SAMSUNG DEVELOPERS: OFFICE HOURS CHICAGO

AGENDA

Mobile Risk Drivers

Top ~~10±~~ 14+ Best Practices

Live Demo - App Hacking and Disassembly

Tools / Certs / etc

Q&A

viaForensics Overview

viaForensics is a **mobile security company** founded in 2009.

Bootstrapped with ~40 employees and a
10 person dedicated mobile security R&D team

Some of our f/oss:

YAFFS2 in TSK

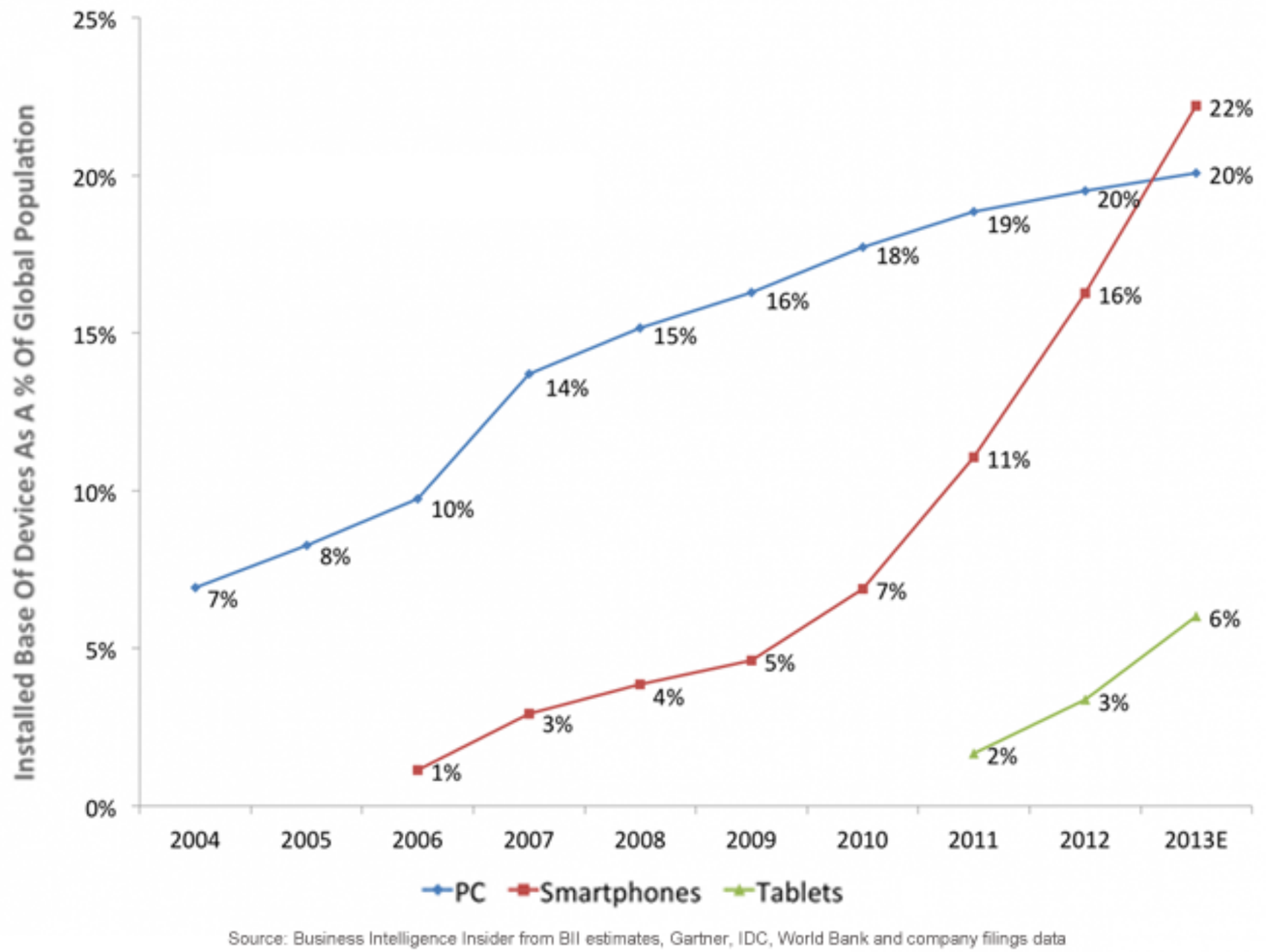
AFLogical OSE

Santoku Linux

4 Mobile Risk Drivers

(1) Mobile Growth / Audience

GLOBAL DEVICE PENETRATION PER CAPITA



(2) Control / Visibility

- Traverses many networks
- Minimal management options
- Limited visibility of activity and network
- No privilege account
  - No 3rd security tools, innovation
  - 0days can sometimes get privilege

