Static instrumentation based on executable file formats

Romain Thomas

Quarkslab
rthomas@quarkslab.com

March 17, 2018

Abstract

The purpose of the talk is to present instrumentation techniques based on executable file formats. Instrumentation is done with LIEF, a Library to Instrumentation Executable Formats

Keywords: Static Instrumentation, Android, ELF, Hooking, Executable Formats.

1. Introduction

The first layer of information when analysing a binary is the format in which it is wrapped. An executable format provides information for the operating system in order to load and execute it. Among these information we can have the address of the first instruction to execute (entry point), external functions used by the executable, libraries linked . . . .

Although there exists plenty of frameworks to disassemble or emulate binaries, regarding to executable file formats tools are either specific to a format (e.g. `pelib`) or to a language (e.g. `pyelftool`). In the proposed talk we would like to introduce `LIEF`, a cross platform library to analyze and instrument executable file formats.

2. LIEF

As mentioned in the introduction, many projects need to parse executable file formats but don't use a standard library and usually re-implement their own parser. Moreover, these parsers are usually bound to one language.
On Unix system one can find the `objdump` and `objcopy` utilities but they are limited to Unix, the API is not user-friendly and they can't handle malformed ELF files.
LIEF aims to fill this void with the following features:

- **Cross-platform**: Ability to handle executable formats independently of the platform (e.g. analyze PE file on Linux)
- **API**: Provide an API for different languages (`C++, Python, C` . . .)
- **Abstraction**: Provide an abstraction of features that are common in the different formats (e.g. relocations, symbols, . . .) [1]

[1] `https://github.com/lief-project/LIEF/blob/master/examples/python/nm.py`

- **Modification**: Enable to modify these formats.

The figure 1 gives an overview of the architecture.

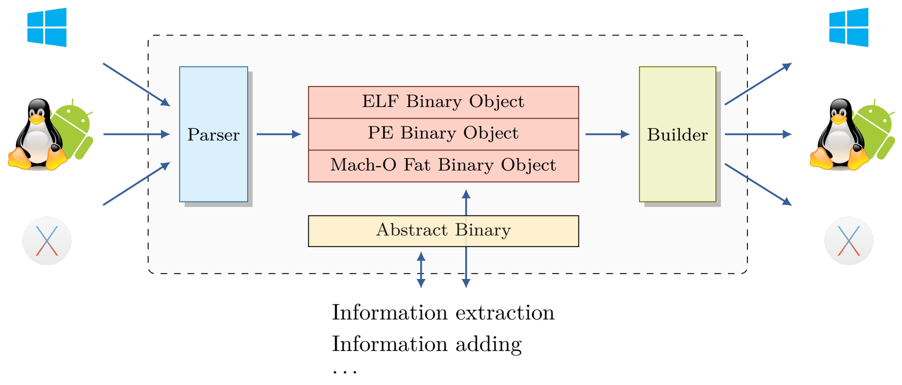


Figure 1: *LIEF Architecture*

Currently LIEF `0.8` is able to parse `ELF, PE, Mach-O` and provide an API for Python, C++ and C languages. Concerning the modification part, the ELF part is very mature, PE part is mature and Mach-O is not mature.

3. Presentation Outline

In the presentation, we would like to introduce LIEF and its use cases in the security field. Here is the plan of the presentation:

1. Introduction of LIEF
2. Static library injection in a Android application
3. Frida injection on **non-rooted** Android devices
4. Hooking on PE and ELF files based on format modification
5. Use of LIEF to enhance fuzzing on closed source binaries
6. Extracting original DEX file from Android OAT files

3.1. LIEF Introduction

In this part we will introduce the project

3.2. Static library injection

Some binary analysis techniques need the *primitive* of injecting code or content within the memory space of a process. The injection can then be used to hook, trace, or patch function, to inspect memory, . . .

As an example Frida[2] has to inject the `frida-agent.so` library to perform hooking and QuarkslaB Dynamic binary Instrumentation (QBDI)[3] injects `libqbdi_tracer.so` to trace the execution flow.

This injection can be done in different ways:

1. Using environment variables like `LD_PRELOAD` or `DYLD_INSERT_LIBRARIES`
2. Using operating system API to inject code directly (e.g. `WriteProcessMemory`, `ptrace`)
3. Using custom kernel modules

Another way is to add a library as a dependency of the executable. Executables are usually linked with shared libraries (e.g. `libc.so`) In the loading phase of the executable, prior jumping on the entry point, the loader maps libraries linked with the executable and call its *contructors*.

Constructors are a set of functions used to initialize internal states of a library or executable. They can be declared in the source code using the `__attribute__((constructors))` attribute.

```
__attribute__((constructor))
void ctor_function(void) {
  ...
}
...
```

If we add a custom library in the executable format having a *constructor* function, this function will be called each times the library is loaded. As the loading is done by the **system**'s loader, it **does not required privileged** environment neither environment variables.

