Unboxing Android

Everything you wanted to know about Android packers

Slava Makkaveev
Avi Bashan
Who Are We?

@Avi

Founder at myDRO, former Mobile R&D Team Leader at Check Point, security researcher at Lacoon Mobile Security.

Experienced in OS Internal research, mobile security, Linux kernel.

@Slava


Vast experience in mobile reverse engineering and Linux internals and malware analysis.
“Boxing” Apps

- Malware authors use various “boxing” techniques to prevent
  - Static Code Analysis
  - Reverse Engineering
- This can be done by proprietary techniques or 3rd party software
- This Includes
  - Code Obfuscation
  - Anti Debugging
  - Anti Tampering
  - Anti Dumper
  - Anti Decompiler
  - Anti Runtime Injection
Maliciousness of Packed Apps

- Malware: 24.3%
- Not Malware: 75.7%

Analyzed 13,000 Apps (July 2017)
Techniques to protect an app's code
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors
Apk Protection Techniques

- **Obfuscators**
- **Packers**
- **Protectors**
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors
Back to Basics!
ART - Android RunTime VM

Provided an Ahead of Time (AOT) compilation approach

- Pre-compilation at install time
  - installation takes more time
  - more internal storage is required

- OAT vs JIT
  - Reduces startup time of applications
  - Improves battery performance
  - Uses less RAM
DEX Loading Process

- App contains minimum one DEX file
- App can load other DEX files during execution
- Each DEX file will be compiled in OAT file
- Android Runtime executes OAT files
- Android Runtime checks DEX files checksum

Zygote process \(\xrightarrow{\text{fork()}}\) App process

\[
\text{classes.dex} \xrightarrow{\text{dex2oat}} \text{OAT version of classes.dex}
\]
OAT - Ahead of Time File

OAT is ELF

- Three special symbols in dynamic section
  - oatdata
  - oatexec
  - aotlastword
- Original DEX file is contained in the oatdata section
- Compiled native instructions are contained in the oatexec section
How to unpack?
Possible Approaches to Unpack an Android App

- Find the algorithm
- Extract DEX from compiled OAT
- Dump DEX from memory
- Runtime environment modification
Notable Previous Work

- **Android Hacker Protection Level 0**
  - Tim Strazzere and Jon Sawyer
  - DEFCON 22, 2014
  - Released a set of unpacking scripts

- **The Terminator to Android Hardening Services**
  - Yueqian Zhang, Xiapu Luo, Haoyang Yin
  - HITCON, 2015
  - Released DexHunter - modified version of Android Dalvik/ART VM
Our Approach
Goals

● What did want
  ○ Find a solution that
    ■ Require minimal changes to Android
    ■ Will work on most of the packers

● How did we do it?
  ○ Reversed most popular packers
  ○ Patched few code rows of Android Runtime
Goals

● What did want
  ○ Find a solution that
    ■ Require minimal changes to Android
    ■ Will work on most of the packers

● How did we do it?
  ○ Reversed most popular packers
  ○ Patched few code rows of Android Runtime

PROFIT
Most popular packers encountered

- Baidu
- Bangcle
- Tencent
- Ali
- 360 Jiagu
- ... (and a few more)
Abstract Packer Model

Load protected DEX
Find a class

Loader DEX

libart.so
Open DEX file
Map data

libc.so
open
read
mmap
...

Open DEX
Read data

Original DEX
Abstract Packer Model

Loader DEX
- Load protected DEX
- Find a class

libart.so
- Open DEX file
- Map data

<protector>.so
- Load native part
- Hook calls

libc.so
- Decrypt DEX
- open
- read
- mmap
- ...

