

Practical Android Application Exploitation

Dinesh Shetty

Security Innovation

dinezh.shetty@gmail.com

Twitter : [@din3zh](https://twitter.com/din3zh)

Aditya Gupta

Attify, Inc.

adi@attify.com

Twitter : [@adi1391](https://twitter.com/adi1391)

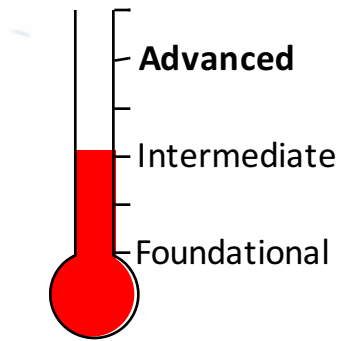


SECURITY INNOVATION

About This Workshop

- Description
 - Learn more about Android application security
 - Understand tools and techniques to exploit Android applications
 - Get skills to perform reverse engineering, static and dynamic tests, binary analysis and other conventional attack vectors
 - Comprises both demos + hands-on exercises
 - Hands-on with InsecureBankv2 and other custom applications

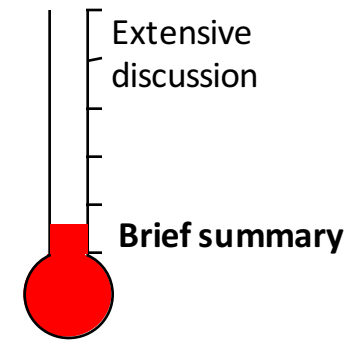
Workshop Focus



Technical Level



**Discovery
(Testing)**



**Remediation
(Design, Coding,
Implementation Guidance)**

- To help you make the most of this workshop , these thermometers give you an idea of the technical level and content focus

Security conferences



WHO AM I ?



- Leads Mobile Security Testing CoE at Security Innovation
- Mobile and Embedded application pentesting
- Knowledgeable in all phases of the secure software development life cycle (SDLC)
- Security Speaker and Trainer at multiple conferences such as BlackHat, OWASP, Brucon, Defcon, ClubHack etc.
- Research work has been published at PacketStorm, Exploit-db, SecurityXploded, ClubHack, Pentest Magazine etc.

WHO AM I WITH?



- Founder and CEO, Attify
- Main role - Break and Exploit any IoT device
- Leads 75+ IoT pentests in past 3 years
- Security Speaker and Trainer at numerous security conferences such as BlackHat, OWASP AppSec, Toorcon, HackInParis, Phdays, ClubHack, Nullcon etc.
- Authored a book on mobile security titled “Learning Pentesting for Android Devices”

Additional Note

- The printed and uploaded copies might be different from the ones used in class
- For reference : Use the updated slides pdf shared with you during the session
- Feel free to spread the word about the training 😊
- Reach out to us for queries or issues during the class

A complex network diagram with numerous blue nodes of varying sizes connected by thin blue lines, forming a web-like structure that fills the background of the slide.

Workshop Agenda

- **Module 1: Mobile Security Introduction**
- **Module 2: Android Primer**
- **Module 3: Android Security Basics**
- **Module 4: Android Vulnerabilities**



SECURITY INNOVATION



Module 1: Mobile Security Introduction

- Basic Tools for Mobile Application Security Testing
- Why Android?
- Types of Android Applications
- Testing Challenges for Mobile Applications
- Mobile Security Threat Landscape
- Attack Surfaces

Mobile Security Introduction

Your Tools: Before and After Training



Image Source: http://ecx.images-amazon.com/images/I/91x7VVI7NVL._SL1500_.jpg



Image Source: <http://cdn.business2community.com/wp-content/uploads/2013/10/inbound-marketing-toolbox.jpg>

Mobile Security Introduction

Why Android?

- This course focuses on Android because it is the market leader and its share is growing
- Low cost Android phones are one of the major reasons for this increase in market share

Operating System	1Q16 Units	1Q16 Market Share (%)	1Q15 Units	1Q15 Market Share (%)
Android	293,771.2	84.1	264,941.9	78.8
iOS	51,629.5	14.8	60,177.2	17.9
Windows	2,399.7	0.7	8,270.8	2.5
Blackberry	659.9	0.2	1,325.4	0.4
Others	791.1	0.2	1,582.5	0.5
Total	349,251.4	100.0	336,297.8	100.0

Source: Gartner (May2016)



Module 1: Mobile Security Introduction

- Basic Tools for Mobile Application Security Testing
- Why Android?
- Types of Android Applications
- Testing Challenges for Mobile Applications
- Mobile Security Threat Landscape
- Attack Surfaces

Mobile Security Introduction

We are Here to Avoid This...

Fandango, Credit Karma Settle FTC Charges that They Deceived Consumers By Failing to Securely Transmit Sensitive Personal Information

Mobile Apps Placed Credit Card Details, Credit Report Data, Social Security Numbers at Risk

FOR RELEASE

March 28, 2014

TAGS: deceptive/misleading conduct | Technology | Bureau of Consumer Protection | Consumer Protection | Privacy and Security | Consumer Privacy

Two companies have agreed to settle Federal Trade Commission charges that they misrepresented the security of their mobile apps and failed to secure the transmission of millions of consumers' sensitive personal information from their mobile apps.

The FTC alleged that, despite their security promises, Fandango and Credit Karma failed to take reasonable steps to secure their mobile apps, leaving consumers' sensitive personal information at risk. Among other things, the complaints charge that Fandango and Credit Karma disabled a critical default process, known as SSL certificate validation, which would have verified that the apps' communications were secure.

As a result, the companies' applications were vulnerable to "man-in-the-middle" attacks, which would allow an attacker to intercept any of the information the apps sent or received. This type of attack is especially dangerous on public Wi-Fi networks such as those at coffee shops, airports and shopping centers.

Fandango

The Fandango Movies app for iOS allows consumers to purchase movie tickets and view show times, trailers, and reviews. According to the FTC's complaint, the Fandango Movies app assured consumers, during checkout, that their credit card information was stored and transmitted securely. Despite this promise, for almost four years – from March 2009 until February 2013 – the company disabled SSL certificate validation and left consumers that used its app to make mobile ticket purchases vulnerable to man-in-the-middle attacks.

The complaint alleges that Fandango could have easily tested for and prevented the vulnerability, but failed to perform the basic security checks that would have caught the issue. In addition, the complaint charges that Fandango failed to have an adequate process for receiving vulnerability reports from security researchers and other third parties, and as a result, missed opportunities to fix the vulnerability.

Credit Karma

The Credit Karma Mobile app for iOS and Android allows consumers to monitor and evaluate their credit and financial status. In its complaint, the FTC alleges that Credit Karma assured consumers that the company followed "industry-leading security precautions," including the use of SSL to secure consumers' information. Despite these promises, the company disabled SSL certificate validation and left consumers that used its credit-monitoring app vulnerable to man-in-the-middle attacks.

According to the FTC, Credit Karma could have easily prevented the vulnerability with basic tests, but did not perform an adequate security review of its iOS app before release. Even after a user warned Credit Karma about the vulnerability in its iOS app, the company failed to test its Android app before launch. As a result, one month after receiving a warning about the issue, the company released its Android app with the very same vulnerability. The complaint charges that Credit Karma failed to appropriately test or audit its apps' security and failed to oversee the security practices of its application development firm.

Settlements

The settlements require Fandango and Credit Karma to establish comprehensive security programs designed to address security risks during the development of their applications and to undergo independent security assessments every other year for the next 20 years. The settlements also prohibit Fandango and Credit Karma from misrepresenting the level of privacy or security of their products and services.



Module 1: Mobile Security Introduction

- Basic Tools for Mobile Application Security Testing
- Why Android?
- Types of Android Applications
- Testing Challenges for Mobile Applications
- Mobile Security Threat Landscape
- Attack Surfaces

Types of Android Applications



Module 1: Mobile Security Introduction

- Basic Tools for Mobile Application Security Testing
- Why Android?
- Types of Android Applications
- Testing Challenges for Mobile Applications
- Mobile Security Threat Landscape
- Attack Surfaces

Mobile Security Introduction

Mobile Testing Challenges - Application Complexity

- Browser-Based Applications
 - Easier to test
 - Using a normal browser, you can change the browser user agent, and the server will send the mobile version
 - Use an emulator/device to read persistent cookies or local data storage
 - All the attacks for traditional web applications apply to browser-based mobile applications
- Native Applications
 - Need a real device or an emulator (Android) / simulator (iOS) to do testing
 - The OS and frameworks are optimized, resulting in poor code readability when reverse engineering binaries
 - Tools are harder to use
 - Not all web based attacks are valid to native applications
 - E.g.: in many cases XSS is not a valid case for a pure native application which does not have a web front-end
- Hybrid Applications
 - Challenges from both browser-based and native applications!
 - Need a real device or an emulator/simulator to do testing
 - All native application test cases + web application test cases are valid for hybrid applications

Mobile Security Introduction

Mobile Testing Challenges - Platform Complexity

- The large number of permutations of OS, hardware, networks, configurations, and software adds extra complexity to security testing
- It is challenging to test on a platform with limited memory, storage, and power
- Multiple releases of operating systems
- Different hardware/chipsets
 - Different hardware sensors, cameras, etc.
- Complex configurations
- Different networks
 - Wi-Fi is easy to test with proxy
 - 3g or 4g network traffic is hard to intercept + Illegal in many countries



SECURITY INNOVATION



Module 1: Mobile Security Introduction

- Basic Tools for Mobile Application Security Testing
- Why Android?
- Types of Android Applications
- Testing Challenges for Mobile Applications
- Mobile Security Threat Landscape
- Attack Surfaces



