RMLL 2017

Romain Thomas - rthomas@quarkslab.com

LIEF: Library to Instrument Executable Formats

quarkslab

SECURING EVERY BIT OF YOUR DATA

Table of Contents

Introduction

Project Overview

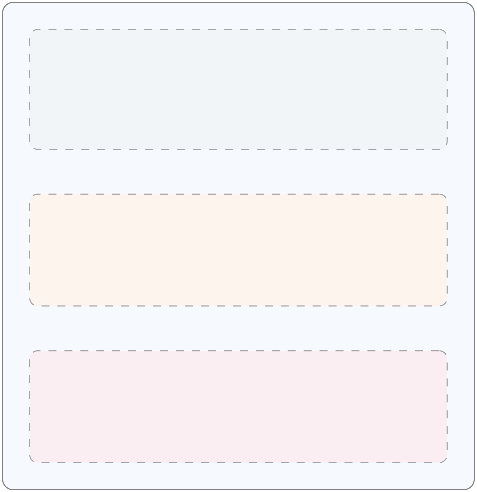
Demo

Conclusion

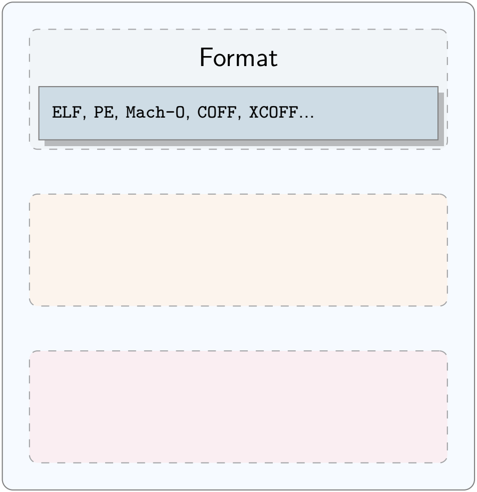
About

- Romain Thomas (rthomas@quarkslab.com) - Security engineer
- Working on obfuscation, software protection and reverse engineering
- Contributor to the Triton project, a dynamic binary analysis framework.

Layers of information



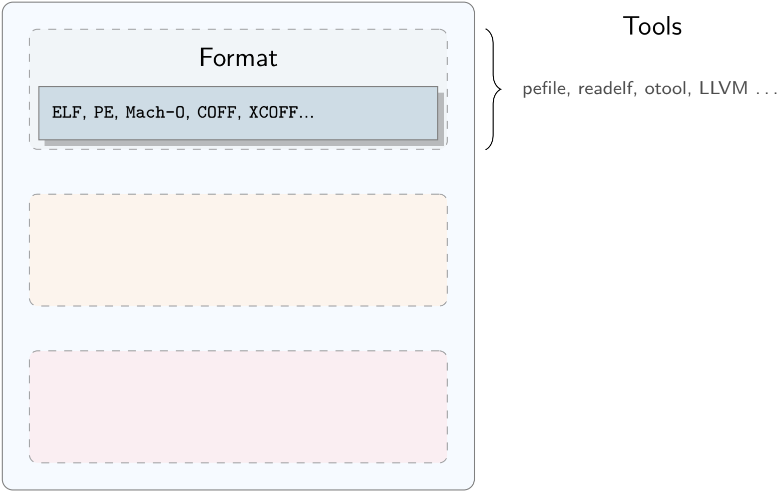
Layers of information



Format

ELF, PE, Mach-O, COFF, XCOFF...

