HAKIN9

PRACTICAL PROTECTION

IT SECURITY MAGAZINE

80+ PAGES

HACKING ANDROID

HOW TO RESEARCH AN APK

ANDROID OS: GETTING STARTED WITH CUSTOMIZING YOUR OWN ROM

DECOMPILING ANDROID WORKSHOP

APPUSE – ANDROID PENTEST PLATFORM UNIFIED STANDALONE ENVIRONMENT

Issue 03/2013(63) ISSN: 1733-7186

PLUS

AN INTERVIEW WITH OMAR KHAN, THE CO-CEO OF NQ MOBILE

If you would like to receive the custom wallpaper used for this article, you can download it for **FREE** from the EaglesBlood™ Development website.

http://www.EaglesBlood.com

Atola Insight

That's all you need for data recovery.

Atola Technology offers *Atola Insight* – the only data recovery device that covers the entire data recovery process: *in-depth* **HDD diagnostics**, **firmware recovery**, **HDD duplication**, and **file recovery**. It is like a whole data recovery Lab in one Tool.

This product is the best choice for seasoned professionals as well as start-up data recovery companies.

Emphasized features at a glance:

- Automatic in-depth diagnostic of all hard drive components
- Automatic firmware recovery and ATA password removal
- Very fast imaging of damaged drives
- Imaging by heads
- Case management
- Real time current monitor
- Firmware area backup system
- Serial port and power control
- Write protection switch

Atola TECHNOLOGY

Visit atola.com for details

PRACTICAL PROTECTION IT SECURITY MAGAZINE

team

Editor in Chief: Krzysztof Samborski
krzysztof.samborski@hakin9.org

Editorial Advisory Board: Jeff Smith, Peter Harmsen, Kishore P.V

Proofreaders: Krzysztof Samborski

Special Thanks to the Beta testers and Proofreaders who helped us with this issue. Without their assistance there would not be a Hakin9 magazine.

Senior Consultant/Publisher: Paweł Marciniak

CEO: Ewa Dudzic
ewa.dudzic@hakin9.org

Product Manager: Krzysztof Samborski
krzysztof.samborski@hakin9.org

Production Director: Andrzej Kuca
andrzej.kuca@hakin9.org

Marketing Director: Krzysztof Samborski
krzysztof.samborski@hakin9.org

DTP: Ireneusz Pogroszewski
Art Director: Ireneusz Pogroszewski
ireneusz.pogroszewski@software.com.pl

Publisher: Hakin9 Media sp. z o.o. SK
02-676 Warszawa, ul. Postępu 17D
Phone: 1 917 338 3631
www.hakin9.org

Dear Hakin9 Readers,

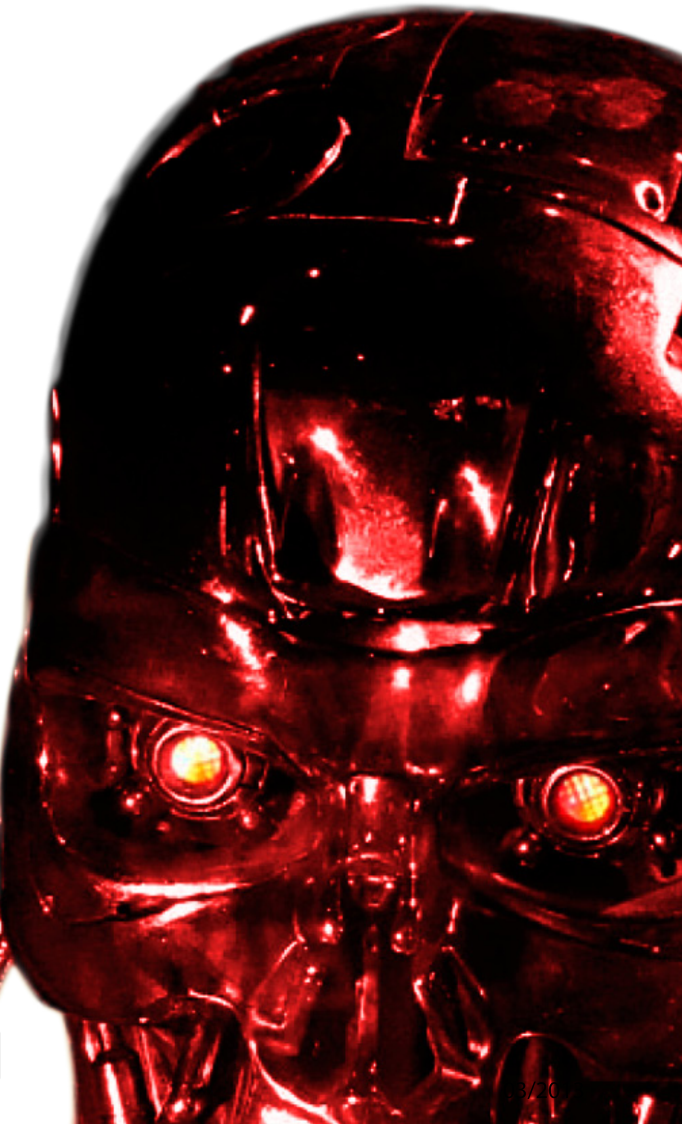
Android is a Linux-based operating system designed for mobile devices such as smartphones and tablet computers. At the beginning, it was developed by Android Inc. and later in 2005 bought by Google.

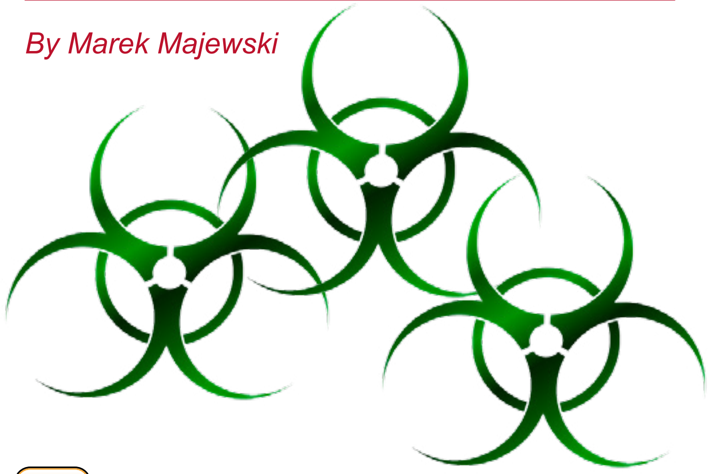
Latest research has shown that Android users become more and more threatened by malware. A number of attacks rises every day and these are getting more dangerous for it's users. We have been asked to do some study and we decided to provide you with an issue addressing this topic.

You can surely notice that we divided the issue into sections. In the first section you will find the articles dedicated to Android security. In the second section you will find the articles dedicated to Android laboratory. In the third section you will find some extra articles.

Hope you enjoy the magazine!

Krzysztof Samborski
Hakin9 Product Manager
and Hakin9 Team





Android Security

Android, as we are all aware, is a Linux-based operating system which was initially developed by Android Inc. and was later purchased by Google. It was designed for touch screen devices like smart phones, tablets, cameras, set-top boxes, etc. and has reached the hands of millions of consumers.

In this period, security firms are publishing detailed reports on analysis conducted on principal cyber threats detected in 2012, the results proposed present a landscape dominated by explosion of menaces, especially for mobile and social media users.

Mobile technology has grown more than any other in the last few years and the IT industry, to respond customer's demands, has designed an impressive number of solutions and services specific for mobile platforms. Due the growing trend, many factors have been attracted by the possibility to exploit the mobile solutions for various purposes; let's think, for example, to cyber criminals or state-sponsored hackers that have started to research possible attack schemes against mobile platforms.

Another factor that must be considered when analyzing the rise of cyber threats against mobile platforms is the lack of awareness of users on the risks related to an improper use of mobile devices, in majority of case users don't apply mechanisms of defense to their mobiles, and often they totally ignore them; a customer's habit could cause serious damage (Figure 1).

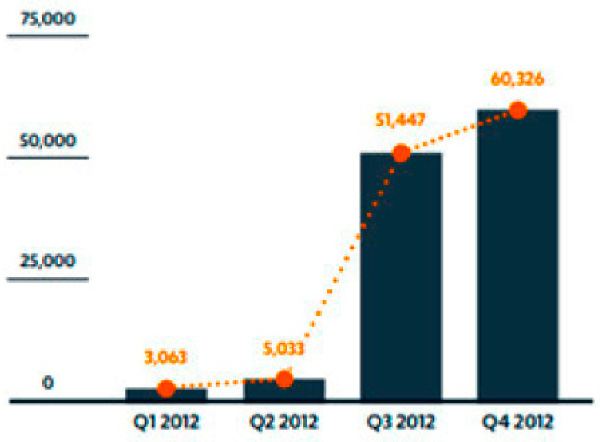


Figure 1. *Number of Android Threats Received per quarter, Q1-Q4 2012*

Background

Hardware

Android runs on a wide range of hardware configurations including smart phones, tablets, and set-top-boxes. Android is processor-agnostic, but it does take advantage of some hardware-specific security capabilities such as ARM v6 eXecute-Never.

OS

The core operating system is built on top of the Linux kernel. All device resources, like camera functions, GPS data, Bluetooth functions, telephony functions, network connections, etc. are accessed through the operating system.

Software

The core operating system is built on top of the Linux kernel. All device resources, like camera functions, GPS data, Bluetooth functions, telephony functions, network connections, etc. are accessed through the operating system.

Google Play is a collection of services that allow users to discover, install, and purchase applications from their Android device or the web. Google Play makes it easy for developers to reach Android users and potential customers. Google Play also provides community review, application license verification, application security scanning, and other security services.

Updates

The Android update service delivers new capabilities and security updates to Android devices, including updates through the web or over the air (OTA).

Services

Frameworks that allow Android applications to use cloud capabilities such as(backing up) application data and settings and cloud-to-device messaging (C2DM) for push messaging.

As per the various reports published by antivirus companies, there has been an exponential increase in the malware attacks for Android devices. These malwares includes SMS Trojans (drain victims' mobile accounts by sending SMS messages), backdoors (give hackers' access to a Smartphone, allowing them to install other malware or steal personal data) and spyware (collects personal data such as contacts and passwords). This brings in the need for developers to develop secure apps to safeguard their customers against various threats.

Architecture

Let's discuss about above architecture of Android (Figure 2).

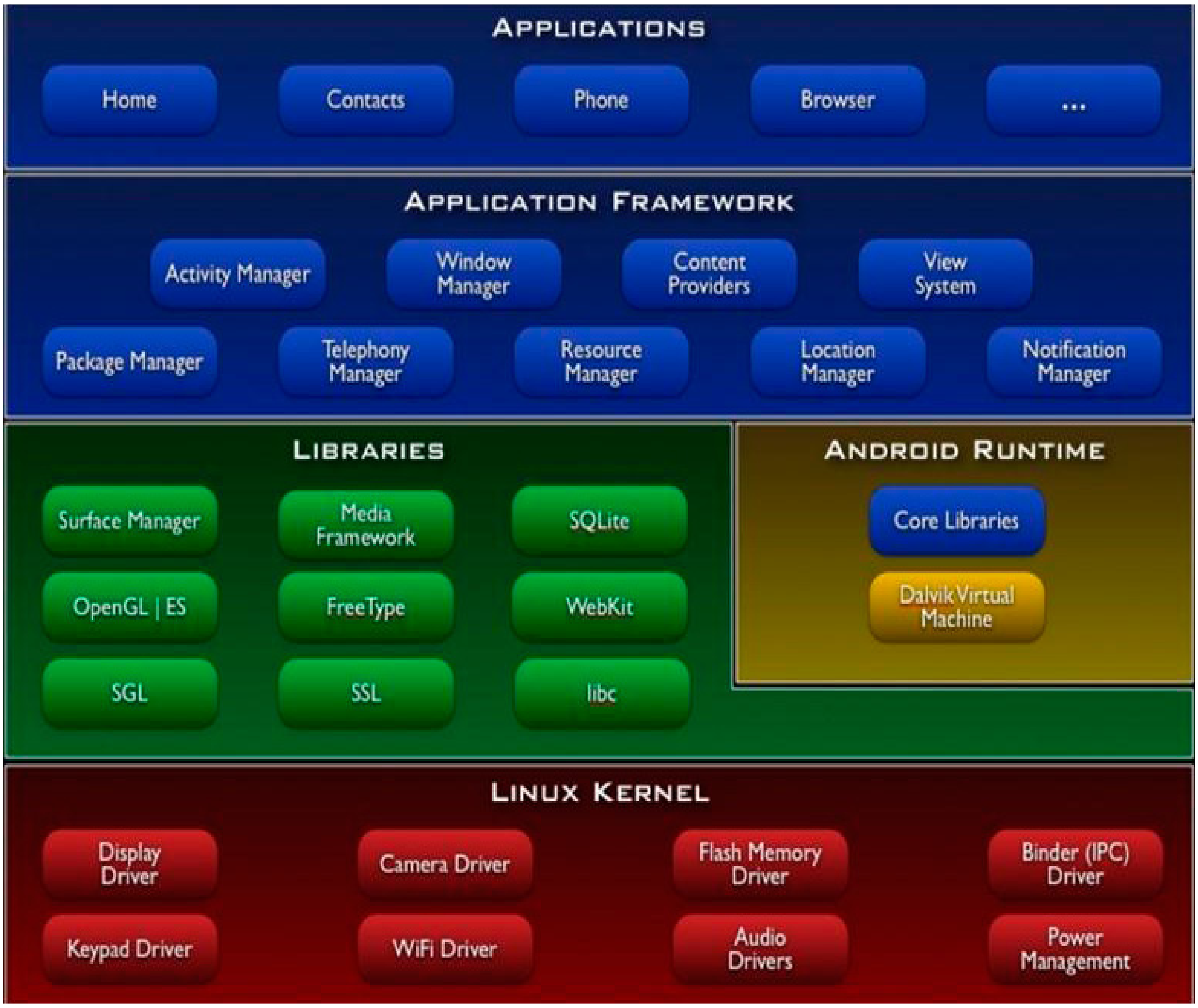


Figure 2. *Android Architecture*

Linux Kernel

The bottom most layer consists of the Linux Kernel. The Android OS is built on the version of the Linux Kernel 2.6 with some architectural changes. This layer consists of the various drivers like camera, audio, Wi-Fi, keypad drivers, etc. Android relies on Linux for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

We all have worked on traditional desktop platforms like Windows or Linux which run the applications under the user who starts them. For example, if a particular user installs and runs a software, it runs with the same set of permissions

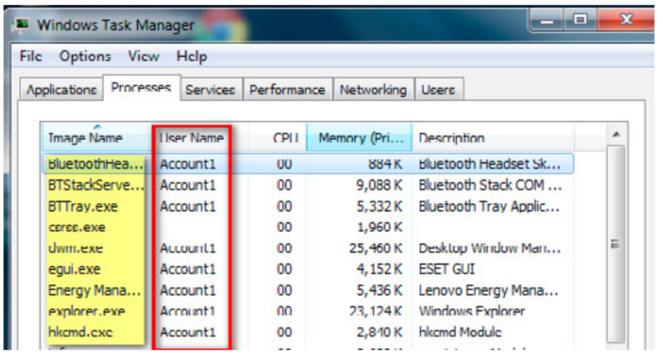
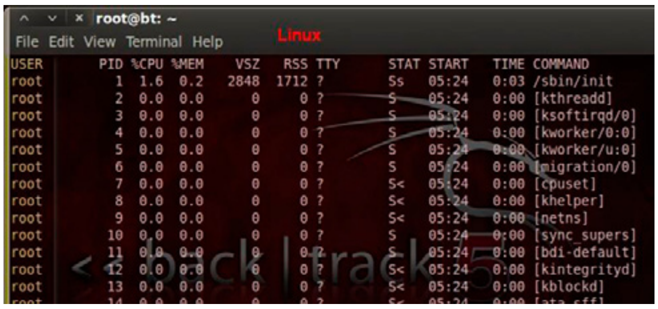

Windows Task Manager
File Options View Help
Applications Processes Services Performance Networking Users

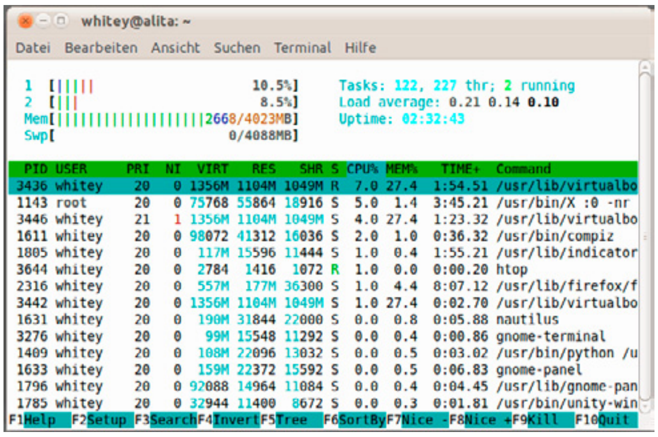
Image Name	User Name	CPU	Memory (Pri...	Description
BluetoothHea...	Account1	00	884 K	Bluetooth Headset Sk...
BTStackServe...	Account1	00	9,088 K	Bluetooth Stack COM ...
BTTray.exe	Account1	00	5,332 K	Bluetooth Tray Applic...
csrss.exe		00	1,960 K	
dwm.exe	Account1	00	25,460 K	Desktop Window Man...
egui.exe	Account1	00	4,152 K	ESET GUI
Energy Mana...	Account1	00	5,436 K	Lenovo Energy Mana...
explorer.exe	Account1	00	23,124 K	Windows Explorer
hkcmd.exe	Account1	00	2,840 K	hkcmd Module

Figure 3. *Windows task manager*



```
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  1.6  0.2   2848  1712 ?        Ss   05:24   0:03 /sbin/init
root          2  0.0  0.0      0     0 ?        S    05:24   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    05:24   0:00 [ksoftirqd/0]
root          4  0.0  0.0      0     0 ?        S    05:24   0:00 [kworker/0:0]
root          5  0.0  0.0      0     0 ?        S    05:24   0:00 [kworker/u:0]
root          6  0.0  0.0      0     0 ?        S    05:24   0:00 [migration/0]
root          7  0.0  0.0      0     0 ?        S<   05:24   0:00 [cpuset]
root          8  0.0  0.0      0     0 ?        S<   05:24   0:00 [khelper]
root          9  0.0  0.0      0     0 ?        S<   05:24   0:00 [netns]
root         10  0.0  0.0      0     0 ?        S    05:24   0:00 [sync_supers]
root         11  0.0  0.0      0     0 ?        S    05:24   0:00 [bdi-default]
root         12  0.0  0.0      0     0 ?        S<   05:24   0:00 [kintegrityd]
root         13  0.0  0.0      0     0 ?        S<   05:24   0:00 [kblockd]
```

Figure 4. *Linux Task manager*



```
  1  [|||||                   10.5%]     Tasks: 122, 227 thr; 2 running
  2  [|||                      8.5%]     Load average: 0.21 0.14 0.10
  Mem[||||||||||||||||||2668/4023MB]     Uptime: 02:32:43
  Swp[                     0/4088MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 3436 whitey     20   0 1356M 1104M 1049M R  7.0 27.4  1:54.51 /usr/lib/virtualbo
 1143 root       20   0 75768 55864 18916 S  5.0  1.4  3:45.21 /usr/bin/X :0 -nr
 3446 whitey     21   1 1356M 1104M 1049M S  4.0 27.4  1:23.32 /usr/lib/virtualbo
 1611 whitey     20   0 98072 41312 16036 S  2.0  1.0  0:36.32 /usr/bin/compiz
 1805 whitey     20   0  117M 15596 11444 S  1.0  0.4  1:55.21 /usr/lib/indicator
 3644 whitey     20   0  2784  1416  1072 R  1.0  0.0  0:00.20 htop
 2316 whitey     20   0  557M  177M 36300 S  1.0  4.4  8:07.12 /usr/lib/firefox/f
 3442 whitey     20   0 1356M 1104M 1049M S  1.0 27.4  0:02.70 /usr/lib/virtualbo
 1631 whitey     20   0  190M 31844 22000 S  0.0  0.8  0:05.88 nautilus
 3276 whitey     20   0   99M 15548 11292 S  0.0  0.4  0:00.86 gnome-terminal
 1409 whitey     20   0  108M 22096 13032 S  0.0  0.5  0:03.02 /usr/bin/python /u
 1633 whitey     20   0  159M 22372 15592 S  0.0  0.5  0:06.83 gnome-panel
 1796 whitey     20   0 92088 14964 11084 S  0.0  0.4  0:04.45 /usr/lib/gnome-pan
 1785 whitey     20   0 32944 11400  8672 S  0.0  0.3  0:01.81 /usr/bin/unity-win
F1Help  F2Setup F3SearchF4InvertF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figure 5. *Limited User Task Manager*

as that of the user. If this software turns out to be malicious, then this software would be allowed by the operating system to steal/access the sensitive details/files stored on the user machine. This is because Windows or Linux run all processes under the same user permission. But if you login with regular user account than you cannot see all the processes.

The Figure 3 and 4 shows the processes running under the same user account for Windows operating system.

If you login with regular user than you can show Figure 5.

Since Linux is the heart of the Android operating system, similar security features are inherited into the Android system. To understand this better, we must first understand the Linux security model. Linux security is based on the concept of users and groups. Linux assigns a unique user-id (UID) whenever a new user is created and these users can be added to a group which has a unique group id (GID), which is used to distinguish between other groups. Each file on Linux has the UID of the particular user assigned to it. Only this user has the highest privilege, access rights for the file and can alter the permission on it.

As Android is developed on Linux, the above concepts apply to it as well. When a new android package is installed, it is assigned a new user-id and all the data (files/database) stored by this application are also assigned the same UID. As a result, the Linux permissions on the data for that application are set to follow the full permissions of the associated UID and no other permissions. Linux security prevents applications that have different UIDs from accessing data, process or memory of other applications, thus providing security and separation between the applications on the Android platform. See Figure 6 for that.

Libraries

The layer above to the Linux kernel is the Android's native libraries. These libraries are written in C/C++ languages. They are used by various components of the android system and are also exposed to the developers through the Android application framework (the layer above it). These libraries also run as processes within the underlying Linux kernel. The libraries are nothing but a set of instructions that tell the device how to handle different kinds of data (e.g. The media libraries support playing or recording various audio/video formats). For example SQLite, Webkit, Surface manager, OpenGL.

Android Runtime

This is located on the same layer as the libraries layer. It consists of the core JAVA libraries and the Dalvik virtual machine. The core Java libraries are used for developing Android based applications.

A virtual machine, as we are aware, is a virtual environment with its own operating system. Android uses the concept of the Dalvik virtual machine, which has been designed to run multiple VMs efficiently. Android OS uses these virtual machines to run each application as its own process.

Dalvik VMs help in achieving the following:

- better memory management
- an application cannot interfere with other applications without permissions
- threading support

The diagram below is a pictorial representation of the Android environment. It can be observed that each Android application runs under a separate virtual instance and each application has a unique user-id assigned to it (Figure 7).

Application Framework

The next layer above the libraries is the Application framework. These include the programs that manage the basic functions of the phone like resource allocation, voice call management, etc. The developers can use these framework APIs to develop further complex applications.

Some of the important blocks of this framework are the resource manager (handles resource management), location manager (location based services like maps and GPS), activity manager (manages the activity of the application life cycle), telephone manager (manages voice calls), and content provider (manages data sharing between applications).

Applications

At the top of the stack are the applications themselves. These include the applications shipped with android like the email client, SMS client, maps, browsers and also the applications developed and distributed through the Android market.

Android Security

Penetration Testing on Android

The Android based applications are complex as compared to the browser-based applications.

These applications deal with local, as well as server-side processing. As a result, a separate approach is taken for testing these applications.

The Android-based applications might involve HTTP/HTTPS traffic as well and might carry out local storage and processing. However, for local storage and processing, the techniques listed below can be used for assessing the application.

In order to test any application in Android, we need to install the application in an emulator. This can be achieved by using the Android Debug Bridge.

The Android Debug Bridge is a part of Google's Android development toolkit which provides a command line interface to connect to the emulator running Android. This can be used to push applications and install the Android application package files (.apk files).

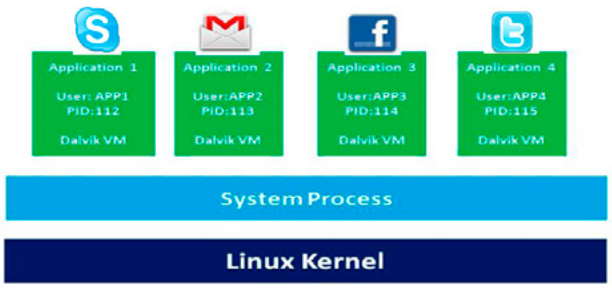
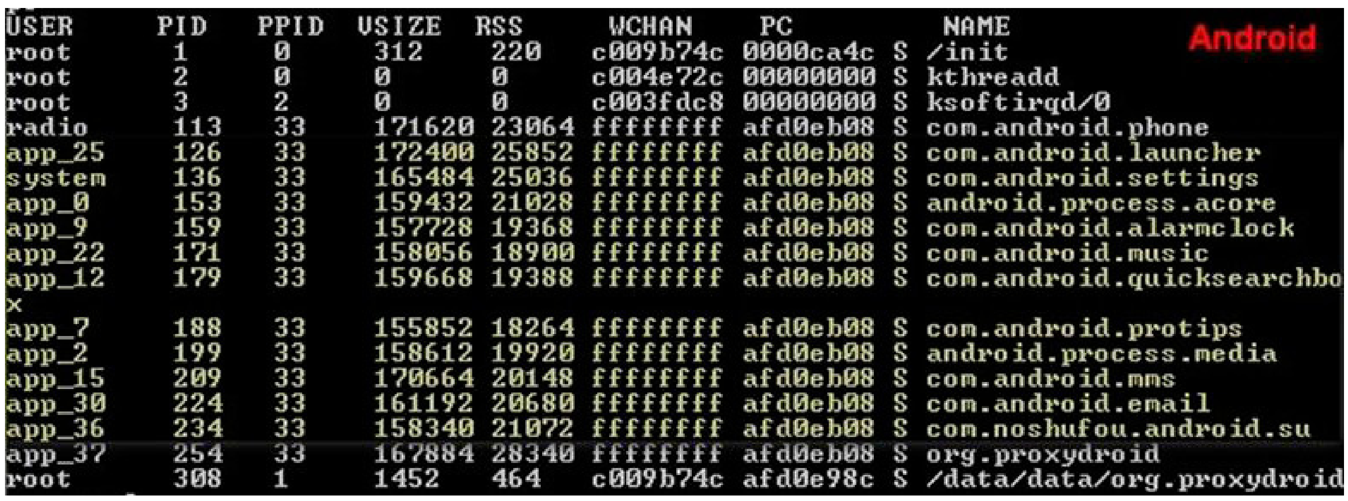


Figure 7. *Android Environment*



```
USER     PID   PPID  VSIZE  RSS    WCHAN    PC         NAME                    Android
root     1     0     312    220    c009b74c 0000ca4c S /init
root     2     0     0      0      c004e72c 00000000 S kthreadd
root     3     2     0      0      c003fdc8 00000000 S ksoftirqd/0
radio    113   33    171620 23064  ffffffff afd0eb08 S com.android.phone
app_25   126   33    172400 25852  ffffffff afd0eb08 S com.android.launcher
system   136   33    165484 25036  ffffffff afd0eb08 S com.android.settings
app_0    153   33    159432 21028  ffffffff afd0eb08 S android.process.acore
app_9    159   33    157728 19368  ffffffff afd0eb08 S com.android.alarmclock
app_22   171   33    158056 18900  ffffffff afd0eb08 S com.android.music
app_12   179   33    159668 19388  ffffffff afd0eb08 S com.android.quicksearchbo
x
app_7    188   33    155852 18264  ffffffff afd0eb08 S com.android.protips
app_2    199   33    158612 19920  ffffffff afd0eb08 S android.process.media
app_15   209   33    170664 20148  ffffffff afd0eb08 S com.android.mms
app_30   224   33    161192 20680  ffffffff afd0eb08 S com.android.email
app_36   234   33    158340 21072  ffffffff afd0eb08 S com.noshufou.android.su
app_37   254   33    167884 28340  ffffffff afd0eb08 S org.proxydroid
root     308   1     1452   464    c009b74c afd0e98c S /data/data/org.proxydroid
```

Figure 6. *Android Task manager*

Some of the key ADB commands are listed as follows:

```
adb install <path to the apk>
```

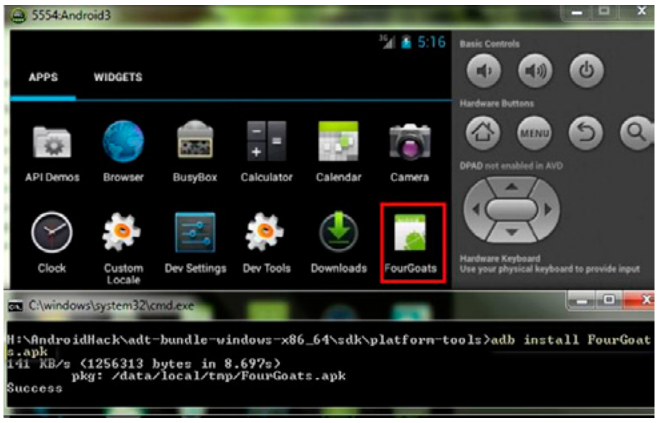
This command will copy and install the application from the user computer to the emulator Instance (Figure 8).

During installation if there are any errors or if the devices is not recognized, then the below two commands are very much useful in restarting the ADB service.

- adb kill-server (terminates the ADB process)
- adb start-server (checks if the ADB server is running and starts if not)

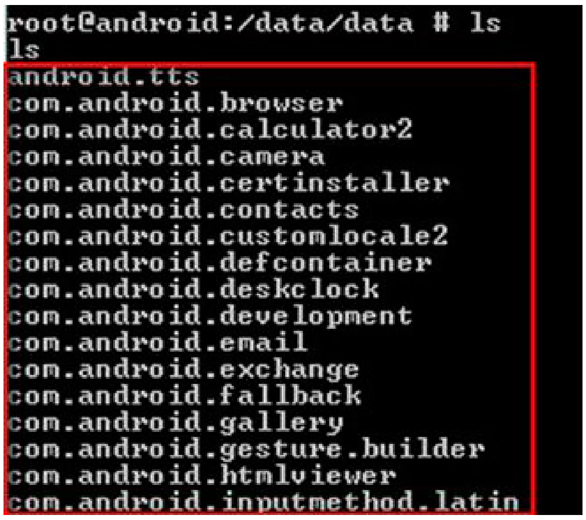
Local Storage Analysis

Android applications store sensitive data such as credentials, credit card numbers, and more in plain text in the local storage. As mobile devices are at a higher risk of getting stolen than a desktop or server, we need to ensure that the applications do not store any sensitive data locally.



```
H:\AndroidHack\adt-bundle-windows-x86_64\sdk\platform-tools>adb install FourGoat
s.apk
141 KB/s (1256313 bytes in 8.697s)
        pkg: /data/local/tmp/FourGoats.apk
Success
```

Figure 8. *Android ADB*



```
root@android:/data/data # ls
ls
android.tts
com.android.browser
com.android.calculator2
com.android.camera
com.android.certinstaller
com.android.contacts
com.android.customlocale2
com.android.defcontainer
com.android.deskclock
com.android.development
com.android.email
com.android.exchange
com.android.fallback
com.android.gallery
com.android.gesture.builder
com.android.htmlviewer
com.android.inputmethod.latin
```

Figure 9. *ADB Shell*

We will use the Android debug bridge (ADB) shell feature to browse the file system to determine if the application insecurely sores data locally.

We will now discuss the various features of ADB that can be used for pen-testing Android apps.

ADB shell

ADB provides an ash shell that can be used to run a variety of commands on the emulator. By using the ADB shell, a variety of vulnerabilities can be tested for against the Android applications.

The command used to access the shell is:

```
#adb shell
```

After getting access to the shell, we can browse the internal directories using Linux commands as shown here:

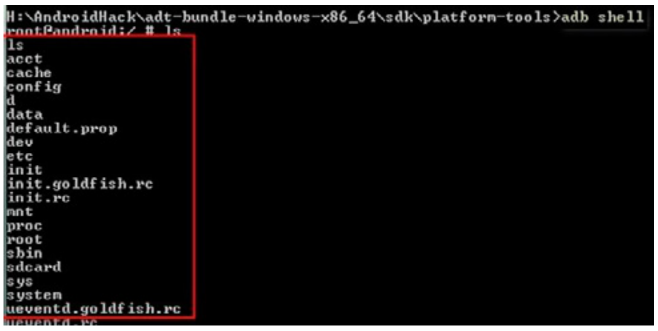
```
cd (change directory)
ls (lists the information about the files)
```

In order to access the data of a particular application, we need to first access that particular application package folder as shown Figure 10 and Figure 11.

Accessing the particular application package gives access to the data stored by that application.

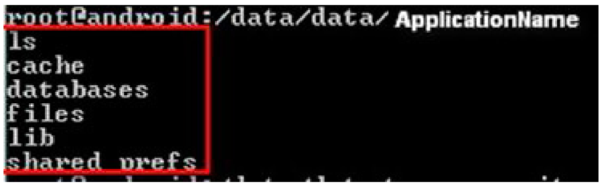
Each of the application packages contains the following folders:

- Cache
- Databases
- Shared_prefs
- Lib
- Files



```
H:\AndroidHack\adt-bundle-windows-x86_64\sdk\platform-tools>adb shell
root@android:/ # ls
ls
acct
cache
config
d
data
default.prop
dev
etc
init
init.goldfish.rc
init.rc
mnt
proc
root
sbin
sdcard
sys
system
ueventd.goldfish.rc
ueventd.rc
```

Figure 10. *List*



```
root@android:/data/data/ ApplicationName
ls
cache
databases
files
lib
shared_prefs
```

Figure 11. *Folders of Application*

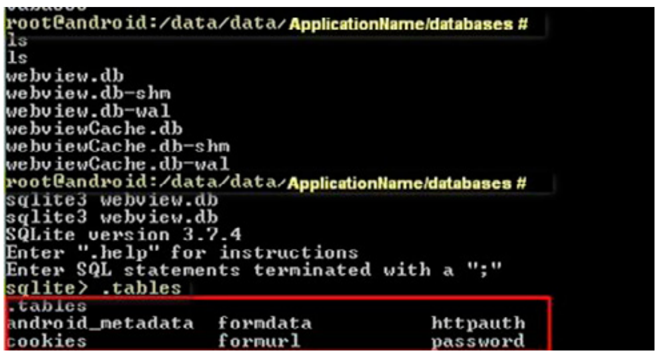
Cache

Cache is the directory which contains the files cached by the applications. These often contain the files from the web browsers or other apps that use the WebKit engine (it is an engine that is used by Android apps to render web pages).

Using the ADB shell, we can browse to the cache folder and check if any sensitive files are being cached by the application.

Database

The Android device contains a SQLite database which is an open source database. It supports standard relational database features like SQL syntax, transactions and prepared statements. Using a SQLite database in Android does not require any database setup or administration. You only have to define the SQL statements for creating and updating the database. Afterwards, the database is automatically managed for you by the Android platform.



```
root@android:/data/data/ApplicationName/databases #
ls
ls
webview.db
webview.db-shm
webview.db-wal
webviewCache.db
webviewCache.db-shm
webviewCache.db-wal
root@android:/data/data/ApplicationName/databases #
sqlite3 webview.db
sqlite3 webview.db
SQLite version 3.7.4
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
.tables
android_metadata   formdata           httpauth
cookies            formurl            password
```

Figure 12. *Database*

The applications might store some sensitive data into these databases. This could include user credentials, encryption keys, credit card details, etc.

Using the ADB shell, we can browse to the database folder and access the data as shown in Figure 12.

Shared_prefs

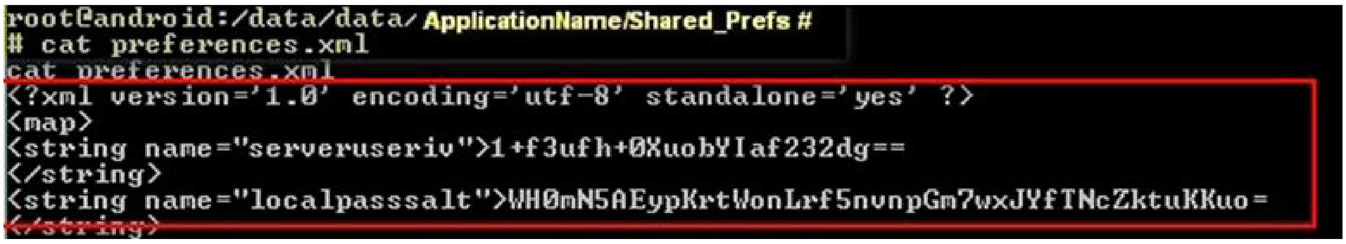
Shared Preferences is a framework that allows the Android applications to store and retrieve key-value pairs of primitive data types (booleans, floats, ints, longs, and strings). This data will persist even if the application is closed (killed). The shared_prefs folder contains the .xml files which should be checked for sensitive values. See the following image (Figure 13).

Files [System Log Inspection]

The logging system of Android provides a mechanism to view the debug output. Logs from various applications are collected in a series of circular buffers, which then can be viewed and filtered by the logcat command. These include the files stored by the Android applications into the files directory. They might be sensitive information like files or images being stored and should be checked (Figure 14).

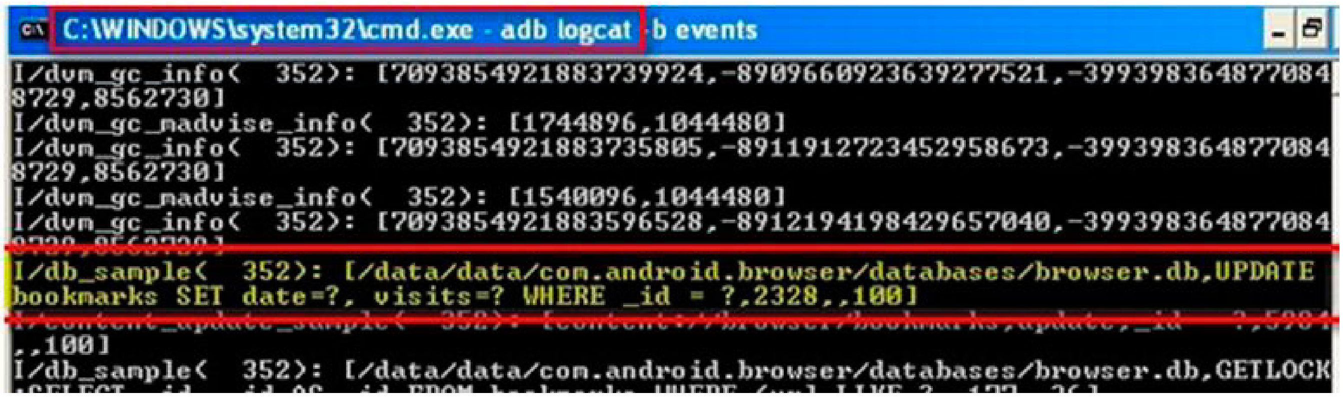
Intent Sniffing

Intent is basically a request for a certain action to take place. The Android applications make use of intents for both inter-application and intra-application communication.



