Romain Thomas - rthomas@quarkslab.com

Static instrumentation based on executable file formats

quarkslab
SECURING EVERY BIT OF YOUR DATA

About

- Romain Thomas - Security engineer at Quarkslab
- Working on various topics: Android, (de)obfuscation, software protection and reverse engineering
- Author of LIEF

Executable Formats

Executable Formats: Overview

Executable Formats

- First layer of information when analysing a binary

[1] entrypoint, libraries, ...

Executable Formats

- First layer of information when analysing a binary
- Provide metadata[1] used by the operating system to load the binary.

[1] entrypoint, libraries, ...

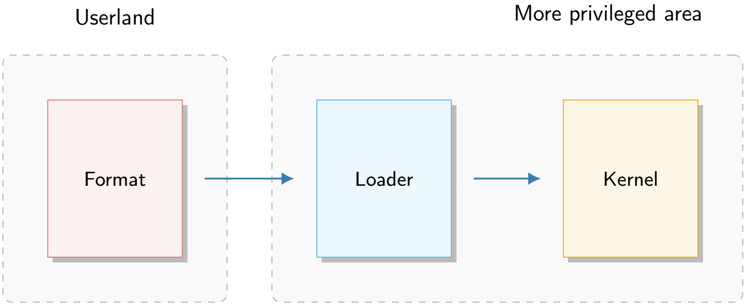
Executable Formats

- **OSX / iOS**: Mach-O
- **Linux**: ELF
- **Windows**: PE
- **Android**: ELF, OAT

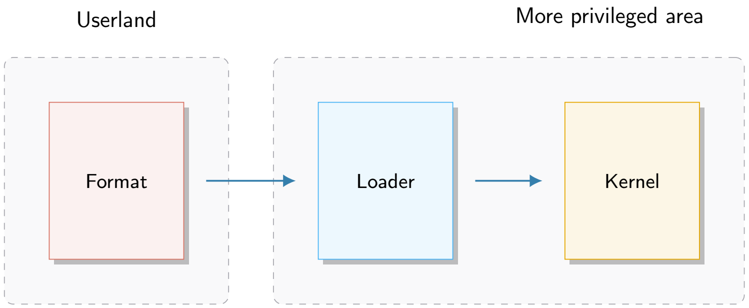
Executable Formats

Why modify formats ?

