State Of The ART
Exploring The New Android KitKat Runtime
Agenda

Introduction

Ahead of time compilation

OAT file format

Security implications

Reverse engineering
Background

- Introduced in Android KitKat 4.4 back in October, 2013
- Still in experimental stage
- Poised to replace Dalvik
Background

- Dalvik
  - Dexopt
  - Just-in-time (JIT) compilation

- ART
  - Ahead-of-time (AOT) compilation
  - Dalvik bytecode -> Native code
Background

- Advantages
  - Better performance
  - Better battery life

- (slight) Disadvantages
  - More storage space
  - Longer installation time
Turning on ART

- Settings > Developer options > Select runtime

Select runtime

- Use Dalvik
- Use ART

Cancel
Turning on ART

- Runtime selection is not possible on some devices using official releases
  - 2012 Nexus 7
  - Nexus 10
- Third-party ROMs
- Build from AOSP
Turning on ART

- To check which runtime is enabled

  `getprop persist.sys.dalvik.vm.lib.1`

- Returns “libart.so” if ART is enabled

- Returns “libdvm.so” if Dalvik
Before we proceed

- ART is still under heavy development
- Some parts of this talk may change
- In some parts will focus on the fundamental principles versus details that may change
Agenda

Introduction

**Ahead of time compilation**

OAT file format

Security implications

Reverse engineering
When?

- Upon reboot after ART is enabled
  - Creates boot.oat and boot image
  - All installed apps will be compiled
  - May take a while

- App installation

- When it meets certain criteria based on profiling results
Dex2oat

- Ex:
  /system/bin/dex2oat --zip-fd=6 --zip-location=/system/app/Email.apk --oat-fd=7 --oat-location=/data/dalvik-cache/system@app@Email.apk@classes.dex --profile-file=/data/dalvik-cache/profiles/com.android.email

- Resulting OAT file will be placed in /data/dalvik-cache
Dex2oat

- Retrieve classes.dex from APK
- Verify each class
- Verify each method
- Verify each Dalvik instruction
- Compile bytecode in all methods in each class into native code
  - Except class initializers (<clinit>)
Boot.oat

- `system@framework@boot.oat`

- Contains libs and frameworks in boot class path – To be pre-loaded in all apps

```
```
Ahead Of Time Compilation

Boot.oat

- /system/framework/core-libart.jar
- /system/framework/conscrypt.jar
- /system/framework/okhttp.jar
- /system/framework/core-junit.jar
- /system/framework/bouncycastle.jar
- /system/framework/ext.jar
- /system/framework/framework.jar
- /system/framework/framework2.jar
- /system/framework/telephony-common.jar
- /system/framework/voip-common.jar
- /system/framework/mms-common.jar

- /system/framework/android.policy.jar
- /system/framework/services.jar
- /system/framework/apache-xml.jar
- /system/framework/webviewchromium.jar
Boot image

- `system@framework@boot.art`
- Contains absolute pointers for methods in `boot.oat`
- `boot.oat` contain absolute pointers to methods in the boot image
- Loaded by zygote along with `boot.oat`
Compilation

- Compiler backends:
  - Quick
  - Optimizing
  - Portable

- "-compile-backend" option for dex2oat

- Current default is Quick
Quick Backend

- Medium level IR (DEX bytecode)
- Low level IR
- Native code
- Some optimizations at each stage
Optimizing backend

- Basically Quick with additional optimizations
- Still in heavy development
Portable backend

- Uses LLVM bitcode as its LIR
- Optimizations using LLVM optimizer
- Code generation is done by LLVM backends
Profiling

- By default, ART compiles methods regardless of impact on performance

- Profiling feature allows ART to be more selective on which methods to compile
Profiling

- Currently disabled by default
- To enable:
  
  setprop dalvik.vm.profiler 1

- No AOT compilation upon app install
  – Reduced install time
  – Save on disk space
Profiling

