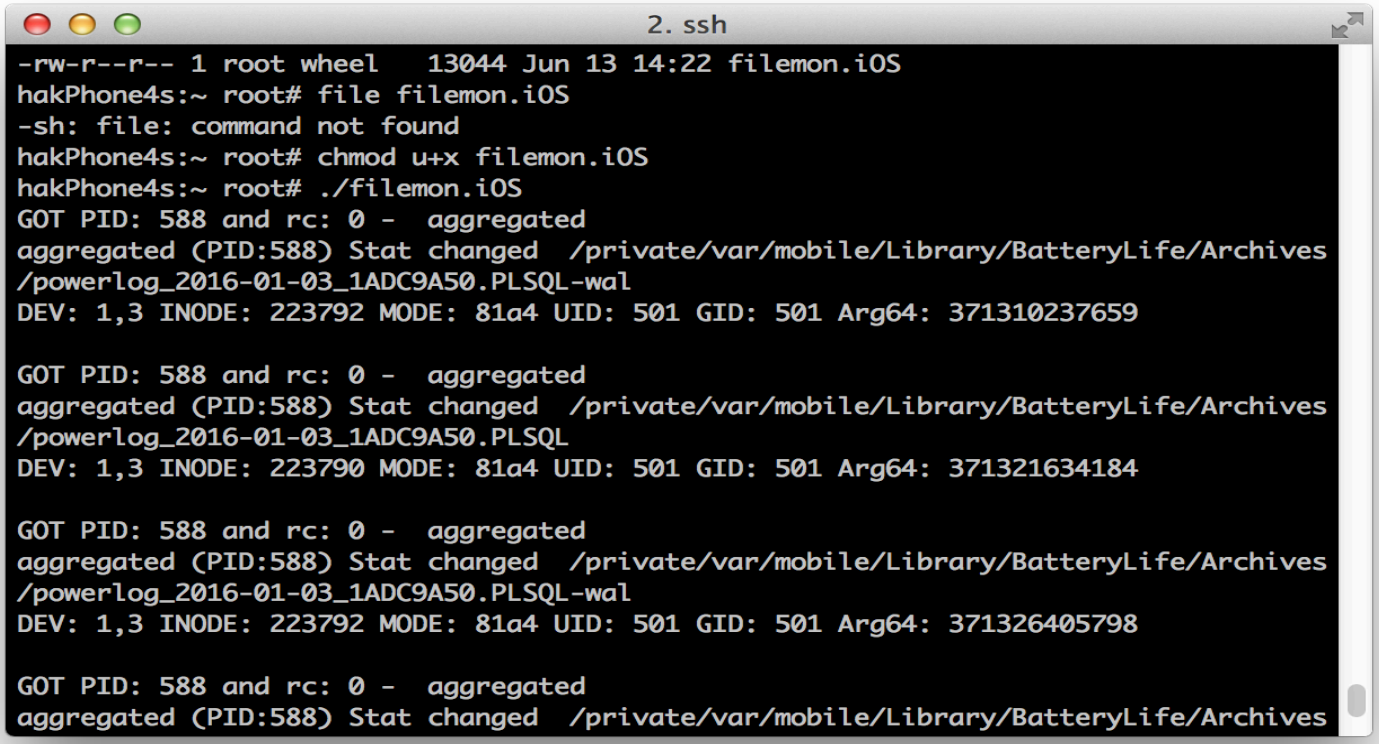
```
[[NSFileManager defaultManager] createFileAtPath:[self filePath]
  contents:[@"file content" dataUsingEncoding:NSUTF8StringEncoding]
  attributes:[NSDictionary dictionaryWithObject:NSFileProtectionComplete
forKey:NSFileProtectionKey]];
```