Executable Formats

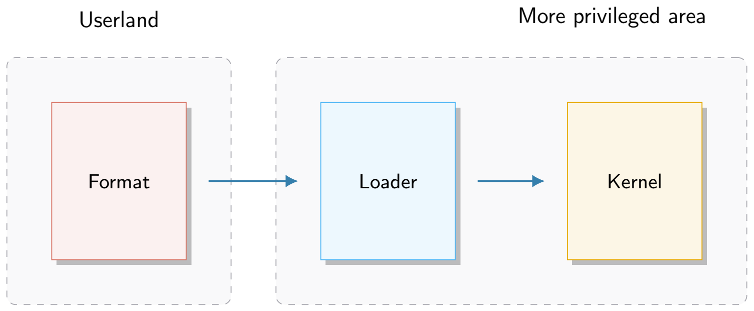


Executable Formats



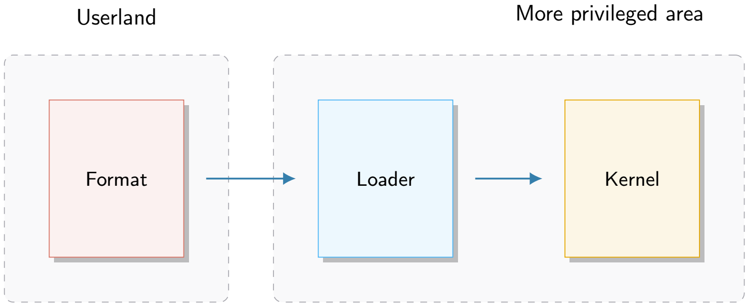
Perform relocations

Executable Formats



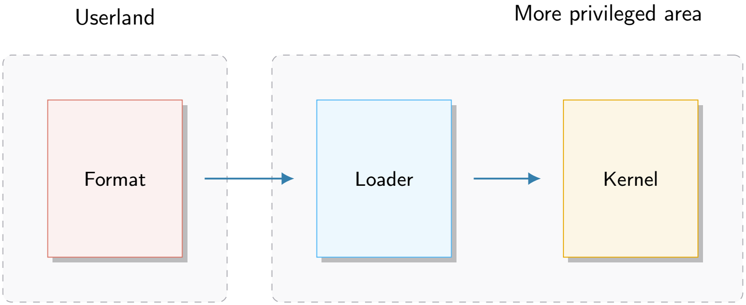
Load shared libraries

Executable Formats



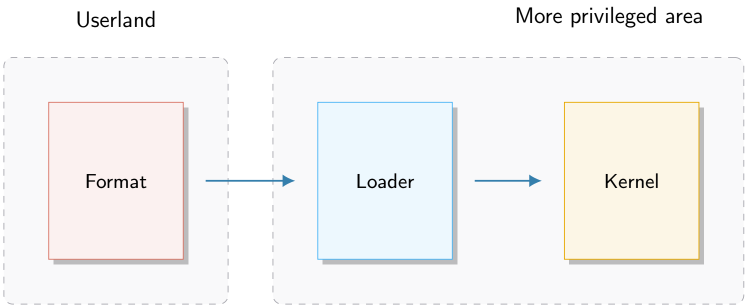
Create Process

Executable Formats



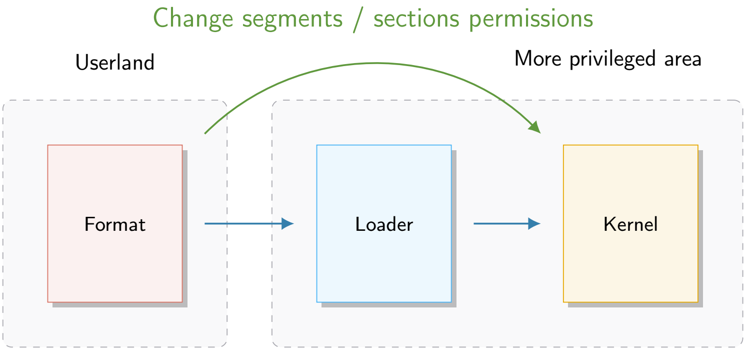
Map content

Executable Formats

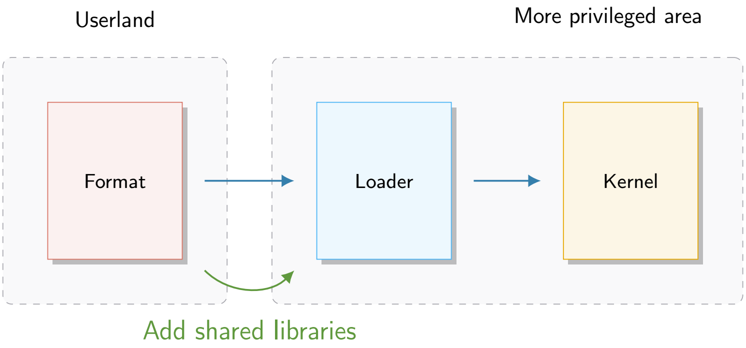


Set permissions

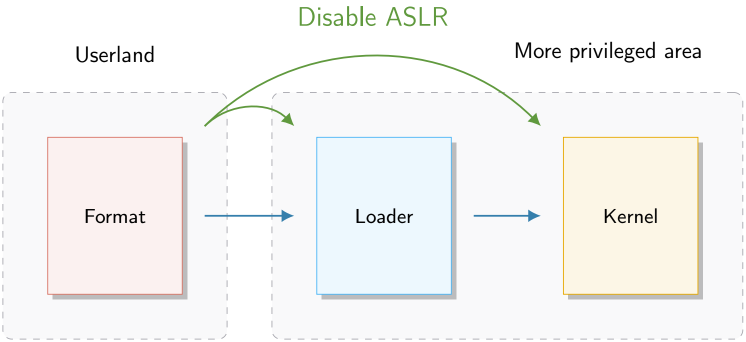
Executable Formats



Executable Formats



Executable Formats



Executable Formats

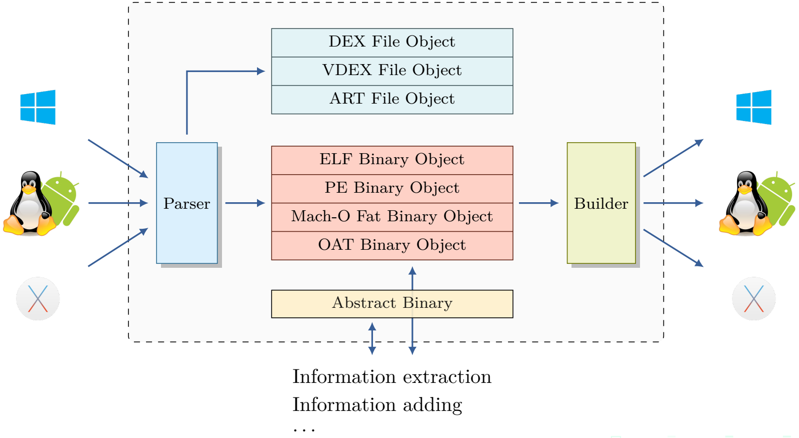
LIEF: Library to Instrument Executable Formats

LIEF

- One library to deal with ELF, PE, Mach-O
- Core in C++
- Bindings for different languages: Python, C [2], ...
- Enable modification on these formats
- User friendly API

[2] C binding is not as mature as Python and C++

Executable Formats



Executable Formats

```python
import lief

target = lief.parse("ELF/PE/Mach-O/OAT")

print(target.entrypoint)
```

Executable Formats

```python
import lief

target = lief.parse("ELF/PE/Mach-O/OAT")

for section in target.sections:
  print(section.virtual_address)
  process(section.content)
```

Executable Formats

```python
import lief

target = lief.parse("some.exe")

target.tls.callbacks.append(0x....)

target.write("new.exe")
```

Executable Formats

```
import lief

target = lief.parse(...)

section = lief.ELF.Section(".text2")
section.content = [0x90] * 0x1000

target += section

target.write("new.elf")
```

Executable Formats

Next parts introduce interesting modifications on formats:

- Hooking
- Exporting *hidden* functions
- Code injection through shared libraries

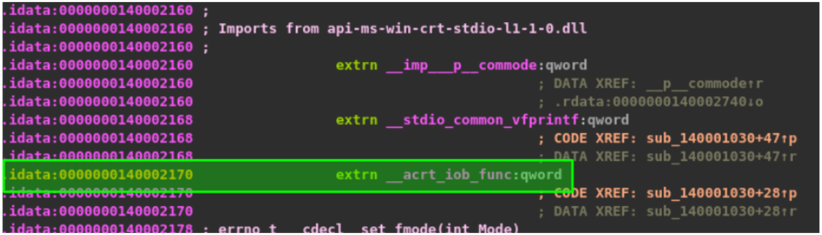
PE Hooking

PE Hooking

PE Hooking

Regarding to PE files, LIEF enables to rebuild the import table **elsewhere** in the binary so that one can add new functions, new libraries or patch the Import Address Table.

Example



```
.idata:0000000140002160 ;
.idata:0000000140002160 ; Imports from api-ms-win-crt-stdio-l1-1-0.dll
.idata:0000000140002160 ;
.idata:0000000140002160                  extrn __imp___p__commode:qword
.idata:0000000140002160                                           ; DATA XREF: __p__commode↑r
.idata:0000000140002160                                           ; .rdata:0000000140002740↓o
.idata:0000000140002168                  extrn __stdio_common_vfprintf:qword
.idata:0000000140002168                                           ; CODE XREF: sub_140001030+47↑p
.idata:0000000140002168                                           ; DATA XREF: sub_140001030+47↑r
.idata:0000000140002170                  extrn __acrt_iob_func:qword
.idata:0000000140002170                                           ; CODE XREF: sub_140001030+28↑p
.idata:0000000140002170                                           ; DATA XREF: sub_140001030+28↑r
.idata:0000000140002178 ; errno_t __cdecl _set_fmode(int Mode)
```

Figure – Original IAT

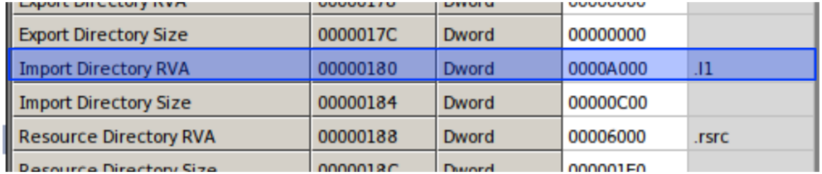
Example

The following code patch the IAT entry of `__acrt_iob_func` with a trampoline to the function `0x140008000`