- Profiling data is collected while app is running
- Profile files are placed in /data/dalvik-cache/profiles
- Profile file name is the package name
- Profile data is used to determine if AOT compilation will be done
Profiling

- First line is the summary information
  - Samples count/Null methods count/Boot path methods count
- Subsequent lines are the profile data
  - Method name/Count/Size

42/2/352
android.database.Cursor com.android.email.provider.EmailProvider.uiAccounts(java.lang.String[])/1/128
void com.android.email.NotificationController.ensureHandlerExists() /1/37
int com.android.email.provider.EmailProvider.getFolderTypeFromMailboxType(int) /2/56
boolean com.android.mail.browse.ConversationCursor$ConversationProvider.onCreate() /1/49
com.google.common.collect.ImmutableList com.google.common.collect.ImmutableList.of() /1/3

<snip>
Profiling

- When?
  - Does the app need to undergo dex2oat?
    • Number of methods comprising 90% of called methods has changed by > 10%
  - If yes, which methods are to be compiled?
    • Methods comprising 90% of called methods
Agenda

Introduction
Ahead of time compilation
OAT file format
Security implications
Reverse engineering
# OAT File Format

## OAT File

- **ELF dynamic object**
- **.oat file extension**

```
<table>
<thead>
<tr>
<th>struct dynamic_symbol_table</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct Elf32_Sym symtab[0]</td>
</tr>
<tr>
<td>struct Elf32_Sym symtab[1]</td>
</tr>
<tr>
<td>struct sym_name32_t sym_name</td>
</tr>
<tr>
<td>Elf32_Addr sym_value</td>
</tr>
<tr>
<td>Elf32_Word sym_size</td>
</tr>
<tr>
<td>sym_name32_t sym_name</td>
</tr>
<tr>
<td>Elf32_Addr sym_value</td>
</tr>
<tr>
<td>Elf32_Word sym_size</td>
</tr>
<tr>
<td>sym_info_t sym_info</td>
</tr>
<tr>
<td>unsigned char sym_other</td>
</tr>
<tr>
<td>Elf32_Half sym_shndx</td>
</tr>
<tr>
<td>char sym_data[892928]</td>
</tr>
<tr>
<td>struct Elf32_Sym symtab[2]</td>
</tr>
<tr>
<td>struct sym_name32_t sym_name</td>
</tr>
<tr>
<td>Elf32_Addr sym_value</td>
</tr>
<tr>
<td>Elf32_Word sym_size</td>
</tr>
<tr>
<td>sym_info_t sym_info</td>
</tr>
<tr>
<td>unsigned char sym_other</td>
</tr>
<tr>
<td>Elf32_Half sym_shndx</td>
</tr>
<tr>
<td>char sym_data[605104]</td>
</tr>
<tr>
<td>struct Elf32_Sym symtab[3]</td>
</tr>
<tr>
<td>struct sym_name32_t sym_name</td>
</tr>
<tr>
<td>Elf32_Addr sym_value</td>
</tr>
<tr>
<td>Elf32_Word sym_size</td>
</tr>
<tr>
<td>sym_info_t sym_info</td>
</tr>
<tr>
<td>unsigned char sym_other</td>
</tr>
<tr>
<td>Elf32_Half sym_shndx</td>
</tr>
<tr>
<td>char sym_data[4]</td>
</tr>
</tbody>
</table>
```
OAT File

