ANDROID SECURITY BASICS

Pt. 1: Theory

DISCLAIMER

Pls: Hack4Good

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 <3

Slides & Lab:

https://github.com/chmodxx/Auditing-Pentesting-Android-Apps/

OVERVIEW

1 **Android Basics**

2 **Tools**

3 **Why So Vuln?**

4 **Common Attack Surfaces**

5 **Finding & Fixing Common Vulns**

6 **Continued Learning**

1

Android Basics

“Java on Linux” (Kind of)

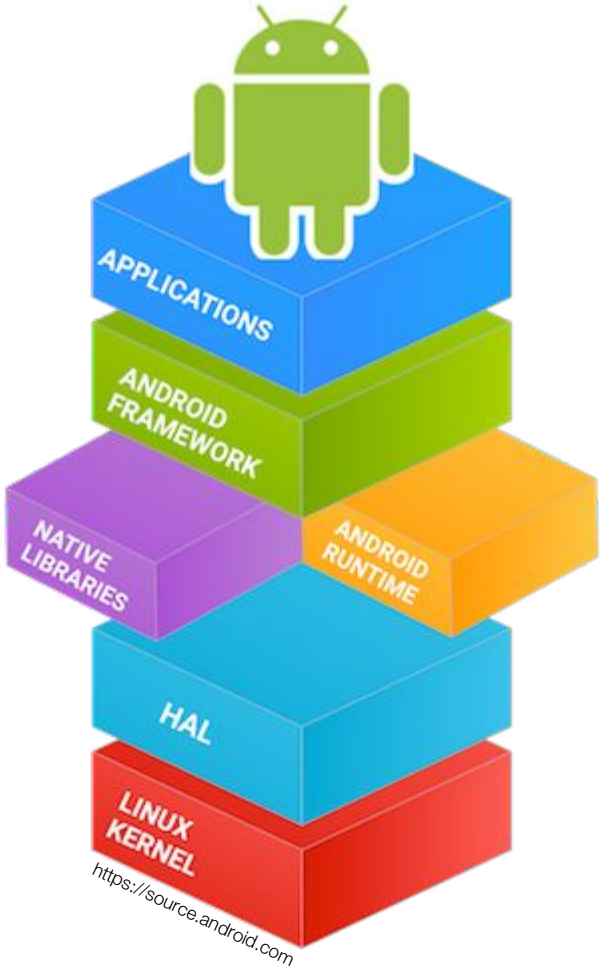
Android applications & framework execute within (initially) DalvikVM and now ART
Provides an abstraction layer to the OS

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

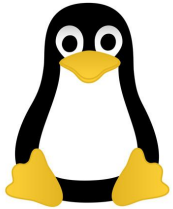
Android runtime
- Defines app permissions



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

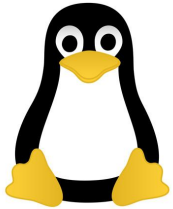
Tools



I <3 linux

KALI

SANTOKU



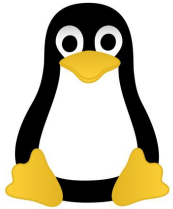
SETTING UP THE ENVIRONMENT

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**

FINDING APKs

★ Download from your connected device via `adb`

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

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

** Sketchy AF. Proceed with caution.

ANALYSIS TOOLS

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/

- ★ 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)

```
$ 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
```

ANALYSIS TOOLS

"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

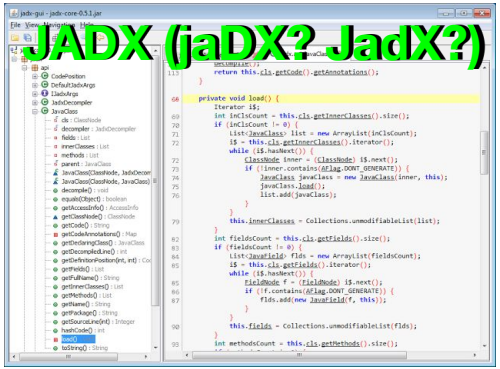
https://ibotpeaches.github.io/Apktool/

- ★ Converts resources back to pretty much their original form
- ★ **DEX > smali**
- ★ Allows you to decompile and recompile APKs

```
$ wget https://bitbucket.org/iBotPeaches/apktool/downloads/apktool_2.3.3.jar
$ wget
https://raw.githubusercontent.com/iBotPeaches/Apktool/master/scripts/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
```

JADX (jaDX? JadX?)



