Unboxing Android

Everything you wanted to know about Android packers

Slava Makkaveev
Avi Bashan
Who Are We?

@Avi

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

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

@Slava

Senior Security Researcher at Check Point, former Security Researcher at Verint.

Vast experience in mobile OS research and linux internals.
“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

Analyzed 13,000 Apps (July 2017)

Malware: 24.3%
Not Malware: 75.7%
Techniques to protect an app’s code
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors
Apk Protection Techniques

- **Obfuscators**
- **Packers**
- **Protectors**

```java
pm.getClass().getMethod("getPackageSizeInfo", String.class,
Class.forName("android.content.pm/IPackageStatsObserver")).invoke(pm, packInfo.packageName,
new IPackageStatsObserver.Stub() {
    public void onGetStatsCompleted(PackageStats pStats, boolean succeeded) {
    }
});
```

```java
v6.getClass().getMethod("getPackageSizeInfo", String.class,
Class.forName("android.a.a.a"),).invoke(v6, (PackageInfo)v0_5.packageName,
new a() {
    public void a(PackageStats arg3, boolean arg4) {
    }
});
```
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors

Diagram:

- Original DEX
- Packing process
- Encrypted DEX
- Packer Loader
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors
Apk Protection Techniques

- Obfuscators
- Packers
- Protectors

Diagram:

1. Original DEX
2. Protection Process
3. Encrypted Modified DEX

Diagram shows an APK before and after the protection process.
Apk Protection Techniques

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

Provided an Ahead of Time (AOT) compilation approach

### DEX to OAT

- **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

1. Zygote process
2. fork()
3. App process
4. classes.dex
5. load app code
6. dex2oat
7. 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
Android Java Native Interface (JNI)

- Allows calling native code directly from JVM.
- Execution path starts from System.loadLibrary
- Used by some of the popular packers for the packing logic.
- Packer library is called after activity is started
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, 2015
  - 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
Analyzed Packers

Most popular packers encountered

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

Load protected DEX
Find a class

libart.so

Open DEX file
Map data

libc.so
open
read
mmap
...

Loader DEX
Abstract Packer Model

1. Loader DEX
   - Load protected DEX
   - Find a class

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

3. libart.so
   - Open DEX file
   - Map data

4. libc.so
   - Decrypt DEX
   - Open
   - Read
   - mmap
   - ... 

5. Protected DEX
   - Read original data

Bangcle - Classification

Classes
- ApplicationWrapper
- FirstApplication
- MyClassLoader
- ACall

Files
- libsecse
- libsecmain
- libsecexe
- libsecpreload
- bangcle_classes (original dex)
Bangcle - Java Loader Implementation

assets/libsecexe.so → /data/data/<pkg>/.cache/libsecexe.so
assets/libsecmain.so → /data/data/<pkg>/.cache/libsecmain.so
assets/libsecpreload.so → /data/data/<pkg>/.cache/libsecpreload.so
assets/bangcle_classes.jar → /data/data/<pkg>/.cache/classes.jar

```java
System.load("/data/data/" + getPackageName() + "/.cache/libsecexe.so");
Acall.getACall().a1(...);
Acall.getACall().r1(...);
Acall.getACall().r2(...);
...

public class MyClassLoader extends DexClassLoader {
    ...
}

cl = new MyClassLoader("/data/data/" + getPackageName() + "/.cache/classes.jar", ...);
realApplication = cl.loadClass(v0).newInstance();
```
public class ACall {
    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);
}