- Dynamic symbol tables pointing to OAT data and code
  - oatdata
  - oatexec
  - oatlastword

```
<table>
<thead>
<tr>
<th>struct dynamic_symbol_table</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct Elf32_Sym symtab[0]</td>
</tr>
<tr>
<td>struct Elf32_Sym symtab[1]</td>
</tr>
<tr>
<td>struct sym_name32_t sym_name</td>
</tr>
<tr>
<td>Elf32_Addr sym_value</td>
</tr>
<tr>
<td>Elf32_Xword sym_size</td>
</tr>
<tr>
<td>struct sym_info_t sym_info</td>
</tr>
<tr>
<td>unsigned char sym_other</td>
</tr>
<tr>
<td>Elf32_Half sym_shndx</td>
</tr>
<tr>
<td>char sym_data[892928]</td>
</tr>
<tr>
<td>struct Elf32_Sym symtab[2]</td>
</tr>
<tr>
<td>struct sym_name32_t sym_name</td>
</tr>
<tr>
<td>Elf32_Addr sym_value</td>
</tr>
<tr>
<td>Elf32_Xword sym_size</td>
</tr>
<tr>
<td>struct sym_info_t sym_info</td>
</tr>
<tr>
<td>unsigned char sym_other</td>
</tr>
<tr>
<td>Elf32_Half sym_shndx</td>
</tr>
<tr>
<td>char sym_data[605104]</td>
</tr>
<tr>
<td>struct Elf32_Sym symtab[3]</td>
</tr>
<tr>
<td>struct sym_name32_t sym_name</td>
</tr>
<tr>
<td>Elf32_Addr sym_value</td>
</tr>
<tr>
<td>Elf32_Xword sym_size</td>
</tr>
<tr>
<td>struct sym_info_t sym_info</td>
</tr>
<tr>
<td>unsigned char sym_other</td>
</tr>
<tr>
<td>Elf32_Half sym_shndx</td>
</tr>
<tr>
<td>char sym_data[4]</td>
</tr>
</tbody>
</table>
```
## OAT File Format

### OAT File

- **oatdata** -> headers, DEX files
- **oatexec** -> compiled code
- **oatlastword** -> end marker

```
<table>
<thead>
<tr>
<th>struct dynamic_symbol_table</th>
</tr>
</thead>
<tbody>
<tr>
<td>▷ struct Elf32_Sym symtab[0]</td>
</tr>
<tr>
<td>▷ struct Elf32_Sym symtab[1]</td>
</tr>
<tr>
<td>▷ struct sym_name32_t sym_name</td>
</tr>
<tr>
<td>Elf32_Addr       sym_value</td>
</tr>
<tr>
<td>Elf32_Xword      sym_size</td>
</tr>
<tr>
<td>▷ struct sym_info_t sym_info</td>
</tr>
<tr>
<td>unsigned char    sym_other</td>
</tr>
<tr>
<td>Elf32_Half       sym_shndx</td>
</tr>
<tr>
<td>▷ char sym_data[892928]</td>
</tr>
<tr>
<td>▷ struct Elf32_Sym symtab[2]</td>
</tr>
<tr>
<td>▷ struct sym_name32_t sym_name</td>
</tr>
<tr>
<td>Elf32_Addr       sym_value</td>
</tr>
<tr>
<td>Elf32_Xword      sym_size</td>
</tr>
<tr>
<td>▷ struct sym_info_t sym_info</td>
</tr>
<tr>
<td>unsigned char    sym_other</td>
</tr>
<tr>
<td>Elf32_Half       sym_shndx</td>
</tr>
<tr>
<td>▷ char sym_data[605104]</td>
</tr>
<tr>
<td>▷ struct Elf32_Sym symtab[3]</td>
</tr>
<tr>
<td>▷ struct sym_name32_t sym_name</td>
</tr>
<tr>
<td>Elf32_Addr       sym_value</td>
</tr>
<tr>
<td>Elf32_Xword      sym_size</td>
</tr>
<tr>
<td>▷ struct sym_info_t sym_info</td>
</tr>
<tr>
<td>unsigned char    sym_other</td>
</tr>
<tr>
<td>Elf32_Half       sym_shndx</td>
</tr>
<tr>
<td>▷ char sym_data[4]</td>
</tr>
</tbody>
</table>
```
OAT File Format

OAT File

- OAT Header
- OAT DEX File Header [0]
- OAT DEX File Header [n]
- DEX File [0]
- DEX File [n]
- OAT Class [0]
- OAT Class [0]

oatdata (.rodata)

oatexec (.text)