Using LIEF we can do such injection on a ELF binary as follows:

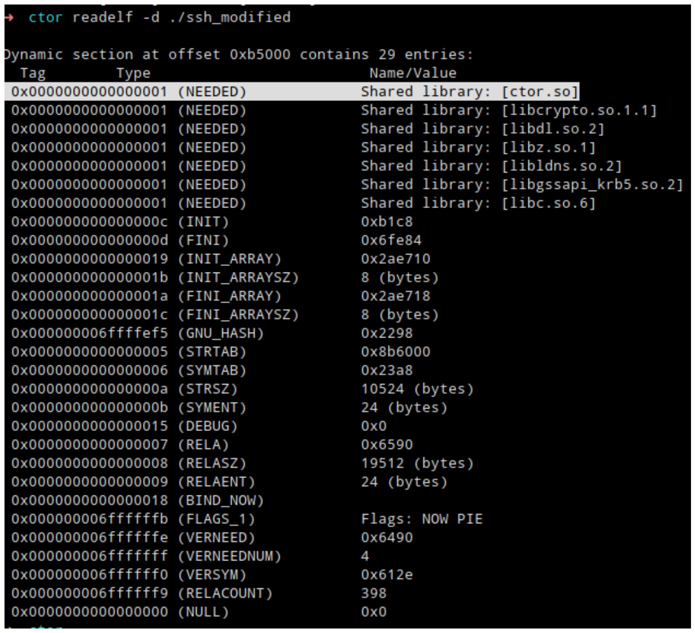
```
import lief
ssh = lief.parse("/usr/bin/ssh")
ssh.add_library("ctor.so")
ssh.write("ssh_modified")
```

For examples, dumping the memory space of the process can be done with this kind of *constructor*:

```
#include <fstream>
#include <iostream>

__attribute__((constructor))
void ctor_function(void) {
  if (std::ifstream ifs{"/proc/self/maps"}) {
    std::string str((std::istreambuf_iterator<char>(ifs)),
                     std::istreambuf_iterator<char>());
    printf("%s\n", str.c_str());
```

[2]https://www.frida.re/
[3]https://qbdi.quarkslab.com/



```
→ ctor readelf -d ./ssh_modified

Dynamic section at offset 0xb5000 contains 29 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [ctor.so]
 0x0000000000000001 (NEEDED)             Shared library: [libcrypto.so.1.1]
 0x0000000000000001 (NEEDED)             Shared library: [libdl.so.2]
 0x0000000000000001 (NEEDED)             Shared library: [libz.so.1]
 0x0000000000000001 (NEEDED)             Shared library: [libldns.so.2]
 0x0000000000000001 (NEEDED)             Shared library: [libgssapi_krb5.so.2]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0xb1c8
 0x000000000000000d (FINI)               0x6fe84
 0x0000000000000019 (INIT_ARRAY)         0x2ae710
 0x000000000000001b (INIT_ARRAYSZ)       8 (bytes)
 0x000000000000001a (FINI_ARRAY)         0x2ae718
 0x000000000000001c (FINI_ARRAYSZ)       8 (bytes)
 0x000000006ffffef5 (GNU_HASH)           0x2298
 0x0000000000000005 (STRTAB)             0x8b6000
 0x0000000000000006 (SYMTAB)             0x23a8
 0x000000000000000a (STRSZ)              10524 (bytes)
 0x000000000000000b (SYMENT)             24 (bytes)
 0x0000000000000015 (DEBUG)              0x0
 0x0000000000000007 (RELA)               0x6590
 0x0000000000000008 (RELASZ)             19512 (bytes)
 0x0000000000000009 (RELAENT)            24 (bytes)
 0x0000000000000018 (BIND_NOW)
 0x000000006ffffffb (FLAGS_1)            Flags: NOW PIE
 0x000000006ffffffe (VERNEED)            0x6490
 0x000000006fffffff (VERNEEDNUM)         4
 0x000000006ffffff0 (VERSYM)             0x612e
 0x000000006ffffff9 (RELACOUNT)          398
 0x0000000000000000 (NULL)               0x0
```