iOS black box testing



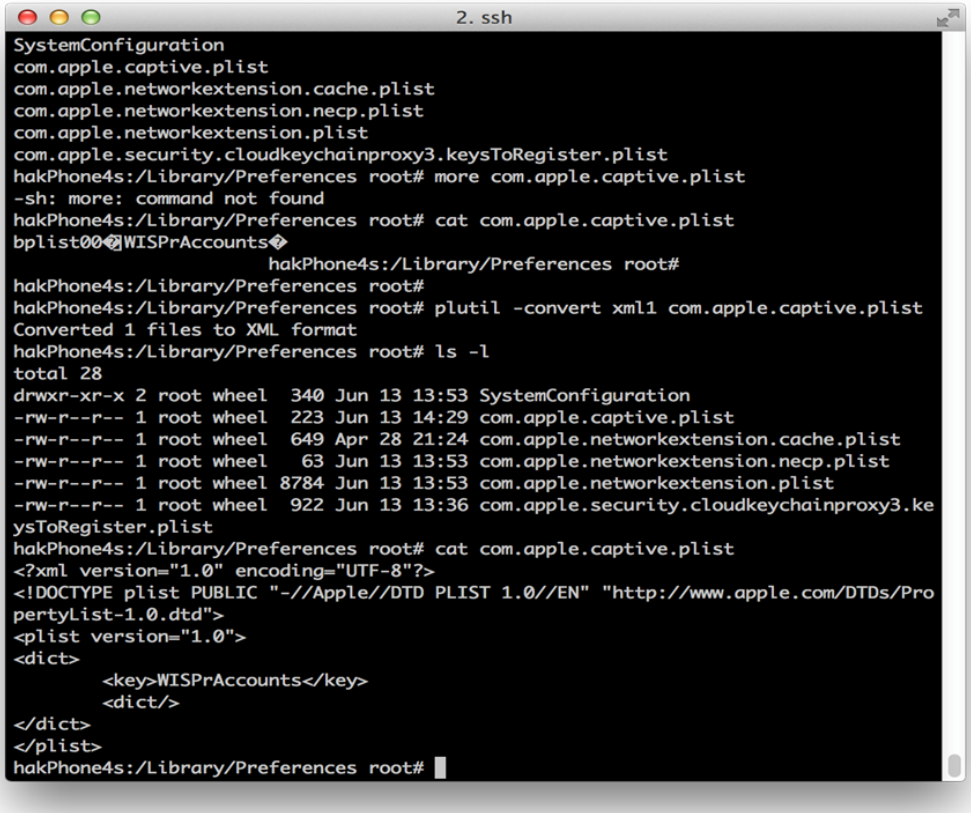
iOS black box testing

- You can use filemon.iOS to monitor file access in realtime



iOS black box testing

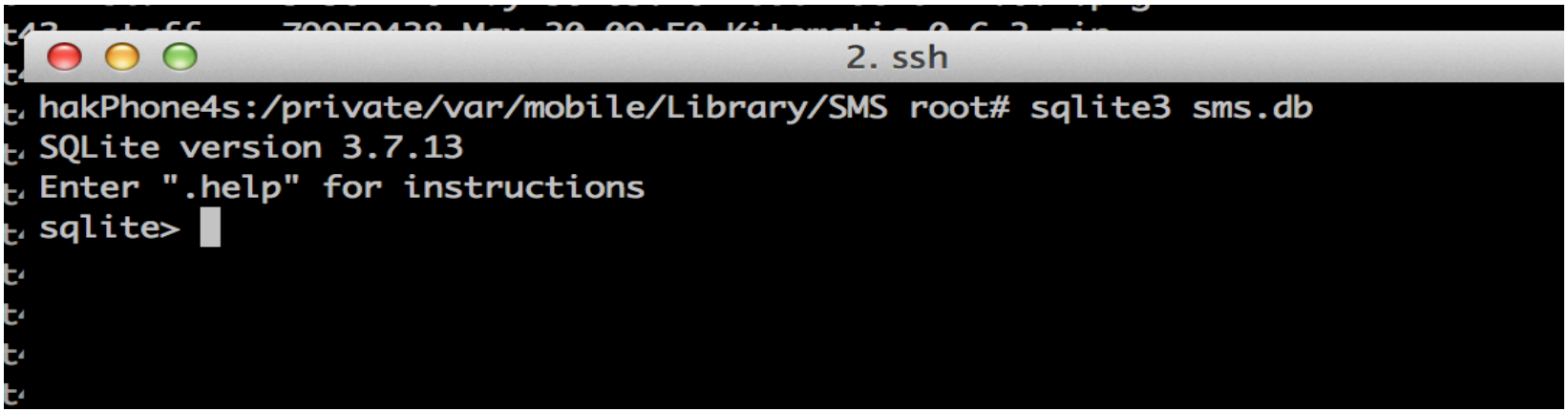
- Use plutil to examine the encoded plist files