OAT Code
## OAT Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>magic</td>
<td>uByte[4]</td>
<td>Magic value, &quot;oat\n&quot;</td>
</tr>
<tr>
<td>adler32_checksum</td>
<td>Uint32</td>
<td>Adler-32 checksum of the executable code data</td>
</tr>
<tr>
<td>instruction_set</td>
<td>Uint32</td>
<td>Instruction set architecture</td>
</tr>
<tr>
<td>instruction_set_features</td>
<td>Uint32</td>
<td>Bitmask of supported features per architecture</td>
</tr>
<tr>
<td>dex_file_count</td>
<td>Uint32</td>
<td>Number of DEX files in the OAT</td>
</tr>
<tr>
<td>executable_offset</td>
<td>Uint32</td>
<td>Offset of executable code section from start of oatdata</td>
</tr>
<tr>
<td>interpreter_to_interpreter_bridge_offset</td>
<td>Uint32</td>
<td>Offset from oatdata start to interpreter_to_interpreter_bridge stub</td>
</tr>
<tr>
<td>interpreter_to_compiled_code_bridge_offset</td>
<td>Uint32</td>
<td>Offset from oatdata start to interpreter_to_compiled_code_bridge stub</td>
</tr>
<tr>
<td>jni_disym_lookup_offset</td>
<td>Uint32</td>
<td>Offset from oatdata start to jni_disym_lookup_stub</td>
</tr>
<tr>
<td>portable_imt_conflict_trampoline_offset</td>
<td>Uint32</td>
<td>Offset from oatdata start to portable_imt_conflict_trampoline stub</td>
</tr>
<tr>
<td>portable_resolution_trampoline_offset</td>
<td>Uint32</td>
<td>Offset from oatdata start to portable_resolution_trampoline stub</td>
</tr>
<tr>
<td>portable_to_interpreter_bridge_offset</td>
<td>Uint32</td>
<td>Offset from oatdata start to portable_to_interpreter_bridge stub</td>
</tr>
<tr>
<td>quick_generic_jni_trampoline_offset</td>
<td>Uint32</td>
<td>Offset from oatdata start to quick_generic_jni_trampoline stub</td>
</tr>
<tr>
<td>quick_imt_conflict_trampoline_offset</td>
<td>Uint32</td>
<td>Offset from oatdata start to quick_imt_conflict_trampoline stub</td>
</tr>
<tr>
<td>quick_resolution_trampoline_offset</td>
<td>Uint32</td>
<td>Offset from oatdata start to quick_resolution_trampoline stub</td>
</tr>
<tr>
<td>quick_to_interpreter_bridge_offset</td>
<td>Uint32</td>
<td>Offset from oatdata start to quick_to_interpreter_bridge stub</td>
</tr>
<tr>
<td>image_file_location_oat_checksum</td>
<td>Uint32</td>
<td>Checksum of image file's path</td>
</tr>
<tr>
<td>image_file_location_oat_data_begin</td>
<td>Uint32</td>
<td>The virtual address of the image file's oatdata section</td>
</tr>
<tr>
<td>image_file_location_size</td>
<td>Uint32</td>
<td>The length of the image file's path</td>
</tr>
</tbody>
</table>
OAT Header

- Supported instruction sets
  - ARM
  - ARM64
  - Thumb2
  - X86
  - X86_64
  - Mips
### OAT DEX File Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dex_file_location_size</td>
<td>uint32</td>
<td>Length of the original input DEX path</td>
</tr>
<tr>
<td>dex_file_location_data</td>
<td>ubyte[dex_file_location_size]</td>
<td>Original path of input DEX file</td>
</tr>
<tr>
<td>dex_file_location_checksum</td>
<td>uint32</td>
<td>Checksum of path string</td>
</tr>
<tr>
<td>dex_file_pointer</td>
<td>uint32</td>
<td>Offset of embedded input DEX</td>
</tr>
<tr>
<td>classes_offsets</td>
<td>uint32[DEX.header.class_defs_size]</td>
<td>List of offsets to OATClassHeaders</td>
</tr>
</tbody>
</table>