Figure 2: *`ctor.so` is now a dependency of `ssh_modified`*

```
  }
}
```

The result on the `ssh` binary is given below:

```
#  ctor LD_LIBRARY_PATH=. ./ssh_modified
56161cfd5000-56161d083000 r-xp 00000000 fe:01 4377201                    /home/romain/tmp/ctor/ssh_modified
56161d283000-56161d286000 r--p 000ae000 fe:01 4377201                    /home/romain/tmp/ctor/ssh_modified
56161d286000-56161d287000 rw-p 000b1000 fe:01 4377201                    /home/romain/tmp/ctor/ssh_modified
56161d287000-56161d28a000 rw-p 00000000 00:00 0
56161d48a000-56161d48b000 rw-p 000b5000 fe:01 4377201                    /home/romain/tmp/ctor/ssh_modified
56161d88b000-56161d88e000 r--p 000b6000 fe:01 4377201                    /home/romain/tmp/ctor/ssh_modified
56161e0ea000-56161e10b000 rw-p 00000000 00:00 0                          [heap]
7fc0395c1000-7fc0395d4000 r-xp 00000000 fe:00 2492531                    /usr/lib/libresolv-2.26.so
7fc0395d4000-7fc0397d4000 ---p 00013000 fe:00 2492531                    /usr/lib/libresolv-2.26.so
7fc0397d4000-7fc0397d5000 r--p 00013000 fe:00 2492531                    /usr/lib/libresolv-2.26.so
7fc0397d5000-7fc0397d6000 rw-p 00014000 fe:00 2492531                    /usr/lib/libresolv-2.26.so
7fc0397d6000-7fc0397d8000 rw-p 00000000 00:00 0
7fc0397d8000-7fc0397db000 r-xp 00000000 fe:00 2501137                    /usr/lib/libkeyutils.so.1.6
7fc0397db000-7fc0399da000 ---p 00003000 fe:00 2501137                    /usr/lib/libkeyutils.so.1.6
7fc0399da000-7fc0399db000 r--p 00002000 fe:00 2501137                    /usr/lib/libkeyutils.so.1.6
7fc0399db000-7fc0399dc000 rw-p 00003000 fe:00 2501137                    /usr/lib/libkeyutils.so.1.6
7fc0399dc000-7fc0399e7000 r-xp 00000000 fe:00 2501161                    /usr/lib/libkrb5support.so.0.1
7fc0399e7000-7fc039be7000 ---p 0000b000 fe:00 2501161                    /usr/lib/libkrb5support.so.0.1
7fc039be7000-7fc039be8000 r--p 0000b000 fe:00 2501161                    /usr/lib/libkrb5support.so.0.1
7fc039be8000-7fc039be9000 rw-p 0000c000 fe:00 2501161                    /usr/lib/libkrb5support.so.0.1
7fc039be9000-7fc039bec000 r-xp 00000000 fe:00 2500879                    /usr/lib/libcom_err.so.2.1
7fc039bec000-7fc039deb000 ---p 00003000 fe:00 2500879                    /usr/lib/libcom_err.so.2.1
7fc039deb000-7fc039dec000 r--p 00002000 fe:00 2500879                    /usr/lib/libcom_err.so.2.1
7fc039dec000-7fc039ded000 rw-p 00003000 fe:00 2500879                    /usr/lib/libcom_err.so.2.1
7fc039ded000-7fc039e1d000 r-xp 00000000 fe:00 2501170                    /usr/lib/libk5crypto.so.3.1
7fc039e1d000-7fc03a01d000 ---p 00030000 fe:00 2501170                    /usr/lib/libk5crypto.so.3.1
7fc03a01d000-7fc03a01f000 r--p 00030000 fe:00 2501170                    /usr/lib/libk5crypto.so.3.1
7fc03a01f000-7fc03a020000 rw-p 00032000 fe:00 2501170                    /usr/lib/libk5crypto.so.3.1
7fc03a020000-7fc03a0f7000 r-xp 00000000 fe:00 2501179                    /usr/lib/libkrb5.so.3.3
7fc03a0f7000-7fc03a2f7000 ---p 000d7000 fe:00 2501179                    /usr/lib/libkrb5.so.3.3
7fc03a2f7000-7fc03a306000 r--p 000d7000 fe:00 2501179                    /usr/lib/libkrb5.so.3.3
```

```
7fc03a306000-7fc03a308000 rw-p 000e6000 fe:00 2501179                    /usr/lib/libkrb5.so.3.3
7fc03a308000-7fc03a368000 r-xp 00000000 fe:00 2498771                    /usr/lib/libssl.so.1.1
7fc03a368000-7fc03a568000 ---p 00060000 fe:00 2498771                    /usr/lib/libssl.so.1.1
7fc03a568000-7fc03a56d000 r--p 00060000 fe:00 2498771                    /usr/lib/libssl.so.1.1
7fc03a56d000-7fc03a572000 rw-p 00065000 fe:00 2498771                    /usr/lib/libssl.so.1.1
7fc03a572000-7fc03a58b000 r-xp 00000000 fe:00 2492629                    /usr/lib/libpthread-2.26.so
7fc03a58b000-7fc03a78a000 ---p 00019000 fe:00 2492629                    /usr/lib/libpthread-2.26.so
7fc03a78a000-7fc03a78b000 r--p 00018000 fe:00 2492629                    /usr/lib/libpthread-2.26.so
7fc03a78b000-7fc03a78c000 rw-p 00019000 fe:00 2492629                    /usr/lib/libpthread-2.26.so
7fc03a78c000-7fc03a790000 rw-p 00000000 00:00 0
7fc03a790000-7fc03a7a6000 r-xp 00000000 fe:00 2510845                    /usr/lib/libgcc_s.so.1
7fc03a7a6000-7fc03a9a5000 ---p 00016000 fe:00 2510845                    /usr/lib/libgcc_s.so.1
7fc03a9a5000-7fc03a9a6000 r--p 00015000 fe:00 2510845                    /usr/lib/libgcc_s.so.1
7fc03a9a6000-7fc03a9a7000 rw-p 00016000 fe:00 2510845                    /usr/lib/libgcc_s.so.1
7fc03a9a7000-7fc03aaf2000 r-xp 00000000 fe:00 2492535                    /usr/lib/libm-2.26.so
7fc03aaf2000-7fc03acf1000 ---p 0014b000 fe:00 2492535                    /usr/lib/libm-2.26.so
