Lookout®

Not the Droid You're Looking For

Evading Android Application Vulnerability Exploitation

OWASP Toronto, 09-17-18

$whoami

KRISTINA BALAAM

- Security Intelligence Engineer @ Lookout
- Formerly Application Security Engineer @ Shopify
- MSc. Student in Information Security Engineering, SANS Tech

@CHMODXX_ @CHMODXX blog.chmodxx.net

DISCLAIMER

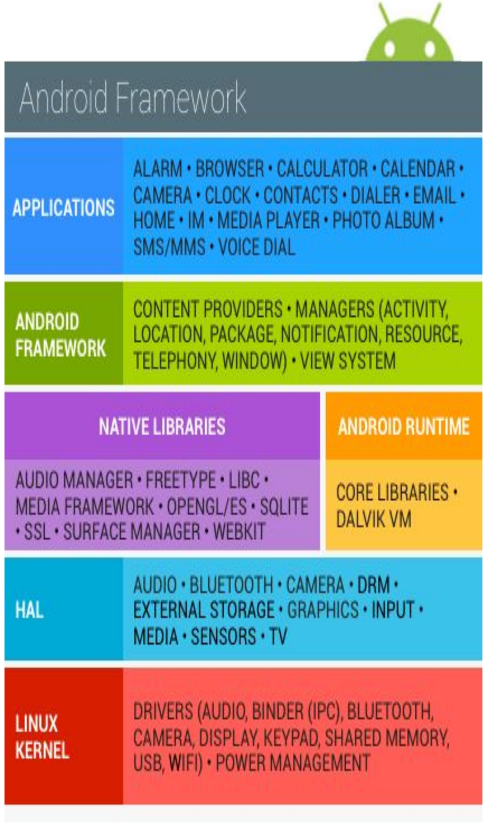
- ✓ The information provided is intended to educate mobile developers & application security engineers to better protect their applications.
- ✓ Use these techniques to build solid apps and audit those within your company.
- ✓ If you want to hack for $$ and fame, *please* join a bug bounty program like **HackerOne** or **BugCrowd**.
- ✓ ***Be responsible and disclose vulnerabilities*** □

NOT THE DROID YOU'RE LOOKING FOR

AGENDA

1. Android Basics
2. Reverse Engineering
3. Why So Vulnerable?
4. Common Attack Surfaces
5. Finding & Fixing Common Vulns
6. Continued Learning

1 > ANDROID BASICS


Android Framework
APPLICATIONS
ALARM • BROWSER • CALCULATOR • CALENDAR • CAMERA • CLOCK • CONTACTS • DIALER • EMAIL • HOME • IM • MEDIA PLAYER • PHOTO ALBUM • SMS/MMS • VOICE DIAL
ANDROID FRAMEWORK
CONTENT PROVIDERS • MANAGERS (ACTIVITY, LOCATION, PACKAGE, NOTIFICATION, RESOURCE, TELEPHONY, WINDOW) • VIEW SYSTEM
NATIVE LIBRARIES
AUDIO MANAGER • FREETYPE • LIBC • MEDIA FRAMEWORK • OPENGL/ES • SQLITE • SSL • SURFACE MANAGER • WEBKIT
ANDROID RUNTIME
CORE LIBRARIES • DALVIK VM
HAL
AUDIO • BLUETOOTH • CAMERA • DRM • EXTERNAL STORAGE • GRAPHICS • INPUT • MEDIA • SENSORS • TV
LINUX KERNEL
DRIVERS (AUDIO, BINDER (IPC), BLUETOOTH, CAMERA, DISPLAY, KEYPAD, SHARED MEMORY, USB, WIFI) • POWER MANAGEMENT

"Java on Linux" (Kind of)

Android applications & framework execute within (initially) DalvikVM and now ART
Provides an abstraction layer to the OS

Security boundaries

divide areas with certain levels of trust

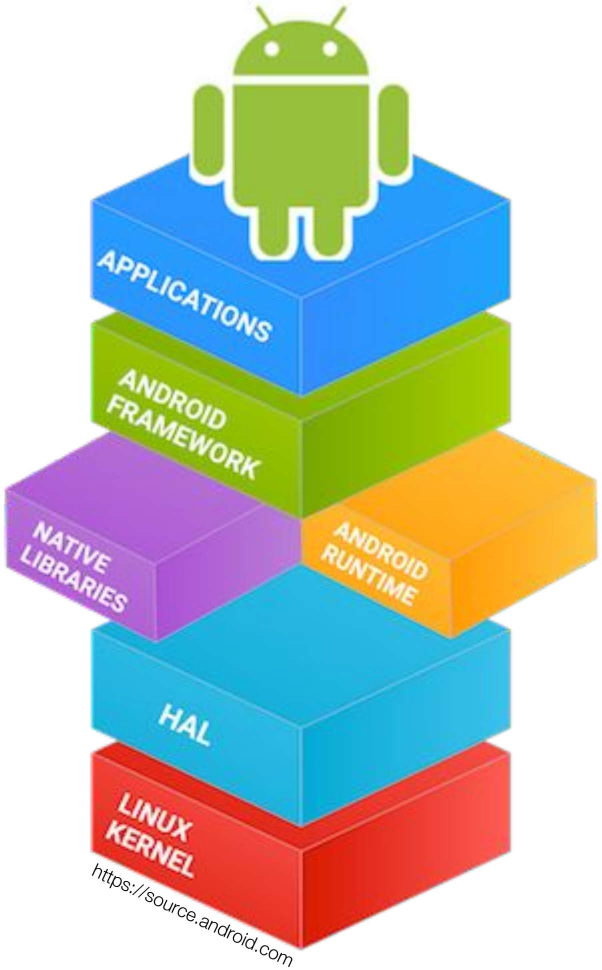
Two Permissions Models

Linux kernel

- Users & groups enforce permissions
- Aka. "Sandbox"
- Limits what can access resources