```
root@android:/data/data/ApplicationName/Shared_Prefs #
# cat preferences.xml
cat preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="serveruseriv">1+f3ufh+0XuobYIaf232dg==
</string>
<string name="localpasssalt">WH0mN5AEypKrtWonLrf5nvnpGm7wxJYfTNcZktuKKuo=
</string>
```

Figure 13. *Shared Preferences*



C:\WINDOWS\system32\cmd.exe - adb logcat -b events

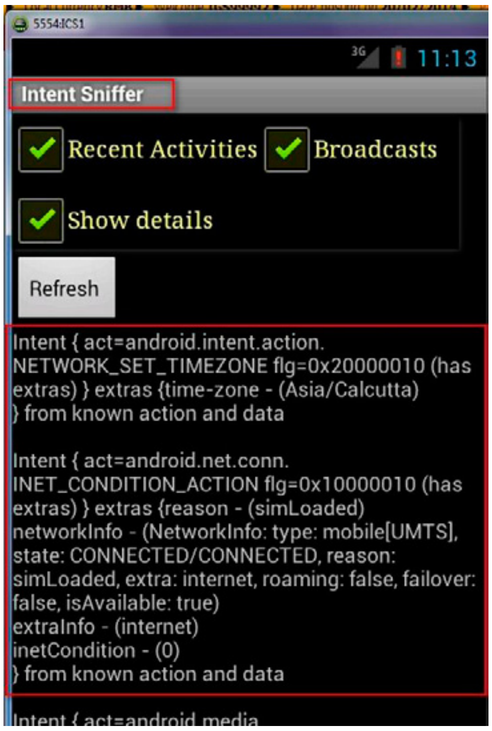
```
I/dvm_gc_info(  352): [7093854921883739924,-8909660923639277521,-39939836487708 4
8729,8562730]
I/dvm_gc_madvise_info(  352): [1744896,1044480]
I/dvm_gc_info(  352): [7093854921883735805,-8911912723452958673,-39939836487708 4
8729,8562730]
I/dvm_gc_madvise_info(  352): [1540096,1044480]
I/dvm_gc_info(  352): [7093854921883596528,-8912194198429657040,-39939836487708 4
I/db_sample(  352): [/data/data/com.android.browser/databases/browser.db,UPDATE
bookmarks SET date=?, visits=? WHERE _id = ?,2328,,100]
,,100]
I/db_sample(  352): [/data/data/com.android.browser/databases/browser.db,GETLOCK
```

Figure 14. *System Log*

The contents of Intents can be sniffed, modified, stolen, or replaced, which can compromise user privacy. Also, a malicious application can inject forged or otherwise malicious Intents, which can lead to violation of application security policies (Figure 15).

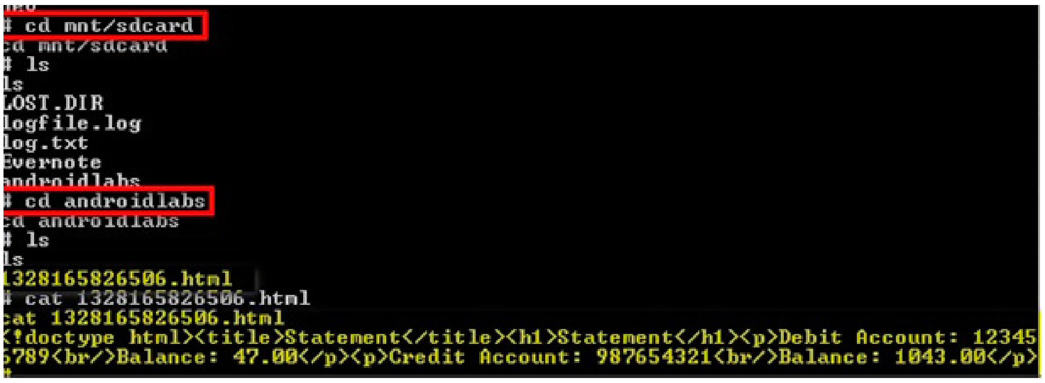
SD Card Storage

Android devices provide both internal and external storage (SD Card). In a typical Android mobile application, the file system is sandboxed into the directories, thus preventing malicious applications from accessing the data of other applications. The storage of data on SD card raises security issues in case the card is placed in another system which may not obey the file permission rules and may be accessible openly. The sensitive files on the SD card can be accessed as follows: Figure 16.



```
Intent { act=android.intent.action.
NETWORK_SET_TIMEZONE flg=0x20000010 (has
extras) } extras {time-zone - (Asia/Calcutta)
} from known action and data

Intent { act=android.net.conn.
INET_CONDITION_ACTION flg=0x10000010 (has
extras) } extras {reason - (simLoaded)
networkInfo - (NetworkInfo: type: mobile[UMTS],
state: CONNECTED/CONNECTED, reason:
simLoaded, extra: internet, roaming: false, failover:
false, isAvailable: true)
extraInfo - (internet)
inetCondition - (0)
} from known action and data

Intent { act=android.media
```

Figure 15. *Intent Sniff*



```
# cd mnt/sdcard
cd mnt/sdcard
# ls
ls
LOST.DIR
logfile.log
log.txt
Evernote
androidlabs
# cd androidlabs
cd androidlabs
# ls
ls
1328165826506.html
# cat 1328165826506.html
cat 1328165826506.html
<!doctype html><title>Statement</title><h1>Statement</h1><p>Debit Account: 12345
6789<br/>Balance: 47.00</p><p>Credit Account: 987654321<br/>Balance: 1043.00</p>
#
```

Figure 16. *SD card Sensitive data*

Conclusions

Rapid diffusion of mobiles is alimenting the interest of ill-intentioned, cyber criminals and state-sponsored hackers are in fact intensifying their attacks against mobile platforms. Security in mobile devices must be considered a fundamental requirement. Mobile devices represent a technological appendix to our persons, and due this reason, it needs a high level of protection.

The correct approach must follow parallel paths; on the manufacturer's side, it is crucial that the mobile and application installed are designed considering all the possible cyber threats and evaluating with care the surface of exposure. On the other side, a user must be aware of the potentiality of their devices and the risks connected to cyber-attacks. The year 2013 presents itself full of challenges in mobile security, Android users will have to face a growing number of cyber threats of increasing complexity. The principal cyber threats will be launched by cyber crooks who want to steal sensitive information and intellectual property, but also cyber-espionage activities of governments and private actors have to be considered.

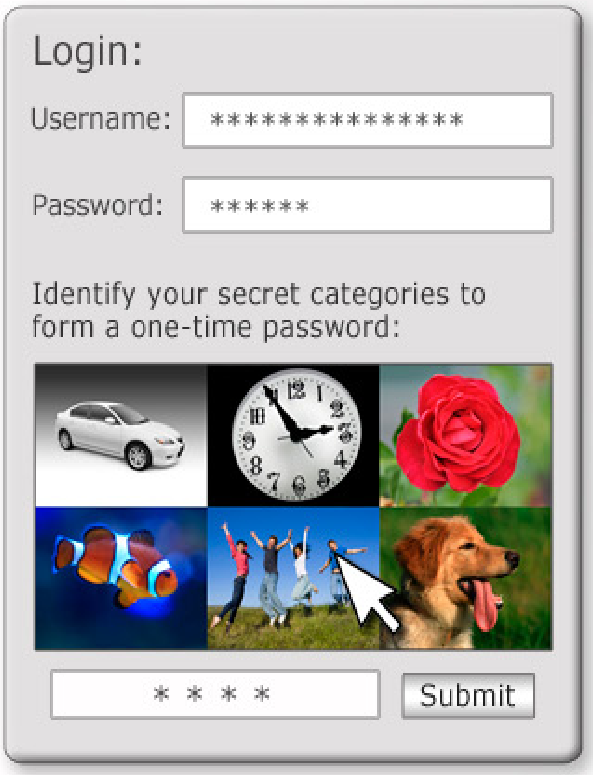
BHADRESHSINH GOHIL

Bhadreshsinh Gohil has a Master of Engineering in Computer Engineering – specialized in IT Systems and Network Security. He has been working in this industry for 2 years. He has also worked as a system administrator for 2 years where he has done his project into web security as a part of his master degree at C-DAC, Pune. He is also group member of null and other hacking group.
http://about.me/bhadu.gohil

Secure User Authentication

Image-Based Authentication
that Generates One-Time Passwords

- Strong user authentication that's easy to use.
- A flexible layer of authentication that can be inserted anywhere needed.
- Generates one-time passwords for strong security.
- Protect user accounts, prevent fraud and stolen credentials.
- For websites, mobile apps, or two-factor authentication.

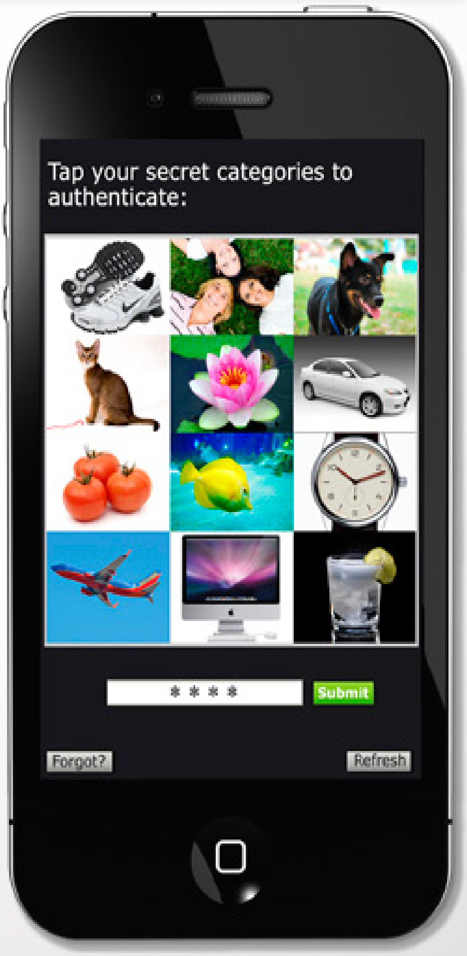


Login:

Username: ***************

Password: ******

Identify your secret categories to form a one-time password:



Tap your secret categories to authenticate:

Get It Now!

Visit www.ConfidentTechnologies.com to get APIs or start a free trial today.

Confident® TECHNOLOGIES

Android Hacking

Made Easy – What You Can Do To Limit Your Exposure

Android devices are extremely popular. From phones to tablets, e-readers, netbooks, smart watches and car computer out there. Over a half billion Android device users are out there with 1.3 million new users added every day [1]. Any technology that is in a lot of hands is a target for hackers. Why not?

When "you can make $10,000 a month for a basic effort at writing malware – you can get more when you distribute this malware to the contact lists and [build botnets]". [2] Worried yet? The statistics are alarming In 2012. Android accounted for 79% of all mobile malware, 96% in the last quarter alone according to F-Secure [3]. What's more we bring our own devices to work, school, everywhere we go, exposing not only our networks but other networks we might connect to. McAfee reports malware broke new records in 2012 with the number of new malware to reach 100 million for the year [4].

There are three types of Android users out there. Those that hack, those that will be hacked and those that will do something about it! Don't despair. Android malware (in the tens of thousands) pale in comparison to Windows malware (over 75 million). [5] Here are some things you can do to prevent your Android device from becoming just another statistic (Figure 2).

Trust Google

Google is well aware of what's going on with Android – the good, the bad and the ugly. Google has taken serious steps to prevent malware from af-

Figure 1. *Android image*

Figure 2. *Bouncer/Android image*

fecting your device. Meet the Bouncer. Hackers, you're next in line. It's time to give your best story about why you need to get into the club. This bouncer is good. It will automatically scan apps uploaded to Google Play (formerly Android Market), Google's application distribution platform for Android developers. The Bouncer isn't perfect. The Bouncer will wait and observe your behavior for a predictable period of time – around 5 minutes or so. If the hacker's app is patient and does not blink during the stare down from the Bouncer it can get in the club. Google is working on this obvious shortcoming (Figure 3).

Download from legitimate vendor sites only

Only download apps from reputable sites like Google Play. Google Play is similar to Apple's App Store. Beware of unofficial sites where hackers can masquerade original code with their own added "features." Google has standards in signing and releasing Android apps on Google Play. Here are some of them:

- APK (Android Package) file signatures are required for all Android developers. If the APK is not signed it will not install without a signature.
- Test and debugging tools are included with Android SDK.
- Self-signed certificates are also allowed to sign an APK. A self-signed certificate is ok for testing purposes. A certificate from a Certificate Authority (CA) is better if you want to a trusted cert.
- At release time developers must sign their APK with their private key. Private keys are generated locally and never shared.

This combination of file and private key signature allow for multiple factors of authenticity. Certificates add yet another layer of signature options (Figure 4).

Update automatically and often

Drippler makes your Android even better. Drippler is a free app you can download today from Google Play. Drippler will help you with tips and tricks specific to your Android device. It will automatically detect any software updates and upgrades your Android needs. Drippler will also keep track of any firmware updates. People love drippler because it provides helpful, customized and accurate Android news and tips to make your experience more relevant to your lifestyle. This may be considered a "soft" layer of security – automatic updates for Android and firmware. Its weakness is at the mercy of known vulnerabilities. What about zero-day vulnerabilities? We don't know what we don't know and vulnerabilities can propagate until discovered and patched. Even vulnerabilities that reach worldwide attention can go unpatched for *years*. Until we can get ahead of known vulnerabilities we need to be working on writing secure code in the first place. First to market is very big deal in just about every line of business still, developers have the responsibility of writing secure code by controlling input to only what is needed and nothing more for example a phone number or postcode has a specific number of digits so only allow input to only those digits. Secure code is the first, *and most important*, step in the process of any security program. The problem is developers aren't security experts and most security experts don't write a lot of code that makes it into a product or service. Remember first to market is everything when rolling out a new app. Look around Google Play for any app. What you'll find are pages and pages of similar ideas available in an app for free or for a nominal fee. Business decisions often overrule security. One reason is the time and cost of writing secure code can be seen as an inhibitor to the next release. What needs to happen is security needs to assign a dedicated person who works side by side with developers to ensure secure code is part of the process on day one of the project. Not at the end or in the middle of a project. Business, *for the sake of business*, should provide due diligence by ensuring developers receive training and certifications in writing secure code. One highly recommended certification is the Certified Secure Software Lifecycle Professional man-

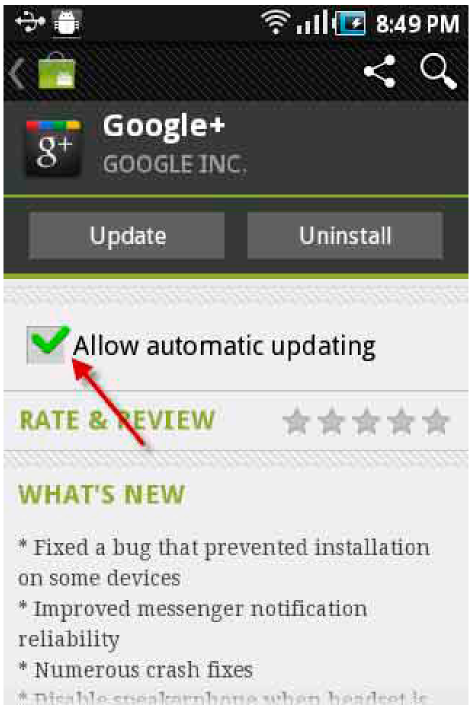
Figure 3. *Google Play image*

Figure 4. *Drippler image*

aged by International Information Systems Security Certification Consortium (ISC)². Organizations or individuals that implement a security program effectively, whether at home or at work, will realize security becomes an enabler and an insurance policy. If security is considered an unnecessary cost or waste of time then the organization (or individual) has already failed (Figure 5).

Don't grant unnecessary permissions

Many apps want to you to enable automatic updates or location services. Ask yourself if you really need a dictionary app, for example, to know your location. Probably not. Permissions can change over time. For example, when you upgrade to a newer version of the software or perhaps reinstall the same software. Generally speaking software vendors don't deliver strict permissions, with their product, regardless of how it is downloaded and installed. A slip of the finger during installation can result in answering, "yes" rather than "no" allowing for permissions you may not have really wanted. Slow down during new application installations to review your options. The permission may not be a configuration item you can change later. You might have to remove the app and reinstall to answer the question properly. One side effect of automatic updates and location services being enable is most people don't know if they should or shouldn't allow such actions. When in doubt decline any feature that automatically performs a software change to your Android device. There are ways to enable and disable some features as needed. It's not always easy to manually toggle on and off app permissions, especially if you have a lot of apps you use regularly. However, it is necessary to be vigilant today. We must take an active role in protecting our own privacy (Figure 6).



* Fixed a bug that prevented installation on some devices
* Improved messenger notification reliability
* Numerous crash fixes

Figure 5. *Google + automatic updating image*

Figure 6. *TrustGo image*

Install reputable, award-winning Anti-virus software for Android Many vendors like Sophos, Avast, F-Secure, Ikarus, Symantec, Lookout, McAfee and Zoner offer a free or affordable version of their products available for Android today [6]. According to AV-Test.org the number one Anti-virus product you can use for your Android v4.1.2 is TrustGo Mobile Security 1.3 [7]. It scored the highest overall for protection and usability. However, others closely followed like Antiy AVL v2.2 and Bitdefender Mobile Security v1.2. Installing award-winning, test-proven Anti-virus software can go a long way to further securing your Android device. Or so it would seem.

Palo Alto Networks has recently discovered an overwhelming majority of "unknown" malware was delivered via web browsing [8]. Over a period of three months Wildire Firewall found more than 26,000 samples of unknown files on data collected from over 1,000 of Palo Alto's enterprise customers. Over 90 percent of the malicious files were delivered via web browsing. This defies the well-known method of malware delivery via email. Malware delivery vectors are changing according to Symantec's White Paper. Cyber criminals are hiding malware in an iframe or obfuscated Javascript where it is invisible to the user browsing a website. [9] A good rule of thumb – be careful where you go on the web!

Maintain a smaller footprint

Delete apps you don't use. Apps are a lot of fun and easy to install. If you share your Android device with other family members or trusted friends your Android may have a lot of apps installed. If you don't use an app often enough you should remove it. On the battle field of cyber war smaller targets may often get overlooked for larger, easier targets. The state of affairs in the world today is we are all at risk for data loss, invasions of privacy and malicious software. The more we do to minimize our exposure the better we protect ourselves against unwanted incidents. Many people may not be overly concerned if anyone is able to discover where they go, what they do or sensitive information they may hold on their Android device. People may feel they have nothing to hide or protect when using their device. However, let's not volunteer our private or sensitive information. Let's not make it easy for a stranger to take what is our own. This just makes it easier for the cyber criminal to continue to take advantage of others (Figure 7).

Get Alerts

Knowing the latest attack vectors will help you realize trends and exposures. There are many organizations that track security incidents and the latest releases from popular vendors. You can sign up for free and start receiving alerts today. Not all alerts will apply to Androids specifically. Many alerts apply to Adobe and Microsoft. However, even the best developers and most trusted companies have flaws in their code. No software company is immune to security flaws. Keeping track and reacting to the latest vulnerabilities will help keep your Android device more secure. Closer investigation of alerts often leads to a patch or a work around (Figure 8).

US-CERT
UNITED STATES COMPUTER EMERGENCY READINESS TEAM
U.S. DEPARTMENT OF HOMELAND SECURITY

Figure 7. *US-Cert image*

Figure 8. *DroidWall image*

Install a Firewall

DroidWall is a Firewall for your Android device. Did you know you could restrict which apps can access the network from your Android? Yes, another layer of security you can add to your device. Installation is easy. Root is required to configure DroidWall. If you are familiar with Linux operating systems (of which Android is based on) then you will be familiar with "iptables" and the rules you can configure to allow or deny apps connectivity to the network. DroidWall users will enjoy the benefits of limiting apps to the network if they have a limited data plan. DroidWall also helps improve battery life. What if you don't want a firewall that does not require root privileges? Mobiwol claims to be the only non-root required firewall also available on Google Play. Mobiwol shares many of the same benefits as DroidWall and then some. Mobiwol will alert you when apps access the Internet giving you control and the knowledge of what apps are doing behind the scenes (Figure 9).

Encrypt Your Android Device

Google Play has myriad encryption apps to choose from. [10] Many for free or for a nominal fee. These encryption apps offer military-grade, strong encryption algorithms like AES, RC6, Blowfish, Serpent, Twofish and GOST. Most come with a standard 256 bit encryption algorithm. At this time it would take 50 supercomputers operating at 20 Peta-FLOPS an estimated 3×10^{51} years to discover the entire 256 bit key space. Encrypt any of your files, photos, contacts, passwords, messages, notes, text and even entire folders. Encryption should come standard with any native operating system. One of the very first things you should do when you get your Android device out of the box is to install and configure an encryption app.

Healthy Habits of An Android User

Now that you now know several ways to secure your Android device using software, let's look at what you can do to live a more secure lifestyle. Next topic, changing your habits to become even more secure. There is no, one piece of software that will solve all of your potential malware problems with your Android device. A more effective approach in addition to the previous section will make you more secure, physical security. This is where Android security takes a manual approach to disrupt, delay and deter further exposure. The more you make he following tips part of your Android lifestyle the better.

- Don't connect to just any wireless network or computer with your Android. Don't allow automatic connections to unknown networks.
- Power off when you are not using your Android.
- Randomize network usage. Don't stay connected to wireless if you aren't using it.
- Never root (aka RootKit) your Android. Never allow an app to run as root.
- Never leave your Android device on a table in a restaurant, halfway in your back pocket, or loosely held when in public places. When not in use keep the device out of sight.
- Password protect your Android. Change your password regularly.
- Configure your phone to be wiped clean or reset to factory default if too many unsuccessful attempts have been made to login. If you have kids you might reconsider this.
- Minimize. Only run the apps you absolutely need and use regularly.
- Don't allow others to shoulder surf to discover your login password.
- Purchase a case to protect and secure your phone.

In summary, there are many ways you can further protect your Android device from unnecessary exposure to malware (and other threats). Set aside some time in your busy schedule to harden your Android with the software and solutions mentioned here. Two themes were presented; using software solutions to secure your Android device and physical lifestyle choices you can make today to be more secure. Security is an individual responsibility that will collectively lead to a more secure world. Vigilance and due diligence are required to achieve a smaller target in today's highly connected and integrated Internet society.

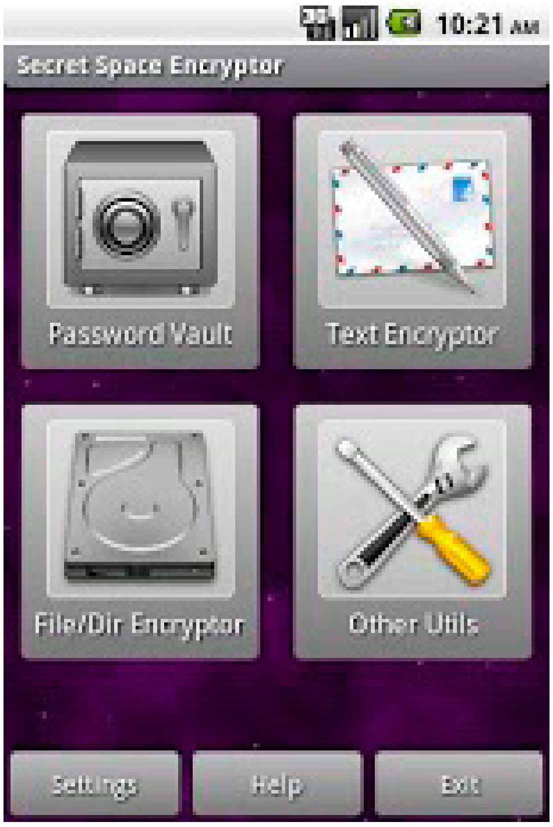


Figure 9. *Secret Space Encryptor image*

References

1. "Google: 500 million Android devices activated". September 12, 2012
2. Modular Android Malware Dev Kit To Be Released. August 3, 2012
3. Android Account for 79% Of All Mobile Malware in 2012, 96% In Q4 Alone, Says F-Secure. Thursday March 7th, 2013.
4. McAfee: Malware breaking records, again. September 5, 2012
5. Is Google Helpless To Stop The Scourge Of Android Malware, December 29, 2012
6. Best Anti-Malware Scanner For Android Devices. November 23, 2012
7. AV-Test Mobile Devices Android Most Recent Test Results. January 2013
8. New study finds malware variant skirting AV, mostly delivered via web. March 27, 2013
9. Symantec White Paper – Malware Security Report: Protecting Your Business, Customers and the Bottom Line. September 2011
10. Encryption Apps available on Google Play. April 3, 2013

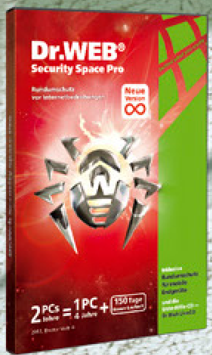
JOHN LEAR

John Lear, CISSP, has worked in IT for over 18 years as a system and security engineer and most recently as a DevOps Engineer. Ten of those years he was involved with building a security program from the ground up. He is a subject matter expert in the areas of hardening operating systems and applications. John is founder of Oomba Security LLC where he provides security as a service, automating compliance solutions, training and vulnerability management. His current project includes writing secure code in Ruby on Rails to scan and ensure system compliance. When he's not working he enjoys spending time with his family and biking.

Dr.WEB®
20 years

Dr.Web SpIDer is 8-legged!

New Package Installer
Improved Technology for Protection against yet-unknown Threats
Improved Parental Control
New Anti-Rootkit Module
Online Service Dr.Web Cloud
Free Upgrade to New Version
Unified User Interface
Better than ever: Dr.Web Security Space version 8.0! 30-day trial available from https://download.drweb.com/?lng=en



New Version 8.0

Dr.Web Security Space
and Dr.Web Antivirus for Windows

Get your free 60-day license under https://www.drweb.com/press/.
Your promo code: **Hakin9**

Doctor Web is a Russian anti-virus vendor with a software development record dating back to 1992.
www.drweb.com

Weak Wi-Fi Security,

Evil Hotspots and Pentesting with Android

In this article we will take a look at some of the most common security issues with Wi-Fi. We will see how a wireless card can be turned into a rogue Access Point using the Social Engineering Toolkit. And also take a look at the latest Android app that can turn your Android device into a pentesting platform.

Wireless networks and mobile Wi-Fi devices have saturated both the home front and business arena. The threats against Wi-Fi networks have been known for years, and though some effort has been made to lock down wireless networks, many are still wide open.

In this article we will look at a few common Wi-Fi security misconceptions. We will also see how a penetration tester (or unfortunately, hackers) could set up a fake Access Point (AP) using a simple wireless card and redirect network users, capture authentication credentials and possibly gain full remote access to the client.

Finally, we will look at the latest app for Android that allows you to turn your Wi-Fi smart phone or tablet into a pentesting tool. With it you can scan your network for open ports, check for vulnerabilities, perform exploits, Man-in-the-Middle (MitM) attacks and even sniff network traffic on both your Wi-Fi network and wired LAN.

So let's get to it!

(*As always, do not connect to any network or computer that you do not have permission to do so*)

Wireless Security Protocols

Though the news is getting out and Wireless manufacturers are configuring better security as the default for their equipment, there are still a large amount of wireless networks that are woefully under secured. One of the biggest things in securing your Wireless network is the Wireless Security Protocol. You have "None," which basically means that you are leaving the door wide open for anyone to access your network. "WEP" which has been cracked a long time ago and basically means that you locked the door, but left the key under the front mat with a big sign saying, "The key is under the Mat," WPA which is much better, and WPA2 is the latest and recommended security setting for your network. The following chart (Figure 1) was created from a recent local city wardrive.

As you can see, 13% of detected Wireless networks had no security set at all, and 29% more

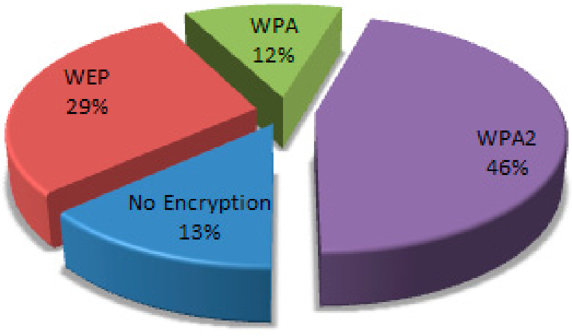


Figure 1. *Chart of Wi-Fi Networks Detected (Created using Kismet and Excel)*

were not much better using WEP. Interestingly enough a whopping 46% were using WPA2, which was actually kind of surprising. But in many cases, it seemed from the beacons captured that the AP was capable of WPA2, but clients were using the lower WPA.

WPA/WPA2 can still be cracked, so set a long complex passkey for them.

Let's take a look at some common wireless security misconceptions, and how a wireless card can be turned into a malicious access point.

Getting Started

First, you will need a copy of Backtrack 5 [1] and a Wireless card capable of entering monitoring mode. Many Wi-Fi adapters are capable of doing this, but some are not. If you are planning on purchasing one, do a little research first to determine if your Wi-Fi adapter will work in monitoring mode and with Backtrack 5.

For this article I used a TP-Link TL-WN722N USB adapter with antennae. The card was very affordable (less than $20 USD from Newegg.com), has great range and works great with Backtrack 5 both in Live CD mode (booting from a CD) and when running in a Virtual Machine.

Viewing Wireless Networks with Airmon-NG

First, let's take a look at how to view available wireless networks using Backtrack and Airmon-NG.

- Start Backtrack 5
- Open a command terminal and type in the command "ifconfig". You should see your wireless network card listed as wlan0 (or wlan1 if you have two). If the interface does not show up, try typing "*ifconfig wlan0 up*". Notice that the ifconfig command displays the physical MAC address of your card. This is a unique identifier hardwired into the card.
- Okay, now all we need to do is put the card in monitoring mode. To do this, just type: *airmon-ng start wlan0*
- A monitoring interface is created called "mon0"
- Now, just start Airodump-ng on mon0 by typing: *airodump-ng mon0*

The Airodump-ng program will start and you will see a list of all available wireless access points (APs) and also a list of clients that are attached (Figure 2).

Airodump-ng lists several pieces of information that are of interest. The first is the MAC address of the AP device. Next is the Power level. You also see the channel number that the AP is operating on, the number of packets sent and the encryption type. Lastly, the AP name is listed (not shown in figure).

From the figure above, you can see that the top AP is using "WPA2", which is the recommended encryption type. You can also see that the next two are not very secure. One is using "WEP", which was cracked a long time ago. And the last one is "Open", which means that there is no security set on the AP and anyone can connect to it.

As shown, there are no clients connected to any of the Access Points. If a client did connect, we would see the MAC address of both the client and the AP they connected to listed under the BSSID STATION section. Thus, you can see one of the inherent security flaws of Wi-Fi. Filtering clients by MAC address is not a very effective security strategy as it is trivial to view which clients are connected to which AP's by their physical address. All an attacker would have to do is view which addresses have connected and then spoof the address to bypass MAC filtering.

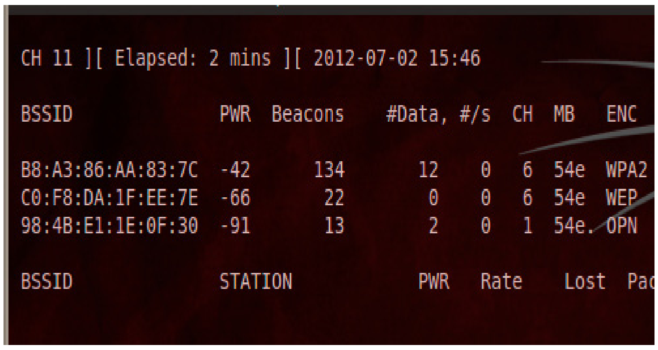
Viewing Wi-Fi Packets and Hidden APs in Wireshark

Okay, we have seen how to view which APs are available, now let's see how we can capture wireless packets and analyze them in the ever popular protocol analyzer Wireshark.

Simply place your Wi-Fi card in monitor mode like we did in the previous example, and then run Wireshark. Placing the card in monitor mode will allow us to see wireless management traffic like AP Beacons and Probes:

```
root@bt:~# airmon-ng start wlan0
root@bt:~# wireshark &
```

Wireshark will open, now all you need to do is select "mon0" from the interface list and you should instantly see a list of all the Wi-Fi Beacon traffic.



```
CH 11 ][ Elapsed: 2 mins ][ 2012-07-02 15:46

BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC

B8:A3:86:AA:83:7C  -42      134       12    0   6  54e  WPA2
C0:F8:DA:1F:EE:7E  -66       22        0    0   6  54e  WEP
98:4B:E1:1E:0F:30  -91       13        2    0   1  54e. OPN

BSSID              STATION            PWR   Rate    Lost  Pa
```

Figure 2. *Available Wi-Fi networks from Airodump-ng*

For example:

```
1  0.000000      Beacon   frame,   SN=3269,   FN=0,
   SSID=Broadcast
2  0.028565      Beacon   frame,   SN=3318,   FN=0,
   SSID=Skynet
```

Here you can see a capture from two separate APs. The second one is called "Skynet", but the first one is different. The SSID is "Broadcast," which tells us that the name for this AP is hidden. This is another ineffective technique used to secure wireless networks, and I will show you why.

If a client attempts to connect to this hidden AP, we automatically capture the SSID name in Wireshark as seen below:

```
93   6.623480   Probe Request, SN=0, FN=0,
                 SSID=YouCantSeeMe
99   7.122094   Probe Response, SN=843, FN=0
                 SSID=YouCantSeeMe
```

The AP name that did not show up in the Beacon frames becomes revealed to us as soon as a client attempts to connect. The client lists the hidden AP name in the probe request, in this case "YouCantSeeMe." And the AP echoes its hidden name back to the client in the Probe Response.
To stop the Wireshark capture, just use the "Stop Capture" button on the menu. You can then search, filter or save the results.

Turning a Wireless Card into an Access Point

One of the interesting features of wireless cards is that they can also act as an Access Point. This feature is of great interest to penetration testers, but unfortunately also to malicious users. You can create an AP using any SSID that you want. If you name your created AP the same as an existing one, the client cannot tell the difference and will connect to the nearest one, or the one with the strongest signal.

Once your card is in monitoring mode, you can turn it into an AP using the Airbase-ng command:

```
root@bt:~# airbase-ng -e "EvilAP" -c 6 mon0
```

This command creates an AP with the name "EvilAP", on channel 6 using the mon0 interface. This AP should now show up on any nearby Wi-Fi clients.

We have now turned our little unassuming wireless card into an "EvilAP". To complete the Dr. Jekyll to Mr. Hyde conversion, we also need to configure the Backtrack system to give out IP addresses to connecting clients (DHCP) and control what websites they can see (DNS spoofing). This allows you to take complete communication control between the internet and the client user. Though this is not hard to do, for this article we will look at the Social Engineering Toolkit (SET) that already has this capability built in and runs from an easy to use menu system.

Social Engineering Toolkit (SET) Wi-Fi Access Point Attacks

Let's take a quick look at how a pentester could use SET to test the security of a network. First you need to install dhcp3-server, at the console prompt type:

```
root@bt:~#  apt-get install dhcp3-server
```

(Note: *When you run the Wi-Fi attack, SET will tell you that you need to manually edit the dhcp3-server config file, and opens it for you. Just type in what SET tells you to enter and save it.*)

Now one other thing you may want to do is edit SET's config settings for the AP attack. SET will create an AP using the name "linksys" on channel 9. If you want to change these settings just edit the AP section of the set_config file. As you can see in the example below, I changed the AP SSID name to "Evil Wi-Fi" and left the default channel at 9:

```
/pentest/exploits/set/config/set_config

# THIS FEATURE WILL ATTEMPT TO TURN CREATE A
ROGUE ACCESS POINT AND REDIRECT VICTIMS BACK TO THE
# SET WEB SERVER WHEN ASSOCIATED. AIRBASE-NG and
                  DNSSPOOF.
ACCESS_POINT_SSID=Evil Wi-Fi
AIRBASE_NG_PATH=/usr/local/sbin/airbase-ng
DNSSPOOF_PATH=/usr/local/sbin/dnsspoof
#
# THIS WILL CONFIGURE THE DEFAULT CHANNEL THAT
THE WIRELESS ACCESS POINT ATTACK BROADCASTS ON
                  THROUGH WIFI
# COMMUNICATIONS.
AP_CHANNEL=9
#
```

To start the attack, run SET, select option 1, "Social-Engineering Attacks". Next select option 8, "Wireless Access Point Attack Vector".

Next you will see a screen with the text (Figure 3). Just start the wireless attack, and SET automatically sets your wireless card to be an access point, configures the AP to give out IP addresses in the range you specify and also directs all DNS traffic to your backtrack system.

Now that you are back at the terminal prompt, enter "99," and return to the main SET menu. From here you can select any of the other SET web attack vectors that you want to run automatically when a client connects. I covered these attacks extensively in a previous Hakin9 article, so I won't cover them in depth here. But let's look at one quickly.

For example, let's choose the Java applet attack and use the Gmail web template. Once it is running, anytime that someone connects to our Access Point, no matter what website that they try to surf to, they will be redirected to our SET server web page that looks like Gmail (Figure 4). The Java applet attack will also fire. If they allow the applet to run, we get a full remote shell.

This is only a simple example, but this demonstrates that if you can get users to connect to your SET AP, you can control where they can surf and what they see. Though this is pretty obvious, "Hey, every time I go to a different website, I end up at Gmail!" a malicious user would be a little more discrete. They could easily set up a fake AP that just harvests credentials without the re-direct. This would allow the victim to surf the web without interruption (and without suspicion) but it would record any usernames and passwords entered in the background, unseen to the user.

If the malicious user ran a program like *sslstrip* [2] in Backtrack along with the rouge AP, he could also capture any SSL encrypted data from the session. Sslstrip performs a man-in-the-middle type attack with secure https communications. Basically it creates a secure connection between the malicious AP and the target site, but communicates with the user's client PC in standard http communication. This effectively "strips" the SSL encryption out of the communication stream.

This could put an unsuspecting user's secure account information and credit card numbers at risk and is why one must be very cautious when using any "open" or "free" Wi-Fi hotspots!

Mobile Wi-Fi Attacks

Many think that they are immune from wireless attacks if they are using a Tablet or even a Smart Phone. But many, if not all Wi-Fi attacks that work against a PC still work against mobile devices. Man-in-the-middle, DNS and credential harvesting attacks all work regardless of what platform you are using. Mobile devices can even be used to test the security of your wired LAN.

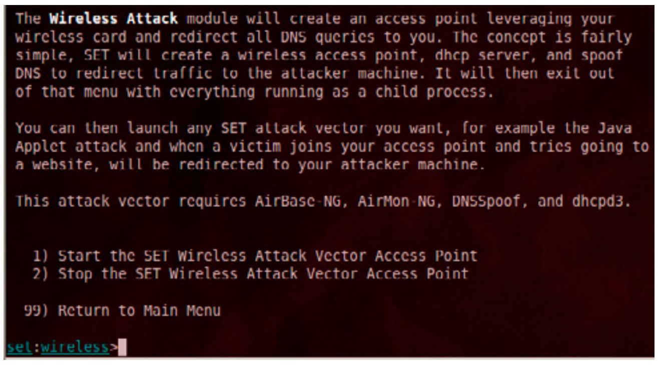
As a matter of fact, as a pentester one of the coolest mobile apps you can get for the mobile platform is zImperium's zAnti [3] – Android Network Toolkit. zAnti will scan for and displays any wireless networks in the area. You can then select the network you want and run several levels of nmap scanning against them, from a quick scan to an intrusive level scan. zAnti also allows you to perform man-in-the-middle attacks, network sniffing, exploits and the ability to create reports.

If you liked the previous version (called Anti) then you will love this update. zAnti seems to be smoother and easier to use than its predecessor. zAnti still comes with a token type credit system that allows you to access the more advanced features, but like the first one, you can still see the power of zAnti with the free version.

So how does it work?

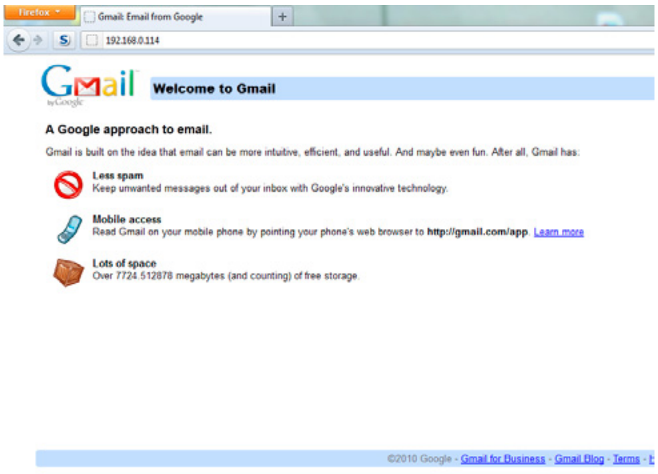
Once you start the App, you will be asked to login. Then zAnti does a quick scan of available Wi-Fi networks and asks which one you want to test. Just select the network and zAnti does a quick scan and shows all the available hosts on the network.

Found a target that looks interesting? Just select it and with a quick swipe of the finger and you



```
The Wireless Attack module will create an access point leveraging your
wireless card and redirect all DNS queries to you. The concept is fairly
simple, SET will create a wireless access point, dhcp server, and spoof
DNS to redirect traffic to the attacker machine. It will then exit out
of that menu with everything running as a child process.

