Unboxing Android

Everything you wanted to know about Android packers

Slava Makkaveev
Avi Bashan

Check Point
SOFTWARE TECHNOLOGIES LTD.

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

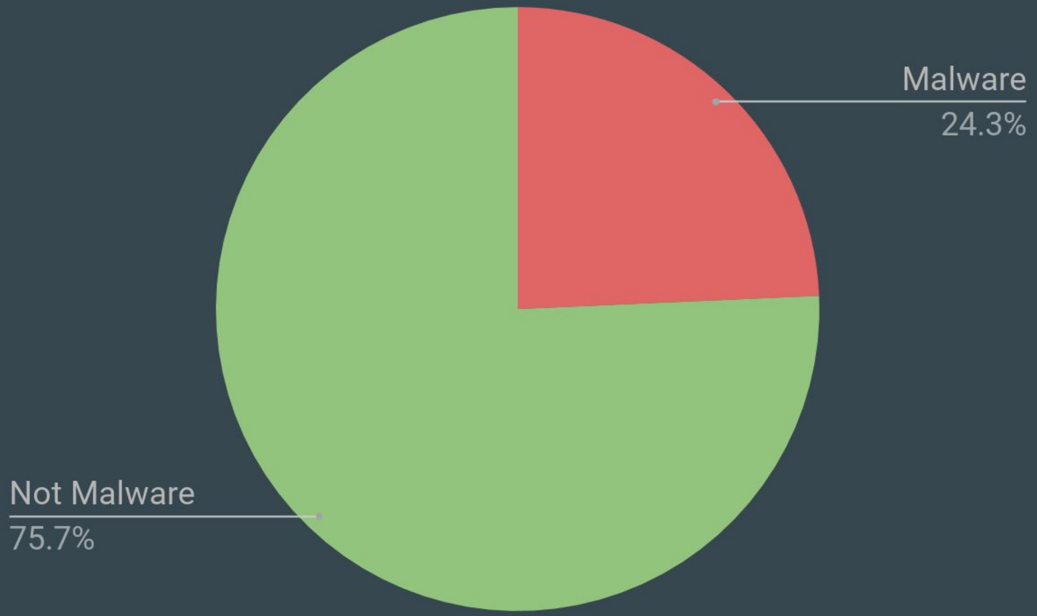
Senior Security Researcher at Check Point, former Security Researcher at Verint. Holds a Phd in Computer Science.

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 Protection
  - Anti Debugging
  - Anti Tampering
  - Anti Dumper
  - Anti Decompiler
  - Anti Runtime Injection

Maliciousness of Packed Apps



Analyzed 13,000 Apps (May 2017)

Techniques to protect an app’s code

Apk Protection Techniques

Obfuscators

Packers

Protectors

Apk Protection Techniques

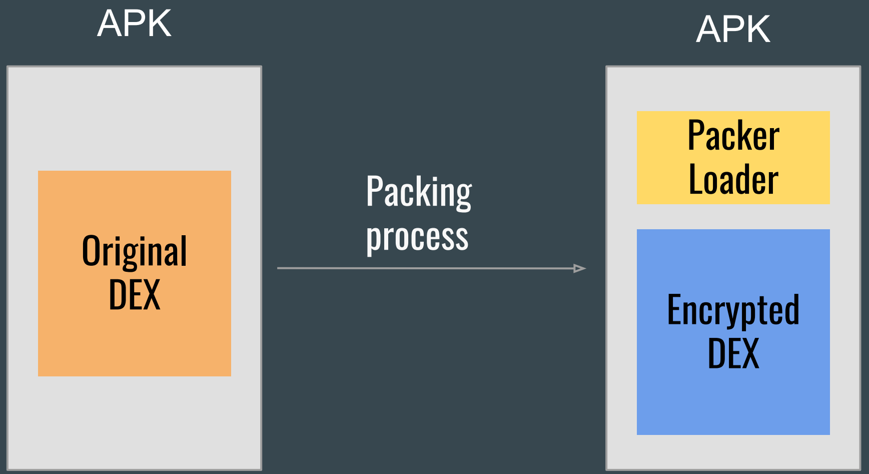
- **Obfuscators**
- Packers
- Protectors

```
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) {
        }
    });
```

```
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) {
        }
    });
```