Original DEX
- Read data
Bangcle - Classification

Classes
- ApplicationWrapper
- FirstApplication
- MyClassLoader
- ACall

Files
- libsecse
- libsecmain
- libsecexe
- libsecpreload
- bangcle_classes (original dex)
System.load("/data/data/" + getPackageName() + "/.cache/libsecexe.so");
Acall.getACall().a1(...);
Acall.getACall().r1(...);
Acall.getACall().r2(...);
...

class MyClassLoader extends DexClassLoader {

    ...
}

cl = new MyClassLoader("/data/data/" + getPackageName() + "/.cache/classes.jar", ...);
realApplication = cl.loadClass(v0).newInstance();
public class ACa11 {
    public native void a1(byte[] arg1, byte[] arg2);
    public native void at1(Application arg1, Context arg2);
    public native void at2(Application arg1, Context arg2);
    public native void c1(Object arg1, Object arg2);
    public native void c2(Object arg1, Object arg2);
    public native void c3(Object arg1, Object arg2);
    public native void c4(Object arg1, Object arg2);
    public native void c5(Object arg1, Object arg2);
    public native void c6(Object arg1, Object arg2);
    public native void c7(Object arg1, Object arg2);
    public native void c8(Object arg1, Object arg2);
    public native void r1(byte[] arg1, byte[] arg2);
    public native void r2(byte[] arg1, byte[] arg2, byte[] arg3);
    public native ClassLoader rc1(Context arg1);
    public native void s1(Activity arg1, Object arg2, Object arg3);
    public native void s2(Activity arg1, ClassLoader arg2);
    public native void s3(Activity arg1, Object arg2);
    public native void s4();
    public native void s5(ContentProvider arg1);
    public native void s6();
    public native void s7();
    public native void s8();
}
### Bangcle - libsecexe.so

**Class:** ELF32  
**Type:** DYN (Shared object file)  
**Machine:** ARM  
**Entry point address:** 0x433c  
**Start of program headers:** 52 (bytes into file)  
**Start of section headers:** 92204 (bytes into file)  
**Size of program headers:** 32 (bytes)  
**Number of program headers:** 6  
**Size of section headers:** 0 (bytes)  
**Number of section headers:** 0

#### Dynamic section:

<table>
<thead>
<tr>
<th>Offset</th>
<th>VirtAddr</th>
<th>PhysAddr</th>
<th>FileSiz</th>
<th>MemSiz</th>
<th>Flg</th>
<th>Align</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000000c</td>
<td>0x00028584</td>
<td>0x00028584</td>
<td>0x00568</td>
<td>0x00568</td>
<td>R</td>
<td>0x4</td>
</tr>
<tr>
<td>0x018c1c</td>
<td>0x00030c1c</td>
<td>0x00030c1c</td>
<td>0x00520</td>
<td>0x01538</td>
<td>RW</td>
<td>0x8000</td>
</tr>
<tr>
<td>0x018e80</td>
<td>0x00030c80</td>
<td>0x00030c80</td>
<td>0x00108</td>
<td>0x00108</td>
<td>RW</td>
<td>0x4</td>
</tr>
<tr>
<td>0x000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000</td>
<td>0x00000</td>
<td>RW</td>
<td>0x4</td>
</tr>
<tr>
<td>0x018e1c</td>
<td>0x00030c1c</td>
<td>0x00030c1c</td>
<td>0x003e4</td>
<td>0x003e4</td>
<td>R</td>
<td>0x1</td>
</tr>
</tbody>
</table>

- **Entry address points to compressed code (anti-debugging)**  
- **Start of section table is out of file bounders**  
- **No section table (anti-debugging)**  
- **Exception Index Table is out of file bounders (IDA crash)**

#### Program headers:

- **Init**  
  - 0x0000000c (INIT)  
  - 0x00000019 (INIT_ARRAY)  

- **Init Array**  
  - 0x125A9  
  - 0x30C1C

- **Real entry point**
Bangcle - libsecexe.so

Copy code sections to an allocated buffer. Decompress 0x247b0 bytes to 0x433c

Registration com.secneo.guard.ACall
native methods: a1, r1, r2, ...

