EVOLUTION OF ANDROID EXPLOITS FROM A STATIC ANALYSIS TOOLS PERSPECTIVE

VB2014: **Anna Szalay & Jagadeesh Chandraiah**

Email {anna.szalay, jagadeesh.chandraiah}@ sophos.com

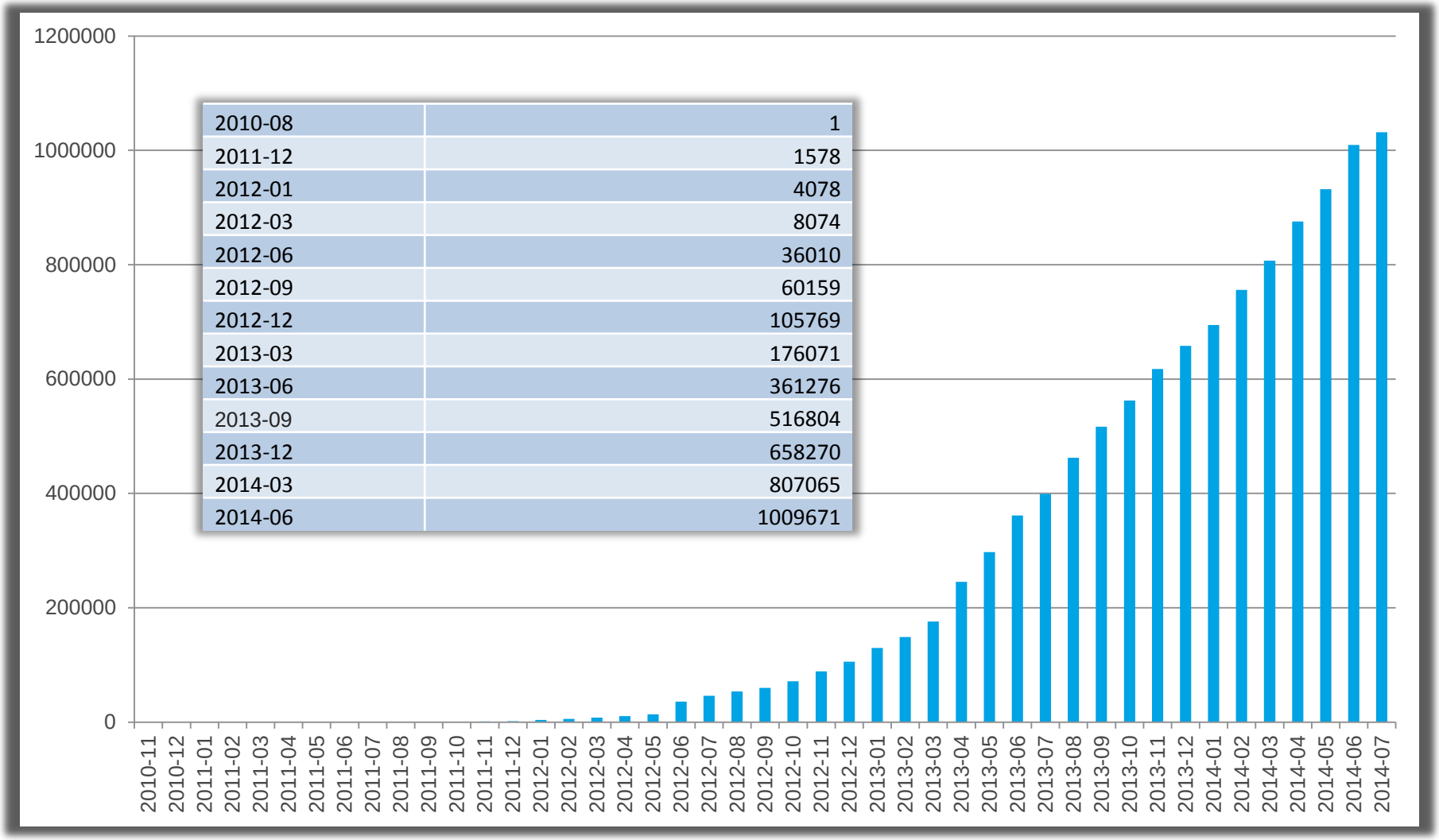
SOPHOS

Devices cumulative

Operating System	2013	2014	2015
Android	879,821	1,170,952	1,358,265
Windows	325,127	339,068	379,299
iOS/Mac OS	241,416	286,436	324,470
Others	873,194	683,519	565,186
Total	**2,319,559**	**2,479,976**	**2,627,221**

Worldwide Device Shipment by Operating System (Thousands of Units). Gartner (March 2014). http://techcrunch.com/2014/03/27/gartner-devices-forecast-2014.

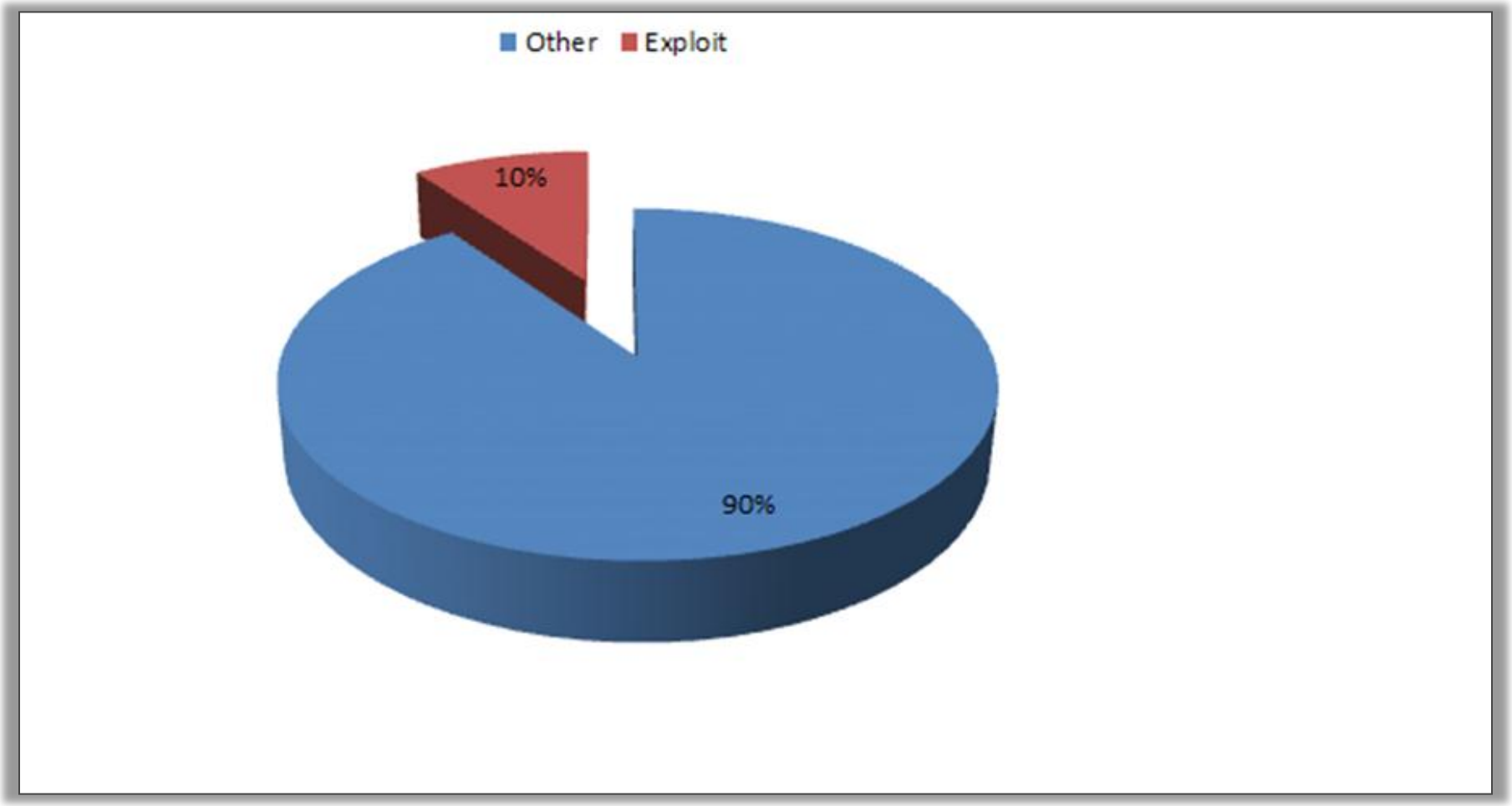
Malware Samples cumulative



2010-08	1
2011-12	1578
2012-01	4078
2012-03	8074
2012-06	36010
2012-09	60159
2012-12	105769
2013-03	176071
2013-06	361276
2013-09	516804
2013-12	658270
2014-03	807065
2014-06	1009671