You can then launch any SET attack vector you want, for example the Java
Applet attack and when a victim joins your access point and tries going to
a website, will be redirected to your attacker machine.

This attack vector requires AirBase-NG, AirMon-NG, DNSSpoof, and dhcpd3.

  1) Start the SET Wireless Attack Vector Access Point
  2) Stop the SET Wireless Attack Vector Access Point

 99) Return to Main Menu

set:wireless>
```

Figure 3. *SET Wireless Attack Menu*



Firefox | Gmail: Email from Google
192.168.0.114

Welcome to Gmail

A Google approach to email.

Gmail is built on the idea that email can be more intuitive, efficient, and useful. And maybe even fun. After all, Gmail has:

Less spam
Keep unwanted messages out of your inbox with Google's innovative technology.

Mobile access
Read Gmail on your mobile phone by pointing your phone's web browser to **http://gmail.com/app**. Learn more

Lots of space
Over 7724.512878 megabytes (and counting) of free storage.

©2010 Google - Gmail for Business - Gmail Blog - Terms -

Figure 4. *Client opens web browser and tries to surf the web*

reach the Action menu. From here you can perform several different attacks including sniffing and exploit attempts. Swipe again and you come to the Nmap menu where you have the option to run several levels of nmap based scanning to attempt to detect OS version and service identification. Swipe once more and you will come to a comment page where you can write notes about the target.

In a test, I ran zAnti on my 7" Android Tablet. Within a few seconds I had a complete list of all the machines on my network. Selecting one of my Windows 7 systems from the menu I performed a deeper nmap scan. The scan found no open ports, and it could not provide much information about the client. But by switching to the Action menu (Figure 5) I choose the "sniffer" option.

And within seconds I was viewing a list of all the webpages that my Windows 7 wired client was visiting, remotely on my droid tablet! (Figure 6)

Obviously some type of ARP (Address Resolution Protocol) cache poisoning was going on here. A quick look at the Windows 7 client's ARP Table showed that zAnti successfully performed a man-in-the-middle attack on the client. And sure enough, the attacking machine switched its MAC address for the client gateway.

Here is a look at the ARP table before running the sniffer program:

```
C:\>arp -a
Interface: 192.168.0.111 --- 0xa
Internet Address      Physical Address      Type
192.168.0.1           b8-a4-86-aa-43-6d     dynamic
```

And after:

```
C:\>arp -a
Interface: 192.168.0.111 --- 0xa
Internet Address      Physical Address      Type
192.168.0.1           00-48-92-9c-da-ec     dynamic
```

Notice the change in the Physical MAC address for the gateway machine. Now my wired client will send all of its network traffic to the Droid thinking that it is the default gateway. The Droid will then analyze the data and forward it to the gateway. And the gateway will send all of the return traffic for my client to the Droid which will then return it to the client.

This effectively put the wireless Droid in between my router and the wired Windows 7 Client so it could sniff all the network traffic. But the fun does not stop there.

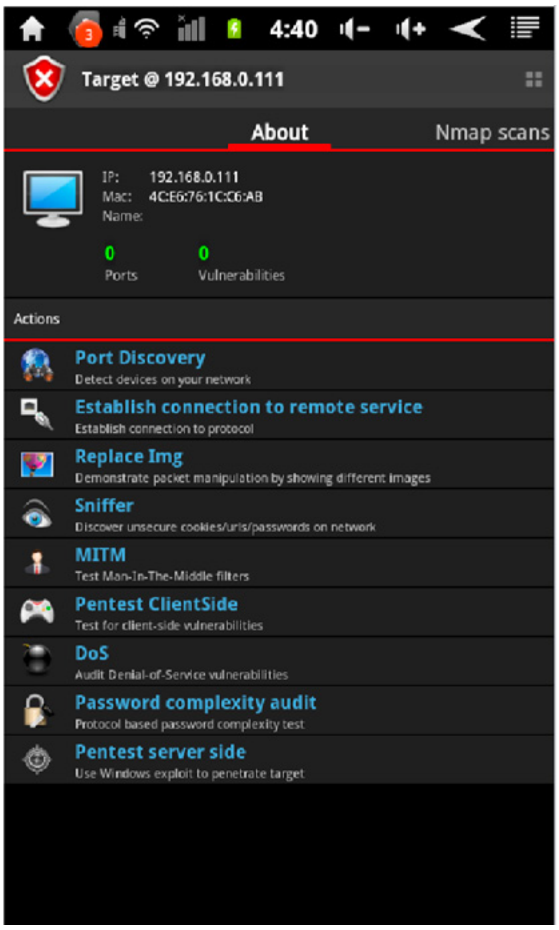

4:40
Target @ 192.168.0.111
About
Nmap scans
IP: 192.168.0.111
Mac: 4C:E6:76:1C:C6:AB
Name:
0 Ports
0 Vulnerabilities
Actions
Port Discovery
Detect devices on your network
Establish connection to remote service
Establish connection to protocol
Replace Img
Demonstrate packet manipulation by showing different images
Sniffer
Discover unsecure cookies/urls/passwords on network
MITM
Test Man-In-The-Middle filters
Pentest ClientSide
Test for client-side vulnerabilities
DoS
Audit Denial-of-Service vulnerabilities
Password complexity audit
Protocol based password complexity test
Pentest server side
Use Windows exploit to penetrate target

Figure 5. *Individual client attack options in zAnti*

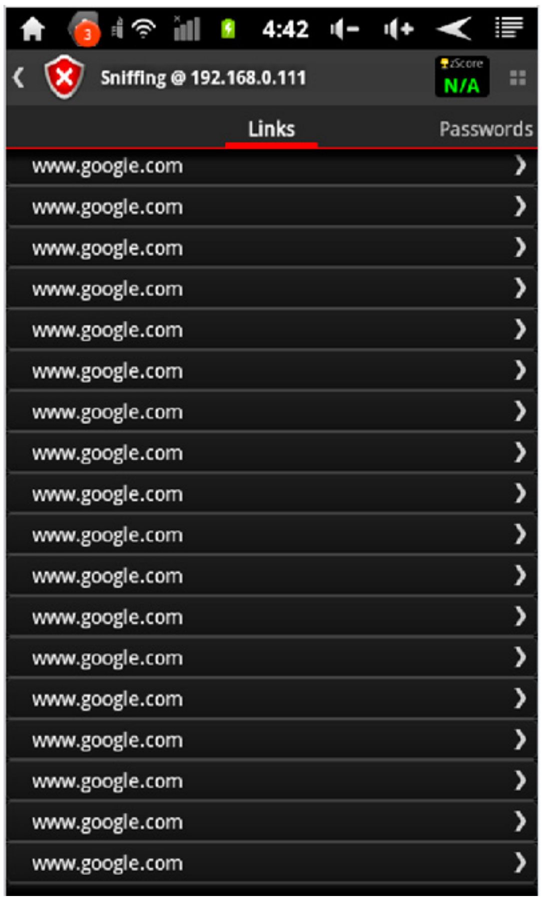

4:42
Sniffing @ 192.168.0.111
zScore N/A
Links
Passwords
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com
www.google.com

Figure 6. *Links to Google searches performed by wired LAN client as viewed on Wi-Fi Droid*

References
[1] *http://www.backtrack-linux.org/downloads/*
[2] *http://www.thoughtcrime.org/software/sslstrip/*
[3] *http://www.zantiapp.com/anti.html*

You can sniff traffic from any device connected to the same network. From my Android tablet I was even able to monitor the internet traffic of my iPad!

Conclusion

As we have shown there are many unsecured wireless routers out there and it is very easy to circumvent some of the common security measures that are implemented. It is also very simple to create a rogue "*Free Wi-Fi Hotspot*", intercept the wireless traffic, and control what a surfer can see in his browser and where he can surf.

The best defense against Wi-Fi attacks is to secure your router! Do not use open or WEP security. One of the main defenses your network has is your firewall; if you allow people inside your firewall you can open yourself up to ARP MitM attacks, packet sniffing and other attacks. Unfortunately, many corporate users do not understand this and will take their business laptops from a very secured environment at work to a very unsecured Wi-Fi network at home.

Be cautious of free Wi-Fi. Don't do online banking or shopping while using public Wi-Fi. Make sure your operating system is using a firewall and preferably internet security software. If your security software monitors your ARP table, that is even better! Use common sense, if you are working on sensitive information, do it at home not at the local coffee shop that offers free Wi-Fi, even if their cinnamon rolls are the best in the world. It is just not worth the risk!

For more information, check out Vivek Ramachandran's excellent book, "*Backtrack 5 Wireless Penetration Testing Beginner's Guide*." Also, David Kennedy's (creator of SET) book, "*Metasploit: The Penetration Tester's Guide*" is an excellent reference on Backtrack 5, SET and the Metasploit Framework.

DANIEL DIETERLE
Daniel Dieterle has 20 years of IT experience and has provided various levels of IT support to numerous companies from small businesses to large corporations. He enjoys computer security topics, is the author of the CyberArms Computer Security Blog (cyberarms.wordpress.com), and a guest author on a top infosec website. Dan was also a technical editor and reviewer for Vivek's book, "Backtrack 5 Wireless Penetration Testing Beginner's Guide".

Build Secure Android

Applications with ITTIA DB SQL

With Android's worldwide success, market dominance and the availability of inexpensive devices, it is easier than ever to deploy a distributed network of data-driven mobile software. With the rise of smart devices and similar mobile platforms for Android, anyone can own a general-purpose computing device that is capable of storing large amounts of data and running sophisticated applications on Android.

Business applications often deal with confidential data, process transactions, and log information for auditing purposes. When developing a mobile, distributed application it is important to not only protect confidential information, but also to prevent tampering and destruction of important data.

Android dominates the worldwide smart devices. Software developers build applications for these devices with a Java API, and hundreds of thousands of applications have already been created. Android uses a unique Activity model to manage interaction between applications and the user. Processes are started automatically, either to perform a task requested by the user, to provide data to an Activity, or to complete a background task.

This article explores the risks associated with handling critical data on Android devices, including the importance of security, performance tuning, scalability, data distribution and synchronization with back-end enterprise RDBMS technologies.

Protecting Data: Android Security Considerations

Android is designed to secure individual applications, using Intents to access Activities in other applications rather than sharing files or library functions directly. As long as data remains on the device in an application's private storage area, Android will protect it from unauthorized access. However, storing data on removable media, exposing data providers to other applications, and communicating with remote systems each introduce new security risks.

Most Android devices include an SD card reader. SD cards are shared between applications, and can be easily removed and replaced. This makes them an excellent tool for backing up critical data, but also an important security concern. Confidential data can be easily copied to another system from a removable card.

Data is often synchronized with back-end systems that service many different devices and users. Providing an Android application access to such a system introduces further risk, beyond the risk to the device itself. At best, an eavesdropper on the network might capture confidential data sent to or from the device. At worst, an attacker might gain full access to the back-end database if it is left open for anonymous access.

Each device usually only needs access to a small subset of the data available in the back-end system. That is, back-end data must be fragmented amongst the devices and filtered to protect private data. This introduces another security concern: a device should only have access to read or modify certain information in the back-end system, and should not be able to imitate a different device.

When a record is shared between several devices, a conflict will occur if more than one device modifies that record between synchronizations. A clever attacker with access to one device might exploit this scenario to overwrite important data. Conflict resolution policies must be enforced in the back-end to protect the consistency of the data.

Securing Android Applications with ITTIA DB SQL

When designing a mission-critical Android application, developers of business intelligence solutions should consider data security as one of the top priorities in the software architecture, whether to protect confidential data, to prevent tampering, or both. To that end, Android applications need a secure solution for on-device storage, data distribution, and synchronization with back-end systems.

ITTIA DB SQL is a secure embedded database technology for Android that supports data distribution through replication and RDMBS synchronization. The encryption and authentication features of ITTIA DB SQL enables Android applications to achieve the greatest level of data security by encrypting storage media, network communications, and password exchanges.

Exceptional Authentication Mechanism

Whether connecting over TCP/IP, shared memory, opening a database file directly, or through Secure Sockets Layer (SSL) protocol, ITTIA DB SQL uses simple database-wide passwords to prevent unauthorized access. The SSL protocol offers further protection from eavesdropping, the unauthorized interception of communications, and uses certificates to prevent session highjacking. Moreover, database passwords are never transmitted over the network, whether or not the communications channel is encrypted.

Secure Encryption Solution for Enterprise Businesses

The comprehensive encryption features of ITTIA DB SQL provide Android developers with the opportunity of configuring remote client applications to use SSL when connecting to ITTIA DB on any desired operating system. This secures communication path for data distribution and transfer of client/server applications that are susceptible to attacks all the way from storage media to the client.

Applications can use storage encryption to protect data on internal and removable media from unauthorized access. The Advanced Encryption Standard (AES) is a specification for the encryption of data that is commonly adopted by the U.S. government and others across the world. ITTIA DB SQL storage encryption provides support for the AES-128 and AES-256 block ciphers.

Data Distribution and Database Design Considerations on Android

Whether you are developing on Android or another platform, every distributed application has different requirements for the secure sharing of data. A few applications may require all data to be distributed across all devices, with changes shared continuously, but most applications do not have the bandwidth or capacity for such an approach. Developers must make several choices about what information should be shared and when.

Distributed Databases

A distributed database is a collection of database sites that share information in some way.

There are two types of distributed databases:

- Homogenous Distributed DBMS. Every site runs the same kind of DBMS software. The same application code and data files can be used at any site.
- Heterogeneous Distributed DBMS. Each site can run a different kind of DBMS software, possibly from different vendors. Application code for one site may not run at other sites, and database files cannot be directly copied between sites.

Though homogenous distributed databases are easier to develop, maintain, and secure, heterogeneous distributed databases are common due to the constraints of device hardware, which is often incapable of running the same database management systems that are often used on back-end servers. Most back-end database software will not run on Android.

Data Fragmentation

In a distributed DBMS, tables are stored across several sites, such as Android devices and one or more back-end servers. If each site serves its own branch of an organization, the whole relation should be partitioned over the sites so that each of them stores only relevant parts of data.

There are two kinds of data fragmentation:

- Horizontal fragmentation. A horizontal fragment is a relation that contains a subset of rows from the original relation.

- Vertical fragmentation. A vertical fragment is a relation that contains a subset of columns from the original relation.

Replication

Replication is a technique in which two or more sites maintain copies of a table fragment. Replication provides a mechanism for duplication of data to increase availability, improve performance, or simplify execution of queries.

Two types of replication are available, synchronous and asynchronous, which differ in how changes to the data are maintained.

Synchronous Replication

Synchronous replication ensures that the distributed database as a whole adheres to the ACID properties: atomicity, consistency, isolation, and durability. From the application's perspective, the data is always consistent. Any query on the shared relation fragment will yield the same result regardless of which site it is issued to.

Synchronous replication accomplishes this by propagating changes to other sites before a transaction commit is completed, ensuring that all sites commit the changes in an atomic way. This is known as read-one write-all database access. If any site cannot commit the transaction, the changes are not applied at any site and the commit operation will fail.

However, data propagation, locking, and coordination of the commits have a high performance cost. A non-distributed database typically has better write performance than a distributed database using synchronous replication. For some applications, it is feasible to mitigate this cost by using in-memory storage and relying on redundancy rather than persistent media to preserve data over time. For other applications, synchronous replication is undesirable.

Asynchronous Replication

To overcome this issue, many database vendors provide asynchronous replication. Asynchronous replication relaxes the ACID properties, allowing the data to be inconsistent between sites at certain times. Rather than propagating changes before commit, they are collected by the DBMS and applied to other sites at a later time.

Under this model, some transactions become visible to other transactions only at a later time. Nevertheless, this is a practical limitation for many applications that significantly increases system throughput.

Asynchronous replication is typically used in one of two ways:

- Master/slave replication (single master replication). In this scheme, each fragment can be updated at a single site only. This site is called the master for that fragment. So while the master site maintains the authoritative copy of a shared relation, the slaves each maintain read-only copies of the fragment.
- Peer-to-peer replication (multiple master replication). In this scheme, some fragments may be updated at more than one site. When these changes are propagated to the other sites, change conflicts can occur and a conflict resolution strategy should be applied in order to merge changes correctly.

Asynchronous replication is performed in two phases: collecting changes from each site and applying them to all other sites.

The collection phase usually occurs during the transaction that modifies the database. The system records each change in some way, so that a complete set of changes can be generated in a reasonable time.

The application phase retrieves the changes from the source site and applies them to a destination site. A conflict detection process is used on each change, and any conflicts are resolved either by refusing the change or applying it, often with further modification. While conflict detection is handled by the DBMS in a general way, conflict resolution requires insight into the application logic.

A Secure Database Solution for Android: ITTIA DB SQL

ITTIA DB SQL is an embedded relational database that is designed to run efficiently within the constraints of mobile devices and other embedded systems. With support for both heterogeneous and homogeneous distributed databases, ITTIA DB SQL is able to both share data between Android devices and communicate with back-end RDBMS products, such as Oracle, MS SQL Server, MySQL, and more.

A number of features in ITTIA DB SQL aid application developers in building secure, distributed and high-availability systems. The following features are supported for homogeneous distributed databases only:

- Prepared transactions
- Distributed transactions
- Table snapshots
- Synchronous commit
- Replication on commit

By connecting to an ITTIA DB Sync server running on a back-end system, ITTIA DB SQL for Android can also distribute data with a number of RDBMS products. The following features are also supported for both homogeneous and heterogeneous distributed databases:

- Single-master replication on demand
- Multiple master replication on demand

Replication on Commit

ITTIA DB SQL can perform synchronous replication on designated tables, automatically applying changes in these tables to database files on other Android devices. When a transaction is committed, changes are sent to other participates using a distributed commit over all participating peers. As a result, committing a transaction will only succeed if all replication peers accept the changes.

Only homogeneous distributed databases can use synchronous replication with ITTIA DB SQL.

Replication on Demand

ITTIA DB SQL for Android supports both single-master and multiple-master replication modes in both homogenous and heterogeneous environments. These cases all have similar behavior, but differ in a few subtle but important ways.

When ad-hoc replication is enabled for a table in an ITTIA DB SQL database, subsequent changes are recorded automatically in the database journal. Journal records are retained, using log rotation, until the changes are replicated to each configured peer. Change events are propagated to each peer on demand and applied to the peer's database.

The back-end RDBMS is handled in a slightly different way: The replicated table has three triggers which catch the INSERT/UPDATE/DELETE events in a shadow table, which maintains one state row per each row in the original table. These two tables are joined using a configured primary key. The shadow table allows recovering the state of each row and finding out the set of changed rows in a reasonable time.

Conflict Detection

When replication is done between two Android devices, conflicts are detected using either row comparison or STAMP columns. Each change event records the row column values before the change is applied and this 'before' image is compared with the row found in the target database.

Another conflict detection option is to use a STAMP column. In this mode the application should designate one 64-bit integer column for this purpose and provide the column name as a part of the replication configuration.

The stamp column is filled during modification. It traces the state of the row owner (original peer address) and the row version. Conflict detection is done just by comparing the row owners. When the owner peer address is different, a conflict is detected. That is a rather simplified schema and is not capable of handling the full range of row conflicts as in the case of row comparison, but it is still practical, especially for single-master mode when conflicts occur when merging data from several Android devices into a single table.

Conflict Resolution

ITTIA DB supports policy-based conflict resolution. For each data event a policy can be defined to limit event processing. This policy consists of the following components:

- Disable/enable operation. Each operation can be disabled and consequently any changes of that type will be rejected. For example, it might be desirable to prohibit row deletion in the archive table.
- Exclusive operation use. An INSERT or DELETE operation could be treated differently depending on the row existence. For example, when an UPDATE event is applied and no such row exists locally, an INSERT statement could be generated instead and, vice versa, an INSERT could be transformed into an UPDATE when the row already exists. The exclusive operation option prohibits such a transformation.
- Acceptable. accept always, reject always, or peer priority. When peer priority is configured the peer's priority is compared with the local one and if the remote priority is higher then the change is applied.

When the ITTIA DB Sync server detects a conflict between data from an Android device and a back-end RDBMS, conflict resolution is accomplished by calling user supplied functions.

Fragmentation

Vertical fragmentation is done using the common column subset, matching columns by name.

Horizontal fragmentation is done in two different ways – using either the peer address or, for a back-end RDBMS, custom filters.

Each table can be configured with a PEER address column. When such a column is configured it is assumed to carry either the address of a peer to exchange the row with or the address of a group of peers. The peer address column is assumed to have positive values, while the group address column has negative values. The group address facility is only available to the ITTIA DB Sync server. This fragment can be limited further using a configurable send-filter. A send-filter is a plain SQL boolean expression that is merged into the WHERE clause.

Android Database API

The Android API for ITTIA DB SQL implements the same interfaces as Android's built-in database. As a result, migrating an Android application to ITTIA DB SQL is a straightforward process that mostly requires replacing the names of packages and classes. After migration, Android applications have access to all of ITTIA DB SQL's features, including shared database access, high-performance memory storage, and replication.

ITTIA DB SQL also uses static typing, where type information is stored in the database schema as part of a table's description. Each column can contain only a specific type of data. This ensures that type mismatch errors are identified early, when there is the best chance to successfully fix the mistake. This is important when the database is shared between applications that are developed separately, as the database schema forms a contract by which all parties must abide.

The Android database API can execute SQL queries, or generate SQL queries automatically for many common operations. ITTIA DB SQL also supports SQL queries and query generation, but also provides direct access to tables and indexes with low-level table cursors. Table cursors have lower overhead than SQL queries and allow modifications to be made directly while browsing a table, without constructing an update query. In many cases, an application can use table cursors to improve performance.

Scalable Flash Storage

ITTIA DB SQL provides safe, scalable on-disk file storage with row-level locking. Transactions provide full ACID protection, guaranteeing atomicity, consistency, isolation, and durability, with lower overhead.

When ITTIA DB SQL commits a transaction, only the write-ahead log must be written immediately because it contains a copy of all the changes made in the transaction. Log entries are written sequentially and only contain information about changes, so fewer pages are written to disk. Other modified pages are written to disk later, after accumulating changes from many transactions. Under high load, ITTIA DB SQL can significantly reduce the amount of write activity compared to Android's built-in database, both boosting performance and reducing wear on flash storage media.

ITTIA DB SQL uses a less restrictive locking technique: row-level locking with isolation levels. The database automatically tracks all rows that are read or modified in a transaction. At the highest level of isolation, known as "serializable," rows are locked in such a way as to prevent all possible conflicts. And for most simple transactions, the isolation level can be reduced to minimize locking even further.

This ensures that a transaction is only blocked when it would create a conflict with another transaction already in progress. In addition, an entire table can be locked manually.

Robust Shared Access

ITTIA DB SQL also supports concurrent shared access, either directly opening the same database file in multiple threads, or using a lightweight server to broker connections. This allows data to be easily shared between:

- Threads in an Activity
- Activities in an app process
- Applications on a device
- Devices on a network

Each connection has its own transaction, so that independent tasks do not interfere with each other in unexpected ways.

The lightweight data server used to negotiate shared access can be run directly from within an Android application. No code changes are required to access a database file on the same device and the application can also open database files remotely over a TCP/IP network.

Secure Encryption

Android devices that store sensitive information must use encryption and authentication to prevent malicious access. ITTIA DB SQL supports file storage encryption that protects data on internal and removable media from theft. When replication exchanges data with a back-end system, the connection can be protected by SSL/TLS encryption and SCRAM authentication. This prevents eavesdropping, the unauthorized interception of communication, and uses certificates to prevent session highjacking.

Conclusion

Android is an important platform for many new applications and the data management challenges on Android devices will only grow as tablets and other powerful devices continue to be developed. Ensuring that data is secured both on the device and when communicating with other devices and systems requires careful consideration and planning.

ITTIA DB SQL provides a replication environment that will improve the system, with respect to:

- Security
- High Availability
- Reliability
- Modularity

Android applications can achieve great mobility, by delaying replication until a stable connection is available and fragmenting data to reduce transmission cost. Applications also gain significant interoperability and scalability, for use in data mining and data warehousing after aggregating data through the ITTIA Sync Server.

In this article we have explored how ITTIA DB SQL will provide capabilities on Android to address the challenges of security, sharing data, synchronizing data with back-end RDBMS software, high throughput, and large data storage.

SASAN MONTASERI

Sasan Montaseri is the founder of ITTIA, a company focused on data management software solutions for embedded systems and intelligent devices. Sasan has worked directly with customers in various vertical markets to define ITTIA's database product roadmap and provide solutions to common data management problems faced by those customers. Under his guidance, ITTIA has grown from its inception to offer a data management solution to prestigious multinational customers throughout North America, Europe and Asia.

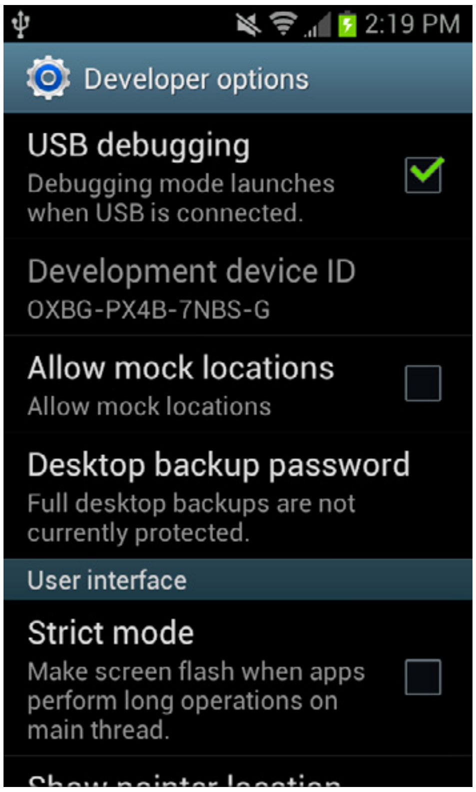
Decompiling Android Workshop

Due to the design of the Java Virtual Machine (JVM), it is relatively easy to reverse-engineer Java code from both Java JAR and class files. While this hasn't been an issue in the past (since most Java files are hidden from view on the web server), it is an issue on Android phones where the client-side Android APK files are easily obtained and just as easy to reverse-engineer or decompile back into Java code.

And if you have access to the code then you also have access to any of the API keys, usernames and passwords or any other information that the developer has stored in the original code. We're going to look at how to recover that static information in this article as well as some of the techniques for looking at information that was stored at runtime.

The first step in reverse engineering an APK is to get a hold of one. There are a number of different ways to do this; the easiest way is to use your favorite file manager such as Astro File Manager and backup your APK to an SDCard where it can be then transferred to a PC or Mac. Or if it's a relatively popular app you can usually find some version of the APK on forums such as *http://xda-developers.com* where APKs are often shared.

Personally, I prefer to use the adb command or android debug bridge tool that comes with the Android Developer Toolkit as part of the Android SDK. Adb allows you to pull copy of the APK off the phone onto your PC for further analysis.



USB debugging ✓
Debugging mode launches when USB is connected.

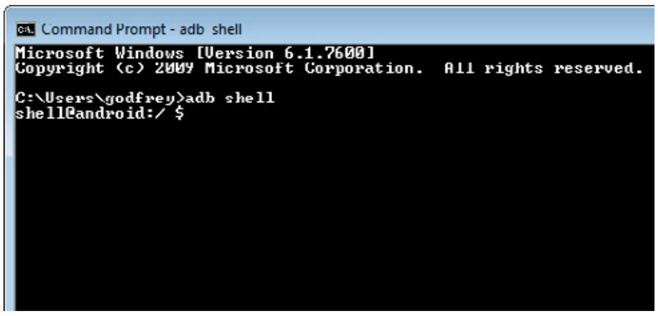
Development device ID
OXBG-PX4B-7NBS-G

Allow mock locations ☐
Allow mock locations

Desktop backup password
Full desktop backups are not currently protected.

Strict mode ☐
Make screen flash when apps perform long operations on main thread.

Figure 1. *Turn on USB debugging on the device*



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\godfrey>adb shell
shell@android:/ $
```

Figure 2. *Opening up a Unix shell on your Android phone*

To download an APK from your phone, first turn on the USB debugging option under developer options and connect the USB, see Figure 1.

Once the device has been connected, open up a command prompt or terminal and type *adb shell* to connect to the device and open up a Unix shell on the device, see Figure 2.

The Android system runs on top of a Linux kernel, so assuming you have the rights you can navigate up and down the Linux tree to see any and all directories on the phone or tablet. However to find your APKs which live in the `/data/app` and `/data/app-private` folders from the Linux shell you will have to root your phone.

Thankfully, we don't need to root the phone to use the USB connection to pull most APKs off the phone as they naming convention and location follow the same basic rules. If we take a look at the WordPress for Android app in Google Play, see Figure 3, you can see the id of the APK in the URL is org.wordpress.android which tells us that the APK name and location will be `/data/app/org.wordpress.android-1.apk`. While I'm not 100% sure that the APK will always end in -1.apk, it currently seems to be a valid pattern. There are some paid apps that are stored in the `/data/app-private` folder so you may have to try there too.

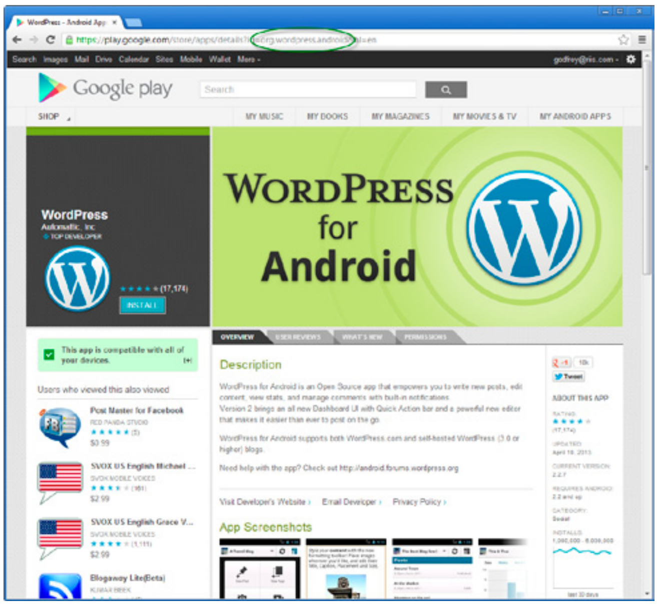
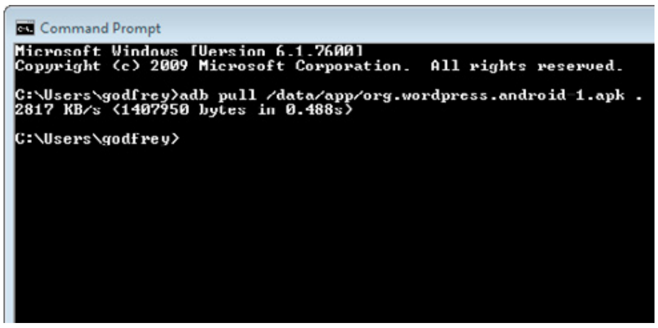


Figure 3. *Identifying the APK name*



```
Command Prompt
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\godfrey>adb pull /data/app/org.wordpress.android-1.apk .
2817 KB/s (1407950 bytes in 0.488s)

C:\Users\godfrey>
```

Figure 4. *Using the adb pull command*

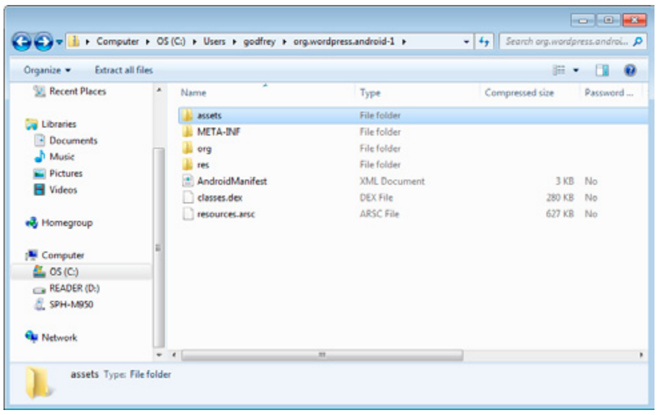
The WordPress app is a great app for our purposes as it's open source so we can see just how close the original code is to the decompiled code. Now we can use the same adb command that we saw earlier but this time with the pull option to get a copy of the APK off the device and onto the PC, see Figure 4.

Before we go any further let's take a look at the APK which is in a zip format. Copy the org.wordpress.android-1.apk to org.wordpress.android-1.zip and unzip it. We can now see the structure of an APK in Figure 5.

The assets folder has all the HTML and CSS. The AndroidManifest file and res(ources) folder should be familiar to any Android developers. However, it's the classes.dex file that we're really interested in as that's where the Java code gets compiled.

It's worth mentioning that if you write your app in HTML5/CSS and use something like PhoneGap to convert it into an Android app then your code is going to be visible too. When an Android app is unzipped all the HTML, CSS and JavaScript is in clear text in the assets folder unless you use a JavaScript obfuscator such as or Google Closure [1].

When your compile your Android Java project, it first gets compiled into a series of Java class files before these get converted into a single classes.dex file in a Dalvik Executable format using the dx command, which comes with the Android SDK, see Figure 6. This classes.dex file is interpreted by the DVM or Dalvik Virtual Machine when you run the app on your Android phone or tablet.



Name	Type	Compressed size	Password ...
assets	File folder		
META-INF	File folder		
org	File folder		
res	File folder		
AndroidManifest	XML Document	3 KB	No
classes.dex	DEX File	280 KB	No
resources.arsc	ARSC File	627 KB	No

assets Type: File folder

Figure 5. *Inside an APK file*

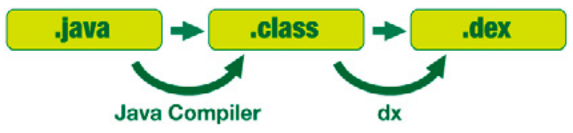


Figure 6. *Android compilation process*

Before we decompile classes.dex lets disassemble it first to see what we can find. There are a couple of ways of doing this, we can look at it in a hexadecimal editor or we can use a tool called baksmali [2].

Looking at the file in a hexadecimal editor probably isn't going to reveal much information unless we know the structure of the file. Unlike the Blackberry a classes.dex specification is available [3] which describes the layout and content of the classes.dex file in all its glorious detail. We can see the overall structure of the file in Figure 7.

If we were to disassemble the classes.dex by hand – which I wouldn't recommend – you would first have to reference the classes.dex file structure [3] to see it's we can find the opcode instructions that gets interpreted by the DVM, The specification tells us that the instructions or insns are in the code_item in the data section.

There is also a hexadecimal editor called 010 [4] which comes with a DEX template [5] that makes it very easy to navigate the classes.dex file if you are so inclined, see Figure 8 for a screenshot of the insns section of the file where the machine instructions or opcodes live. The opcode instructions specification is also available [6].

Baksmali makes disassembly much more straightforward and although it can be used on its own, it is often used as part of the apktool [7] suite of tools which has the added benefit of converting the AndroidManifest.xml file back into a readable file using AXMLPrinter2 [8], see Figure 9.

Baksmali strips out most of the noise and displays all the information in a much more readable smali text format. It is also common to make small changes to a smali file and then using the Smali assembly tool reassemble the classes.dex file. A simple smali file from the wordpress APK is shown in Figure 10.

While there are plenty of Java decompilers there aren't as yet any publically available classes.dex decompilers. However there is a tool to convert classes.dex back into a java jar files called dex2jar [9], which makes it possible to decompile an APK. Dex2Jar reverses the dx conversion from classes.