Android runtime

- Defines app permissions



APPLICATION COMPONENTS

ACTIVITY	★ Single, focused graphical window ★ Interacts with the user
INTENTS	★ Used for messaging between components ★ Implicit vs. Explicit
BROADCAST RECEIVERS	★ Inter-process communication (IPC) endpoint ★ Allows an application to register for certain events
SERVICE	★ Background operation / operation that doesn't require a user ★ Started VS Bound
CONTENT PROVIDERS	★ Manages the storage of application data ★ Used for sharing data between applications ★ Defined by developer; often SQLite
WEBVIEW	★ Act like a web browser ★ WebKit, 4.4+ Chromium
PERMISSIONS	★ The activities an application can perform are restricted to its permissions ★ Applications sandboxed by the OS so they can't access another apps data
MANIFEST	★ Defines application components ★ Only those defined in manifest are usable (except broadcast receivers) ★ Where you define permissions (!!!)

2 > REVERSE ENGINEERING

REVERSE ENGINEERING

The process of analyzing a subject system to identify the system's components and their interrelationships and to create representations of the system in another form or at a higher level of abstraction

IEEE

WHY DO WE REVERSE ENGINEER OUR APPS?

https://github.com/OWASP/owasp-mstg/blob/master/Document/0x04c-Tampering-and-Reverse-Engineering.md

- ✓ **To enable black-box testing of mobile apps.**
- ✓ **To enhance static analysis in black-box security testing.**
- ✓ **To assess resilience against reverse engineering.**

I ♥ LINUX

These examples are going to use Linux but most - if not all of these tools - will run on other OSes as well.

KALI LINUX

“Hacking” distro; super popular, well-supported and documented

SANTOKU LINUX

Mobile-specific security distribution; comes loaded with all the tools you need

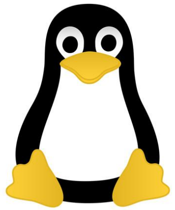
UBUNTU

User-friendly, great UI, well-maintained and documented

LINUX MINT

Another favourite amongst the Linux internet population; “Vanilla” Linux

SETTING UP THE ENVIRONMENT



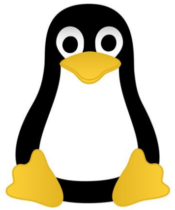
Android SDK

```
$ sudo apt-get install android-sdk
$ sudo ln -s /usr/share/android-sdk/platform-tools/adb /bin/adb
$ sudo chmod +x /usr/share/android-sdk/tools/android
```

★ **adb** interacts with devices, emulators to give a shell, read logs, etc.

★ **android** manages emulators

SETTING UP THE ENVIRONMENT



Creating an Android Emulator

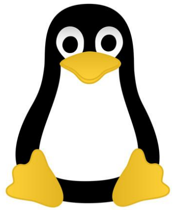
```
$ android sdk # install SDK platforms and tools
$ android avd # create an emulator
$ emulator -avd [emulator name]# run your emulator
```

Getting an Interactive Shell

```
$ adb devices # list all devices on your computer
$ adb -s DEVICE_ID shell # start an interactive shell for DEVICE_ID
```

★ Emulators get root by default (!!!) but don't always work properly for hardware tests

SETTING UP THE ENVIRONMENT



★ Download from your connected device via adb

```
$ adb pull [REMOTE] [LOCAL]
```

★ Third-party download sites like https://apkpure.com/ and https://apkbucket.net**

** Sketchy. Proceed with caution.

Drozer

- ✓ Released 2012 at Blackhat EU
- ✓ **Finds vulnerabilities / Provides exploits & payloads**
- ✓ **Agent** (runs on the device and facilitates testing)**, Console** (CLI to interact with the device)**, Server** (routes sessions between console & agents)

drozer

"Drozer allows you to assume the role of an Android app, and to interact with other apps, through Android's Inter-Process Communication (IPC) mechanism, and the underlying operating system." - MWR Labs

https://labs.mwrinfosecurity.com/tools/drozer/

```
$ git clone https://github.com/mwrlabs/drozer/
$ cd drozer
$ make deb
$ sudo dpkg -i drozer-2.x.x.deb
$ adb install drozer-agent-2.x.x.apk
$ adb forward tcp:31415 tcp:31415
$ drozer console connect
```

APKTool

- ✓ Converts resources back to pretty much their original form
- ✓ DEX (binary Dalvik bytecode) → **smali** (human readable)
- ✓ Allows you to decompile and recompile APKs

"A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications. It also makes working with an app easier because of the project like file structure and automation of some repetitive tasks like building apk, etc." - APKTool