SOPHOS

Malware exploits samples share



Exploits overview

Android Vulnerabilities Exploitation Timeline

Q4 2010	Android WebKit browser exploit
Q4 2010	Android Data Stealing Vulnerability
Q1 2011	Android Local Rooot Exploit aka “Rage against the cage” or Lotoor exploit
Q3 2011	Android ClientLogin Protocol Vulnerability
Q3 2011	Android Gingerbreak root exploit
2012	…
Q2 2013	DEX2JAR exploitation
Q2 2013	“Master Key” vulnerability
Q2 2013	“Extra Field” vulnerability

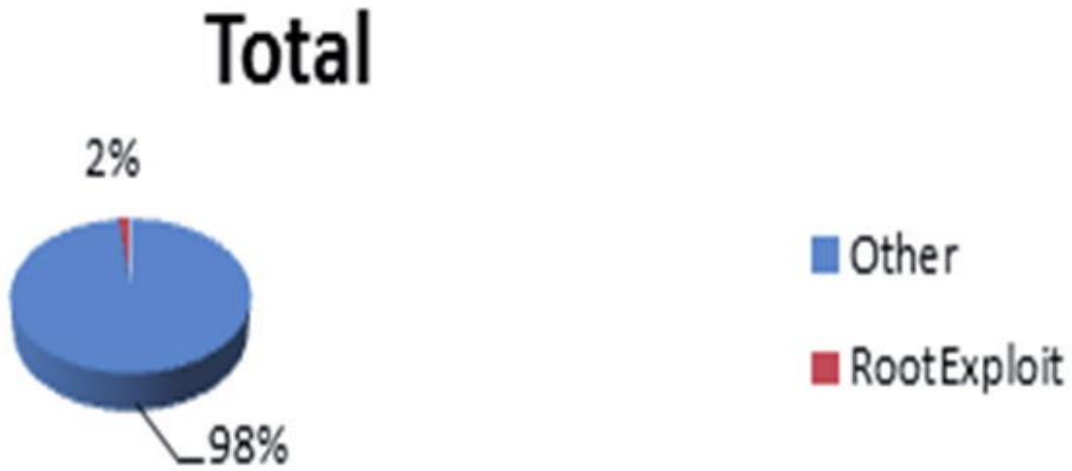
Exploits overview

Q1 2011

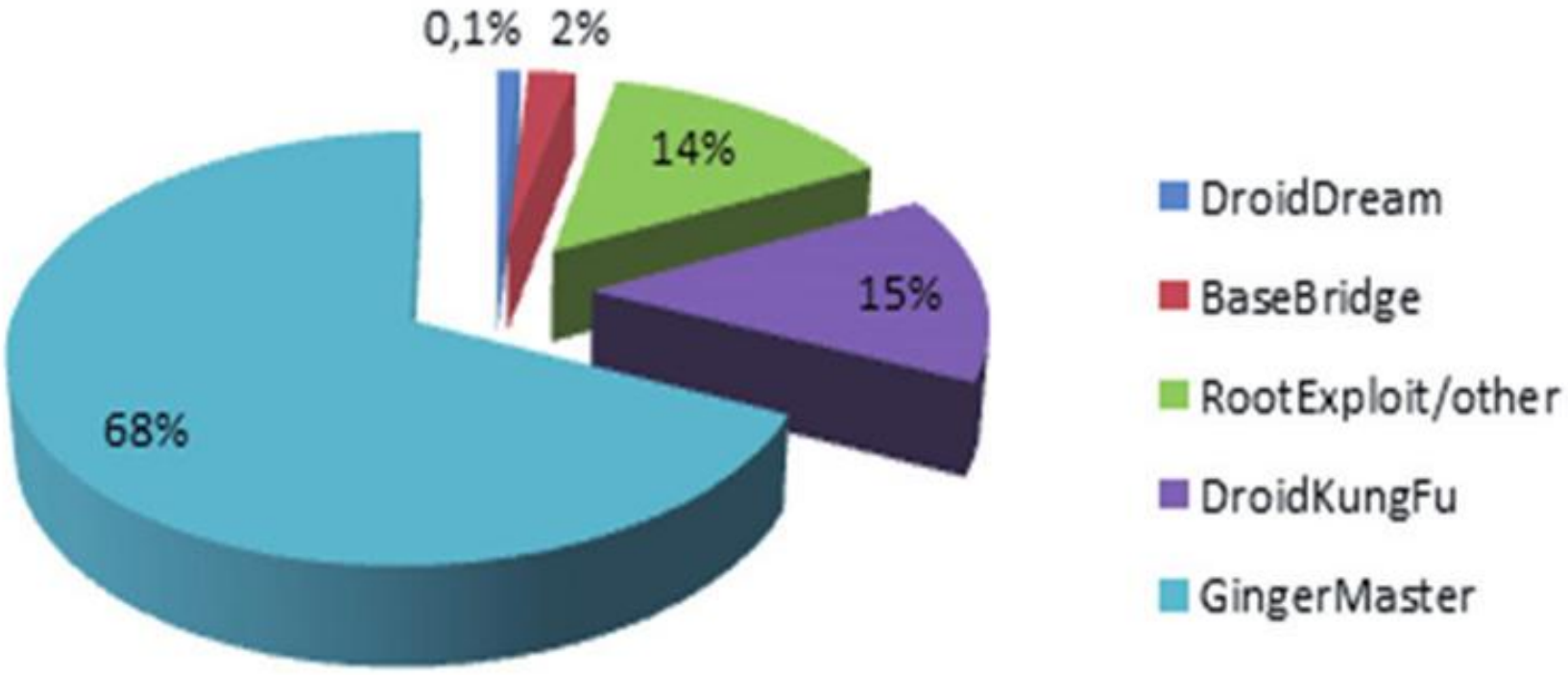
Android Local Root Exploit aka “Rage against the cage” or Lotoor exploit

- **Andr/DroidRt aka DroidRoot (ELF EXECUTABLE)**
- **Andr/DroidD aka DroidDream**
- **Andr/KongFu aka DroidKungFu**