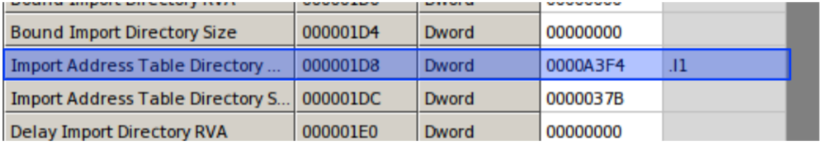
```
pe = lief.parse("some.exe")
pe.hook_function("__acrt_iob_func", 0x140008000)
```

```
builder = lief.PE.Builder(pe)
builder.build_imports(True).patch_imports(True)
builder.build()
builder.write("hooked.exe")
```

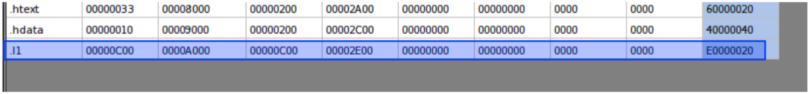
Example



Export Directory Size	0000017C	Dword	00000000	
Import Directory RVA	00000180	Dword	0000A000	.I1
Import Directory Size	00000184	Dword	00000C00	
Resource Directory RVA	00000188	Dword	00006000	.rsrc

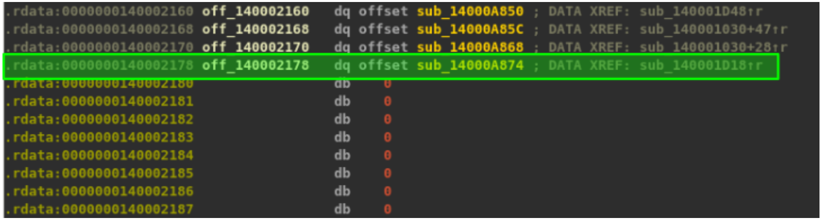


Bound Import Directory Size	000001D4	Dword	00000000	
Import Address Table Directory ...	000001D8	Dword	0000A3F4	.I1
Import Address Table Directory S...	000001DC	Dword	0000037B	
Delay Import Directory RVA	000001E0	Dword	00000000	



.htext	00000033	00008000	00000200	00002A00	00000000	00000000	0000	0000	60000020
.hdata	00000010	00009000	00000200	00002C00	00000000	00000000	0000	0000	40000040
.I1	00000C00	0000A000	00000C00	00002E00	00000000	00000000	0000	0000	E0000020

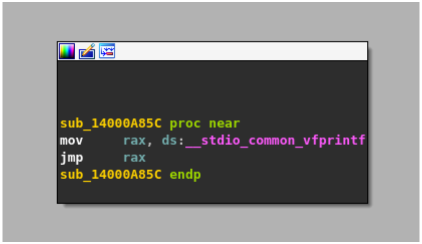
Example



```
.rdata:0000000140002160 off_140002160   dq offset sub_14000A850 ; DATA XREF: sub_140001D48↑r
.rdata:0000000140002168 off_140002168   dq offset sub_14000A85C ; DATA XREF: sub_140001030+47↑r
.rdata:0000000140002170 off_140002170   dq offset sub_14000A868 ; DATA XREF: sub_140001030+28↑r
.rdata:0000000140002178 off_140002178   dq offset sub_14000A874 ; DATA XREF: sub_140001D18↑r
.rdata:0000000140002180                 db    0
.rdata:0000000140002181                 db    0
.rdata:0000000140002182                 db    0
.rdata:0000000140002183                 db    0
.rdata:0000000140002184                 db    0
.rdata:0000000140002185                 db    0
.rdata:0000000140002186                 db    0
.rdata:0000000140002187                 db    0
```

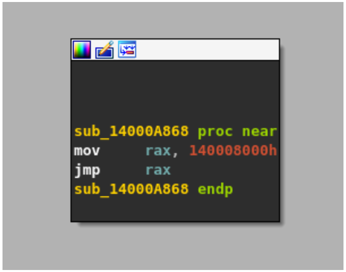
Figure – Original IAT patched with trampoline functions

Example



```
sub_14000A85C proc near
mov     rax, ds:__stdio_common_vfprintf
jmp     rax
sub_14000A85C endp
```

Figure – Trampoline for non-hooked function



```
sub_14000A868 proc near
mov     rax, 140008000h
jmp     rax
sub_14000A868 endp
```

Figure – Trampoline for hooked function

Example

Limitations

This method only works if accesses to the IAT are performed with *call* instructions. Especially it doesn't if there is `lea` on the original IAT

ELF Hooking

Regarding to ELF files, hooking can be done with a patch of the plt/got.

ELF plt/got

.text

```
...
400637: jmp 400480 <memcmp@plt>
...
```

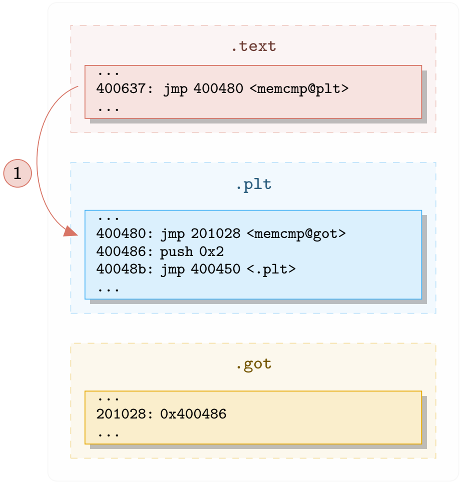
.plt

```
...
400480: jmp 201028 <memcmp@got>
400486: push 0x2
40048b: jmp 400450 <.plt>
...
```

.got

```
...
201028: 0x400486
...
```

ELF plt/got

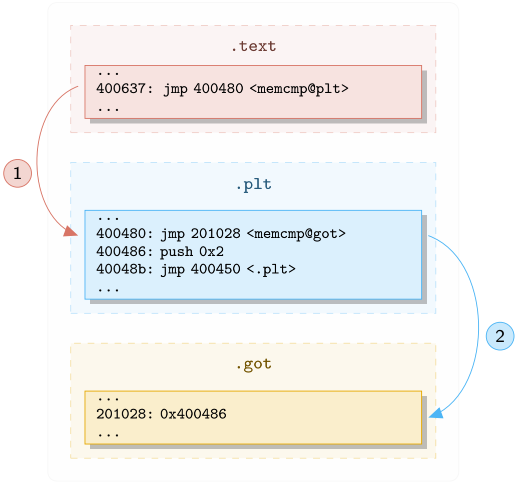


```
...
400637: jmp 400480 <memcmp@plt>
...
```

```
...
400480: jmp 201028 <memcmp@got>
400486: push 0x2
40048b: jmp 400450 <.plt>
...
```

```
...
201028: 0x400486
...
```