SECURITY INNOVATION



Mobile Security Introduction Mobile Security Threat Landscape

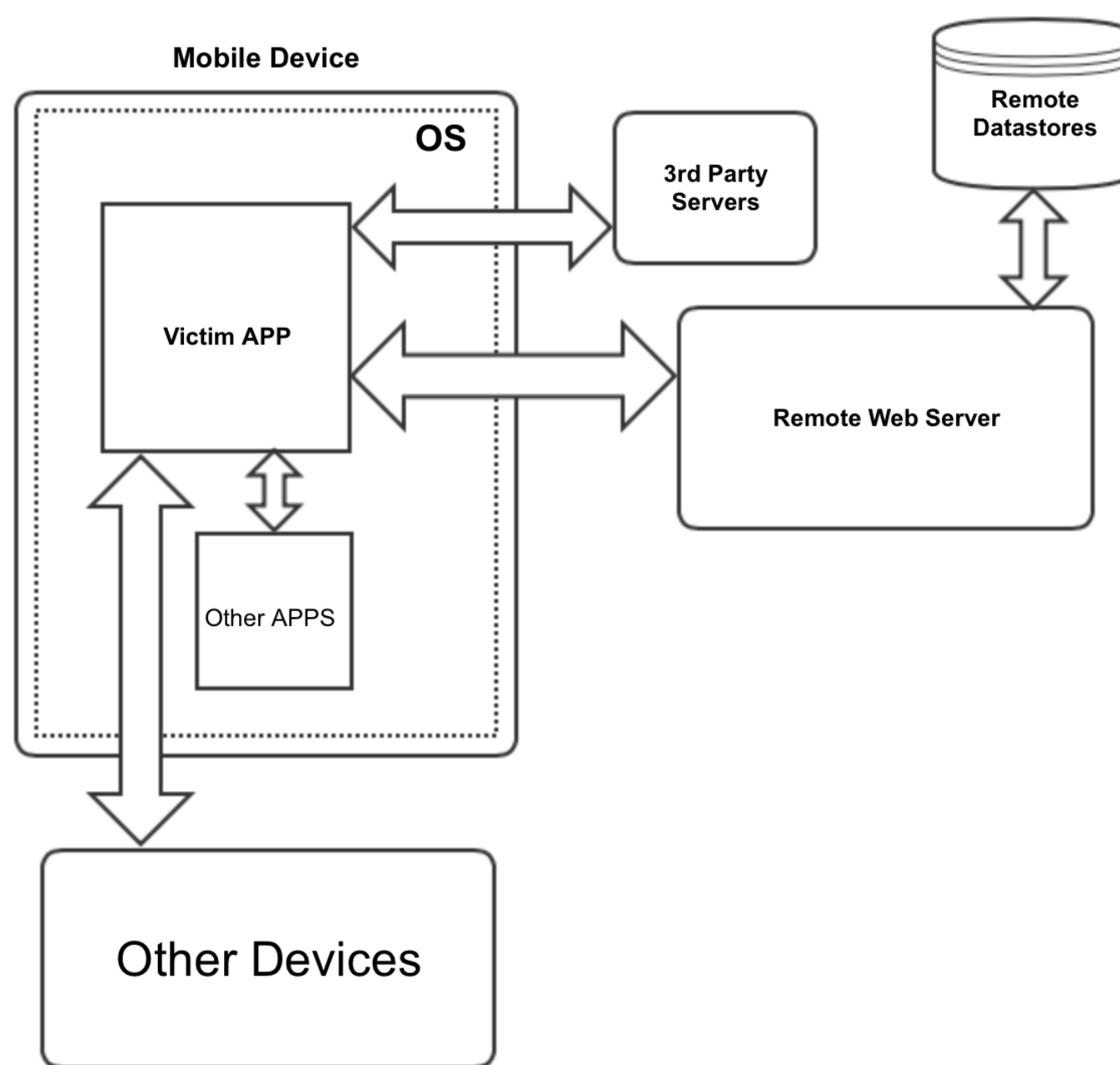


Image Source:
<https://drawception.com/pub/panels/2013/12-2/2ZaA95WLG4-8.png>



SECURITY INNOVATION





Module 1: Mobile Security Introduction

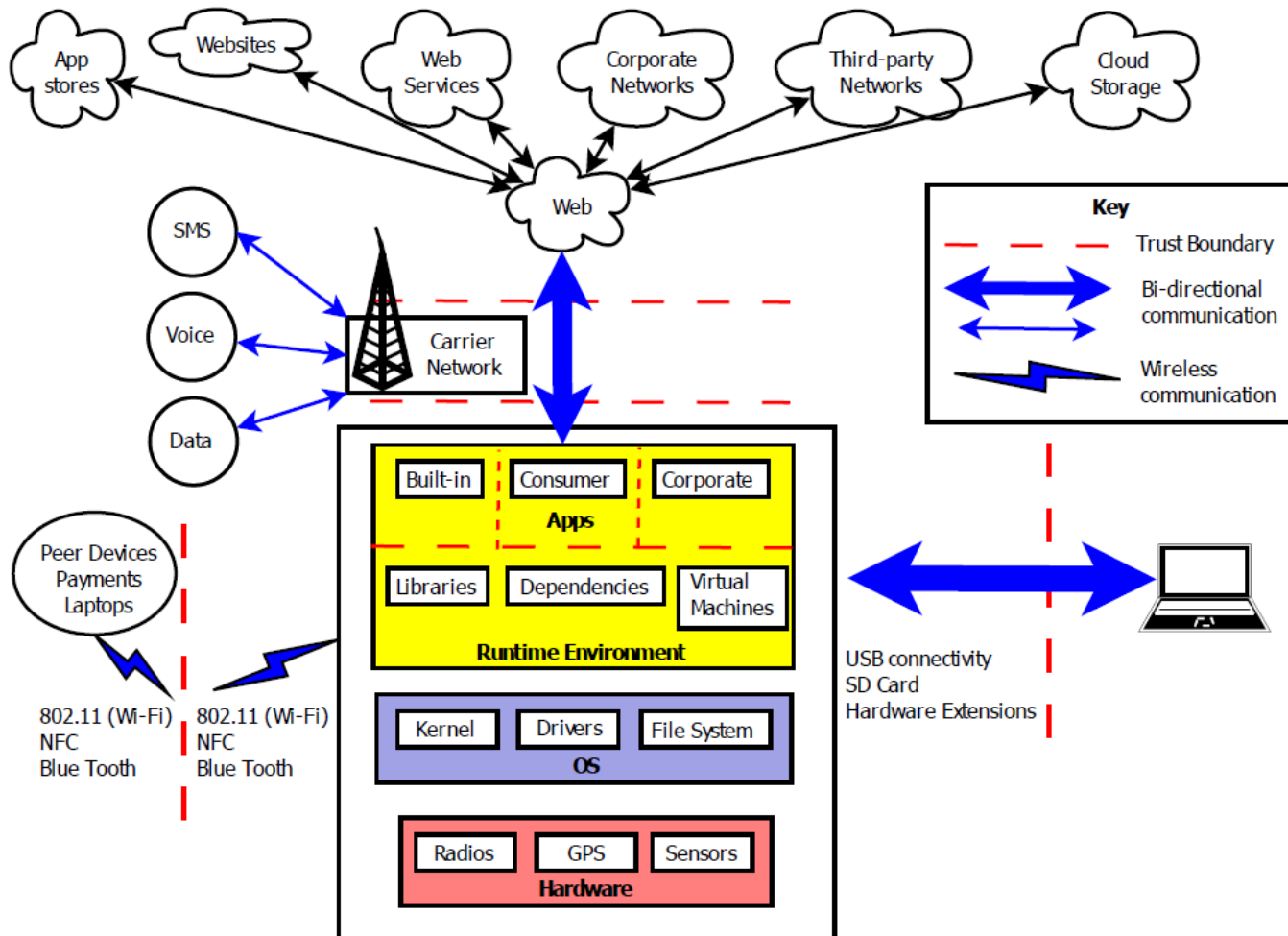
- Basic Tools for Mobile Application Security Testing
- Why Android?
- Types of Android Applications
- Testing Challenges for Mobile Applications
- Mobile Security Threat Landscape
- Attack Surfaces

Mobile Security Introduction Mobile Avenues of Attack

- Sensitive information stored in the APK client Binary
- Vulnerabilities between APP <=> Server
- Vulnerabilities between APP <=> 3rd Party Servers
- Vulnerabilities related to data stored by the Android Operating System
- Vulnerabilities related to data stored in application sandbox
- Static Vulnerabilities in the APP
- Vulnerabilities in the APP during runtime
- Vulnerabilities between APP <=> APP
- Vulnerabilities in the Web Server
- Vulnerabilities in the Remote Datastores

Mobile Security Introduction

Mobile Device Infrastructure and Trust Boundaries



- The mobile environment is very complex
 - This is particularly notable in a BYOD environment
- Many trust boundaries have to be considered from an application development standpoint
- Many opportunities for failure
- Modified from: *Mobile OWASP Top 10*
<http://www.youtube.com/watch?v=pQAIwyXAZBk>



Workshop Agenda

- Module 1: Mobile Security Introduction
- **Module 2: Android Primer**
- Module 3: Android Security Basics
- Module 4: Android Vulnerabilities



SECURITY INNOVATION





Module 2: Android Primer

- Introducing the Android IDE and SDK
- Understanding the Android Package Structure
- Android Security Model
- Android Debug Bridge
- Android File Structure
- Android Rooting Background
- Understanding Android Components



SECURITY INNOVATION



Android Primer

Introducing the Android IDE and SDK



TASK

- Write a simple Hello World application in Android Studio
- Write a simple application that has a text field and a button on it
- Modify the code in a way that the text view is updated on the basis of the input given in the text field
- Play 5 mins in Android Studio and explore the different options



Module 2: Android Primer

- Introducing the Android IDE and SDK
- Understanding the Android Package Structure
- Android Security Model
- Android Debug Bridge
- Android File Structure
- Android Rooting Background
- Understanding Android Components



SECURITY INNOVATION



Android Primer

Understanding the Android Package Structure

- Application are developed using Android Studio
- Distributed as .apk files through the Google Play Store
- Applications are signed using Developer certificate
- No DRM signing by Google

Understanding the Android Package Structure



TASK

- Install *InsecureBankv2.apk*
- Unzip *InsecureBankv2.apk* to find the corresponding resource files
- Find *classes.dex* from the extracted folder
- Can you read the actual contents of *classes.dex* file?
- Can you read the *AndroidManifest.xml* file?



Module 2: Android Primer

- Introducing the Android IDE and SDK
- Understanding the Android Package Structure
- **Android Security Model**
- Android Debug Bridge
- Android File Structure
- Android Rooting Background
- Understanding Android Components



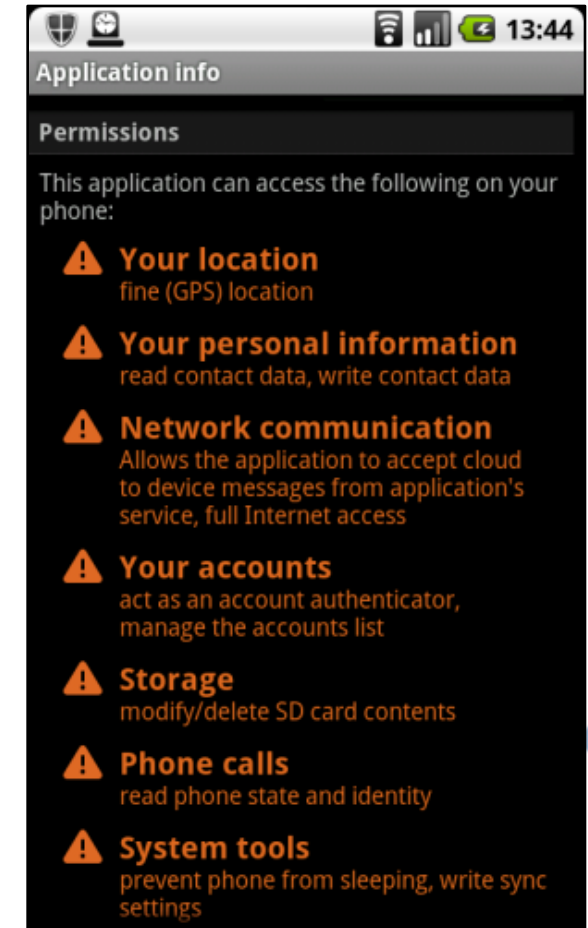
SECURITY INNOVATION



Android Primer

Android Security Model

- Permissions (a.k.a. manifest or Android perms)
 - As opposed to file or object permissions
 - Permissions are declared in the AndroidManifest.xml file
 - An app can only use the declared permissions
 - No permissions are granted by default (default deny)
 - Each application must explicitly request permissions before it can be installed
- The list of possible permissions is quite extensive:
<http://developer.android.com/reference/android/Manifest.permission.html>



Android Primer

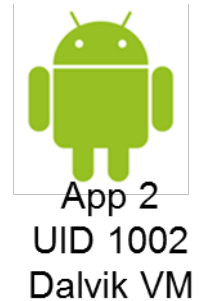
Android Security Model

- Android code must be signed
 - Typically done with self-signed certificates
 - No need for a certificate authority
 - Easy for developers to generate without anyone's permission
- Cryptographic signatures are used to...
 - Verify the origin and integrity of apps
 - Establish trust relationships between apps
- Allows developers to update their applications easily
 - Applications signed with the same key (and therefore signed by the same developer) can ask to run with the same UID
 - Simplifies the updating and patching process

Android Primer

Android Security Model

- The Android OS is a multi-user Linux system
 - Each application is run within a Dalvik Virtual Machine in a sandbox
 - Each app has a unique UID:GID
 - Each application runs as a different user
 - Any program can ask Activity Manager to launch almost any other application, which runs with the application's UID



SECURITY INNOVATION



Android Primer

Android Security Model

- Permissions for all the files in an application are set so that only the user ID assigned to that application can access the files
- Each process has its own VM (sandbox), so an app's code runs in isolation from other apps
 - By default, every application runs in its own Linux process
 - Android starts the process when any of the application's components need to be executed, then shuts down the process when it's no longer needed/when the system must recover memory for other applications



Module 2: Android Primer

- Introducing the Android IDE and SDK
- Understanding the Android Package Structure
- Android Security Model
- **Android Debug Bridge**
- Android File Structure
- Android Rooting Background
- Understanding Android Components



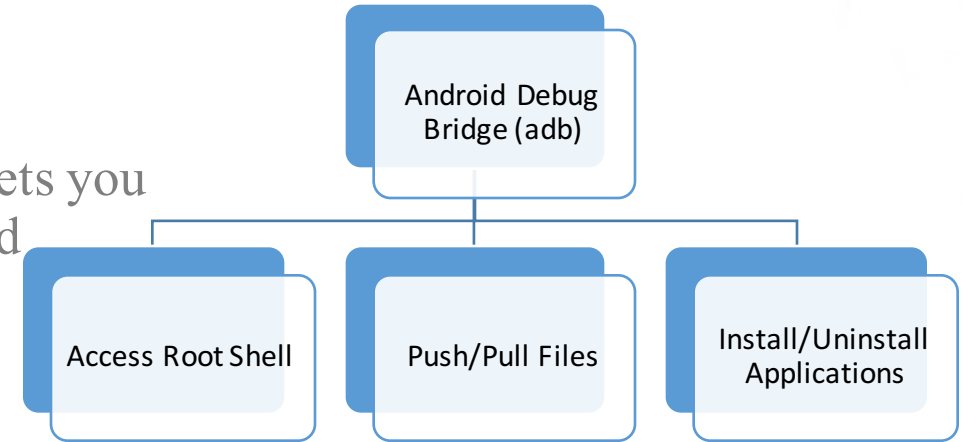
SECURITY INNOVATION



Android Primer

Android Debug Bridge (ADB)

- The Android OS helps control and enforce security
- Android Debug Bridge (**adb**) is a command line tool that lets you communicate with an emulator instance or with a connected Android device
 - Used for testing; uploading payloads and tools
 - For more information on the various commands see <http://developer.android.com/tools/help/adb.html>
- In a non-rooted device, certain parts of the directory structure are blocked
 - Analogous to Linux, where **sudo** must be run to gain access
 - Rooting a device allows root access to those directories
- The Android emulators are rooted by default
 - The Developer has access to everything



Android Primer

Android Debug Bridge (ADB)



SECURITY INNOVATION

attify

Android Primer

Android Command Reference

- Open the excel file `AndroidCommandReference.xls`

TASK

- Launch an avd
- Run *ps aux* to view the current process details.
 - What weird fact do you notice?