iOS black box testing

- SQLite databases

- Use the sqlite3 command to read data



iOS black box testing

- Read SMS messages

```
sqlite> select * from message;
1|76694598-F845-523B-E758-D5643FD039D9|Uw tegoed voor deze periode is bijgeschreven
. Uw internetbundel is weer aangevuld.|0||1|||
                                                streamtyped???@???NSMutableAttributed
String|10|0|SMS|e:|2BABFF44-9950-4348-A3FA-4D7773C6A3C8|0|449052586|449065311|0|1|1
|0|0|0|0|0|0|1|0|0|0|0|0|0|0|0||1|0|0|0|0|0|0||0|0|0|0|0|0|0
2|6382363B-EB34-4A00-A085-B4AC1114AC89||0||2||||10|0|iMessage|p:+31637422847|667CC0
B3-D401-4C99-80C1-3AC6AA0F6738|0|449064641|0|0|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0||1
|0|0|0|0|1|3||0|0|0|0|0|0|0
3|874E9F55-9540-4A9D-9ADC-3248CE154027|Testje|0||3|||
                                                        streamtyped???@???NSMutableAtt
ributedString|10|0|iMessage|p:+31637422847|667CC0B3-D401-4C99-80C1-3AC6AA0F6738|0|4
49064576|449065312|0|1|1|0|0|0|0|0|0|1|0|0|0|0|0|0|0|0||1|0|0|0|0|0|0||0|0|0|0|0|0|
0
4|98F6AA3C-DBC7-F62B-4C2A-9E42BF3B32C8|Your allocated phone no. is 310820152. Thank
 you for joingin the REDteam BTS. Call the Yatebts IVR at(32843)|0||4|||
                                                                    streamtyped
???@???NSMutableAttributedString|10|0|SMS|e:|2BABFF44-9950-4348-A3FA-4D7773C6A3C8|0
|449141542|449141545|0|1|1|0|0|0|0|0|0|1|0|0|1|0|0|0|0|0||1|0|0|0|0|0|0||0|0|0|0|0|
0|0
```

iOS black box testing

- Keychain
  - Securely stores password, certificates, keys
  - Is encrypted with device and keybag specific keys
  - iOS has a single keychain (is a sqlite database) that stores all items for all applications
  - Application only has access to its own keys
  - /private/var/Keychains/keychain-2.db
  - Uses dataprotection classes as well (interesting: ThisDeviceOnly)

iOS black box testing

```
hakPhone4s:/private/var/Keychains root# ls -l
total 1328
-rw------- 1 _securityd wheel  16384 Apr 10 10:54 TrustStore.sqlite3
-rw------- 1 _securityd wheel     47 Jan  1  1970 accountStatus.plist
-rw------- 1 _securityd wheel      0 Feb  1 12:20 caissuercache.sqlite3
-rw------- 1 _securityd wheel    512 Jan  1  1970 caissuercache.sqlite3-journal
-rw------- 1 _securityd wheel 315392 Jun 13 13:41 keychain-2.db
-rw------- 1 _securityd wheel  32768 Jun 13 14:04 keychain-2.db-shm
-rw------- 1 _securityd wheel 416152 Jun 13 13:51 keychain-2.db-wal
-r-------- 1 root       wheel 141368 Jun 13 13:51 keychain-ota-backup.plist
-rw------- 1 _securityd wheel  45056 Jun 13 13:19 ocspcache.sqlite3
-rw------- 1 _securityd wheel  32768 Jun 13 14:07 ocspcache.sqlite3-shm
-rw------- 1 _securityd wheel 346112 Jun 13 13:53 ocspcache.sqlite3-wal
hakPhone4s:/private/var/Keychains root#
```

iOS black box testing

Keychain table	Description
`genp`	Generic passwords – `kSecClassGenericPassword`
`inet`	Internet passwords – `kSecClassInternetPassword`
`cert` and `keys`	Certificates, keys and digital identity (cert+key) items – `kSecClassCertificates` and `kSecClassIdentity`

iOS black box testing

- Keychain

```
Generic Password
----------------
Service: CommCenter
Account: SIM_PIN
Entitlement Group: apple
Label: (null)
Generic Field: <38393331 30373131 31323030 38323031 353232>
Keychain Data: 0123
```

iOS black box testing

- Log files
  - /var/log directory

```
total 0
drwxr-xr-x 3 root wheel 102 Feb  1 12:26 CoreCapture
drwxr-xr-x 2 root wheel 102 Mar  7 10:02 apt
drwxr-xr-x 2 root wheel  68 Feb  1 12:09 asl
drwx------ 2 root wheel 102 Feb  1 12:41 com.apple.revisiond
drwxr-xr-x 2 root wheel  68 Feb  1 12:09 com.apple.xpc.launchd
drwxr-xr-x 2 root wheel  68 Feb  1 12:09 ppp
drwxr-xr-x 2 root wheel  68 Feb  1 12:09 sa
hakPhone4s:~ root#
```