Malware root exploit samples share



Share of root-exploit-based malware samples by threat



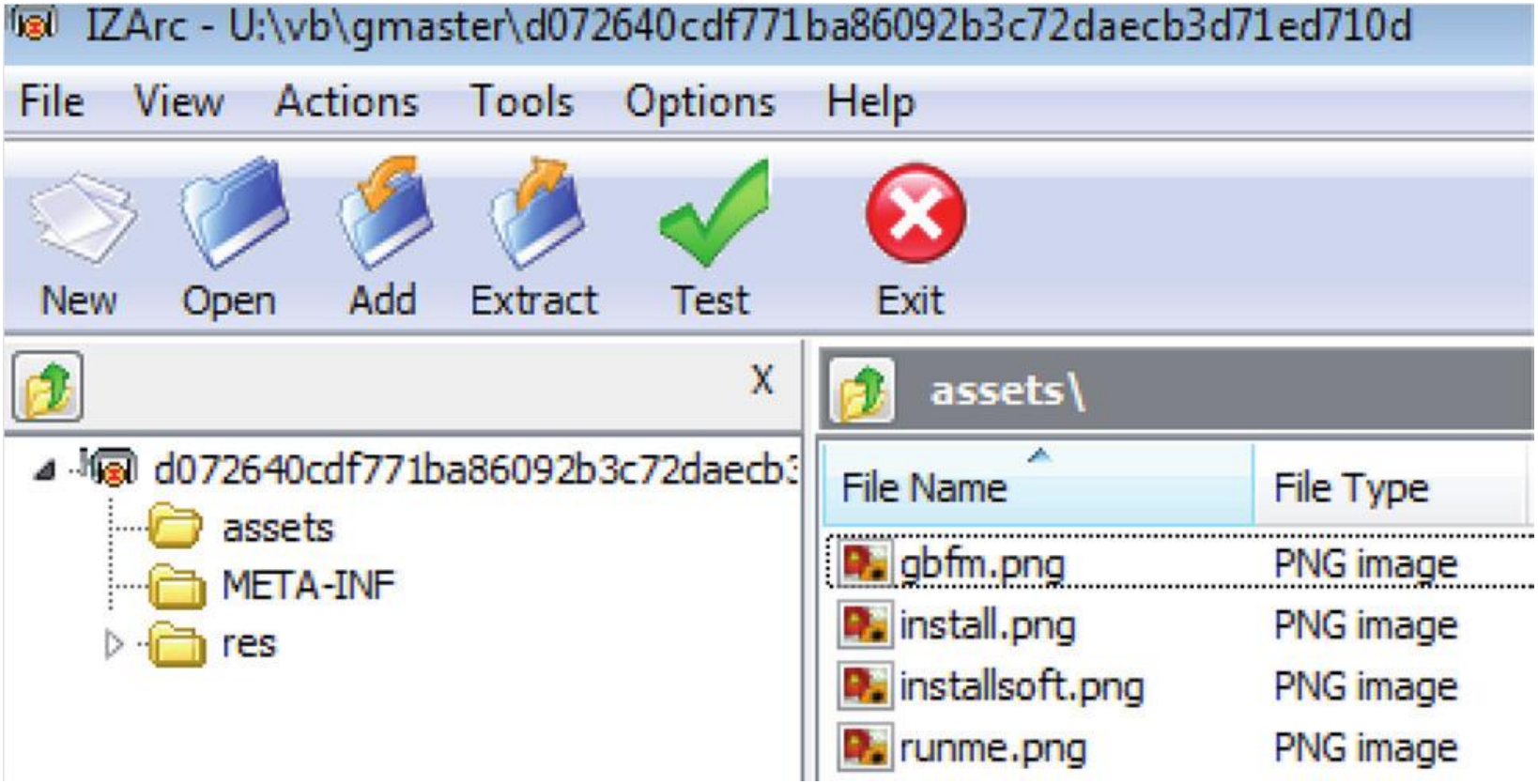
Exploits overview

Q3 2011

Android Gingerbreak root exploit

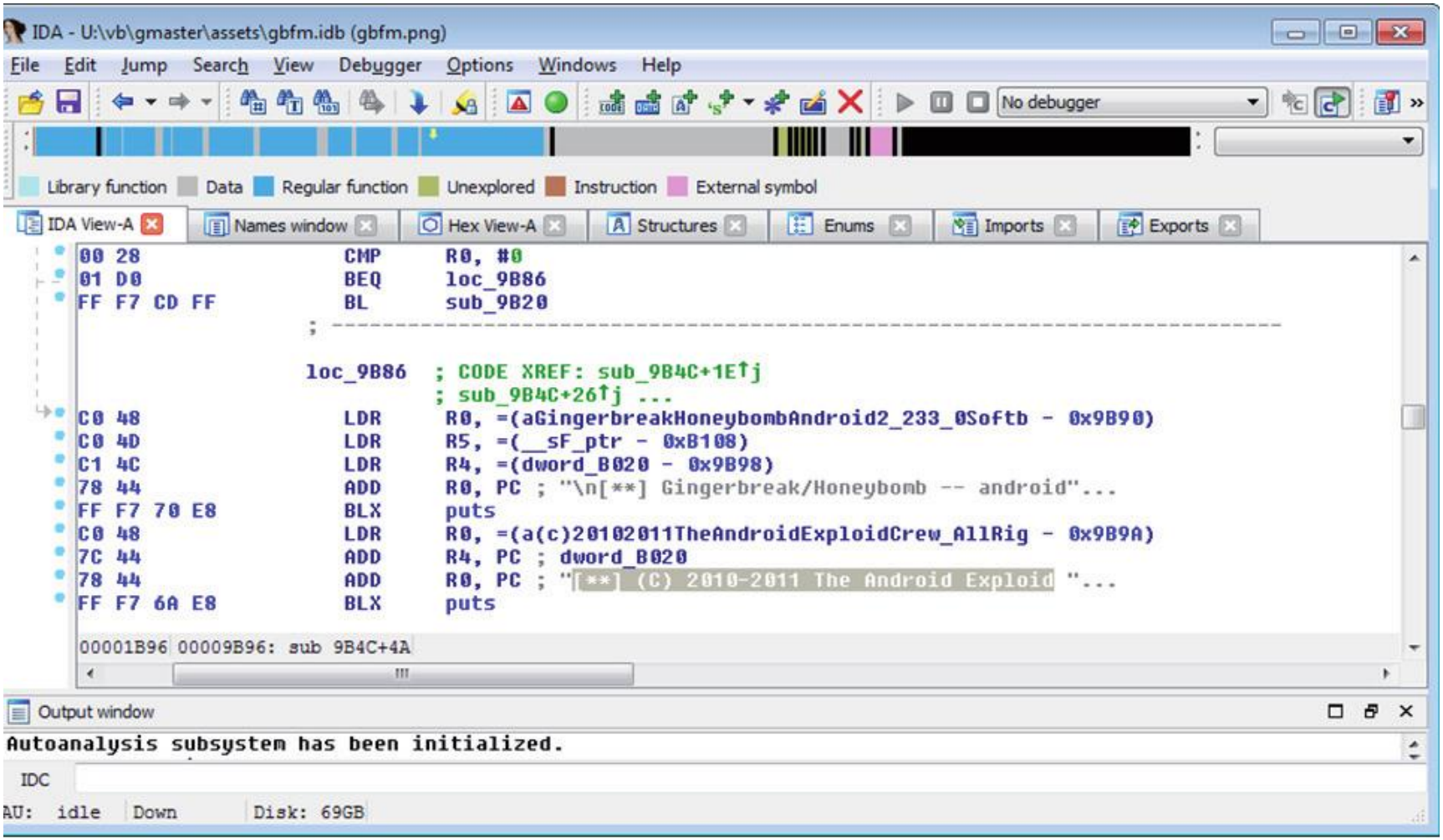
- **CVE-2011-1823**
- **allows local users to execute arbitrary code and gain root privileges via a negative index that bypasses a maximum-only signed integer check on the DirectVolume::handlePartitionAdded method, which triggers memory corruption**
- **Andr/Gmaster-A aka GingerMaster**

Android Gingerbreak root exploit : Inside GMaster APK, exploit code in a picture file.