(3a) OS Vulnerabilities

2013 Mobile pwn2own

Samsung Knox

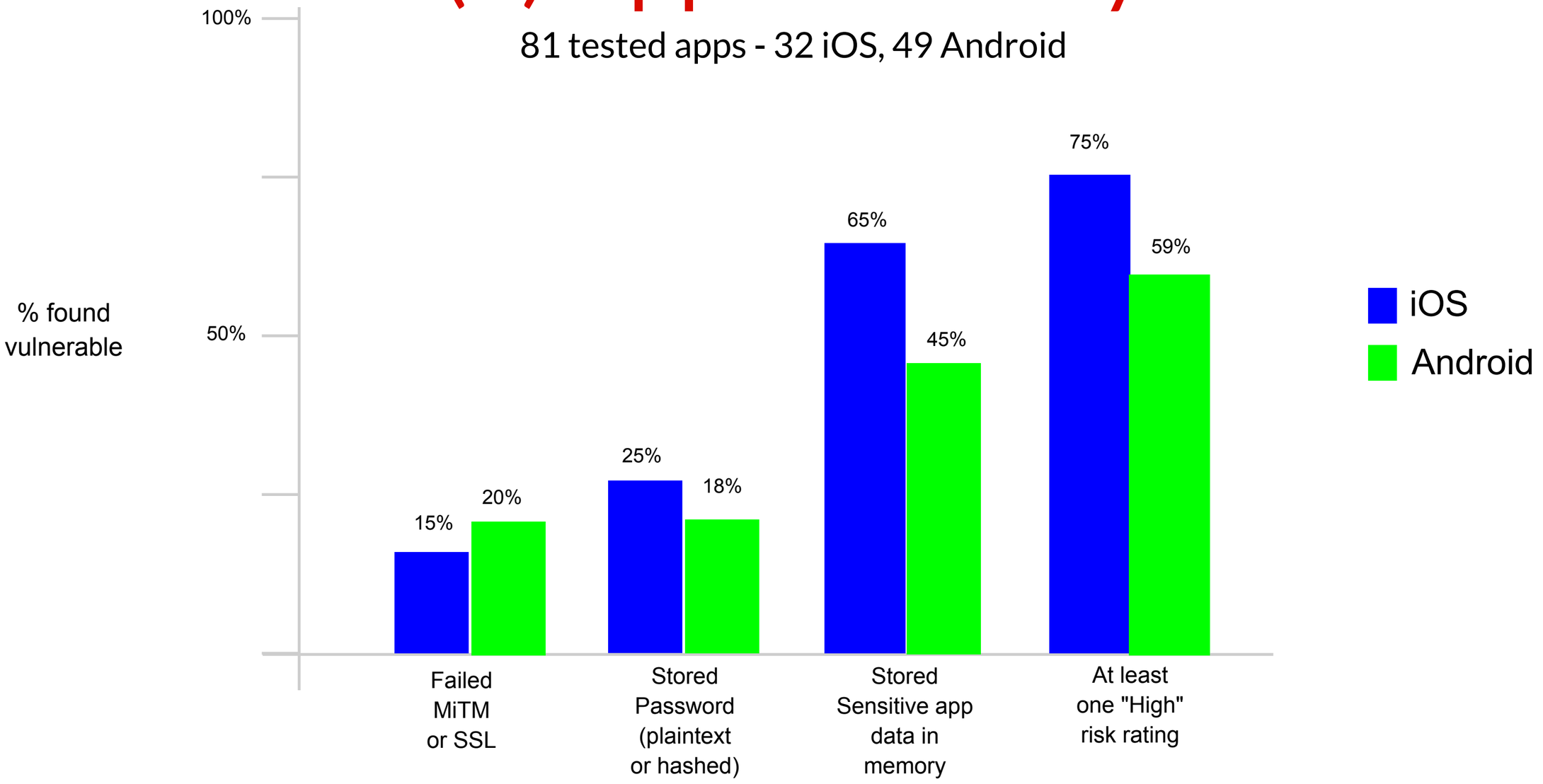
iOS 7 jailbreak with backdoors

(3b) Data

- Unprecedented access to sensitive data
  - Business
  - Personal
- Data protection techniques limited on small form factor
- Rich target

(4) App Insecurity

81 tested apps - 32 iOS, 49 Android



Industry tackling wrong problem

- Focused on malware
- Using static signatures
- Operating inside sandbox

Real mobile security issues

- App security
- Lack of visibility
- Other issues
  - Malware
  - Insider threat
  - Advanced adversaries

SECURE MOBILE DEVELOPMENT:

TOP ~~10~~± 14+ BEST PRACTICES

(note: these are in no particular order except…)

#1 – ASSUME THE WORST

The phone is rooted

—

There is a “man in the middle”

—

Other apps are malicious

—

etc ...

