WHO ARE WE

• Banded together by the love of 0days and hot sauces
• Random out of work collaboration and pursuit of up-leveling the community
• Disclosures / Code / Lessons available on GitHub
• rednaga.io
• github.com/RedNaga
WHO ARE WE

• Researcher @ SentinelOne
• Former Researcher @ SourceClear
  Former Researcher @ Lookout
• Texan at heart, Californian based on shorts
  and sandals 24/7
• Creator of “Simplify”
• @CalebFenton
• github.com/CalebFenton
WHO ARE WE

• Researcher @ SentinelOne
• Former Researcher @ Lookout
• Obfuscation and Packer Junkie
• Makes own hot sauce - cause why not?
• @timstrazz
• github.com/strazzere
WHY ARE WE HERE

More importantly - why should you care?

• Threat Intel is important!

• Used for many purposes:
  • What are people researching now?
  • What should you research next?
  • Anticipate attack patterns
  • Avoid overlap with others!

• We like drinking…
THE TAKE AWAYS
What should you learn from us today?

• How to fingerprint compilers (generically)
• Abnormalities in DEX structure or values
• Signals modification / tampering
• Compiler fingerprinting
• Sophisticated agents
• Related PC stuff
  • PEID - http://www.aldeid.com/wiki/PEiD
CURRENT ANDROID TOOL LANDSCAPE

Tools and Evolution

REDNAGA
ANDROMEDA APPLICATION PACKAGING (APK)

application/vnd.android.package-archive

Blah.apk

META-INF/
  MANIFEST.MF
  CERT_NAME.(RSA | DSA)
  CERT_NAME.SF

lib/
  armeabi(-v7a/)
  arm64-v8a/
  x86/
  mips/

res/
  drawable-*/
  xml/
  raw/
  ...

assets/
  *

AndroidManifest.xml

classes.dex

resources.arsc

Two resources we care about for this presentation specifically
ANDROID APPLICATION PACKAGING (APK)

application/vnd.android.package-archive

**Blah.apk**

- **META-INF/**
  - MANIFEST.MF
  - CERT_NAME.(RSA | DSA)
  - CERT_NAME.SF

- **lib/**
  - armeabi(-v7a)/
  - lib*.so
  - arm64-v8a/
  - x86/
  - mips/

- **res/**
  - drawable-*/
  - *.png
  - xml/
  - *.xml
  - raw/
  - *

- **assets/**
  - *

- **AndroidManifest.xml**
- **classes.dex**
- **resources.arsc**
- *

---

**Android Manifest**

Compiled AndroidXML

Contains:

- entry points for app
- Activities
- Services
- Receivers
- Intents
- ... app permissions
- app meta-data
- package name
- version code/name
- debuggable
- referenced libraries

Reverse with:

- axmlprinter2
- apktool
- jeb / jeb2
- androguard
- 010Editor Templates
ANDROID APPLICATION PACKAGING (APK)
application/vnd.android.package-archive

Blah.apk

META-INF/
  MANIFEST.MF
  CERT_NAME.(RSA | DSA)
  CERT_NAME.SF

lib/
  armeabi(-v7a)/
  arm64-v8a/
  x86/
  mips/
  lib*.so

res/
  drawable-*/
  *.png
  xml/
  *.xml
  raw/
  ...

assets/
  *

AndroidManifest.xml
  classes.dex
  resources.arsc
  *

Android Manifest
Compiled AndroidXML

Created by:
aapt
axmlprinter2 (new ver)
apktool
(axmlprinter2 mod)
random Python scripts
ANDROiD APPLICATION PACKAGING (APk)

application/vnd.android.package-archive

Used by normal devs

Android Manifest
Compiled AndroidXML

Created by:
aapt
axmlprinter2 (new ver)
apktool
(axmlprinter2 mod)
random Python scripts

META-INF/
MANIFEST.MF
CERT_NAME.(RSA | DSA)
CERT_NAME.SF

lib/
armeabi(-v7a)/
arm64-v8a/
x86/
mips/

lib*.so

res/
drawable-*/
xml/
raw/
...