Module 2: Android Primer

- Introducing the Android IDE and SDK
- Understanding the Android Package Structure
- Android Security Model
- Android Debug Bridge
- **Android File Structure**
- Android Rooting Background
- Understanding Android Components



SECURITY INNOVATION



Android Primer

Android File Structure



SECURITY INNOVATION

attify

Android Primer

Android File Structure - Important Locations

- **/data/data** : All applications data installed by user
- **/data/app** : APKs of applications installed by user
- **/system/app** : System Applications
- **/data/system** : Files such as gesture.key
- **/data/local/tmp** : Writeable (without root)



Module 2: Android Primer

- Introducing the Android IDE and SDK
- Understanding the Android Package Structure
- Android Security Model
- Android Debug Bridge
- Android File Structure
- **Android Rooting Background**
- Understanding Android Components



SECURITY INNOVATION



Android Primer

Rooting Background

- Rooting is the process of gaining root access to the entire device
 - Gives access to restricted file system areas on the device
 - Allows you to run apps that require additional privileges or access to certain restricted system settings
 - Alters the logic of the platform at a low level
 - Can result in a bricked device and voids the warrantee
 - The emulator is rooted by default
 - For non-rooted device, you can shell into it but can't get to secure areas
 - A rooted/jailbroken device allows malware to run on the device, too
- **Rooting** is the Android term
- **Jailbreaking** is the iOS term
- **Unlocking** refers to configuring the phone to use any carrier (very different than rooting/jailbreaking, but might be confused with rooting/jailbreaking)



Android Primer

Rooting and Jailbreaking Background

- Android applications store data in the application sandbox which is not accessible to public (but is available to root and the application itself)
 - Without root access, a tester cannot access the application sandbox and cannot see what data is being stored and how is it stored
 - Most the system level files are owned by root
- A code-assisted pentest can find more issues than you can find with just a jailbroken device
 - Having access to the jailbroken device + the source code is the ideal testing approach
 - For example, weak encryption can not be detected by a simple pentest... in such cases, having access to the code makes finding some types of vulnerabilities much faster

Everybody stand back!



I'M ROOTING MY DROID!

Image Source: <http://www.ijailbreak.com/wp-content/uploads/2011/08/root-android-device.jpg>



Module 2: Android Primer

- Introducing the Android IDE and SDK
- Understanding the Android Package Structure
- Android Security Model
- Android Debug Bridge
- Android File Structure
- Android Rooting Background
- Understanding Android Components



SECURITY INNOVATION



Android Primer

Android App Components

- Activities
- Services
- Intents
- Content Providers
- Broadcast Receivers

Android Primer

Identifying Android app components

- Start any application analysis with AndroidManifest.xml
- Look for components such as `<activity>`, `<provider>`, `<receiver>` and so on.
- Use the `android:name` element to determine the exact name of the component, which will be used for later analysis

```
Applications Places System
AndroidManifest.xml (~/.Lab/Android-Ins...Bankv2/InsecureBankv2/app/src/main) - VIM
File Edit View Search Terminal Help

android:label="@string/app_name"
android:theme="@android:style/Theme.Holo.Light.DarkActionBar">
<!--
android:theme="@style/AppTheme"-->
<activity
    android:name=".LoginActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".FilePrefActivity"
    android:label="@string/title_activity_file_pref"
    android:windowSoftInputMode="stateVisible|adjustResize|adjustPan">
</activity>
<activity
    android:name=".DoLogin"
    android:label="@string/title_activity_do_login" >
</activity>
<activity
    android:name=".PostLogin"
    android:exported="true"
    android:label="@string/title_activity_post_login" >
</activity>
<activity
```

58,1 44%



Android Primer

Android App Components – Activities

- Visual Screens of an Android application
- Contains of different layouts
- Anything you could interact with
- Wallet applications

Android Primer

Android App Components – Services

- Background events in an Android application
- Eg : Downloading files / Playing Music

Android Primer

Android App Components – Intents

- To bind different android components
- Used to perform different kind of actions
- Changing activities, invoking activities in another application, starting an action etc.
- Intent Filters

Android Primer

Android App Components - Content Providers

- Storing, Retrieving and Sharing application data
- Acts as a middle layer
- SQLite, XML, PlainText
- With the prefix “content://”



Android Primer

Android App Components – Broadcast Receivers

- Receives broadcast from various events
- Could be from the phone or another app
- Used by a lot of malwares



Workshop Agenda

- Module 1: Mobile Security Introduction
- Module 2: Android Primer
- **Module 3: Android Security Basics**
- Module 4: Android Vulnerabilities

Module 3: Android Security Basics

- Reversing Android Applications – Level 1
- Reversing Android Applications – Level 2
- Proxying Android Traffic



SECURITY INNOVATION



Android Security Basics

Reversing Android Applications – Level 1

- Using APKTool
- Wrapper around smali and baksmali
- `apktool d appname.apk`

TASK

- Unzip the contents of *InsecureBankv2.apk* binary
- Is the *AndroidManifest.xml* file readable?
- Use *apktool* to decompile the *InsecureBankv2.apk* binary
- View the contents of the */res* folder
- View the contents of the *.smali* files and see if it makes any sense

Android Security Basics

Reversing Android Applications – Level 1

Examining Smali files

- Example Java code:

```
if (flag == 1)
    flag = 2
else
    flag = 3
```

- Equivalent Smali code:

```
const/4 v1, 0x1
if-ne v0, v1, :cond_0
const/4 v2, 0x2
move v0,v2
goto :goto_0
:cond_0
const/4 v2, 0x3
move v0,v2
:goto_0
```



Android Security Basics

Reversing Android Applications – Level 1

Examining Smali files

- v0, v1, v2, p0, p1, p2: Used for indicating registers
- Iput: Used to put values
- Const: Used for indicating constants
- Invoke-virtual or invoke-method: Used to indicate method call

Module 3: Android Security Basics

- Reversing Android Applications – Level 1
- Reversing Android Applications – Level 2
- Proxying Android Traffic



SECURITY INNOVATION



Android Security Basics

Reversing Android Applications – Level 2

- Apktool is not fun
- Smali is not easy to read
- Welcome – dex2jar and JADX

Android Security Basics

Reversing Android Applications – Level 2



LET'S
DO IT!

Android Security Basics

Reversing Android Applications

- What to Look for:
 - Hardcoded secrets
 - Hardcoded keys
 - Client-side dependencies
 - Hardcoded private information
 - Hardcoded developer information

Module 3: Android Security Basics

- Reversing Android Applications – Level 1
- Reversing Android Applications – Level 2
- Proxying Android Traffic