“Command line and GUI tools for producing Java source code from Android Dex and Apk files” - Skylot

https://github.com/skylot/jadx

ANALYSIS TOOLS

★ DEX decompiler

★ CLI and GUI available

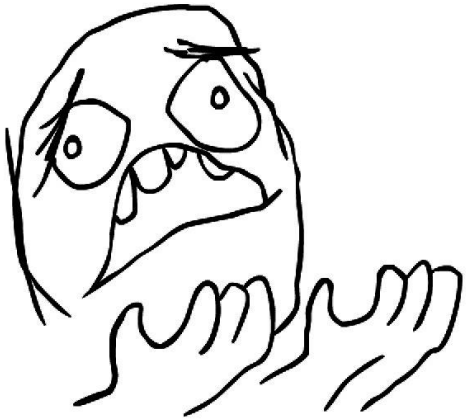
```
$ git clone https://github.com/skylot/jadx.git
$ cd jadx
$ ./gradlew dist


$ bin/jadx-gui [APK].apk # open jadx gui
```

3

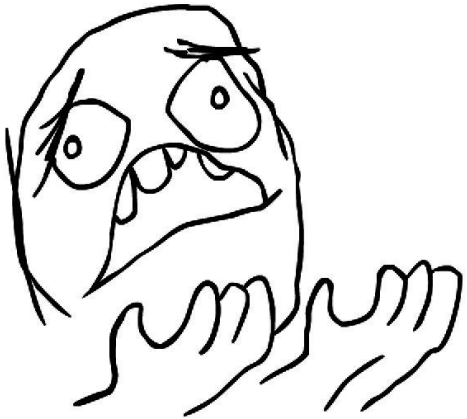
Why So Vuln?

Fragmentation



Fragmentation

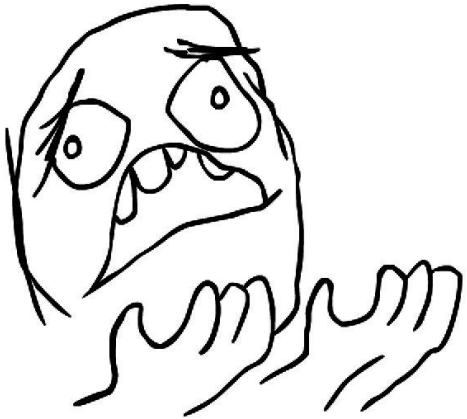
Update Frequency

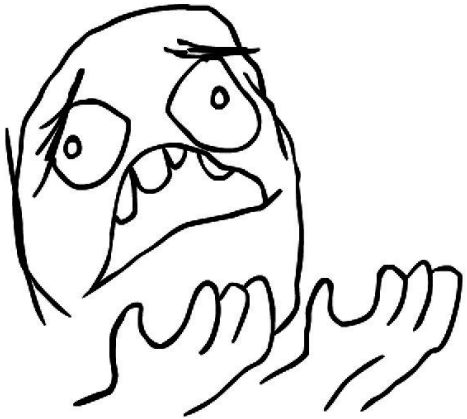


Fragmentation

Update Frequency

Back-porting

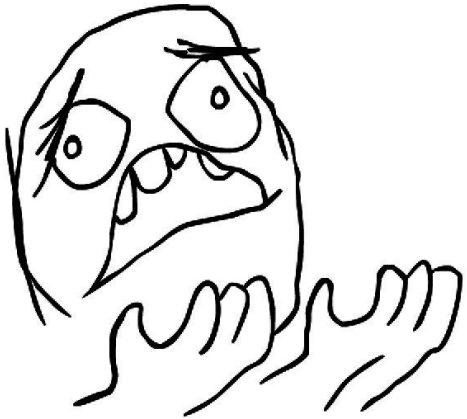




Fragmentation

Update Frequency

Back-porting (or lack thereof) 