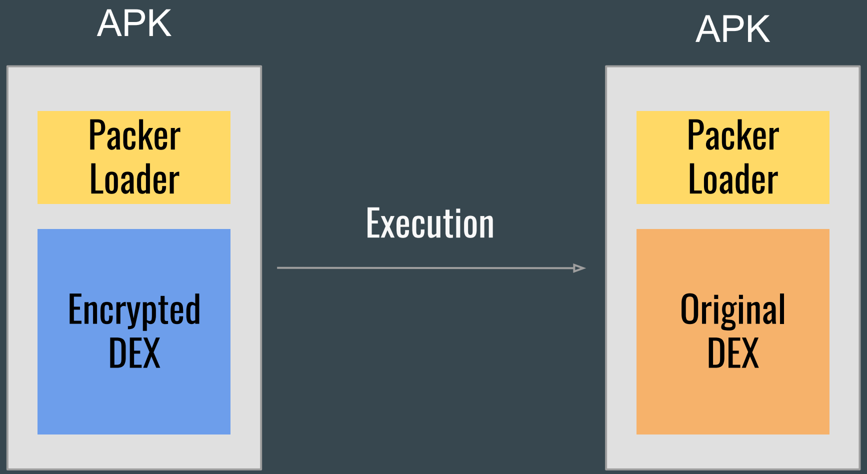
Apk Protection Techniques

- Obfuscators
- **Packers**
- Protectors



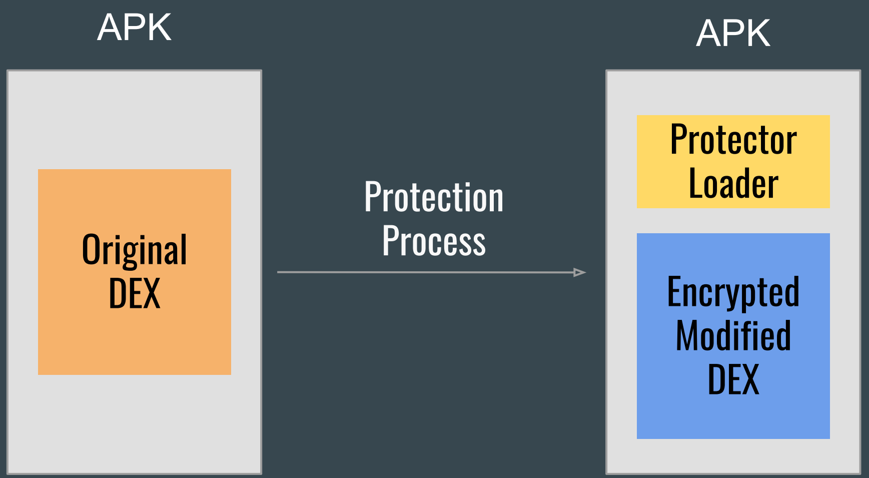
Apk Protection Techniques

- Obfuscators
- **Packers**
- Protectors



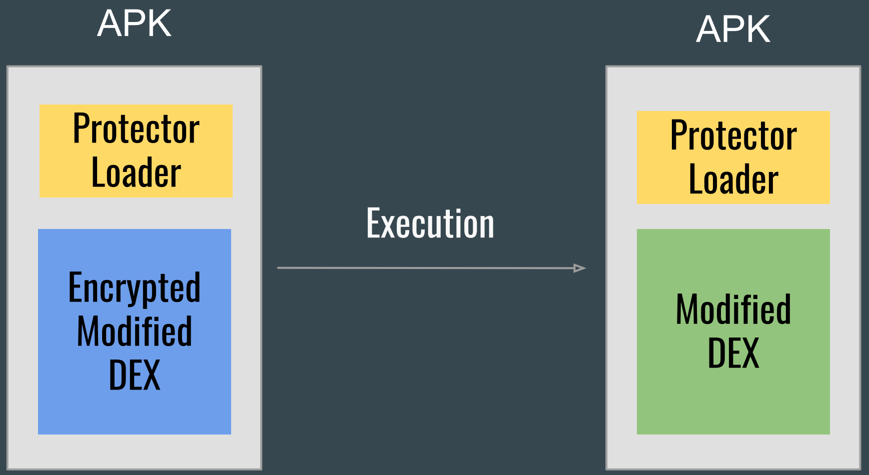
Apk Protection Techniques

- Obfuscators
- Packers
- **Protectors**



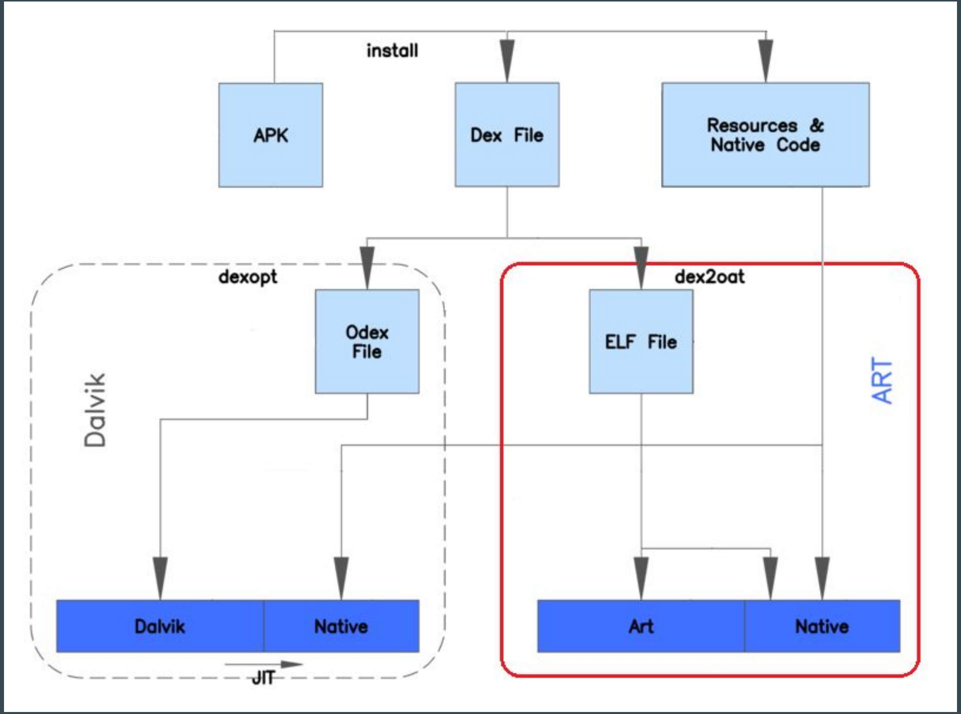
Apk Protection Techniques

- Obfuscators
- Packers