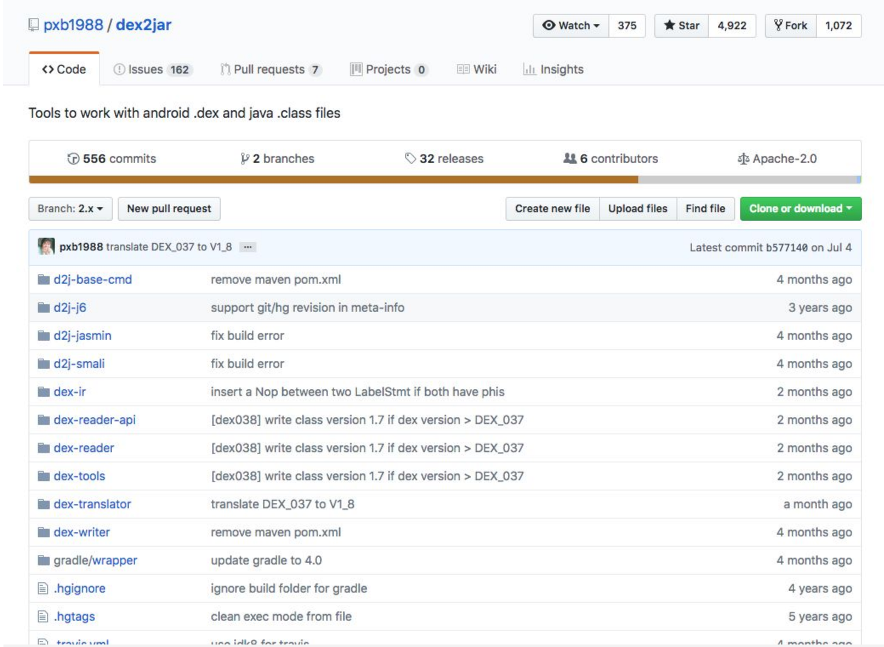
https://ibotpeaches.github.io/Apktool/

```
$ wget
https://bitbucket.org/iBotPeaches/apktool/downloads/apktool_2.3
.3.jar
$ wget
https://raw.githubusercontent.com/iBotPeaches/Apktool/master/sc
ripts/linux/apktool

$ mv apktool_2.3.3.jar apktool.jar
$ mv -t /usr/local/bin/ apktool.jar apktool
$ chmod +x apktool; chmod +x apktool.jar
$ apktool d [APPLICATION].apk# decompile the apk
```

dex2jar

- ✓ Converts .dex files to .class files and packages them into a jar.
- ✓ Necessary step for analysis in JD-GUI



pxb1988 / dex2jar

Watch 375 | Star 4,922 | Fork 1,072

Code | Issues 162 | Pull requests 7 | Projects 0 | Wiki | Insights

Tools to work with android .dex and java .class files

556 commits | 2 branches | 32 releases | 6 contributors | Apache-2.0

Branch: 2.x | New pull request | Create new file | Upload files | Find file | Clone or download

pxb1988 translate DEX_037 to V1_8 … Latest commit b577140 on Jul 4

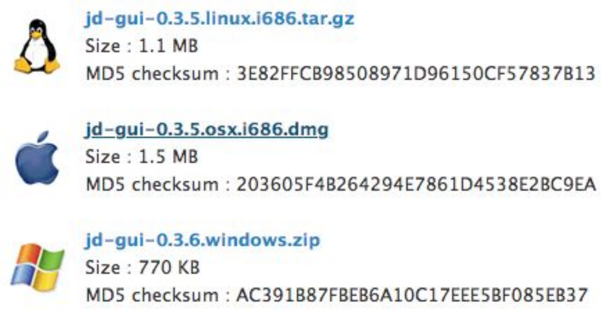
d2j-base-cmd	remove maven pom.xml	4 months ago
d2j-j6	support git/hg revision in meta-info	3 years ago
d2j-jasmin	fix build error	4 months ago
d2j-smali	fix build error	4 months ago
dex-ir	insert a Nop between two LabelStmt if both have phis	2 months ago
dex-reader-api	[dex038] write class version 1.7 if dex version > DEX_037	2 months ago
dex-reader	[dex038] write class version 1.7 if dex version > DEX_037	2 months ago
dex-tools	[dex038] write class version 1.7 if dex version > DEX_037	2 months ago
dex-translator	translate DEX_037 to V1_8	a month ago
dex-writer	remove maven pom.xml	4 months ago
gradle/wrapper	update gradle to 4.0	4 months ago
.hgignore	ignore build folder for gradle	4 years ago
.hgtags	clean exec mode from file	5 years ago
.travis.yml	use jdk8 for travis	4 months ago

https://github.com/pxb1988/dex2jar

```
$ git clone https://github.com/pxb1988/dex2jar.git
```

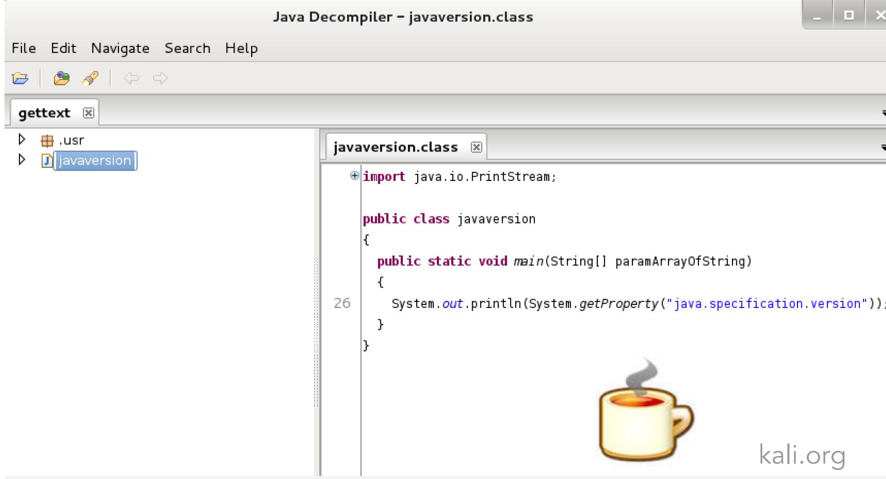
JDGUI

✓ Displays a somewhat accurate representation of the .class files converted by Dex2Jar


jd-gui-0.3.5.linux.i686.tar.gz
Size : 1.1 MB
MD5 checksum : 3E82FFCB98508971D96150CF57837B13

jd-gui-0.3.5.osx.i686.dmg
Size : 1.5 MB
MD5 checksum : 203605F4B264294E7861D4538E2BC9EA

jd-gui-0.3.6.windows.zip
Size : 770 KB
MD5 checksum : AC391B87FBEB6A10C17EEE5BF085EB37