#2 - PROTECT APP DATA STORED LOCALLY

When possible, do not store/cache data

—

App data is often found (unencrypted) in SQLite databases stored locally on the device

—

If storage is necessary, encrypt data that is considered sensitive using password-based key derivation function (KDF):

#3 - SECURE STORAGE OF CREDENTIALS

Credentials often recovered from Preferences file (both Android/iOS)

—

iOS Keychain can be compromised

—

Bad
Credentials stored in plain text (XML/Plist)

Not so bad
Credentials stored in plain text in Keychain

Good
Hash of username and password stored

Great
Credentials never stored/password hashed during transmission

#4 – DON'T FORGET ABOUT LOG FILES

During development, debugging can be useful

—

Log files on both iOS and Android devices can be recovered,
and it is not uncommon to find user credentials/app data in a production app

—

Ensure debugging is disabled for production app builds to avoid data recovery from log files

#5 - SECURE DATA IN TRANSIT

Use SSL/TLS (HTTPS)!

—

Use secure SSL/TLS protocols (i.e. SSL v3, TLS v1.1/1.2)
and secure ciphers (128 bit or higher)

—

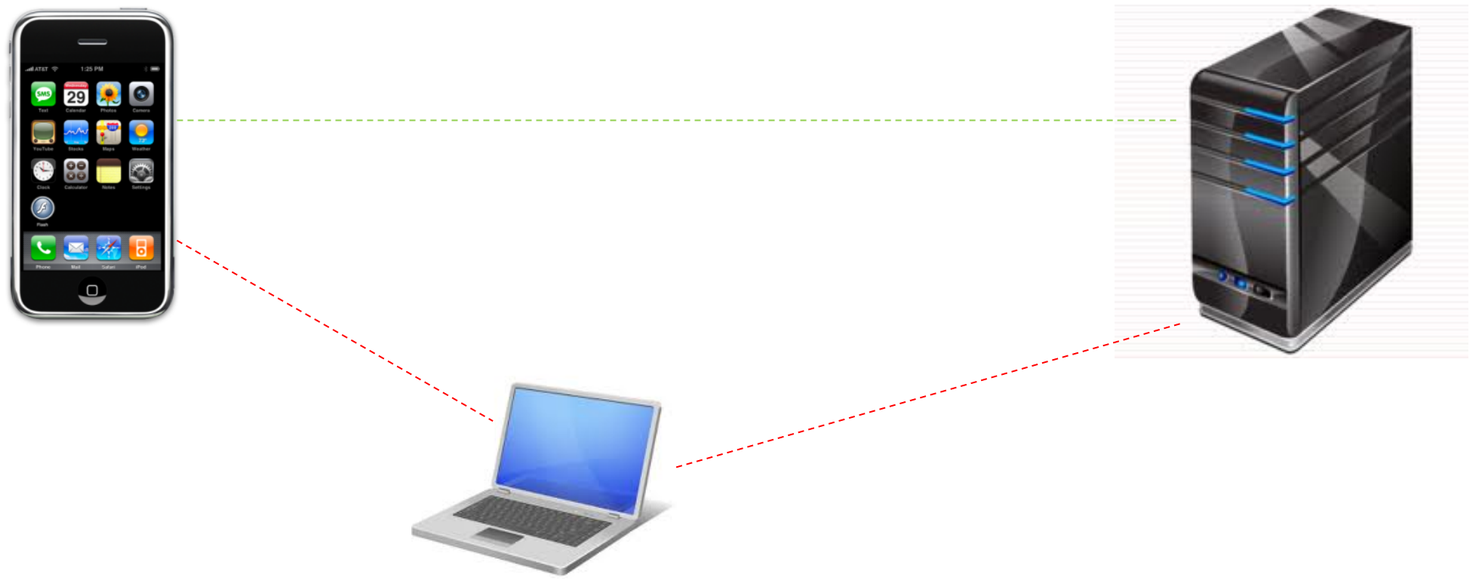
Even if app-specific encryption is used,
SSL should be implemented as additional layer of security

#6 – PERFORM CERTIFICATE VALIDATION

Validate that the certificate is authentic to prevent man-in-the-middle attack

—

SSL Cert should be fully validated and signed by a trusted root CA



#7 – IMPLEMENT CPROS CAREFULLY

Content providers can separate read and write access

Do not enable write access if not needed

—

Use record level delegation to provide the minimum access required for an operation.

For example, share a single “Instant Message” with an external application without the external app having access to all messages

#8 – PROTECT YOUR CODE

Obfuscated code (Android) with Proguard

Included with standard Android SDK; Rewrites names of packages, classes, methods, etc. with non-descript names (like a, b, c, etc).

Simply add “proguard.config” property to your “project.properties” file

—

Restrict debuggers

iOS: Specify through system call that the OS should not allow a debugger to attach to the process

Android: Set “android:debuggable=“false” in app manifest

#9 - PROPERLY IMPLEMENT SESSION TIMEOUT

Require user to enter PIN/Password after set amount of time

—

Implement both local and server-side session timeout
(local can sometimes be bypassed)

—

When timeout occurs: re-direct to login screen so app data is not displayed;
also clear all app data from memory

#10 – IMPLEMENT FILE PERMISSIONS CAREFULLY

Unless required, do not create files with permissions of

MODE_WORLD_READABLE
or
MODE_WORLD_WRITABLE

—

Any app would be able to read / write the file

#11 – VALIDATE INPUT FROM CLIENT

All input from the client should be untrusted

—

It is possible to intercept and manipulate data, even when coming from your app

—

Properly sanitize all user input before transmitting

#12 – AVOID CRASH LOGS

Crash logs can provide valuable data to attackers

For example, crash logs were used in Gingerbreak attacks to discover key address used in root