assets/ *

AndroidManifest.xml

classes.dex

resources.arsc

*
ANDROID APPLICATION PACKAGING (APK)

application/vnd.android.package-archive

Blah.apk

META-INF/
  MANIFEST.MF
  CERT_NAME.(RSA | DSA)
  CERT_NAME.SF

lib/
  armeabi(-v7a)/
  arm64-v8a/
  x86/
  mips/

lib*.so

res/
  drawable-*/
  *.png
  xml/
  *.*

AndroidManifest.xml

classes.dex

resources.arsc

Used by normal devs

Used by “abnormal” devs
  - Security tools
  - “injection” tools

Almost all used for post-compilation modification

Created by:
  aapt
  axmlprinter2 (new ver)
  apktool
    (axmlprinter2 mod)
  random Python scripts
All of these things are “interesting” depending on how you look at it…

New malware?
New security tool (ab)using the system?
Play Store APKs look different than in the wild binaries?
AXML OPEN SOURCE CYCLE

Who is using what?

AXMLPrinter2 is a very, very old project with bugs...

- Was the standard which people found breakages in

- Code used by APKTool (licenses appear stripped)

- JEB imported APKTool (seen in licenses)

- JEB author back ported fixed into APKTool

- Library to break them all! (until jeb2)
AXML OPEN SOURCE CYCLE

Who is using what?

AXMLPrinter2 is a very, very old project with bugs...

- Was the standard which people found breakages in
- Code used by APKTool (licenses appear stripped)
- JEB imported APKTool (seen in licenses)
- JEB author back ported fixed into APKTool
- Library to break them all! (until jeb2)

FOSS remake released; https://github.com/rednaga/axmlprinter

~85% TCC

Allows reading / writing AXML
Avoids previous breakages
Can be used to detect these changes
ANDROİD APPLICATION PACKAGING (APK)

Blah.apk

- META-INF/
  - MANIFEST.MF
  - CERT_NAME.(RSA | DSA)
  - CERT_NAME.SF

- lib/
  - armeabi(-v7a)/
  - lib*.so
  - arm64-v8a/
  - x86/
  - mips/

- res/
  - drawable-*/
  - *.png
  - xml/
  - *.xml
  - raw/
  - ...

- assets/
  - *

- AndroidManifest.xml

- classes.dex

- resources.arsc
  - *

Dalvik Executable
Compiled classes for DVM

Contains executable Dalvik code

Optimized on install to:
ODEX for DVM runtime
OAT for ART runtime

Reverse with:
smali / apktool
IDA Pro
jeb / jeb2
androguard
enjarify
dex2jar + jad/jd
jadx
radare
010Editor Templates
ANDROiD APPLICATION PACKAGING (APK)

application/vnd.android.package-archive

Blah.apk

META-INF/MANIFEST.MF
CERT_NAME.(RSA | DSA)
CERT_NAME.SF
MANIFEST.MF

lib/
armeabi(-v7a)/
arm64-v8a/
x86/
mips/

lib*.so

res/
drawable-*/
xml/
raw/

*.png
*.xml

resources.arsc

AndroidManifest.xml
classes.dex

Dalvik Executable
Compiled classes for DVM

Contains executable Dalvik code

Optimized on install to:
ODEX for DVM runtime
OAT for ART runtime

Reverse with:
smali / apktool
IDA Pro
jeb / jeb2
androguard
enjarify
dex2jar + jad/jd
jadx
radare
010Editor Templates

All open sourced tools

androguard used by VT
(acquired by Google)

smali creator/maintainer now works at Google, used in AOSP

enjarify made by Google

dex2jar creator/maintainer works(ed?) at Trend

radare creator/maintainer works at NowSecure
ANDROID APPLICATION PACKAGING (APK)

application/vnd.android.package-archive

Blah.apk

META-INF/ MANIFEST.MF
CERT_NAME.(RSA | DSA)
CERT_NAME.SF

lib/ armeabi(-v7a)/ lib*.so

META-INF/ MANIFEST.MF
CERT_NAME.(RSA | DSA)
CERT_NAME.SF

Dalvik Executable
Compiled classes for DVM

Contains executable
Dalvik code

Optimized on install to:
ODEX for DVM runtime
OAT for ART runtime

Reverse with:
smalii / apktool
IDA Pro
jeb / jeb2
androguard
enjarify
dex2jar + jad/jd
jadx
radare
010Editor Templates