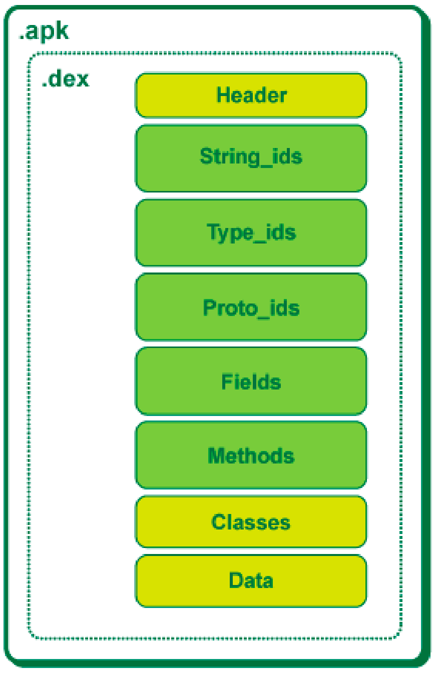


Figure 7. *classes.dex structure*

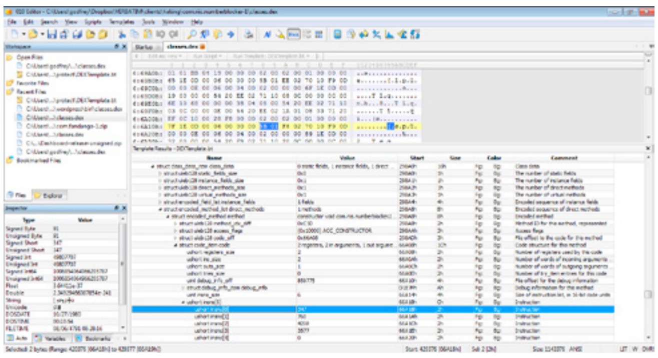
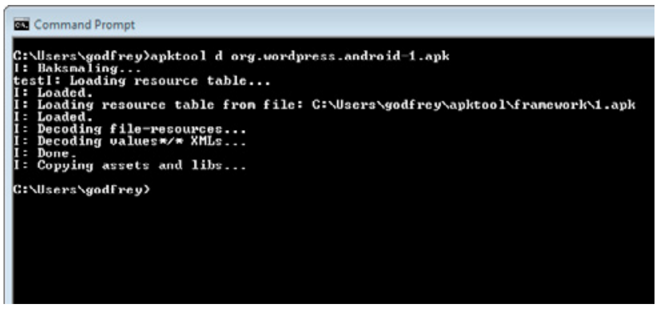
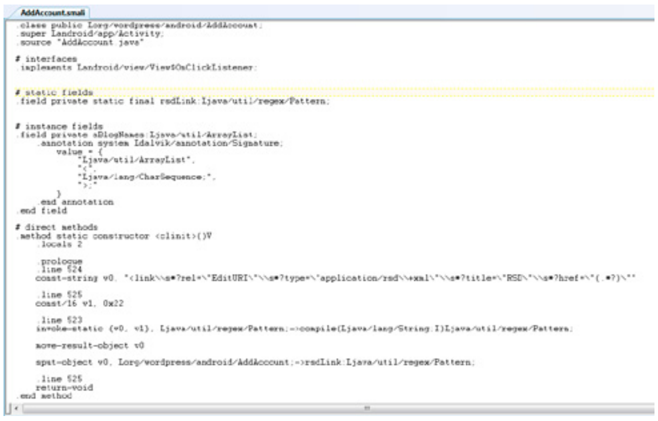


Figure 8. *classes.dex opcodes in 010 hexadecimal editor*



```
C:\Users\godfrey>apktool d org.wordpress.android-1.apk
I: Baksmaling...
testI: Loading resource table...
I: Loaded.
I: Loading resource table from file: C:\Users\godfrey\apktool\framework\1.apk
I: Loaded.
I: Decoding file-resources...
I: Decoding values*/* XMLs...
I: Done.
I: Copying assets and libs...

C:\Users\godfrey>
```

Figure 9. *apktool usage*



```
.class public Lorg/wordpress/android/AddAccount;
.super Landroid/app/Activity;
.source "AddAccount.java"

# interfaces
.implements Landroid/view/View$OnClickListener;


# static fields
.field private static final rsdLink:Ljava/util/regex/Pattern;


# instance fields
.field private sDlogNames:Ljava/util/ArrayList;
    .annotation system Ldalvik/annotation/Signature;
        value = {
            "Ljava/util/ArrayList",
            "<",
            "Ljava/lang/CharSequence;",
            ">;"
        }
    .end annotation
.end field

# direct methods
.method static constructor <clinit>()V
    .locals 2

    .prologue
    .line 524
    const-string v0, "<link\\s*?rel=\"EditURI\"\\s*?type=\"application/rsd\\+xml\"\\s*?title=\"RSD\"\\s*?href=\"(.*?)\""

    .line 525
    const/16 v1, 0x22

    .line 523
    invoke-static {v0, v1}, Ljava/util/regex/Pattern;->compile(Ljava/lang/String;I)Ljava/util/regex/Pattern;

    move-result-object v0

    sput-object v0, Lorg/wordpress/android/AddAccount;->rsdLink:Ljava/util/regex/Pattern;

    .line 525
    return-void
.end method
```

Figure 10. *AddAccount.smali*

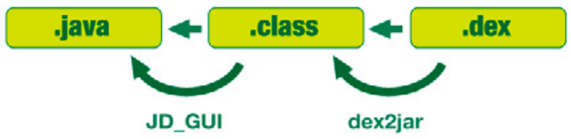


Figure 11. *Android Decompilation process*

dex to a Java class file format. We can now see in Figure 11 the two stage process for decompiling our classes.dex file back into Java.

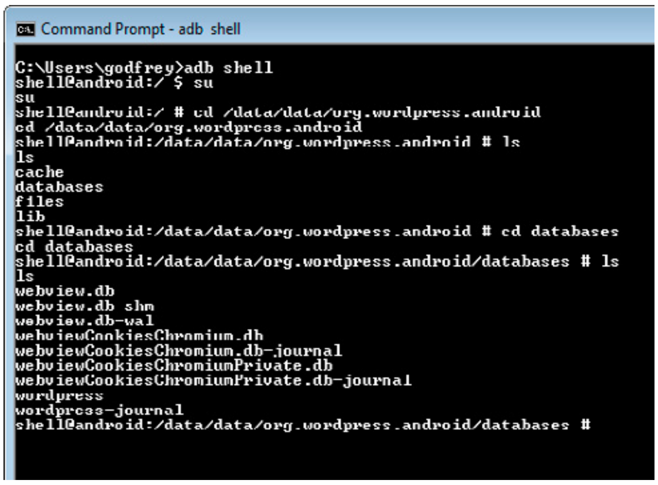
So if we run dex2jar on our org.wordpress.android-1.apk, see Figure 12, we get a file called org.wordpress.android-1_dex2jar.jar which we can now decompile using your favorite Java decompiler such as JD-GUI [10].

The decompiled Wordpress AddAccount class can be seen in Figure 13. Note that the code is readable, the variable names etc. are all intact and if you compare the code to the original source all that is really missing are the developer's comments.

Usually dex2jar isn't 100% successful when it converts the code back to Java, so although it would be possible it's unlikely that anyone would recompile the code and pass it off as a new app. However, what is decompiled is so readable that any Web Service URLs, API keys, usernames and passwords that are hard coded in the Java source will be visible to any attacker who can then use that information to attack back end databases and web servers.

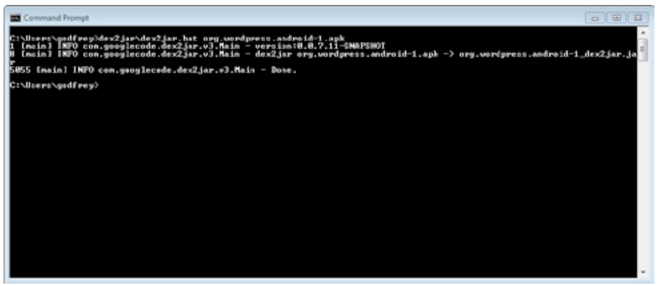
Any user data is stored at runtime, typically credit card information. That won't be found in the code but it's often just as easy to find information on the phone. While we don't need a rooted phone to gain access to the runtime data on the phone it's very helpful for finding where it lives on the phone. Instead of the `/data/app` folder go to the `/data/data` folder on the phone and cd to the org.android.wordpress folder where we can find several database files showing the app is using SQLite. Figure 14 shows the contents of the `/data/data/org.android.wordpress`.

Just like earlier we don't necessarily need a rooted phone to gain access to any SQLite databases. If you're using Ice Cream Sandwich or above you can use the adb command this time using the backup option. A simple adb backup -noapk org.wordpress.android will give us the contents of the databases folder. When you run the command the phone will ask you for a backup password but you don't need one, click on Backup my data and the backup will start.



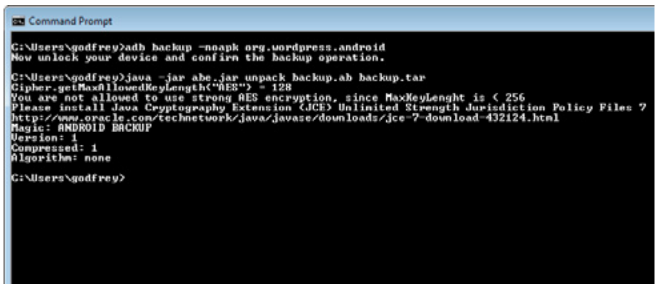
```
C:\Users\godfrey>adb shell
shell@android:/ $ su
su
shell@android:/ # cd /data/data/org.wordpress.android
cd /data/data/org.wordpress.android
shell@android:/data/data/org.wordpress.android # ls
ls
cache
databases
files
lib
shell@android:/data/data/org.wordpress.android # cd databases
cd databases
shell@android:/data/data/org.wordpress.android/databases # ls
ls
webview.db
webview.db-shm
webview.db-wal
webviewCookiesChromium.db
webviewCookiesChromium.db-journal
webviewCookiesChromiumPrivate.db
webviewCookiesChromiumPrivate.db-journal
wordpress
wordpress-journal
shell@android:/data/data/org.wordpress.android/databases #
```

Figure 14. *Viewing the APKs runtime information*



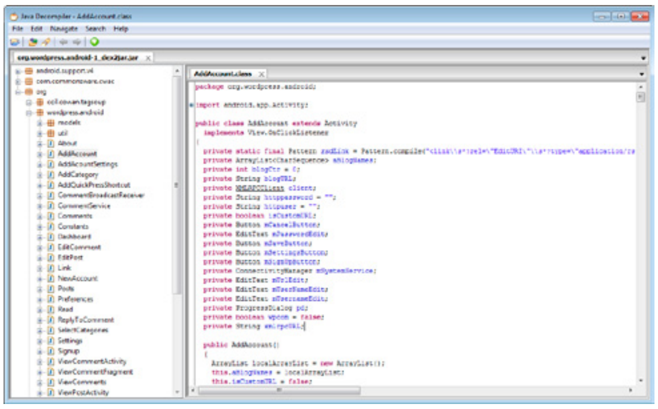
```
C:\Users\godfrey>dex2jar\dex2jar.bat org.wordpress.android-1.apk
1 [main] INFO com.googlecode.dex2jar.v3.Main - version:0.0.7.11-SNAPSHOT
8 [main] INFO com.googlecode.dex2jar.v3.Main - dex2jar org.wordpress.android-1.apk -> org.wordpress.android-1_dex2jar.ja
r
5055 [main] INFO com.googlecode.dex2jar.v3.Main - Done.

C:\Users\godfrey>
```

Figure 12. *dex2jar usage*



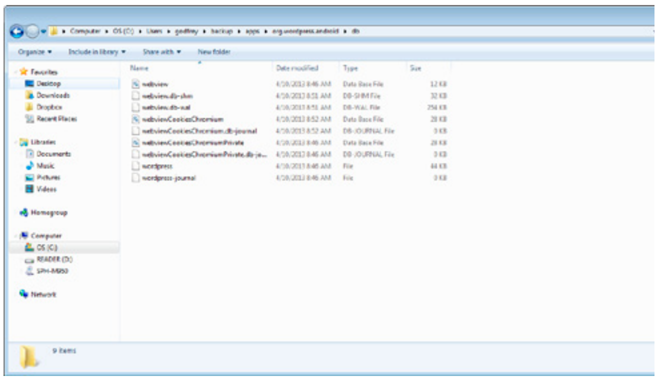
```
C:\Users\godfrey>adb backup -noapk org.wordpress.android
Now unlock your device and confirm the backup operation.

C:\Users\godfrey>java -jar abe.jar unpack backup.ab backup.tar
Cipher.getMaxAllowedKeyLength("AES") = 128
You are not allowed to use strong AES encryption, since MaxKeyLenght is < 256
Please install Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7
http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html
Magic: ANDROID BACKUP
Version: 1
Compressed: 1
Algorithm: none

C:\Users\godfrey>
```

Figure 15. *Getting a copy of the SQLite databases off the phone*



```
package org.wordpress.android;

import android.app.Activity;

public class AddAccount extends Activity
  implements View.OnClickListener
{
  private static final Pattern rsdLink = Pattern.compile("<link\\s*?rel=\"EditURI\"\\s*?type=\"application/rs
  private ArrayList<CharSequence> aBlogNames;
  private int blogCtr = 0;
  private String blogURL;
  private XMLRPCClient client;
  private String httppassword = "";
  private String httpuser = "";
  private boolean isCustomURL;
  private Button mCancelButton;
  private EditText mPasswordEdit;
  private Button mSaveButton;
  private Button mSettingsButton;
  private Button mSignUpButton;
  private ConnectivityManager mSystemService;
  private EditText mUrlEdit;
  private EditText mUserNameEdit;
  private EditText mUsernameEdit;
  private ProgressDialog pd;
  private boolean wpcom = false;
  private String xmlrpcURL;

  public AddAccount()
  {
    ArrayList localArrayList = new ArrayList();
    this.aBlogNames = localArrayList;
    this.isCustomURL = false;
```

Figure 13. *Using JD-GUI to decompile*



Name	Date modified	Type	Size
webview	4/18/2013 8:46 AM	Data Base File	12 KB
webview.db-shm	4/18/2013 8:51 AM	DB-SHM File	32 KB
webview.db-wal	4/18/2013 8:51 AM	DB-WAL File	254 KB
webviewCookiesChromium	4/18/2013 8:52 AM	Data Base File	28 KB
webviewCookiesChromium.db-journal	4/18/2013 8:52 AM	DB-JOURNAL File	3 KB
webviewCookiesChromiumPrivate	4/18/2013 8:46 AM	Data Base File	28 KB
webviewCookiesChromiumPrivate.db-jo...	4/18/2013 8:46 AM	DB-JOURNAL File	3 KB
wordpress	4/18/2013 8:46 AM	File	44 KB
wordpress-journal	4/18/2013 8:46 AM	File	3 KB

Figure 16. *Wordpress' SQLite databases*

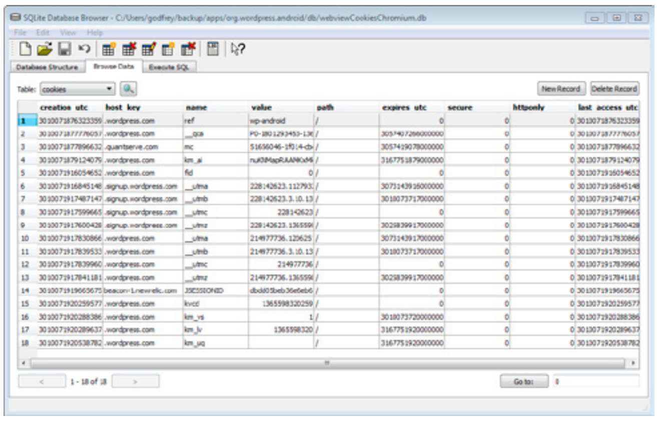
Once completed the backup will be in an Android backup or ab format so we'll need to convert that into something we can read before we can look at the databases. We do this using the Android Backup Extractor tool [12] which will convert our backup.ab into a tar format which can then be untarred leaving the databases in a native SQLite format. The convert the backup.ab file use the following command java -jar abe.jar unpack backup.ab backup.tar. Figure 15 shows the series of backup commands.

Use tar –xvf or your unzip to extract the tar file and you'll see the list of files in the database folder, see Figure 16.

SQLite Database Browser [13], see Figure 17 is an excellent way of looking at offline SQLite databases. The example shown does not contain any sensitive user information, but many Android APKs use the SQLite database for storing a user's login information and credit card details if they are used in the application to make the app have a better user experience, especially if the user is expected to enter the information multiple times.

There are several ways to mitigate the issues such as using obfuscation and hiding information in C++ using the Android NDK but they are not always 100% effective because of the nature of the Android platform. Google did try to introduce Google Encryption at the last Google I/O but it broke many of the existing apps so it was pulled. The best approach is to follow the OWASP top 10 mobile controls [14] which basically say don't hardcode any credentials in your code and don't store any important user data on the device, store it on the server side and always transmit any sensitive data securely.

This article is a cut down version of the Decompiling Android workshop that I've presented at several Android conferences and local user groups. I believe I've shown how easy it is to reverse engineer an Android APK as well as how to obtain any runtime database information. You can find out more information about the workshop, our tools and the Decompiling Android book at *http://www.decompilingandroid.com*.



	creation_utc	host_key	name	value	path	expires_utc	secure	httponly	last_access_utc
1	3010071876323359	.wordpress.com	ref	wp-android	/	0	0	0	3013071876323359
2	3010071877776057	.wordpress.com	__qca	P0-1801293453-136	/	3057407266000000	0	0	3013071877776057
3	3010071877896632	.quantserve.com	mc	51656046-1f0:4-db	/	3057419078000000	0	0	3013071877896632
4	3010071879124079	.wordpress.com	km_ai	nuKtMapRJAAKxM6	/	3167751879000000	0	0	3013071879124079
5	3010071916054652	.wordpress.com	ñd	0	/	0	0	0	3013071916054652
6	3010071916845148	.signup.wordpress.com	__utma	228142623.112793	/	3073143916000000	0	0	3013071916845148
7	3010071917487147	.signup.wordpress.com	__utmb	228142623.3.10.13	/	3010073717000000	0	0	3013071917487147
8	3010071917599665	.signup.wordpress.com	__utmc	228142623	/	0	0	0	3013071917599665
9	3010071917600428	.signup.wordpress.com	__utmz	228142623.136559	/	3025839917000000	0	0	3013071917600428
10	3010071917830866	.wordpress.com	__utma	214977736.120625	/	3073143917000000	0	0	3013071917830866
11	3010071917839533	.wordpress.com	__utmb	214977736.3.10.13	/	3010073717000000	0	0	3013071917839533
12	3010071917839960	.wordpress.com	__utmc	214977736	/	0	0	0	3013071917839960
13	3010071917841181	.wordpress.com	__utmz	214977736.136559	/	3025839917000000	0	0	3013071917841181
14	3010071919665675	beacon-1.newrelic.com	JSESSIONID	dbdd05beb36e6eb6	/	0	0	0	3013071919665675
15	3010071920259577	.wordpress.com	kvcd	1365598320259	/	0	0	0	3013071920259577
16	3010071920288386	.wordpress.com	km_vs	1	/	3010073720000000	0	0	3013071920288386
17	3010071920289637	.wordpress.com	km_lv	1365598320	/	3167751920000000	0	0	3013071920289637
18	3010071920538782	.wordpress.com	km_uq		/	3167751920000000	0	0	3013071920538782

Figure 17. *SQLite Database Browser*

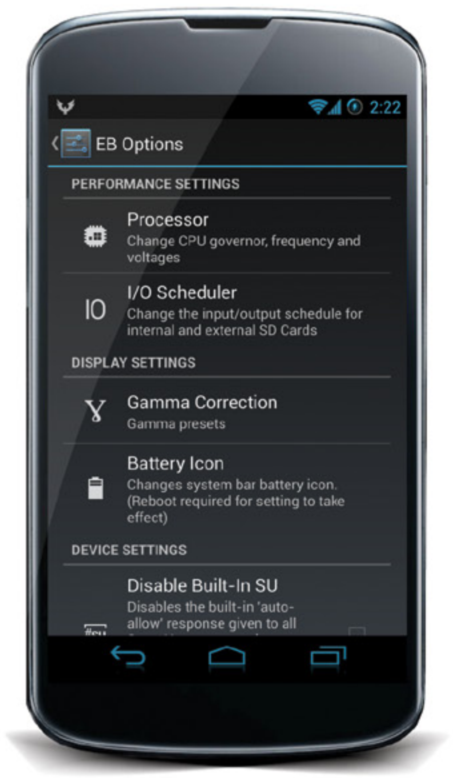
References

[1] *https://developers.google.com/closure/*
[2] *https://code.google.com/p/smali/*
[3] *http://www.netmite.com/android/mydroid/dalvik/docs/dex-format.html*
[4] *http://www.sweetscape.com/010editor/*
[5] *http://www.sweetscape.com/010editor/templates/files/DEXTemplate.bt*
[6] *http://www.netmite.com/android/mydroid/dalvik/docs/instruction-formats.html*
[7] *https://code.google.com/p/android-apktool/*
[8] *https://code.google.com/p/android4me/*
[9] *https://code.google.com/p/dex2jar/*
[10] *http://java.decompiler.free.fr/?q=jdgui*
[11] *http://android.svn.wordpress.org/*
[12] *http://sourceforge.net/projects/adbextractor/files/*
[13] *http://sourceforge.net/projects/sqlitebrowser/*
[14] *https://www.owasp.org/index.php/OWASP_Mobile_Security_Project*

GODFREY NOLAN

Godfrey Nolan is the President of RIIS LLC. He is also the author of Decompiling Android and Decompiling Java both published by Apress and is currently working on Android Best Practices. Decompiling Android looks at the reason why Android apps can be decompiled to recover their source code, what it means to Android developers and how you can protect your code from prying eyes. Godfrey has talked at conferences such as AnDevCon, Google DevFest, CodeMash and JavaOne as well as many local user group events on mobile and web development topics. RIIS is an IT consulting firm based in Southfield, MI. Our primary service includes accelerated application development using agile processes for both the web and mobile technologies. Industry experience includes software, retail, advertising, defense, insurance, banking/finance, and telecommunications.

fast. stable. clean.



EB Options

PERFORMANCE SETTINGS

Processor
Change CPU governor, frequency and voltages

I/O Scheduler
Change the input/output schedule for internal and external SD Cards

DISPLAY SETTINGS

Gamma Correction
Gamma presets

Battery Icon
Changes system bar battery icon. (Reboot required for setting to take effect)

DEVICE SETTINGS

Disable Built-In SU
Disables the built-in 'auto-allow' response given to all

EaglesBlood™ Development

EaglesBlood.com

Android OS

Getting Started with Customizing Your Own Rom

It's no secret that today we rely on our smartphones more than ever before. This theory is only going to grow truer as time progresses. We are very close to having desktop capabilities in the palm of our hands and more and more site visits are logged from a mobile device than ever before.

When it comes to the mobile arena we basically have three main platforms to choose from: iOS, Windows, and Android. Out of those three there is one that stands out to the more tech-savvy crowd – Android. The reason Android is appealing is because of its nature and characteristics.

Android is an open-source platform that is owned and currently developed by Google, Inc. While relatively new, compared to its mobile counterparts, Android has deep long-standing roots in a system most are familiar with or at least have heard of – Linux. Android runs on a Linux-based kernel and has a system directory tree very similar to what one might see on their local Linux PC distribution (such as Debian or Ubuntu). If you are familiar with Linux than you will find yourself right at home as you begin venturing into the realm of Android. In fact, the techniques and practices discussed in this article require the user to be running a Linux-based platform (again such as Ubuntu) as it makes for a much smoother experience and some tasks can become exponentially more difficult if trying to do so on a Windows or Mac machine.

To get started you will need to download a few basic key components: the Android SDK (Software Development Kit), the Android API for Eclipse, and the Android Source Code. All of these tools can be downloaded from the Android Development websites below and you should follow the guides to begin setting up your development environment. The site also includes key fundamentals, how-to instructions, sample codes, APIs and best practices that are a must for anyone getting started with Android development. These tools can be accessed through this landing page: *http://developer.android.com/tools/index.html*.

The source code is available to download and hosted under their public moniker Android Open Source Project (AOSP). Since the latest releases are ever-changing it is best to visit the Official AOSP website to obtain the latest release and follow the instructions to setup a local checkout of the repository: *http://source.android.com/source/downloading.html*.

The last item you will need to begin building your own Android ROM is a device. Any device will work as long as it has "root" level access. This means that the user has access and "-rw" rights to the "/" root directory of the filesystem. Generally, in order to root a device you will also need to unlock the bootloader if the device has one. The steps required to do this vary from device to device so by doing a few site-searches you should be able to find the steps required to root your own device model. For purposes of this article we will be referencing the Galaxy Nexus i-9250 (maguro) smartphone. This is an open GSM model (meaning it uses a SIM-card on any GSM mobile network) and is distributed by Google directly (*https://play.google.com/store/devices*) The Galaxy Nexus (*maguro*) is quite similar to its current successor the Nexus 4 (*mako*).

Recovery

When rooting your device many users will typically find that the method of choice often includes installing a custom recovery interface. The Recovery area resides before the OS boots up and can be used to install or modify parts of the Operating System prior to the system loading. This is where you may often hear the term "flash" being used; such as "flash a custom ROM" …in short the term flash simply means "install." We prefer to use *ClockworkMod Recovery*, which is developed and owned by Koushik Dutta. You can find more on this recovery by visiting the site (*http://clockworkmod.com/*).

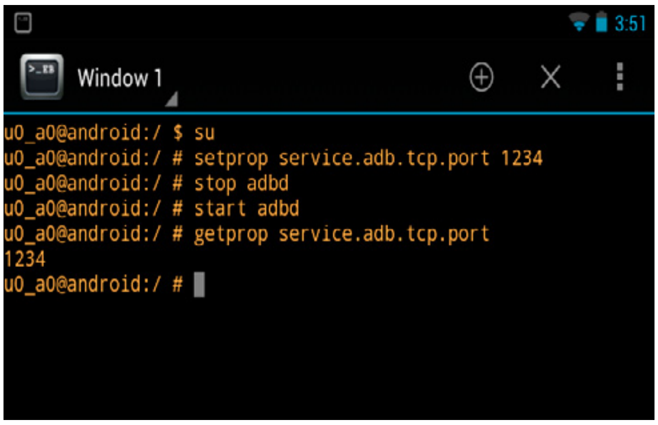
ADB Over WIFI

Android offers a direct link interface that allows you to control your Android device via the *Android Debugging Bridge (ADB)*. Generally, this is done using a direct USB cable connection from the host machine to the device. However, one can bypass the USB method by using a wifi-based connection using the following method (Figure 1).

Note

That while using the ADB method over wifi your device will be vulnerable and exposed for open access by any other machines on the same network so do use caution and deactivate the WiFi-ADB once you have finished using it.

Using any Terminal Emulator application on your Android device run:



```
u0_a0@android:/ $ su
u0_a0@android:/ # setprop service.adb.tcp.port 1234
u0_a0@android:/ # stop adbd
u0_a0@android:/ # start adbd
u0_a0@android:/ # getprop service.adb.tcp.port
1234
u0_a0@android:/ #
```

Figure 1. *Using a terminal emulator on an Android device to set the ADB listening port to TCP*

```
su
setprop service.adb.tcp.port 1234
stop adbd
start adbd
```

Note

"*1234*" can be any port you choose, so change the digits as desired.

Confirm the connection by using:

```
getprop service.adb.tcp.port
```

If the command above returns your port, in this case "1234," then run this command below in the terminal (or command prompt) on your host PC, *(Enter your device's IP address in place of 192.168.x.xx)*:

```
adb connect 192.168.x.xx
```

Note

Your machine must have ADB installed and your device & machine must both be on the same network.

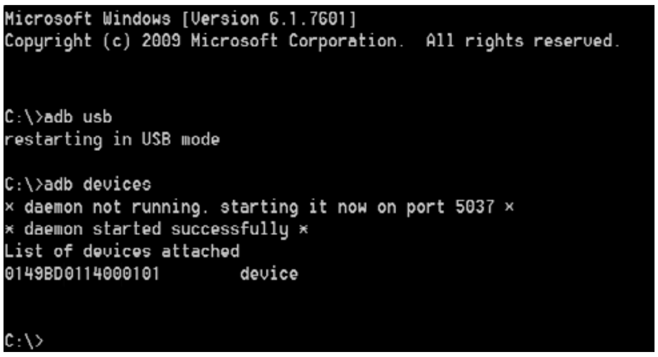
Once complete, you can switch back to USB and shutdown the Wifi-ADB by telling the device to use the USB port instead of TCP (Figure 2).

On the Android device, enter this command in the Terminal Emulator app:

```
su
setprop service.adb.tcp.port -1
stop adbd
start adbd
```

Now back on our host PC let's enter the following command to tell it to listen on the USB port instead of TCP:

```
adb usb
```



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.


C:\>adb usb
restarting in USB mode

C:\>adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
0149BD0114000101        device


C:\>
```

Figure 2. *Setting the ADB listening port back to USB on a Windows PC using command prompt*

Or you can simply reboot the Android device which will reset the ADB interface preference back to the USB port.

APK Tool

Note: This can be done using both Windows and/or Linux platforms.

APK's are compiled Android Application files and are identified by the .apk file extension. You can think of these as being similar to ".exe" files on a Windows platform. APK's are essentially Apps and will execute an install procedure natively when attempting to open them. APK's are compiled *(meaning built against any supporting libraries or other dependencies)* and packaged into a finished project; thus creating the *.apk file. During the final stage of the compiling process almost all applications are processed through a tool called Proguard, which is an obfuscation method used to encrypt the XML and JAVA files within the finished APK file. This helps prevent unauthorized duplication or copying of applications in a verbatim manner. If one does not have access to source code then these proprietary files are quite difficult to manipulate and/or alter since the only code a user has access too has been jumbled using the obfuscation method mentioned above.

There is a tool however than can use sort of a reverse-engineering method to make the obfuscated code more readable. This can be done using a program called "APK Tool." The download and setup instructions can be found here: *https://code.google.com/p/android-apktool/*.

Once you have the tool installed and the framework-res directory identified you can executive the following command to decompile the APK file, we'll use `somefile.apk` as an example (Figure 3):

```
apktool d somefile.apk somefile
```

The screenshot above shows a successful execution of the command using apktool. If you receive an error it is because either the "framework-res.apk" file has not yet been identified or because the APK file you are attempting to unpack is corrupt or invalid. Read the instructions from the download link above to properly setup the APKtool or run the help command for more info:

On Linux:

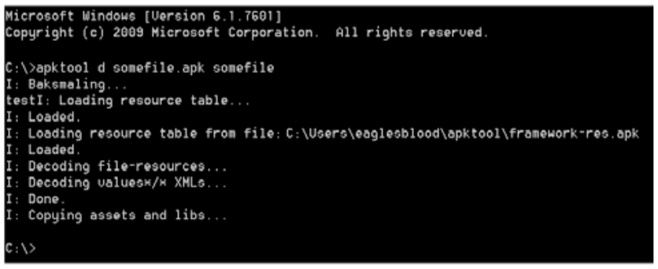
```
apktool --help
```

On PC:

```
apktool /help
```

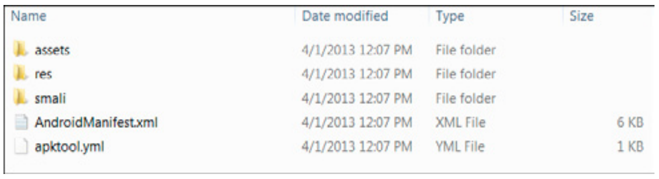
Once the command completes you should have a folder `/somefile` with the contents similar to what is shown in Figure 4.

From here you can view XML files in the /res directory and also the AndroidManifest.xml file in human readable formats. This is useful if you have two versions of a particular APK file and you are trying to identify which is the newer incremental version of the application. Additionally, you can edit certain things such as hex color codes shown in `/res/values/*.xml` or edit the text names in `/res/values/strings.xml`. Once your edits are complete you will need an archive manager tool (such as 7zip or FileRoller) and you will need to go back to



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\>apktool d somefile.apk somefile
I: Baksmaling...
testI: Loading resource table...
I: Loaded.
I: Loading resource table from file: C:\Users\eaglesblood\apktool\framework-res.apk
I: Loaded.
I: Decoding file-resources...
I: Decoding values*/* XMLs...
I: Done.
I: Copying assets and libs...

C:\>
```

Figure 3. *Running the apktool decompile command on a Windows PC using command prompt*



Name	Date modified	Type	Size
assets	4/1/2013 12:07 PM	File folder	
res	4/1/2013 12:07 PM	File folder	
smali	4/1/2013 12:07 PM	File folder	
AndroidManifest.xml	4/1/2013 12:07 PM	XML File	6 KB
apktool.yml	4/1/2013 12:07 PM	YML File	1 KB

Figure 4. *A view of what the resulting folder should look like and contain after successfully completing the apktool decompile process*

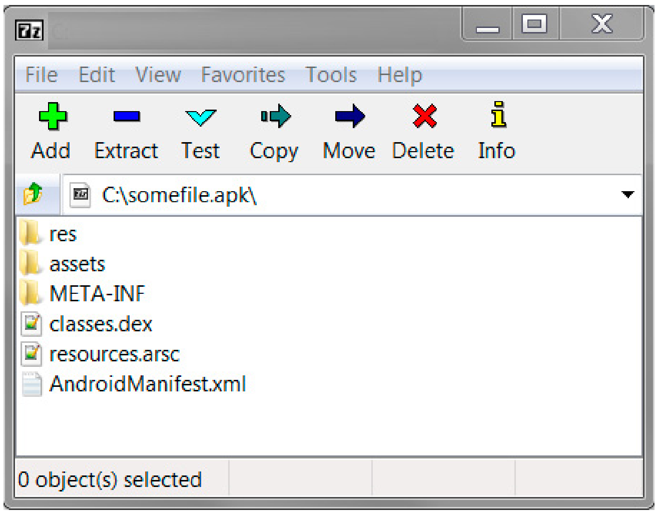


Figure 5. *A screenshot of what the .apk file contains when exploring with an archive tool such as 7zip or FileRoller. This is where you can slip your edited files back into place overwriting the current ones*

the original .apk file that is unedited and explore the contents. In the original .apk file archive find and replace the original files with the edited ones from your /somefile folder (Figure 5). Simply drag-and-drop the edited files in place and confirm to overwrite and update the archive .apk file. This newly altered APK file can now be sent to your android device and installed and your changes will be reflected in the application.

Note
Images and graphic files should not be used or pulled from the extracted apktool /somefile folder. If you wish to edit image files, you should manually pull out any images from the source .apk file and slip them back in using the method mentioned above. The apktool decryption script has a tendency to corrupt image files and they should not be used.

.9.PNG Images

Android runs on multiple devices with many different screen-sizes and layouts can be displayed in both Portrait and Landscape. With that being said, you can imagine the massive amount of bitmap background and icon sizes that would be required to support all screens and layouts. The way that Android works around this issue is by using "stretchable bitmap" images – .9.png's. There is a convenient tool included in the `Android-SDK/tools/` called the `Draw9Patch` which makes this job easier when converting your own icons or graphics.

Let's take the image icon below as an example (Figure 6). We'll call this image "button.png" for this example.

As you can see it's a simple square button with some readable text in the center shown as simply "TEXT." When Android uses this button.png in various layouts it needs to know how to stretch this button to fill the desired space and where to stretch it. This is where the 9.png portion comes into play.

There are two schools of thought when it comes to modifying or making .9.png images. You can use the *Draw9Patch* that comes with the Android SDK or you can use a 3rd party graphical editing program such as *GIMP* or *Photoshop*. The Draw9Patch is the recommended method for most people just starting to learn how this file format works and makes for an easier and safer learning experience.

To use the Draw9Patch tool simply click to open it and drag-and-drop your .png image into the tool's window. You will notice that the tool automatically enlarges the canvas by 1-pixel on each side. This area is transparent and known as the *alpha channel*, which allows you to identify which parts of the image you want to allow for expansion. You identify these stretchable areas by drawing a solid black line on the left vertical column and on the top horizontal row of pixels on the alpha channel (Figure 7). The preview on the right-hand side of the tool shows you what the stretched image will look like in various layouts. You can note in the example that the black line breaks around components that should not be stretched. A few examples of things that should not be stretched are text, action symbols, numbers, etc..

Once you are happy with your image you can click *File* > *Save As…* and select your destination folder and it will save the image with the *.9.png* file extension; in our example it will be saved as "button.9.png." If done correctly the alpha channel containing the stretchable pixel marks will not be visible to the user when displayed on a device. There are more advanced features that you can achieve and identify using the Draw9Patch tool and .9.png's in general but this is the basic functionality of the way that Android handles single images for many screen-sizes.

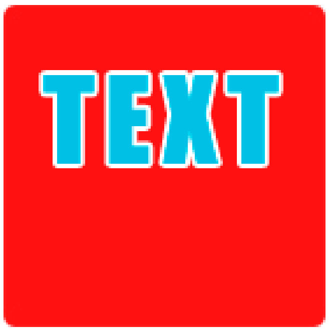


Figure 6. *An example of a button.png image file before converting to a .9.png image*

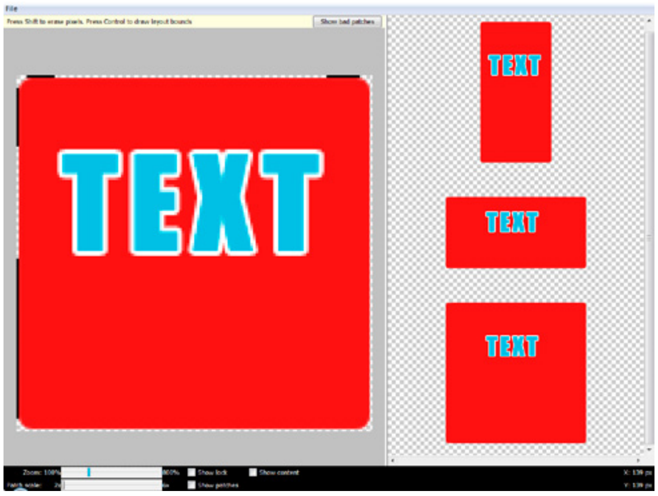


Figure 7. *A screenshot while using the Draw9Patch tool from the Android SDK. The right-hand column shows what the stretched images will look like based on your patches made in the left column*

Building Kernels

When it comes to getting involved with building a rom you eventually are faced with the need to work with kernels. This may also be the first instance of dealing with a piece of software in the rom that falls under a license that requires you to make any source changes available. You have to be very conscious of that fact, the kernel and the rom are separate and no matter what your mode of development is for the rom, you have to adhere to GPL requirements for the kernel(s) you work with and possibly modify.

Working with kernels is something Linux veterans might have never even done. The procedure there is pretty much the same for working with android device kernels. The main difference is that you'll be building for architecture other than the one you're working on, and you'll likely be using the toolchains provided through your rom repository so that there are no questions of compatibility.

For the purposes of the following paragraphs, we'll assume you have downloaded AOSP source for the rom in a directory like `/usr/src/android/rom` and that you want to use the toolchain provided by it. Otherwise you'll have to download the appropriate cross compiling packages for your particular distribution and any appropriate or necessary android patches.

The first step you need to take is to retrieve the kernel source for your device. A good start is to go get the stock kernel source for your device and make sure you can build and install a stock kernel before making any changes or working off someone else's source repository.

Let's assume you are using a Samsung Galaxy Nexus. Since this is a Nexus phone, Google controls the software so we can find the locations on their AOSP pages: *http://source.android.com/source/index.html*.