BDL Interface

Java Interface

Native Functions

Mapping

<table>
<thead>
<tr>
<th>Func</th>
<th>Offset</th>
<th>Func</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>0x4638</td>
<td>set1</td>
<td>0xCFFC</td>
</tr>
<tr>
<td>at1</td>
<td>0x8A44</td>
<td>set2</td>
<td>0x9BC8</td>
</tr>
<tr>
<td>at2</td>
<td>0x9184</td>
<td>set3</td>
<td>0x566C</td>
</tr>
<tr>
<td>c1</td>
<td>0xF984</td>
<td>set3</td>
<td>0x8CE8</td>
</tr>
<tr>
<td>c2</td>
<td>0x103E8</td>
<td>set4</td>
<td>0x63B4</td>
</tr>
<tr>
<td>c3</td>
<td>0x12E48</td>
<td>set5</td>
<td>0x4AA0</td>
</tr>
<tr>
<td>r1</td>
<td>0x4938</td>
<td>set8</td>
<td>0x16828</td>
</tr>
<tr>
<td>r2</td>
<td>0xDE38</td>
<td>s1</td>
<td>0x126B4</td>
</tr>
<tr>
<td>jniCheckRawDexAvailable</td>
<td>0x4408</td>
<td>rc1</td>
<td>0xBFE4</td>
</tr>
<tr>
<td>jniGetRawDexAvailable</td>
<td>0x44A0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

Type Offset VirtAddr PhysAddr FileSiz MemSiz Flg Align
EXIDX 0x028584 0x00028584 0x00028584 0x000568 0x000568 R 0x4
LOAD 0x000000 0x00000000 0x00000000 0x0131ec 0x0131ec RE 0x8000
LOAD 0x018c1c 0x00030c1c 0x00030c1c 0x000520 0x010538 RW 0x8000
DYNAMIC 0x018c80 0x00030c80 0x00030c80 0x00108 0x00108 RW 0x4
GNU_STACK 0x000000 0x00000000 0x00000000 0x000000 0x000000 RW 0x4
GNU_RELRO 0x018c1c 0x00030c1c 0x00030c1c 0x0003e4 0x0003e4 R 0x1

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:

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, ...

Compressed 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
    –zip-fd=9 –zip-location=/data/data/<pkg>/cache/classes.jar –oat-fd=13
    –oat-location=/data/data/<pkg>/.cache/classes.dex –instruction-set=arm

Function r2

- u0_a76 28644 5019 1531220 49108 ffffffff b6e6b6d4 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

<table>
<thead>
<tr>
<th>libc func</th>
<th>Offset</th>
<th>libc func</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>munmap</td>
<td>0x15BD8</td>
<td>close</td>
<td>0x14FAC</td>
</tr>
<tr>
<td>msync</td>
<td>0x15F88</td>
<td>__openat</td>
<td>0x14DA4</td>
</tr>
<tr>
<td>read</td>
<td>0x15118</td>
<td>pread64</td>
<td>0x162F8</td>
</tr>
<tr>
<td>__mmap2</td>
<td>0x15420</td>
<td>pwrite64</td>
<td>0x166DC</td>
</tr>
<tr>
<td>__open</td>
<td>0x14B9C</td>
<td>write</td>
<td>0x152FC</td>
</tr>
</tbody>
</table>

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();
}
Baidu - libbaiduprotect.so

- Change self protection 0x0 - 0x1000
- Remove ELF header
- Decrypt code 0x0x2e6d - 0x3ca78
- Change self protection 0x2000 - 0x3d000

- TEXT (Entry point 1)
  - V, Z, B, C, S, I, J, F, D, L
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`
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>/classes.jar file loading
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.
NOT SURE IF REPOST
OR JUST DEJA VU
libc::open == decryption

Bangle

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

Expect to see:
OAT

Baidu

Filter by file path:
/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?

**DEX**
- `dalvik.system.DexClassLoader::DexClassLoader`
- `dalvik.system.DexFile::DexFile`
- `DexFile::openDexFileNative`

**OAT**
- `DexFile_openDexFileNative`
- `ClassLinker::OpenDexFilesFromOat`
- `OatFileAssistant::MakeUpToDate`
- `OatFileAssistant::OatFileIsUpToDate`

**Additional Calls**
- `OatFileAssistant::GivenOatFileIsUpToDate`
- `OatFileAssistant::GetRequiredDexChecksum`
- `DexFile::GetChecksum`
- `OpenAndReadMagic`

**OatFile Assistant**
- `OatFile::Open`
- `OatFile::OpenElfFile → DexFile::DexFile`
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();

  ...
}
Demo Time!
Unpacking modification instructions to AOSP can be found @checkpoint github repo
Questions?