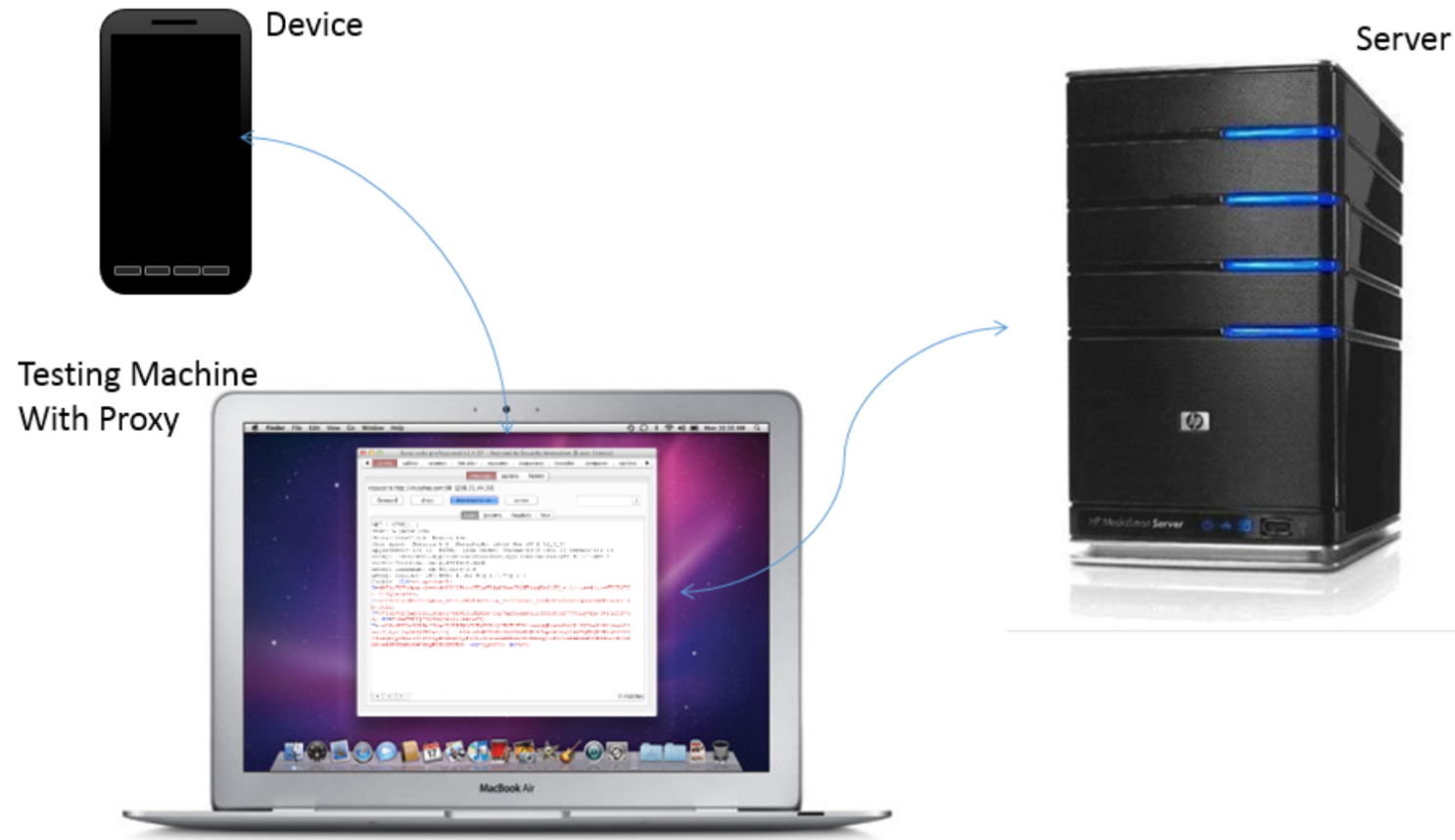


SECURITY INNOVATION



Android Security Basics

Proxying Android Traffic



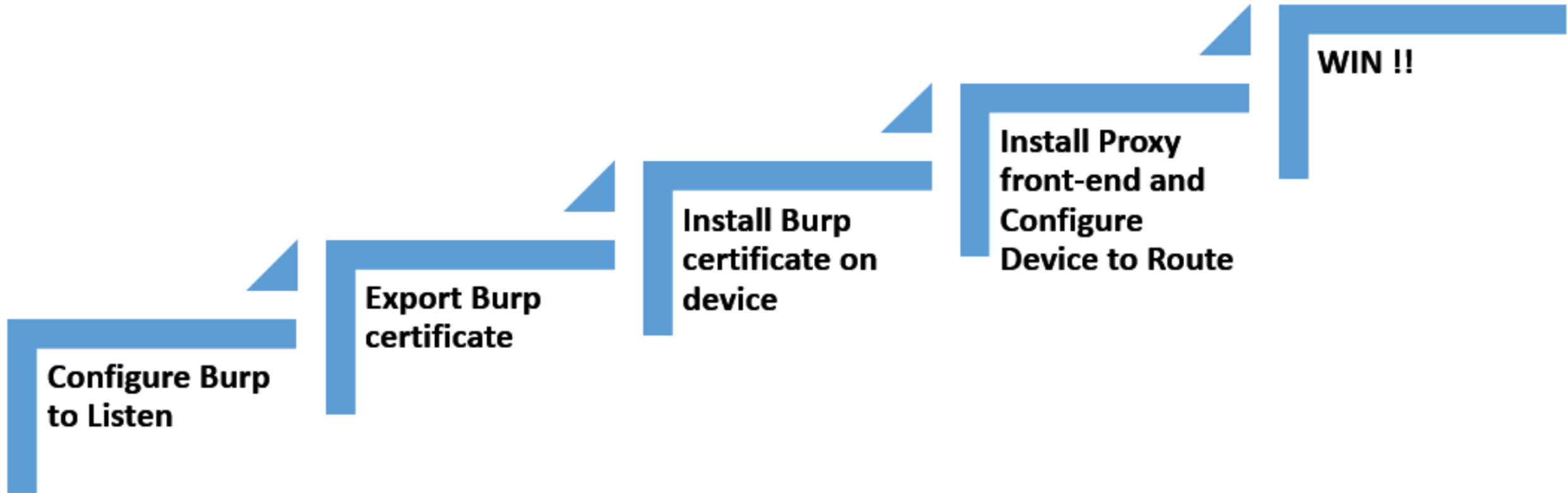
SECURITY INNOVATION



Android Security Basics

Proxying Android Traffic

Setting up the Test Environment



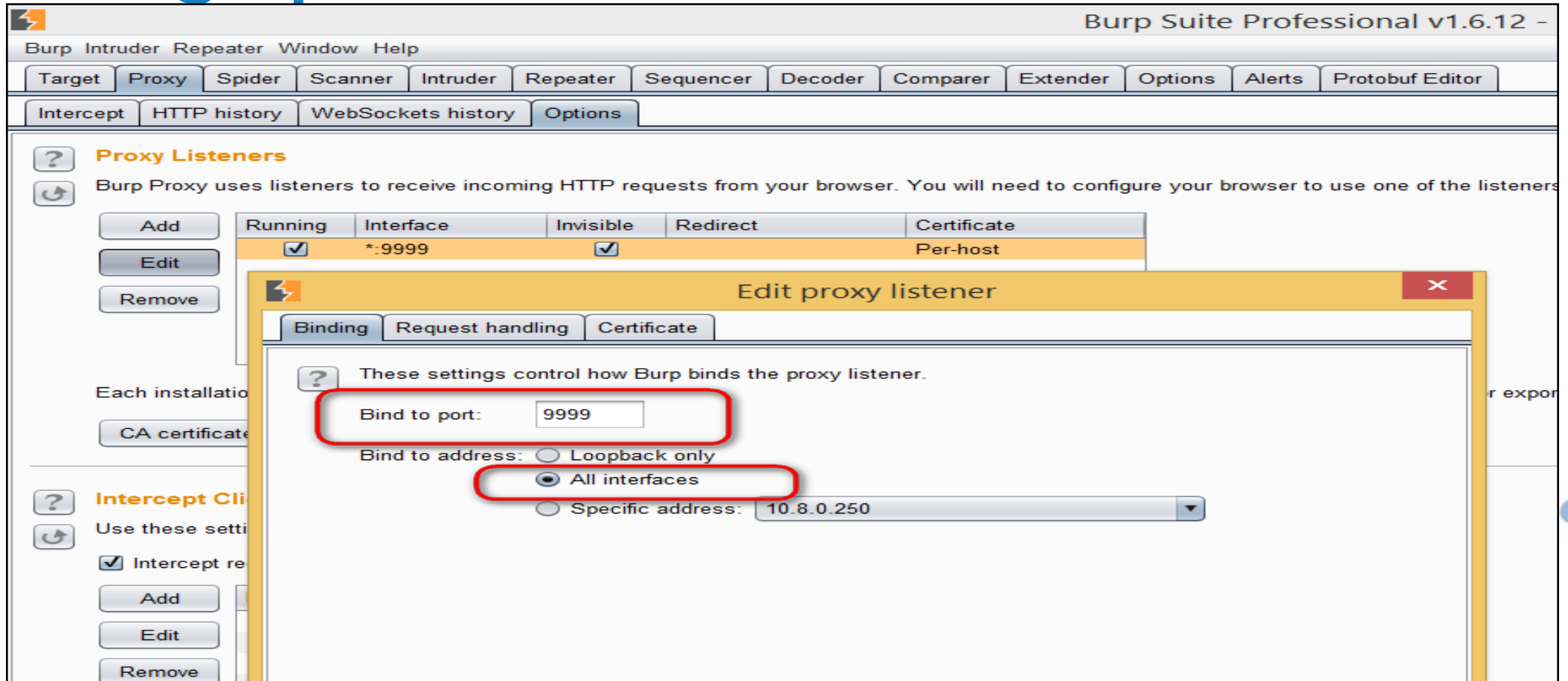
SECURITY INNOVATION



Android Security Basics

Proxying Android Traffic

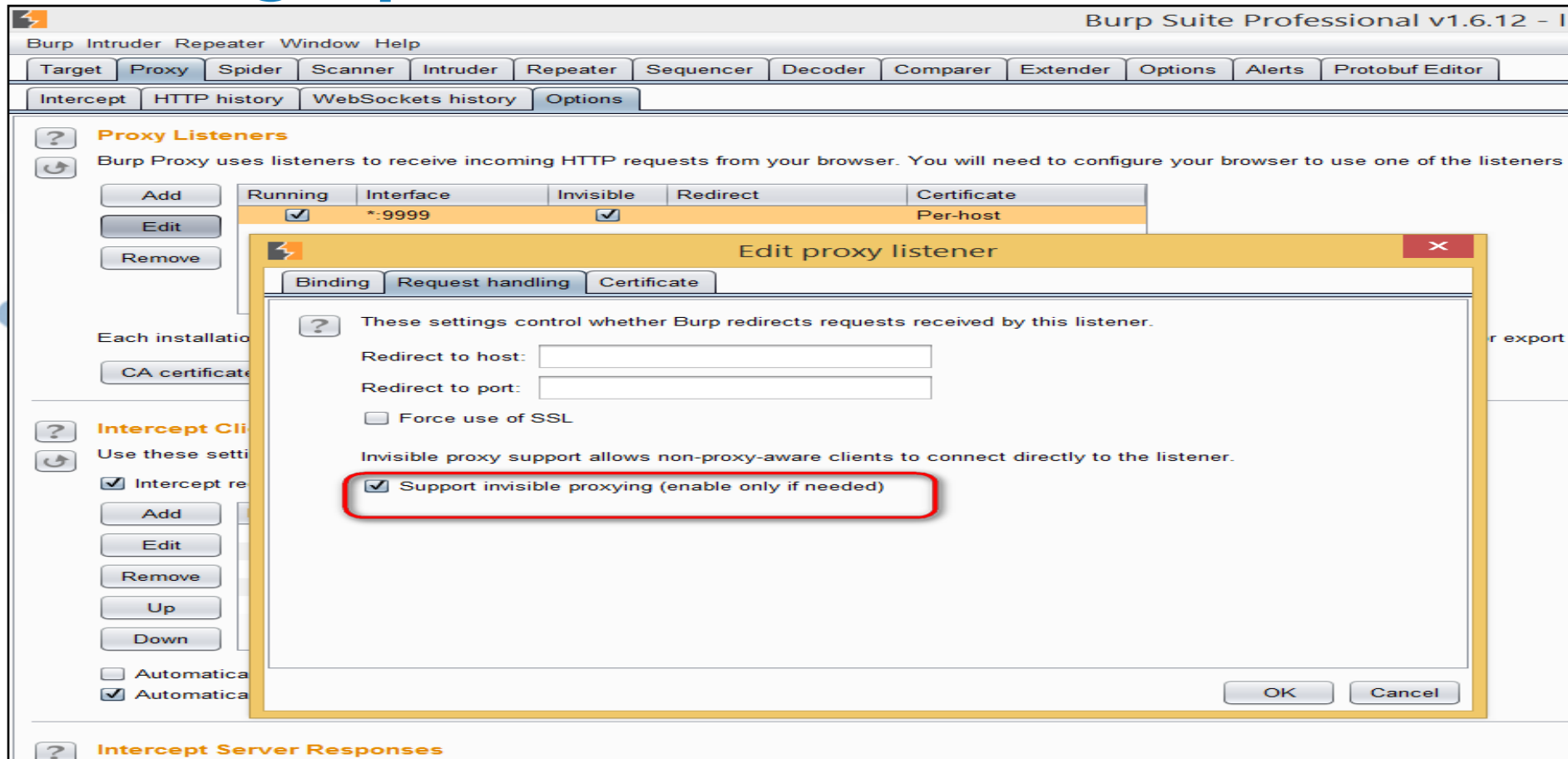
Setting up the Test Environment



Android Security Basics

Proxying Android Traffic

Setting up the Test Environment



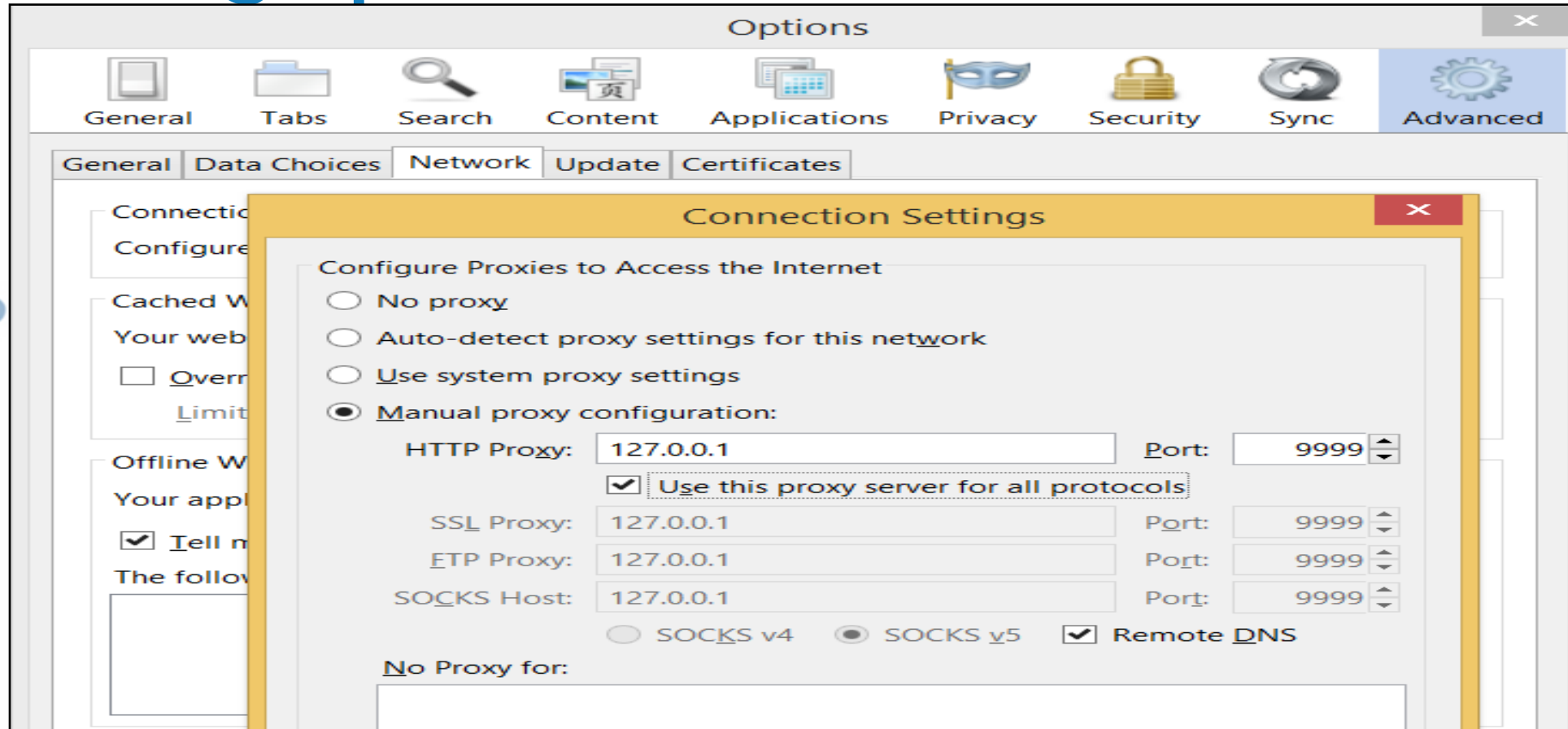
SECURITY INNOVATION



Android Security Basics

Proxying Android Traffic

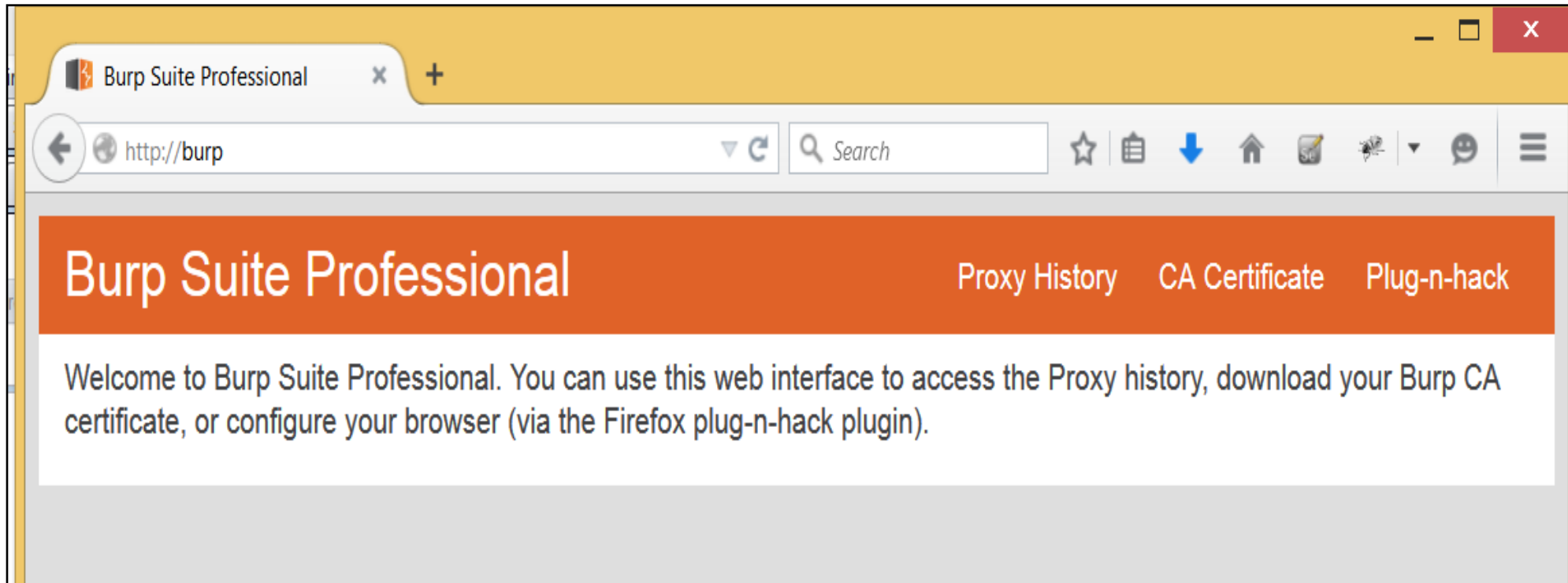
Setting up the Test Environment



Android Security Basics

Proxying Android Traffic

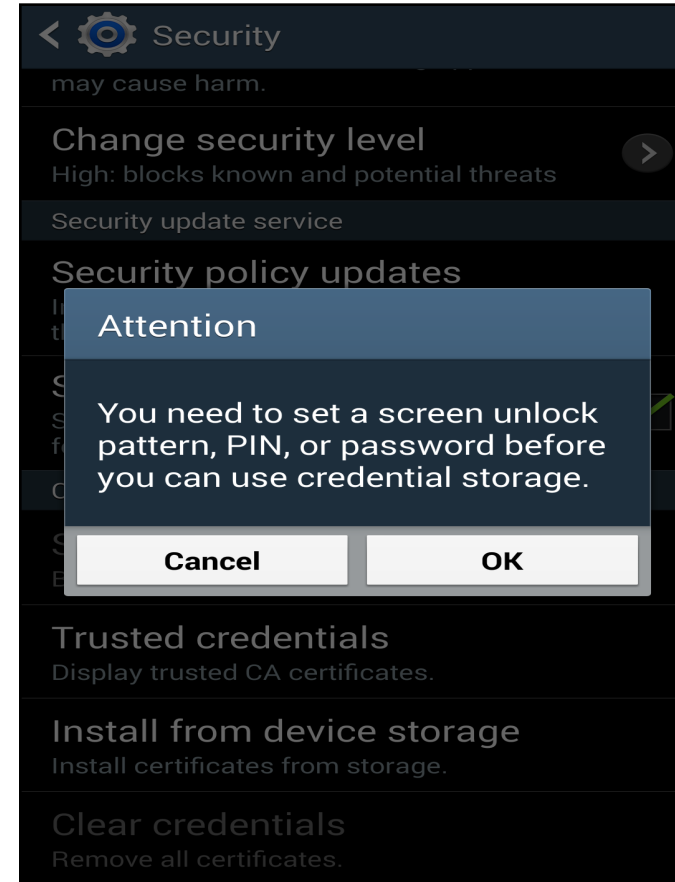
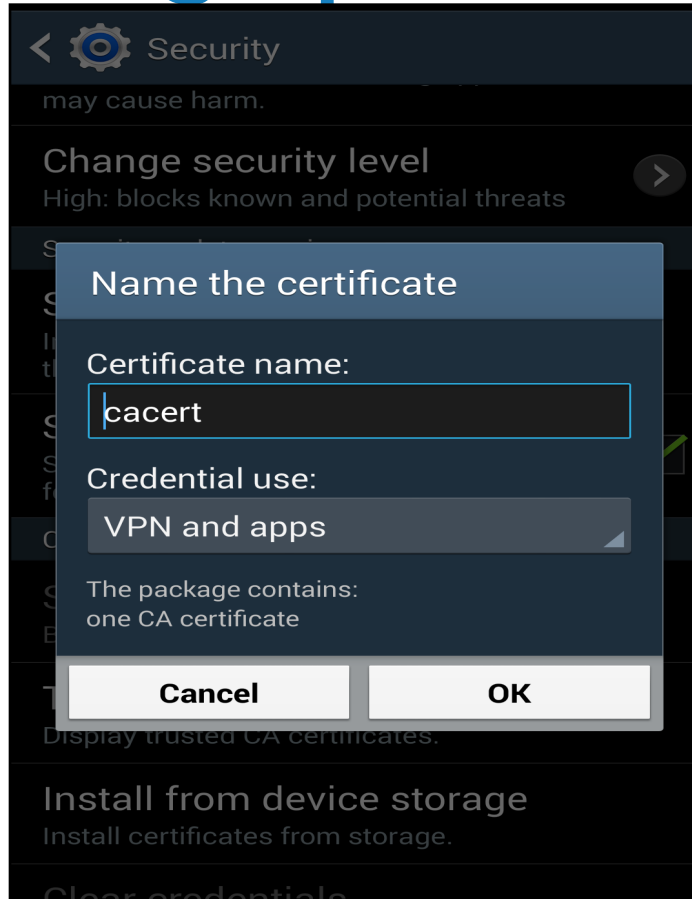
Setting up the Test Environment



Android Security Basics

Proxying Android Traffic

Setting up the Test Environment



SECURITY INNOVATION



Android Security Basics

Proxying Android Traffic

Setting up the Test Environment



ProxyDroid