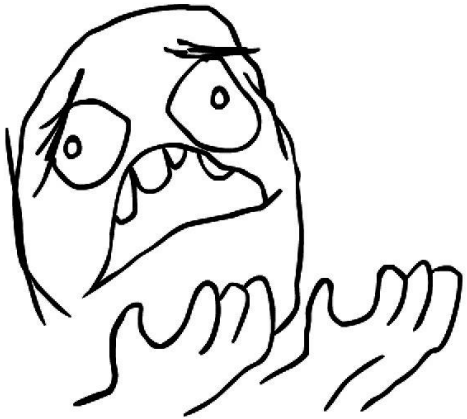


Fragmentation

Update Frequency

Back-porting (or lack thereof) 

Android Update Alliance

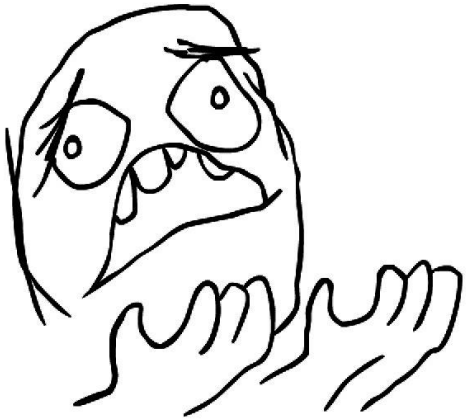


Fragmentation

Update Frequency

Back-porting (or lack thereof) 

Android Update Alliance (or lack thereof) 



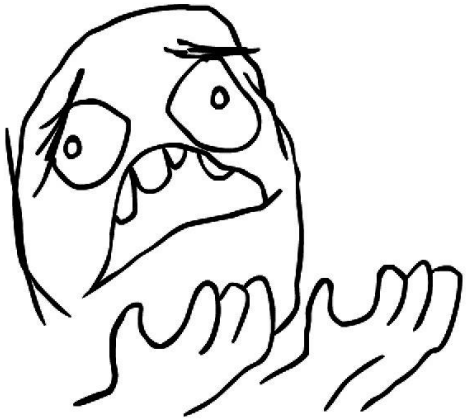
Fragmentation

Update Frequency

Back-porting (or lack thereof) 

Android Update Alliance (or lack thereof) 

Updating Dependencies



Fragmentation

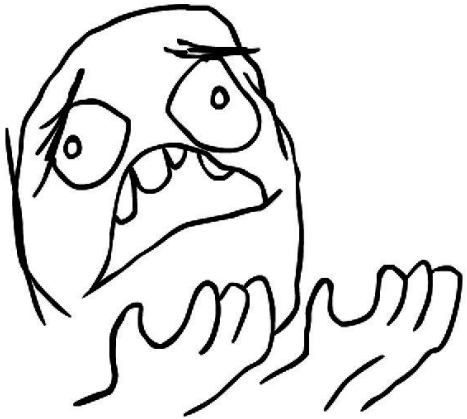
Update Frequency

Back-porting (or lack thereof) 

Android Update Alliance (or lack thereof) 

Updating Dependencies

Open-Source != Secure



Fragmentation

Update Frequency

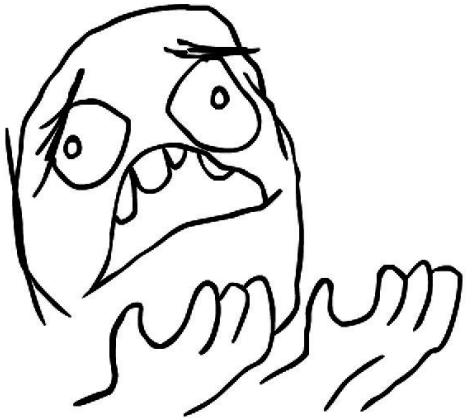
Back-porting (or lack thereof) 

Android Update Alliance (or lack thereof) 

Updating Dependencies

Open-Source != Secure

Public Disclosures



Fragmentation

Update Frequency

Back-porting (or lack thereof) 

Android Update Alliance (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!

STORAGE!

IPC's!

INTENTS!

PERMISSIONS!

BROADCAST RECEIVERS!

WEBVIEWS!

STORAGE!

INTENTS!

PERMISSIONS!

WEBVIEWS!

BROADCAST RECEIVERS!

STORAGE!

INTENTS

CONTENT PROVIDERS!

PERMISSIONS!

WEBVIEWS!

LOGGING!

BROADCAST RECEIVERS!

I/O

STORAGE!

INTENTS

CONTENT PROVIDERS!

PERMISSIONS!