apkiotool used original
axmlprinter2 code, now
mostly refactored out

jeb (maybe jeb2?) originally
used apktool for resource
parsing and back ported
patches for resources which
broke the non-free tool
ANDROID APPLICATION PACKAGING (APK)

application/vnd.android.package-archive

Blah.apk

META-INF/
MANIFEST.MF
CERT_NAME.(RSA | DSA)
CERT_NAME.SF

lib/
armeabi(-v7a)/
lib*.so
arm64-v8a/
x86/
mips/

res/
drawable-*/
xml/
raw/
…

assets/
*

AndroidManifest.xml

classes.dex

resources.arsc

Contains or is a **disassembler** which can provide a more direct translation to what the Android VM will see.

Usually requires learning simple Jasmin like language syntax.

Dalvik Executable
Compiled classes for DVM

Contains executable Dalvik code

Optimized on install to:
ODEX for DVM runtime
OAT for ART runtime

Reverse with:
smali / apktool
IDA Pro
jeb / jeb2
androguard
enjarify
dex2jar + jad/jd
jadx
radare
010Editor Templates
ANDROiD APPLICATION PACKAGING (APK)
application/vnd.android.package-archive

Blah.apk

META-INF/
MANIFEST.MF
CERT_NAME.(RSA | DSA)
CERT_NAME.SF

lib/
armeabi(-v7a)/
lib*.so
arm64-v8a/
x86/
mips/

res/
drawable-*/
*.png

drawable-*/
xml/
raw/
*

AndroidManifest.xml
classes.dex
resources.arsc
*

Contains or is a **decompiler** which will attempt to translate actual code to (usually) Java code.

Can allow leveraging usual Java tools and code review style of reverse engineering.

**Dalvik Executable**
Compiled classes for DVM

Contains executable Dalvik code

Optimized on install to:
ODEX for DVM runtime
OAT for ART runtime

Reverse with:
smalli / apktool
IDA Pro
jeb / jeb2
androguard
enjarify
dex2jar + jad/jd
jadx
radare
010Editor Templates
ANDROID APPLICATION PACKAGING (APK)

application/vnd.android.package-archive

Blah.apk

META-INF/
METANNIST.MF
CERT_NAME.(RSA | DSA)
CERT_NAME.SF

lib/ armeabi(-v7a)/
arm64-v8a/
x86/
mips/

res/ drawable-*/
xml/
raw/

... *

AndroidManifest.xml
classes.dex
resources.arsc

Scriptable or accessible via an APIs to allow plugins or potential automation.

Dalvik Executable
Compiled classes for DVM
Contains executable Dalvik code
Optimized on install to:
ODEX for DVM runtime
OAT for ART runtime

Reverse with:
smali / apktool
IDA Pro
jeb / jeb2
androguard
enjarify
dex2jar + jad/jd
jadx
jad
radare
010Editor Templates
ANDROID APPLICATION PACKAGING (APK)

Blah.apk

META-INF/
  MANIFEST.MF
  CERT_NAME.(RSA | DSA)
  CERT_NAME.SF

lib/
  armeabi(-v7a)/
  arm64-v8a/
  x86/
  mips/

res/
  drawable-*/
  *.png
  xml/
  *.xml

raw/

assets/

AndroidManifest.xml

classes.dex

resources.arsc

Easy to understand hex viewer with FOSS templates for Dalvik.

Excellent for determining forensic differences between files, looking for “oddities”, etc.

Android Executable
Compiled classes for DVM

Contains executable Dalvik code

Optimized on install to:
ODEX for DVM runtime
OAT for ART runtime

Reverse with:
smalii / apktool
IDA Pro
jeb / jeb2
androguard
enjarify
dex2jar + jad/jd
jadx
radare
010Editor Templates
ANDROID APPLICATION PACKAGING (APK)

application/vnd.android.package-archive

Blah.apk

META-INF/
MANIFEST.MF
CERT_NAME.(RSA | DSA)
CERT_NAME.SF
lib/
armeabi(-v7a)/
arm64-v8a/
x86/
mips/

res/
drawable-*/
xml/
raw/…

*.