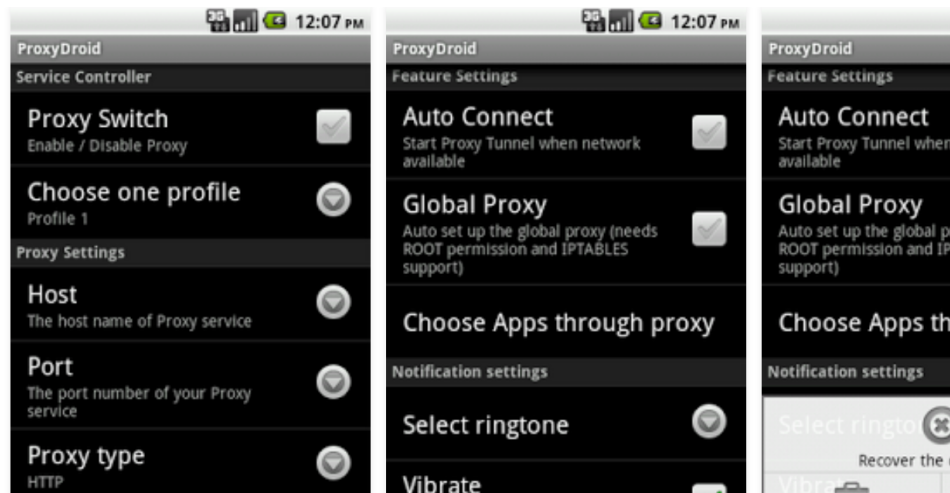
Max Lv Communication

★★★★★ 17,640

3+

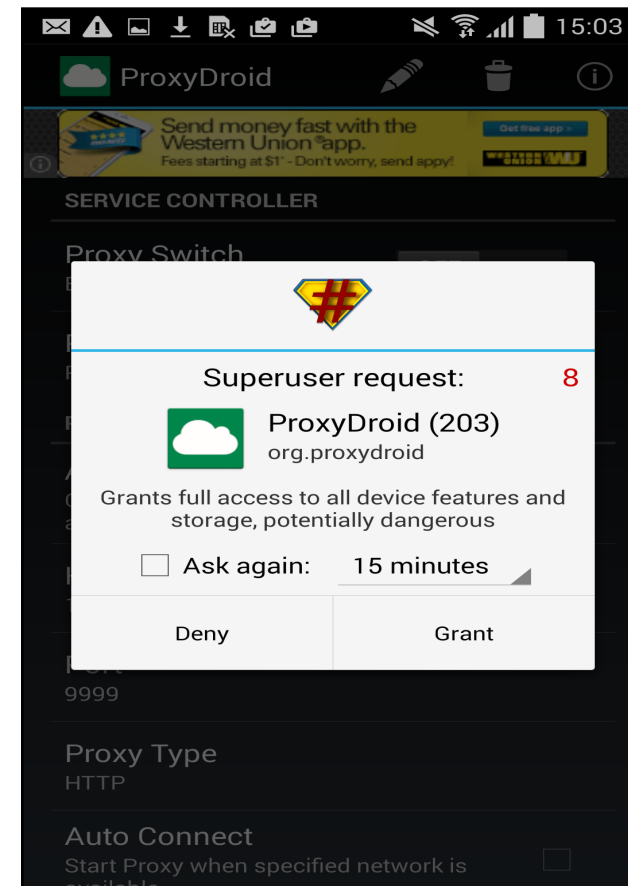
Add to Wishlist

Install



ProxyDroid is an app that can help you to set the proxy (http / socks4 / socks5) on your android devices.

Help to translate ProxyDroid: <http://crowdin.net/project/proxydroid/invite>



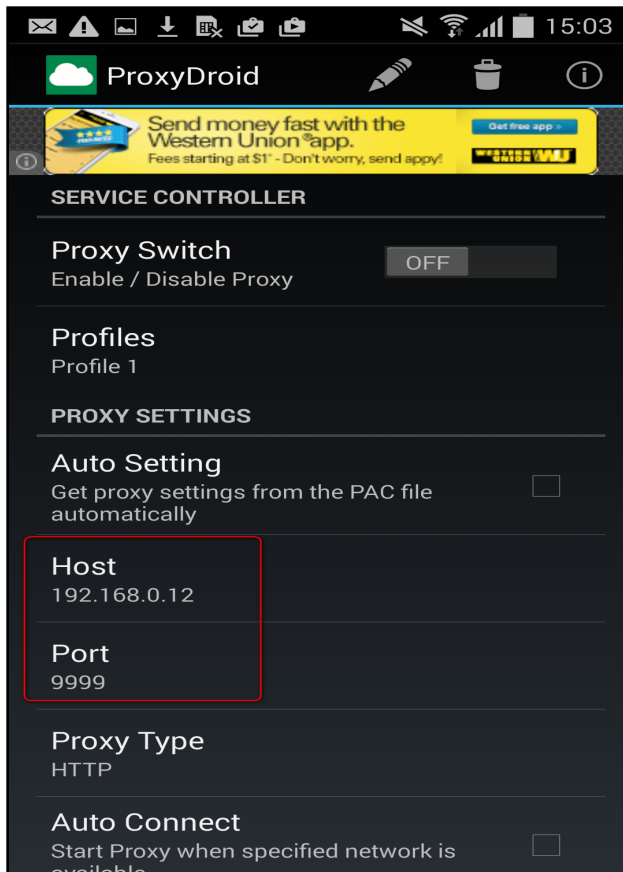
SECURITY INNOVATION

attify

Android Security Basics

Proxying Android Traffic

Setting up the Test Environment



SECURITY INNOVATION

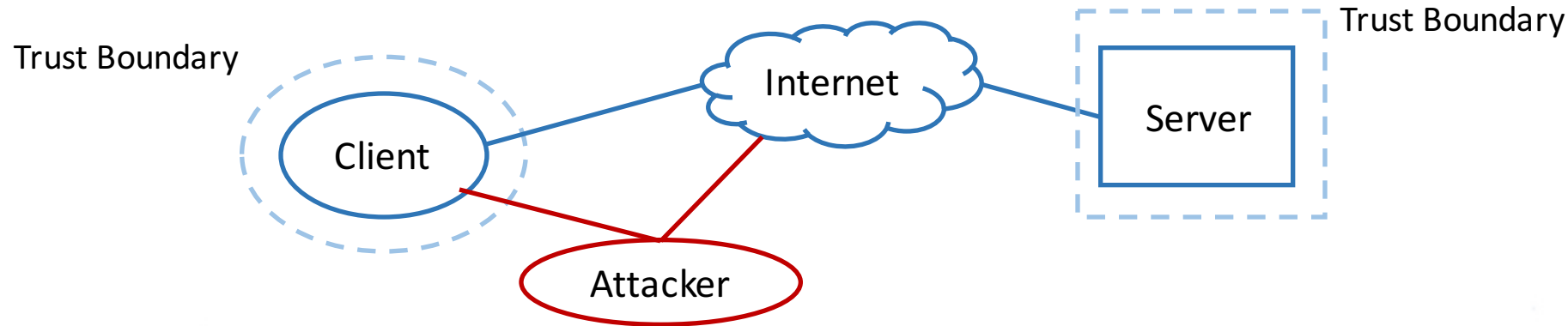


Android Security Basics

Proxying Android Traffic

Setting up the Test Environment

- Parameter tampering is an attack where the data in messages exchanged between the client and server are modified by the attacker



- The attacker can view and tamper with requests on the client, or at any location between the client and the Trust Boundary
- The tampering can occur in either direction
 - Even with a secure connection

A complex network diagram with numerous blue nodes of varying sizes connected by thin blue lines, forming a web-like structure that fills the background of the slide.