WEBVIEWS!

BROADCAST RECEIVERS!

LOGGING!

I/O

STORAGE!

INTENTS!

CONTENT PROVIDERS!

DATA VALIDATION!

PERMISSIONS!

WEBVIEWS!

BROADCAST RECEIVERS!

LOGGING!

I/O

STORAGE!

INTENTS

CONTENT PROVIDERS!

DATA VALIDATION!

OBFUSCATION!

PERMISSIONS!

WEBVIEWS!

BROADCAST RECEIVERS!

LOGGING!

TAMPERING!

STORAGE!

INTENTS!

CONTENT PROVIDERS!

DATA VALIDATION!

OBFUSCATION!

PERMISSIONS!

TAMPERING!

LOGGING!

DATA VALIDATION!

OBFUSCATION!

5

Finding & Fixing Common Vulns

PERMISSIONS

APPLICATION PERMISSIONS

Developers often request 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.

What to Do:

- ★ 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.

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

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

What to Do:

- ★ 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

IPCs? Interprocess Communications

Endpoints aren't always secured
`Services, Activities, BroadcastReceivers, ContentProviders`

They act as data sinks *and* sources.

Broadcast messages allow any application to receive a broadcasted 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

REAL WORLD EXAMPLE

Samsung Kies app on GalaxyS3

- ★ 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

IPCs

What to Do:

★ 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

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

★ Even content providers that *aren't* exported can be accessed by privileged users

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

★ Similarly, exported activities require no permissions for interaction

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

IPCs

- 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

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

- 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!

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.

Don't embed any encryption key in source code.

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

REAL WORLD EXAMPLE

Skype circa 2011

- ★ Created SQLite databases and XML files with world readable and writable permissions
- ★ Was unencrypted
- ★ Included config data and message logs

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

WhatsApp circa 2014

- ★ Stored database backup on SD card
- ★ Malicious app could have asked for permission to read external storage

Reported by Bas Bosschert, http://bas.bosschert.nl

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

INSECURE COMMS

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 inside a browser and allows applications to add Javascript and a whole bunch of fun things.

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*.

You see this a lot in apps with advertisements.

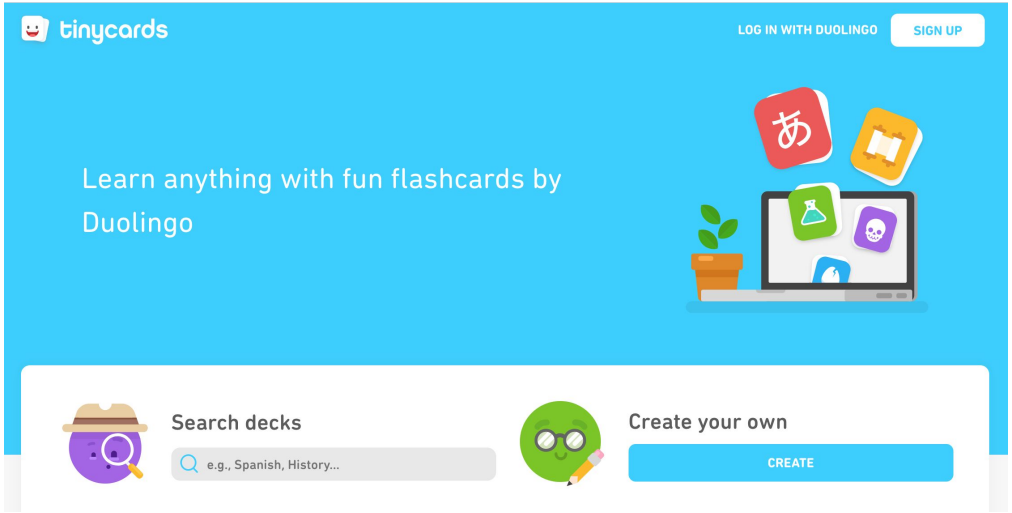
How often does this happen?
Stanford study in 2013:

- 40k apps minimum using a “javascript bridge”
- ⅓ could be reached by untrusted content

WebViews

REAL WORLD EXAMPLE

RCE in TinyCards for Duolingo on Android, Jan 2018
https://hackerone.com/reports/281605



"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

WebViews

What to Do:

- ★ Restrict users to the application domain
- ★ Don’t call setJavaScriptEnabled() until you absolutely have to process Javascript
- ★ APIs 17+ require @JavascriptInterface for any method being 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 by someone trying to inject script

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 coe -- makes it harder to trace what's happening in your app

PRIVATE KEYS & TAMPER DETECTION

PRIVATE KEYS

1. Signing apps
2. 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

People are bad at coming up with passwords, so don't think a password will necessarily foil hackers.