Java Decompiler – javaversion.class
File Edit Navigate Search Help
gettext
.usr
javaversion
javaversion.class
kali.org

```
import java.io.PrintStream;

public class javaversion
{
  public static void main(String[] paramArrayOfString)
  {
    System.out.println(System.getProperty("java.specification.version"));
  }
}
```

JD is a Decompiler for the Java programming language. JD is provided as a GUI tool as well as in the form of plug-ins for the Eclipse and IntelliJ IDEA integrated development environments.

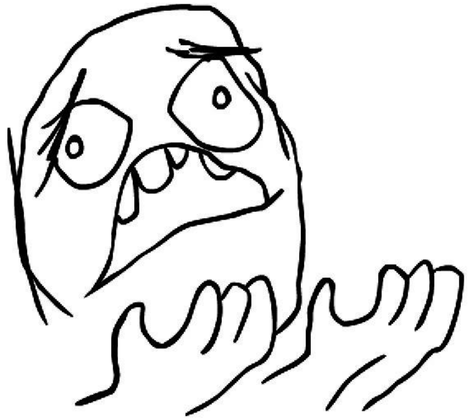
https://github.com/java-decompiler/jd-gui

```
$ wget
https://github.com/java-decompiler/jd-gui/releases/download/v1.4.0/jd-gui_1.4.0-0_all.d
eb
$ sudo dpkg -i jd-gui_x.x.x-x_all.deb
```

3 > WHY SO VULNERABLE?

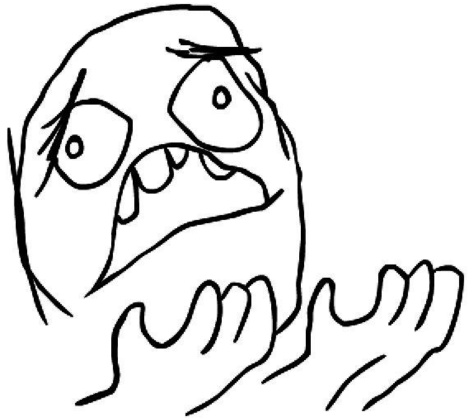
HACKING IN PROGRESS
GIFSec.com

Fragmentation



Fragmentation

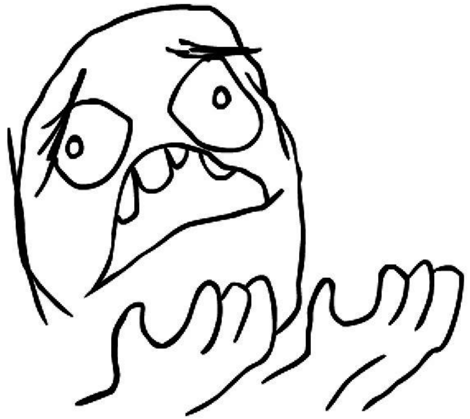
Update Frequency

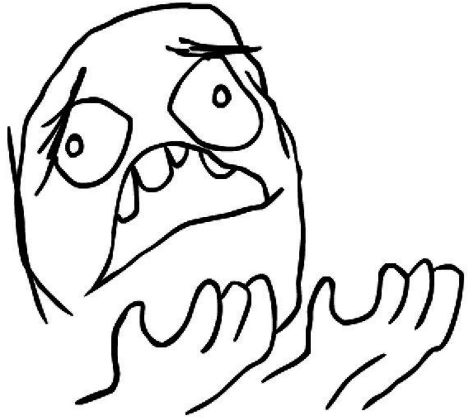


Fragmentation

Update Frequency

Back-porting

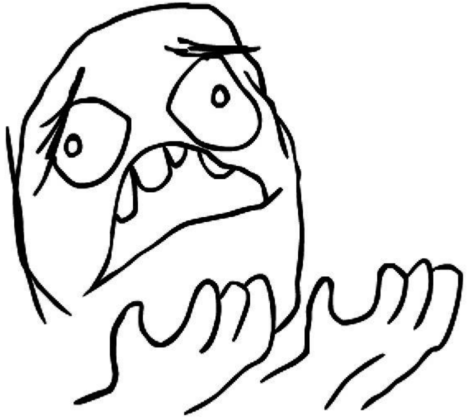




Fragmentation

Update Frequency

Back-porting (or lack thereof)

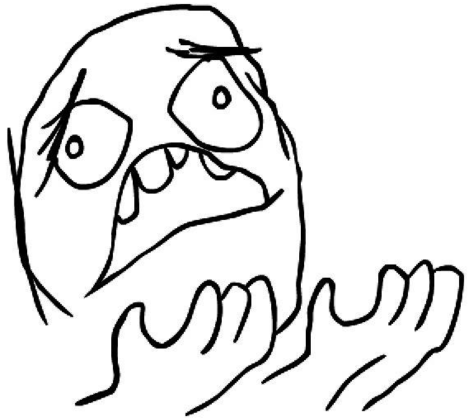


Fragmentation

Update Frequency

Back-porting (or lack thereof)