- **Protectors**



Back to Basics!

ART - Android RunTime VM

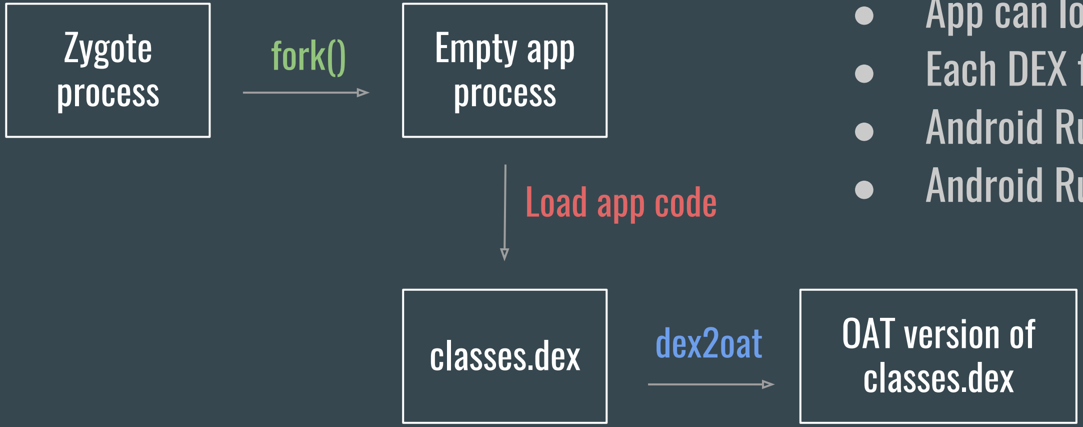


Provided an Ahead of Time (AOT) compilation approach

DEX to OAT

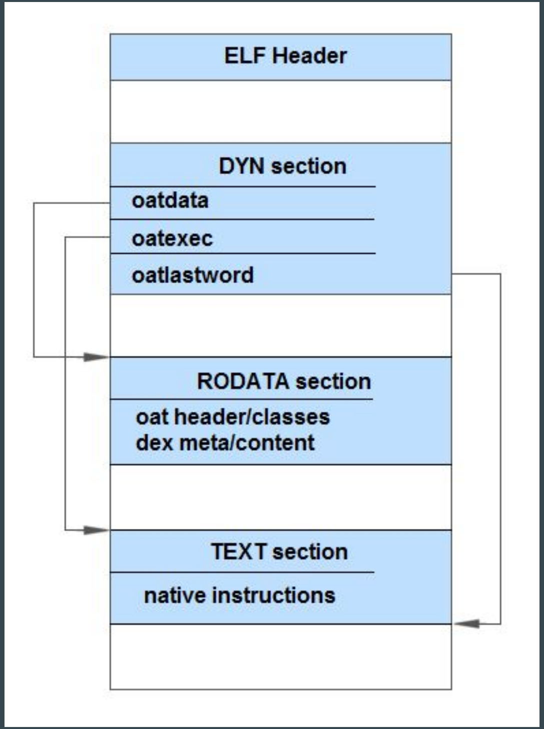
- Pre-compilation at install time
  - installation takes more time
  - more internal storage is required
- OAT vs JIT
  - Reduces startup time of applications
  - Improves battery performance
  - Uses less RAM