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.

PRIVATE KEYS & TAMPER DETECTION

File Type	Count
APK (Android applications)	1,701,930
PKCS#1/5	549
PKCS#8	240
PKCS#12	2,119
Java Keystore (JKS)	3,215
Bouncy Castle Keystore (BKS) V1	8,450
Bouncy Castle Keystore (BKS) V2	1,668
Openvpn (OVPN)	103 (64 unique)

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.

PRIVATE KEYS & TAMPER DETECTION

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

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.

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 & TAMPER DETECTION

PRIVATE KEYS & TAMPER DETECTION

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.

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%

PRIVATE KEYS & TAMPER DETECTION

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.

4

Common Vulns & How to Avoid Them

PRIVATE KEYS & TAMPER DETECTION

PRIVATE KEYS

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

4

Common Vulns & How to Avoid Them

PRIVATE KEYS & TAMPER DETECTION

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.

4

Common Vulns & How to Avoid Them

PRIVATE KEYS & TAMPER DETECTION

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.

¯_(ツ)_/¯

4

Common Vulns & How to Avoid Them

PRIVATE KEYS & TAMPER DETECTION

TAMPER DETECTION

What to Do:

- ★ Avoid client-side checks
- ★ Google's SafetyNet has some nifty tamper-detection features (that's been fooled :()
  - ○ 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
- ★ SafetyNet also allows server-side checking for application tampering
  - ○ That can be stripped out, too. But, it's better than pure client-side checking

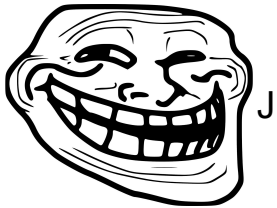
WHAT TO TELL DEVELOPERS?


BE PARANOID

ALL THE TIME

BE PARANOID

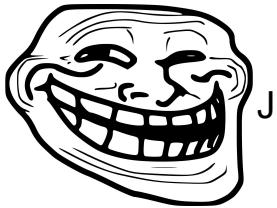
ALL THE TIME



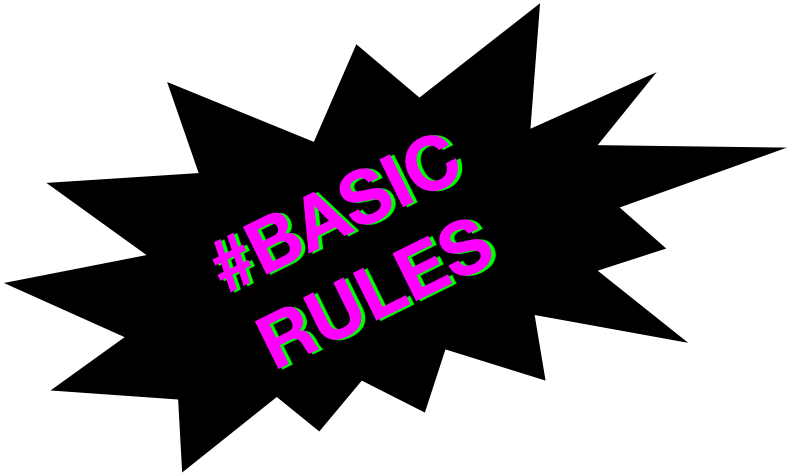
JK.

BE PARANOID

ALL THE TIME



JK. Sort of.



1. Never trust the client.
2. Follow the Principle of Least Privilege
   *(each component or process should have **only** the permissions necessary to perform its tasks)*
3. Turn on the security linter in Android studio
   File > Settings > Editor > Inspections || http://tools.android.com/tips/lint-checks
4. **NEVER STORE YOUR PRIVATE KEY IN THE APP.**
5. Be as explicit as possible about your app's intentions
   Explicit intents where you can, explicit permissions, etc.
6. **Seriously. *Never*** trust the client.

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

Stanford | Continuing Studies

SANS Technology Institute

blackhat®