7fc03acf1000-7fc03acf2000 r--p 0014a000 fe:00 2492535                    /usr/lib/libm-2.26.so
7fc03acf2000-7fc03acf3000 rw-p 0014b000 fe:00 2492535                    /usr/lib/libm-2.26.so
7fc03acf3000-7fc03ae6b000 r-xp 00000000 fe:00 2510804                    /usr/lib/libstdc++.so.6.0.24
7fc03ae6b000-7fc03b06a000 ---p 00178000 fe:00 2510804                    /usr/lib/libstdc++.so.6.0.24
7fc03b06a000-7fc03b076000 r--p 00177000 fe:00 2510804                    /usr/lib/libstdc++.so.6.0.24
7fc03b076000-7fc03b077000 rw-p 00183000 fe:00 2510804                    /usr/lib/libstdc++.so.6.0.24
7fc03b077000-7fc03b07a000 rw-p 00000000 00:00 0
7fc03b07a000-7fc03b228000 r-xp 00000000 fe:00 2492604                    /usr/lib/libc-2.26.so
7fc03b228000-7fc03b427000 ---p 001ae000 fe:00 2492604                    /usr/lib/libc-2.26.so
7fc03b427000-7fc03b42b000 r--p 001ad000 fe:00 2492604                    /usr/lib/libc-2.26.so
7fc03b42b000-7fc03b42d000 rw-p 001b1000 fe:00 2492604                    /usr/lib/libc-2.26.so
7fc03b42d000-7fc03b431000 rw-p 00000000 00:00 0
7fc03b431000-7fc03b47c000 r-xp 00000000 fe:00 2501198                    /usr/lib/libgssapi_krb5.so.2.2
7fc03b47c000-7fc03b67c000 ---p 0004b000 fe:00 2501198                    /usr/lib/libgssapi_krb5.so.2.2
7fc03b67c000-7fc03b67e000 r--p 0004b000 fe:00 2501198                    /usr/lib/libgssapi_krb5.so.2.2
7fc03b67e000-7fc03b67f000 rw-p 0004d000 fe:00 2501198                    /usr/lib/libgssapi_krb5.so.2.2
7fc03b67f000-7fc03b6d7000 r-xp 00000000 fe:00 2534584                    /usr/lib/libldns.so.2.0.0
7fc03b6d7000-7fc03b8d6000 ---p 00058000 fe:00 2534584                    /usr/lib/libldns.so.2.0.0
7fc03b8d6000-7fc03b8d8000 r--p 00057000 fe:00 2534584                    /usr/lib/libldns.so.2.0.0
7fc03b8d8000-7fc03b8dc000 rw-p 00059000 fe:00 2534584                    /usr/lib/libldns.so.2.0.0
7fc03b8dc000-7fc03b8f2000 r-xp 00000000 fe:00 2496881                    /usr/lib/libz.so.1.2.11
7fc03b8f2000-7fc03baf1000 ---p 00016000 fe:00 2496881                    /usr/lib/libz.so.1.2.11
7fc03baf1000-7fc03baf2000 r--p 00015000 fe:00 2496881                    /usr/lib/libz.so.1.2.11
7fc03baf2000-7fc03baf3000 rw-p 00016000 fe:00 2496881                    /usr/lib/libz.so.1.2.11
7fc03baf3000-7fc03baf6000 r-xp 00000000 fe:00 2492537                    /usr/lib/libdl-2.26.so
7fc03baf6000-7fc03bcf5000 ---p 00003000 fe:00 2492537                    /usr/lib/libdl-2.26.so
7fc03bcf5000-7fc03bcf6000 r--p 00002000 fe:00 2492537                    /usr/lib/libdl-2.26.so
7fc03bcf6000-7fc03bcf7000 rw-p 00003000 fe:00 2492537                    /usr/lib/libdl-2.26.so
7fc03bcf7000-7fc03bf4a000 r-xp 00000000 fe:00 2498770                    /usr/lib/libcrypto.so.1.1
7fc03bf4a000-7fc03c149000 ---p 00253000 fe:00 2498770                    /usr/lib/libcrypto.so.1.1
7fc03c149000-7fc03c167000 r--p 00252000 fe:00 2498770                    /usr/lib/libcrypto.so.1.1
7fc03c167000-7fc03c171000 rw-p 00270000 fe:00 2498770                    /usr/lib/libcrypto.so.1.1
7fc03c171000-7fc03c174000 rw-p 00000000 00:00 0
7fc03c174000-7fc03c177000 r-xp 00000000 fe:01 4377199                    /home/romain/tmp/ctor/ctor.so
7fc03c177000-7fc03c376000 ---p 00003000 fe:01 4377199                    /home/romain/tmp/ctor/ctor.so
7fc03c376000-7fc03c377000 r--p 00002000 fe:01 4377199                    /home/romain/tmp/ctor/ctor.so
7fc03c377000-7fc03c378000 rw-p 00003000 fe:01 4377199                    /home/romain/tmp/ctor/ctor.so
7fc03c378000-7fc03c39d000 r-xp 00000000 fe:00 2492605                    /usr/lib/ld-2.26.so
7fc03c54a000-7fc03c554000 rw-p 00000000 00:00 0
7fc03c59a000-7fc03c59c000 rw-p 00000000 00:00 0
7fc03c59c000-7fc03c59d000 r--p 00024000 fe:00 2492605                    /usr/lib/ld-2.26.so
7fc03c59d000-7fc03c59e000 rw-p 00025000 fe:00 2492605                    /usr/lib/ld-2.26.so
7fc03c59e000-7fc03c59f000 rw-p 00000000 00:00 0
7ffdc25c1000-7ffdc25e3000 rw-p 00000000 00:00 0                          [stack]
7ffdc25f3000-7ffdc25f6000 r--p 00000000 00:00 0                          [vvar]
7ffdc25f6000-7ffdc25f8000 r-xp 00000000 00:00 0                          [vdso]

usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-b bind_address] [-c cipher_spec]
           [-D [bind_address:]port] [-E log_file] [-e escape_char]
           [-F configfile] [-I pkcs11] [-i identity_file]
           [-J [user@]host[:port]] [-L address] [-l login_name] [-m mac_spec]
           [-O ctl_cmd] [-o option] [-p port] [-Q query_option] [-R address]
           [-S ctl_path] [-W host:port] [-w local_tun[:remote_tun]]
           [user@]hostname [command]
```