ELF plt/got

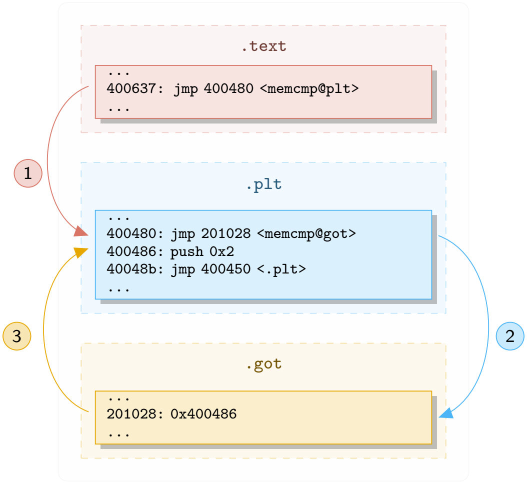


```
...
400637: jmp 400480 <memcmp@plt>
...
```

```
...
400480: jmp 201028 <memcmp@got>
400486: push 0x2
40048b: jmp 400450 <.plt>
...
```

```
...
201028: 0x400486
...
```

ELF plt/got

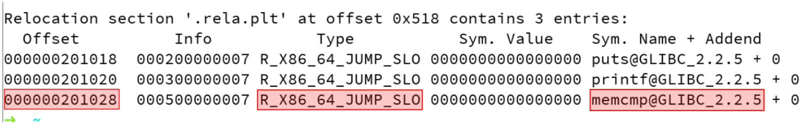


```
...
400637: jmp 400480 <memcmp@plt>
...
```

```
...
400480: jmp 201028 <memcmp@got>
400486: push 0x2
40048b: jmp 400450 <.plt>
...
```

```
...
201028: 0x400486
...
```

ELF plt/got



```
Relocation section '.rela.plt' at offset 0x518 contains 3 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000201018  000200000007 R_X86_64_JUMP_SLO 0000000000000000 puts@GLIBC_2.2.5 + 0
000000201020  000300000007 R_X86_64_JUMP_SLO 0000000000000000 printf@GLIBC_2.2.5 + 0
000000201028  000500000007 R_X86_64_JUMP_SLO 0000000000000000 memcmp@GLIBC_2.2.5 + 0
→  ~
```

Figure – Relocations associated with plt/got

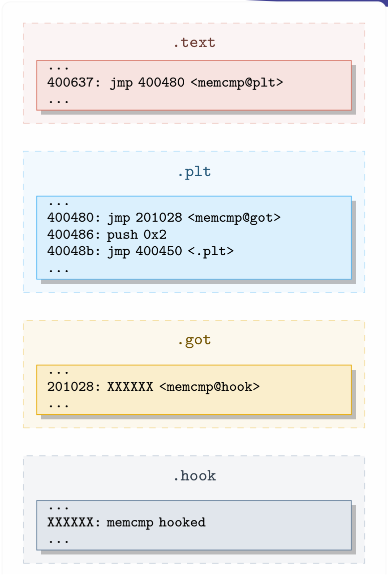
ELF plt/got

```python
import lief
elf = lief.parse("some_elf")

elf.patch_pltgot("memcmp", 0xAAAAAAAAA)

elf.write("elf_modified")
```

ELF plt/got



.text

```
...
400637: jmp 400480 <memcmp@plt>
...
```

.plt

```
...
400480: jmp 201028 <memcmp@got>
400486: push 0x2
40048b: jmp 400450 <.plt>
...
```

.got

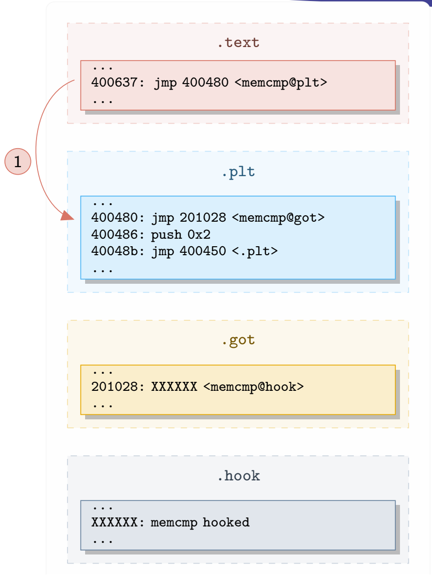
```
...
201028: XXXXXX <memcmp@hook>
...
```

.hook

```
...
XXXXXX: memcmp hooked
...
```

https://lief.quarkslab.com/pts18/demo1

ELF plt/got



```
...
400637: jmp 400480 <memcmp@plt>
...
```

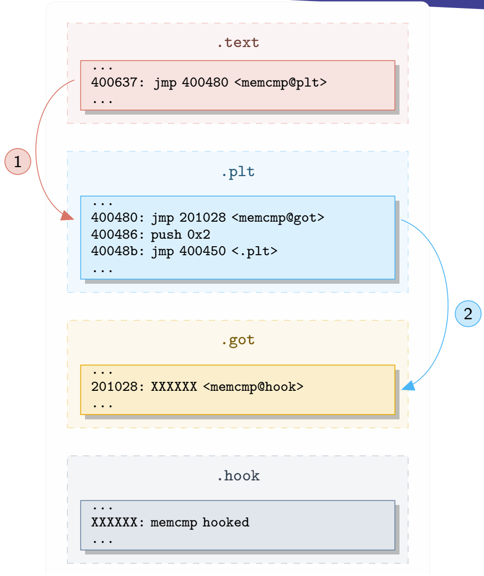
```
...
400480: jmp 201028 <memcmp@got>
400486: push 0x2
40048b: jmp 400450 <.plt>
...
```

```
...
201028: XXXXXX <memcmp@hook>
...
```

```
...
XXXXXX: memcmp hooked
...
```