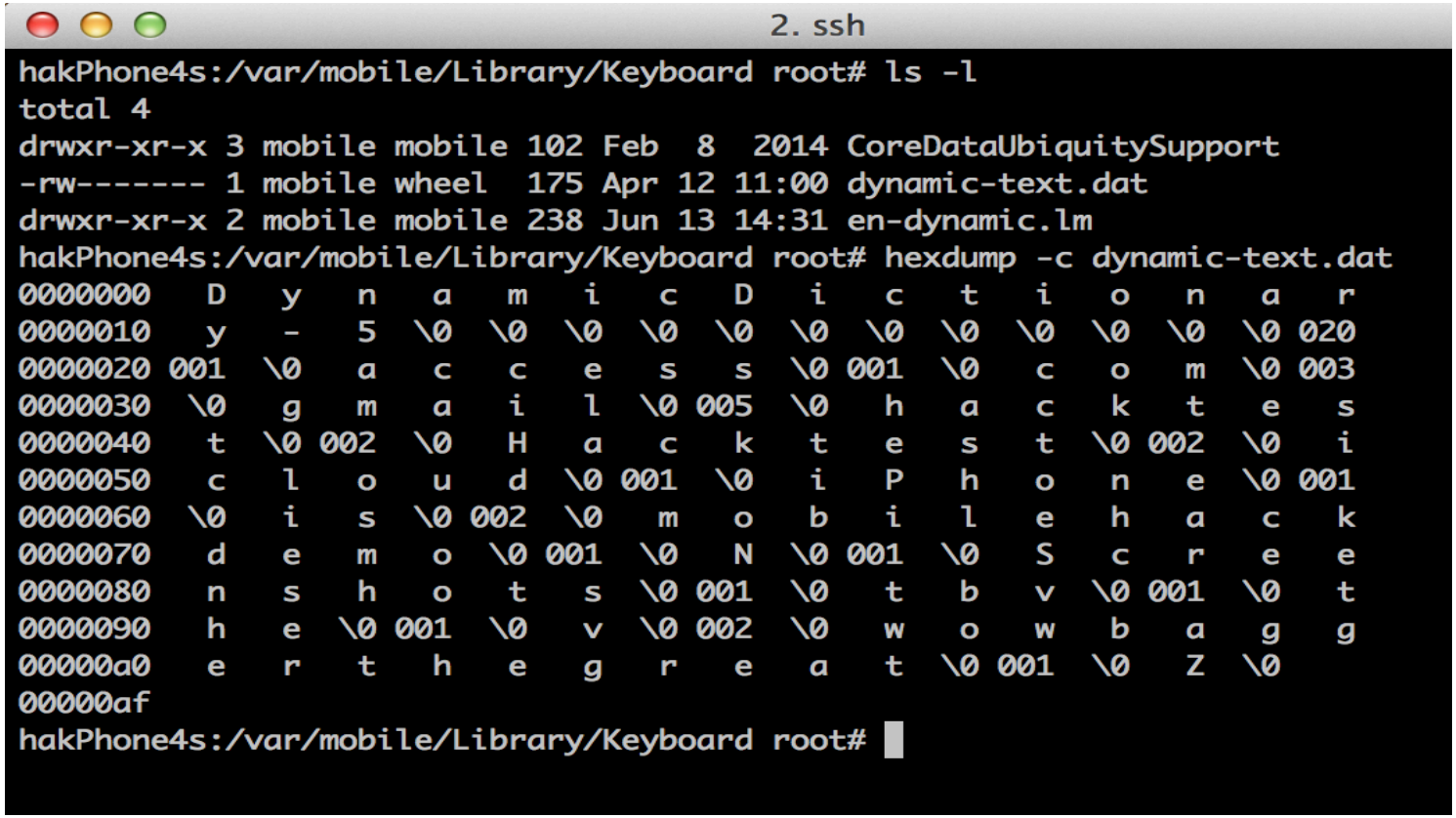
iOS black box testing

- Log files
  - May contain interesting system information

iOS black box testing

- Cache
  - Snapshots are taken in the Cache directory of the Application root
  - Can contain sensitive information from screenshot
  - Keyboard cache in /var/mobile/Library/Keyboard/dynamic- text.dat