—

Test apps thoroughly for crashes

—

Build apps without warnings

#13 – CERT VALIDATION ONE STEP FURTHER: CERTIFICATE PINNING

Since a mobile app knows where it will be connecting, it can specify which CA/certificate it will accept

—

The app can restrict which CAs can issue certs for their app **or** validate that the certificate presented to the app is actually the correct certificate

—

This will help prevent compromised CAs from being trusted

http://www.thoughtcrime.org/blog/authenticity-is-broken-in-ssl-but-your-app-ha/

#14 - ASSUME THE WORST

The phone is rooted

—

There is a “man in the middle”

—

The app will crash

—

The same device has a malicious app

—

etc ...

What you can do to avoid these pitfalls:

If you are not building your app with security as part of your development life cycle then you should have someone doing reverse engineering AND penetration testing of your application for every major release.

Reverse Engineering
Security "Testing" for Other Purposes

- A security tester will analyze and/or reverse engineer an app to learn how it works and to find vulnerabilities. This is an expected and "good" use of the tools and techniques we have just seen.
- The other side of the coin is that an attacker will use the same tools and techniques to learn how it app works and find vulnerabilities.
  - An attacker may want to learn how to compromise an app to steal data, perform malicious actions against the user, or otherwise compromise a device it runs on.
  - An attacker may want to alter the app and release a modified version so that unsuspecting users will install an app that is legitimate but contains some hidden, malicious, functionality.
- Regardless of the intent, reverse engineering consists of studying the app to learn something about it, its control flow, the data that it processes, the keys used for cryptography, any included passwords or other information, or anything else about the app. If something is part of the app, it can be reverse engineered.

1. Ensure App components (e.g content provider, services, and activities) aren't exported by accident - listen to the lint tool checks! Make them private (non-exported) using the android:exported="false" attribute in the manifest.

2. Use the latest crypto libraries - BouncyCastle is default or pull the latest SpongyCastle.

3. Encrypt sqlite databases using - SQLCipher

4. Generate encryption keys via KDF and ensure proper timeout in memory.

5. Add tamper check for debuggable and verify apk signature [CRC or Hash check] context.GetApplicationInfo() method can allow an app to read its own files.

6. Obfuscate with Proguard or Dexguard.

7. Remove all of your debug logs with Proguard or Dexguard.

8. Validate SSL certs using Onion Kit or use Cert Pinning.

9. Protect your private keys at all costs. (Password protected, Full disk encrypted, machines that are not connected to the internet.)

10. Do not overpermission your app or any of its components. Occam's Razor.

apktool

apktool is a tool for reverse engineering Android apk, it disassembles the code to .smali files, decoding also the resources contained into the apk.

It can also repackage the applications after you have modified them.

We can run it on a Malware sample:

$ apktool d **ru.blogspot.playsib.savageknife.apk** savage_knife_apktool/
I: Baksmaling...
I: Loading resource table...
I: Loaded.
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/santoku/apktool/framework/1.apk
I: Loaded.
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Done.
I: Copying assets and libs…

Source: https://code.google.com/p/android-apktool/

apktool -> smali

—

We can grep for known sensible method calls and strings

—

$ grep -R getDeviceId .

./smali/com/mobidisplay/advertsv1/AdvService.smali: invoke-virtual {v1}, Landroid/telephony/TelephonyManager;->getDeviceId()Ljava/lang/String;

—

$ grep -R BOOT_COMPLETED .

./AndroidManifest.xml: <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
./AndroidManifest.xml: <action android:name="android.intent.action.BOOT_COMPLETED" />
./smali/com/mobidisplay/advertsv1/BootReceiver.smali: const-string v2, "android.intent.action.BOOT_COMPLETED"

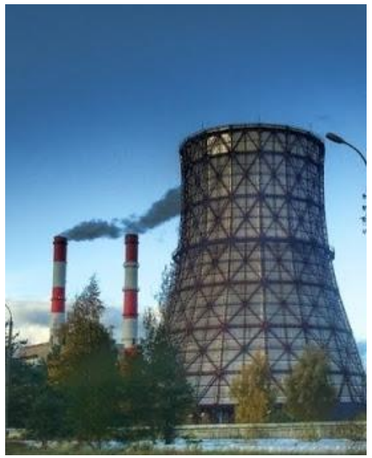
apktool -> smali

We can manually analyze the disassembled smali code provided by apktool.

For example here we see a broadcast receiver that will listen for BOOT_COMPLETED intents and react to them starting a service in the application.