Updating Dependencies



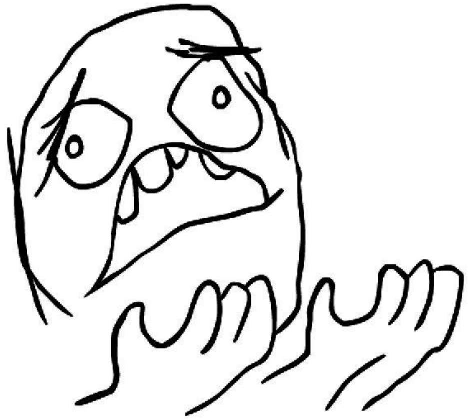
Fragmentation

Update Frequency

Back-porting (or lack thereof) 😆

Updating Dependencies

Open-Source != Secure



Fragmentation

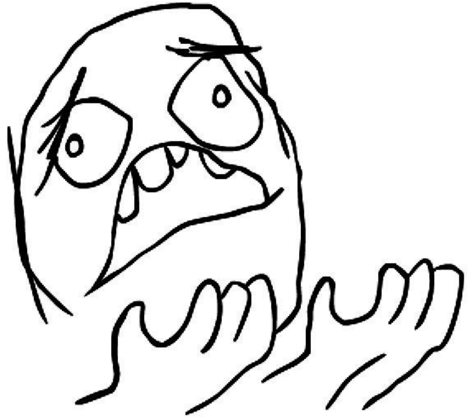
Update Frequency

Back-porting (or lack thereof)

Updating Dependencies

Open-Source != Secure

Public Disclosures



Fragmentation

Update Frequency

Back-porting (or lack thereof) 😈

Updating Dependencies

Open-Source != Secure

Public Disclosures (or lack thereof) 😈

4 > COMMON ATTACK SURFACES

PERMISSIONS!

PERMISSIONS!

BROADCAST RECEIVERS!

PERMISSIONS!

BROADCAST RECEIVERS!

IPCS!

PERMISSIONS!

BROADCAST RECEIVERS!

IPCS!

STORAGE!

PERMISSIONS!

BROADCAST RECEIVERS!

IPCS!

STORAGE!

INTENTS!

PERMISSIONS!

WEBVIEWS!

BROADCAST RECEIVERS!

IPCs!

STORAGE!

INTENTS!

PERMISSIONS!

WEBVIEWS!

BROADCAST RECEIVERS!

IPCS

STORAGE!

INTENTS

CONTENT PROVIDERS!

PERMISSIONS!

WEBVIEWS!

STORAGE!

BROADCAST RECEIVERS!

IPCS

LOGGING!

INTENTS

CONTENT PROVIDERS!

PERMISSIONS!

WEBVIEWS!

BROADCAST RECEIVERS!

LOGGING!

IPCS!

STORAGE!

INTENTS!

CONTENT PROVIDERS!

DATA VALIDATION!

PERMISSIONS!

WEBVIEWS!

BROADCAST RECEIVERS!

LOGGING!

STORAGE!

IPCS!

INTENTS!

CONTENT PROVIDERS!

DATA VALIDATION!

OBFUSCATION!

PERMISSIONS!

WEBVIEWS!

TAMPERING!

STORAGE!

LOGGING!

BROADCAST RECEIVERS!

[illegible]

INTENTS!

CONTENT PROVIDERS!

DATA VALIDATION!

OBFUSCATION!

PERMISSIONS!

TAMPERING!

STORAGE!

WEBVIEWS!

BROADCAST RECEIVERS!

LOGGING!

INTENTS!

CONTENT PROVIDERS!

DATA VALIDATION!

OBFUSCATION!

5 > FINDING & FIXING COMMON VULNS

oh no,

PERMISSIONS

APPLICATION PERMISSION

- Developers often ask for more permissions than they need.
- Most users will accept anything they're asked.
- Err on the side of caution and do whatever you can to make the attack surface smaller.

```
dz> run app.package.info -a [PACKAGE-NAME]
```

<u>What to Do:</u>

- Follow the Principle of Least Privilege
- Define permissions with signature protection so no other applications can access components or request permissions
- Make sure the permissions you're requesting are really necessary -- let native apps handle functionality
  - Eg. Only need READ permissions? Don't grant READWRITE.

PERMISSIONS

FILE PERMISSIONS

- Only for files stored externally.
- You can define file permissions in AndroidManifest *and* in the code.
- Malware loves searching for files in SD Cards

<u>What to Do:</u>

- Don’t give MODE_WORLD_READABLE or MODE_WROLD_WRITABLE permissions if you can help it
    - they allow other applications to access the file
- Share files between the Content Provider; avoid external storage where you can

IPCS

(INTERPROCESS COMMUNICATIONS)

- Services, Activities, BroadcastReceivers, ContentProviders
  - Endpoints aren’t always secured
  - They act as data sinks *and* sources.
- **Broadcast messages allow any application to receive an intent!**
  - Malicious apps could gain access to another’s data
- **ContentProviders:** could expose access to data and directory traversal or SQL injection attacks; when not permissions protected, any application can invoke

- **Activities:** could be used in a UI-redressing attack
- **BroadCast Receivers**: could be hijacked to intercept an Intent & its data; null values can also be sent to DoS applications
- **Services:** Could expose application-specific functionality

IPCS

(INTERPROCESS COMMUNICATIONS)

<u>What to Do:</u>

- Share files using **ContentProvider**; avoid external storage (like SDCards) where you can
- Android versions before 4.2 export content providers by default. Ensure this is false for any apps whose targeted SDK version is <= 16
- Even content providers that *aren't* exported can be accessed by privileged users
- Similarly, exported activities require no permissions for interaction
- When using ContentProviders, always ensure a permission is set for the required application
- Sanitize inputs or use prepared statements with ContentProviders to avoid SQL injection attacks
- Use explicit intents wherever possible
- Use custom permissions with services, too (can be checked by service when external service makes a request)
- Use the local broadcast manager for local intents
  - No other application can access the data
- sendBroadcast(intent); and sendStickyBroadcast(intent); are susceptible to IPC sniffing. Use intents signed with permissions so an unauthorized app can't receive the intent!
- Check the data being received from any broadcast and ensure that it's valid!