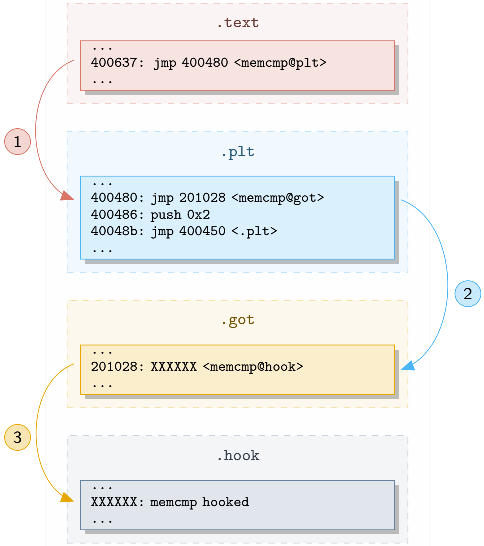
https://lief.quarkslab.com/pts18/demo1

ELF plt/got


.text
...
400637: jmp 400480 <memcmp@plt>
...
1
.plt
...
400480: jmp 201028 <memcmp@got>
400486: push 0x2
40048b: jmp 400450 <.plt>
...
2
.got
...
201028: XXXXXX <memcmp@hook>
...
.hook
...
XXXXXX: memcmp hooked
...

https://lief.quarkslab.com/pts18/demo1

ELF plt/got



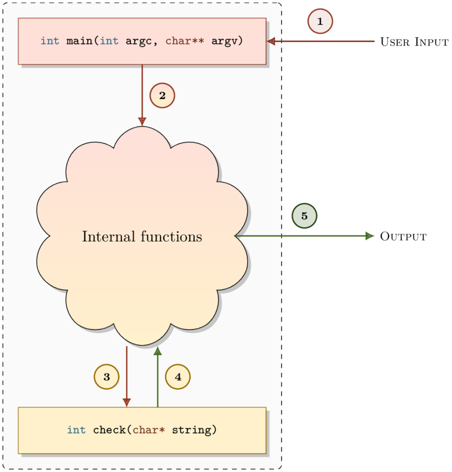
.text
...
400637: jmp 400480 <memcmp@plt>
...
1
.plt
...
400480: jmp 201028 <memcmp@got>
400486: push 0x2
40048b: jmp 400450 <.plt>
...
2
.got
...
201028: XXXXXX <memcmp@hook>
...
3
.hook
...
XXXXXX: memcmp hooked
...

https://lief.quarkslab.com/pts18/demo1

Exporting Functions

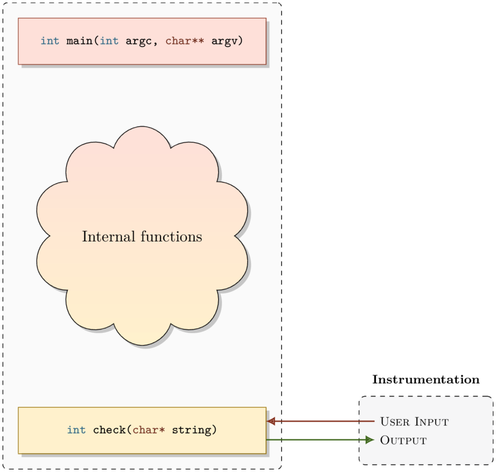
Exporting Functions

Idea



int main(int argc, char** argv)
1
USER INPUT
2
Internal functions
5
OUTPUT
3
4
int check(char* string)

Idea



int main(int argc, char** argv)
Internal functions
int check(char* string)
Instrumentation
USER INPUT
OUTPUT

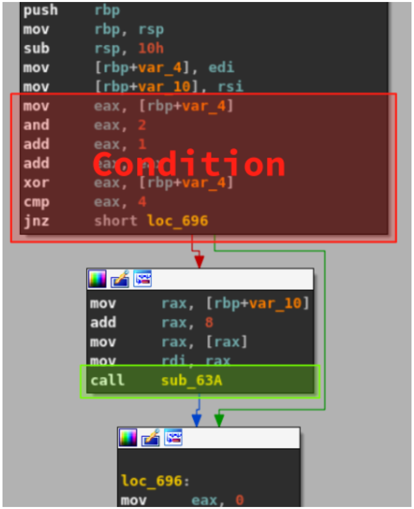
Example

```
int main(int argc, char argv[]) {

  if (COMPLICATED CONDITION) {
    fuzz_me(argv[1]);
  }

  return 0;
}
```

Example

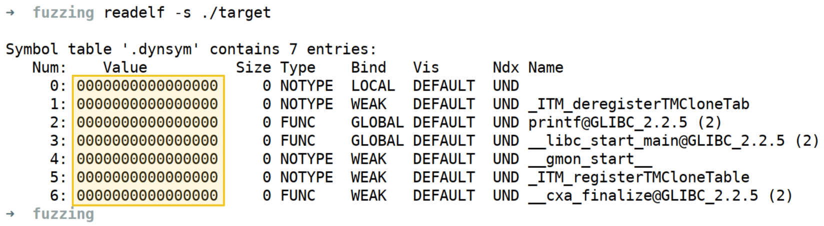


```
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_4], edi
mov     [rbp+var_10], rsi
mov     eax, [rbp+var_4]
and     eax, 2
add     eax, 1
add     eax, eax
xor     eax, [rbp+var_4]
cmp     eax, 4
jnz     short loc_696
```

```
mov     rax, [rbp+var_10]
add     rax, 8
mov     rax, [rax]
mov     rdi, rax
call    sub_63A
```

```
loc_696:
mov     eax, 0
```

Example



```
→  fuzzing readelf -s ./target

Symbol table '.dynsym' contains 7 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND 
     1: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_deregisterTMCloneTab
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND printf@GLIBC_2.2.5 (2)
     3: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.2.5 (2)
     4: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
     5: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_registerTMCloneTable
     6: 0000000000000000     0 FUNC    WEAK   DEFAULT  UND __cxa_finalize@GLIBC_2.2.5 (2)
→  fuzzing
```