SOPHOS

Android Gingerbreak root exploit : Inside Gmaster APK, exploit code in a picture file, IDA image



SOPHOS

Exploits overview

Q2 2013

DEX2JAR exploitation : Andr/Obad-A

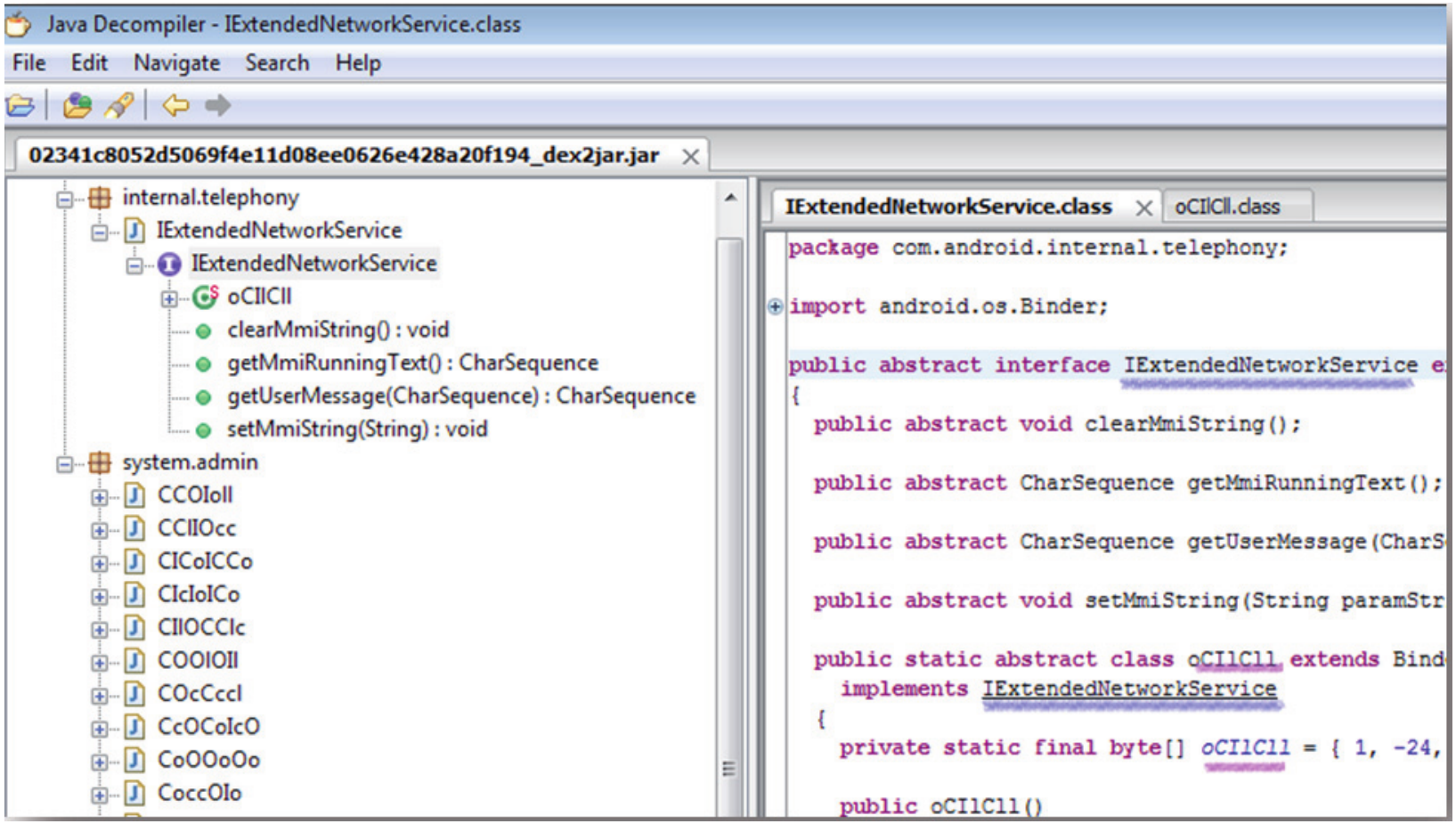
- **disrupts the conversion of Dalvik bytecode into Java bytecode**
- **exploits AndroidManifest.xml processing**
- **gains extended Device Admin privileges**

Andr/Obad-A : classes.dex obfuscation

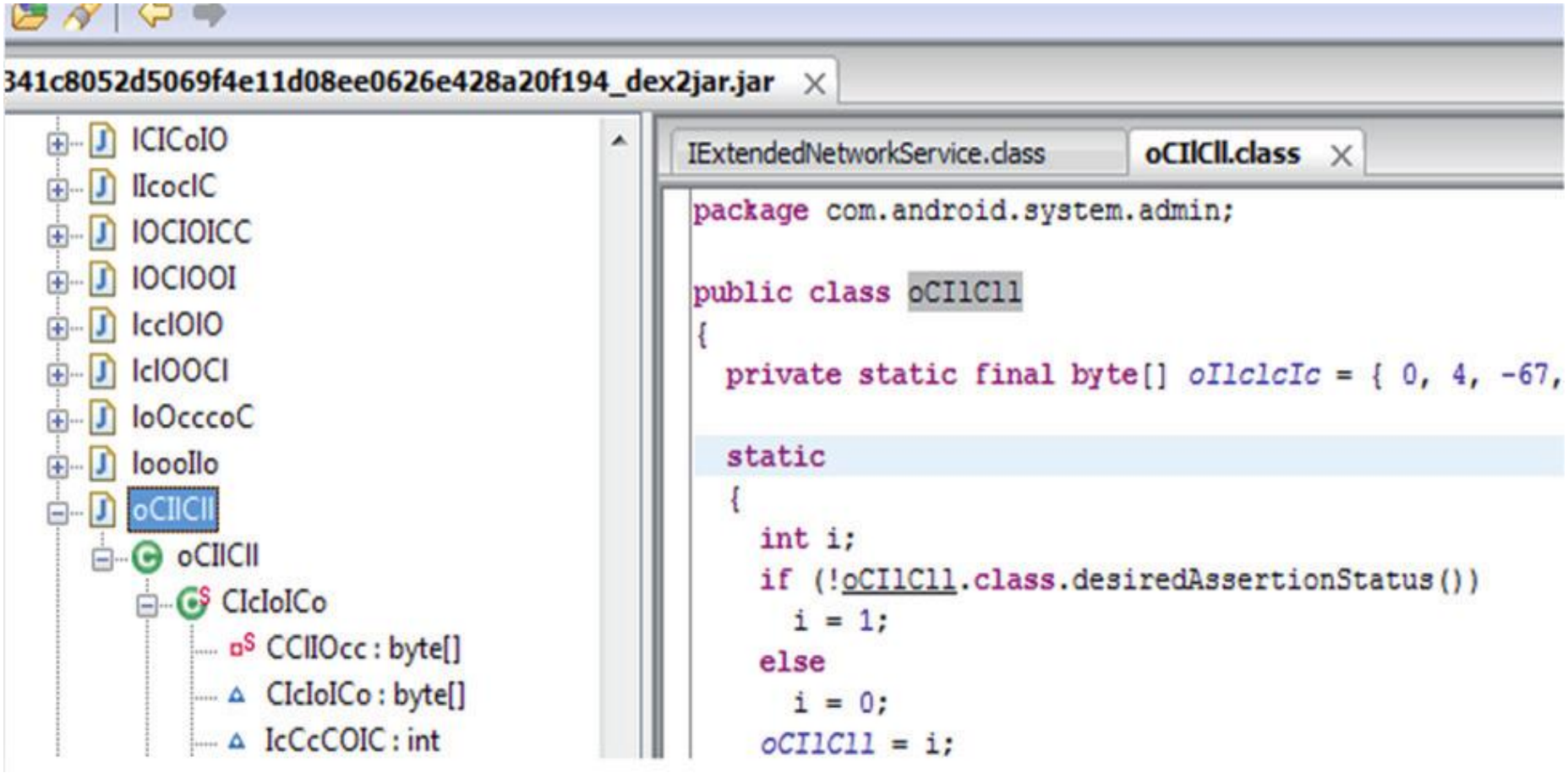
complex code encryption

- all external methods are called via reflection
- all strings are encrypted, including the names of classes and methods
- each class has a local descriptor method which obtains the string required for encryption from the locally updated byte array
- all strings are hidden in this array
- additional stage of decryption for C&C addresses