AndroidManifest.xml
classes.dex
resources.arsc
*

“Official” / standard tools included in the Android SDK.

dx compiles Java .class to .dex

dexmerge combines .dex files and is used by some IDEs for “incremental builds”

Dalvik Executable
Compiled classes for DVM

Contains executable Dalvik code

Optimized on install to:
ODEX for DVM runtime
OAT for ART runtime

Reverse with:
smali / apktool
IDA Pro
jeb / jeb2
androguard
enjarify
dex2jar + jad/jd
jadx
radare
010Editor Templates
COMPILER FINGERPRINTING

diff / caleb

REDNAGA
AXML FILES

Relatively Simplistic...

- Normal tools create AXML file in a simple order
- AXML files don’t need to be in a specific order
- Most tools **append** new structures to the file
Normal Files

AndroidManifest.xml

- **Header**
  - Package Name
  - Version String
  - Version Code

- **Uses SDK**
  - Min version
  - Max version

- **Permissions**
  - alphabetical order

- **Application**
  - alphabetical order

- **Activities**
  - alphabetical order

- **Services**
  - alphabetical order
AXML FILES

Abnormal Files

AndroidManifest.xml

Header
Package Name
Version String
Version Code
Uses SDK
Min version
Max version
Permissions
alphabetical order
Application
alphabetical order
Activities
alphabetical order
Services
alphabetical order
Permissions
etc

Orders mismatch
AXML FILES

Normal Files
**AXML FILES**

**Normal Files**

Spacing between characters

Due to this flag (in spec)
AXML FILES

Abnormal files which broke old AXMLPrinter2 lib

No spacing between characters

Due to this flag (in spec)

This was back ported from JEB to APKTOOL...
AXML FILES

Protectors / Anti* tricks

```
[42%] diff@rocksteady:[axml_tests] $ axml power_profile.xml
<?xml version="1.0" encoding="utf-8"?>
java.lang.ArrayIndexOutOfBoundsException: 140 at android.content.res.StringBlock.getShort(StringBlock.java:231)
at android.content.res.StringBlock.getString(StringBlock.java:91)
at android.content.res.AXmlResourceParser.getName(AXmlResourceParser.java:140)
at test.AXMLPrinter.main(AXMLPrinter.java:56)
```
Protectors / Anti* tricks

```java
java.lang.ArrayIndexOutOfBoundsException: 140
at android.content.res.StringBlock.getShort(StringBlock.java:231)
at android.content.res.StringBlock.getString(StringBlock.java:91)
at android.content.res.AXmlResourceParser.getName(AXmlResourceParser.java:140)
at test.AXMLPrinter.main(AXMLPrinter.java:56)
```
AXML FILES

Protectors / Anti* tricks

Originally found by dexguard, didn’t work on all Android versions

Replicated by malware

Tools expected name tags
AXML FILES

APKTOOL Specifics... easy, easy
AXML FILES

APKTOOL Specifics... easy, easy

Uhhh, thanks?
DEX FILES

• DEX format is ... flexible
• Only a few different compilers
• Slight variations between each one
• Obfuscators do really weird stuff too
INVESTIGATION

• Built lots of DEX files with different tools
• Compared files with 010Editor
• Found some differences but wanted to know all of them
• Read DEX format specification
• Gave up since it doesn’t include enough detail
• Very carefully read the source code
• Found many fingerprintable “characteristics”
CHARACTERISTICS

These may be abnormal...