Figure – Original Symbol Table

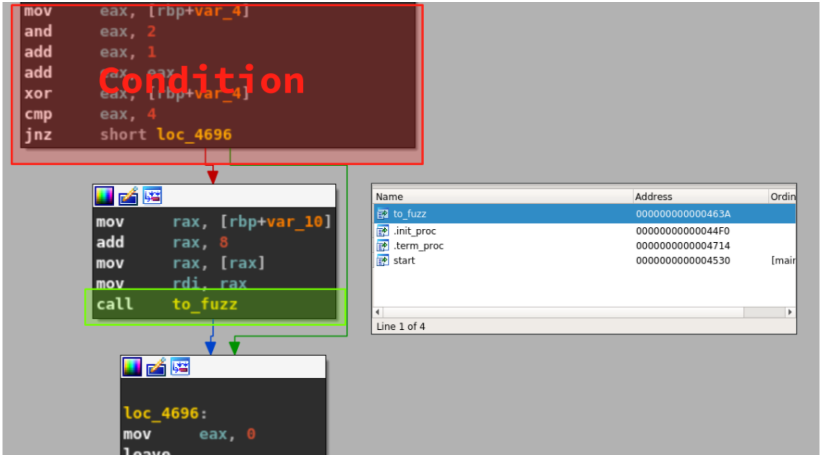
Example

```python
import lief
target = lief.parse("target")

target.add_exported_function(0x63A, "to_fuzz")

target.write("target_modified")
```

Example



```
mov     eax, [rbp+var_4]
and     eax, 2
add     eax, 1
add     eax, eax
xor     eax, [rbp+var_4]
cmp     eax, 4
jnz     short loc_4696
```

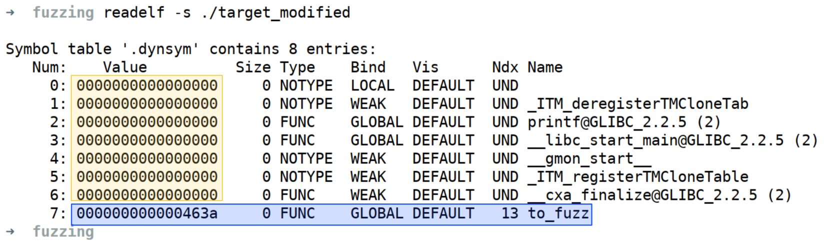
```
mov     rax, [rbp+var_10]
add     rax, 8
mov     rax, [rax]
mov     rdi, rax
call    to_fuzz
```

```
loc_4696:
mov     eax, 0
leave
```

Name	Address	Ordin
to_fuzz	000000000000463A	
.init_proc	00000000000044F0	
.term_proc	0000000000004714	
start	0000000000004530	[mair

Line 1 of 4

Example



```
→  fuzzing readelf -s ./target_modified

Symbol table '.dynsym' contains 8 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_deregisterTMCloneTab
     2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND printf@GLIBC_2.2.5 (2)
     3: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.2.5 (2)
     4: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
     5: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_registerTMCloneTable
     6: 0000000000000000     0 FUNC    WEAK   DEFAULT  UND __cxa_finalize@GLIBC_2.2.5 (2)
     7: 000000000000463a     0 FUNC    GLOBAL DEFAULT   13 to_fuzz
→  fuzzing
```

Figure – New Symbol Table

Example

```c
typedef void(*fnc_t)(const char*);

// Access with dlopen / dlsym
void* hdl = dlopen("./target_modified", RTLD_LAZY);
fnc_t to_fuzz = (fnc_t)dlsym(hdl, "to_fuzz");

to_fuzz(TO FEED);
```

https://lief.quarkslab.com/pts18/demo2

Code injection

Code injection through shared libraries

Context

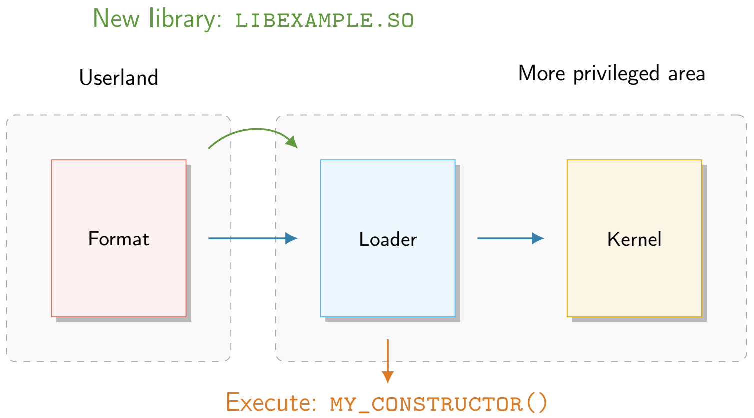
Different techniques exist to inject code:

- Using environment variables: `LD_PRELOAD`, `DYLD_INSERT_LIBRARIES`, ...
- Using operating system API: `WriteProcessMemory`, `ptrace`, ...
- Using custom kernel drivers
- Using executable formats

Context

Depending on the scenario, methods can be suitable or not. Next part shows a method based on shared libraries and executable formats to leverage code injection.

Linked Libraries - Loading



Injection process

1. Declare a constructor

```c
__attribute__((constructor))
void my_constructor(void) {
  printf("Run payload\n");
}
```

```
gcc -fPIC -shared libexample.c -o libexample.so
gcc -fPIC -shared libexample.c -o libexample.dylib
```

Injection process

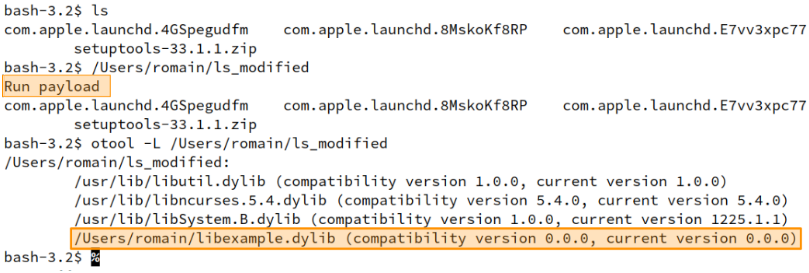
2. Add a dependency

```python
import lief
# ELF
elf = lief.parse("/usr/bin/ssh")
elf.add_library("libexample.so")
elf.write("ssh_modified")

# Mach-O
macho = lief.parse("/bin/ls")
macho.add_library("/Users/romain/libexample.dylib")
macho.write("ls_modified")

# PE: Not implemented yet
```