IPCS

USEFUL DROZER COMMANDS

```
dz> run app.provider.info -a [PACKAGE NAME]
# list all exported content providers
```

```
dz> run app.provider.info -a [PACKAGE NAME] -u
# list all non-exported content providers
```

```
dz> run app.activity.info -a [PACKAGE NAME]
# list all exported activities
```

```
dz> run scanner.provider.injection -a
# scan for sql injection vulns
```

REAL WORLD EXAMPLE
Samsung Kies, Galaxy S3

- Kies was highly privileged: connects mobile phone to your PC
- Had a BroadCast receiver that restored APKs from the SDcard
- Tldr; Kies has a call chain that iterates through the sdcard/restore directory and installs every APK
- A researcher was able to add their app to the SD card by exploiting a WRITE_EXTERNAL_STORAGE privilege issue with the clipboard service on the S3, and then had Kies call that function with an intent

http://sh4ka.fr/android

INSECURE STORAGE

- **Apps are super easy to RE**
  - .apks are basically just .zip files
- **Data should be stored in either:**
  - `/data/data/<package>`
    - Only accessible by the application unless it gives permission or if the device is rooted
  - `/sdcard`
    - Accessible by *everyone*

- Process information can be dumped to access sensitive info.
- <u>Don't embed any encryption key in source code.</u>
- If an attacker has access to a phone, and the memory isn't cleared after the app is closed, they could access anything stored.
- WebViews allow HTML data to cache locally.

INSECURE STORAGE

What to Do:

- Look for code that stores data locally: make sure it's not storing sensitive data
- When you absolutely *have* to store something client-side, make sure it's encrypted if it's sensitive
- When you're encrypting, use a *strong encryption algorithm*: avoid MD5/SHA1 hashing for passwords and instead use PBKDF2, bcrypt or scrypt.
- If you're using webviews, look at clearCache() or "no-cache" to prevent caching data altogether
- Re-initialize the Application class with dummy values once it closes to prevent saved information since it remains active even when the app is closed

REAL WORLD EXAMPLE
Skype, 2011

- Skype was storing contacts, profile, IM logs plus additional personal data at `/data/data/com.skype…/files/<USERNAME>` in SQLite database and XML files

- Permissions were improperly set -- world readable and writable -- so any app could read them. Everything was unencrypted

- Skype also stored the usernames in static locations, so a malicious application could parse the `shared.xml` file, find the usernames and then use that value to find the files stored in data.

Reported by Justin Case (jcase), http://AndroidPolice.com

REAL WORLD EXAMPLE
What's App, 2014

- Conversations stored on the phone's SD Card
- Researcher Bas Bosschert built a fake "malicious" app with a distracting load screen that requested the necessary permissions and downloaded user's WhatsApp data while the loading screen ran
- Database files were "encrypted", but easily decrypted with a Python script and a key found on an XDA Developers forum post

Reported by Bas Bosschert, TechCrunch

INSECURE COMMUNICATIONS

- **Web traffic inspection is an important part of the audit process**
  - (A surprising number of developers don't realize you can intercept web traffic -- especially on mobile)

- **Burp Suite** is a great (free) tool for setting up an intercepting proxy for mobile testing.

- **You can set up a proxy on an emulator.**
  - Set the Access Point Name to 10.0.2.2 and the port to the same as what's been specified by your Burp listener port.

- **New in Android P!**
  - TLS by default, but ability to opt out for legacy domains
    - https://developer.android.com/training/articles/security-config

Fix: Never send plaintext requests

WEBVIEWS

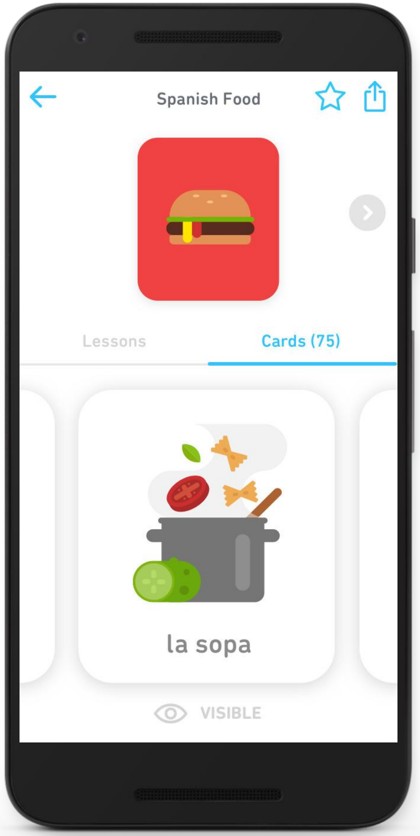
- **Renders web pages within the application**
- **WebView lets you break out of the app sandbox and bypass same origin.**
- **Also makes it possible to load malicious .js: any web page accessed by the frame in the app can call back to the application.** And can call back *Java*.
- Loading cleartext content is the single biggest mistake you can make