0x433c
0x4638
0x4938
0xde38
0xe050
JNI_OnLoad

0x12590
0x125a9
0x13150
0x28aec

Compressed code

TEXT (init code)
Bangcle - Processes

Function a1

Extract ELF /data/data/<pkg>/.cache/<pkg> from apk (Assets)

fork app process
  execl /data/data/<pkg>/.cache/<pkg> <pkg> -1114751212 1 /data/app/<pkg>/base.apk 34 <pkg> 43 44 0

fork pkg process (from libsecmain.so::so_main)
  anti-debugging thread

fork pkg process if .cache/classes.dex (OAT) does not exist
  LD_PRELOAD=/data/data/<pkg>/.cache/libsecpreload.so
  LD_PRELOAD_ARGS=<pkg> 9 13
  LD_PRELOAD_SECSO=/data/data/<pkg>/.cache/libsecmain.so
  execl /system/bin/dex2oat
    –oat-location=/data/data/<pkg>/.cache/classes.dex –instruction-set=arm

Function r2

u0_a76 28644 5019 1531220 49108 ffffffff b6eb6d44 S <pkg name>
u0_a76 28881 28644 3516 768 ffffffff b6eb3504 S <pkg name>
u0_a76 28882 28881 2464 624 ffffffff b6eb3504 S <pkg name>
Bangcle - libc.so hook

Function r1

Protection was changed
Bangcle - Summary

- Creates a stub in Java activity to load native library.
- Native library is protected with different anti research techniques.
- Native library hooks libc for handling the opening of the OAT file.
Baidu - Classification

Classes
- StubApplication
- StubProvider

Files
- libbaiduprotect
- baiduprotect1 (original dex)
public class A implements Enumeration {
    public static native byte B(int arg0, Object arg1, ...);
    public static native char C(int arg0, Object arg1, ...);
    public static native double D(int arg0, Object arg1, ...);
    public static native float F(int arg0, Object arg1, ...);
    public static native int I(int arg0, Object arg1, ...);
    public static native long J(int arg0, Object arg1, ...);
    public static native Object L(int arg0, Object arg1, ...);
    public static native short S(int arg0, Object arg1, ...);
    public static native void V(int arg0, Object arg1, ...);
    public static native boolean Z(int arg0, Object arg1, ...);
    public static native void a();
    public static native void b();
    public static native String[] c();
}

<table>
<thead>
<tr>
<th>Func</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0x23459</td>
</tr>
<tr>
<td>b</td>
<td>0x2345d</td>
</tr>
<tr>
<td>c</td>
<td>0x23461</td>
</tr>
<tr>
<td>V, Z, B, C, S, I, J, F, D, L</td>
<td>0x25861</td>
</tr>
</tbody>
</table>
Baidu - libbaiduprotect.so

Change self protection
0x0 - 0x1000
Remove ELF header

0x1000
0x2e6d
0x2ea4
0x23459
0x2345d
0x23461
0x25861
0x3ca78
0x4286c

JNI_OnLoad

0x2000 - 0x3d000
Decrypt code
0x2e6d - 0x3ca78

Change self protection

0x23459 a
0x2345d b
0x23461 c
0x25861 V, Z, B, C, S, I, J, F, D, L
0x3ca78
0x4286c

TEXT (Entry point 1)
Baidu - JNI_OnLoad

Anti-debugging

Registration of native methods: a, b, c, ...

Extract packed DEX /Assets/baiduprotect1.jar to /data/data/<pkg>/1/1.jar
Create empty DEX file /data/data/<pkg>/1/classes.jar

Hook libart.so

Create DexClassLoader(/data/data/<pkg>/1/classes.jar) + Merge with main class loader by extending BaseDexClassLoader::pathList::dexElements
Baidu - Anti-debugging

- Obfuscation
- Logs disabling
- For each `/proc/` check that `/proc/<pid>/cmdline` does not contain `gdb`, `gdbserver`, `android_server`
- For each `/proc/self/task` check that `/proc/self/task/<pid>/status` does not contain `TracerPid`
- For each `/proc/self/task` check that `/proc/self/task/<pid>/comm` does not contain `JDWP`
- Check `android.os.Debug.isDebuggerConnected`
- `select` call (timer) based technique
- `inotify` watch (IN_ACCESS + IN_OPEN) of
  - `/proc/self/mem`
  - `/proc/self/pagemap`
  - For each `/proc/self/task`
    - `/proc/self/task/<pid>/mem`
    - `/proc/self/task/<pid>/pagemap`
Baidu - libart.so hook