Obad obfuscation, example of decompiled by dex2jar dex code output

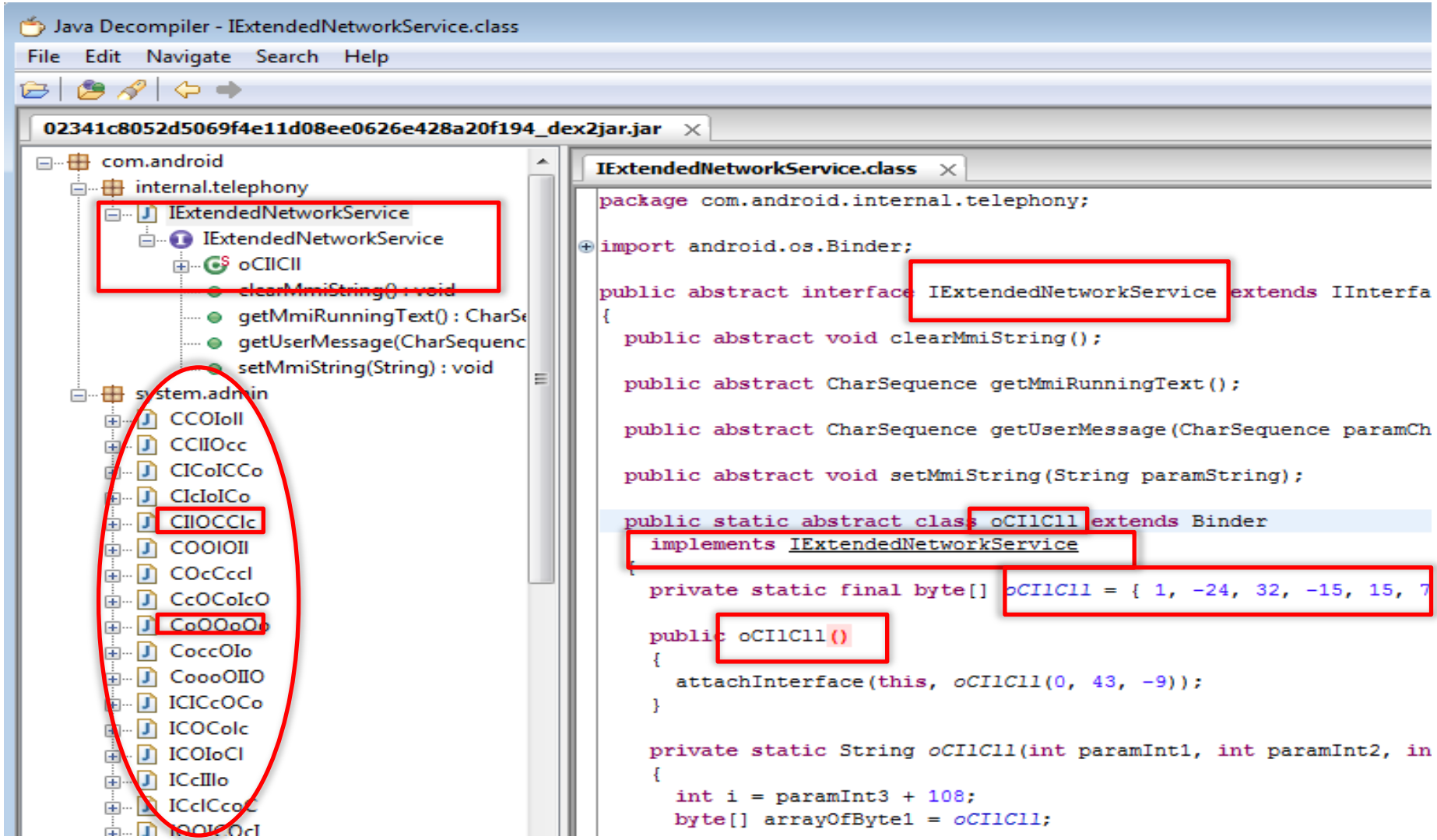


Obad obfuscation, example of decompiled by dex2jar dex code output



SOPHOS

Obad obfuscation, example of decompiled by dex2jar dex code output



SOPHOS

Exploits overview

Q2 2013

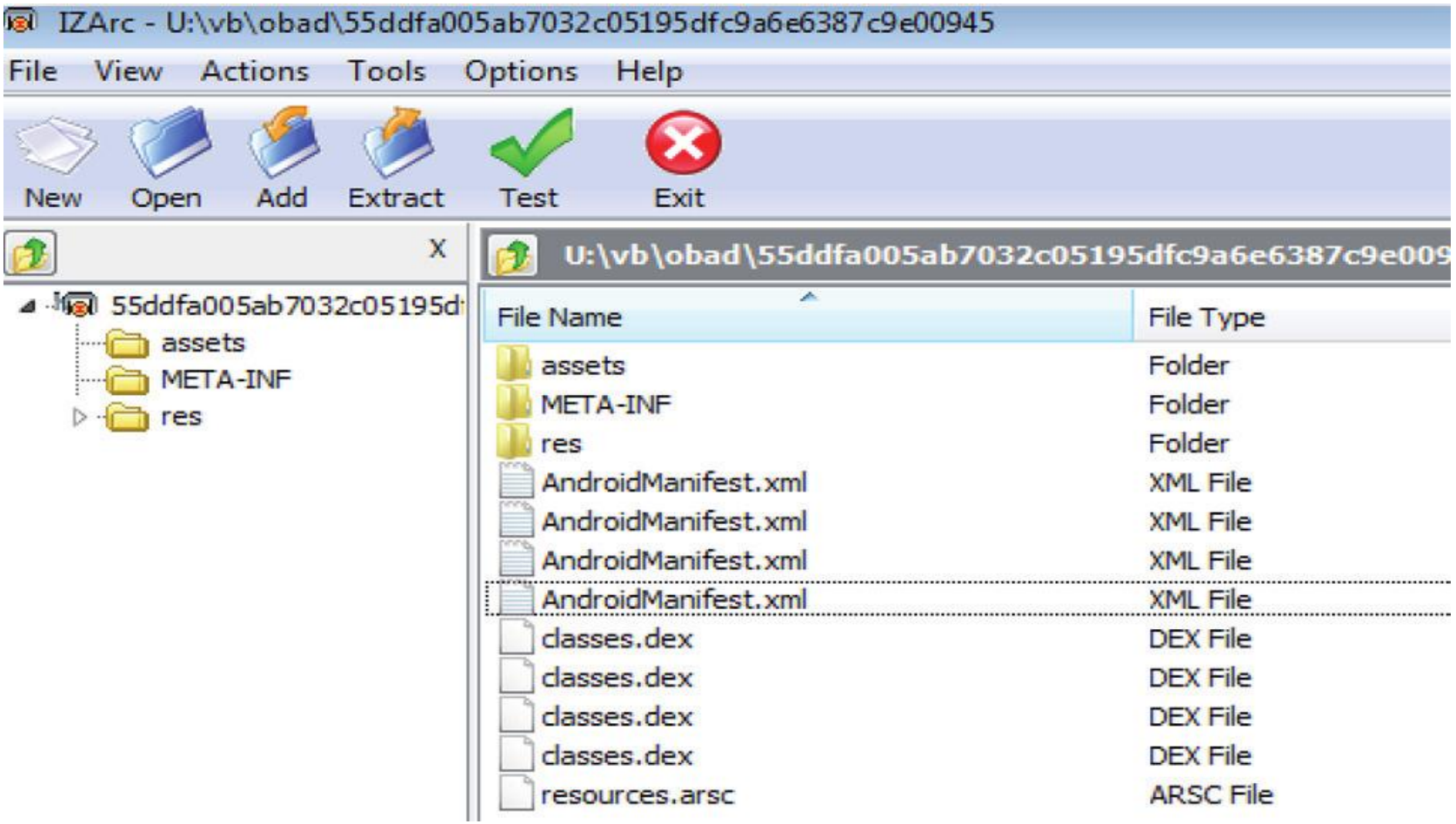
‘Master Key’ vulnerability

- **CVE-2013-4787**
- **reported to affect 99% of devices**
- **apk signature ‘compromised’**
- **allows to replace installed app with the compromised ‘update’**

“Master Key” vulnerability : Andr/MstrKey-A

- exploited apk file has more than one classes.dex and AndroidManifest.xml files
- dodgy manifest file shows permissions that won’t be seen when we look at its actual fingerprint

Inside ‘Master Key’ malware, example of multiple files.