Workshop Agenda

- **Module 1: Mobile Security Introduction**
- **Module 2: Android Primer**
- **Module 3: Android Security Basics**
- **Module 4: Android Vulnerabilities**



Module 4: Android Vulnerabilities

- Exploiting Local Storage
- Exploiting Weak Cryptography
- Exploiting Android Components
- Exploiting Side Channel Data Leakage
- Patching Android Applications
- Root Detection and Bypass
- Exploiting Backup functionality

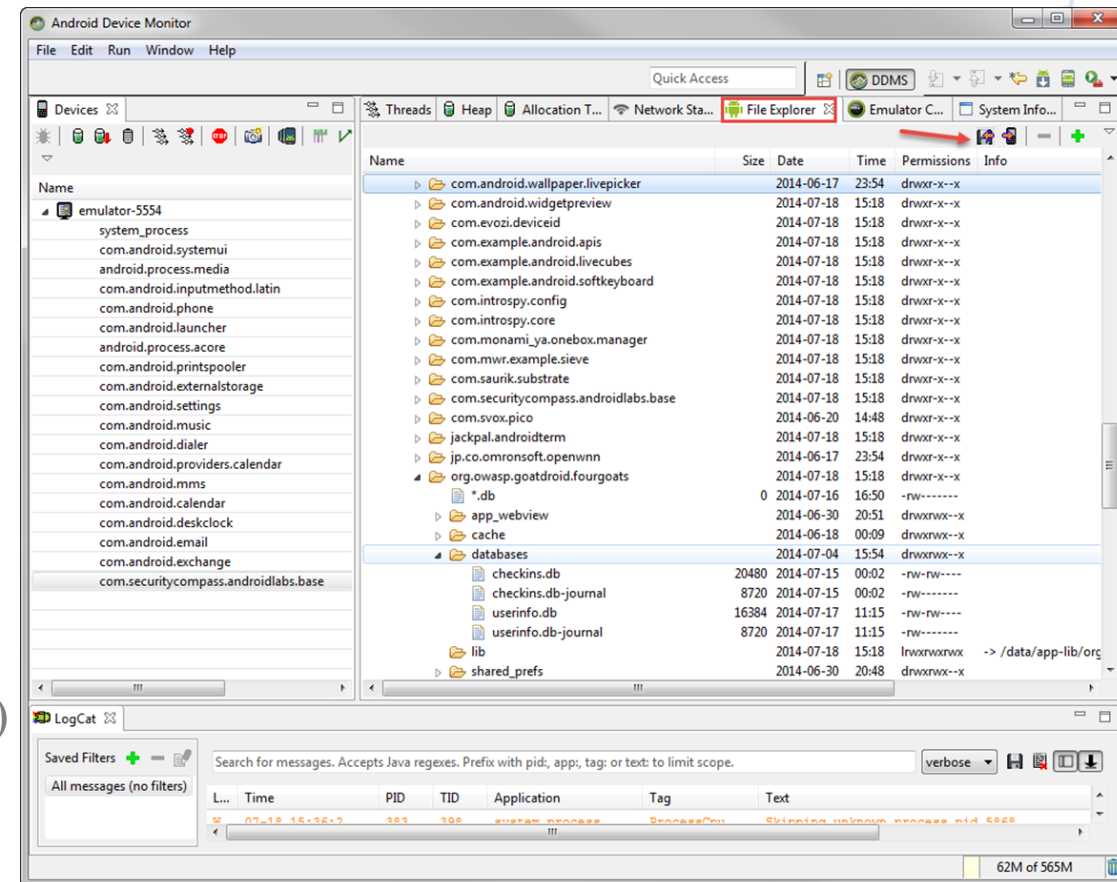


SECURITY INNOVATION



Exploiting Local Storage

- There are several ways to read local data:
- **On a rooted device**
 - **SSHDroid** + **WinSCP/ssh/sftp/PuTTY**
 - SSHDroid installs an SSH server on the device
- **On a non-rooted device**
 - **adb shell** or
 - Use the **Monitor** tool
 - Go to the tools directory of the SDK, and run **monitor** or **DDMS** (deprecated)
 - Click on the file **Explorer** tab to use a GUI to navigate
 - There are buttons to push and pull files



SECURITY INNOVATION



Exploiting Local Storage

Android Shared Preference Files

- Shared Preference files are similar to Android property files
 - Shared Preferences are used for storing key-value pairs of primitive data types
 - For example, Shared Preferences are used to store user's settings and application data
- Extension:
 - .xml
- Location:
 - `/data/data/<package-name>/shared_prefs/<prefs_name>.xml`
- Tools
 - Notepad



Exploiting Local Storage Android Shared Preference Files



SECURITY INNOVATION



Exploiting Local Storage Android SQLite Database

- Description
 - SQLite is an open-source database that does not need to have a server
 - Provides robust persistent data storage
 - Native SQL database within the complete control of application
 - Examine these databases to look for sensitive data, such as user settings, application data
- Extension:
 - `.sqlite`
- File Location:
 - `/data/data/<package-name>/databases/`
- Tools
 - Windows/Mac – SQLite Browser (<http://sourceforge.net/projects/sqlitebrowser/>)
 - Windows/Mac/Linux – SQLite Studio (<http://sqlitestudio.pl/>)



Exploiting Local Storage Android SQLite Database

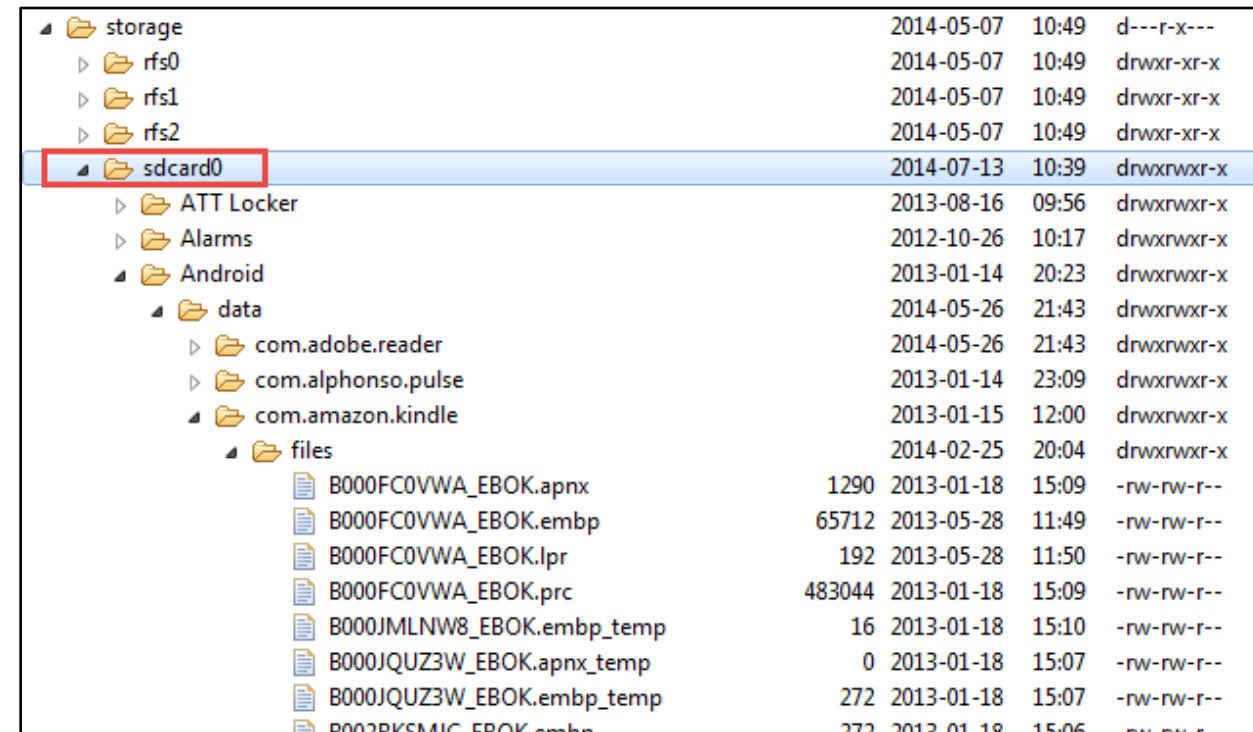


SECURITY INNOVATION



Exploiting Local Storage Android SD Card Data

- Description
 - Examine the files on the SD Card to look for sensitive data, such as user settings, application data
 - There are no file permissions on SD Card
 - Any application can read the contents of the SD Card
- Extension:
 - Multiple extensions
- File Location:
 - `/sdcard/`
- Tools
 - Use any file explorer to read SD card data



storage		2014-05-07	10:49	d---r-x---
rfs0		2014-05-07	10:49	drwxr-xr-x
rfs1		2014-05-07	10:49	drwxr-xr-x
rfs2		2014-05-07	10:49	drwxr-xr-x
sdcard0		2014-07-13	10:39	drwxrwxr-x
ATT Locker		2013-08-16	09:56	drwxrwxr-x
Alarms		2012-10-26	10:17	drwxrwxr-x
Android		2013-01-14	20:23	drwxrwxr-x
data		2014-05-26	21:43	drwxrwxr-x
com.adobe.reader		2014-05-26	21:43	drwxrwxr-x
com.alphonso.pulse		2013-01-14	23:09	drwxrwxr-x
com.amazon.kindle		2013-01-15	12:00	drwxrwxr-x
files		2014-02-25	20:04	drwxrwxr-x
B000FC0VWA_EBOK.apnx	1290	2013-01-18	15:09	-rw-rw-r--
B000FC0VWA_EBOK.embp	65712	2013-05-28	11:49	-rw-rw-r--
B000FC0VWA_EBOK.lpr	192	2013-05-28	11:50	-rw-rw-r--
B000FC0VWA_EBOK.prc	483044	2013-01-18	15:09	-rw-rw-r--
B000JMLNW8_EBOK.embp_temp	16	2013-01-18	15:10	-rw-rw-r--
B000JQUZ3W_EBOK.apnx_temp	0	2013-01-18	15:07	-rw-rw-r--
B000JQUZ3W_EBOK.embp_temp	272	2013-01-18	15:07	-rw-rw-r--
B002BKSMIC_EBOK.embp	272	2013-01-18	15:06	-rw-rw-r--



Exploiting Local Storage Android SD Card Data

- Sensitive data storage
- The Whatsapp database is stored on SDCARD
- Data is accessible by all the applications

```
localhost / # ls -lh /sdcard/WhatsApp/Databases/
---xrwxr-x   1 system   sdcard_r   48.8K May  3 18:56 msgstore-2012-05-03.1.db.crypt
---xrwxr-x   1 system   sdcard_r    8.0K May  3 18:57 msgstore-2012-05-04.1.db.crypt
---xrwxr-x   1 system   sdcard_r    8.0K May  4 01:01 msgstore.db.crypt
localhost / # ls -l /data/data/com.whatsapp/databases/
-rw-r--r--   1 app_58   app_58      8192 May  3 23:15 msgstore.db
-rw-rw----   1 app_58   app_58     21504 May  4 00:12 wa.db
localhost / # ls -l /data/data/com.whatsapp/files/Logs/
-rw-----   1 app_58   app_58     46283 May  1 00:00 whatsapp-2012-05-01.1.log.gz
-rw-----   1 app_58   app_58     60722 May  2 00:00 whatsapp-2012-05-02.1.log.gz
-rw-----   1 app_58   app_58     43027 May  3 00:00 whatsapp-2012-05-03.1.log.gz
-rw-----   1 app_58   app_58     60701 May  4 00:00 whatsapp-2012-05-04.1.log.gz
-rw-----   1 app_58   app_58      9512 May  4 01:03 whatsapp.log
localhost / #
```

Image Source:
<http://www.securitybydefault.com/2012/05/whatsapp-forensics.html>



SECURITY INNOVATION

attify

TASK

- Find the sensitive information stored in property files, database files and flat files by the given application.
 - You can use “find”.