- The original DEX file is embedded in the OAT data section
OAT Class Header

- **Status**
  - kStatusError
  - kStatusNotReady
  - kStatusIdx
  - kStatusLoaded
  - kStatusResolved
  - kStatusVerifying
  - kStatusRetryVerificationAtRuntime
  - kStatusVerifyingAtRuntime
  - kStatusVerified
  - kStatusInitializing
  - kStatusInitialized

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>uint16</td>
<td>State of class during compilation</td>
</tr>
<tr>
<td>type</td>
<td>uint16</td>
<td>Type of class</td>
</tr>
<tr>
<td>bitmap_size</td>
<td>uint32</td>
<td>Size of methods bitmap</td>
</tr>
<tr>
<td>bitmap_pointer</td>
<td>uint32</td>
<td>Offset to methods bitmap</td>
</tr>
<tr>
<td>methods_pointer</td>
<td>uint32</td>
<td>Offset to methods</td>
</tr>
</tbody>
</table>
OAT Class Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>uint16</td>
<td>State of class during compilation</td>
</tr>
<tr>
<td>type</td>
<td>uint16</td>
<td>Type of class</td>
</tr>
<tr>
<td>bitmap_size</td>
<td>uint32</td>
<td>Size of methods bitmap</td>
</tr>
<tr>
<td>bitmap_pointer</td>
<td>uint32</td>
<td>Offset to methods bitmap</td>
</tr>
<tr>
<td>methods_pointer</td>
<td>uint32</td>
<td>Offset to methods</td>
</tr>
</tbody>
</table>

- **Type**
  - kOatClassAllCompiled
  - kOatClassSomeCompiled
  - kOatClassNoneCompiled
OAT Class Header

- `kOatClassAllCompiled`
  - All methods in the class were compiled

- `kOatClassSomeCompiled`
  - Some of the methods in the class were compiled

- `kOatClassNoneCompiled`
  - None of the methods in the class were compiled
Bitmaps are used to represent which methods are compiled

Each bit represents every method in the class, starting with direct methods, then virtual methods

If bit it is set, the method was compiled
**OAT Method**

- **OatMethodOffset**

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code_offset</td>
<td>uint32</td>
<td>Offset of compiled code from start of oatdata</td>
</tr>
<tr>
<td>frame_size_in_bytes</td>
<td>uint32</td>
<td>Frame size for this method when executed</td>
</tr>
<tr>
<td>core_spill_mask</td>
<td>uint32</td>
<td>Bitmap of spilled machine registers</td>
</tr>
<tr>
<td>fp_spill_mask</td>
<td>uint32</td>
<td>Bitmap of spilled floating point machine registers</td>
</tr>
<tr>
<td>gc_map_offset</td>
<td>uint32</td>
<td>Offset to the GC map</td>
</tr>
</tbody>
</table>

- Corresponds to each compiled method
OAT File Format

OAT Method

- OATMethodHeader

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapping_table_offset</td>
<td>uint32</td>
<td>Offset from the start of the mapping table</td>
</tr>
<tr>
<td>vmap_table_offset</td>
<td>uint32</td>
<td>Offset form the start of the vmap table</td>
</tr>
<tr>
<td>code_size</td>
<td>uint32</td>
<td>Method’s code size in bytes</td>
</tr>
</tbody>
</table>

- Appears right before method code
Agenda

Introduction
Ahead of time compilation
OAT file format
Security implications
Reverse Engineering
Compiler vulnerabilities

- New technology means new code
- New code means more potential mistakes
Fuzzing the AOT compiler

- Used dumb fuzzing methods
- Generated DEX files with mutated method code
- Ran dex2oat against them
Fuzzing the AOT compiler