iOS black box testing



iOS black box testing

- Applications are encrypted using FairPlay DRM
- To be able to test an application, you need to decrypt it
- Check with the otool

```
Tests-iPhone:/var/mobile/Applications/84593008-5B13-4CFC-BCEB-727DDC29DD50/Twitter.app root# otool -l Twit
          cmd LC_ENCRYPTION_INFO
      cmdsize 20
 cryptoff  8192
 cryptsize 5476352
 cryptid   1
```

Encrypted

iOS black box testing

- Use Clutchpatched or dumpdecrypted for decryption

```
DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib /var/mobile/Applications/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/Scan.app/Scan
```

iOS black box testing

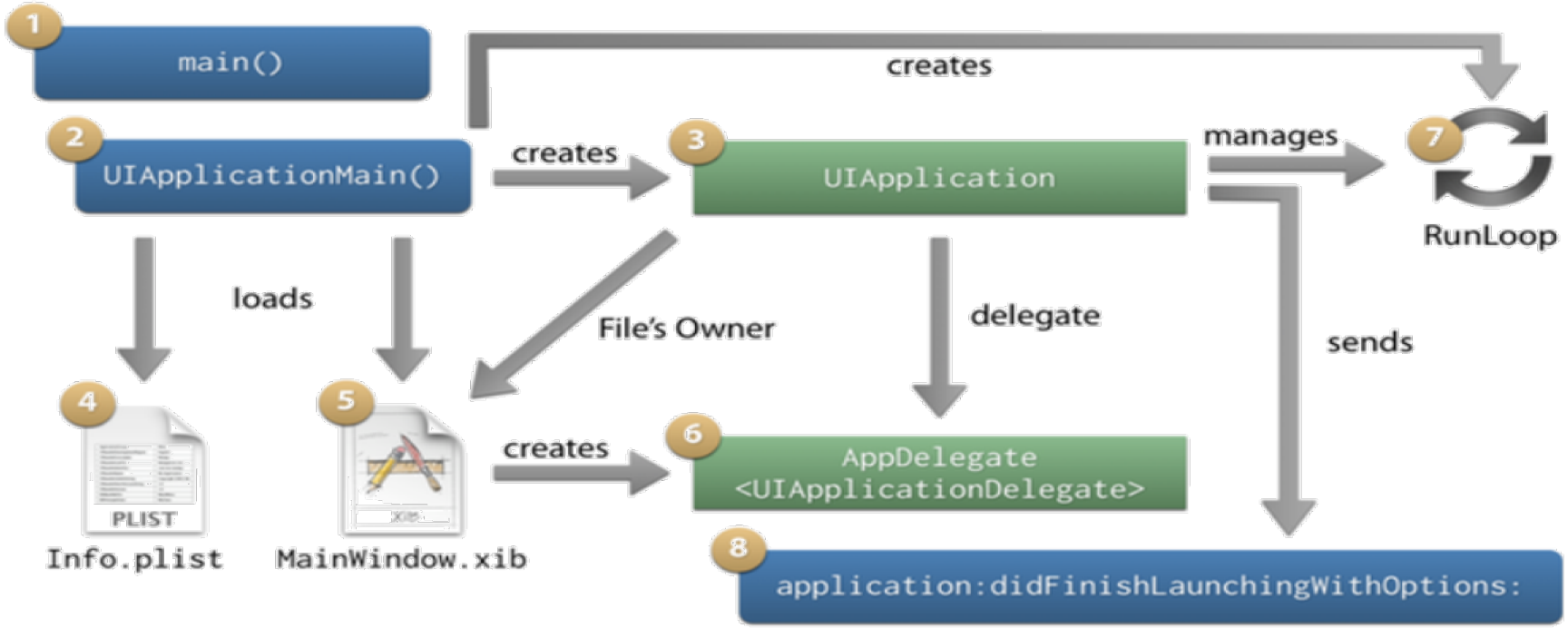
- Runtime manipulation
  - Cycript

```
cy# function printMethods(className) {
cy>     var count = new new Type("I");
cy>     var methods = class_copyMethodList(objc_getClass(className), count);
cy>     var methodsArray = [];
cy>     for(var i = 0; i < *count; i++) {
cy>         var method = methods[i];
cy>         methodsArray.push({selector:method_getName(method), implementation:method_getImplementatio
n(method)});
cy>     }
cy>     free(methods);
cy>     free(count);
cy>     return methodsArray;
cy> }
cy#
```