Module 4: Android Vulnerabilities

- Exploiting Local Storage
- Exploiting Weak Cryptography
- Exploiting Android Components
- Exploiting Side Channel Data Leakage
- Patching Android Applications
- Root Detection and Bypass
- Exploiting Backup functionality



SECURITY INNOVATION


attify

Exploiting Weak Cryptography



SECURITY INNOVATION





Module 4: Android Vulnerabilities

- Exploiting Local Storage
- Exploiting Weak Cryptography
- **Exploiting Android Components**
- Exploiting Side Channel Data Leakage
- Patching Android Applications
- Root Detection and Bypass
- Exploiting Backup functionality



SECURITY INNOVATION



Exploiting Android Components

- Activity Manager
- Could be used to launch applications and pass data while launching
- Could also be used to launch specific activities within the application

Exploiting Android Components

- **Activities**
- Services
- Content Providers
- Broadcast Receivers

Exploiting Android Components Activities



SECURITY INNOVATION



Exploiting Android Components Activities With Intent Filter



SECURITY INNOVATION



Exploiting Android Components

- Activities
- **Services**
- Content Providers
- Broadcast Receivers

Exploiting Android Components Services



SECURITY INNOVATION



Exploiting Android Components

- Activities
- Services
- **Content Providers**
- Broadcast Receivers

Exploiting Android Components Content Providers



SECURITY INNOVATION



Exploiting Android Components

- Activities
- Services
- Content Providers
- **Broadcast Receivers**

Exploiting Android Components Broadcast Receivers



SECURITY INNOVATION

attify



Module 4: Android Vulnerabilities

- Exploiting Local Storage
- Exploiting Weak Cryptography
- Exploiting Android Components
- Exploiting Side Channel Data Leakage
- Patching Android Applications
- Root Detection and Bypass
- Exploiting Backup functionality



SECURITY INNOVATION

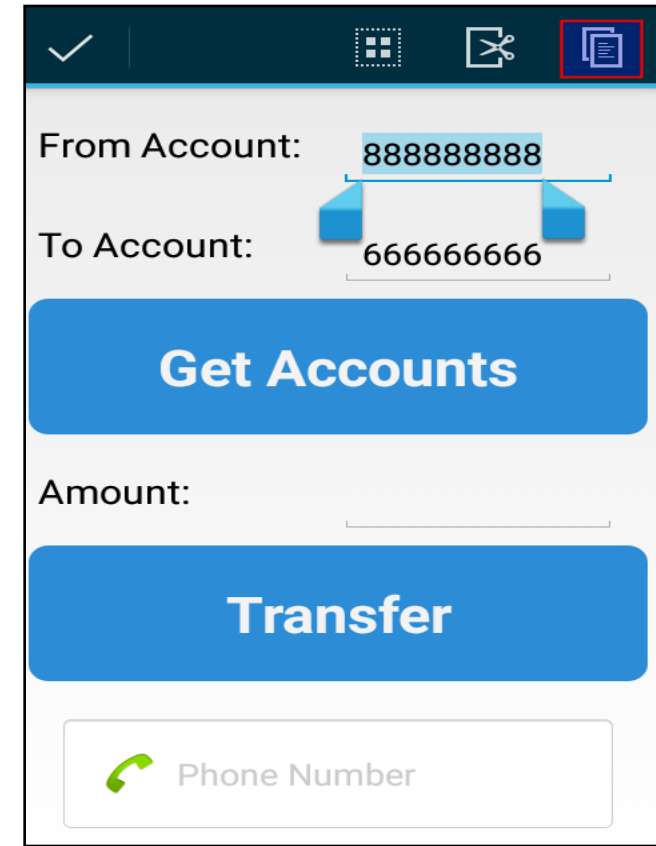


Exploiting Side Channel Data Leakage

- Vulnerabilities Include:
 - Copy Paste Buffer
 - Android Keyboard Cache
 - Android Log Files

Exploiting Side Channel Data Leakage Copy Paste Buffer

- Android has a common copy-paste buffer for all the applications
 - Data in this buffer is shared across different applications
- When copy/paste is allowed on sensitive fields like credit card numbers, SSN, usernames, etc., the data in the buffer is available to other applications
 - This could result in the leakage of sensitive information
- The copied text is stored locally



✓ [Grid Icon] [Share Icon] [Copy Icon]


From Account: 888888888

To Account: 666666666

Get Accounts

Amount:

Transfer

 Phone Number

Exploiting Side Channel Data Leakage Copy Paste Buffer

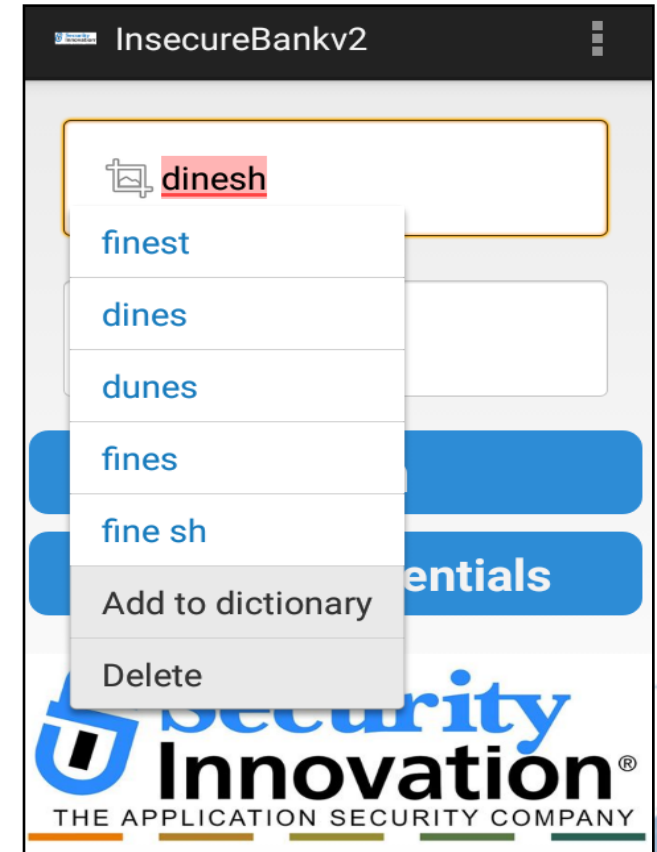


SECURITY INNOVATION



Exploiting Side Channel Data Leakage Android Keyboard Cache

- Keystrokes are logged to help with auto-correction and form completion
- New words are cached as part of the Android auto-correction feature
 - It is turned on by default for all textfields except password fields
- When auto correction is allowed on sensitive fields like credit card numbers, SSN, usernames, etc., this could allow sensitive information to be stored as keywords on the device
- The keywords are stored locally at `/data/data/com.android.providers.userdictionary/databases/user_dict.db`



Exploiting Side Channel Data Leakage Android Keyboard Cache



SECURITY INNOVATION

attify

Exploiting Side Channel Data Leakage Android Log Files

- Android maintains a centralized logcat of all the apps and device logs
- Android uses circular buffers for log messages
 - `adb logcat [-b buffer_name]` dumps the logs to `stdout`
 - `main` is the default buffer
 - `radio` contains radio/telephony related messages
 - `events` contains events-related messages
 - The `-f <filename>` parameter dumps the buffer to `<filename>`
- Can be accessed with `READ_LOGS` permission before 4.1
- Logcat permission has now changed to `signature|system|development`

Exploiting Side Channel Data Leakage Android Log Files





Module 4: Android Vulnerabilities

- Exploiting Local Storage
- Exploiting Weak Cryptography
- Exploiting Android Components
- Exploiting Side Channel Data Leakage
- **Patching Android Applications**
- Root Detection and Bypass
- Exploiting Backup functionality



SECURITY INNOVATION



Patching Android Applications

- APK files contain a file called `classes.dex` which is the app bytecode for the Dalvik VM
- The bytecode in `classes.dex` file can be disassembled into “Android assembler” (`.smali`)
 - It’s not like Java, but it is pretty high-level compared to “real” assembly code
 - Unless the developer ran code through an obfuscator like ProGuard or DexGuard, method and member names are preserved
 - This means it may be possible to understand and modify the code
 - Then you can recompile the code 😊
- Patching applications can be done to bypass authentication rules or root detection routines
- This is something that people do when...
 - Pirating games, etc.
 - Bypassing root detection or SSL Pinning checks
 - Removing authentication checks, etc.

Patching Android Applications



SECURITY INNOVATION



Patching Android Applications



SECURITY INNOVATION



Patching Android Applications

Signing in Android

- **keytool** -genkey -v -keystore <<keystore_name>> - alias <<your_key_alias>> -keyalg RSA -keysize 2048 - validity <<no_of_days>>
- **jarsigner** -verbose -sigalg MD5withRSA - digestalg SHA1 - keystore <<keystore_name>> <<.apk file>> <<your_key_alias>>
- **jarsigner** -verify -verbose <<path_to_apk>>



Patching Android Applications



SECURITY INNOVATION



TASK

- Decompile the cocon_apk.apk file using a combination of dex2jar and Apktool
- Make the necessary changes to enable the key in the cocon.smali file
- Recompile the application
- Sign the application
- Reinstall the application



Module 4: Android Vulnerabilities

- Exploiting Local Storage
- Exploiting Weak Cryptography
- Exploiting Android Components
- Exploiting Side Channel Data Leakage
- Patching Android Applications
- **Root Detection and Bypass**
- Exploiting Backup functionality



SECURITY INNOVATION

attify

Root Detection and Bypass

Detecting a Rooted Device

- Developers may want to detect if a device has been rooted for a variety of reasons
 - For example, they may not want the application to be run on a rooted device
- There are many ways to detect a rooted device
 - Default files and configurations that are found on rooted devices (not able to access on non-rooted devices due to permissions)
 - Release-keys tags, Over the Air (OTA) certificates
 - Installed files and packages present on rooted devices
 - `Superuser.apk`, `su`
 - Changed directory permissions
 - Rooted devices will have certain directories that are writable
 - `/data` is readable
 - If certain commands run, that can be an indication of a rooted device
- Not all methods work consistently
 - Every developer has their own “secret sauce” for detecting a rooted device

Root Detection and Bypass

Detecting a Rooted Device

```
private boolean doesSUexist() {  
    Process process = null;  
    try {  
        process = Runtime.getRuntime().exec(new String[] { "/system/xbin/which", "su" });  
        BufferedReader in = new BufferedReader(new InputStreamReader(process.getInputStream()));  
        if (in.readLine() != null) return true;  
        return false;  
    } catch (Throwable t) {  
        return false;  
    } finally {  
        if (process != null) process.destroy();  
    }  
}
```

Root Detection and Bypass

Detecting a Rooted Device

```
private boolean doesSuperuserApkExist(String s) {  
  
    File rootFile = new File("/system/app/Superuser.apk");  
    Boolean doesexist = rootFile.exists();  
    if(doesexist == true)  
    {  
        return(true);  
    }  
    else  
    {  
        return(false);  
    }  
}
```



Root Detection and Bypass

Detecting a Rooted Device

- Check for the BUILD tag
 - `cat /system/build.prop | grep ro.build.tags`
 - `ro.build.tags=release-keys`
- Check for Over The Air (OTA) certificates
 - `ls -l /etc/security/otacerts.zip`
 - `-rw-r--r--root root 1125 2016-05-16 15:29 otacerts.zip`



Root Detection and Bypass

Bypassing Root Detection

```
me$ adb shell
```

```
$ su
```

```
# cd /system/bin/; mount -o remount,rw -o rootfs rootfs /; mount -o  
remount,rw -o yaffs2 /dev/block/mtdblock3 /system
```

```
# echo $PATH /sbin:/system/sbin:/system/bin:/system/xbin
```

```
# mv /system/sbin/su /system/xbin/
```



Root Detection and Bypass

Bypassing Root Detection

```
me$ java -jar apktool.jar d app.apk source
```

```
[...]
```

```
me$ sed -i "" 's/system\ sbin \ su /system\ sbin \ CEW1PFSLK/g'  
source/ smali /net/example/ checks.smali
```