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

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
- Custom Android ROM

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

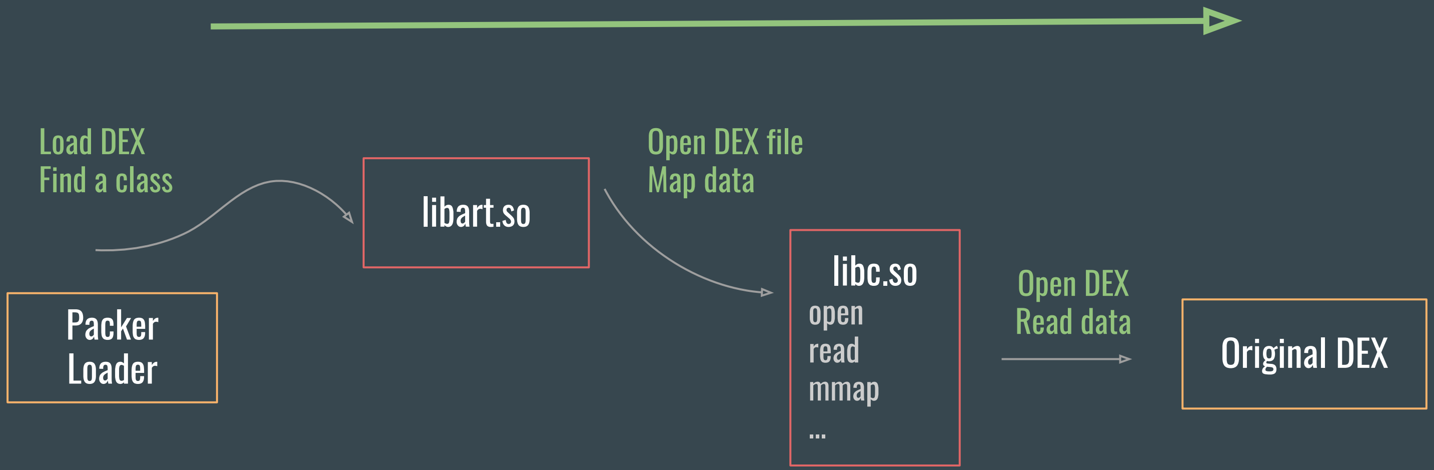
- What did want
    - Find a solution that
        - Require minimal changes to Android
        - Will work on most of the packers
- How did we do it?
    - Reversed most popular packers
    - Analyzed the DEX loading process
    - Patched a few lines of Android runtime code

Analyzed Packers

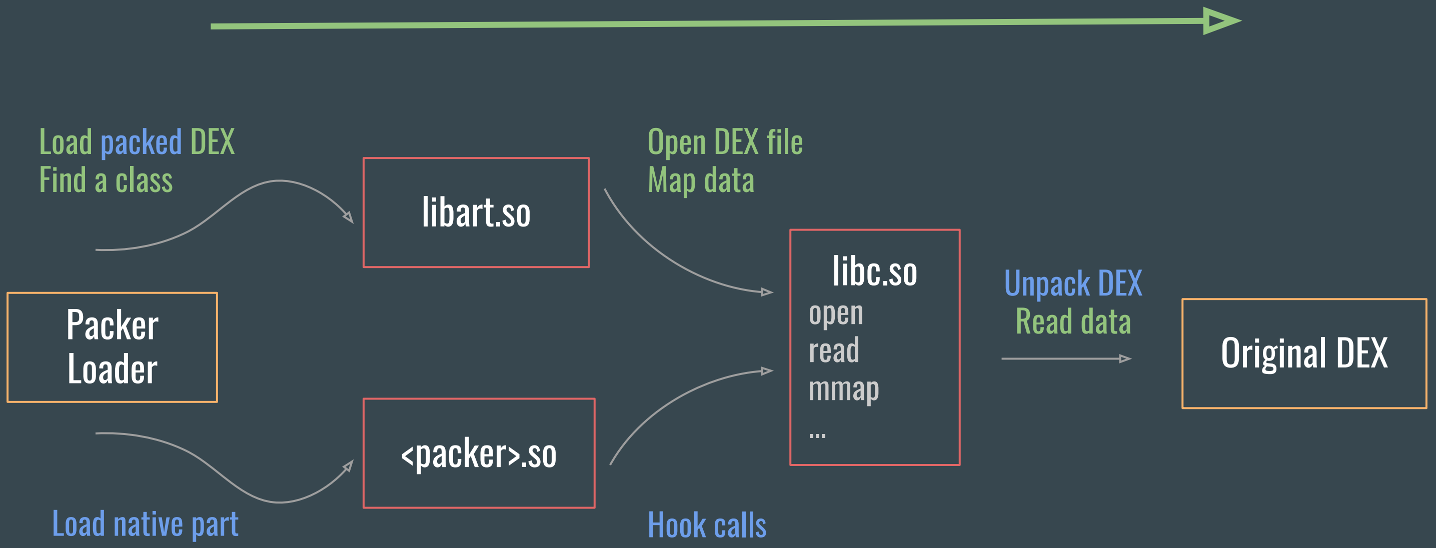
Most popular packers encountered

- Baidu
- Bangcle
- Tencent
- Ali
- 360 Jiagu
- … (and a few more)