iOS black box testing

- Cycript
  - Combination of Interactive Objective-C interpreter and Javascript
  - App runtime behaviour can be manipulated
  - Can connect to an active process
  - Gives access to all classes and variables of an app
  - Used for runtime analysis to:
    - Bypass security locks
    - Access sensitive information in memory
    - Bypass authentication
    - Access restrictive parts of the application

iOS black box testing

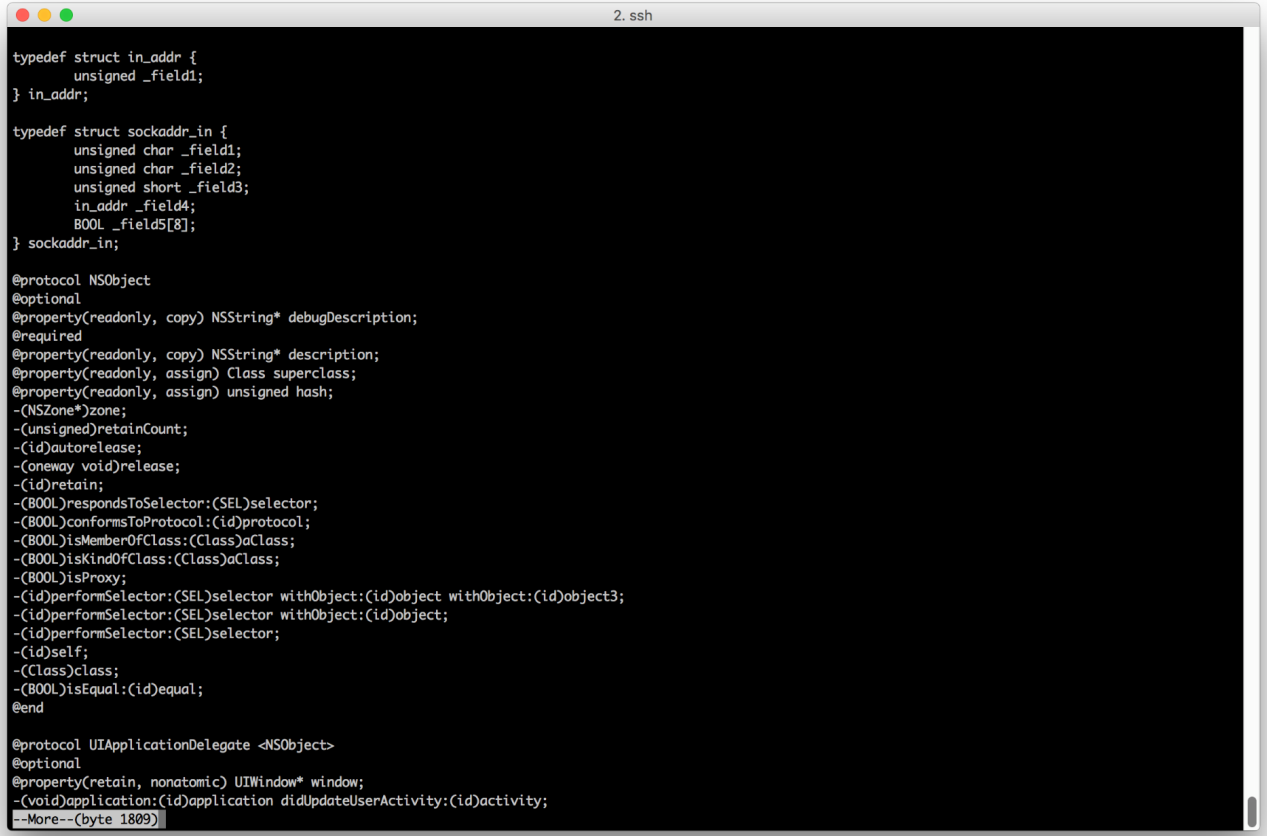


iOS black box testing

- To be able to determine the object structure of the app you can dump it with class-dump or class-dump-z