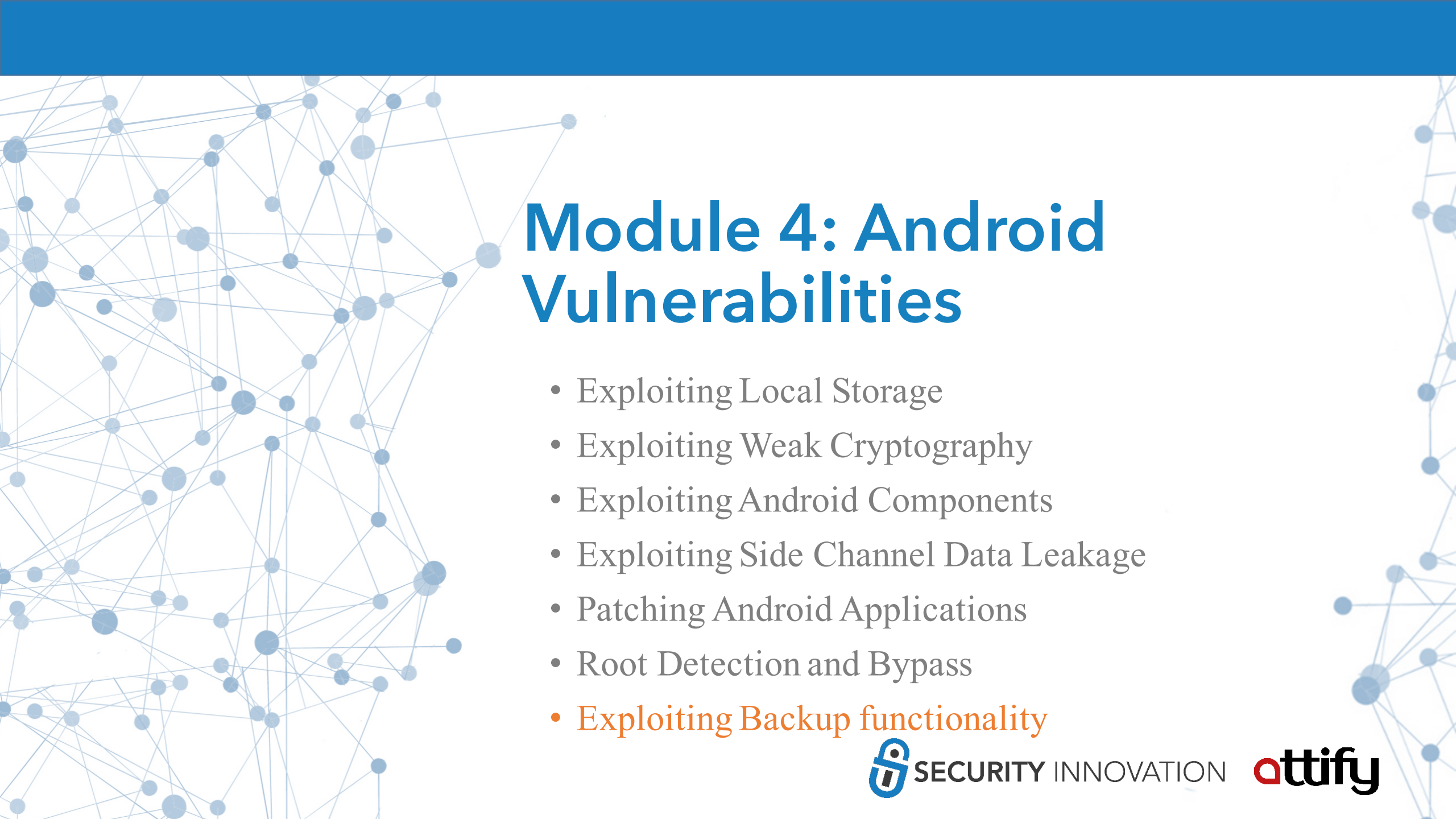
```
me$ java -jar apktool.jar b source/ fake.apk [...]
```

```
me$ keytool -genkey -alias someone -validity 100000 -keystore  
someone.keystore [...]
```

```
me$ jarsigner -keystore someone.keystore fake.apk someone
```

```
me$ adb install fake.apk
```

Source: http://www.floyd.ch/download/Android_0sec.pdf



Module 4: Android Vulnerabilities

- Exploiting Local Storage
- Exploiting Weak Cryptography
- Exploiting Android Components
- Exploiting Side Channel Data Leakage
- Patching Android Applications
- Root Detection and Bypass
- Exploiting Backup functionality



SECURITY INNOVATION


attify

Exploiting Backup functionality

- If Backup is enabled on the application, the sensitive data stored by the application is also backed up.
- Android allows backups and restoration of its data [without root]
- Attacker could take the backup of an app, modify the contents and restore it back again
- Lastpass Vulnerability (Patched now, found by Chris John Riley)


Exploiting Backup functionality

LastPass ****
The Last Password You'll Ever Need.

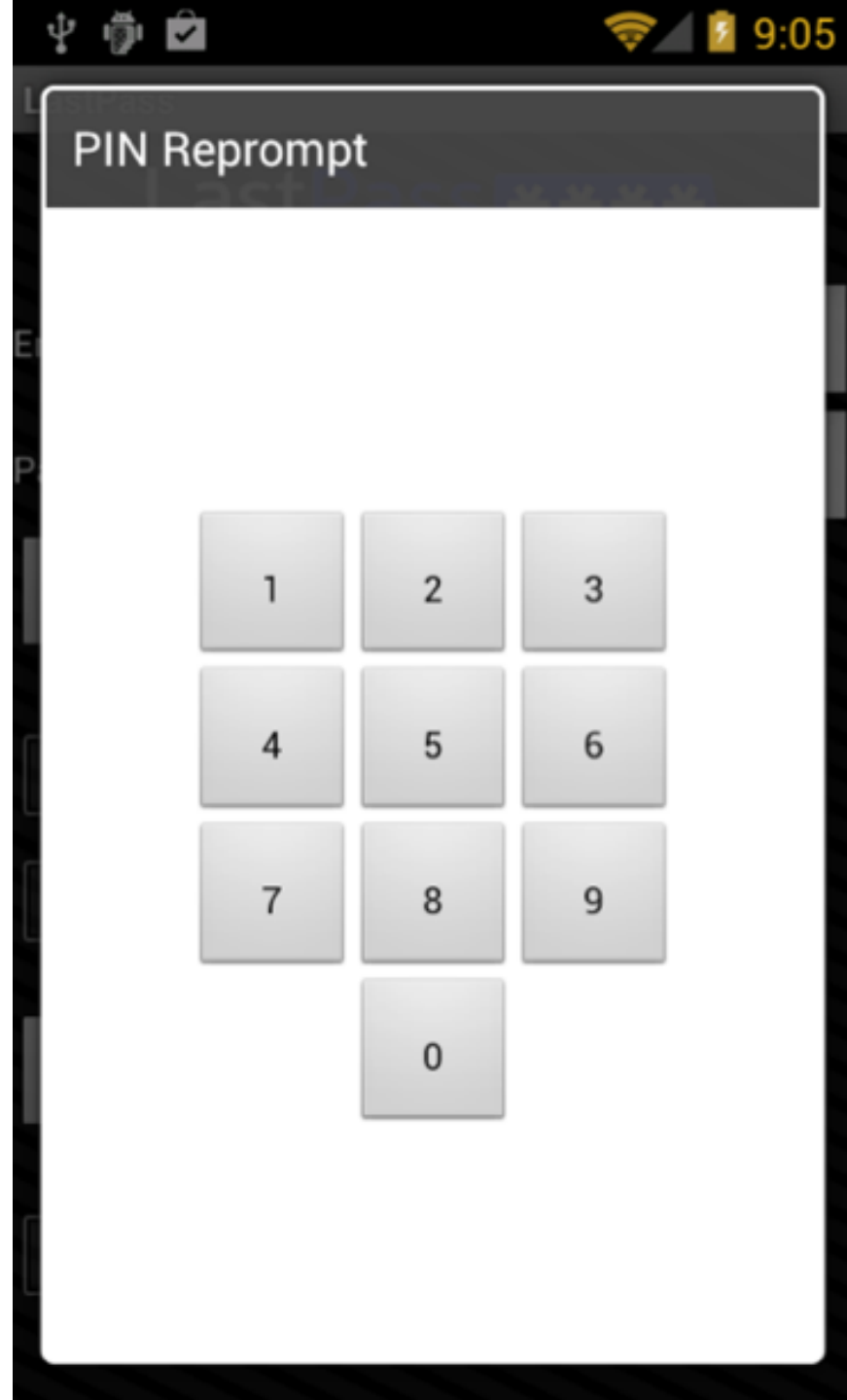


LastPass Password Mgr Premium*
LastPass - February 19, 2014
Productivity

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

 This app is compatible with all of your devices.

★★★★☆ (11,755) [g+1](#) +9639 Recommend this on Google



Exploiting Backup functionality



SECURITY INNOVATION

attify

Exploiting Backup functionality

```
mobisec# tree
```

```
.
|-- apps
    |-- com.lastpass.lpandroid
        |-- f
            |-- 632da2a7521caa7b4976a55d4353c5884ec366a8d525058bbba1d9498dfa94c8.xml
            |-- aaeb396e4c351e9c063952f0c169e409c6efa56a520eb785598b2fc5ded34350.xml
            |-- fde8b1731134c654b4c7aa6661a951a6ad9a448f7f99147b17033006a9b6c57d.xml
            |-- vcsfp.db3
            |-- vcsfp.db3-journal
        |-- _manifest
        |-- sp
            |-- LPandroid.xml
```


TASK

- Exploit the given application and check if the password is leaked when backup is enabled

Capture The Flag

OWASP Mobile Top 10 Risks

M1 – Weak Server
Side Controls

M2 – Insecure
Data Storage

M3 - Insufficient
Transport Layer
Protection

M4 - Unintended
Data Leakage

M5 - Poor
Authorization and
Authentication

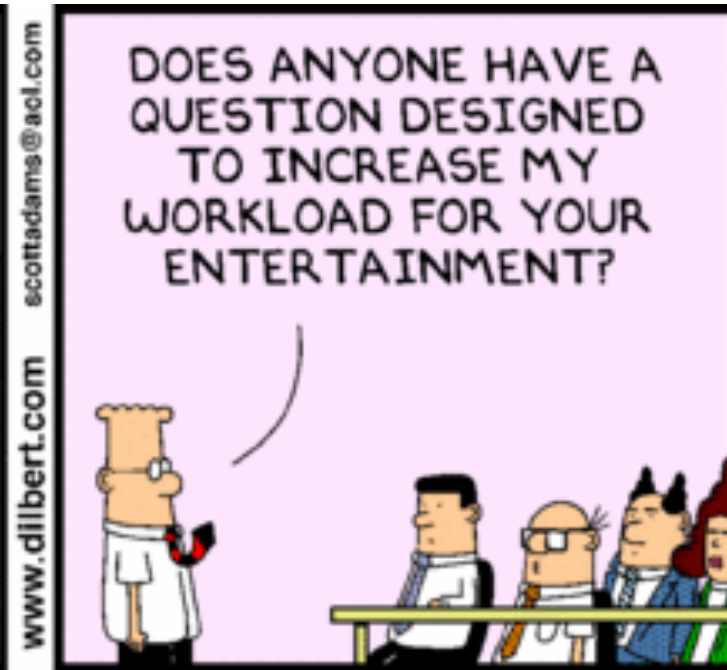
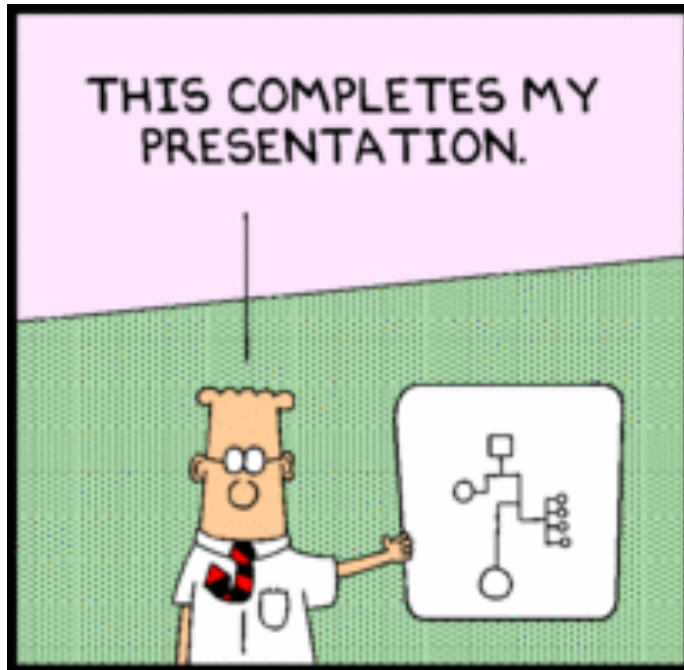
M6 - Broken
Cryptography

M7 - Client Side
Injection

M8 - Security
Decisions Via
Untrusted Inputs

M9 - Improper
Session Handling

M10 - Lack of
Binary Protections



K BYE !!!