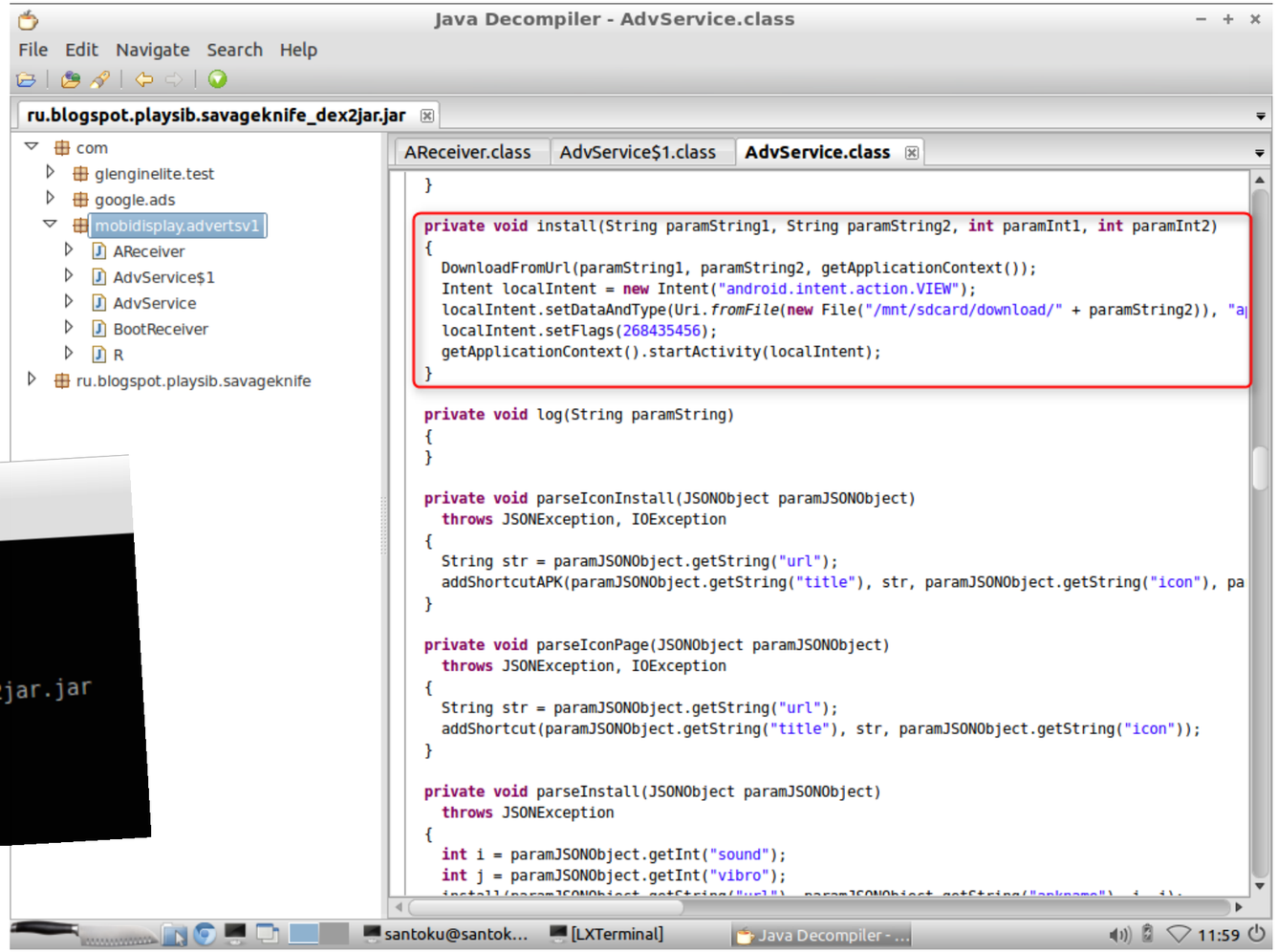
```
# virtual methods
.method public onReceive(Landroid/content/Context;Landroid/content/Intent;)V
    .locals 3
    .parameter "context"
    .parameter "intent"

    .prologue
    .line 16
    invoke-virtual {p2}, Landroid/content/Intent;->getAction()Ljava/lang/String;

    move-result-object v1

    const-string v2, "android.intent.action.BOOT_COMPLETED"

    invoke-virtual {v1, v2}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result v1

    if-eqz v1, :cond_1

    .line 18
    new-instance v0, Landroid/content/Intent;

    invoke-direct {v0}, Landroid/content/Intent;-><init>()V

    .line 19
    .local v0, serviceIntent:Landroid/content/Intent;
    const-string v1, "com.mobidisplay.advertsv1.AdvService"

    invoke-virtual {v0, v1}, Landroid/content/Intent;->setAction(Ljava/lang/String;)Landroid/content/Intent;

    .line 20
    invoke-virtual {p1, v0}, Landroid/content/Context;->startService(Landroid/content/Intent;)Landroid/content/ComponentName;
```