- **You see this a lot in apps with advertisements.**
- **How often does this happen?**
  - Stanford study in 2013:
    - 40k apps minimum using a "javascript bridge"
    - ⅓ could be reached by untrusted content

WEBVIEWS

What to Do:

- Restrict users to the application domain
- Don't call setJavaScriptEnabled() until you absolutely have to process Javascript
- APIs 17+ require methods to be marked `@JavascriptInterface` for any method exposed to Javascript and this prevents malicious code from accessing the lower-level OS commands
- Create a whitelist of domains that are allowed to render content
- Send all traffic over SSL to prevent man-in-the-middle attacks where someone tries to inject script



REAL WORLD EXAMPLE
TinyCards, 2018

RCE in TinyCards for Duolingo on Android

“TinyCards loads a website via webview when starting, but that site is loaded over http then redirected to https. A MITM attack that controls either the network or the DNS, can inject their own web content into the webview. You can confirm this by using an MITM proxy to capture the traffic.” - @nightwatch-cybersecurity, HackerOne

Publicly Disclosed on HackerOne:
https://hackerone.com/reports/281605

LOGGING

- **Logging is great for debugging.***
    - *It's also great for hackers.
- **Even system processes (eg. ActivityManager) log detailed messages.**
- Even though the READ_LOGS permission was removed for 3P Applications after 4.1, rooted devices can still access it.

```
$ logcat
# running from shell shows sys and app logs
$ adb logcat
# same as above, just direct
```

REAL WORLD EXAMPLE
Firefox, 2012

- Logged browsing activity, including plain text URLs and even session IDs
- Malicious application or attacker could use session IDs to hijack a victim's session

Reported by Neil Bergman

OBFUSCATION

It's easy to RE Android apps.

You can make it harder by obfuscating your code.

- **Pros**: Harder for people to steal your stuff or exploit
- **Cons**: Ongoing maintenance can be tricky.

What to Do:

- ProGuard obfuscates your code *lexically*: meaningful names replaced by machine-generated garble
- Using native code makes decompilation harder: attacker has to resort to assembly level reversing
  - BUT be aware: more susceptible to issues like buffer overflows
- Java reflection: code that's able to inspect other code -- makes it harder to trace what's happening in your app

PRIVATE KEYS

- **Used for:**
  - Signing apps
  - Encrypting https traffic

- **Private keys are included in apps ALL. THE. TIME.**

- **Java keystore**
  - Container for public/private keys and certificates
  - Password protection is optional (!!!!)
  - No container-level encryption
  - Private keys housed within share same password as keystore container by default

What to Do:

- **DON'T STORE YOUR PRIVATE KEYS IN YOUR APP**

- **Cloud KMS:** Google offers cloud storage for secrets (https://cloud.google.com/kms/)

- If you don't trust Google, keep it somewhere else. Safe. Separate from the app.

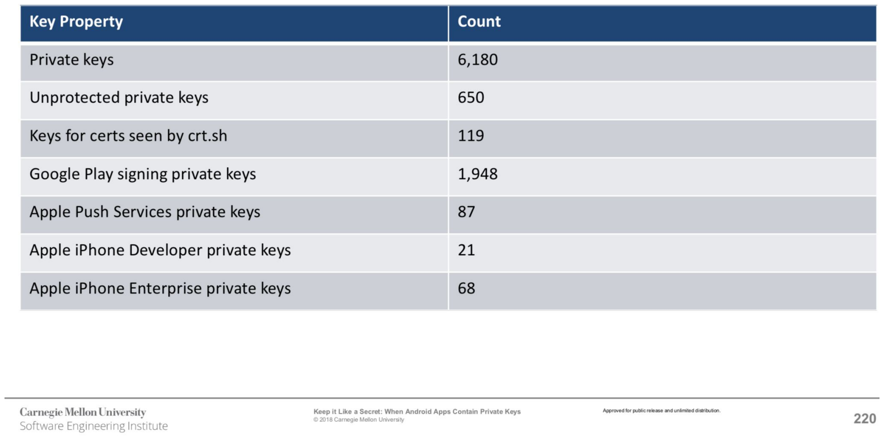
- Google I/O 2018 announced "**StrongBox**": resistant to shared resource attacks, side channel attacks, physical attacks
  - Only some new devices that ship with Android P

PRIVATE KEYS

June 2017, an IT security journalist finds a private key in a CISCO app.

Will Dormann of Carnegie Mellon does a study analyzing apps for exposed private keys.

- Looked at 1.7M APKs



Key Property	Count
Private keys	6,180
Unprotected private keys	650
Keys for certs seen by crt.sh	119
Google Play signing private keys	1,948
Apple Push Services private keys	87
Apple iPhone Developer private keys	21
Apple iPhone Enterprise private keys	68

Carnegie Mellon University
Software Engineering Institute

Keep it Like a Secret: When Android Apps Contain Private Keys
© 2018 Carnegie Mellon University

Approved for public release and unlimited distribution.

220

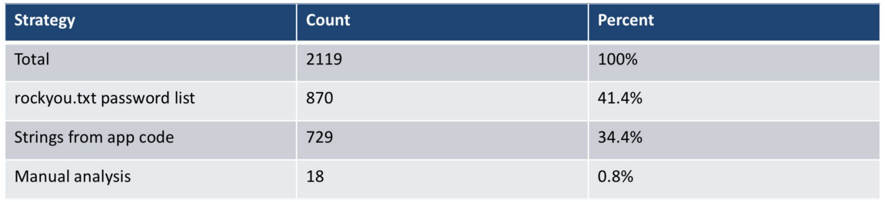
PRIVATE KEYS

June 2017, an IT security journalist finds a private key in a CISCO app.