For this device, the repo we need to clone is:

```
https://android.googlesource.com/kernel/omap.git
```

If you are using another device, you'll have to go to the manufacturer's website and search for the kernel source yourself or contact them by email for the link.

With the Nexus phones, you can check what the exact revision is that your ROM or the source of the rom you're going to build has for the stock kernel. Let's assume you have pulled the latest android AOSP rom repository. You would execute:

```
cd device/samsung/tuna
git log kernel
```

The first response will tell you what kernel git revision was used to create that kernel. In 4.2.2's case this would be the beginning of `9f818de mm: Hold a file reference in madvise _ remove`. Now, you know what revision to checkout of your cloned kernel source to get the same source that the AOSP rom supplies. So cd back to your kernel repo you cloned with:

```
git clone https://android.googlesource.com/kernel/
                    omap.git
```

Then execute:

```
git checkout 9f818de
```

This will create a detached source tree (*means you can't commit back upstream but that's ok, we don't care about doing that right now*).

Now, we have the correct revision of kernel source downloaded we can begin configuring it. This is best done by starting with a configuration that is known to work (stock kernel config). There are a number of ways to get this. The easiest is on a running kernel, since most have enabled the kernel config proc interface, providing you with the config file used in `/proc/config.gz`. Uncompressing that and placing it in your kernel source tree as .config will allow you to base your kernel build off the same build options. The second way is to check the default config file in your kernel source tree `arch/arm/configs/tuna_defconfig` and copy it to `.config` in the top level of your kernel source. There are also other ways of retrieving the config.gz file that is used in the proc interface in a non-running kernel but that's out of the scope of this little tutorial.

With the kernel config in place, we can now begin compiling the kernel:

```
make ARCH=arm CROSS_COMPILE=/usr/src/android/rom/
prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/bin/arm-
                    eabi-
```

You can save the value of `CROSS _ COMPILER` to an environment variable for your shell if you plan on doing this often. Then just use that instead of the full path.

This will automatically run after configuring the kernel with the .config file in the kernel directory you copied. Depending on your development environment, you can run one of the various config programs to change settings. I prefer `menuconfig`, (Figure 8) this requires `ncurses` development headers and libs (installed via your Linux distribution package manager). Also, if you have a system with multiple CPUs or cores you should run the make command with the `-j` option followed by the number of cores you have. So the above command would look like this on our system:

```
make -j8 ARCH=arm CROSS_COMPILE=/usr/src/android/

rom/prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/bin/
                  arm-eabi-
```

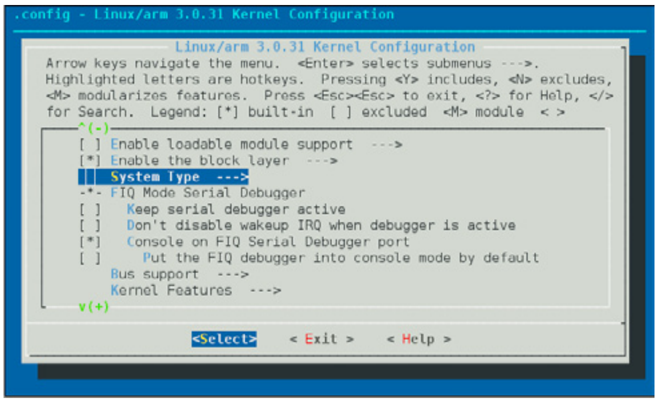
Once it finishes building (*potentially anywhere from 1-minute to 20-minutes or more*) the build output will tell you the path to the finished kernel. There shouldn't be any modules for the Nexus devices but if your particular device uses modules, they'll also be listed in the output and have the extension of `.ko` (Figure 9). You will then copy this kernel to your ROM's device directory and name it "kernel" (`device/samsung/tuna/kernel`) if you intend to build your ROM with this new kernel or create a new boot image. Also, copy any kernel modules if your device utilizes them and replace the modules in your device directory by overwriting them.

If you want to test this kernel while only changing the kernel you will need to plug your device into your computer and using adb, and push the kernel to your sdcard. Once it's uploaded, open a shell via adb on your device and su to root:

```
adb shell
su
```

On your host computer, check for `recovery.fstab` in your device directory for your rom. It will tell you what partition is the `boot` partition. For the Galaxy Nexus it is `/dev/block/platform/omap/omap _ hsmmc.0/by-name/boot`.

On your adb shell terminal (*this is assuming you're using a Galaxy Nexus from our example above; replace paths as needed*) type:



```
.config - Linux/arm 3.0.31 Kernel Configuration
           Linux/arm 3.0.31 Kernel Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus --->.
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
 <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
 for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
     ^(-)
     [ ] Enable loadable module support  --->
     [*] Enable the block layer  --->
         System Type  --->
     -*- FIQ Mode Serial Debugger
     [ ]   Keep serial debugger active
     [ ]   Don't disable wakeup IRQ when debugger is active
     [*]   Console on FIQ Serial Debugger port
     [ ]     Put the FIQ debugger into console mode by default
         Bus support  --->
         Kernel Features  --->
     v(+)

            <Select>    < Exit >    < Help >
```

Figure 8. *An screenshot of what you will see when you run: make menuconfig on the Galaxy Nexus (maguro) kernel*

```
dd if=/dev/block/platform/omap/omap_hsmmc.0/
by-name/boot of=/sdcard/currentboot.img
```

This will copy your current boot image to a file you can use adb pull to download to your host computer. `Exit` the shell and `adb pull` the file to your host computer:

```
# exit
$ exit
$ adb pull /sdcard/currentboot.img
```

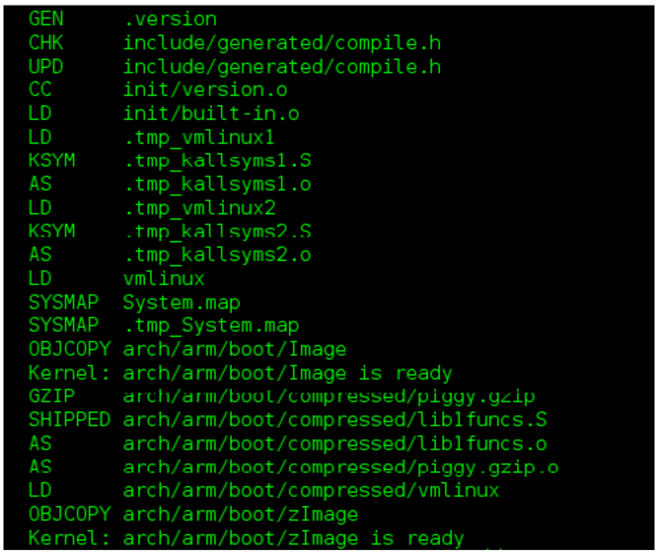
Next, you have two choices, follow the directions here: *http://android-dls.com/wiki/index.php?title=HOWTO:_Unpack%2C_Edit%2C_and_Re-Pack_Boot_Images*.

Or you download an "anykernel" zip templates from the files at: *https://github.com/koush/AnyKernel*.

And replace the *zImage* file with the kernel you built. Some further modification of the zip may be needed depending on which method you choose and if you choose to use a base zip file.

The easiest method if you're going to be running your own rom is to just build the rom and flash that zip. If you're already running your own rom you can rebuild just the boot image by running `make bootimage` in your rom source directory (*assuming you have run the prerequisite commands for setting up the device*) and copying that bootimage to your device's sdcard with adb. Then switch to root:

```
adb shell
su
```



```
  GEN     .version
  CHK     include/generated/compile.h
  UPD     include/generated/compile.h
  CC      init/version.o
  LD      init/built-in.o
  LD      .tmp_vmlinux1
  KSYM    .tmp_kallsyms1.S
  AS      .tmp_kallsyms1.o
  LD      .tmp_vmlinux2
  KSYM    .tmp_kallsyms2.S
  AS      .tmp_kallsyms2.o
  LD      vmlinux
  SYSMAP  System.map
  SYSMAP  .tmp_System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  GZIP    arch/arm/boot/compressed/piggy.gzip
  SHIPPED arch/arm/boot/compressed/lib1funcs.S
  AS      arch/arm/boot/compressed/lib1funcs.o
  AS      arch/arm/boot/compressed/piggy.gzip.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
```

Figure 9. *This is a screenshot of what results after building a kernel. The last line that says "Kernel" is the location of the actual kernel that will be copied into ROM device directory and named "kernel." No kernel modules were built this time, but if they were, they would have been a little above the kernel line and will end in a .ko extension*

Once running as root, you can `dd *your-boot-image*` to your boot partition as we determined above and once `dd` finishes, you can reboot into your new kernel.

That pretty much sums up android kernel compiling and installation. All of the patching and modification of the kernel is no different than the general methods used for patching any Linux kernel and no matter what you do (as long as you're copying the boot image to the correct partition) you can always boot to the rescue partition and fix things if you find yourself unable to boot *(the exception being that there is always the possibility of a patched kernel corrupting file systems or file system tables resulting in the need to do a full clean re-installation via Fastboot)*. The kernel has a number of configuration options even without patches that can alter performance and stability. It's always good to test these changes in small sets. Also, some people may think rebuilding the kernel with newer GCC's and enabling auto-vectorization is a easy way to get some cheap performance but you will almost certainly end up with a kernel that doesn't boot. Neon creates too many rounding issues and breaks math standards and while not always important in UserSpace, is much more important in the kernel. We've found that a stable kernel will almost always appear faster and smoother to a user than a kernel with aggressive compiler optimizations. The important non-patch requiring changes you can make to the kernel involve the scheduler *(both I/O and process)*. Phones with plenty of processing power will benefit from schedulers that take a little extra overhead to create a fair system for processes, while io benefits from a much more simpler scheduler like deadline or one of it's derivatives since flash doesn't require nearly as much logic as mechanical hard-drive schedulers that are the default in linux kernels.

Our Samsung Galaxy Nexus kernel can be found at: *http://kernel.eaglesblood.com/egb_tuna_kernel* …and its current config is in the `arch/arm/configs/egb_tuna_defconfig` and it's built with the AOSP toolchain that was referenced in the above paragraphs. We find it to perform very well and it contains numerous third party patches on top of the AOSP kernel. As with any kernel, for any ROM, you are welcome to build it and change it and make it your own, just remember, your changes are not proprietary and must be shared.

KELLEN RAZZANO

An Entrepenuer and Co-Founder of several startup companies. Kellen approaches situations with a "why not?" attitude and is more interested in focusing on "what is not working", rather than the things that do. Skill summary reaching over 15-years: Digital media creative, Android development, HTML/CSS/XML, ActionScript, web development, electronic hardware, Windows and Linux platforms.

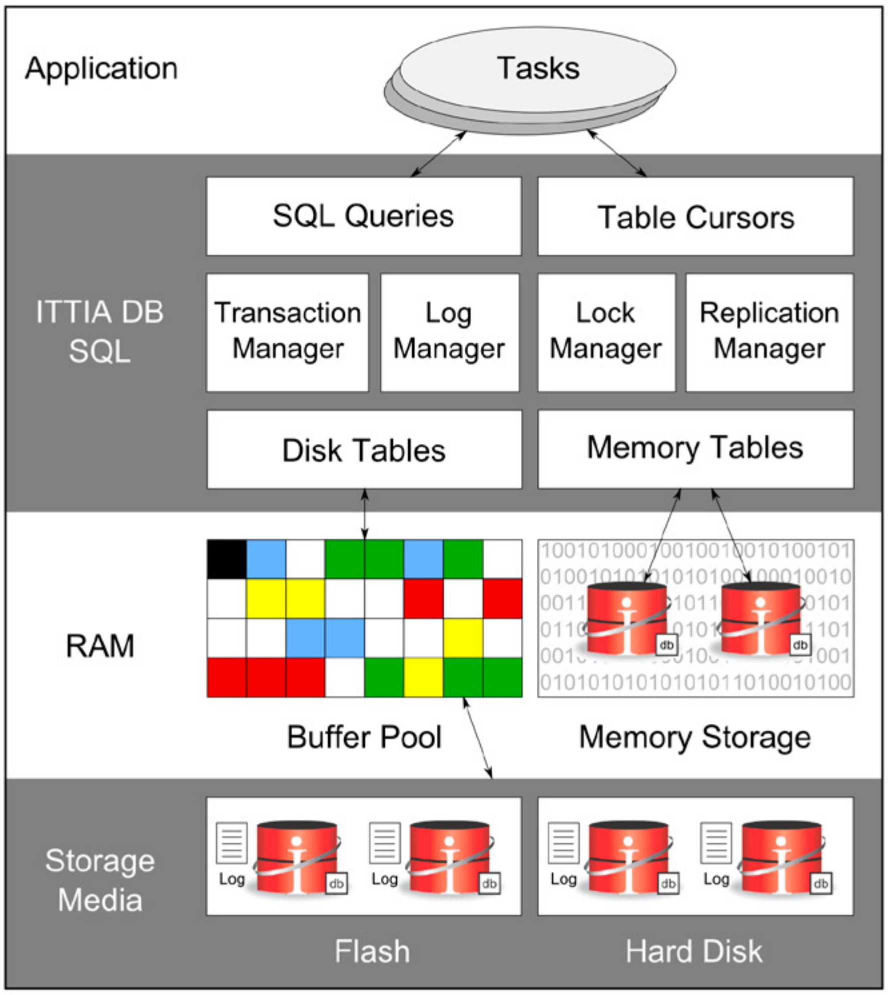
ED SWEETMAN

C/C++ Programmer and Co-Founder of a startup company. Ed is able to tackle problems programmatically and find solutions that others can not. A forward-thinker with a "can do" attitude provide his success in several open source projects. Skill summary includes: C/C++, Java, Android development, Linux kernels, HTML/CSS/XML, PHP and AJAX.

EaglesBlood™ Development, General Partners: Kellen Razzano, Ed Sweetman, Doni Kallmi.

A NEW MODERN WAY

OF EMBEDDED DATA MANAGEMENT FOR DEVICES



Save significant time and cost in your development!

Supported Hardware: ARM, MIPS, PowerPC, Intel x86, x86_64, and more…

Supported Operating Systems: Windows, Linux, Android, QNX, ThreadX, µC/OS-II, VxWorks, no OS, and more…

Leading-edge embedded database software for connected devices, with enterprise characteristics

ITTIA DB SQL

WWW.ITTIA.COM

For more information, please call +001 425-462-0046 or contact info@ittia.com

How to Research an APK

The amount of malware seen on mobile devices has sky rocketed in the last couple of years. The primary target for malware authors is Android devices which use Application Package (APK) files to run apps. Malware can send premium text messages in the background, steal personal information, root your device, or whatever else they can devise.

Some malware authors create a new APK that is malicious, while others hide their code within a legitimate APK. By using a couple simple tools, you can research APK to find what malicious intent may be lurking.

What's in an APK?

An APK is simply a compressed file. If you turn the APK into a ZIP file and extract it, this is what you'll see:

Assets Directory

This is used to store raw asset files. It can contain things like another APK used as a dropper, text or sql lite files that contain SMS numbers, etc. The assets directory is not always present.

META-INF Directory

This contains the certificate information for the APK. When a legitimate app is compromised, the digital certificate changes because the malware author has to sign it with a new certficate. This is a good way to identity compromised APKs.

Res Directory

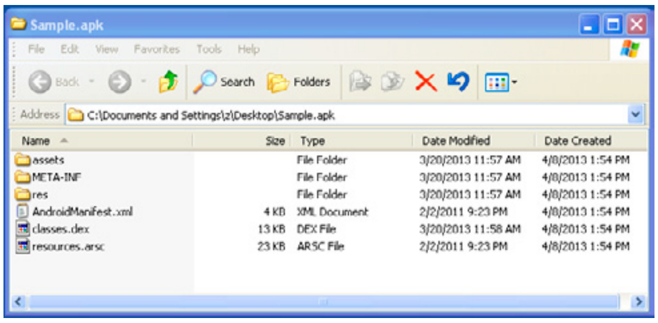
This contains all the resources for the APK such as images, layout files, and strings values, etc. The "values" folder contains the "strings.xml" file. This file can be potentially suspicious. Some malware authors will store premium SMS numbers, fake EULA messages, or other malicious strings in here.

AndroidManifest.xml

This file is what controls the APK. It contains permissions, services, receivers, activites, etc. It is the starting point when analyzing any APK since it contains everything the app is going to run, and what permissions it has. Note that this is not readable until decompiled using a tool such as 'APKtool'.

Classes.dex

This is where all the code can be found. It can be converted to Java, or the assembly language Smali. The classes.dex is a JAR file so when it is decompiled it is then arranged in a tree-hierarchy structure.


Sample.apk
File Edit View Favorites Tools Help
Back Search Folders
Address C:\Documents and Settings\z\Desktop\Sample.apk

Name	Size	Type	Date Modified	Date Created
assets		File Folder	3/20/2013 11:57 AM	4/8/2013 1:54 PM
META-INF		File Folder	3/20/2013 11:57 AM	4/8/2013 1:54 PM
res		File Folder	3/20/2013 11:57 AM	4/8/2013 1:54 PM
AndroidManifest.xml	4 KB	XML Document	2/2/2011 9:23 PM	4/8/2013 1:54 PM
classes.dex	13 KB	DEX File	3/20/2013 11:58 AM	4/8/2013 1:54 PM
resources.arsc	23 KB	ARSC File	2/2/2011 9:23 PM	4/8/2013 1:54 PM

Figure 1. *Contents of an APK*

Listing 1. *Manifest receiver example*

```
<receiver android:name=".MessageReceiver">
   <intent-filter android:priority="999">
      <action android:name="android.provider.Telephony.SMS_RECEIVED" />
   </intent-filter>
<receiver>
```

Listing 2. *MessageReceiver code*

```
public class MessageReceiver extends BroadcastReceiver
{
   public void onReceive9Context paramContext, Intent paramIntent)
   {
      if (!paramIntent.getAction().equals("android.provider.Telephony.SMS_RECEIVED"))
         return;
      ...
      SmsMessage[] arrayOfSmsMessage1 = (SmsMessage[])0;
      ...
      while (true)
      {
         String str1;
         try
         {
            Object[] arrayOfObject = (Object[])localBundle.get("pdus");
            SmsMessage[] arrayOfSmsMessage2 - new SmsMessage[arrayOfObject.length];
            int i - 0;
            int j = arrayOfSmsMessage2.length;
            if (i >- j)
               return;
            SmsMessage localSmsMessage = SmsMessage.createFromPdu((byte[])arrayOfObject[1]);
            arrayOfSmsmMessage2[i] = localSmsMessage;
            str1 = arrayOfSmsMessage2[i].getOriginatingAddress();
            String str2 = arrayOfSmsMessage2[1].getMessageBody();
            if ((str1.equals("111")))
            {
               ...
               abortBroadcast();
               1 +=1;
               continue;
            }
            ...
         }
         catch (Exception localException)
         {
            return;
         }
      }
   }
}
```

Resources.arsc

This is the list of resource location. There is nothing fun or of interest to be had here (Figure 1).

Using apktool

Apktool is a simple command line tool to decompile or build an APK. Using apktool, you decompile the APK. In order to decompile an APK, from the apktool directory, use the following command:

```
apktool d <APK>
```

When it was ran, it created a folder with the name of the APK with an ".out" at the end of folder name. Within this folder, you will find all the directories of the APK, the AndroidManifest.xml file in a readable state, and a folder named "Smali." Smali is a readable assembly language that the classes.dex file was then converted. To build an APK using the created .out directory, you would use the following command:

```
Apktool b <.out directory>
```

This will create a folder named "dist" in the .out folder with the built APK.

Using dex2jar

Dex2jar is a command line tool that decompiles the classes.dex file into a readable jar file where you can see code in Java. This is much easier to read than the assembly language Smali.

Analyzing the APK

The AndroidManifest.xml is the best place to start while researching an APK. The manifest controls everything, and nothing can run without it. The first place to look is at the permissions. Without the proper permissions the APK cannot run. For instance, *android.permission.SEND_SMS* gives permission to send text. Knowing that an APK has permission to send SMS messages, you must then ask yourself whether or not that is appropriate for the app. If a wallpaper app has permissions to send text, you know something is wrong. Next, look at the receivers. Receivers listen for actions to occur and when they do run a simple activity. For instance: Listing 1.

Listing 3. *Manifest example (clean)*

```
<manifest android:versionCode="12" android:versionName="2.1 (bugfixes for new phones) "
                 package="com.appspot.swisscodemonkeys.steam" xmlns:android="http://schemas.
                 android.com/apk/res/android">
   <uses-sdk android:minSdkVersion="3" android:targetSdkVersion="4" />
   <application android:label="@string/app_name" android:icon="@drawable/icon">
      <activity android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen"
                 android:label="@string/app_name" android:name=".Steam"
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name="com.appspot.swisscodemonkeys.steam.Preferences" />
      <activity android:name="com.appspot.swisscodemonkeys.steam.Calibrate" />
      <activity android:theme="@android:style/Theme.Dialog" android:name="cmn.AboutActivity />
      <mata-data android:name="analytics_id" android:value="UA-7184590-13" />
      <receiver android:name="vw.FixedAnalyticsReceiver" android:exported="true">
         <intent-filter>
            <action android:name="com.android.vending.INSTALL_REFERRER" />
         </intent-filter>
      </receiver>
   </application>
   <uses permission android:name="android.permission.INTERNET" />
   <uses permission android:name="android.permission.RECORD_AUDIO" />
   <uses permission android:name="android.permission.ACCESS_NETWORK_STATE" />
   <supports-screens />
</manifest>
```

The phone will broadcast the action *android.provider.Telephony.SMS_RECEIVED* whenever a SMS is received by the phone. If a receiver is set to listen for this action, it will run an activity whenever this action is called. In this case, *.MessageReceiver* will be run. Note the intent-filter with *android:priority="999"* in it. Whenever you receive a text, your phone's notification is set to listen to the action and notify you that you received a text unless a receiver with a higher priority receives it first. This is a big red flag when looking for malicious intent. Let us look at the code of *.MessageReceiver*: Listing 2.

Looking at the code, it appears that whenever an SMS is received, MessageReceiver will look through all of your text using an array and whenever the originating address is from 111, it aborts the broadcast so you will not be notified of it. This is done to hide the confirmation text from the premium SMS number. The only notification you will get is when you receive your phone bill with an extra charge and the hackers already have your money.

Services are also a hot spot to look for malicious activity. Services are listed in the manifest so you can easily track them down in code. They run in the background so the user never knows what has occurred. They can do an array of malicious things such as send texts, root the device, download and install an APK, etc.

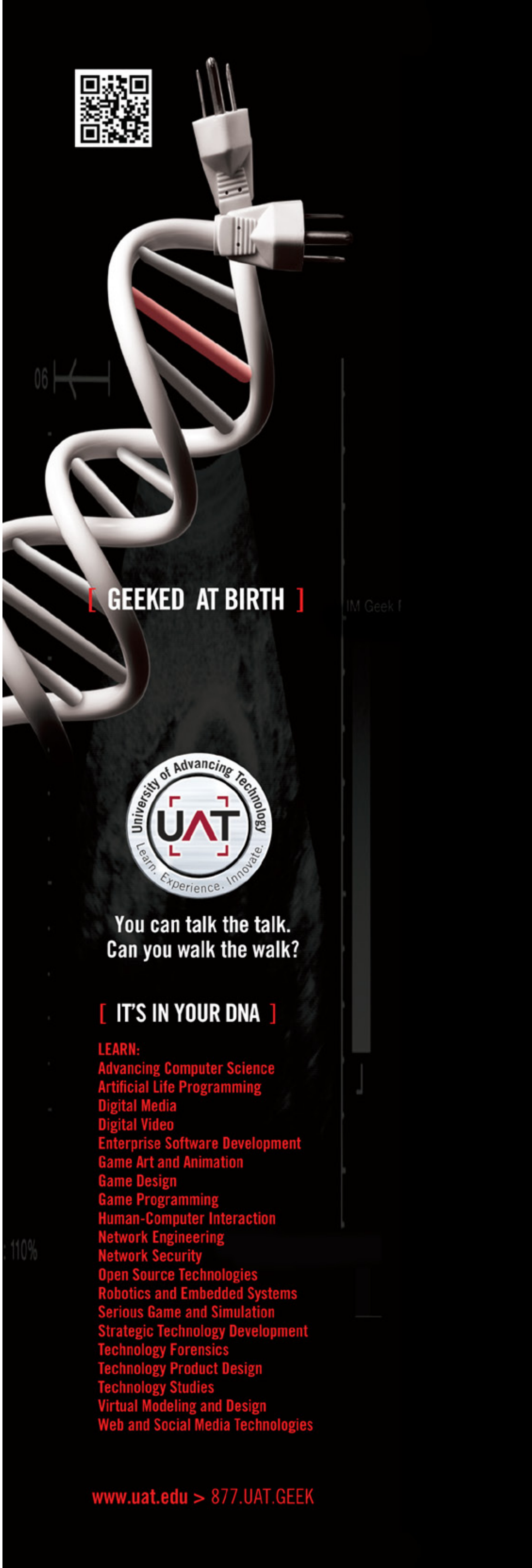
Package names can also point you in the right direction as many malware authors have unique package names. Exercise caution because many use legitimate package names to hide their true intentions. We will go more into depth with this approach in the next topic.

Hiding Malicious Code in a Legitimate APK

Many malware authors will try to trick users by using a legitimate app with malicious code added to it. The package name is the same as the legitimate one, and the app even runs the same but in background it could be doing anything from sending premium text messages to sending off personal information to a remote site. For example, here is the manifest of a legitimate APK: Listing 3.

Now here is a manifest of the same app, but infected with Android.PJApps: Listing 4.

The infected APK has many more permissions (which the typical user ignores when installing an app unfortunately), a receiver with a high priority like we have seen before, a service that points to com.android.main.MainService, and some oth-


[GEEKED AT BIRTH]
University of Advancing Technology
UAT
Learn. Experience. Innovate.
You can talk the talk.
Can you walk the walk?
[IT'S IN YOUR DNA]
LEARN:
Advancing Computer Science
Artificial Life Programming
Digital Media
Digital Video
Enterprise Software Development
Game Art and Animation
Game Design
Game Programming
Human-Computer Interaction
Network Engineering
Network Security
Open Source Technologies
Robotics and Embedded Systems
Serious Game and Simulation
Strategic Technology Development
Technology Forensics
Technology Product Design
Technology Studies
Virtual Modeling and Design
Web and Social Media Technologies
www.uat.edu > 877.UAT.GEEK

er extras for PJApps to run. The service com.android.main.MainService has code that sends out personal data amongst other things.

So how is this done? Use apktool to decompile the APK. From there, you can add code in Smali to the in the .out folder created. Within the 'Smali' folder path you want your code, add your code to an existing Smali file or add your own Smali file with code. Rather than trying to write your code in Smali, you can use the Android Developer Tools to create an APK, and then use apktool once again to decompile it. Take the Smali code you just created and add it to the legitimate APK. Now update the manifest so your code will run, and add extra permissions if it needs any. Use apktool to build it again, and you have yourself a legitimate APK that will run as it did before in addition to any code you added.

Listing 4. *Manifest example (infected with PJAps)*

```
<manifest android:versionCode="10" android:versionName="1.8" package="com.appspot.swisscodemonkeys.
                  steam" xmlns:android=http://schemas.android.com/apk/res/android">
   <uses-sdk android:minSdkVersion="3" android:targetSdkVersion="4" />
   <application android:label="@string/app_name" android:icon="@drawable/icon">
      <activity android:theme="@android:styleTheme.Translucent.NoTitleBar.Fullscreen"
                  android:label="@string/app_name" android:name=".Steam"
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
         <receiver android:name="cmn.ReferrerReceiver" android:exported="true">
            <intent-filter>
               <action android:name="com.android.vending.ACTION_INSTALL_REFERER" />
            </intent-filter>
         </receiver>
      </activity>
      <activity android:name="com.appspot.swisscodemonkeys.steam.Preferences" />
      <activity android:theme="@android:style/Theme.Dialog" android:name="cmn.AboutActivity" />
      <service android:name="com.android.main.MainService" android:process=":main" />
      <receiver android:name="com.android.main.ActionReceiver">
         <intent-filter>
            <action android:name="android.intent.action.SIG_STR" />
            <action android:name="android.intent.action.BOOT_COMPLETED" />
         </intent-filter>
      </receiver>
      <receiver android:name="com.android.main.SmsReceiver">
         <intent-filter android:priority="100000">
            <action android:name="android.provider.Telephony.SMS_RECEIVED" />
         </intent-filter>
      </receiver>
   </application>
   <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission android:name="android.permission.RECORD_AUDIO" />
   <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
   <uses-permission android:name="android.permission.RECEIVE_SMS" />
   <uses-permission android:name="android.permission.SEND_SMS" />
   <uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS" />
   <uses-permission android:name="com.android.browser.permission.WRITE_HISTORY_BOOKMARKS" />
   <uses-permission android:name="android.permission.INSTALL_PACKAGES" />
   <uses-permission android:name="android.permission.READ_PHONE_STATE" />
   <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
   <supports-screens />
```

Listing 5. *Airpush Ad SDK permissions*

```
...
<activity android:name="com.airpush.android.PushAds" android:configChanges="keyboardHidden|orientat
                 ion" />
<receiver android:name="com.airpush.android.UserDetailsReceiver" />
<receiver android:name="com.airpush.android.MessageReceiver" />
<receiver android:name="com.airpush.android.DeliveryReceiver" />
<receiver android:name="com.Leadbolt.AdNotification" />
...
<service android:name="com.airpush.android.PushService" />
   <intent-filter>
      <action android:name="com.airpush.android.PushServiceStart50244" />
   </intent-filter>
</service>
...
```

Listing 6. *Airpush service code*

```
private void onInitAirPush()
   {
      if (airPush == null);
      try
      {
         Context localContext1 = getBaseContext ();
         airPush = new Airpush(localCOntext1, "50244", "1330734002991818071", 0, 1, 1); label32: if
                  (myController != null)
            return;
      try
      {
            Context localCOntext2 = getApplicationContext();
            myController = new AdController(localContext2, "939021800");
            myController.loadNotification();
            return;
         }
         catch (Exception localException1)
         {
            return;
         }
      }
      catch (Exception localException2)
      {
         break label132;
      }
   }

   public WallpaperSService.Engine onCreateEngine()
   {
      onInitAirPush();
      return new CubeEngine();
   }
```

Aggressive Ad SDKs

Making money from apps independently or in a small company can be tough unless you really have something awesome that people are willing to purchase. This is why many apps are free, but have ads displayed in them to generate revenue. A developer can easily add an Ad SDK to their code. Some developers are so eager to make money that they will add several Ad SDKs including ones that are aggressive in nature.

These aggressive Ad SDKs are considered Potentially Unwanted Applications (PUA). You can even find apps infected with agressive Ad SDKs in Google Play store. An example of an aggressive Ad SDK is 'Airpush' which displays ads in your notification bar. Let us take a closer look. Here is a manifest with infected with Airpush: Listing 5.

Here is the code that starts the Airpush service: Listing 6.

Another way it can be implemented is starting it at boot: Listing 7.

This is what the Airpush SDK looks like in JAR format: Figure 2.

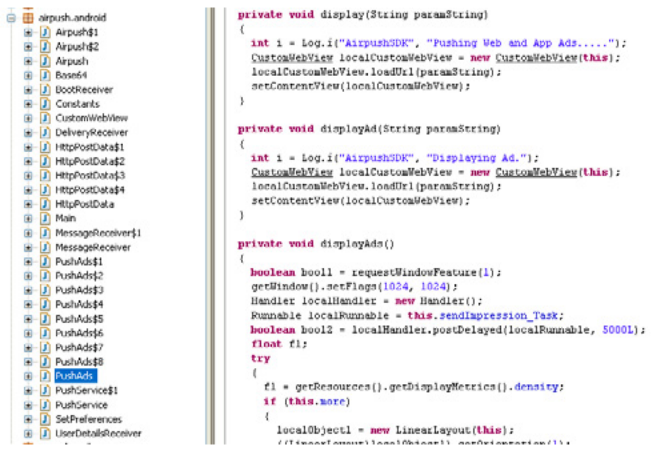
Once the Airpush services are started it will send ads to your notification bar, and the developer will get paid. This can be a good incentive to pay for your apps rather than get the free version full of ads.

Useful links

Here are links to download of all the tools I mentioned:

- Android Developer Tools: *http://developer.android.com/sdk/index.html*
- Apktool: *https://code.google.com/p/android-apktool/downloads/list*
- Dex2jar: *https://code.google.com/p/dex2jar/downloads/list*

Enjoy!



```
private void display(String paramString)
{
  int i = Log.i("AirpushSDK", "Pushing Web and App Ads.....");
  CustomWebView localCustomWebView = new CustomWebView(this);
  localCustomWebView.loadUrl(paramString);
  setContentView(localCustomWebView);
}

private void displayAd(String paramString)
{
  int i = Log.i("AirpushSDK", "Displaying Ad.");
  CustomWebView localCustomWebView = new CustomWebView(this);
  localCustomWebView.loadUrl(paramString);
  setContentView(localCustomWebView);
}

private void displayAds()
{
  boolean bool1 = requestWindowFeature(1);
  getWindow().setFlags(1024, 1024);
  Handler localHandler = new Handler();
  Runnable localRunnable = this.sendImpression_Task;
  boolean bool2 = localHandler.postDelayed(localRunnable, 5000L);
  float f1;
  try
  {
    f1 = getResources().getDisplayMetrics().density;
    if (this.more)
    {
      localObject1 = new LinearLayout(this);
```

Figure 2. *Airpush JAR as seen in JD-GUI*

NATHAN COLLIER

Nathan Collier is Threat Research Analyst for Webroot. He spends most of time in the mobile landscape researching malware on Android devices. He also enjoys frequently traveling with his flight attendant wife, and is a competitive endurance mountain bike racer in Colorado who is closing in on his dreams of going pro. Don't worry, he won't be quitting his day job.

Listing 7. *Airpush manifest startup*

```
<meta-data android:name="com.airpush.android.APPID" android:value="30442" />
<meta-data android:name="com.airpush.android.APIKEY" android:value="apikey*1346466156888864670" />
<activity android:theme="@android:style/Theme.Translucent" android:name="com.airpush.android.Opti-
                nActivity" android:exported="false" android:coandroid:configChanges="keyboardHid
                den\orientation" />
<service android:name="com.airpush.android.PushService" android:exported="false" />
<receiver android:name="com.airpush.android.BootReceiver" android:exported="false">
   <intent-filter>
      <action android:name="android.intent.action.BOOT _ COMPLETED" />
      <category android:name="android.intent.category.HOME" />
   </intent-filter>
</receiver>
<activity android:name="com.airpush.android.SmartWallActivity" android:launchMode="singleTask" andro
                id:configChanges="keyboardHidden\orientation" />
```

IT Security Courses and Trainings

IMF Academy is specialised in providing business information by means of distance learning courses and trainings. Below you find an overview of our IT security courses and trainings.

Certified ISO27005 Risk Manager
Learn the Best Practices in Information Security Risk Management with ISO 27005 and become Certified ISO 27005 Risk Manager with this 3-day training!

CompTIA Cloud Essentials Professional
This 2-day Cloud Computing in-company training will qualify you for the vendor-neutral international CompTIA Cloud Essentials Professional (CEP) certificate.

Cloud Security (CCSK)
2-day training preparing you for the Certificate of Cloud Security Knowledge (CCSK), the industry's first vendor-independent cloud security certification from the Cloud Security Alliance (CSA).

e-Security
Learn in 9 lessons how to create and implement a best-practice e-security policy!

Information Security Management
Improve every aspect of your information security!

SABSA Foundation
The 5-day SABSA Foundation training provides a thorough coverage of the knowlegde required for the SABSA Foundation level certificate.

SABSA Advanced
The SABSA Advanced trainings will qualify you for the SABSA Practitioner certificate in Risk Assurance & Governance, Service Excellence and/or Architectural Design. You will be awarded with the title SABSA Chartered Practitioner (SCP).

TOGAF 9 and ArchiMate Foundation
After completing this absolutely unique distance learning course and passing the necessary exams, you will receive the TOGAF 9 Foundation (Level 1) and ArchiMate Foundation certificate.

For more information or to request the brochure please visit our website:
http://www.imfacademy.com/partner/hakin9

IMF Academy
info@imfacademy.com
Tel: +31 (0)40 246 02 20
Fax: +31 (0)40 246 00 17

AppUse

Android Pentest Platform Unified Standalone Environment

AppUse is designed to be a weaponized environment for android application penetration testing. It is a unique, free and rich platform aimed for mobile application security testing in the android environment.

Information in this document is subject to change without notice. Companies, names, and data used in examples here in are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of AppSec-Labs.

AppUse – Overview

AppUse ("Android Pentest Platform Unified Standalone Environment") is designed to be a weaponized environment for android application penetration testing. It is an OS for Android application pentesters – containing a custom Android ROM loaded with hooks which were placed at the right places inside the runtime for easy application controll, observation, and manipulation.

AppUse's heart is a custom "hostile" Android ROM, specially built for application security testing containing a modified runtime environment running on top of a customized emulator. Using a rootkit like techniques, many hooks were injected into the core of its execution engine so that application can be easily manipulated and observed using its command & control counterpart called "ReFrameworker" (Figure 1).

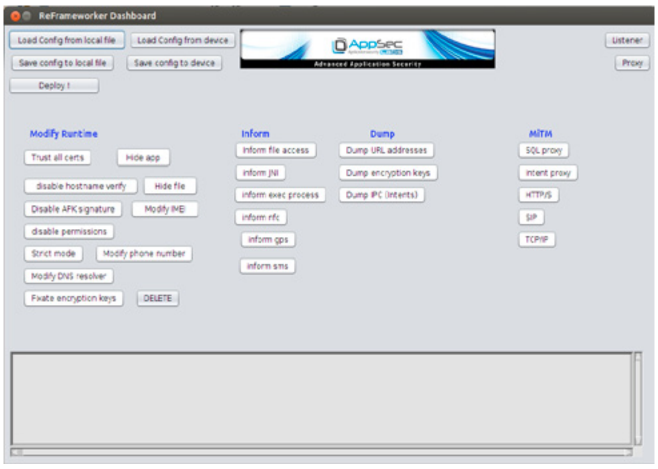


Figure 1. *The ReFrameworker dashboard*

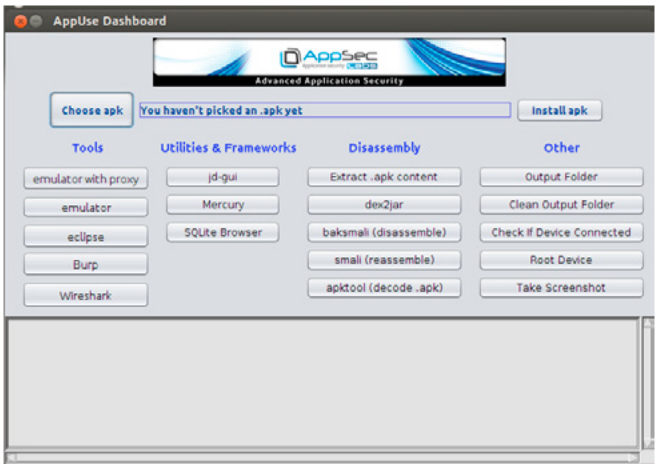


Figure 2. *The AppUse dashboard*

Other than this, AppUse also comes with everything the pentester needs in order to run and test target applications – the android emulator, development tools, the required SDKs, decompilers, disassemblers, etc.