BadNews Malware Sample -> Dex2Jar -> JD-GUI



Contagio MiniDump
Malware Repository
contagiominidump.blogspot.com

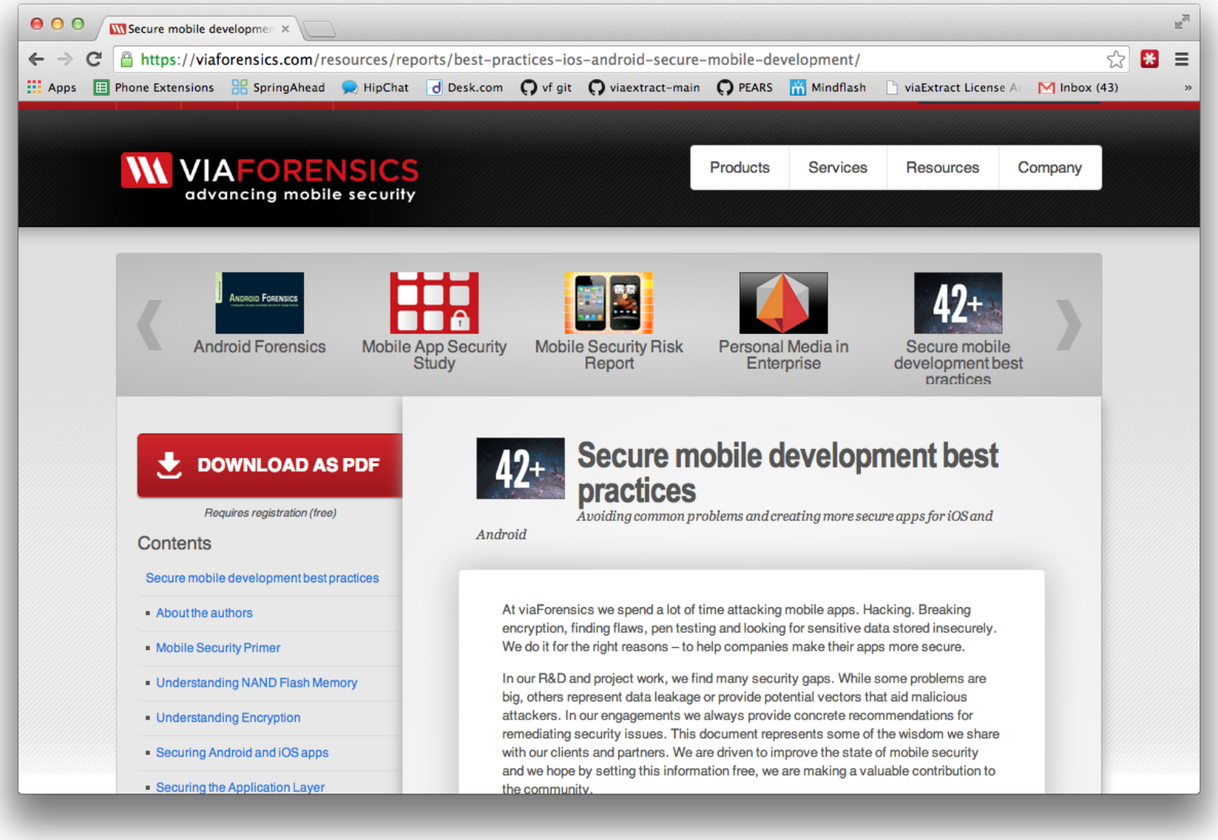
```
santoku@santoku: ~/badnews
File  Edit  Tabs  Help
santoku@santoku:~/badnews$ ls
ru.blogspot.playsib.savageknife.apk
santoku@santoku:~/badnews$ dex2jar ru.blogspot.playsib.savageknife.apk
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.9
dex2jar ru.blogspot.playsib.savageknife.apk -> ru.blogspot.playsib.savageknife_dex2jar.jar
Done.
santoku@santoku:~/badnews$ ls
ru.blogspot.playsib.savageknife.apk  ru.blogspot.playsib.savageknife_dex2jar.jar
santoku@santoku:~/badnews$
```