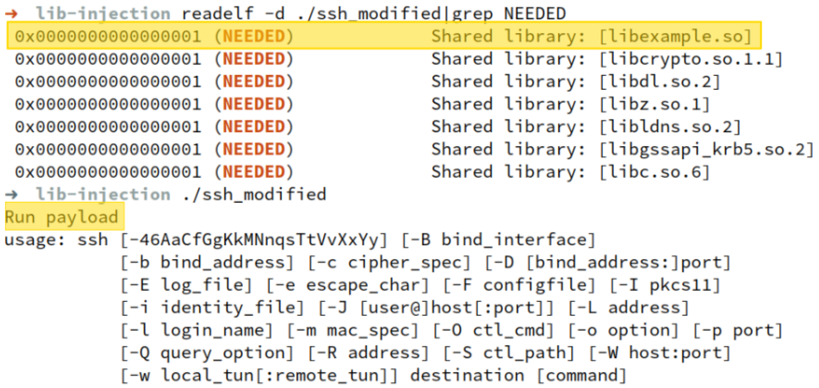
Injection process



```
bash-3.2$ ls
com.apple.launchd.4GSpegudfm    com.apple.launchd.8MskoKf8RP    com.apple.launchd.E7vv3xpc77
        setuptools-33.1.1.zip
bash-3.2$ /Users/romain/ls_modified
Run payload
com.apple.launchd.4GSpegudfm    com.apple.launchd.8MskoKf8RP    com.apple.launchd.E7vv3xpc77
        setuptools-33.1.1.zip
bash-3.2$ otool -L /Users/romain/ls_modified
/Users/romain/ls_modified:
        /usr/lib/libutil.dylib (compatibility version 1.0.0, current version 1.0.0)
        /usr/lib/libncurses.5.4.dylib (compatibility version 5.4.0, current version 5.4.0)
        /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1225.1.1)
        /Users/romain/libexample.dylib (compatibility version 0.0.0, current version 0.0.0)
bash-3.2$ %
```

https://lief.quarkslab.com/pts18/demo3

Injection process



```
→ lib-injection readelf -d ./ssh_modified|grep NEEDED
 0x0000000000000001 (NEEDED)                  Shared library: [libexample.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]
→ lib-injection ./ssh_modified
Run payload
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-B bind_interface]
           [-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]] destination [command]
```

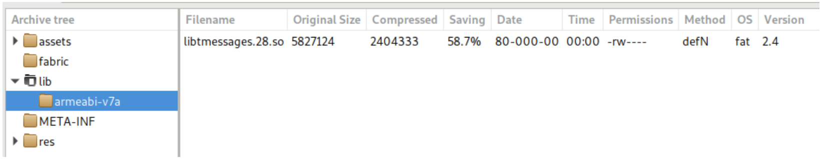
https://lief.quarkslab.com/pts18/demo3

Frida & LIEF: Frida injection in an Android application

Frida & LIEF

Using the techniques previously described, we can use Frida on an APK having at least one native library without root privileges.

Frida & LIEF

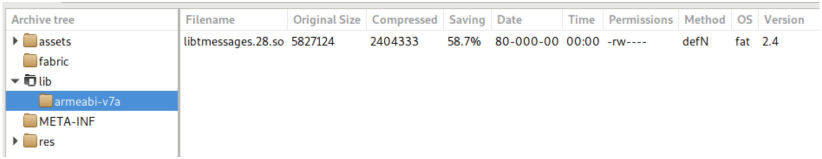


Archive tree
- assets
- fabric
- lib
  - armeabi-v7a
- META-INF
- res

Filename	Original Size	Compressed	Saving	Date	Time	Permissions	Method	OS	Version
libtmessages.28.so	5827124	2404333	58.7%	80-000-00	00:00	-rw----	defN	fat	2.4

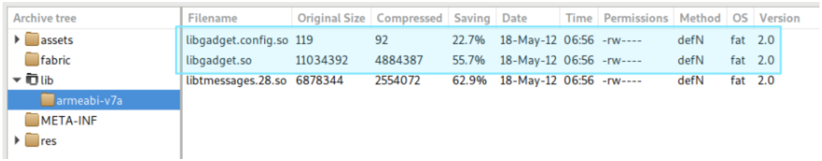
Figure – Original APK

Frida & LIEF



Archive tree	Filename	Original Size	Compressed	Saving	Date	Time	Permissions	Method	OS	Version
assets	libtmessages.28.so	5827124	2404333	58.7%	80-000-00	00:00	-rw----	defN	fat	2.4
fabric										
lib										
armeabi-v7a										
META-INF										
res										

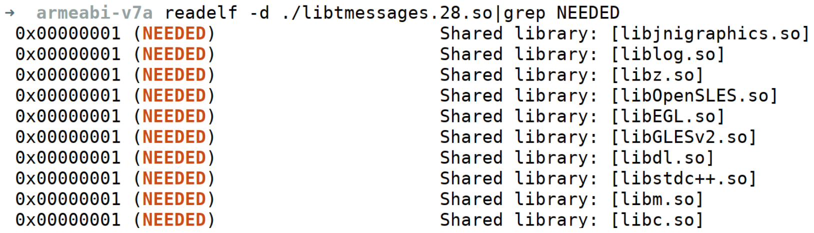
Figure – Original APK



Archive tree	Filename	Original Size	Compressed	Saving	Date	Time	Permissions	Method	OS	Version
assets	libgadget.config.so	119	92	22.7%	18-May-12	06:56	-rw----	defN	fat	2.0
fabric	libgadget.so	11034392	4884387	55.7%	18-May-12	06:56	-rw----	defN	fat	2.0
lib	libtmessages.28.so	6878344	2554072	62.9%	18-May-12	06:56	-rw----	defN	fat	2.0
armeabi-v7a										
META-INF										
res										

Figure – APK embedding Frida

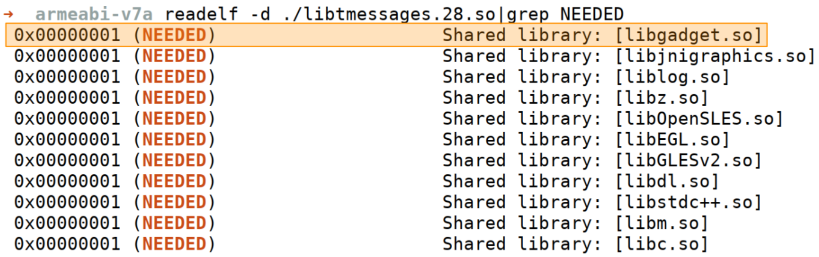
Frida & LIEF



```
➜  armeabi-v7a readelf -d ./libtmessages.28.so|grep NEEDED
 0x00000001 (NEEDED)                     Shared library: [libjnigraphics.so]
 0x00000001 (NEEDED)                     Shared library: [liblog.so]
 0x00000001 (NEEDED)                     Shared library: [libz.so]
 0x00000001 (NEEDED)                     Shared library: [libOpenSLES.so]
 0x00000001 (NEEDED)                     Shared library: [libEGL.so]
 0x00000001 (NEEDED)                     Shared library: [libGLESv2.so]
 0x00000001 (NEEDED)                     Shared library: [libdl.so]
 0x00000001 (NEEDED)                     Shared library: [libstdc++.so]
 0x00000001 (NEEDED)                     Shared library: [libm.so]
 0x00000001 (NEEDED)                     Shared library: [libc.so]
```