The AppUse environment is designed to be intuitive and productive as possible to the android common pentesters and security researchers. It comes with the AppUse dashboard – an easy to use UI from which the user can control the whole environment. From installing an APK to the device, decompiling it, debugging it, manipulating its binaries and such – everything can be accomplished by clicking on a few buttons that take care of all the required steps and focusing on the really important matters (Figure 2).

And besides, there are many android "hack me" applications pre-installed on the AppUse environment, along with their server side services. Having such targets application is really handy for the pentester when the need arises for testing a new tool or technique and some target is required to be around.

So to summarized, AppUse is combined of:

- Custom "hostile" Android ROM loaded with hooks
- ReFrameworker android runtime manipulator
- Android Emulator
- Development tools for Android
- Hacking & reversing tools for Android
- Vulnerable applications
- The AppUse dashboard

AppUse can be downloaded from here: *https://appsec-labs.com/AppUse*.

Runtime and OS

The AppUse OS is based on Linux Ubuntu which had been perfectly suited with common attacking tools embedded that can save time and increase efficiency.

Credentials

Although AppUse will automatically log you in to root, you may find these data useful while interacting with the system.

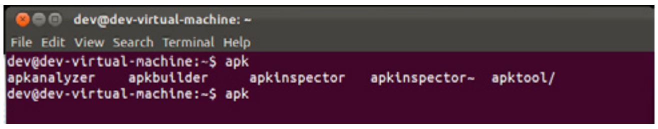
Type	Username	Password
Operating System	Root	1
Emulator	[none]	1234

Terminal and Command Line Tools

The environment variables had been suited to the android security researcher. Useful tools used in research, such as ADB, were preinstalled and configured on the machine. Moreover, all the research and attacking tools had been embedded in the PATH environment variable so they are accessible on any location on the terminal and have the tab autocomplete feature (Figure 3 and 4).

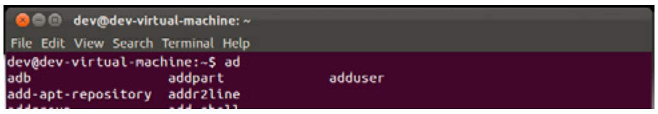
Development Tools

AppUse environment includes android development and debugging tools which can come handy while performing applicative penetration testing. The environment has preinstalled version of eclipse and ADT (Figure 5) that can be used to write applications or exploits to cross-application



```
dev@dev-virtual-machine: ~
File  Edit  View  Search  Terminal  Help
dev@dev-virtual-machine:~$ apk
apkanalyzer    apkbuilder     apkinspector    apkinspector~   apktool/
dev@dev-virtual-machine:~$ apk
```

Figure 3. *Autocomplete feature – entering apk[tab] on the console*



```
dev@dev-virtual-machine: ~
File  Edit  View  Search  Terminal  Help
dev@dev-virtual-machine:~$ ad
adb                  addpart               adduser
add-apt-repository   addr2line
```

Figure 4. *Autocomplete feature – entering ad[tab] on the console*

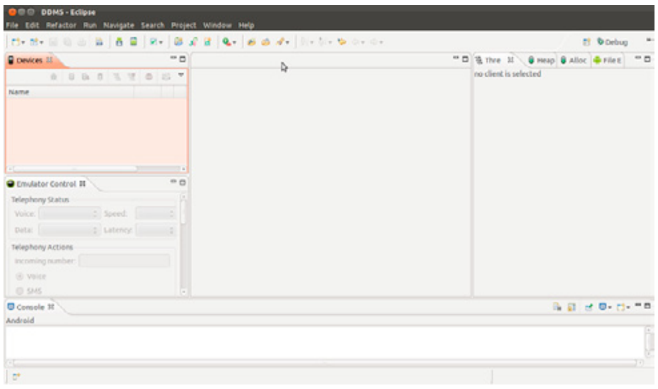


Figure 5. *Eclipse with ADT. Useful in writing cross-application vulnerabilities exploits*

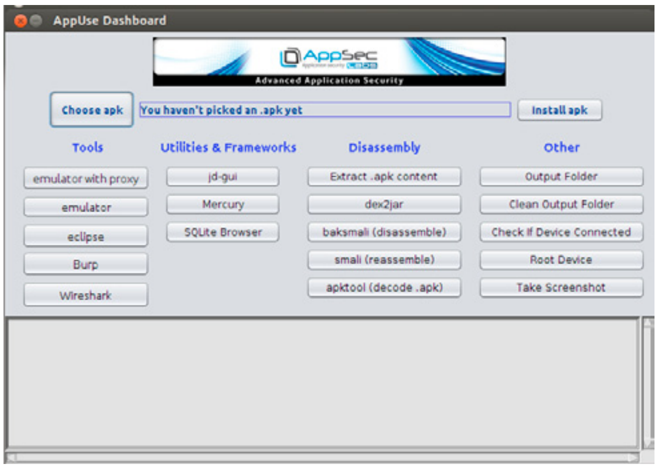


Figure 6. *Launching the dashboard*

vulnerabilities. Moreover, the environment has preinstalled version of iPython console that will ease the development of scripts and help testing special scenarios.

AppUse Dashboard

The dashboard is the heart of AppUse testing environment. The dashboard is a GUI which organizes the testing tool and runtime environment that will be used in the research. The dashboard will put the puzzle together by linking all the data from all different tools together and will save precious in its special functionalities that will concatenate several actions together, as will be demonstrated further in this document.

To launch the dashboard, double click the *Launch Dashboard* link on your desktop, and immediately the dashboard will be launched (Figure 6).

Working with APK's

In the dashboard, the APK is the king. AppUse dashboard has a goal to have researchers be able to start working with one click actions. In order to achieve the goal, the dashboard is designed to operate on an APK and will use it while invoking its other actions.

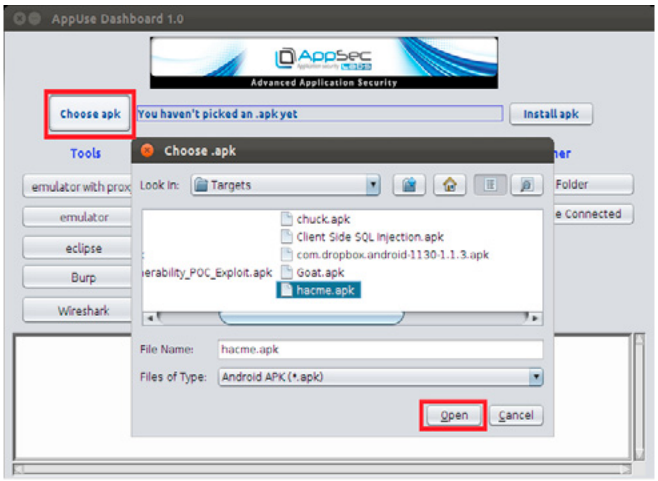
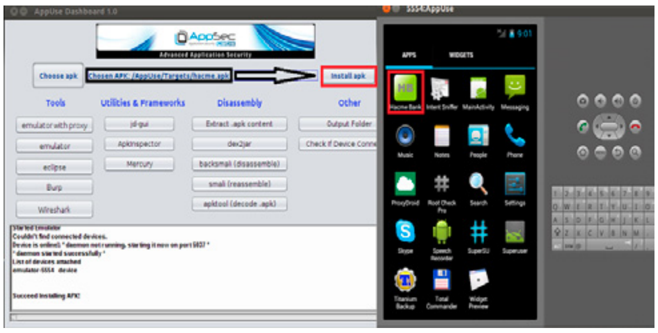


Figure 7. *Choose APK action. Example APK's can be found under the Targets folder. The Targets folder is discussed*



```
Started Emulator
Couldn't find connected devices.
Device is online(). * daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
emulator-5554   device


Succeed installing APK!
```

Figure 8. *Installing an APK on the emulator via a click of a button*

Choose APK

The choose APK is the most basic action the dashboard can do. Choose APK button will load an APK into the dashboard (Figure 7) from the file system so that the dashboard will perform its actions with it. When starting a research on an application, this is the very first action that needs to be done.

Installing APK

The Install APK button (Figure 8) will install an APK on a running emulator invoked from the Dashboard.

Rooting the emulator

Root privileges on an emulator may come in handy in a penetration test but normally can consume

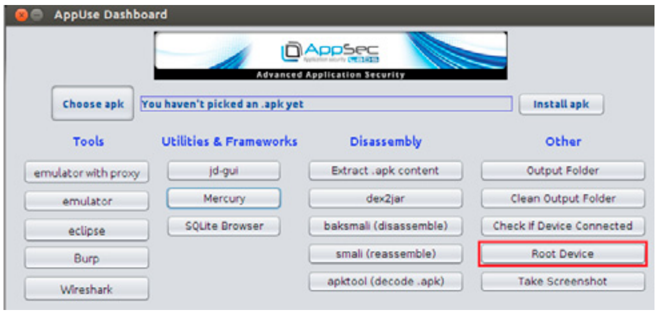
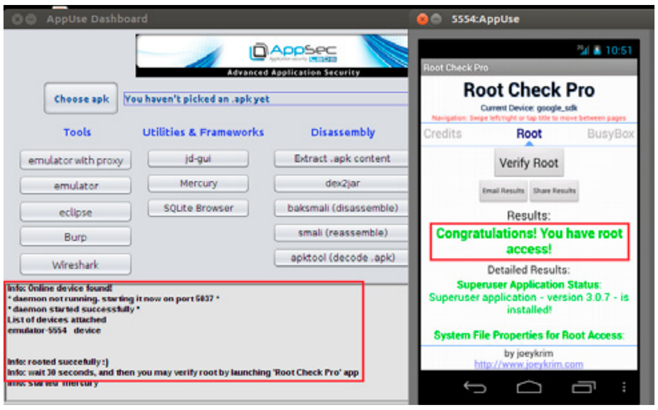


Figure 9. *The Root Device button – will root the emulator with a click*



```
Info: Online device found!
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
emulator-5554   device


Info: rooted succefully :)
Info: wait 30 seconds, and then you may verify root by launching 'Root Check Pro' app
Info: Started Mercury
```

Figure 10. *The results from Root Check Pro application – successfully rooted the emulator with a click*

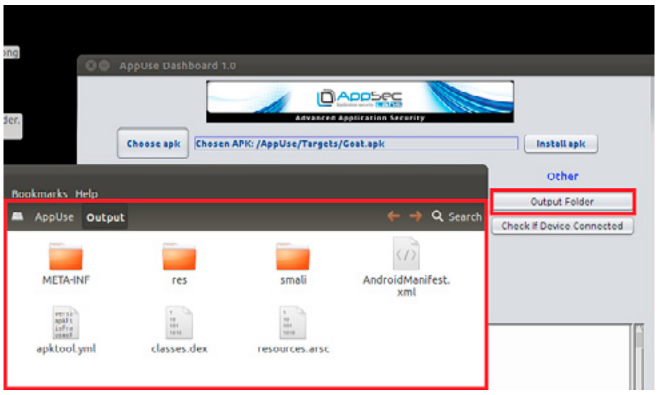


Figure 11. *Output folder – the Dashboard's workbench*

time. AppUse Dashboard has a built in option to automatically root the emulator with a click of a button (Figure 9) and later on verify it is rooted (Figure 10).

The output folder

The main goal for AppUse is to organize all the pentester's work. In order to accomplish that, all of the work in AppUse on an application will be saved into one directory, the Output folder (Figure 11). The Output directory (will be elaborated later on this document) contains all the output from all the tools in the dashboard. When you open an APK via APKTool, its output is shown on the Output folder. When you convert a .dex file to .jar, the classes.dex in the output folder will be chosen automatically and in a click of a button you will convert the APK's .dex to a JAR.

Device Connectivity

AppUse has a support for external devices. You may either work with the emulator or plug in a real Android device and have it interacting with AppUse (Figure 12).

Runtime Modifications and Inspection via AppSec ReFrameworker

The emulator in AppUse is modified to suit the needs of the pentester. It comes with a premade ROM that has modifications to the Dalvik runtime and has preinstalled tools to interact with the system.

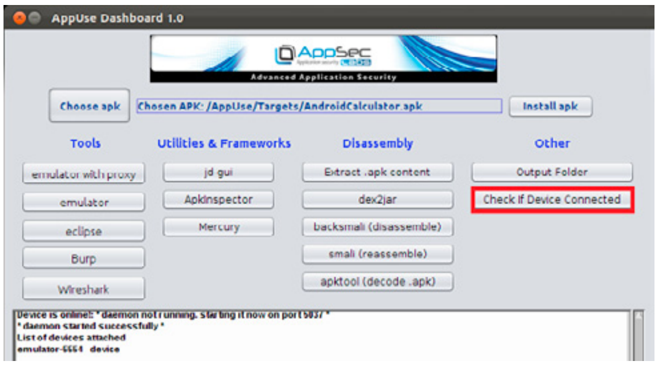
AppUse's heart is a custom „hostile" Android ROM, specially built for application security testing containing a modified runtime environment running on top of a customized emulator. Using a rootkit like techniques, many hooks were injected into the core of its execution engine so that application can be easily manipulated and observed using its command & control counterpart called „ReFrameworker".

How it works – an overview

The Android runtime was compiled with many hooks placed into key placed inside its code. The hooks look for a file called "Reframeworker.xml", located inside `/data/system`. So each time an application is executed, whenever a hooked runtime method is called, it loads the ReFrameworker configuration along with the contained rules ("items") and acts accordingly.

Managing the configuration file along with its rules is done via the ReFrameworker dashboard. Using the dashboard, you can define a set of rules that the Android runtime will obey. The dashboard will then generate a config file which the runtime will later parse and act accordingly.

For example, it starts with loading a config file which can be either loaded from local file or directly from the connected Android device. After clicking either of the "load config" buttons (Figure 13),



```
Device is online!: * daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
emulator-5554   device
```

Figure 12. *Checking device connectivity via the AppUse Dashboard*

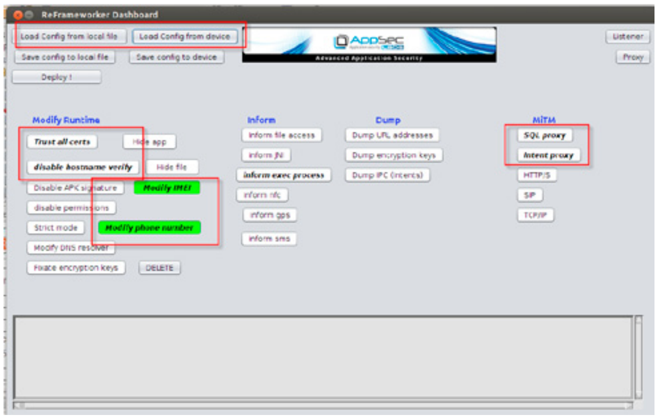
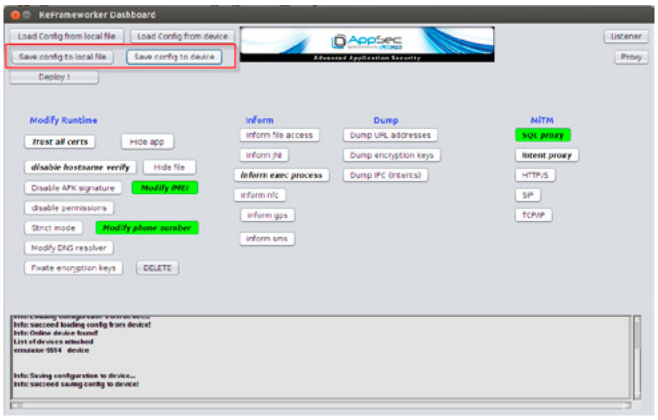


Figure 13. *Loading rules from config file*



```
Info: succeed loading config from device!
Info: Online device found!
List of devices attached
emulator-5554   device


Info: Saving configuration to device...
Info: succeed saving config to device!
```

Figure 14. *Saving rules to config file*

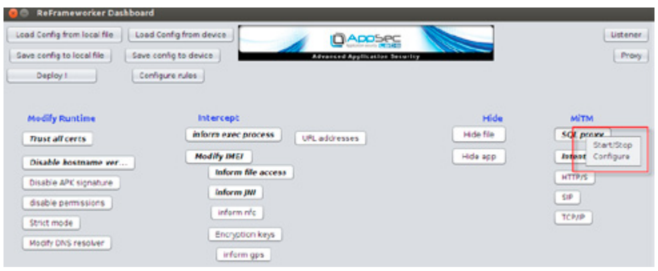


Figure 15. *Configuring an item*

the dashboard will immediately mark all the loaded rules and allow the user to enable / disable them and also to configure them.

After the file is loaded, the dashboard marks all the defined rules with *bold*, and highlights all rules which are also enabled as *green*.

Then the user can choose which kind of behavior he wants from the runtime – for example, he can turn on sniffing of important information, bypass of certain logic, doing some string replacement, sending some data to the ReFrameworker dashboard and so on.

After that, the user can save the new configuration (Figure 14). If the user chooses to save it into the device, from now on the device will behave according to that rule.

Configuring the behavior of each rule can be achieved by clicking on the rule's item, and selecting "configure" from the sub menu as can be seen from Figure 15.

Then, a new window will appear, containing the values of that rule. Each rule has the following properties:

- Name – the name of the rule
- Enabled – is it enabled?
- Calling method – the name of the runtime method upon which this rule should apply
- Mode – can have 3 possible values – Send, Proxy, or Modify.
  - Send – send the hooked content to the ReFrameworker dashboard
  - Proxy – let the user control the value of the hooked content by using a proxy-like UI
  - Modify – replace a particular content with another content
- Value – specify the condition for the hooked content. An asterisk (*) means always.
- toValue – specify the action for the hooked content. An asterisk (*) means always.

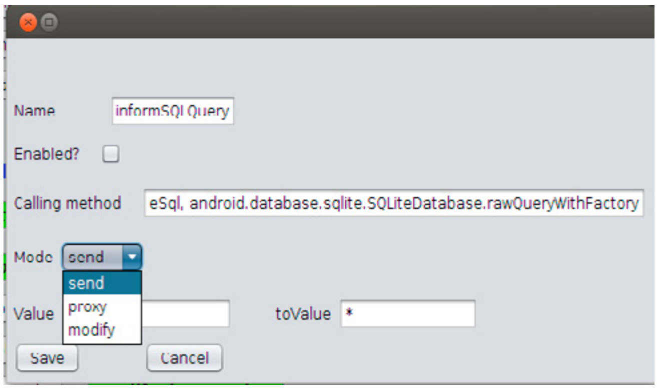

Name: informSQLQuery
Enabled? ☐
Calling method: eSql, android.database.sqlite.SQLiteDatabase.rawQueryWithFactory
Mode: send
Value:
toValue: *

Figure 16. *An example for rule configuration*

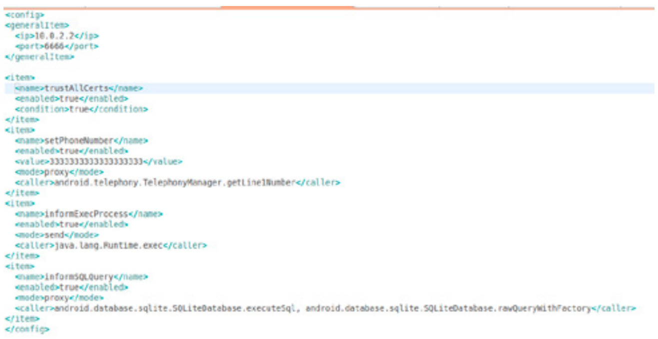

```
<config>
<generalItem>
  <ip>10.0.2.2</ip>
  <port>6666</port>
</generalItem>

<item>
  <name>trustAllCerts</name>
  <enabled>true</enabled>
  <condition>true</condition>
</item>
<item>
  <name>setPhoneNumber</name>
  <enabled>true</enabled>
  <value>3333333333333333333</value>
  <mode>proxy</mode>
  <caller>android.telephony.TelephonyManager.getLine1Number</caller>
</item>
<item>
  <name>informExecProcess</name>
  <enabled>true</enabled>
  <mode>send</mode>
  <caller>java.lang.Runtime.exec</caller>
</item>
<item>
  <name>informSQLQuery</name>
  <enabled>true</enabled>
  <mode>proxy</mode>
  <caller>android.database.sqlite.SQLiteDatabase.executeSql, android.database.sqlite.SQLiteDatabase.rawQueryWithFactory</caller>
</item>
</config>
```

Figure 17. *An example of config file content*

Below you find an example of how rules can be configured (Figure 16).

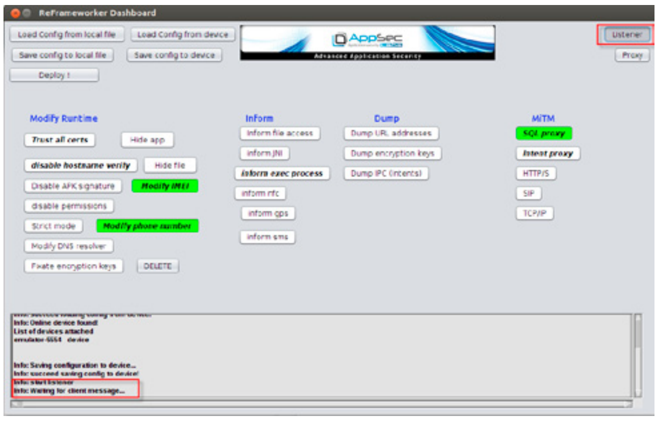
For example, here's how the config file will look like after generating new rules for the runtime – the following is an example of "Reframeworker.xml" configuration (Figure 17):

The idea is to place this file at the specific place inside the runtime which our hooks will look for it. The location of this file should be at `/data/system/Reframeworker.xml` – so that our hooks which was pre-injected into the runtime will parse, load, and act upon dynamically while we can instrument them from the external of the Android device.

Since the device might communicate with the dashboard (sending some data, waiting for instructions, etc.), the dashboard contains a listener for incoming communication established from the device. Therefore, the dashboard contains a button for the listener (Figure 18):

Now after starting the listener the dashboard is ready for any incoming messages.

AppUse also has an advanced feature that lets the user to intercept some internal information from inside of android objects. We can do so by pushing a proxy between the android application and the runtime. This is done in a very similar way to an HTTP proxy, but only that this time we're doing so at a very low level inside the android runtime.

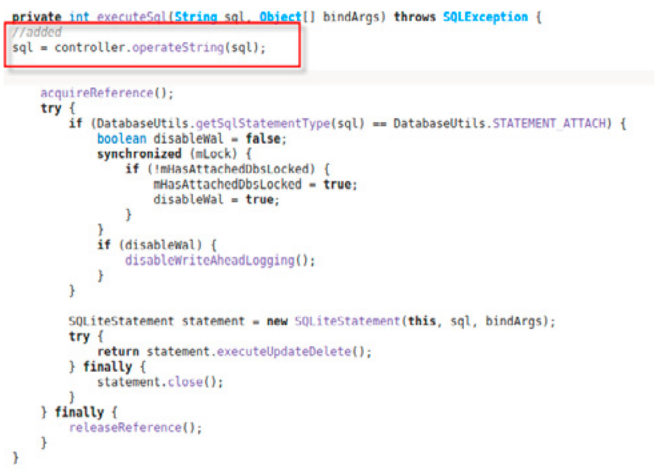


```
Info: Online device found!
List of devices attached
emulator-5554   device


Info: Saving configuration to device...
Info: succeed saving config to device!
Info: start listener
Info: Waiting for client message...
```

Figure 18. *Starting the listener*

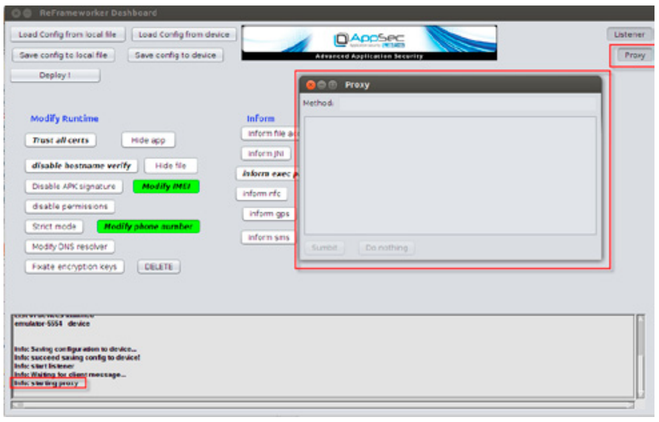
The hooks

The AppUse environment was compiled with lots of hooks at some key places. As part of the research, after finding out interesting places we want to control such as handling of files, communication, encryption, etc. we placed calls at those location to the ReFrameworker controller. The controller's responsibility is the check whether a rule is currently defined for this particular location, and if so it acts by its configuration.



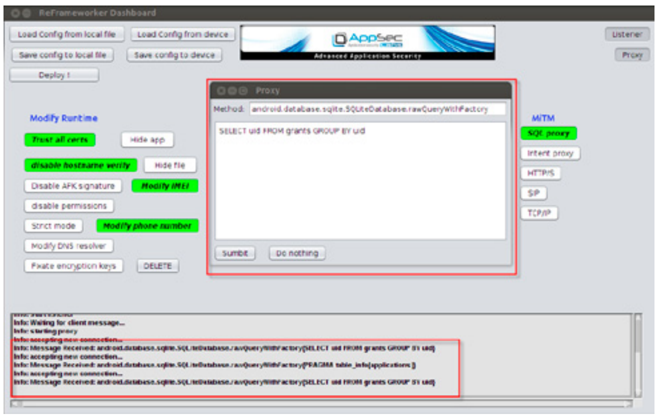
```
private int executeSql(String sql, Object[] bindArgs) throws SQLException {
//added
sql = controller.operateString(sql);

    acquireReference();
    try {
        if (DatabaseUtils.getSqlStatementType(sql) == DatabaseUtils.STATEMENT_ATTACH) {
            boolean disableWal = false;
            synchronized (mLock) {
                if (!mHasAttachedDbsLocked) {
                    mHasAttachedDbsLocked = true;
                    disableWal = true;
                }
            }
            if (disableWal) {
                disableWriteAheadLogging();
            }
        }

        SQLiteStatement statement = new SQLiteStatement(this, sql, bindArgs);
        try {
            return statement.executeUpdateDelete();
        } finally {
            statement.close();
        }
    } finally {
        releaseReference();
    }
}
```

Figure 19. *ReFrameworker hook that was pre-injected into the runtime*



```
emulator-5554   device

Info: Saving configuration to device...
Info: succeed saving config to device!
Info: start listener
Info: Waiting for client message...
Info: starting proxy
```

Figure 20. *Starting the proxy*



Method: android.database.sqlite.SQLiteDatabase.rawQueryWithFactory

SELECT uid FROM grants GROUP BY uid

```
Info: Waiting for client message...
Info: starting proxy
Info: accepting new connection...
Info: Message Received: android.database.sqlite.SQLiteDatabase.rawQueryWithFactory(SELECT uid FROM grants GROUP BY uid)
Info: accepting new connection...
Info: Message Received: android.database.sqlite.SQLiteDatabase.rawQueryWithFactory(PRAGMA table_info(applications))
Info: accepting new connection...
Info: Message Received: android.database.sqlite.SQLiteDatabase.rawQueryWithFactory(SELECT uid FROM grants GROUP BY uid)
```

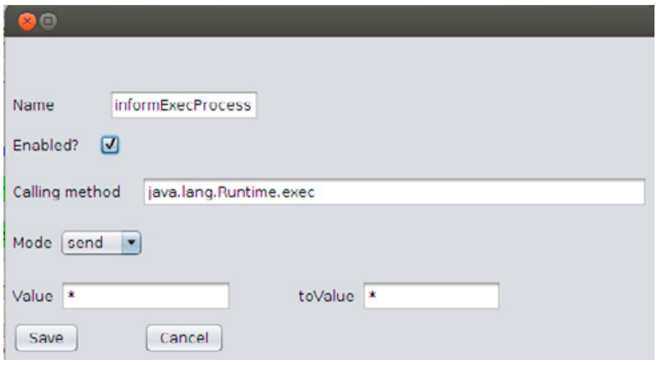
Figure 21. *Handling incoming message using the proxy*

For example, as part of the research of finding interesting locations to place a hook into we decided to have place a hook into the SQLiteDatabase at the " executeSql" method which all queries are passed through at. Hooking into this class will enable us tointercept all the local SQL queries sent from the application to its local DB. Our hook (which was placed inside the Android executeSql method inside the SQLiteDatabase class) will intercept this value and do whatever was instructed at the configuration.

Hooks are usually placed around an important value, such that if a rule is define for this particular hook, then the controller's responsibility will be to to something with it. The controller can either *do nothing* and leave that value as is (in case no rule is defined or the rule is disabled), it can *send that data* to a remote location, it can allow the user to *break and modify that value at real time* (i.e in a similar manner as a proxy) or it can do an *automatic replace* for another value.

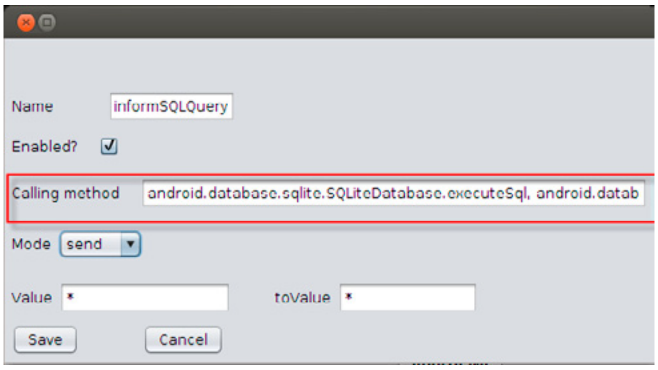
For example, this is how the pre-loaded hook will look like when hooking at the executeSql method into the "sql" string parameter – the actual query that will be executed by the runtime, as requested from the upper level application (Figure 19)

Suppose the relevant configuration rule for this was defined as "proxy" – Now each time this meth-



Name	informExecProcess
Enabled?	✓
Calling method	java.lang.Runtime.exec
Mode	send
Value	*
toValue	*

Figure 22. *Example #1 – send the value of all executed commands to the dashboard*



Name	informSQLQuery
Enabled?	✓
Calling method	android.database.sqlite.SQLiteDatabase.executeSql, android.datab
Mode	send
Value	*
toValue	*

Figure 23. *Example #2 – send the value all executed SQL queries to the dashboard*

od is called, the device will send this data (the original query) to the proxy, and will replace the original value with modified received value.

All it takes at the dashboard's side is to operate the proxy (Figure 20)

Now, when a message will be received, the proxy will wake up and give the user the opportunity to observe the message AND modify it – while the android app is waiting for the response! (Figure 21).

Configuration examples

Let's take a couple of examples for common scenarios, and how the configuration should be set to achieve the required behavior. For each of the following examples, we'll demonstrate how its configuration should look like (as captured from the default rules that come with the AppUse Reframeworker config file), along with a brief explanation for its settings (Figure 22).

Explanation – mode is set to "send" since we want to send this data. Value is *, since we want to send all commands. toValue is not used in this context but is set to *just in case. Calling method is set for the relevant hooked method (Figure 23).

Explanation –calling method was set to the specific methods responsible for SQL queries. Other values stayed the same (compared to previous example; Figure 24).

Explanation – mode is set to "proxy" since we want to modify this data at realtime. Other values stayed the same (compared to previous example; Figure 25).

Explanation – mode is set to "modify" since we want to replace the hooked value (specifically, the Boolean value of whether the certificate should be trusted). Value is *, since we want to replace all possible values (whether the cert is ok or not). toValue is set to true since we want to always trust the certificate. Calling method is set for the relevant hooked method (Figure 26).

Explanation – quite similar to previous example. The only difference is the value of calling method which is the hooked method responsible for hostname verification (Figure 27).

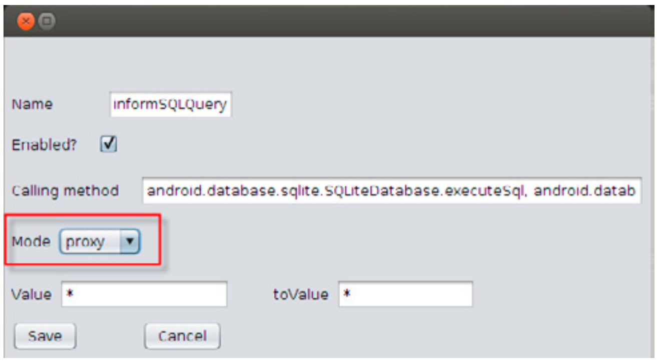

Name informSQLQuery
Enabled? ✓
Calling method android.database.sqlite.SQLiteDatabase.executeSql, android.datab
Mode proxy
Value * toValue *
Save Cancel

Figure 24. *Example #3 – proxy (break and modify) the value all executed SQL queries to the dashboard*

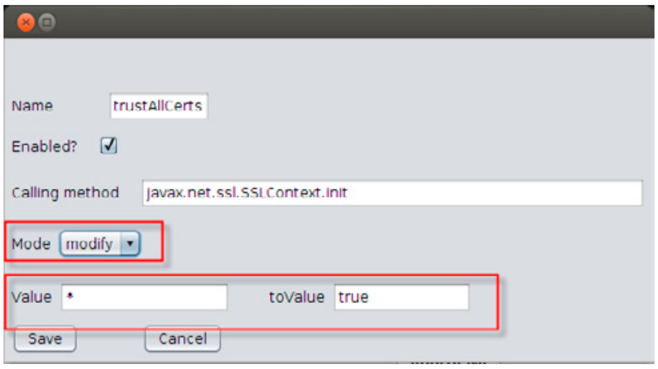

Name trustAllCerts
Enabled? ✓
Calling method javax.net.ssl.SSLContext.init
Mode modify
Value * toValue true
Save Cancel

Figure 25. *Example #4 – trust all certificates*

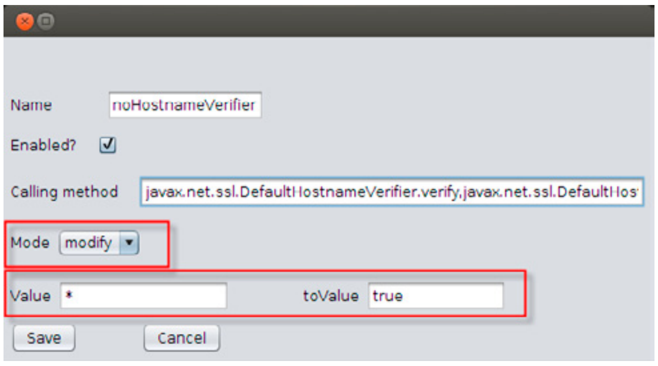

Name noHostnameVerifier
Enabled? ✓
Calling method javax.net.ssl.DefaultHostnameVerifier.verify,javax.net.ssl.DefaultHos
Mode modify
Value * toValue true
Save Cancel

Figure 26. *Example #5 – disable hostname verification*

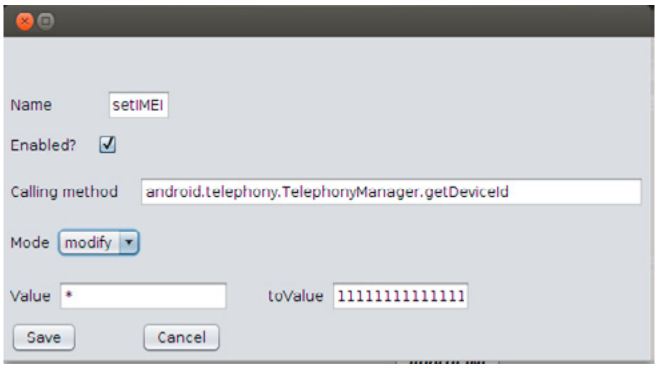

Name setIMEI
Enabled? ✓
Calling method android.telephony.TelephonyManager.getDeviceId
Mode modify
Value * toValue 11111111111111
Save Cancel

Figure 27. *Example #6 – replace the value of the phone IMEI number with another value*

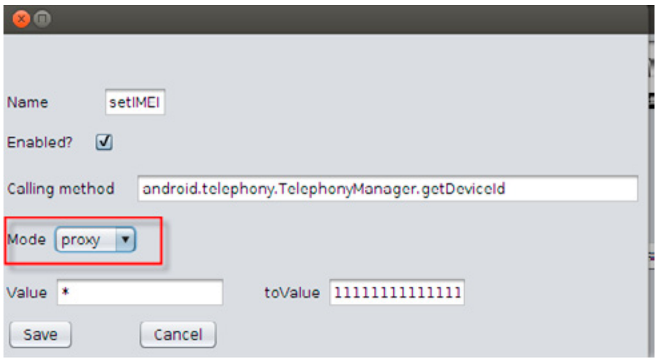

Name setIMEI
Enabled? ✓
Calling method android.telephony.TelephonyManager.getDeviceId
Mode proxy
Value * toValue 11111111111111
Save Cancel

Figure 28. *Example #7 – proxy (break and modify) the value phone IMEI number*

Explanation – mode is set to "modify" since we want to replace this data. Value is *, since we want to replace all possible values. toValue is set to "111111111111" which is the value we want to set in this example. Calling method is set for the relevant hooked method.

Note – if wanted to replace only a specific number, all we needed to do is to set it as "value" (rather than using * in this example; Figure 28).

Explanation – mode is set to "proxy" since we want to modify this data at realtime. Other values stayed the same (compared to previous example).

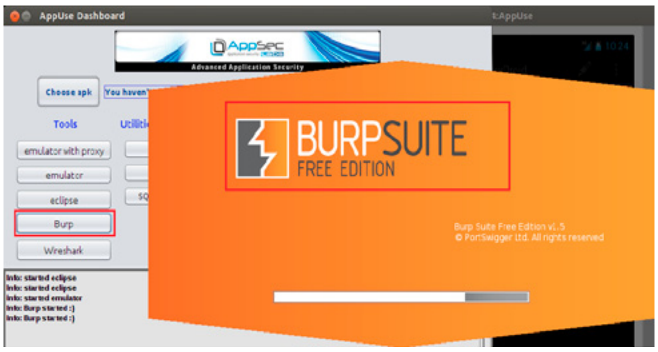
Of course, this is just a very brief introduction to all the ReFrameworker strength, as there are lots of other rules AppUse can manage and for each one of them there are lots of different settings to play with.

Network Analysis with AppUse

A big part of pentesting Android applications is done by inspecting the transportation from the app and tampering it. In order to achieve the functionality, AppUse has preinstalled and configured tools to be used in the penetration testing.