SOPHOS

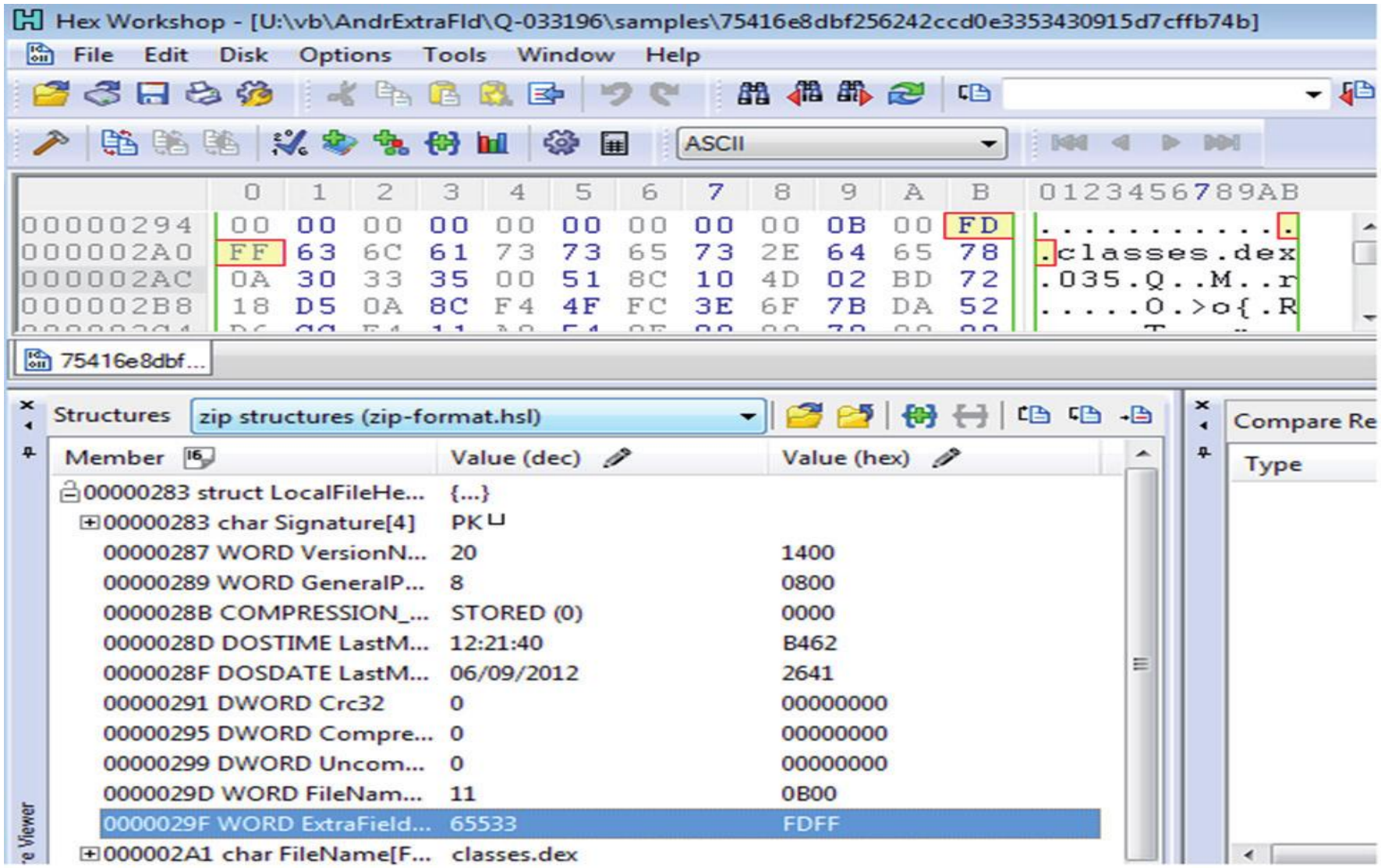
Exploits overview

Q3 2013

‘Extra Field’ vulnerability

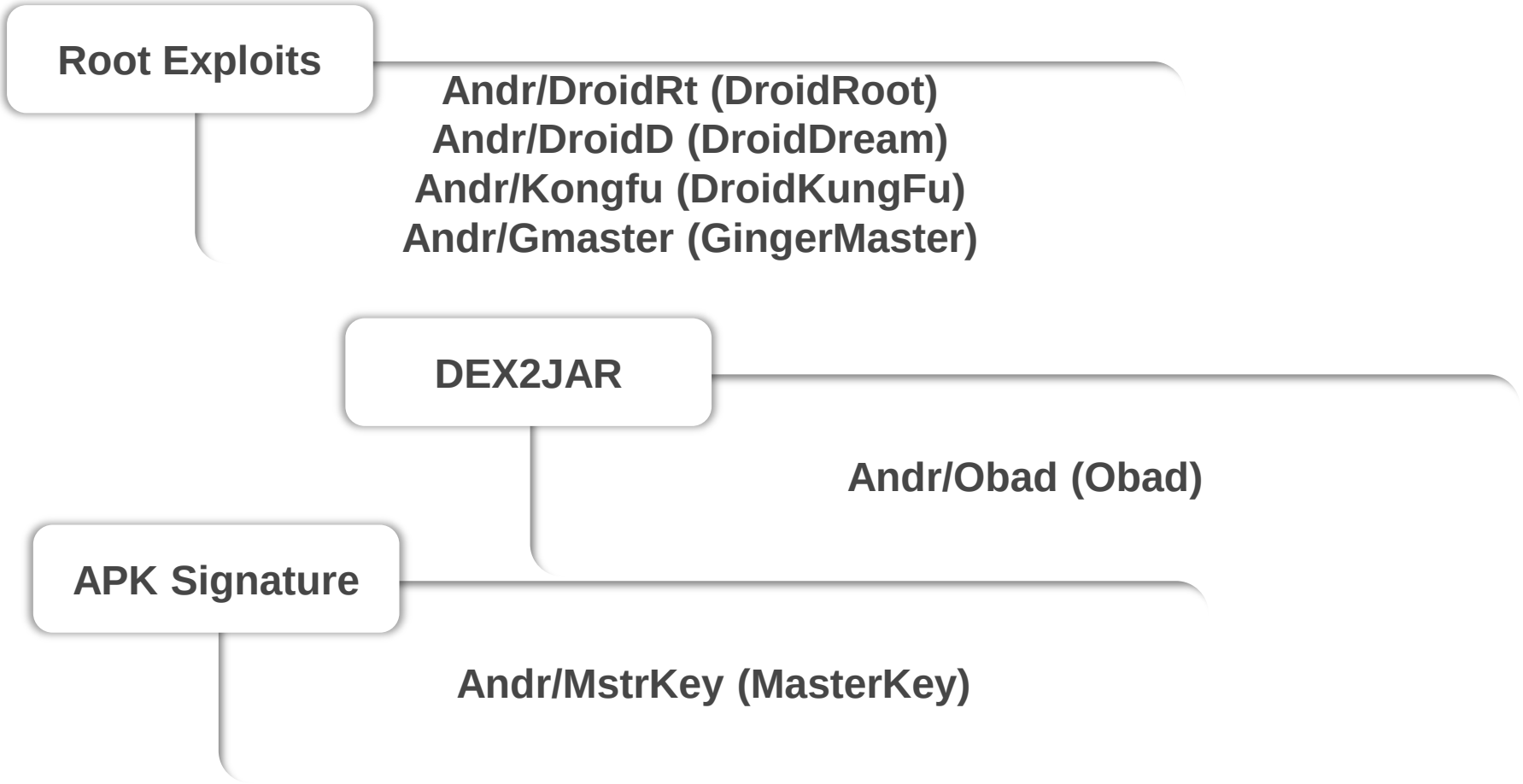
- **allows to bypass code verification**
- **uses an extra field during an APK verification as archive**
- **based on signed-unsigned integer mismatch**
- **exploits fundamentals of the Android APK as a ZIP archive with the some special object fields**