Will Dormann of Carnegie Mellon does a study analyzing apps for exposed private keys.

- Looked at 1.7M APKs

PKCS#12 Password Cracking Statistics



Strategy	Count	Percent
Total	2119	100%
rockyou.txt password list	870	41.4%
Strings from app code	729	34.4%
Manual analysis	18	0.8%

PRIVATE KEYS

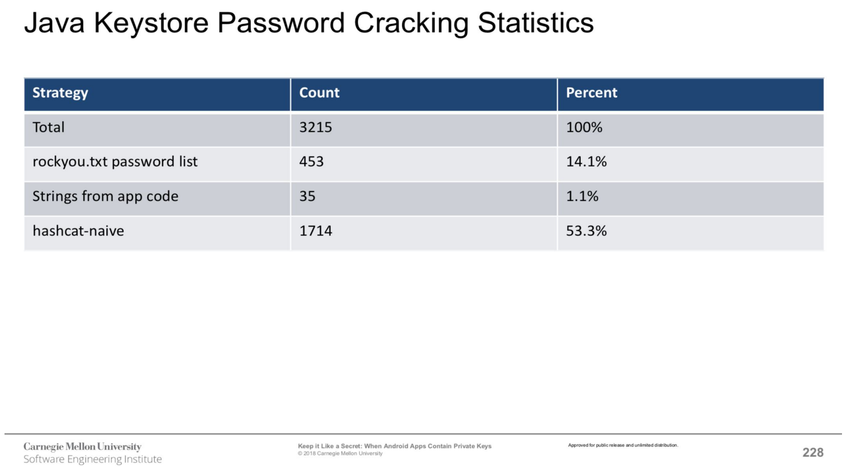
June 2017, an IT security journalist finds a private key in a CISCO app.

Will Dormann of Carnegie Mellon does a study analyzing apps for exposed private keys.

- Looked at 1.7M APKs

Watch the BSidesSF Talk

https://www.youtube.com/watch?v=-VjK0FMmGm4



Java Keystore Password Cracking Statistics

Strategy	Count	Percent
Total	3215	100%
rockyou.txt password list	453	14.1%
Strings from app code	35	1.1%
hashcat-naive	1714	53.3%

Carnegie Mellon University
Software Engineering Institute

Keep it Like a Secret: When Android Apps Contain Private Keys
© 2018 Carnegie Mellon University

Approved for public release and unlimited distribution.

228

TAMPER DETECTION

- **Attackers can download an APK, modify it, re-sign it**
  - The certificate hash would change, so it'd be obvious it wasn't the same developer (UNLESS YOU INCLUDE YOUR PRIVATE KEY)
  - But the attacker could still re-upload the app as a clone and fool people into downloading it

- **You can add signature checks to your code…but you'd have to be sneaky.**
  - Determined attackers could just figure out where you're checking for the signature and remove it.

What to Do:

- Avoid client-side checks

- Google's SafetyNet has some nifty tamper-detection features
  - Can detect whether a device is rooted (with some level of certainty)
  - Can determine whether a device has malware (to an extent)

- Android P's "Keystore Attestation API": signed statement from secure hardware that the device hasn't been tampered with

- You can run system calls to check whether your application is being accessed by the Android Debug Bridge or whether the app is running on an emulator

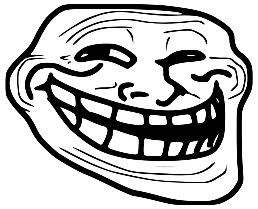
BE PARANOID

BE PARANOID

ALL THE TIME

BE PARANOID

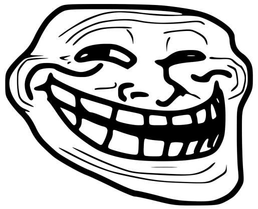
ALL THE TIME



JK.

BE PARANOID

ALL THE TIME



JK. Sort of.

BASIC RULES

- ✓ Never trust the client
- ✓ Follow the Principle of Least Privilege
- ✓ Turn on the security linter in Android Studio
- ✓ NEVER STORE YOUR PRIVATE KEY IN THE APP
- ✓ Be as explicit as possible about your app’s intentions
- ✓ **Seriously, never trust the client**

How to Turn on Android Security Linter
File > Settings > Editor > Inspections
http://tools.android.com/tips/lint-checks

6 > CONTINUED LEARNING

RESOURCES

Android Application Security Overview
https://source.android.com/security/overview/app-security

Android Developers: App Security Best Practices
https://developer.android.com/topic/security/best-practices

OWASP Mobile Security Testing Guide (MSTG)
https://github.com/OWASP/owasp-mstg

Android REing Series
https://www.youtube.com/c/chmodxx

BOOKS VIDEOS & COURSES

CYBRARY
Cyber Security & IT Learning

Udemy

lynda.com

Stanford | Continuing Studies

SANS Technology Institute

blackhat®

Android Application Security Essentials
Testing and Securing Android Studio Applications
Security Certification: GMOB
Android Security Internals
An In-Depth Guide to Android's Security
MSTG Mobile Security Testing Guide
EARLY ACCESS
The Mobile Application Hacker's Handbook
WILEY
Android Hacker's Handbook
WILEY

Lookout®

Thank you!

Questions?

@CHMODXX_ @CHMODXX blog.chmodxx.net