- Found several crashes

- Did not pursue further due still evolving code in ART

- Viable target once ART stabilizes
User mode rootkits

- Post exploitation scenario
- Attacker already has elevated privileges
- Some past examples in Android
  - Erez Metula in his book “Managed Code Rootkits”
  - Tsukasa Oi’s “Yet Another Android Rootkit” paper
User mode rootkits

- Technologies such as dm-verity introduced in KitKat makes rootkits relying on /system partition modifications obsolete

- No write to /system, or anywhere else except boot.oat, no memory modifications, no ptrace
User mode rootkits

- Example idea
  - Parse the boot image to locate address of methods to hook
  - Patch the target compiled method in boot.oat to jump to your code
  - Hide your code inside boot.oat using ELF virus techniques
User mode rootkits

- Ongoing research
ASLR bypass

Base address of boot image is fixed at 0x700000

```
5d5ba000-5ee19000 r-xp 00000000 b3:03 922 /system/lib/libwebviewchromium.so
5ee19000-5ee1a000 ---p 00000000 00:00 0
5ee1a000-5ef2e000 r--p 0185f000 b3:03 922 /system/lib/libwebviewchromium.so
5ef2e000-5ef48000 rw-p 02973000 b3:03 922 /system/lib/libwebviewchromium.so
5ef48000-6013e000 ---s 00000000 b3:03 1202 /system/usr/icu/icudt53l.dat
6013e000-6073e000 rw-p 00000000 00:04 7375
6081d000-60e1d000 rw-p 00000000 00:04 7376
60e1d000-60e1e000 ---p 00000000 00:00 0
60e1e000-60f21000 rw-p 00000000 00:00 0
60f2e000-60fe1000 ---p 00000000 00:00 0
60fe1000-610e4000 rw-p 00000000 00:00 0
61122000-61123000 ---p 00000000 00:00 0
61123000-61226000 rw-p 00000000 00:00 0
70000000-70b28000 rw-p 00000000 b3:09 425155 /data/dalvik-cache/system@framework@boot.art
70b28000-7286e000 r-xp 00000000 b3:09 425271 /data/dalvik-cache/system@framework@boot.oat
7286e000-74257000 r-xp 01d46000 b3:09 425271 /data/dalvik-cache/system@framework@boot.oat
74257000-74258000 rw-p 0372f000 b3:09 425271 /data/dalvik-cache/system@framework@boot.oat
74258000-74687000 rw-p 00000000 00:04 3462 /dev/ashmem/dalvik-allocspace main rosalloc space live-bitmap 2 (deleted)
74687000-74688000 rw-p 00000000 00:04 7376 /dev/ashmem/dalvik-allocspace main rosalloc space mark-bitmap 2 (deleted)
74688000-77a59000 ---p 00001000 00:04 7377 /dev/ashmem/dalvik-allocspace main rosalloc space (deleted)
77a59000-78258000 rw-p 00000000 00:04 3461 /dev/ashmem/dalvik-allocspace main rosalloc space (deleted)
78258000-78459000 rw-p 00000000 00:04 3461 /dev/ashmem/dalvik-allocspace main rosalloc space (deleted)
78459000-90258000 ---p 00201000 00:04 3461 /dev/ashmem/dalvik-main space (deleted)
be94f000-be970000 rw-p 00000000 00:00 0 [stack]
ffffff0000-ffffff1000 r-xp 00000000 00:00 0 [vectors]
```
ASLR bypass