LIVE DEMO - SANTOKU

RESOURCES

42+ BEST PRACTICES



https://viaforensics.com/resources/reports/

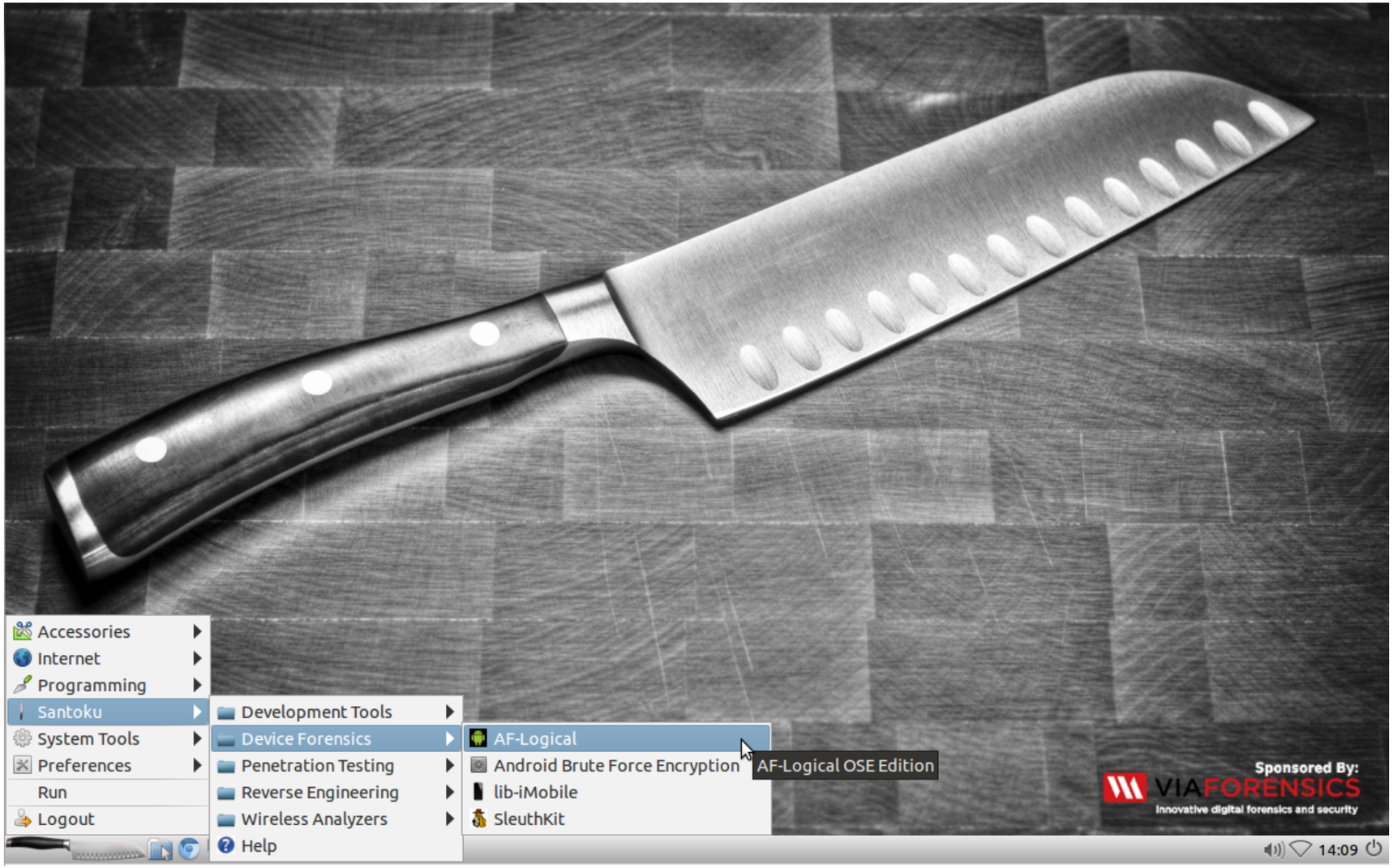
MOBILE APP SECURITY+ CERTIFICATION

Developed by viaForensics and Comptia

"The **Mobile App Security+ Certification Exam (Android Edition)** will certify that the successful candidate has the knowledge and skills required to securely create a native Android mobile application, while also ensuring secure network communications and back-end Web services."

https://viaforensics.com/services/training/secure-developer-certification-ios-android/

SANTOKU LINUX



SANTOKU LINUX

Development Tools:

- Android SDK Manager
- AXMLPrinter2
- Fastboot
- Heimdall
- Heimdall GUI
- SBF Flash

Wireless Analyzers:

- Chaosreader
- dnschef
- DSniff
- TCPDUMP
- Wireshark
- Wireshark (As Root)

Penetration Testing:

- Burp Suite
- Ettercap
- Mercury
- nmap
- OWASP ZAP
- SSL Strip
- w3af (Console)
- w3af (GUI)
- Zenmap (As Root)