Example of malware exploiting the ‘Extra Field’ vulnerability, showing a changed field



SOPHOS

Milestones in exploiting



SOPHOS

SOPHOS

simple + secure

Evolution of Android Exploits from a Static Analysis Tools Perspective - A Szalay, J Chandraiah

What's Next ?

- **Static Analysis Tools and Techniques**
  - Discuss various Android Static Tools and Techniques
- **Evaluation of Tools against Exploit Samples**
  - Android Master Key Vulnerability
  - Dex Header
  - Unfamiliar Opcodes
  - Decompilation Issues

Why ?

- To verify if tools really meet the requirements
- Highlight issues encountered by analysts
- Identify causes that break various tools
- Suggest best approach for improvement based on the outcome of the research

APK and key elements

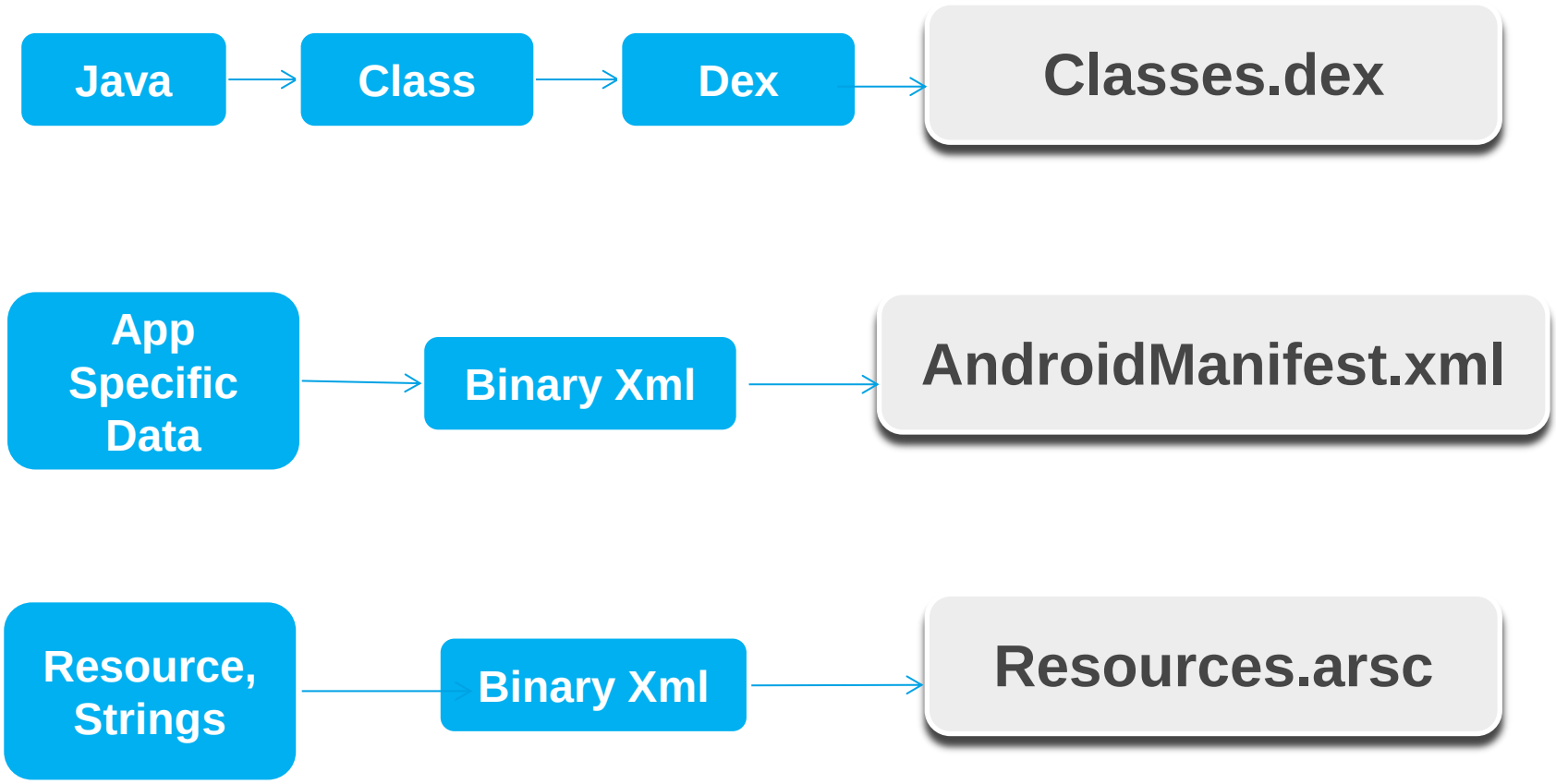
APK

Classes.dex

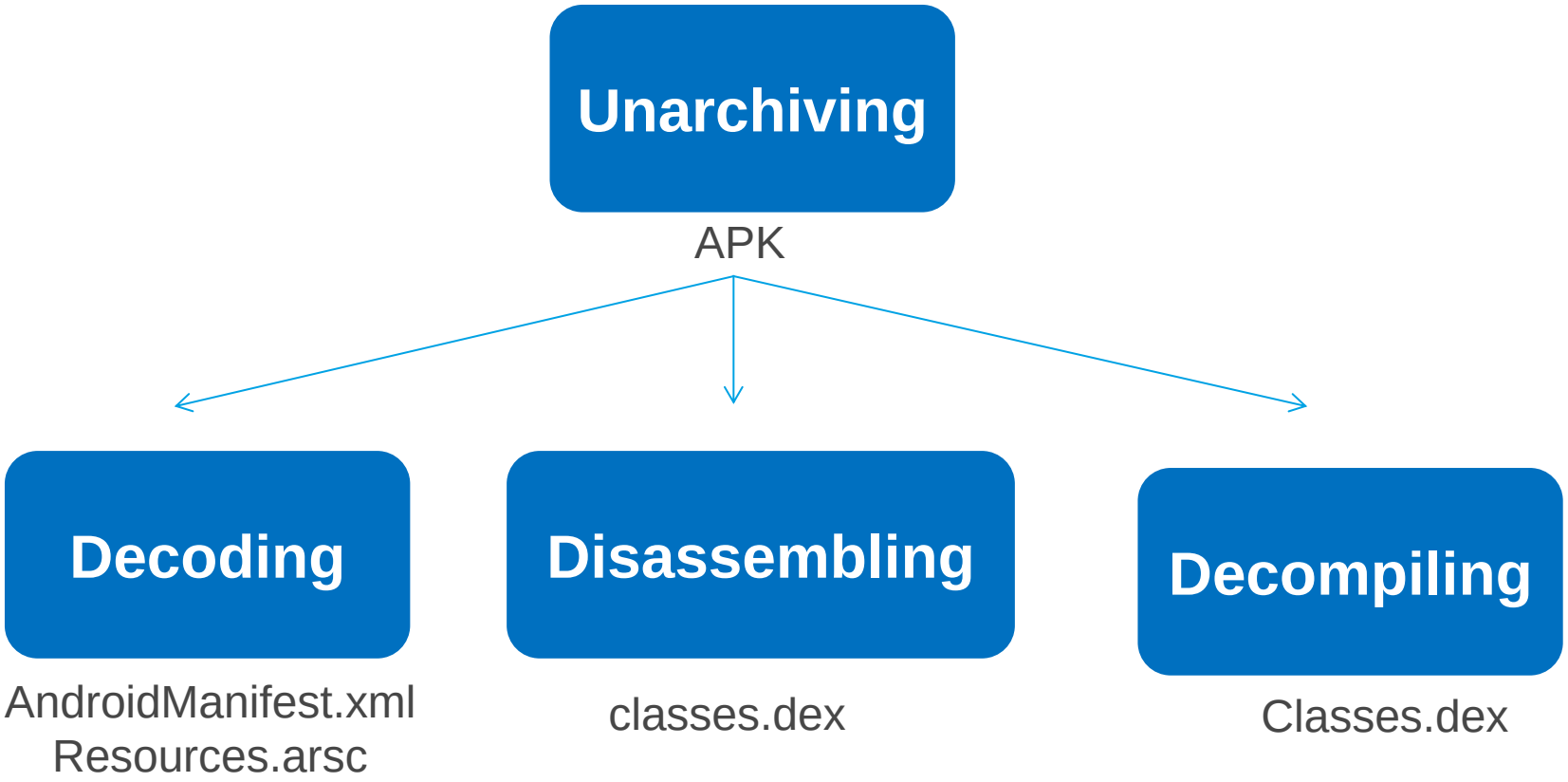
AndroidManifest.xml

Resources.arsc

APK and key elements



Static Analysis Techniques



Static Analysis Tools

- **Unarchiving** – Unzip or any similar archive extraction tools.
- **Decoding** – Apktool, Androguard
- **Decompilation** - Dex2jar, Jdgui, JEB, ded and many other java decompilers.
- **Disassembly** – smali/baksmali, dexdump and IDA pro

Methodology

- Most commonly used analysis technique and Tools used for testing.
- Selected group of popular and relevant Android Exploit samples were used for testing.
  - Andr/MstrKey, Andr/DroidD (DroidDream), Andr/DroidRt, Andr/Obad, Andr/Kongfu (DroidKungfu), Andr/Gmaster (GinMaster)
- Discuss cases that makes analysis difficult
- Also applies to similar non exploit Android samples.

Andr/MstrKey-A- Overwriting?

- Trivial but Important in the case of this Exploit analysis
- Might end up analysing the wrong file

```
inflating: assets/fashion.jxp
inflating: res/layout/main.xml
inflating: AndroidManifest.xml
replace AndroidManifest.xml? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

```
 inflating: res/drawable-hdpi/icon.png
  inflating: res/drawable-ldpi/icon.png
  inflating: res/drawable-mdpi/icon.png
  inflating: classes.dex
replace classes.dex? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

Andr/MstrKey-A- Duplicate files!

- Watch out if you want to use Apktool d <apk>

```
--------  ----------  -----   ----
2320255   21/07/2013  05:31   assets/fashion.jxp
    640   21/07/2013  05:31   res/layout/main.xml
   6780   21/07/2013  05:31   AndroidManifest.xml
   1404   21/07/2013  05:31   AndroidManifest.xml
   1476   21/07/2013  05:31   resources.arsc
   7542   21/07/2013  05:31   res/drawable-hdpi/icon.png
   2760   21/07/2013  05:31   res/drawable-ldpi/icon.png
   4278   21/07/2013  05:31   res/drawable-mdpi/icon.png
 900080   21/07/2013  05:31   classes.dex
   4908   21/07/2013  05:31   classes.dex
    635   21/07/2013  05:31   META-INF/MANIFEST.MF
    688   21/07/2013  05:31   META-INF/CERT.SF
   1007   21/07/2013  05:31   META-INF/CERT.RSA
--------                     -------