- Base address of boot image is fixed at 0x700000
ASLR bypass

- Base address of boot image is fixed at 0x700000

```
60405000-60409000  rw-p  00000000 00:00 0
60461000-60464000  rw-p  00000000 00:00 0
60474000-6047a000  rw-p  00000000 00:00 0
60514000-60517000  rw-p  00000000 00:00 0
60601000-60605000  rw-p  00000000 00:00 0
606cf000-606d2000  rw-p  00000000 00:00 0
6076b000-60770000  rw-p  00000000 00:00 0
60e68000-61468000  rw-p  00000000 00:00 0
61469000-6156c000  rw-p  00000000 00:00 0
6156d000-61670000  rw-p  00000000 00:00 0
70000000-70b28000  rw-p  00000000 b3:09 425155     /data/dalvik-cache/system@framework@boot.art
70b28000-7286e000  r--p  00000000 b3:09 425154     /data/dalvik-cache/system@framework@boot.oat
7286e000-74257000  r-xp  01d46000 b3:09 425154     /data/dalvik-cache/system@framework@boot.oat
74257000-74258000  rw-p  0372f000 b3:09 425154     /data/dalvik-cache/system@framework@boot.oat
74258000-74687000  rw-p  00000000 00:04 5472     /dev/ashmem/dalvik-zygote / non moving space (deleted)
74687000-74688000  rw-p  00000000 00:04 6274     /dev/ashmem/dalvik-alloc space (deleted)
74688000-77a39000  ----p  00001000 00:04 6274     /dev/ashmem/dalvik-alloc space (deleted)
77a59000-78258000  rw-p  033d2000 00:04 5472     /dev/ashmem/dalvik-alloc space (deleted)
78258000-78459000  rw-p  00000000 00:04 5471     /dev/ashmem/dalvik-main space (deleted)
78459000-90258000  ----p  00201000 00:04 5471     /dev/ashmem/dalvik-main space (deleted)
bed25000-bed46000  rw-p  00000000 00:00 0          [stack]
ffff0000-ffff1000  r-xp  00000000 00:00 0          [vectors]
```
ASLR bypass

- Base address of boot image is fixed at 0x700000
- Rich source of ROP gadgets
- boot.oat code section has 27 mb of code

7286e000-74257000 r-xp 01d46000 b3:09 425154 /data/dalvik-cache/system@framework@boot.oat
Agenda

Introduction
Ahead of time compilation
OAT file format
Security implications
Reverse engineering
Static analysis

- Still better to read Dalvik bytecode disassembly (unless you’re weird)
- If you are, you can use oatdump to dump the native code disassembly
  - You can find it in your ART enabled device

  oatdump -oat-file=<oat-file>
Static analysis

DEX CODE:
00049758: invoke-direct {v0}, void android/app/Activity-><init>() // method@(11, 0x000b)
0004975e: return-void

COMPiled CODE:
0x00000000: ldr.w ip, [sb, #0x78]
0x00000004: push.w {r5, r6, lr}
0x00000008: subs.w sp, sp, #0x14
0x0000000c: cmp sp, ip
0x0000000e: blo.w #0x38
0x00000012: adds r6, r0, #0
0x00000014: str r0, [sp]
0x00000018: movw lr, #0xbd05
0x0000001c: movt lr, #0x72f0 ; entryPointFromQuickCompiledCode
0x00000020: movw r0, #0x7053 ; void android.app.Activity.<init>()
0x00000024: adds r5, r1, #0
0x00000026: blx lr
0x00000028: b #0x32
0x0000002a: movs r0, r0
Static analysis

- oatdump dumps the whole OAT file

- Need to have a tool to dump individual classes or methods and display xrefs

- Or better yet, an IDA plugin
Dynamic analysis

- Debugging Java code
  - ART supports JDWP, so you can use jdb (theoretically, haven’t tried)

- Use gdb to debug native code
  - Get address of method using oatdump
  - Set breakpoint
  - trace
Dynamic analysis

- Dynamic instrumentation
  - Cydia Substrate for Android by saurik
  - Xposed Framework by rovo89

- ART not supported yet in these tools

- But work is ongoing
Dynamic analysis

- For now, static instrumentation is still the way to go
  - unpack
  - disassemble
  - add instrumentation
  - assemble
  - repackage
Conclusion

- ART is poised to supersede Dalvik in (hopefully) the near future

- Ripe for more security research

- RE tools need to adapt
Questions?
Thanks for listening!

Paul Sabanal
@polsab