1. Class interfaces
2. Class paths
3. Endian tag
4. Header size
5. Link section
6. String sorting
7. Map type order
8. Section contiguity
ABNORMAL_CLASS_INTERFACES

- Implies: early dexlib 2.x (smali)

If class has no interface, dx uses interfaces_off = 0, dexlib gives offset to address with null bytes (10156 is null)
ABNORMAL_CLASS_PATH

- Implies: anti-decompiler

<table>
<thead>
<tr>
<th>struct class_def_item_list dex_class_defs</th>
<th>317 classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct class_def_item class_def[0]</td>
<td>public final android.media.AmrInputStream</td>
</tr>
<tr>
<td>struct class_def_item class_def[1]</td>
<td>public final o.if</td>
</tr>
<tr>
<td>struct class_def_item class_def[2]</td>
<td>public final o.Con</td>
</tr>
<tr>
<td>struct class_def_item class_def[3]</td>
<td>public abstract o.á§</td>
</tr>
<tr>
<td>struct class_def_item class_def[4]</td>
<td>public abstract o.CON</td>
</tr>
<tr>
<td>struct class_def_item class_def[5]</td>
<td>public final o.Ø</td>
</tr>
<tr>
<td>struct class_def_item class_def[6]</td>
<td>final o.áμ</td>
</tr>
<tr>
<td>struct class_def_item class_def[7]</td>
<td>public final o.áμ£</td>
</tr>
<tr>
<td>struct class_def_item class_def[8]</td>
<td>public abstract o.á±</td>
</tr>
<tr>
<td>struct class_def_item class_def[9]</td>
<td>public final o.í¶</td>
</tr>
<tr>
<td>struct class_def_item class_def[10]</td>
<td>public abstract o.á&quot;</td>
</tr>
<tr>
<td>struct class_def_item class_def[11]</td>
<td>public final o.í¹³</td>
</tr>
</tbody>
</table>

Invalid Windows filenames:

- CON, PRN, AUX, CLOCK$, NUL
- COM1, COM2, COM3, COM4
- LPT1, LPT2, LPT3, LPT4

Decompilers output filenames based on class name
ABNORMAL_CLASS_PATH

- Implies: anti-decompiler

<table>
<thead>
<tr>
<th>struct class_def_item class_def[377]</th>
<th>public final com.maxmpz.audioplayer.data.ÑLKwleklijkj5w3lkjlkjêjOWEmNWHEHKSPIJLNWLHNWLHJDKWFWISJNNNHBHWKEWYHEYWPWW</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint class_idx</td>
<td>(0x2E8) com.maxmpz.audioplayer.data.ÑLKwleklijkj5w3lkjlkjêjOWEmNWHEHKSPIJLNWLHNWLHJDKWFWISJNNNHBHWKEWYHEYWPWW</td>
</tr>
<tr>
<td>enum ACCESS_FLAGS access_flags</td>
<td>(0x11) ACC_PUBLIC ACC_FINAL</td>
</tr>
<tr>
<td>uint superclass_idx</td>
<td>(0x79A) java.lang.Object</td>
</tr>
<tr>
<td>uint interfaces_off</td>
<td>0</td>
</tr>
<tr>
<td>uint source_file_idx</td>
<td>(0x19B) ****</td>
</tr>
<tr>
<td>uint annotations_off</td>
<td>0</td>
</tr>
<tr>
<td>uint class_data_off</td>
<td>1648319</td>
</tr>
<tr>
<td>struct class_data_item class_data</td>
<td>2 static fields, 0 instance fields, 6 direct methods, 0 virtual methods</td>
</tr>
<tr>
<td>uint static_values_off</td>
<td>0</td>
</tr>
</tbody>
</table>

"com.maxmpz.audioplayer.data.ÑLKwleklijkj5w3lkjlkjêjOWEmNWHEHKSPIJLNWLHNWLHJDKWFWISJNNNHBHWKEWYHEYWPWWKELWJEKWEWNELEWJEHWELKWUEW
ekWLRJFKWNENWJEKWEKJHWKEWJHRKLWJEKKWEHJEHHJkWkhrwcnwemnnrwkj5n4m4mwn54mnkhjJNndenrrrrr3453nmNMEWERTENRNERMERJEJRNNWKJEWNEWWKEJWED"”

Looks legit!

Class name used for filename!