3.3. Frida Injection

We can use the technique introduced in the previous section to inject `frida-gadget.so` in an APK which has at least one native library. Doing so enables to use Frida on non rooted device an bypassing Frida detection based on library name.
During the talk, this part will be followed with a demonstration on a real Android application.

3.4. Hooking

In this part we plan to expose two (well-known) techniques to hook functions.

3.4.1 ELF

For the ELF format we will present the `plt/got` infection technique with LIEF. Details of the technique is available in the documentation of the project[5] and can be summarized with the following snippet:

```
import lief

binary = lief.parse(...)
binary.patch_pltgot('memcmp', my_memcmp_addr)
binary.write("binary_modified")
```

3.4.2 PE

Concerning the PE part, we will present the IAT patching technique. Details of the technique is available in the documentation of the project[6] and can be summarized with the following snippet:

```
import lief

binary = lief.parse(...)
...
binary.hook_function("IsDebuggerPresent", my_address)
builder = lief.PE.Builder(binary)
...
builder.write("binary_modified")
```

4. LIEF for fuzzing

As explained in this tutorial[4] and this blog post[3], exporting internal function in an executable enables to fuzz the targeted function. This technique can also be used to export functions from Android native libraries that are not a part of the JNI.

In the talk, we would like to present this technique by a demo with AFL that will fuzz a *non-exported* function from a closed source binary.

5. Android OAT files

Since Lollipop (Android 5.0), Android moves from the Dalvik runtime to the ART runtime. ART runtime makes use of ahead of time compilation to transform DEX files into OAT files.

OAT files are basically ELF shared libraries embedding a custom Android format (OAT format). The OAT format internal has been presented across various talks[4][5] and some attacks [8] [7] requires to modify some parts of these OAT file.