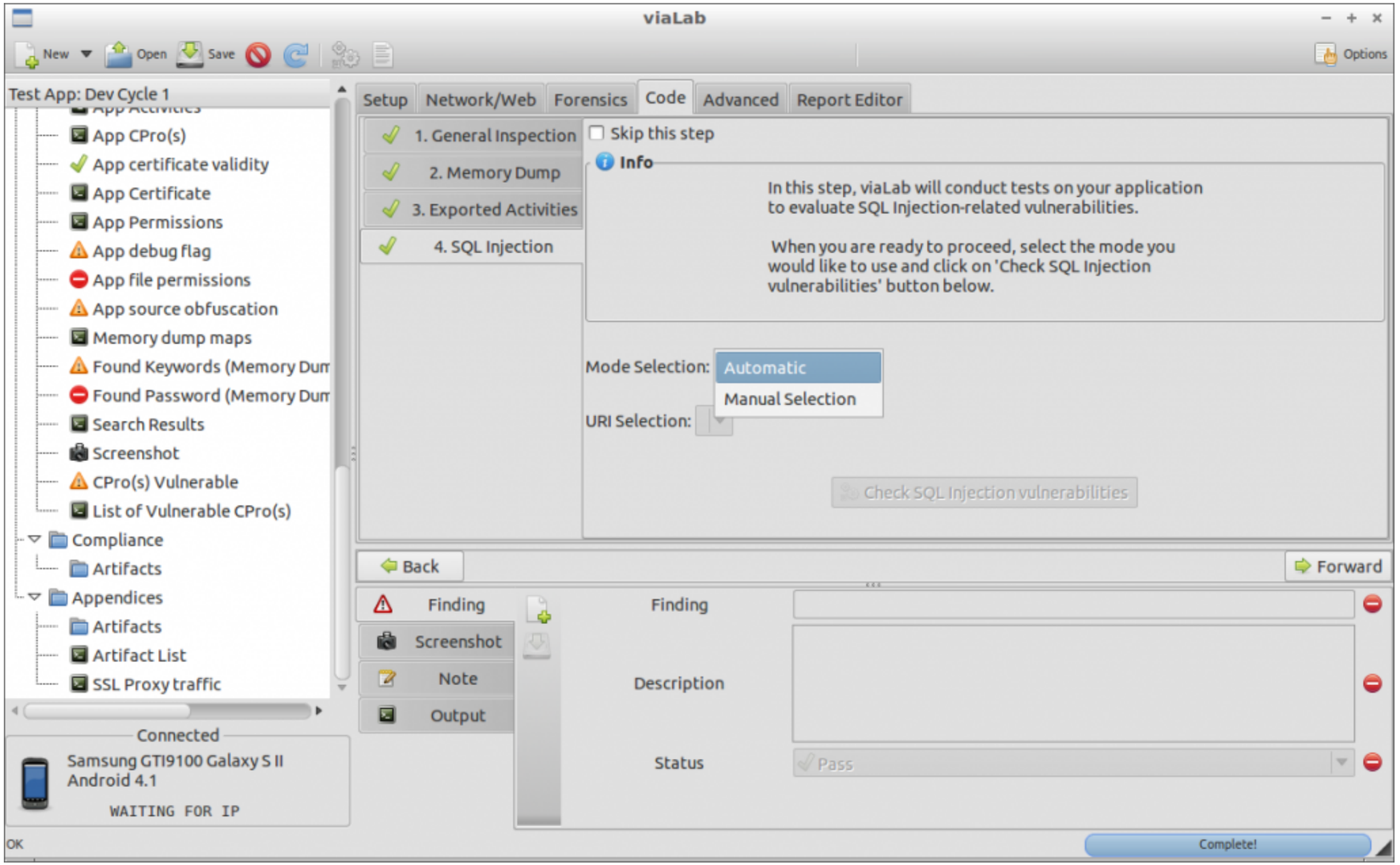
Device Forensics:

- AFLogical Open Source Edition
- Android Brute Force Encryption
- ExifTool
- iPhone Backup Analyzer (GUI)
- libimobiledevice
- scalpel
- Sleuth Kit

Reverse Engineering:

- Androguard
- Antilvl
- APK Tool
- Baksmali
- Dex2Jar
- Jasmin
- JD-GUI
- Mercury
- Radare2
- Smali

Automated Testing



Mobile App Testing

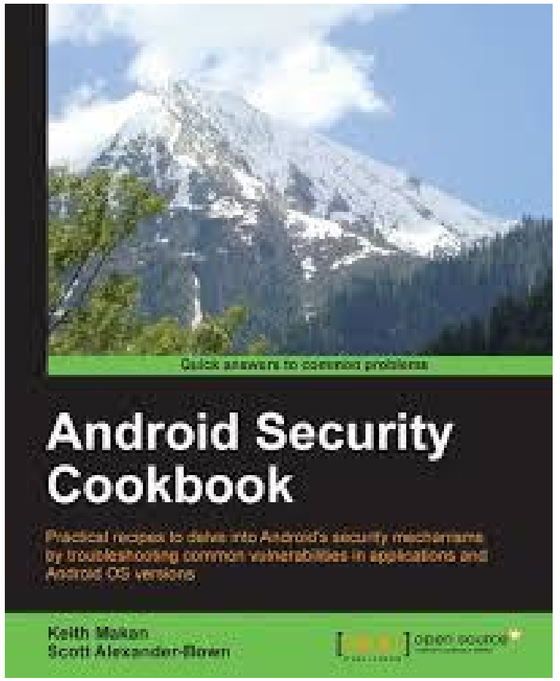
- Integrate into the SDLC
- Test 3rd party apps
- Malware analysis

Make security testing part of the SDLC - from the beginning

SPECIAL THANKS

@scottyab

and the rest of the @viaForensics team



Q&A

https://viaforensics.com/company/careers/

CONTACT INFO

Bob Hanson	James Drake	Kevin Swartz
Executive Director, Global Sales	Mobile Security Researcher	Mobile Security Analyst
bhanson@viaforensics.com	jdrake@viaforensics.com	kswartz@viaforensics.com
+1.847.858.9280	@JMDlux	@kevinswartz_1

viaForensics:

312-878-1100 | viaforensics.com | @viaforensics