Too long for Windows

Most Linux file systems have no limit

NTFS limited to 255 characters per part
ABNORMAL_ENDIAN_MAGIC

- Implies: weird, shouldn’t run on any Android device
ABNORMAL_HEADER_SIZE

- Implies: weird, possibly hiding data after header before string table

header_size normally 0x70 (112) bytes
ABNORMAL_LINK_SECTION

- Implies: anti-decompiler

link_offset and size always 0 in DEX files
ABNORMAL_STRING_SORT

- Implies: dexlib 1.x

<table>
<thead>
<tr>
<th>Normal</th>
<th>Abnormal</th>
</tr>
</thead>
<tbody>
<tr>
<td>string[0] starts @2162</td>
<td>string[1] starts way after string[0]</td>
</tr>
<tr>
<td>string[1] starts immediately after string[0]</td>
<td>2234 + len(&quot;&lt;init&gt;&quot;) != 3427</td>
</tr>
</tbody>
</table>

- 2234 + len("<init>") != 3427
ABNORMAL_TYPE_ORDER

- Implies: something other than dx or dexmerge

<table>
<thead>
<tr>
<th>dx Map Item Order</th>
<th>dexmerge Map Item Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. HEADER_ITEM</td>
<td>1. HEADER_ITEM</td>
</tr>
<tr>
<td>2. STRING_ID_ITEM</td>
<td>2. STRING_ID_ITEM</td>
</tr>
<tr>
<td>3. TYPE_ID_ITEM</td>
<td>3. TYPE_ID_ITEM</td>
</tr>
<tr>
<td>4. PROTO_ID_ITEM</td>
<td>4. PROTO_ID_ITEM</td>
</tr>
<tr>
<td>5. FIELD_ID_ITEM</td>
<td>5. FIELD_ID_ITEM</td>
</tr>
<tr>
<td>6. METHOD_ID_ITEM</td>
<td>6. METHOD_ID_ITEM</td>
</tr>
<tr>
<td>7. CLASS_DEF_ITEM</td>
<td>7. CLASS_DEF_ITEM</td>
</tr>
<tr>
<td>8. ANNOTATION_SET_REF_LIST</td>
<td>8. MAP_LIST</td>
</tr>
<tr>
<td>9. ANNOTATION_SET_ITEM</td>
<td>9. TYPE_LIST</td>
</tr>
<tr>
<td>10. CODE_ITEM</td>
<td>10. ANNOTATION_SET_REF_LIST</td>
</tr>
<tr>
<td>11. ANNOTATIONS_DIRECTORY_ITEM</td>
<td>11. ANNOTATION_SET_ITEM</td>
</tr>
<tr>
<td>12. TYPE_LIST</td>
<td>12. CLASS_DATA_ITEM</td>
</tr>
<tr>
<td>13. STRING_DATA_ITEM</td>
<td>13. CODE_ITEM</td>
</tr>
<tr>
<td>14. DEBUG_INFO_ITEM</td>
<td>14. STRING_DATA_ITEM</td>
</tr>
<tr>
<td>15. ANNOTATION_ITEM</td>
<td>15. DEBUG_INFO_ITEM</td>
</tr>
<tr>
<td>16. ENCODED_ARRAY_ITEM</td>
<td>16. ANNOTATION_ITEM</td>
</tr>
<tr>
<td>17. CLASS_DATA_ITEM</td>
<td>17. ENCODED_ARRAY_ITEM</td>
</tr>
<tr>
<td>18. MAP_LIST</td>
<td>18. ANNOTATIONS_DIRECTORY_ITEM</td>
</tr>
</tbody>
</table>

**not made with dx or dexmerge, probably dexlib because TYPE_STRING_DATA_ITEM comes after TYPE_CLASS_DEF_ITEM**
Non-Contiguous Section

- Implies: weird, maybe dexmerge

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint type_ids_size</code></td>
<td>61</td>
</tr>
<tr>
<td><code>uint type_ids_off</code></td>
<td>7784</td>
</tr>
<tr>
<td><code>uint proto_ids_size</code></td>
<td>38</td>
</tr>
<tr>
<td><code>uint proto_ids_off</code></td>
<td>12124</td>
</tr>
</tbody>
</table>

Proto_ids should come after type_ids

- `type_id_item size = 4 bytes`
- `type_ids_size * 4 = 244`
- `type_ids_off + 244 = 8028`
- `proto_ids actually starts 12124! weird!`
MALWARE AND PIRACY DETECTION
THE QUESTION

Three main compilers:

1. dx → Java .class files (source code)
2. dexmerge → Not used manually, only by IDEs (source code)
3. smali (dexlib) → DEX files (not source code)

Why would a legitimate developer ever need to use smali?