Function __android_log_print

No logs

Function execv

dex2oat hook:
Add environment variable ANDROID_LOG_TAGS=*:f
Prevent code compilation: add --compiler-filter=verify-none command line parameter

Function open
Decrypt /data/data/<pkg>/1.1.jar in case of /data/data/<pkg>/1.1/classes.jar file loading

<table>
<thead>
<tr>
<th>libc func</th>
<th>Libart hook</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>0x309BC8</td>
<td>0xA75C</td>
</tr>
<tr>
<td>open</td>
<td>0x309BDC</td>
<td>0x8FAC</td>
</tr>
<tr>
<td>close</td>
<td>0x309BE4</td>
<td>0x9168</td>
</tr>
<tr>
<td>mmap</td>
<td>0x309BE8</td>
<td>0x9474</td>
</tr>
<tr>
<td>strstr</td>
<td>0x309C58</td>
<td>0x8BD8</td>
</tr>
<tr>
<td>fork</td>
<td>0x309F3C</td>
<td>0x92DC</td>
</tr>
<tr>
<td>waitpid</td>
<td>0x309F40</td>
<td>0xA5E4</td>
</tr>
<tr>
<td>execv</td>
<td>0x309F4C</td>
<td>0xA324</td>
</tr>
<tr>
<td>__android_log_print</td>
<td>0x309FAC</td>
<td>0xA750</td>
</tr>
</tbody>
</table>
Baidu - Summary

- Creates a stub in Java activity to load native library.
- Native library is protected with different anti research techniques.
- Native library hooks libc for handling the opening of the DEX file.
I'VE ALREADY SEEN THAT!!!
libc::open == decryption

Bangle

Filter by file path:
/data/data/<pkg>/.cache/classes.dex

Expect to see:
OAT

Baidu

/data/data/<pkg>/.1/classes.jar

Expect to see:
DEX
Using the DEX Loading Process to Unpack Apps

Where is first call of DEX/OAT file opening?

OAT

dalvik.system.DexClassLoader::DexClassLoader
dalvik.system.DexFile::DexFile
DexFile::openDexFileNative

DEX

DexFile_openDexFileNative
ClassLinker::OpenDexFilesFromOat
OatFileAssistant::MakeUpToDate
OatFileAssistant::OatFileIsUpToDate

OatFileAssistant::GetOatFile
OatFile::Open
OatFile::OpenElfFile → DexFile::DexFile

OatFileAssistant::GivenOatFileIsUpToDate
OatFileAssistant::GetRequiredDexChecksum
DexFile::GetChecksum
OpenAndReadMagic
platform/art/runtime/dex_file.cc patch

```cpp
static int OpenAndReadMagic(const char* filename, uint32_t* magic, std::string* error_msg) {
  CHECK(magic != nullptr);
  ScopedFd fd(open(filename, O_RDONLY, 0));
  ...
  char* fn_out = new char[PATH_MAX];
  strcpy(fn_out, filename);
  strcat(fn_out, "__unpacked");
  int fd_out = open(fn_out, O_WRONLY|O_CREAT|O_EXCL, S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH);
  struct stat st;
  if (!fstat(fd.get(), &st)) {
    char* addr = (char*)mmap(NULL, st.st_size, PROT_READ, MAP_PRIVATE, fd.get(), 0);
    write(fd_out, addr, st.st_size);
    munmap(addr, st.st_size);
  }
  close(fd_out);
  delete fn_out;
  ...
}

DexFile::DexFile(const uint8_t* base, size_t size, const std::string& location, uint32_t location_checksum, MemMap* mem_map, const OatDexFile* oat_dex_file) :
  begin_(base),
  size_(size),
  ...
{
  ...
  std::ofstream dst(location + "__unpacked", std::ios::binary);
  dst.write(reinterpret_cast<const char*>(base), size);
  dst.close();
  ...
}
```

DEX

OAT
Demo Time!
Tool can be found at -
github.com/CheckPointSW/android_unpacker
Questions?