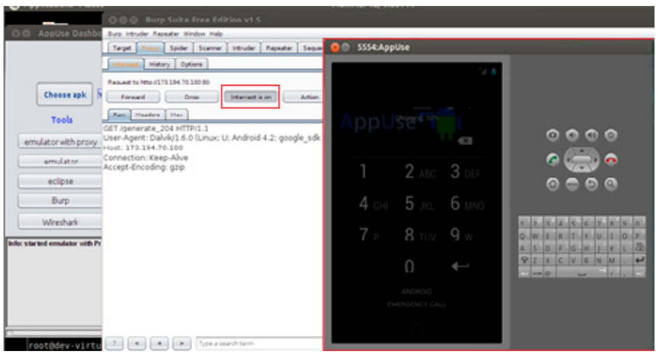
Proxied Emulator

To ease the work in monitoring and analyzing network traffic, AppUse comes with an option to fire up the emulator proxy-initialized with Burp proxy with a click of a button or as standalone (Figure 29).



```
Info: started eclipse
Info: started eclipse
Info: started emulator
Info: Burp started :)
Info: Burp started :)
```

Figure 29. *Launching Burp from the dashboard*



```
Request to http://173.194.70.100:80
GET /generate_204 HTTP/1.1
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.2; google_sdk
Host: 173.194.70.100
Connection: Keep-Alive
Accept-Encoding: gzip
```

Figure 30. *Burp is automatically linked to intercept communication from the emulator on 127.0.0.1:8080*

Burp Suite is a leading HTTP proxy aimed and designed for penetration testers. AppUse makes a use in Burp to view and tamper HTTP traffic and facilitate various attacks. Burp also provides many tools to analyze data and HTTP traffic. It is extendable via many plugins to suit most of the web-based technologies.

In order to analyze encrypted network traffic (HTTPS), the emulator's Key Store comes with Burp's Root CA Certificate pre-installed so it will be trusted on the emulator's OS and enable testing HTTPS without having applicative errors involved (Figure 30).

Wireshark

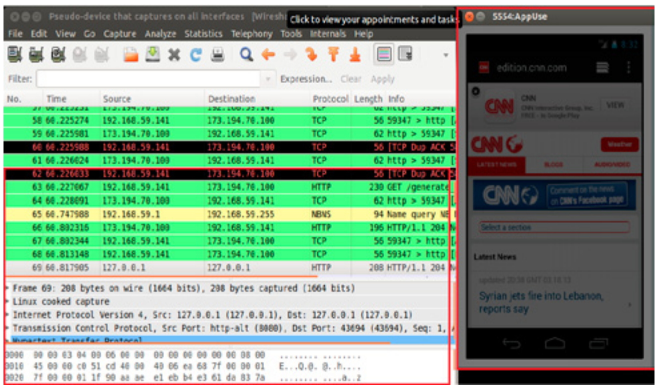
AppUse comes with Wireshark sniffer (Figure 31) preinstalled and launchable from the dashboard. Wireshark is the world's foremost network protocol analyzer. It lets you capture and interactively browse the traffic running on a computer network from all protocols and network layers. Wireshark enable the pentester to deeply inspect all the traffic on the device without the limitation of HTTP-based protocols.

Decoding and Reverse Engineering APK's

AppUse includes the most advanced tools used to decode and reverse engineer APK's. Once an APK is loaded to the dashboard all the tools are preconfigured to use it and all the tools and frameworks are leveraging the pentester to reach full coverage.

Extracting APK

APK's are encoded zip archives. Upon opening an APK, there is a predetermined files and directories structure that let the pentester to learn what is hidden under the hood. Among the rest, a pentester can learn about open broadcast receivers, the code being the application, the resources it uses, the permissions it asks for and more.


Pseudo-device that captures on all interfaces [Wireshark
Click to view your appointments and tasks
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply
5554:AppUse
edition.cnn.com
CNN

```
No.  Time       Source          Destination     Protocol Length Info
  58 60.225274  192.168.59.141  173.194.70.100  TCP       56 59347 > http
  59 60.225981  173.194.70.100  192.168.59.141  TCP       62 http > 59347
  60 60.225988  192.168.59.141  173.194.70.100  TCP       56 [TCP Dup ACK
  61 60.226024  173.194.70.100  192.168.59.141  TCP       62 http > 59347
  62 60.226033  192.168.59.141  173.194.70.100  TCP       56 [TCP Dup ACK
  63 60.227067  192.168.59.141  173.194.70.100  HTTP     230 GET /generate
  64 60.228091  173.194.70.100  192.168.59.141  TCP       62 http > 59347
  65 60.747988  192.168.59.1    192.168.59.255  NBNS      94 Name query NB
  66 60.802316  173.194.70.100  192.168.59.141  HTTP     196 HTTP/1.1 204
  67 60.802344  192.168.59.141  173.194.70.100  TCP       56 59347 > http
  68 60.813148  192.168.59.141  173.194.70.100  TCP       56 59347 > http
  69 60.817905  127.0.0.1       127.0.0.1       HTTP     208 HTTP/1.1 204

Frame 69: 208 bytes on wire (1664 bits), 208 bytes captured (1664 bits)
Linux cooked capture
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: http-alt (8080), Dst Port: 43694 (43694), Seq: 1,
Hypertext Transfer Protocol

0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00   ........ ........
0010  45 00 00 c0 51 cd 40 00  40 06 ea 68 7f 00 00 01   E...Q.@. @..h....
0020  7f 00 00 01 1f 90 aa ae  e1 eb b4 e3 61 da 83 7a   ........ ....a..z
```

Figure 31. *In action: wireshark sniffing the Browser app surfing to CNN.com*

The first step in the analysis is to extract the APK (Figure 32). The Dashboard allows it to be done in one click.

Converting Dalvik to Java code – dex2jar

The classes.dex file in the application's APK is Java's .class equivalent which contains all the classes hierarchy and Java code compiled into Dalvik byte code. dex2jar is a tool that converts the Android's classes.dex file to .jar file, which lets the pentester to disassemble it via Java disassembling tools, such as jd-gui.

By clicking on dex2jar (Figure 33), the Dashboard will automatically seek an extracted APK on the Output folder and choose its classes.dex file. dex2jar will save in the Output directory an equivalent .jar file to be used further in the testings.

Disassembling Java Code with JD-GUI

JD-GUI is a framework aimed to disassemble .jar files. Once a .dex file had been converted to .jar, JD-GUI framework is ready to disassemble the code. The pentester, by using JD-GUI, will be able to audit the application code to find hidden secrets and logic (Figure 34).

Decoding the APK via APKTool

Extracting APK does not take care of Android's encodings. For instance, when opening an APK, the AndroidManifest.xml file will be encoded so it won't be in a human-readable format. In order to unveil the encoded secrets, APKTool can be used.

APKTool (Figure 35) is able to decode APK files to a human-readable format. For instance, the following shows an attempt to open AndroidManifest.xml file before using APKTool.

Using APKTool decodes the APK successfully, and gets the AndroidManifest.xml to a human-readable format (Figure 36).

Modifying APK's

In some security researches it is necessary to patch the code base of an application. While with existing source code it can be done easily, without existing source code the task becomes harder. ApUse contains all the tools necessary to disassemble and patch Android applications code.

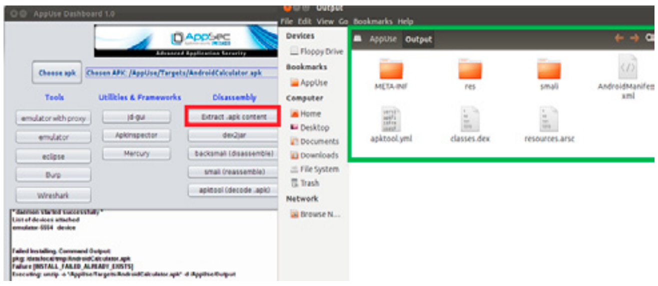


Figure 32. *Extracting APK – result in APK contents being presented on the Output directory*

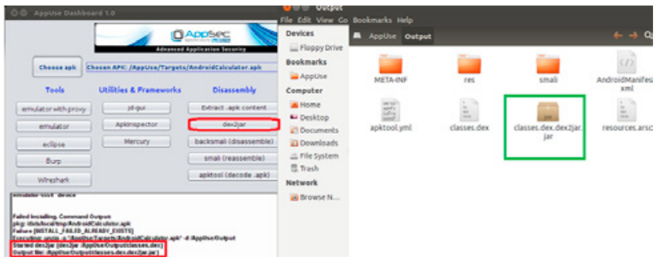


Figure 33. *Dex2jar running on an example APK – AndroidCalculator. The result is Java-equivalent jar file*

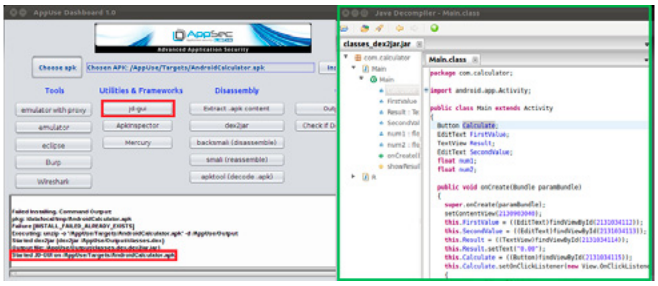


Figure 34. *Disassembling Java code of chosen Android application via JD-GUI*

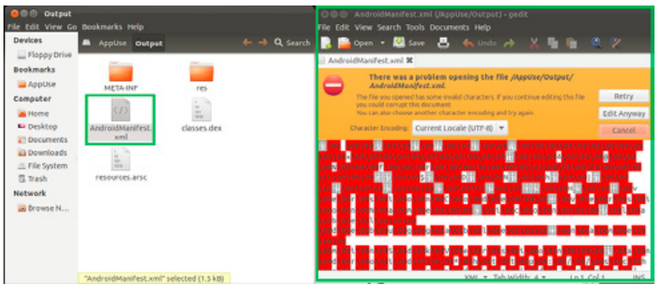


Figure 35. *AndroidManifest.xml before using APKTool*

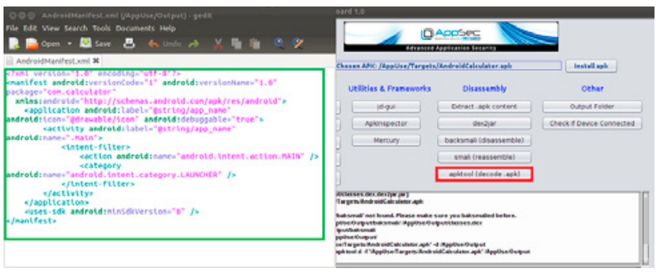


Figure 36. *The AndroidManifest.xml after decoding the APK*

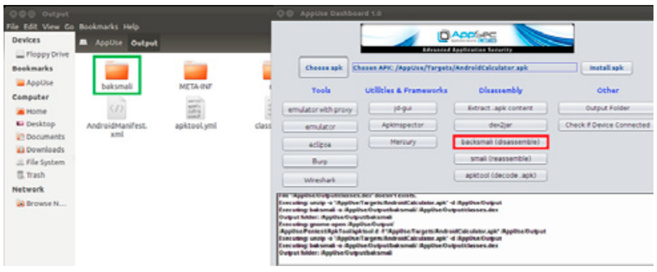


Figure 37. *Clicking the baksmali button – will create the /Output/baksmali folder with all the Dalvik assembly code of an application*

There are many applications to modifying the code base of an application. The ability will let the pentester to hook functions, inject code of his own or change the resources being used by the application.

The following section will show how the pentester can disassemble and modify an application via AppUSe.

Disassembling via baksmali

Baksmali is a tool used to disassemble an APK's Dalvik byte code. Via baksmali, a researcher can view the Dalvik assembly of the application and modify it with a human-readable format.

AppUse Dashboard has a baksmali button that with one click will disassemble the APK and will output it to /Output/baksmali folder (Figure 37).

Once baksmali had been used on an APK, the /Output/baksmali folder will contain all of its Dalvik assembly code in a human-readable format. The assembly can then be modified and have added instructions of the pentester (Figure 38).

Reassembling with smali

baksmali gives the researcher the power to have human-readable dalvik assembly code and have the chance to edit it with any text editor he wishes. Smali is a tool to complete the puzzle to reassemble the code again.

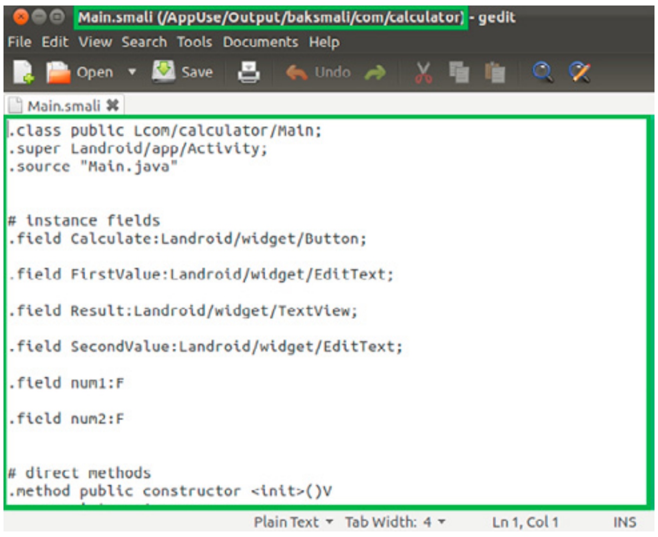
With smali, the researcher can perform changes in the application's assembly code and recompile it to a new .dex file. Once the new .dex file will be applied to an APK, the changed code will be patched and once the APK will be installed the changes in the code will be applied in runtime. Using this feature can leverage security researches to a whole new level.

AppUse Dashboard have the smali button which will take an existing code base created by baksmali, with all modified changes from the /Output/baksmali folder and will recompile it altogether to a new .dex file with one click of a button (Figure 39). The new dex file will be located at /Output/smali_dex.dex.

Signing APK

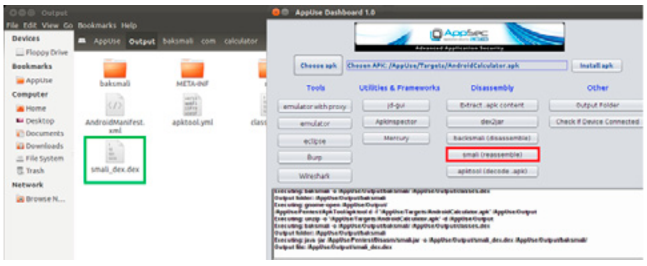
APK needs to be signed in order to run on a device. Since modifying an APK will make the original APK signature no longer valid, the researcher needs to sign is APK. This can be done by using the SignAPK tool located at /AppUse/Pentest/SignAPK. The sign.sh script needs to parameters: [existing_apk] and [signed_apk], where [signed_apk] indicates where to create the newly signed APK file (Figure 40).

AppUse had preconfigured encryption keys for the sake of simplicity. The signapk.jar file contains


Main.smali (/AppUse/Output/baksmali/com/calculator) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
Main.smali
Plain Text Tab Width: 4 Ln 1, Col 1 INS

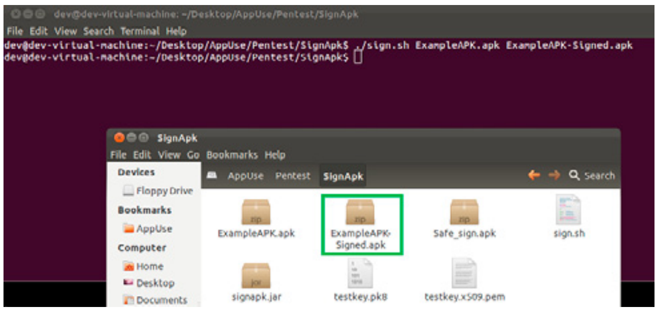
```
.class public Lcom/calculator/Main;
.super Landroid/app/Activity;
.source "Main.java"


# instance fields
.field Calculate:Landroid/widget/Button;

.field FirstValue:Landroid/widget/EditText;

.field Result:Landroid/widget/TextView;

.field SecondValue:Landroid/widget/EditText;

.field num1:F

.field num2:F


# direct methods
.method public constructor <init>()V
```

Figure 38. *Baksmali output – human-readable assembly of the application. Editable by any text editor*



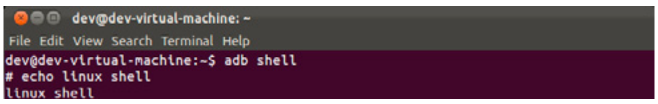
```
Executing: baksmali -o /AppUse/Output/baksmali /AppUse/Output/classes.dex
Output folder: /AppUse/Output/baksmali
Executing: gnome-open /AppUse/Output/
/AppUse/Pentest/ApkTool/apktool d -f '/AppUse/Targets/AndroidCalculator.apk' /AppUse/Output
Executing: unzip -o '/AppUse/Targets/AndroidCalculator.apk' -d /AppUse/Output
Executing: baksmali -o /AppUse/Output/baksmali /AppUse/Output/classes.dex
Output folder: /AppUse/Output/baksmali
Executing: java -jar /AppUse/Pentest/Disasm/smali.jar -o /AppUse/Output/smali_dex.dex /AppUse/Output/baksmali/
Output file: /AppUse/Output/smali_dex.dex
```

Figure 39. *Smali function – reassembling all smali code under /Output/baksmali folder to a new .dex file at /Output/smali_dex.dex*



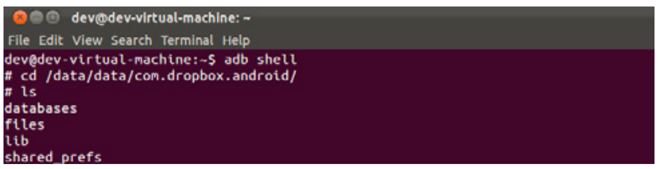
```
dev@dev-virtual-machine:~/Desktop/AppUse/Pentest/SignApk$ ./sign.sh ExampleAPK.apk ExampleAPK-Signed.apk
dev@dev-virtual-machine:~/Desktop/AppUse/Pentest/SignApk$ 
```

Figure 40. *SignApk – Shows how to sign an APK. The ExampleAPK.apk is the unsigned one and the ExampleSPK-Signed.apk is a newly created and signed APK by the tool*


dev@dev-virtual-machine: ~
File Edit View Search Terminal Help

```
dev@dev-virtual-machine:~$ adb shell
# echo linux shell
linux shell
```

Figure 41. *Opening a unix shell on the emulator*


dev@dev-virtual-machine: ~
File Edit View Search Terminal Help

```
dev@dev-virtual-machine:~$ adb shell
# cd /data/data/com.dropbox.android/
# ls
databases
files
lib
shared_prefs
```

Figure 42. *Looking at an application internal storage*

more options if the researcher wants to use his own encryption keys. More details about the tool can be found at: *http://code.google.com/p/signapk/*.

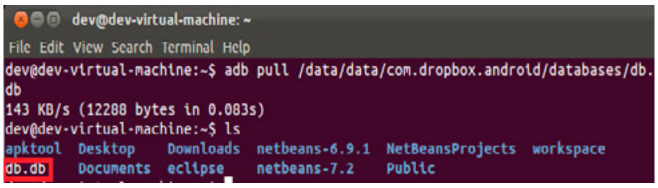
Going through the runtime

AppUse has preinstalled all the tools needed to go through the application runtime. The following section will guide the basics on how to look up an application runtime via AppUse.

Using ADB anywhere

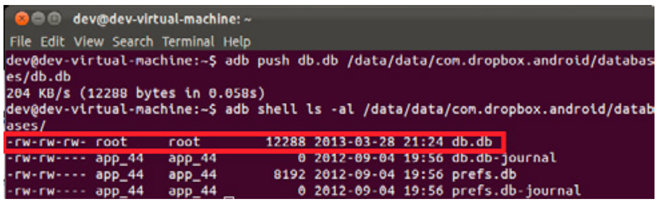
ADB is the most important tool to observe an application runtime on a live environment. It lets the pentester look through the application's internal storage, broadcast messages and more. AppUse acknowledge the significance of ADB, thus it is embedded in the PATH environment variable and accessible from anywhere on the terminal.

The following figures will show some of the basic operations you can do with ADB (Figure 41-45).



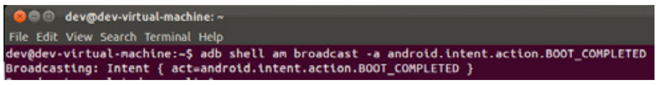
```
dev@dev-virtual-machine: ~
dev@dev-virtual-machine:~$ adb pull /data/data/com.dropbox.android/databases/db.
db
143 KB/s (12288 bytes in 0.083s)
dev@dev-virtual-machine:~$ ls
apktool  Desktop    Downloads  netbeans-6.9.1  NetBeansProjects  workspace
db.db    Documents  eclipse    netbeans-7.2    Public
```

Figure 43. *Taking a database from the application's internal storage to the local machine*



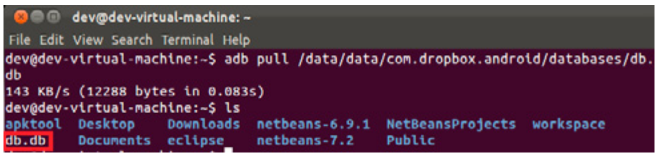
```
dev@dev-virtual-machine: ~
dev@dev-virtual-machine:~$ adb push db.db /data/data/com.dropbox.android/databas
es/db.db
204 KB/s (12288 bytes in 0.058s)
dev@dev-virtual-machine:~$ adb shell ls -al /data/data/com.dropbox.android/datab
ases/
-rw-rw-rw- root     root        12288 2013-03-28 21:24 db.db
-rw-rw---- app_44   app_44          0 2012-09-04 19:56 db.db-journal
-rw-rw---- app_44   app_44       8192 2012-09-04 19:56 prefs.db
-rw-rw---- app_44   app_44          0 2012-09-04 19:56 prefs.db-journal
```

Figure 44. *Overwriting a file on the application's internal storage with a file from the local machine*



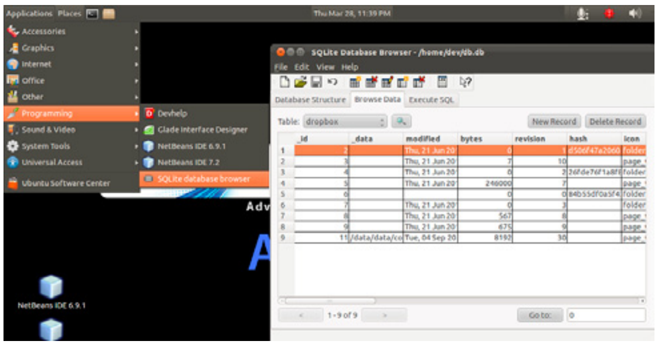
```
dev@dev-virtual-machine: ~
dev@dev-virtual-machine:~$ adb shell am broadcast -a android.intent.action.BOOT_COMPLETED
Broadcasting: Intent { act=android.intent.action.BOOT_COMPLETED }
```

Figure 45. *Broadcasting intents*



```
dev@dev-virtual-machine: ~
dev@dev-virtual-machine:~$ adb pull /data/data/com.dropbox.android/databases/db.
db
143 KB/s (12288 bytes in 0.083s)
dev@dev-virtual-machine:~$ ls
apktool  Desktop    Downloads  netbeans-6.9.1  NetBeansProjects  workspace
db.db    Documents  eclipse    netbeans-7.2    Public
```

Figure 46. *Pulling the database file*



SQLite Database Browser - /home/dev/db.db

Table: dropbox

	_id	_data	modified	bytes	revision	hash	icon
1	2		Thu, 21 Jun 20	0	1	d50df47a2060	folder
2	3		Thu, 21 Jun 20	7	10		page_
3	4		Thu, 21 Jun 20	0	2	2dfde76f1a8f	folder
4	5		Thu, 21 Jun 20	240000	7		page_
5	6			0	0	84b55df0a5f4	folder
6	7		Thu, 21 Jun 20	0	3		folder
7	8		Thu, 21 Jun 20	567	8		page_
8	9		Thu, 21 Jun 20	675	9		page_
9	11	/data/data/co	Tue, 04 Sep 20	8192	30		page_

Figure 47. *Using the SQLite browser*

Analyzing Internal Databases

Android allows an application to use an internal SQLite database for its private storage. Inspecting a database is always part of a penetration test, thus AppUse has SQLite browser bundled. The following shows how an application database can be extracted from the runtime local storage and inspected via the SQLite browser.

Pulling the database from the internal storage via adb pull (Figure 46). Browsing the database via SQLite browser (Figure 47).

The /Pentest Folder

AppUse is a project that is found under constant development. One of the main goals is to always be up to date with the latest attack tools, enabling a researcher to achieve full attack coverage of a given application.

The directory structure of AppUse contains the /Pentest folder, which is where all the tools are allocated. A brief view over the folder will give us this (Figure 48).

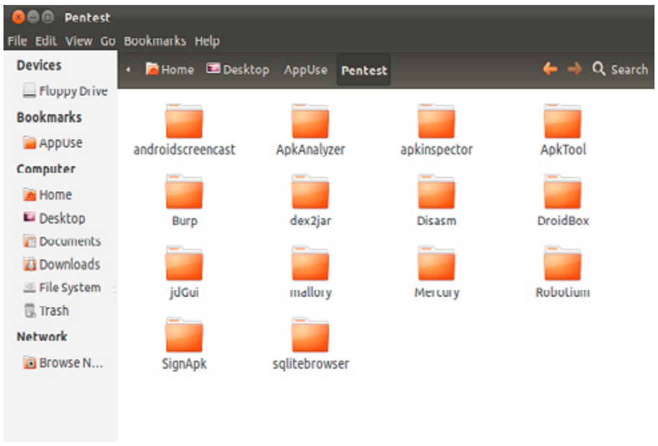
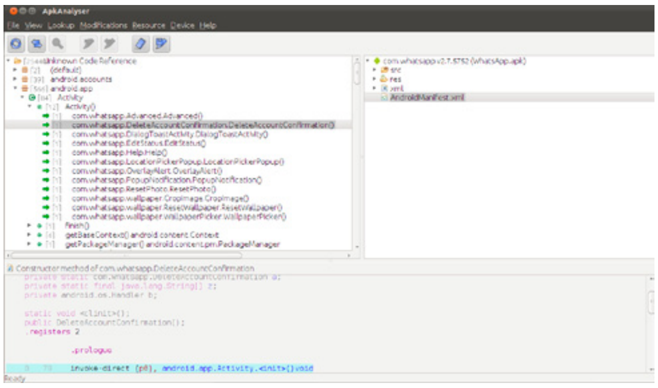


Figure 48. *The Pentest folder*



```
Unknown Code Reference
  [2]   (default)
  [39]  android.accounts
  [566] android.app
    [84]  Activity
      [12]  Activity()
        [1]  com.whatsapp.Advanced.Advanced()
        [1]  com.whatsapp.DeleteAccountConfirmation.DeleteAccountConfirmation()
        [1]  com.whatsapp.DialogToastActivity.DialogToastActivity()
        [1]  com.whatsapp.EditStatus.EditStatus()
        [1]  com.whatsapp.Help.Help()
        [1]  com.whatsapp.LocationPickerPopup.LocationPickerPopup()
        [1]  com.whatsapp.OverlayAlert.OverlayAlert()
        [1]  com.whatsapp.PopupNotification.PopupNotification()
        [1]  com.whatsapp.ResetPhoto.ResetPhoto()
        [1]  com.whatsapp.wallpaper.CropImage.CropImage()
        [1]  com.whatsapp.wallpaper.ResetWallpaper.ResetWallpaper()
        [1]  com.whatsapp.wallpaper.WallpaperPicker.WallpaperPicker()
      [1]  finish()
      [4]  getBaseContext() android.content.Context
      [1]  getPackageManager() android.content.pm.PackageManager

com.whatsapp.v2.7.5752 (WhatsApp.apk)
  src
  res
  xml
  AndroidManifest.xml

Constructor method of com.whatsapp.DeleteAccountConfirmation
    private static com.whatsapp.DeleteAccountConfirmation a;
    private static final java.lang.String[] z;
    private android.os.Handler b;

    static void <clinit>();
    public DeleteAccountConfirmation();
    .registers 2

            .prologue

            invoke-direct {p0}, android.app.Activity.<init>()void
Ready
```

Figure 49. *The ApkAnalyzer tool*

As you might have noticed, not all the tools are currently present in the Dashboard. While the Dashboard have many functionalities that in a standard research will fully cover all the researcher needs, some others are still being in development and not embedded yet to the Dashboard.

AppUse won't stop you from using those, as we are familiar that for some researchers it is a need. Those weapons will be found under the /Pentest folder.

For instance, the following shows the APK Analyser in action (Figure 49):

Of course more tools are available, such as Robotium, DroidBox and SQLite Browser. This folder is constantly updated and new weapons are added all the time.

Link to appuse

AppUse can be downloaded from here: *https://appsec-labs.com/AppUse*.

EREZ METULA

Erez Metula is a world renowned application security expert, spending most of his time finding software vulnerabilities and teaching developers how they should avoid them. Erez has an extensive hands-on experience performing security assessments, code reviews and secure development trainings for worldwide organizations, and had previously talked at international security conferences such as BlackHat, Defcon, OWASP, RSA, SOURCE, CanSecWest and more. His latest research on Managed Code Rootkits, presented at major conferences throughout the world, was published recently as a book by Syngress publishing. He is the founder of AppSec Labs, where he focuses on advanced application security topics. Erez holds an MSc in computer science and he is CISSP.

MOVE TOMORROW'S BUSINESS TO THE CLOUD TODAY
YOUR TRUSTED ADVISOR ON CLOUD COMPUTING
MULTI-VENDOR
ANY DEVICE
HYBRID CLOUD
CLOUD ITERATION

How to Provide Strong

Authentication for Your Users

Alphanumeric passwords have long been the primary method of authentication and access control on the Web. In recent years, however, the use of passwords as the sole method of authenticating users has become an outdated, insecure and unsustainable approach.

More than 85% of websites ask visitors to create an account requiring a username and password [1]. Many sites do this simply as a way to gather marketing information on the user; not because they are storing sensitive user information. The practice has become unsustainable, as people have become overwhelmed by the number of passwords they must remember for all their online accounts and mobile applications. To cope, people reuse the same passwords or they choose weak passwords, which are easier to remember but also easier to guess or hack. As a result, the average Internet user has more than 25 online accounts for which they use just 6 passwords [2], and the top 5,000 most common passwords on the Web are shared by 20% of the population! [3]

Static passwords are not only unsustainable as the sole layer of authentication, they also provide a very low level of security for the account or data they are meant to protect. Hackers can often guess a user's password by trying combinations of names, birthdates, common words or personal information gathered from social networking sites. Powerful processors (GPUs) available today enable hackers to quickly crack even strong passwords using brute force attacks. A personal computer running a $400 GPU can try an average of 8.2 billion password combinations each second! Such technologies allow hackers to crack lists of 100,000 passwords in just hours [4].

Additionally, millions of user credentials are already posted online from previous large-scale password leaks like the 2009 breach of 32 million user credentials from RockYou.com, the 2010 breach of 1.5 million user credentials from Gawker Media Group, the 2011 breach of 100 million passwords from Sony, and the 2012 breach of 24 million user credentials of Zappos customers. Such large lists of leaked credentials not only enable hackers to write programs that make their password cracking algorithms even faster, they also trigger a domino effect across the Web. Knowing that most people use the same credentials on multiple sites and applications, spammers and hackers immediately use those leaked credentials to try accessing user accounts on other websites. In the case of the Gawker breach, hundreds of thousands of user accounts on Twitter were compromised and used to spread spam and malicious links. Amazon and LinkedIn had to enforce password resets for their entire user communities to prevent accounts from being compromised.

This domino effect from large password breaches is exacerbated by the fact that most websites and applications today still do not enforce strong password policies or authentication standards.

Shockingly, a study of authentication standards on the Web showed that most sites – including high profile websites like eBay and Amazon – do not take the simple precaution of limiting the number of password guesses at login to prevent brute force attacks [5].

What password breaches mean for the business

The problems associated with relying on static passwords as the sole layer of authentication harms not only the individual user who has their account compromised, but also the online business or website itself. Once an attacker has access to user accounts, the negative repercussions are costly for the business and include legal liability, fines, loss of customers, damage to the company's brand and reputation, plus the cost of fixing security and IT systems while in a crisis state that is the aftermath of a data breach.

In 2012, hackers stole more than 8 million user passwords to LinkedIn and eHarmony accounts and published many of them online. LinkedIn estimates that it spent more than $1 million to clean up the debacle and will need to spend another $2 – $3 million for additional security upgrades [6].

- 96% of compromised records from data breaches are accessed from outside the organization, often through the use of stolen login credentials
- 30% – 50% of Help Desk support calls are related to password reset requests
- Only 52% of businesses require some form of authentication on employees' smartphones

In 2011, Sony suffered a data breach that resulted in the leaking of more than 100 million passwords and other personal information of PlayStation users. *Sony spent more than $170 million to remedy the fallout from the password breach* [7].

The growth of mobile exacerbates the problem

The problems with passwords are even worse when it comes to mobile applications. People are increasingly using smartphones and tablets for a mix of personal and professional uses, from browsing the web and social networking sites, to connecting to work email and business applications, to conducting mobile banking and mobile shopping. Despite accessing such sensitive information on these devices, both consumers and developers of mobile applications are carrying over their bad password habits and weak authentication schemes to mobile.

Research has shown the majority of smartphone owners do not password protect their devices, despite having them connected to sensitive applications including work networks and banking applications. Meanwhile, many mobile app developers provide users the option of leaving the application perpetually logged-in, and users do so because typing a password on a smartphone to log in each time is too cumbersome [8]. As a result, people walk around with small computers in their pockets that are connected to all sorts of sensitive, personal and professional data and none of it is password protected. Some mobile experts have even ventured to say that until we adopt better methods of authentication for use on mobile, the adoption of mobile commerce will remain stunted as consumers struggle with the inconvenience of entering passwords and concerns over security [9].

Finding balance among competing priorities: security, usability and cost

With all the security threats described above, security professionals might be tempted to simply enforce more stringent password requirements from users, requiring longer passwords or requiring users to change their passwords every few months. However, such requirements only increase the likelihood that users will forget their passwords or attempt to find ways to circumvent the security requirements, which can increase costs for the business. Many companies report that 30% – 50% of Help Desk support calls are related to password reset requests.

To achieve effective, strong user authentication on websites, web applications and mobile applications, security professionals must find a balance among three separate forces whose goals are often at odds: the cost and security needs of the company, the impact on usability and user behavior, and the motivations of the would-be attacker.

The goal of the company is to make security on the website or application as rigorous as possible while minimizing the cost and effort spent implementing security controls. However, to do this, the organization must also take into account the behavior and motivations of both its users and the attackers.

In most cases, the attacker also conducts a cost vs. benefit analysis when it comes to compromising login credentials. The attacker's goal is to maximize profits while minimizing the cost and effort spent achieving the payoff. The more the attacker can do to automate the attack, the better the cost vs. payoff becomes. That is why the use of keylogging malware, botnets and brute-force attacks are still the most pervasive threats, while more sophisticated and time-consuming threats such as man-in-the-middle attacks remain relatively rare in the wild.

The user also instinctively performs their own evaluation of costs vs. benefits and behaves in a rational way as a result. Although it's easy to blame the users for choosing weak passwords or reusing passwords, the reality is that creating a unique, strong password for every website is not a rational choice. The cognitive burden of remembering so many complex passwords is too high a cost – especially if the user believes the odds of their credentials being stolen are small or that the business that owns the website will absorb any losses resulting from fraud [10]. Thus, users reject security advice about choosing unique, strong passwords as impractical and inconvenient with negligible benefit – or in other words, a poor cost/benefit tradeoff.

The motives of the business, the user and the attacker are often competing but they are all intertwined and security professionals should not think of them as separate islands of behavior. They must all be considered when developing an effective security strategy.

The goal is to achieve the optimal balance, having optimized the cost/benefit tradeoff for the business, made the security requirements easy enough for users to adhere to, and made it just difficult enough for the would-be attacker that it is not worth their effort, so they instead seek an easier target.

Recommendations For Strong Authentication

The domino effect caused by the large password leaks mentioned above demonstrates to security professionals that a data breach at one site can lead to a network intrusion on your own website or application. Since you can't control the security practices at other companies, you must implement measures to identify risk and add layers or additional factors of authentication on your site, application or enterprise system.

Evaluate your business needs and consider the most common security threats

First, consider the industry in which your business operates. What type of data needs to be protected and why? What form would an attack most likely take? (e.g. Is an attacker likely to steal user credentials and sell them for profit, or more likely to use stolen credentials to access user accounts and commit fraud? Are you most concerned about stopping brute force attacks, or could your site be a target for a more sophisticated threat such as a man-in-the-middle attack?) Are there data security regulations with which the company must comply? Who is the user population – are they employees, business partners or the general public? How security savvy is the user population? Is there significant incentive for the user population to invest extra effort to follow more stringent security practices to protect their accounts, or is it a site or application that users may be ambivalent about making extra effort to securely protect?

Conducting an evaluation of the business needs, the most prevalent threats and the user behavior will help determine the level of risk and how stringent the authentication requirements should be.

Make sure you have the basics covered:

To strengthen authentication, websites, applications and businesses need to stop relying solely on passwords as the only method of authentication. I will elaborate later in this article on the additional layers and factors of authentication that can be added, but since you will likely still use a password as the first layer of authentication, let's make sure you have these basic measures covered:

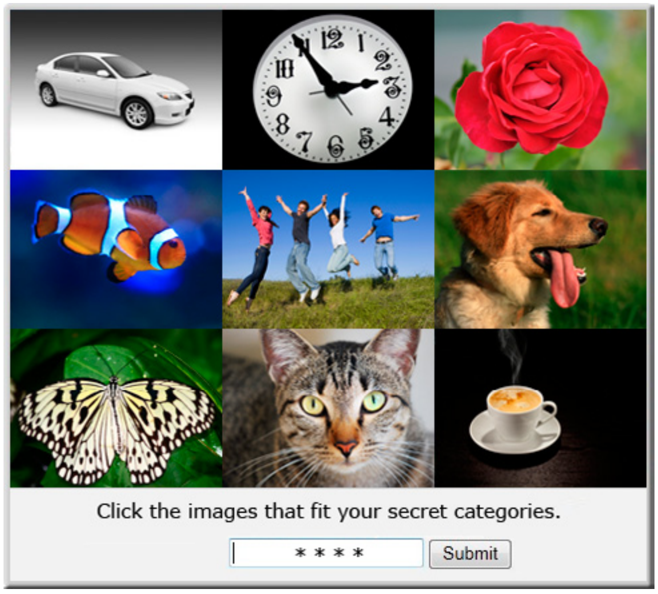
- Enforce a dictionary check on passwords to ensure that the user cannot choose a common word for their password.
- Require a strong username that includes a numeric character. Often the username is the easiest portion of the login credentials for a hacker to guess. Do not use the user's email address as their username.
- Limit the number of failed login attempts. If a user fails the login three times, temporarily suspend the account until they authenticate through other means.
- If login failed, don't identify which user credential is incorrect. Stating that the 'password is incorrect' or the 'username doesn't exist' allows hackers to harvest account information. A general statement such as "Incorrect login, please try again" helps prevent account harvesting.
- Use SSL to create an encrypted link between your server and the user's Web browser during account enrollment, the login process and the password reset process.
- Provide users with contextual advice on how to choose a strong username and password. Research shows that users do choose better passwords when given advice on how to do so. One option is to have a password strength meter built into the page.