They have the source.
If app compiled with dexlib, probably tampered

If tampered, probably was not the developer

Tampered apps are likely either:

- pirated / cracked
- malware

∴ app is tampered -> app is interesting
SAMPLE SET

- 20,000 APKs from each market

- Top Play Apps, Aptoide, BlapkMarket, etc.

- 10,000 highest scoring “fraudulent” apps

- Scored by experimental statistical model

- Fraud may just mean modified XML (not DEX)

- Up to 10 APKs per variant of all malware families
THE METHOD

• Scanned the DEX of each APK

• Did not scan AXML files

• Tampered means:
  • abnormal string sort, class path, type order

• Weird means:
  • abnormal endian magic, header size, type descriptor, class path
RESULTS: SOURCE TAMPERING

- Sample: 32%
- Fraud: 15%
- BlapkMarket: 20%
- Aptoide: 15.5%
- AndroidMarketTopApps: 4%
- Other: 12%
RESULTS: MALWARE TAMPERING

• 756 malware families (many variants each)

• 17508 malicious APKs

• 50% families have some tampering

• 50% families have no tampering

• 85 families are 100% tampered

• Each family has a tampering profile
100% TAMPERED FAMILIES

AdultFreedom Alsalah AncientSMSThief AppleService AvariceYY BadSerial
BadSub Badaccents BankMirage Bgserv BiggBoss CastilStyle CataChar
CnSky CoinKrypt DeCerTasks DidStall DirtyAir DoubleZero EasyPine
EdeFraud EmmentalCrupt ErrthangSms Euroxbox ExplicitHorse FadeSMS
FakeActivate FakeKakao FastUninstallRepackaged FineFocusAds
FlaccidForest GauerCloud Geinimi GoGuangGo Gone60 ImAdPush KHSms
Kakabet KhpowSms KrabBot Krysanec LidLocker LoveMii MMarketPay
MaClickFraud MirvspySMS MixedSmoke MmsMore MocheYY Moghava
Obad OccupyYourPrivacy PVAFraud PhoXinhSms PicSysCom PirateShame
PlusTV PopTest RootSmart RuSms Samsapo SandroRat SimpleTemai
SixPointFourSMS SlyInstall SmsMonitur SneakyBeeSMS Stask Stoqx
StorgeSMS SwfScam SwiftLogger Taotobo Tornika
UniversalAndrootRepackaged VDLoader Vchargelet VideoBoss
VservSubscription WinAdOffers WrongPath XSider Xybot YobaSMS ZxtdPay
CONCLUSION

• Few legitimate apps are tampered

• Tampering good signal for malware / piracy

• Better able to understand malware family evolution
APKID DEMO

REDNAGA
EXTENDED READING

https://github.com/rednaga/training
https://github.com/strazzere/anti-emulator/tree/master/slides
https://github.com/strazzere/android-unpacker/blob/master/AHPL0.pdf
http://www.droidsec.org/wiki/#whitepapers
http://calebfenton.github.io/
http://androidcracking.blogspot.com/
THANKS!

TIM “DIFF” STRAZZERE
@TIMSTRAZZ

CALEB FENTON
@CALEB_FENTON

Special Thanks for Jacob Soo and Mikachu for all your assistance!

Join use on Freenode on #droidsec and #rednaga

Good people to follow on Twitter for Android /reversing /malware / hacking information:

@_jsoo_ @droidsec @jcase @marcwrogers @moong1ider @msolnik
@osxreverser @PatrickMcCanna @rotlogix @snare @tamakikusu @trimosx
#MalwareMustDie

07.22.2016

HITCON COMMUNITY

REDNAGA