Abstract Packer Model



Abstract Packer Model



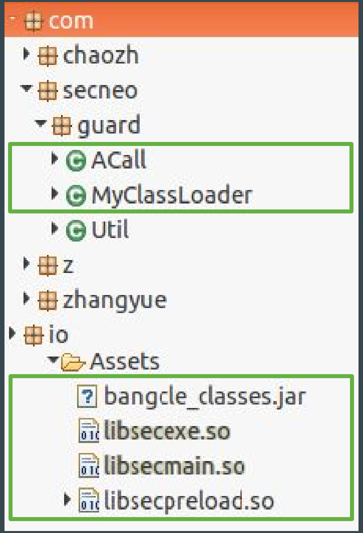
Bangcle - Classification

Classes

- ApplicationWrapper
- FirstApplication
- MyClassLoader
- ACall

Files

- libsecse
- libsecmain
- libsecexe
- libsecpreload
- bangcle_classes (original dex)



Bangcle - Java Loader Implementation

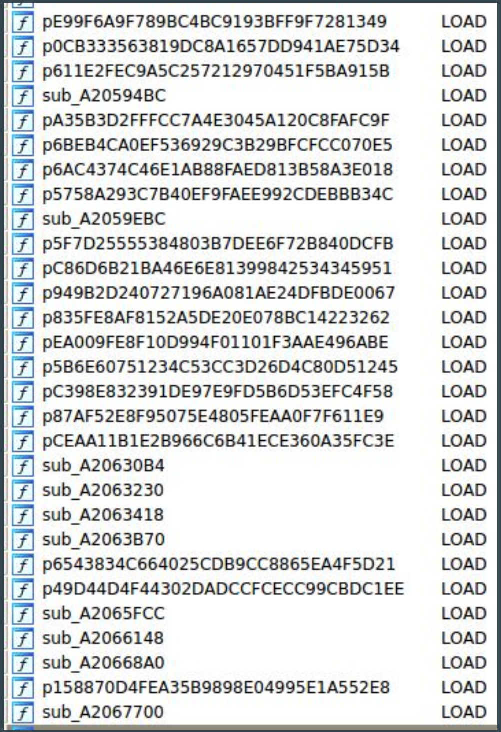
```
System.load("/data/data/" + getPackageName() + "/.cache/libsecexe.so");
Acall.getACall().a1(...);
Acall.getACall().r1(...);
Acall.getACall().r2(...);
...
public class MyClassLoader extends DexClassLoader {
    ...
}
cl = new MyClassLoader("/data/data/" + getPackageName() + "/.cache/classes.jar", ...);
realApplication = cl.loadClass(v0).newInstance();
```

Bangcle - Native Loader Implementation

Java Interface

```
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 Object c3(Object arg1, Object arg2);
    public native void jniCheckRawDexAvailable();
    public native boolean jniGetRawDexAvailable();
    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(Object arg1, Object arg2, Object arg3);
    public native Object set1(Activity arg1, ClassLoader arg2);
    public native Object set2(Application arg1, ...);
    public native void set3(Application arg1);
    public native void set3(Object arg1, Object arg2);
    public native void set4();
    public native void set5(ContentProvider arg1);
    public native void set8();
}
```

Native Functions



```
pE99F6A9F789BC4BC9193BFF9F7281349   LOAD
p0CB333563819DC8A1657DD941AE75D34   LOAD
p611E2FEC9A5C257212970451F5BA915B   LOAD
sub_A20594BC                        LOAD
pA35B3D2FFFCC7A4E3045A120C8FAFC9F   LOAD
p6BEB4CA0EF536929C3B29BFCFCC070E5   LOAD
p6AC4374C46E1AB88FAED813B58A3E018   LOAD
p5758A293C7B40EF9FAEE992CDEBBB34C   LOAD
sub_A2059EBC                        LOAD
p5F7D2555384803B7DEE6F72B840DCFB    LOAD
pC86D6B21BA46E6E8139984253434595 1  LOAD
p949B2D240727196A081AE24DFBDE0067   LOAD
p835FE8AF8152A5DE20E078BC14223262   LOAD
pEA009FE8F10D994F01101F3AAE496ABE   LOAD
p5B6E60751234C53CC3D26D4C80D51245   LOAD
pC398E832391DE97E9FD5B6D53EFC4F58   LOAD
p87AF52E8F95075E4805FEAA0F7F611E9   LOAD
pCEAA11B1E2B966C6B41ECE360A35FC3E   LOAD
sub_A20630B4                        LOAD
sub_A2063230                        LOAD
sub_A2063418                        LOAD
sub_A2063B70                        LOAD
p6543834C664025CDB9CC8865EA4F5D21   LOAD
p49D44D4F44302DADCCFCECC99CBDC1EE   LOAD
sub_A2065FCC                        LOAD
sub_A2066148                        LOAD
sub_A20668A0                        LOAD
p158870D4FEA35B9898E04995E1A552E8   LOAD
sub_A2067700                        LOAD
```