Figure – Original native library

Frida & LIEF



```
→  armeabi-v7a readelf -d ./libtmessages.28.so|grep NEEDED
 0x00000001 (NEEDED)                     Shared library: [libgadget.so]
 0x00000001 (NEEDED)                     Shared library: [libjnigraphics.so]
 0x00000001 (NEEDED)                     Shared library: [liblog.so]
 0x00000001 (NEEDED)                     Shared library: [libz.so]
 0x00000001 (NEEDED)                     Shared library: [libOpenSLES.so]
 0x00000001 (NEEDED)                     Shared library: [libEGL.so]
 0x00000001 (NEEDED)                     Shared library: [libGLESv2.so]
 0x00000001 (NEEDED)                     Shared library: [libdl.so]
 0x00000001 (NEEDED)                     Shared library: [libstdc++.so]
 0x00000001 (NEEDED)                     Shared library: [libm.so]
 0x00000001 (NEEDED)                     Shared library: [libc.so]
```

Figure – Modified native library

Frida & LIEF

libgadget.config.so

```
"interaction": {
  "type": "script",
  "path": "/data/local/tmp/myscript.js",
  "on_change": "reload"
}
```

/data/local/tmp/myscript.js

```
Java.perform(function () {
  var Log = Java.use("android.util.Log");
  var tag = "frida-lief";
  Log.v(tag, "I'm in the process!");

  Process.enumerateModules({
    onMatch: function (module) {
      Log.v(tag, "Module: " + module.name);
    },
    onComplete: function () {}
  });});
```

Frida & LIEF

Demo

`https://lief.quarkslab.com/pts18/demo4`

Format modification

Such modifications on formats are not new[3][4].

However, it's implemented in LIEF with a new approach that doesn't rely on replacing existing entries, using padding, removing entries, . . .

[3] Mayem Phrack #61

[4] `https://github.com/Tyilo/insert_dylib`

Format modification

Instead, it keeps a consistent state of the format:

- Export trie
- Symbol hash tables
- Relocations
- Symbol versions
- Rebase opcodes
- ...

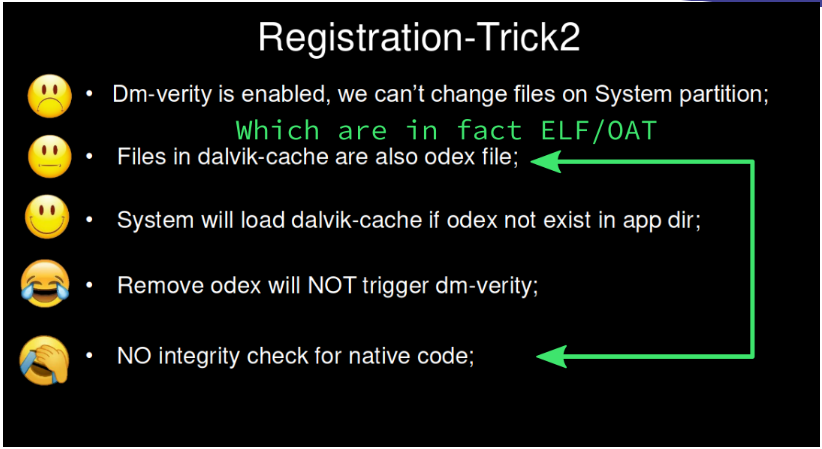
What's next

LIEF 0.9 comes with new formats related to Android:

- OAT
- VDEX
- DEX
- ART

Modification of these formats is not available yet but further version will support it.

What's next

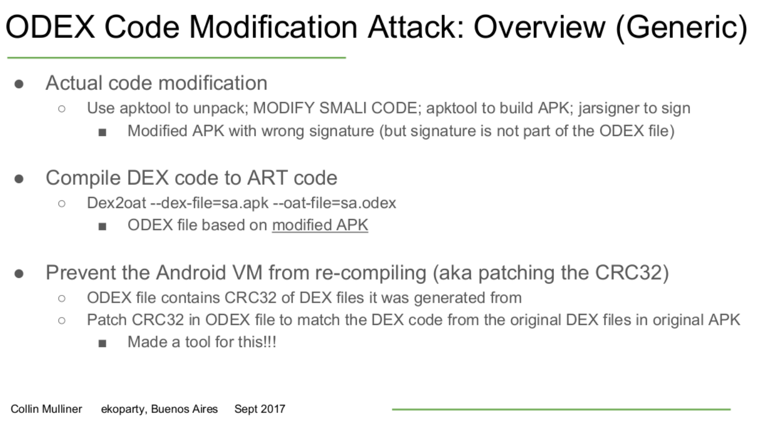


Registration-Trick2

- Dm-verity is enabled, we can't change files on System partition;
- Files in dalvik-cache are also odex file;
- System will load dalvik-cache if odex not exist in app dir;
- Remove odex will NOT trigger dm-verity;
- NO integrity check for native code;

Which are in fact ELF/OAT

What's next



ODEX Code Modification Attack: Overview (Generic)

- Actual code modification
  - Use apktool to unpack; MODIFY SMALI CODE; apktool to build APK; jarsigner to sign
    - Modified APK with wrong signature (but signature is not part of the ODEX file)
- Compile DEX code to ART code
  - Dex2oat --dex-file=sa.apk --oat-file=sa.odex
    - ODEX file based on modified APK
- Prevent the Android VM from re-compiling (aka patching the CRC32)
  - ODEX file contains CRC32 of DEX files it was generated from
  - Patch CRC32 in ODEX file to match the DEX code from the original DEX files in original APK
    - Made a tool for this!!!

Collin Mulliner ekoparty, Buenos Aires Sept 2017

What's next

Next version will also include support for Mach-O modifications:

- Add unlimited number of Load commands
- Add libraries
- Change signature
- ...

Thank You

https://lief.quarkslab.com

https://github.com/lief-project/LIEF

@LIEF_Project - @rh0main

quarkslab

SECURING EVERY BIT OF YOUR DATA