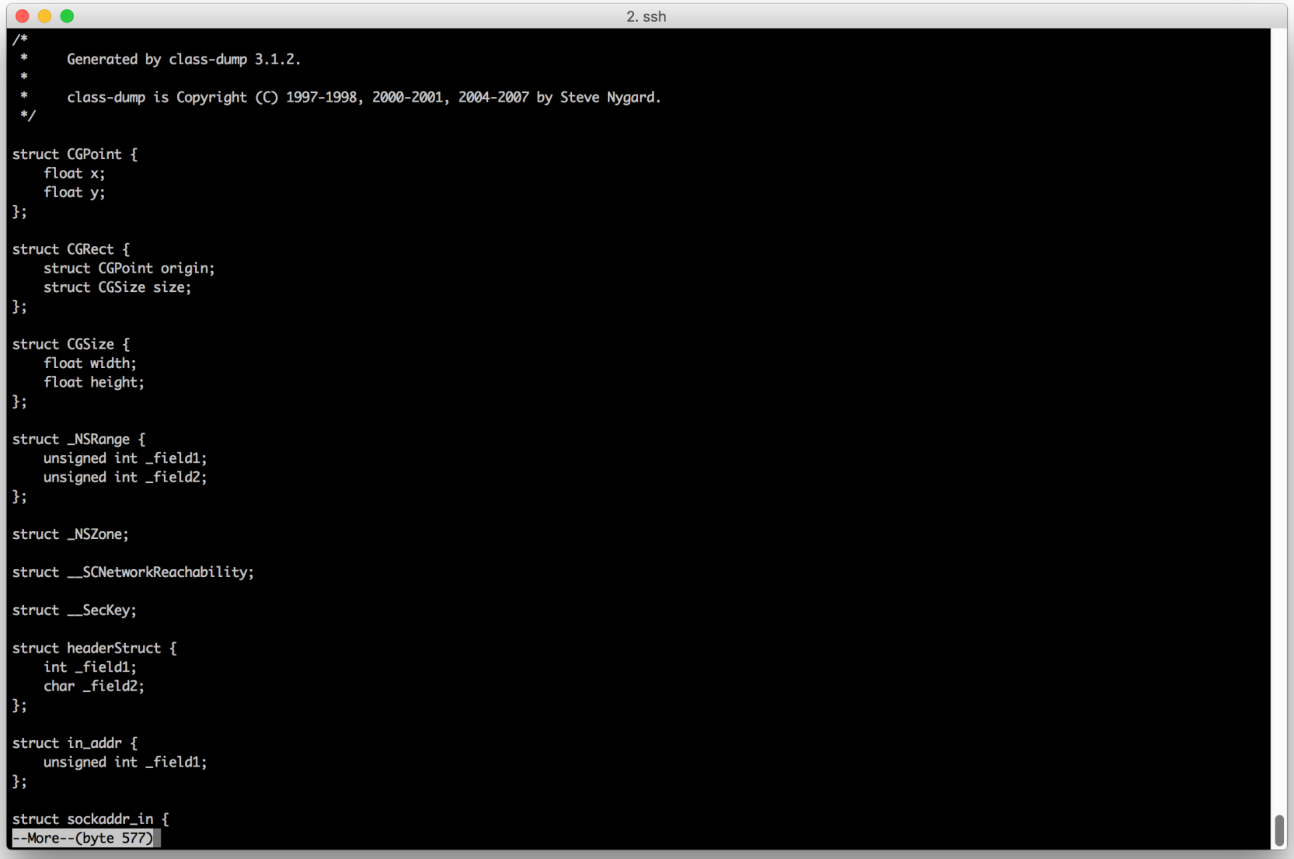
iOS black box testing

- class-dump-z



iOS black box testing

- class-dump



Challenges?

- No jailbreak available
- Extensive rooting detections
- Runtime detection by applications

Solution?

Frida

http://www.frida.re/

Quick-start Instructions

```
~ $ sudo pip install frida
~ $ frida-trace -i "recv*" Twitter
recvfrom: Auto-generated handler: …/recvfrom.js
Started tracing 21 functions.
1442 ms     recvfrom()
# Live-edit recvfrom.js and watch the magic!
5374 ms     recvfrom(socket=67, buffer=0x252a618,
length=65536, flags=0, address=0xb0420bd8,
address_len=16)
```

Frida

Provides a runtime environment that allows you to hook functions in an application and manipulate the flow, read data etc.

Frida

WORKSHOP!!!

Find your goodies at:

https://github.com/theart42/hack.lu