Mapping

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

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

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)

Dynamic section:

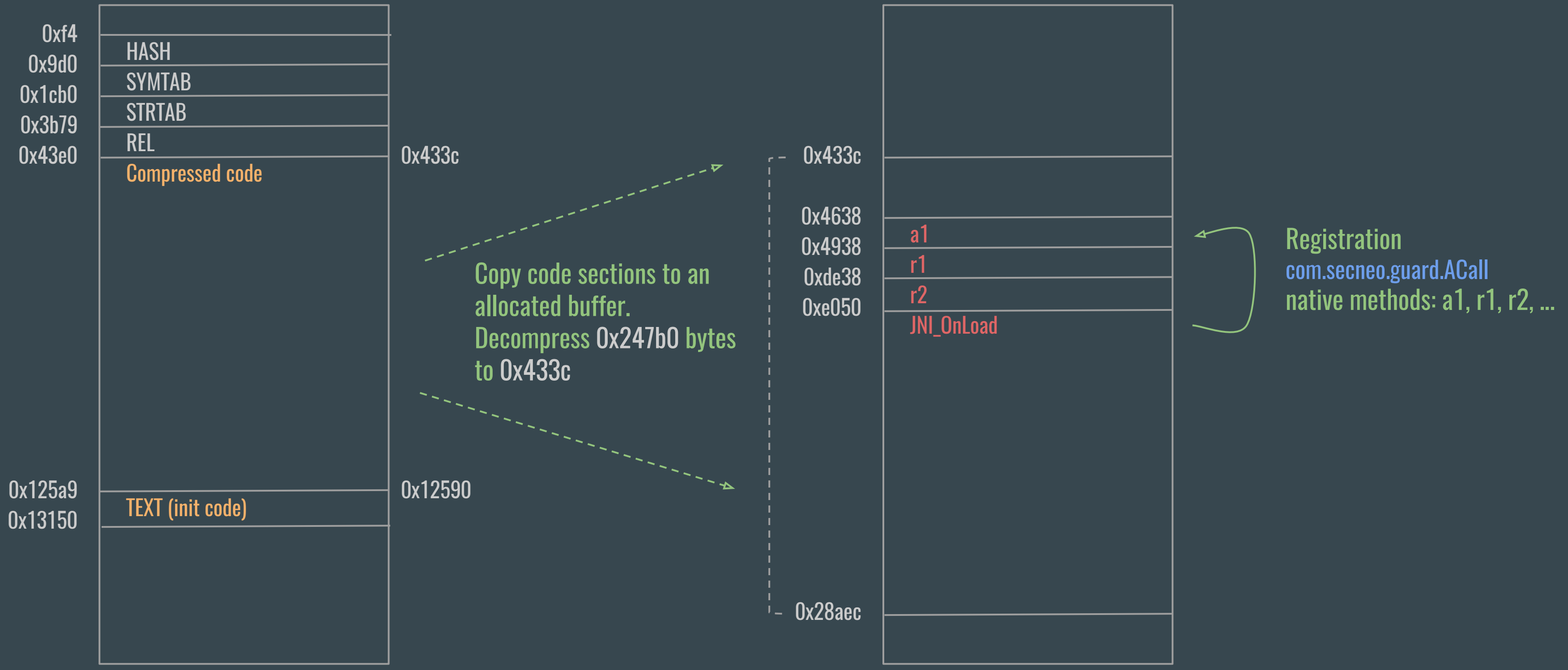
0x0000000c (INIT)	0x125A9
0x00000019 (INIT_ARRAY)	0x30C1C
...	

Real entry point

Program headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
EXIDX	0x028584	0x00028584	0x00028584	0x00568	0x00568	R	0x4
LOAD	0x000000	0x00000000	0x00000000	0x131ec	0x131ec	RE	0x8000
LOAD	0x018c1c	0x00030c1c	0x00030c1c	0x00520	0x01538	RW	0x8000
DYNAMIC	0x018c80	0x00030c80	0x00030c80	0x00108	0x00108	RW	0x4
GNU_STACK	0x000000	0x00000000	0x00000000	0x00000	0x00000	RW	0x4
GNU_RELRO	0x018c1c	0x00030c1c	0x00030c1c	0x003e4	0x003e4	R	0x1