```

Andr/DroidD - Dex header

- IDA Pro 6.4adv default installation displayed corrupt message

Andr/DroidD - Dex header

Offset	Description
0x0 – 0x7	DEX_FILE_MAGIC

DEX_FILE_MAGIC = "dex\n<Version number>\0"

- 0x64 0x65 0x78 0x0a 0x30 0x33 0x36 0x00 = “**dex.036**”
- Version ‘036’ – Current version (4.x)
- Version ‘035’ – older api level 13 and earlier , most malware had this version number in our DB.

http://www.strazzere.com/blog/2013/02/loose-documentation-leads-to-easy-disassembler-breakages

Unfamiliar Opcodes

- Familiar technique employed in PC world
- Simple but still effective way
- Lot of research already done and now see them being used
- Packers, malware use to break tools and slow analysis

Unfamiliar Opcodes

- Dex2jar and baksmali v 1.4.1* failed to work

Caused by: java.lang.RuntimeException: **opcode format for 64 not found!**
at com.googlecode.dex2jar.reader.OpcodeFormat.get(OpcodeFormat.java:362)
at
... 8 more

```
unknown opcode encountered - 40. Treating as nop.


UNEXPECTED TOP-LEVEL EXCEPTION:
org.jf.dexlib.Util.ExceptionWithContext: Index: 22789, Size: 269
        at org.jf.dexlib.Util.ExceptionWithContext.withContext(ExceptionWithContext.java:54)
        at org.jf.dexlib.IndexedSection.getItemByIndex(IndexedSection.java:77)
        at org.jf.dexlib.Code.InstructionWithReference.lookupReferencedItem(InstructionWithReference.java:79)
        at org.jf.dexlib.Code.InstructionWithReference.<init>(InstructionWithReference.java:57)
        at org.jf.dexlib.Code.Format.Instruction22c.<init>(Instruction22c.java:59)
        at org.jf.dexlib.Code.Format.Instruction22c.<init>(Instruction22c.java:40)
        at org.jf.dexlib.Code.Format.Instruction22c$Factory.makeInstruction(Instruction22c.java:103)
```

*Works in baksmali version 2.03

Unfamiliar Opcodes

- **40** - unused_40*
- **64** - Reads the char static field identified by the field_id into vx.
- 6400 0200 - sget-byte v0, Test3.bs1:B // field@0002
  Reads byte field@0002 (entry #2 in the field id table) into v0

*src -http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html

Decompilation

- Decompilation is preferred method of analysis.
- Dex2jar followed by jdgui/jad is the most used method.
- Junk insertion, obfuscation , not maintained.

```
}

// ERROR //
public final void uncaughtException(java.lang.Thread paramThread, java.lang.Throwable paramThrowable)
{
  // Byte code:
  //   0: new 80   java/lang/StringBuilder
  //   3: dup
  //   4: bipush 254
```

```
Caused by: java.lang.ArrayIndexOutOfBoundsException: 105
        at
com.googlecode.dex2jar.v3.V3CodeAdapter.visitMoveStmt(V3CodeA
dapter.java:554) at
```

Outcome ?

- Popular tools aren’t enough for complete analysis
- Tools update still catching up with research Implementation
- Support for tools maintenance and development by Android/Google
- IDE for malware analysis ?????
- Better to combine with dynamic Analysis

...Questions…