The next version of LIEF[6] will come with new formats related to Android internal: `OAT`, `DEX`, `VDEX`, `ART`.

These formats evolve for each version of Android and the existing tools usually have the following limitations:

- **Android core tools**[7]: Doesn't support format versions other than the one used during the compilation and there is no API.
- **Dextra**: Closed source, doesn't support VDEX, bug in the OAT parser, no API
- **Sami/Baksmali**[8]: Require to have the `framework/` directory, doesn't produce a `.dex` file

In the same manner as it has been done for the ELF, PE and Mach-O format, we are integrating these formats in LIEF[9].

Currently we are supporting these ranges of versions associated with formats:

1. **OAT** format is supported from `OAT 64` (Android M 6.0) up to `OAT 131` (Android O 8.1)
2. **DEX** format is supported from `DEX 35` up to `DEX 39`
3. **VDEX** format is supported from `VDEX 6` up to `VDEX 10` (Android 0reo 8.X)
4. **ART** format is supported from `ART 17` (Android M 6.0) up to `ART 46` (Android O 8.1)

`LIEF 0.9.0` will only provide an API to read these formats, not to modify them. Future versions will enable to modify it.

Some applications embedded in constructor ROM remove the original DEX file. For example, in the Samsung Galaxy S8, the application `SecSettings2.apk` located in the `system/priv-app` directory doesn't contain the original DEX file:

```
# unzip -l system/priv-app/SecSettings2/SecSettings2.apk|grep -c "\.dex"
0
```

As `SecSettings2` is a part of the Samsung framework, the ROM embedded the OAT directly in the `oat` folder

[4] `http://www.blackhat.com/docs/asia-15/materials/asia-15-Sabanal-Hiding-Behind-ART-wp.pdf`
[5] `http://newandroidbook.com/files/ArtOfDalvik.pdf`
[6] LIEF `0.9` is planned for June
[7] oatdump, dexdump, ...
[8] `https://github.com/JesusFreke/smali/wiki/DeodexInstructions`
[9] Including `C++` and Python API, documentation, examples, ...

```
# ls SecSettings2/oat/arm64
SecSettings2.odex
```

Actually the `SecSettings2.odex` is an OAT file and the original DEX file is still present in a section of the OAT format **but**. However the given DEX file is *optimized* with special (undocumented) Dalvik instructions. It makes decompilers and common DEX analysis tools inefficient on this kind of DEX file.
As an example one can use `dextra`[1] to extract the DEX file and run it in JEB:

```
# dextra.ELF64 -dextract ./SecSettings2.odex
N OAT file (079)
OFF: cda
Dex header @0x7f3c50bc2d1c (7081 classes) at 0xd1c: /system/priv-app/SecSettings2/SecSettings2.apk
 Written to system@priv-app@SecSettings2@SecSettings2.apk@classes.dex
Location Length: 46
# jeb ./system@priv-app@SecSettings2@SecSettings2.apk@classes.dex
...
Invalid DEX checksum (expected=DAF16B6E, actual=9327E1FD)
Invalid DEX signature (expected=CD A8 A2 82 39 8C FA 82 7D E1 7C 4A B8 E9 C1 6E 72 97 64 A3 , actual=DA 39 A3 EE 5E
  ↪  6B 4B 0D 32 55 BF EF 95 60 18 90 AF D8 07 09 )
Parsing error at Lcn/com/xy/sms/sdk/service/c/g;-><clinit> (+14h/@Ah): Reserved opcode
Parsing error at Lcn/com/xy/sms/sdk/service/c/g;-><clinit> (+16h/@Bh): Out of bytecode boundaries
Parsing error at Lcn/com/xy/sms/sdk/service/msgurlservice/MsgUrlService;-><clinit> (+14h/@Ah): Reserved opcode
Parsing error at Lcn/com/xy/sms/sdk/service/msgurlservice/MsgUrlService;-><clinit> (+16h/@Bh): Out of bytecode
  ↪  boundaries
Parsing error at Lcn/com/xy/sms/sdk/smsmessage/BusinessSmsMessage;-><clinit> (+6h/@3h): Reserved opcode
Parsing error at Lcn/com/xy/sms/sdk/smsmessage/BusinessSmsMessage;-><clinit> (+8h/@4h): Out of bytecode boundaries
...
```

These errors came from the optimized instructions that are not recognized by JEB. Making methods non-decompilable as shown in figure 3
With LIEF, we can extract the **original** and de-optimized DEX file as follows:

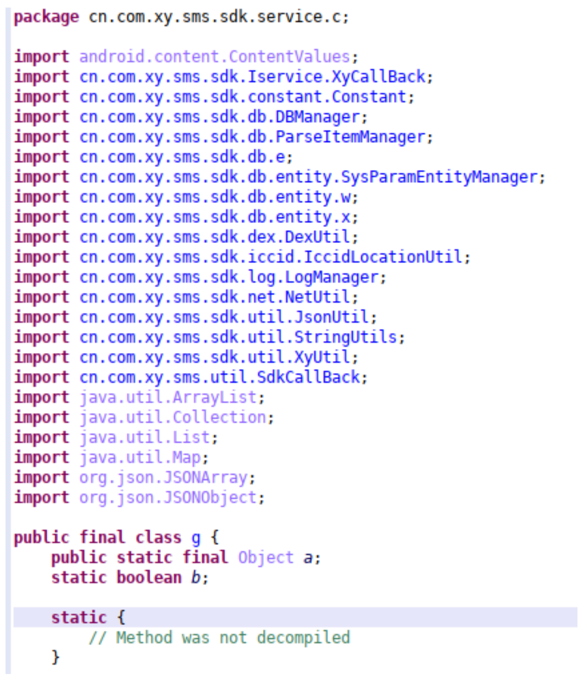
```python
import lief

SecSettings2 = lief.parse("SecSettings2.odex") # LIEF OAT object
dex = SecSettings2.dex_files[0]                # No multi-dex in this app
dex.save("classes.dex", deoptimize=True)       # Save and de-optimize it
```

Use of JEB on `classes.dex` doesn't throw errors and the figure 4 shows the method decompiled.

In fact starting from **Android 7.0.0**, information to de-optimize the DEX files are located in the *mapping table* section of the OAT file or in the *quickening information* section of the VDEX file. The diff exposed in the listing 1 shows the lines enabling to recover the original instruction.

Therefore we can use the `QuickenedInfo` structure to patch the optimized DEX and regenerate a de-optimized one.



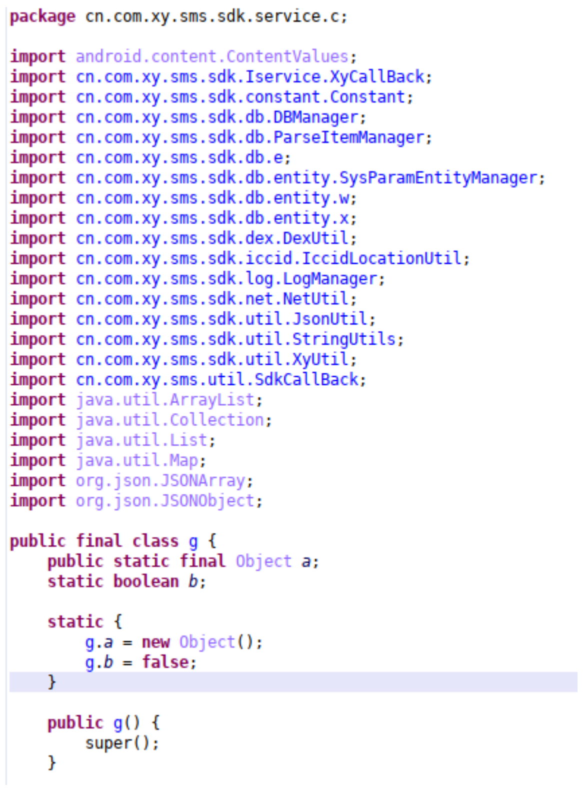
```java
package cn.com.xy.sms.sdk.service.c;

import android.content.ContentValues;
import cn.com.xy.sms.sdk.Iservice.XyCallBack;
import cn.com.xy.sms.sdk.constant.Constant;
import cn.com.xy.sms.sdk.db.DBManager;
import cn.com.xy.sms.sdk.db.ParseItemManager;
import cn.com.xy.sms.sdk.db.e;
import cn.com.xy.sms.sdk.db.entity.SysParamEntityManager;
import cn.com.xy.sms.sdk.db.entity.w;
import cn.com.xy.sms.sdk.db.entity.x;
import cn.com.xy.sms.sdk.dex.DexUtil;
import cn.com.xy.sms.sdk.iccid.IccidLocationUtil;
import cn.com.xy.sms.sdk.log.LogManager;
import cn.com.xy.sms.sdk.net.NetUtil;
import cn.com.xy.sms.sdk.util.JsonUtil;
import cn.com.xy.sms.sdk.util.StringUtils;
import cn.com.xy.sms.sdk.util.XyUtil;
import cn.com.xy.sms.util.SdkCallBack;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import org.json.JSONArray;
import org.json.JSONObject;

public final class g {
    public static final Object a;
    static boolean b;

    static {
        // Method was not decompiled
    }
```