Bangcle - libsecexe.so



Bangcle - Processes

Function a1

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

Function r2

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

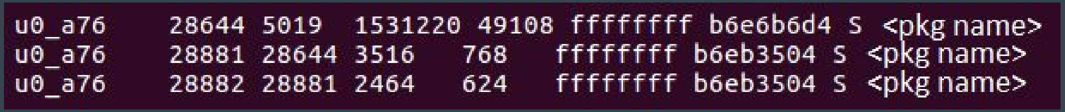
    LD_PRELOAD_ARGS=<pkg> 9 13

    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

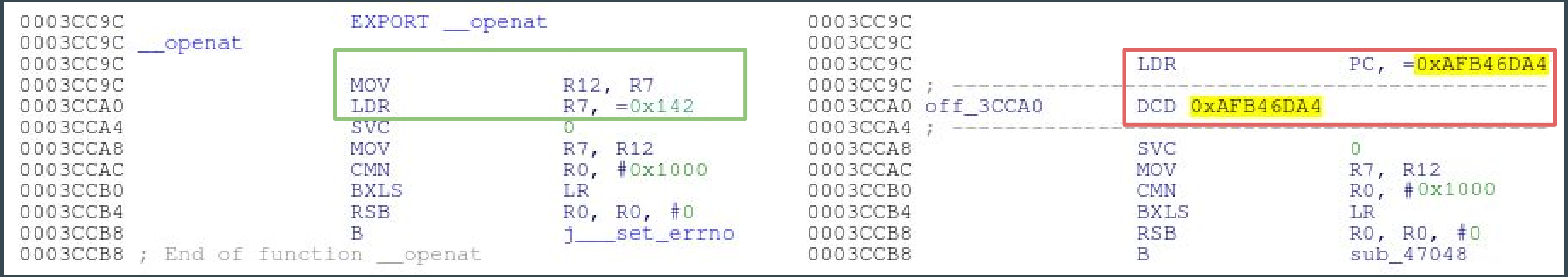
        –oat-location=/data/data/<pkg>/.cache/classes.dex –instruction-set=arm



```
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>
```

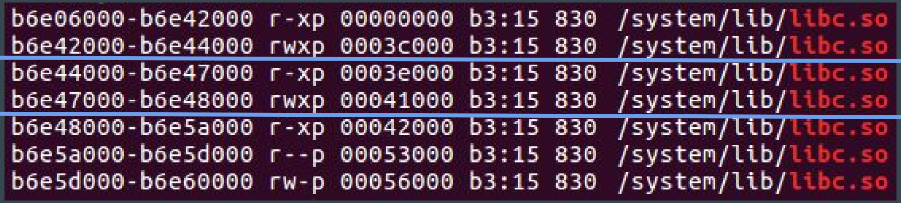
Bangcle - libc.so hook

Function r1



```
0003CC9C                EXPORT __openat
0003CC9C __openat
0003CC9C
0003CC9C                MOV             R12, R7
0003CCA0                LDR             R7, =0x142
0003CCA4                SVC             0
0003CCA8                MOV             R7, R12
0003CCAC                CMN             R0, #0x1000
0003CCB0                BXLS            LR
0003CCB4                RSB             R0, R0, #0
0003CCB8                B               j___set_errno
0003CCB8 ; End of function __openat
```

```
0003CC9C
0003CC9C
0003CC9C                LDR             PC, =0xAFB46DA4
0003CC9C ; ---------------------------------------------------------------
0003CCA0 off_3CCA0      DCD 0xAFB46DA4
0003CCA4 ; ---------------------------------------------------------------
0003CCA8                SVC             0
0003CCAC                MOV             R7, R12
0003CCB0                CMN             R0, #0x1000
0003CCB4                BXLS            LR
0003CCB8                RSB             R0, R0, #0
0003CCB8                B               sub_47048
```



```
b6e06000-b6e42000 r-xp 00000000 b3:15 830  /system/lib/libc.so
b6e42000-b6e44000 rwxp 0003c000 b3:15 830  /system/lib/libc.so
b6e44000-b6e47000 r-xp 0003e000 b3:15 830  /system/lib/libc.so
b6e47000-b6e48000 rwxp 00041000 b3:15 830  /system/lib/libc.so
b6e48000-b6e5a000 r-xp 00042000 b3:15 830  /system/lib/libc.so
b6e5a000-b6e5d000 r--p 00053000 b3:15 830  /system/lib/libc.so
b6e5d000-b6e60000 rw-p 00056000 b3:15 830  /system/lib/libc.so
```

Protection was changed

libc func	Offset	libc func	Offset
munmap	0x15BD8	close	0x14FAC
msync	0x15F88	__openat	0x14DA4
read	0x15118	pread64	0x162F8
__mmap2	0x15420	pwrite64	0x166DC
__open	0x14B9C	write	0x152FC

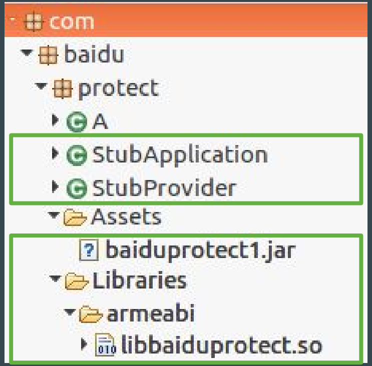
Bangcle - Summary

- 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)