- Hash user passwords using bcrypt, scrypt, or other hash algorithms specifically designed to store passwords. Do not use SHA1, MD5 or other algorithms that were not specifically designed for hashing passwords as they are not secure.
- Use Salt. Use a unique salt for each user account/password and store that salt with the password. An additional layer of system wide salt that is not stored with the password can also add extra strength if the database is stolen because it is not stored with the passwords but is known to you.

These steps may seem rudimentary to some readers, but a study conducted by researchers at Cambridge University showed that most websites did not even enforce these minimum standards [11].

Web-based authentication solutions for generating one-time passwords

With the growth of Software-as-a-Service (SaaS) providers, it's easier than ever to adopt solutions that generate one-time passwords for users, without any hardware investment or significant integration efforts. While one-time passwords will not stop a sophisticated man-in-the-middle threat, they do protect against the most common security threats: users choosing weak passwords, reusing the same password or having their passwords stolen using keystroke logging malware.

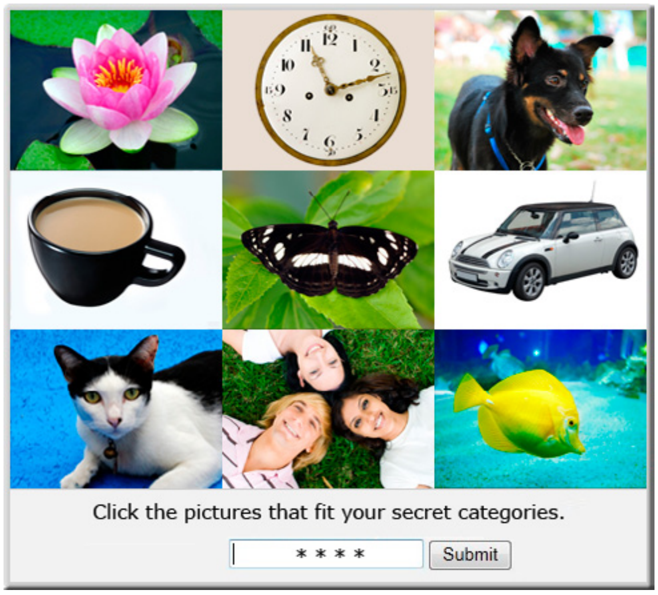


Example of image-based authentication. During enrollment, the user chooses a few categories, such as dogs, flowers, cars and cats. When authentication is needed, the user is presented with the ImageShield™ and must identify their secret categories by clicking the correct pictures. The images are different every time and in a different location every time, but the user's categories remain the same

By generating one-time passwords for users each time authentication is needed, you can ensure strong passwords are being used and that stolen or leaked passwords cannot be used to access accounts.

One way to add a flexible layer of authentication that generates one-time passwords without overly-burdening users is to use an image-based approach. Being a cloud-based solution, it requires no hardware or software downloads from the user and very little cognitive burden. By adding one-time passwords in an easy-to-use manner, image-based authentication increases security while also increasing usability and improving user experience.

While there are other image-based, pattern-based and graphical approaches to authentication, most do not generate one-time passwords. Confident Technologies' approach asks users to pick and memorize a few categories of things during a one-time enrollment. The user might choose categories such as dogs, flowers and cars. Each time authentication is needed, the user is presented with an image-based challenge. To authenticate, the user must identify which images fit his/her previously-chosen, secret categories. The specific pictures are different every time, but the user's categories always remain the same. The administrator can choose whether or not to require users to identify their secret categories in the same order each time. As the user clicks or taps on the pictures, a one-time password is generated behind the scenes and submitted to the Confident Tech-



Example of another ImageShield™ that would be presented to the same user. The user always looks for the same categories, but the specific images and their location are different every time, forming a one-time password on the back-end nologies server and checked to determine if the user identified the correct categories.

One of the benefits of an image-based approach to generating one-time passwords is that the cognitive burden on users is low. Research has shown that people are much better at remembering categories and recognizing pictures of common things than remembering a long string of alphanumeric characters. Additionally, when the user sees the image grid presented before them, their memory is jogged, helping them to recall which categories they chose during enrollment. This process is called "Guided Recall."

SaaS one-time password solutions are well-suited to the business objective of increasing security with minimal cost (no need for hardware or infrastructure integrations) while also increasing usability (no need to carry authentication tokens), making it more likely the user will adopt the stronger security practice. Additionally, SaaS one-time password generators can be used as a stand-alone authentication solution, or better still, inserted as a second layer of authentication in addition to the traditional password. Using a multi-layered approach to authentication provides greater security than any single layer of authentication. When used as a second layer of authentication, the image-based authentication approach described here can strengthen the security of a login by more than 99.999% [12].

Example code snippets

Below are two example code snippets for creating and verifying an image-based authentication solution that organizations can implement on their servers. The Confident ImageShield™ API is based on RESTful architecture for all of the services exposed by the server (Listing 1 and Listing 2).

Risk-based, progressive authentication

Security professionals at organizations requiring even stronger security should consider integrating

Listing 1. *Generating a new encrypted ImageShield for a user*

```
ApiClient apiClient = ApiClientFactory.createApiClientWithProperties();

User user = (SiteUser) httpSession.getAttribute("site _ user");

try {
    imageShield = apiClient.generateImageShieldResource(user.getAccountNumber(), user.getEncrypted-
                 Config(), user.getEncryptedPasscode(), false);

    request.setAttribute("image _ shield", imageShield);
}
```

Listing 2. *Verifying an ImageShield from an http servlet request*

```
// Create a Confident Technologies API client
ApiClient apiClient = ApiClientFactory.createApiClient();

// Instantiate the response code
boolean serverResponse = false;

// Set the image shield id to validate
String imageShieldId = request.getParameter("image _ shield _ id");
apiClient.setImageShieldId(imageShieldId);

// Set the users code that was submitted for that image shield
String verificationCode = request.getParameter("image _ shield _ verification _ code");

apiClient.setVerificationCode(verificationCode);

serverResponse = apiClient.verifyImageShieldSolution();
```

a risk engine with their authentication solutions. Using behavioral and contextual risk profiling tools and techniques, the system can dynamically trigger additional layers of authentication only when needed. Risk-based authentication solutions should identify device reputation and evaluate the geo-location of the user's IP address and time of day that they are accessing the site. Also examine the frequency of the login attempts, which could indicate a brute force attack. If a high-risk or suspicious situation is identified, require an additional authentication step from the user. The additional authentication step could simply be another layer of authentication, it could be a second factor of authentication, or it could be a request for progressive authentication.

With progressive authentication, the user may only need to identify a portion of their shared secret in order to authenticate and gain access to systems, applications or data that are low-risk or require a lower level of security. If, during the same session, the user attempts to access other systems, applications or data that are higher-risk, they would then need to identify additional pieces of their shared secret.

For example, using the image-based authentication solution described previously, the user may select four secret categories during enrollment when setting up their online bank account. When logging into their bank account, the user may be asked to enter their username and traditional password, and identify one or two of their secret categories. With that initial level of authentication, the user may be able to access account details, check their balance or other basic tasks. However, if the user attempts to transfer money to an outside account, or if they have triggered the risk engine with unusual behavior, they can be triggered to identify the remaining two secret image categories, or to identify all four of their secret categories in a predetermined order.

Not only does this type of progressive authentication generate one-time passwords, but the image-based approach is unique in that the user's secret image categories are all pieces of one authentication secret that can be used as a whole, or incrementally for progressive authentication.

Multifactor Authentication

Organizations in regulated industries such as finance and healthcare, or even those organizations that are not in regulated industries but who may be a high-profile target for hackers are recommended to go even further and adopt out-of-band multifactor authentication. With the widespread use of mobile phones, two-factor authentication using the phone as the second factor is growing quickly. One of the most common methods is for the website or organization to send a one-time authentication code to the user's mobile phone via an SMS text message. The user then types the code into the web page to authentication.

Unfortunately, cybercriminals know that SMS is increasingly used for sending authentication text messages in banking and other industries, and as a result they are increasingly targeting the channel for attack. Using a variant of the infamous Zeus malware, they are able to compromise user's online accounts and then intercept and reroute the authentication text messages to their own phones. Because the authentication code is written in clear text in the text message, the hacker is able to read it, type it in to a web page on their computer and gain access to the user's account.

A better approach is to adopt a multi-layered, multifactor authentication solution. If the user base primarily has smartphones, the online organization can use push technology or an app on the phone to send an out-of-band authentication challenge to the user's mobile phone. Again taking the image-based authentication approach described earlier, when a user logs into their online bank account on their PC, they enter their username and traditional password. Then, the bank sends an image-based authentication challenge to appear on the user's smartphone display. The user must tap the images that fit his/her secret categories and tap a submit button in order to authenticate out of band. The user not only had to have possession of the registered second factor device (their phone) but also had to apply a shared secret (knowledge of their secret categories) on the phone. The entire authentication process remains out of band from the PC – the user did not have to type anything into the web page. Additionally, even if someone were able to intercept the communication or had possession of the user's mobile phone when the image-based challenge is delivered, they would not know which images to tap because they would not know the secret categories. In this way, it is a truly out-of-band, multilayered, multifactor authentication solution.

With any multifactor authentication solution relying on the user's mobile phone as the second factor device, the authentication process will be most secure if the solution relies on push technology and is a server to server communication, keeping the entire process out-of-band by not requiring the user to type any data into the web page.

Using image-based authentication combined with a risk engine, an organization can create a seamless, cross-platform authentication solution. For example, if the risk engine determines that the level of risk is low, it may trigger a second layer of authentication (the image-based challenge) on the web page for the user to solve. However, if the risk engine determines that risk is greater, it can trigger the image-based challenge to be sent to the user's mobile phone, for the user to solve out-of-band for multi-layered, multifactor authentication.

Although multifactor authentication has primarily been adopted in the financial services vertical (due to industry regulations), it is becoming increasingly common on all types of websites and applications, primarily due to high-profile password breaches at many of these sites. Sites such as Apple, Facebook, Evernote, Amazon Web Services, PayPal, Drop Box and Microsoft Live have all implemented two-factor authentication using end-users' mobile phones.

Biometrics and Behavioral biometrics

Biometrics and behavioral biometrics are becoming viable authentication options. For example, most laptops, smartphones and tablets now come with built-in video cameras that can be used for facial recognition. Fingerprint scanners are quite common in mobile and desktop environments and there are even snap-on devices that allow you to add a small fingerprint scanner to a device that does not have one built in. Smartphone applications can be used for voice recognition. Retina scanners, palm-scanners and ear-scanners have all been used in biometric identification. However, drawbacks of biometric authentication include the need to maintain the equipment and 'body parts' to get accurate readings; biometric id data must also be stored in databases and is, therefore, susceptible to malicious theft and forgery. Depending upon the type of organization, account or data being accessed, a user may or may not be willing to provide biometric data for authentication. For example, users may be willing to use a fingerprint scanner to authenticate for their bank account, but not for their favorite social networking or shopping site.

Behavioral biometrics are also gaining popularity. Behavioral biometric techniques include software that tracks the user's behavioral patterns such as keystroke speed and mouse movements. It has been demonstrated that these and other behavioral profiling techniques can help to successfully identify an individual user, especially when used in conjunction with another authentication factor.

Conclusion

Authentication standards on most websites are woefully lacking. Relying solely on username and passwords puts the business, its users and

Refernces

[1] Bonneau, J., & Preibusch, S. (2010). The password thicket: technical and market failures in human authentication on the web. Proceedings of WEIS 2010: The Ninth Workshop on the Economics of Information Security, Retrieved from *http://weis2010.econinfosec.org/papers/session3/weis2010_bonneau.pdf*
[2] Florêncio, D., & Herley, C. (2007). A large-scale study of web password habits. Proceedings of the International World Wide Web Conference Committee, *http://research.microsoft.com/pubs/74164/www2007.pdf*
[3] Kahn, R. (2010, January 21). 123456 most common password?. Discover, Retrieved from *http://blogs.discovermagazine.com/gnxp/2010/01/123456-most-common-password/*
[4] Goodin, D. (2012). Why passwords have never been weaker – and crackers have never been stronger. Ars Technica, Retrieved from *http://arstechnica.com/security/2012/08/passwords-under-assault/*
[5] Bonneau, J., & Preibusch, S. (2010). The password thicket: technical and market failures in human authentication on the web. Proceedings of WEIS 2010: The Ninth Workshop on the Economics of Information Security, Retrieved from *http://weis2010.econinfosec.org/papers/session3/weis2010_bonneau.pdf*
[6] Lennon, M. (2012). LinkedIn: Breach cost up to $1 M, says $2-3 million in security upgrades coming. Security Week, Retrieved from *http://www.securityweek.com/linkedin-breach-cost-1m-says-2-3-million-security-upgrades-coming*
[7] Martinez, E. (2011). Playstation network breach has cost Sony more than $171 million. CBS News. Retrieved from *http://www.cbsnews.com/8301-504083_162-20065621-504083.html*
[8] Confident Technologies. (2011). Survey shows smartphone owners choose convenience over security. Retrieved from http://www.confidenttechnologies.com/news_events/survey-shows-smartphone-users-choose-convenience-over-security
[9] Kats, R. (2011). Authentication going to be more revolutionary than commerce transaction: CTIA panelist. Mobile Marketer. Retrieved from *http://www.mobilemarketer.com/cms/news/commerce/11217.html*
[10] "So Long, And No Thanks for the Externalities: The Rational Rejection of Security Advice by Users" by Cormac Herley, Microsoft Research
[11] "The password thicket: technical and market failures in human authentication on the web" by Joseph Bonneau and Sören Preibusch
[12] Confident Technologies. Technology Overview: *http://www.confidenttechnologies.com/technology*

its valuable information at risk. Not every business needs true multifactor authentication, but most businesses can benefit from implementing relatively simple security controls, such as adding one-time passwords. To develop the right authentication strategy, security professionals must evaluate the security needs of the company and balance the cost/benefit tradeoff of stringent security with the impact on usability and user behavior, while thwarting the objectives of the would-be attacker.

User education is also critical for improving authentication security. Unless the user clearly understands the reasons for and personal benefits of additional authentication requirements, they will find ways to circumvent the policies.

Finally, it's important to remember that 'security' is a process – IT professionals must continually re-evaluate the company's security needs, identify areas for improvement and make a security roadmap for future improvements. Incident response is critical – always have a contingency plan in place to help mitigate the damage as quickly as possible.

The website can never be 100% secure, but IT professionals should aim to be in the optimal zone that balances the costs with the benefits, helps its users and is strong enough to deter most attackers.

ROMAN YUDKIN

Roman Yudkin is Chief Technology Officer at Confident Technologies. He is responsible for Research & Development, Engineering and general oversight of all corporate technical functions.

Yudkin has more than 25 years of hands-on leadership experience in high technology sector – architecting, building, and bringing to market complex enterprise software systems and professional services across multiple industries and international settings. Key positions in his career include VP of Engineering at Websense, Sr. VP and CTO at Pharmatica, COO at Mindport, VP of Software Development at Groupe Bull, and Director of Software Engineering at Bell Labs. Most recently, Yudkin served as CTO and co-founder of Affeo Inc. in Carlsbad, California. Yudkin's educational background includes a Bachelor's degree in Computer Science and Mathematics from the University of Wisconsin-Madison, a Masters degree in Computer Science from the University of Wisconsin-Madison, and a Masters degree in Cognitive Science/Artificial Intelligence from the Rutgers University, New Jersey.

a d v e r i s e m e n t

OOMBA Security LLC

Managed Security Services
Automated Compliance (PCI-DSS, SOX, HIPPA)
Secure Server Build Automation
Centralized Logging and Event Management
Business Continuity and Disaster Recovery
Security Services to meet any budget
Always Compliant © System Monitoring

Contact: oombasecurity.com/contact.htm

Quantum IQ

How the Worlds Military's Intend to Capitalize on the Future of Mobile and Neuroscience Technologies

"Mobile" used to mean a laptop and while the laptop is technically still mobile, the term now means phone or tablet. The next generation of mobile is not seen or touched as an interface. It is simply comprehended.

While communication is still a major component around the mobile of the future, it is not the backbone of application development. Quantum Intelligence drives the next wave of mobile technology.

What is "Quantum Intelligence"?

Quantum intelligence is the ability to think something like "How deep is that river? What known predators to man live in it? What is its temperature and how likely is it possible I could swim across it in 10 minutes?" and have the answer comprehended in a number of seconds." The current mobile revolution has put a handful of technologies on a trajectory to all crash into each other in a way that will make all of this possible.

Microprocessors

The mobile revolution has the throttle fully pressed on the constant and relentless advancement in making processors smaller, faster and more efficient (less battery/energy usage). Powerful microprocessors are the size of a fingernail now.

Wireless Data

What started as a luxury, rapidly became a staple. Rapid data coverage is spreading across the globe as it if were a virus. In many countries a computer unaffordable for an average family. Instead a phone with wireless data that is 2 generations old is the family's computer.

Artificial Intelligence

(For the lack of a better term) The world is demanding that its information searches become increasingly agile and require less steps. Only a couple of years ago I could not say to my phone "Do I need an umbrella", "Closest place to watch that Russell Crowe movie." Instead it took a handful of steps to figure this out. This isn't truly artificial intelligence as a smarter mapping of terminology to searches.

Mobile sensors

I don't need to tell the computer where I am to find out about the weather or movie theaters because it knows where I am. Phones come with several very useful mobile sensors that are useful to everyday needs but an epic amount of useful lightweight open source mobile sensors have been created in the last 5 years. Everything from radiation to distance sensors are currently available. (Phidgets.com)

Big Data

Each of the previously mentioned advancements has led us to answers and knowledge in places, times, speeds and formats that were unimaginable. We got "it" fast. We got "it" everywhere. We got "it" with better precision and less steps. Now we want MORE of "it".

Neuroscience

A combination of interface advancement and human sensor development. We created cameras that can record a scene, capture the information digitally and display it onto flat surfaces in a way that our eyes (sensors) can be pointed at and send a signal to the brain that comprehends the scene. The next step is bypassing the eyeballs. We can control games and prosthetics with our minds, demonstrating a real ability to send output from the mind in order to control technologies. Getting information back into the mind is the next step. We currently only have two recordable senses. Sound and Sight. If we recorded what signals your hand sends to the brain when you touch sandpaper, we should be able to send what sandpaper feels like over a network to someone else if we could just get information directly into the brain without recreating it the way we do with cameras. Imagine the implications this has on training, or reducing shock, allowing someone to feel some level of the sensation ahead of time. You can currently take a picture of a type of dog and Google will analyze the image and show you results. In the future your eyes will just look at the dog and the brain signal will just go to a microcomputer that you are wearing and the results will be sent back right to your brain. You won't talk or type about it. You won't look at a screen. You will just think your questions and the results will return and furthermore the caliber of questions and answers will be incredible. In today's world you might see a cake you desire to make yourself, research what it is, lookup instructions to make it, lookup directions for the best grocery store, text home about what ingredients you have and don't have, go down more aisles than you need to until you have the ingredients, wait in line, pay with a credit card, drive home, pull up the instructions on your ipad and proceed to make the cake. In tomorrow's world you would just look at it, which would give you the sensation of its average taste, mentally agree to pay and when you got home the ingredients you lack would be waiting for you. You would see how to make it in your minds eye. You would share how your cake tasted specifically with others who were nowhere close to you.

Walk, Crawl, Run

The President of the United States has put forth 100 Million in funding to speed up the advancements in neuroscience. We aren't as far as it might seem. If I can say "how far is the moon" and immediately get a response and I can *think* about how a prosthetic hand should react and it does, then I should be able to think "How far is the moon" and at least hear the response said back to me right? There are dozens of home sensors that can be purchased for Brain Computer Interfaces on the Internet for between $100-$500 with API's and SDK's. What I implore any developer reading this article to do, is to start integrating these into mobile technologies and map human languages to the brain rather than directions and controls. If we can demonstrate thinking "what is 15% of .0065 and getting the answer spoke to an ear piece then we are halfway to Quantum Intelligence.

JERE SIMPSON

Jere Simpson is the founder, President, and sole owner of KiteWire, Inc. – a software development company founded in early 2007. At 32, he is also Managing Partner and Chief Technology Officer of PENme, LLC and Advisor to the Department of Defense (DoD) and Federal Government. He, also, sits on the board of three other companies. Known for being a serial entrepreneur, KiteWire, Inc. is Mr. Simpson's fourth company, of which he has founded and been President. Each of his companies has experienced more than 800% growth in their first three years. The first three companies were sold in less than 4 years. He has not only advised some of the biggest companies, such as Facebook, Google, and Apple, but actively architects enterprise systems for the Federal Government and DoD to include, but not limited to the FBI, CIA, U.S. Army, Navy, Department of Homeland Security, Air Force, and USSOCOM. His current company, KiteWire, Inc., develops a mobile security product, Steel Talon, used by the United States Government. KiteWire created what is considered the first Military iPad app. Mr. Simpson also has a deep passion for philanthropy. He created the "Strike Lightning" initiative in 2009 to help under-privileged young people with professional development. Having been one of the first commercial internet users (1,032nd), he believes in the power that access to information can give to under-privileged children and, as such, his company works to diminish the digital divide.

Mobile Antivirus Is a Myth

So, why is mobile antivirus a myth you ask? A true antivirus for mobile devices is not possible given the SDKs (software development kits) provided by most mobile platforms.

In the mobile security space there are more than a few companies selling what they like to call "antivirus" applications for smartphones. The problem is that the term is being used erroneously – sadly it's no accident.

A virus, as it relates to any computing device, is a form of software that can replicate itself by way of documents and executable files in order to infect other devices, either automatically through a network or through a storage device such as a flash drive. The end goal of most viruses is the corruption of data and/or the damaging of the operating system.

In order to detect and mitigate real viruses, a software solution would need to be capable of running as a root process on a system, something that is just not possible on most mobile platforms currently where applications typically run in a sandboxed environment.

Take for example the fact that none of the Android antivirus apps on the market can provide any zero-day protection. The best they can do is to monitor for a package to be installed, then do simple signature-based check. If there were an actual kernel exploit in the wild, that sandboxed third-party app would not do a darn thing to protect your device. In fact, nothing short of an OTA patch from Apple, Google, OEM or some mobile operator would suffice.

Applications claiming to be "antiviruses" are merely detecting what has the potential to be malware, something that a developer of an application may have snuck into the software code that is meant to steal data or interact with the device in such a way as to cause it to send premium SMS messages at the victim's expense, something that is more correctly defined as being a Trojan or some form of spyware.

Although these detection capabilities may be marginally useful to the end user, they do not by any stretch of the imagination fit the definition of an "antivirus" or replace common sense – that is to say, being cautious about which applications you download and then carefully reviewing the permissions for each application if you do install it.

There have been many suspect applications that have been removed from the various markets and both Google and Apple, and there are other forms of malware like Zeus and SpyEye that have been employed in toolkits aimed at harvesting banking credentials, but for the most part there have been no wide-scale self-replicating viruses targeting the most popular smartphone platforms.

So, why do these companies call their products an "antivirus" when it isn't? The simple answer is marketing.

Like all good social engineers, marketers know the that technical newbs don't know the true definition of anti-virus from a hole in the ground, as the term is ingrained in our heads as meaning "protection" from years from security firms pushing them. To make matters worse, these companies tend to amplify threats in their marketing materials by employing generous amounts of FUD (Fear Uncertainty and Doubt), often feeding baseless statistics from their own "research" to the press to generate hysteria, all the while hoping reporters don't check up on their "facts." Unfortunately, most don't.

The false sense of security these "antivirus" applications try to provide is quite irresponsible. Promising to protect us from "viruses" can be more dangerous than the "viruses" themselves, as it may convince someone they don't need to install a critical security patch from their vendor, as they might believe a third-party application is protecting them from malware, when in fact it is not.

KEN WESTIN

Ken Westin (@KWestin) of Tripwire, Inc. is a creative technologist with over 13 years experience in building and breaking things through the use/misuse of technology. In the past he has been an avid cyber criminologist with a knack for empowering electronic devices to defend themselves from malefactors. His technology exploits have been featured in Forbes, Good Morning America, Dateline, New York Times, The Economist and he has won awards from MIT, CTIA, Oregon Technology Awards, SXSW, Web Visions, Entrepreneur and others.

a d v e r t i s e m e n t

IT-Securityguard

Lets secure IT

Android Vulnerability Scan

Web Penetration testing

Secure hosting

contact: contact@it-securityguard.com

www.it-securityguard.com

An interview with

Omar Khan

the Co-CEO of NQ Mobile

Omar Khan joined NQ Mobile in January 2012 as co-CEO. In this role, Mr. Khan is responsible for the global direction of the company while also focusing on the business expansion across markets including North America, Latin America, Europe, Japan, Korea and India. He joined NQ Mobile from Citigroup, where he was Managing Director & Global Head of the Mobile Center of Excellence and led the company's mobile development and delivery efforts globally.

Prior to that, Mr. Khan served in multiple senior executive roles at Samsung Mobile. During this tenure, he served as Chief Strategy Officer and the Chief Product & Technology officer and was responsible for Samsung Mobile's strategy, product, technology, content and services functions.

Before joining Samsung, Mr. Khan spent eight years at Motorola, where his last role was Vice President, Global Supply Chain and Business Operations for the Mobile Devices Business.

Most recently, Mr. Khan was named an under 40 "mobilizer" by FORTUNE magazine in the October 11, 2012 issue. Mr. Khan has also been honored by Crain's Chicago Business Magazine with a "40 under 40" award and was nominated by Androinica as Android Person of the Year. In addition, Mr. Khan was also named among FORTUNE Magazine's 36 "most powerful disrupters."

He holds Bachelor's and Master's degrees in Electrical Engineering from Massachusetts Institute of Technology (MIT). He completed his graduate work in System Dynamics in conjunction with MIT's Sloan School of Management.

Could you please introduce yourself briefly.

Omar Khan, Co-CEO of NQ Mobile. I am a recognized thought leader in the mobile industry, with deep experience leading global companies on both the device and software side.

Prior to joining NQ Mobile, I was Managing Director & Global Head of the Mobile Center of Excellence for Citigroup, where I led the company's mobile development and delivery efforts globally. I was directly responsible for the launch of the Google Wallet payment system.

Prior to Citigroup, I was CTO of Samsung Mobile. In that role, I was responsible for Samsung Mobile's strategy, product, technology, content and services functions. I played a fundamental role in Samsung's adoption of the Android platform and for developing the Galaxy series of mobile devices for the US market. During my tenure with Samsung Mobile, we became the number one provider of mobile phones in the United States, as well as the leader in Android phone technology.

I am a frequent speaker at industry conferences around the world, and I provide counsel on privacy and mobile security to the Federal Trade Commission and the Federal Communications Commission.

I have been honored by Crain's Chicago Business Magazine with a "40 under 40" award and was nominated by Androinica as Android Person of the Year. Most recently, I was named among Fortune Magazine's 36 "most powerful disruptors" and as a Mobilizer Under 40 in the Fortune's annual 40 Under 40 list.

I hold Bachelor's and Master's degrees in Electrical Engineering from Massachusetts Institute of Technology (MIT). And, completed graduate work in System Dynamics in conjunction with MIT's Sloan School of Management.

Present your company and yourself within its structures.

NQ Mobile is the world's largest mobile security and privacy software provider, with over 280 million registered user accounts globally – larger than all other mobile security competitors combined.

Our company is headquartered in Dallas, TX, and Beijing, China and I share leadership with fellow co-CEO Dr. Henry Lin.

The company was founded in 2005, and through our proprietary mobile security engine, we have been the first to discover over 90 percent of all known Android malware, including identifying 65,000 new malware threats across platforms in 2012. We are strategic security partners with some of the biggest names in mobile, including HTC, Motorola, MediaTek, Verizon Premium Wireless Retailers and Vodafone.

I joined NQ Mobile in January 2012 to focus on global expansion into markets such as North America, Latin America, Europe, Japan, Korea and India. Since I joined the company, I've recruited a team of top-level mobile-industry executives including former Samsung Mobile colleagues: NQ Mobile's Chief Commercial Officer, Gavin Kim; Chief Experience Officer, Conrad Edwards; Head of Product Management, Victoria Repice; and Head of Corporate Communication, Kim Titus. These hires have significantly raised our global visibility.

During my tenure, we've completed agreements with premier mobile industry frontrunners such as Brightstar, Telefonica, Vodafone, A Wireless, TD-Mobility, The Cellular Connection and Vox Mobile.

I have also spearheaded several major consumer and industry security initiatives, including providing an advisory role to the FCC, pressing for industry standards on effective privacy policies by app developers, and working to educate consumers through third party researchers and security experts.

What does your company deal with?

NQ Mobile is making the world safer for our increasingly mobile lifestyles. Mobile devices have become an integral part of our daily lives, as both powerful business tools and family-oriented entertainment platforms. But the rapid adoption of these devices has left mobile users vulnerable. Platforms are diverse and consumers are unaware of the dangers posed by rogue applications and URLs. At the same time, cybercriminals are increasingly taking advantage of the gaps in mobile security and sheer scale of this mass market opportunity using mobile devices to spread malicious code and steal personal, financial data; or profit from sending Premium SMS, calling Premium numbers then covering their tracks, without a user's consent or knowledge.

According to our estimates, more than 10.8 Million Android devices around the world were infected with malware in 2011. Our flagship application, NQ Mobile Security™, was developed to protect devices from such attacks, detecting and deleting viruses, malicious URLs, and other threats before the infections can inflict their damage. In 2012, we discovered over 65,227 new pieces of mobile malware – a 163% increase over 2011.

NQ Mobile Security also helps users protect their personal and financial data, keeps their devices running at optimum speed, backs up and restores contacts, and even remotely alarms/

locates, locks and deletes information from lost or stolen phones. NQ Mobile Security's cloud-based database is the world's largest catalog of mobile threat intelligence in existence, containing well over one billion risk assessed and rated URLs and 1,400,000 mobile applications. Today, this database is responsible for first detecting and resolving over 90% of all new Android mobile threats. Every NQ Mobile Security user forms part of the company's global security cloud. With no effort on their part, consumers constantly contribute new non-attributable security knowledge that helps us to detect virus samples, malicious URLs, and other threats.

Describe the team you work with.

NQ Mobile has an amazing team of mobile-industry experts. Besides Dr. Henry Lin, and myself our executive team includes recognized leaders including Vincent Shi, Gavin Kim and Conrad Edwards.

Dr. Shi is co-founded the company with Dr. Lin and serves as director and chief operating officer. He is responsible for the operations of our company, including management of business operations, channel development, and online business development.

Gavin Kim is our chief commercial officer, leading the vision, strategy, design, and development of our global consumer and business product portfolio. His primary focus is on accelerating product leadership and global sales in the mobile security, privacy, and productivity markets. Before joining NQ Mobile, Gavin served as Microsoft's General Manager for Windows Phone Product Marketing, leading the company's product marketing and platform planning teams and driving Windows Phone's application and developer ecosystem efforts. Prior to that, he was Samsung Mobile's Vice President of Content, Services and Enterprise Business, and held leadership positions at Motorola and Advanced Technology Ventures.

Conrad Edwards is our chief experience officer. He leads our global marketing, user experience and product innovation initiatives, as well as driving customer engagement and loyalty across consumer and business channels. Conrad has a long track record in shaping and delivering experience-based designs and products that help brands become more desirable and empowering to customers. Most recently, he held a senior executive role at Samsung Mobile, where he was responsible for building and leading a team that was dedicated to next-generation design and technology experiences. Prior to Samsung, Conrad led the Experience Engineering and Interactive Media teams at Motorola Mobility.

What services do you provide?

NQ Mobile is one of the first companies to be dedicated to the protection of mobile devices for both consumers and enterprises. To stay ahead of potential threats, we maintain one of the world's largest professional mobile security research and development (R&D) teams. With this heavy investment in R&D, the company has developed more than 30 proprietary core technologies and mobile information security patents and is first to identify 90% of global Android mobile threats.

Besides our flagship NQ Mobile Security product, we offer several other award-winning consumer applications including NQ Mobile Vault™ and NQ Family Guardian™.

NQ Mobile Vault solves the all-or-nothing problem with sharing your device – in the past, you either chose to share your phone and expose ALL of your content or you kept your phone to yourself.

NQ Mobile Vault benefits users by eliminating the privacy risks consumers face if their phone is stolen or simply if they have something they'd prefer others not see, protecting users' data in a safe, password-protected and encrypted place. It's a must-have application for bring your own device (BYOD) employees, families, celebrities, and mobile users that share devices. NQ Mobile Vault is available on both Android and iOS platforms.

NQ Family Guardian is our unique parental mobile management services suite that offers parents and their kids the needed balance of protection and communication. The safety and monitoring tool comprises a mobile app that is downloaded and installed on the child's smartphone along with a web-based control center that is accessible from any desktop or mobile browser. The app is currently available for Android devices, while the control-center is compatible with any web browser.

To meet businesses' security needs, NQ Mobile also offers NQ Enterprise Shield. The corporate-licensed product enables businesses to protect all of their employee mobile devices from malicious threats, even uninstalling threats that have already infected a device. The product also provides backup and restore functions, gives control over running apps and power consumption, and monitors data consumption by mobile apps.

Through our subsidiary, NationSky, we've also just released our NQSky™ mobile device management (MDM) platform.

NQSky is a robust, end-to-end solution for enterprise-level mobile management and security. It is a complete solution that assists companies to seamlessly integrate mobile devices with their existing IT policies. With NQSky, businesses are empowered to meet the demands of an evolving mobilized workforce, which can significantly improve productivity, reduce IT management expenses and take advantage of BYOD--a win-win strategy for employees and enterprises.

NQSky mobile management platform offers an expanded range of functionalities beyond the traditional MDM and put a greater emphasis on the entire device lifecycle.

What are your target clients?

NQ Mobile's portfolio includes mobile security and mobile games & advertising for the consumer market and consulting, mobile platforms and mobility services for the enterprise market. As of December 31, 2012, we had a user base of 283 million registered user accounts and 98 million monthly active user accounts through our consumer mobile security business, 65 million registered user accounts and 13 million monthly active user accounts through our mobile games & advertising business and over 1,200 enterprise customers.

Our consumer applications are available through major app stores like iTunes and Google Play, and we have point-of-purchase deals in place with five of the six largest Verizon Premium Wireless Retailers in the US (The Cellular Connection, A Wireless, Wireless, Go Wireless and Diamond Wireless), major UK retailer Phones4u and Australian retailer ePay. We also have agreements with top global carriers including Telefonica and Vodafone in Europe and US Cellular and Cricket Wireless in the US. For enterprise customers, NQ Mobile has partnered with Vox Mobile and TDMobility.

Do you look for new employees? If so, What kind of candidates do you look for?

We are continuously looking for new employees. We are looking for talent in many disciplines including engineering, marketing, business development and product management. We are looking for people who are looking to revolutionize the future of the mobile Internet. Our employees are challenging the limits of the mobile Internet and its applications within our daily lives at home and at work. As the installed base of smartphones grows from 1.3B today to several billion over the next few years, it will become the primary lifeline for every consumer. Our employees are striving to identify, enable and secure the applications of how these devices will be used in multiple aspects of our daily lives. This will include emerging uses such as mobile payments and mobile health applications.

What distinguishes you from other companies?

First, we are a truly global company, which affords us a unique advantage. We also have a dedicated Security Lab with over 250 mobile security professionals, scientists and developers around the world who proactively monitor the mobile landscape for new malware threats and mobile hacking methods.

Our cloud-based database is the world's largest catalog of mobile threat intelligence in existence, containing well over one billion risk assessed and rated URLs and 1,400,000 mobile applications. Every NQ Mobile Security user forms part of our global security cloud. With no effort on their part, consumers constantly contribute new non-attributable security knowledge that helps us to detect virus samples, malicious URLs, and other threats and inoculate against them before they can cross borders and oceans.

What do you think about Hakin9 Magazine and its readers?

We're happy to see publications like Hakin9 Magazine who are taking a pro-active role in educating consumers about the increasing number of security threats. By providing valuable information about hacking and other IT security issues, you're helping to prevent the spread of malware infections and giving people the knowledge they need to ensure they won't become victims.

What message would you convey to our readers?

Mobile Security is a real concern. Cybercriminals will always follow the money, so we are increasingly seeing threats that target devices beyond China and Eastern Europe. In fact the United States is among the top five most infected countries.

We're also seeing increasingly creative approaches to malware distribution including social engineering.

The time to be complacent is over. We all need to start safeguarding our mobile devices and data with the same diligence we do our PCs.

by Marek Majewski

Steel Talon

Mobile Defense System

Steel Talon is a Mobile Defense System intended to do more than scratch the surface of mobile security. The System uses a multitude of techniques to:

- ✓ Detect Intrusion Events
- ✓ Identify Malicious Alterations
- ✓ Transmit Forensic Information

What is Steel Talon?

Steel Talon is uniquely designed to be transparent to the mobile user, and to empower the user's security administrators.

Steel Talon has the ability to remotely disable the phone to protect it from further intrusion as well as automatically and temporarily disable the phone based on intrusion detection.

Locally installed software on the device, and a remote server work in tandem to deliver Steel Talon. An intuitive interface design on both the device and web server allow for immediate adoption.

A KiteWire Product: KiteWire.com

Get more information

Due to the nature of our clients, any further architectural details and system features will require authorization. For more information, please e-mail steeltalon@kitewire.com or call 703-224-8090. This product is not available outside of the United States per the Export Administration Regulations (EAR) of the US Department of Commerce.

Protection for any device anywhere.

Webroot® SecureAnywhere™ Business solutions deliver the ultimate in endpoint security and protection for all your PCs, smartphones, tablets, servers and virtual machines.

Simplicity - one license covers up to ***4 devices per user.***

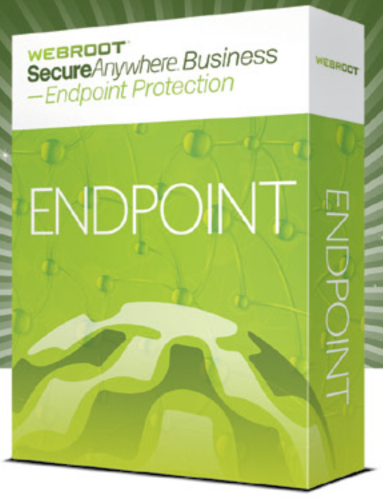
Lower Costs - as users look to BYOD you won't incur additional costs.

Total Flexibility - a single license covers desktops, laptops, smartphones, tablets, servers and virtual environments.

Powered by the Cloud - secures all your users' devices as well as the infrastructure required to support your business.

Multiple Devices - all managed with a single, intuitive web-based management console that delivers critical visibility to all user devices and every endpoint.

Get Your FREE 30-Day Trial Now!
Visit **webroot**.com or call **1-800-870-8102**



WEBROOT®
SecureAnywhere™
Business