Figure 3: *Static constructor can't be decompiled due to optimizations*



```java
package cn.com.xy.sms.sdk.service.c;

import android.content.ContentValues;
import cn.com.xy.sms.sdk.Iservice.XyCallBack;
import cn.com.xy.sms.sdk.constant.Constant;
import cn.com.xy.sms.sdk.db.DBManager;
import cn.com.xy.sms.sdk.db.ParseItemManager;
import cn.com.xy.sms.sdk.db.e;
import cn.com.xy.sms.sdk.db.entity.SysParamEntityManager;
import cn.com.xy.sms.sdk.db.entity.w;
import cn.com.xy.sms.sdk.db.entity.x;
import cn.com.xy.sms.sdk.dex.DexUtil;
import cn.com.xy.sms.sdk.iccid.IccidLocationUtil;
import cn.com.xy.sms.sdk.log.LogManager;
import cn.com.xy.sms.sdk.net.NetUtil;
import cn.com.xy.sms.sdk.util.JsonUtil;
import cn.com.xy.sms.sdk.util.StringUtils;
import cn.com.xy.sms.sdk.util.XyUtil;
import cn.com.xy.sms.util.SdkCallBack;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import org.json.JSONArray;
import org.json.JSONObject;

public final class g {
    public static final Object a;
    static boolean b;

    static {
        g.a = new Object();
        g.b = false;
    }

    public g() {
        super();
    }
```

Figure 4: *Static constructor decompiled*

```
+struct QuickenedInfo {
+  QuickenedInfo(uint32_t pc, uint16_t index) : dex_pc(pc), dex_member_index(index) {}
+
+  uint32_t dex_pc;
+  uint16_t dex_member_index;
+};
+
...
@@ -248,6 +267,7 @@ void DexCompiler::CompileInstanceFieldAccess(Instruction* inst,
     inst->SetOpcode(new_opcode);
     // Replace field index by field offset.
     inst->SetVRegC_22c(static_cast<uint16_t>(field_offset.Int32Value()));
+    quickened_info_.push_back(QuickenedInfo(dex_pc, field_idx));
   }
 }

@@ -287,24 +307,69 @@ void DexCompiler::CompileInvokeVirtual(Instruction* inst, uint32_t dex_pc,
       } else {
         inst->SetVRegB_35c(static_cast<uint16_t>(vtable_idx));
       }
+      quickened_info_.push_back(QuickenedInfo(dex_pc, method_idx));
     }
   }
 }
```

Listing 1: *Diff of the file* `compiler/dex/dex_to_dex_compiler.cc` *between Android 6.0.1 and Android 7.0.0*

In the talk we plan to explain the process to extract and de-optimize DEX from OAT/VDEX files. It will be followed with a similar demonstration to the one with `SecSettings2.apk`.

6. Why this talk ?

With this talk we would like to demonstrate how executable file formats can be instrumented with LIEF and how it can be useful in various scenarios from a Windows PE to an Android OAT file. Moreover integrity check is usually performed on assembly code and not on the executable format itself.

Note: If needed, we can provide to the reviewers a preview of LIEF for formats OAT, VDEX, DEX, and ART.

7. LIEF references

- Project website: `https://lief.quarkslab.com/` or `https://lief-project.github.io`
- Github: `https://github.com/lief-project/LIEF`
- Twitter: @LIEF_Project
- Documentation: `https://lief-project.github.io/doc/latest/`
- Blog posts:
  - `https://blog.quarkslab.com/have-fun-with-lief-and-executable-formats.html`

- https://blog.quarkslab.com/lief-library-to-instrument-executable-formats.html

References

[1] Dextra from Jonathan Levin `http://newandroidbook.com/tools/dextra.html`

[2] Bot vs. Bot for Evading Machine Learning Malware Detection `https://www.blackhat.com/us-17/briefings/schedule/index.html#bot-vs-bot-for-evading-machine-learning-malware-detection-6461`

[3] Fuzzing arbitrary functions in ELF binaries `https://blahcat.github.io/2018/03/11/fuzzing-arbitrary-functions-in-elf-binaries/`

[4] Transforming an ELF executable into a library `https://lief-project.github.io/doc/latest/tutorials/08_elf_bin2lib.html`

[5] Infecting the plt/got `https://lief-project.github.io/doc/latest/tutorials/05_elf_infect_plt_got.html`

[6] PE hooking based on the Import Address Table `https://lief-project.github.io/doc/latest/tutorials/06_pe_hooking.html`

[7] Inside Android's SafetyNet Attestation: Attack and Defense `https://www.mulliner.org/collin/publications/inside_safetynet_attestation_attacks_and_defense_mulliner2017_ekoparty.pdf`

[8] How Samsung Secures Your Wallet & How To Break It `https://www.blackhat.com/docs/eu-17/materials/eu-17-Ma-How-Samsung-Secures-Your-Wallet-And-How-To-Break-It.pdf`