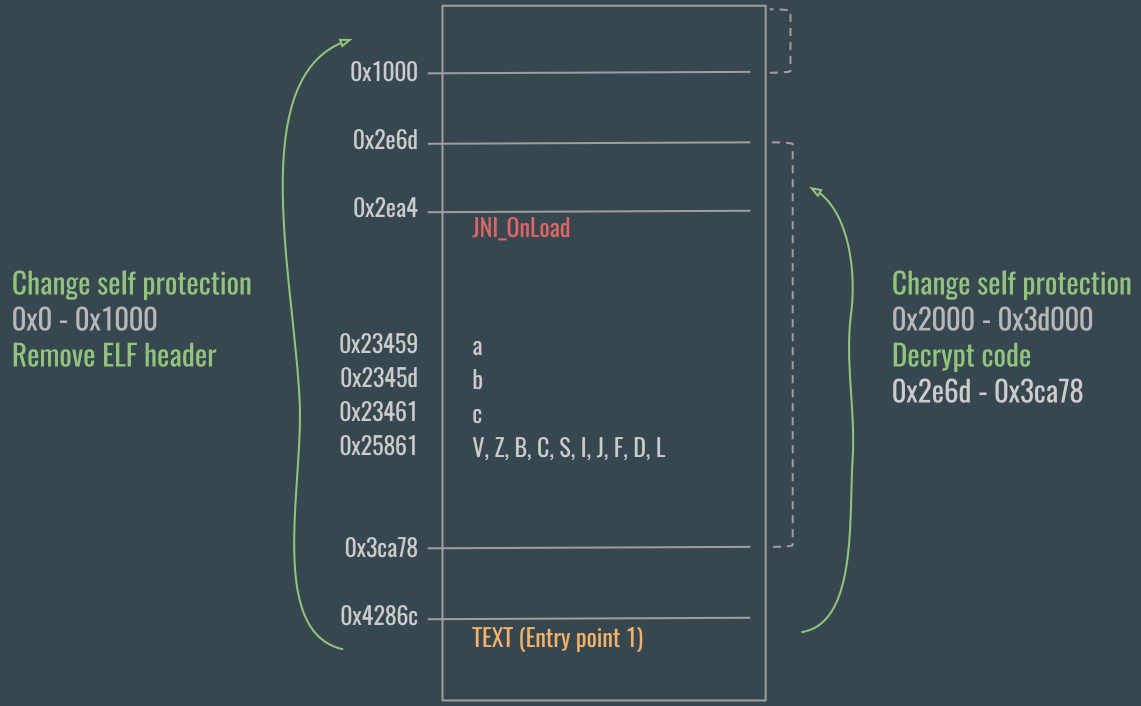
```
com
  baidu
    protect
      A
      StubApplication
      StubProvider
    Assets
      baiduprotect1.jar
    Libraries
      armeabi
        libbaiduprotect.so
```

Baidu - Native Loader Implementation

```
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();
}
```

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

Baidu - libbaiduprotect.so



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

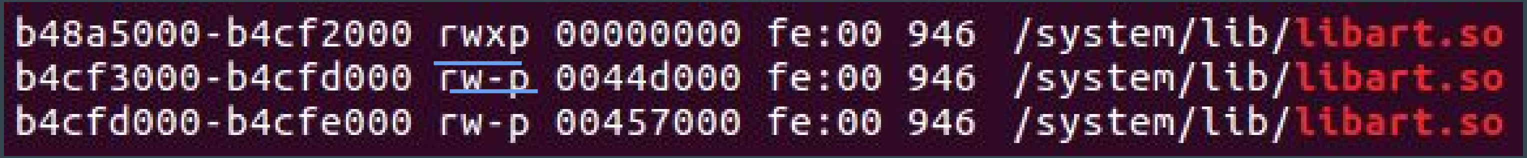
Hook libart.so

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



```
b48a5000-b4cf2000 rwxp 00000000 fe:00 946  /system/lib/libart.so
b4cf3000-b4cfd000 rw-p 0044d000 fe:00 946  /system/lib/libart.so
b4cfd000-b4cfe000 rw-p 00457000 fe:00 946  /system/lib/libart.so
```

- 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/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 libart for handling the opening of the DEX file.

I'VE ALREADY SEEN THAT!!!
imgflip.com

libc::open == decryption

Bangle		Baidu
	Filter by file path:	
/data/data/<pkg>/.cache/classes.dex		/data/data/<pkg>/.1 /classes.jar
	Expect to see:	
OAT		DEX

Using the DEX Loading Process to Unpack Apps

Where is first call of DEX/OAT file opening?

OAT

DEX

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

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

OAT

```
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

```
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;

  ...
}
```

Demo Time!

Tool can be found at -
github.com/CheckPointSW/android_unpacker

Summary

- A few minor changes to the ART VM enables a wide coverage of packers.
- Since rollout to production we have witnessed a 50% increase in detection.

Questions?

github.com/CheckPointSW/android_unpacker

avi@mydro.co
slavam@checkpoint.com