Layers of information



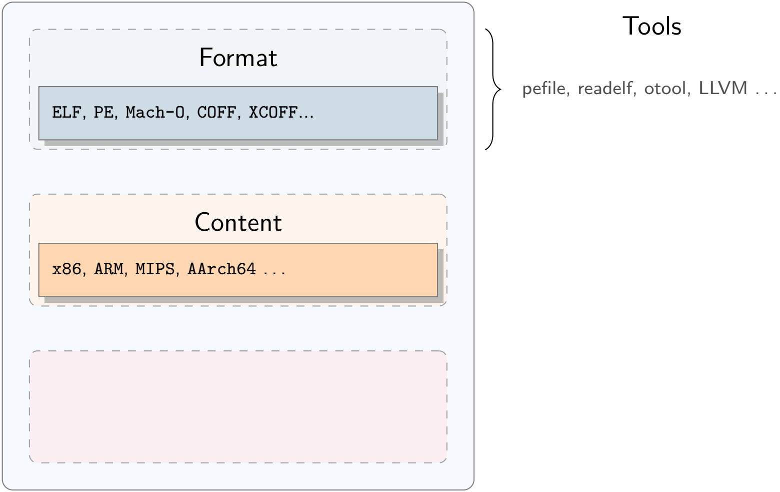
Format

`ELF, PE, Mach-O, COFF, XCOFF...`

Tools

pefile, readelf, otool, LLVM . . .

Layers of information



Format

`ELF, PE, Mach-O, COFF, XCOFF...`

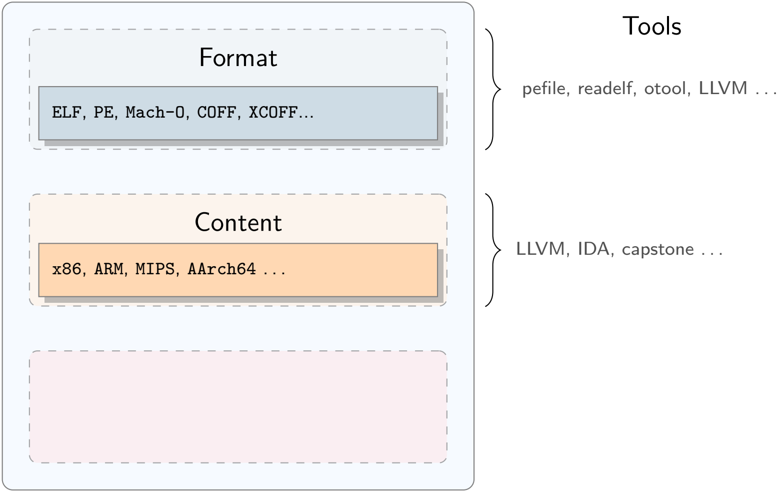
Content

`x86, ARM, MIPS, AArch64 ...`

Tools

pefile, readelf, otool, LLVM ...

Layers of information



Tools

Format

ELF, PE, Mach-O, COFF, XCOFF…

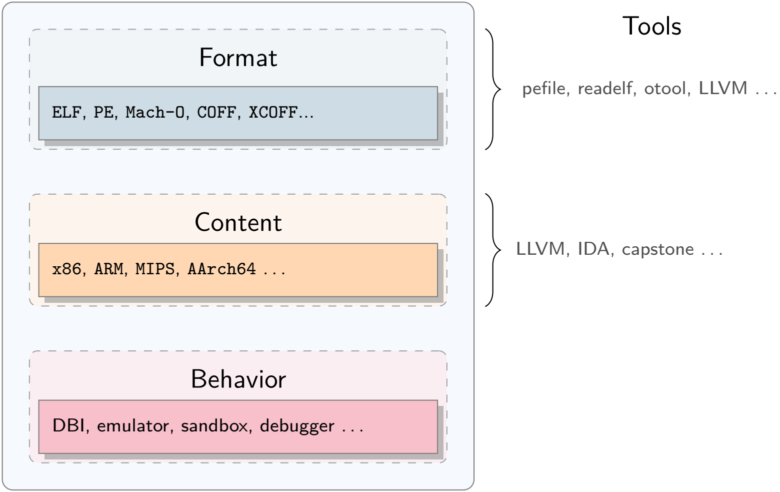
pefile, readelf, otool, LLVM . . .

Content

x86, ARM, MIPS, AArch64 . . .

LLVM, IDA, capstone . . .

Layers of information



Tools

Format

ELF, PE, Mach-O, COFF, XCOFF...

pefile, readelf, otool, LLVM . . .

Content

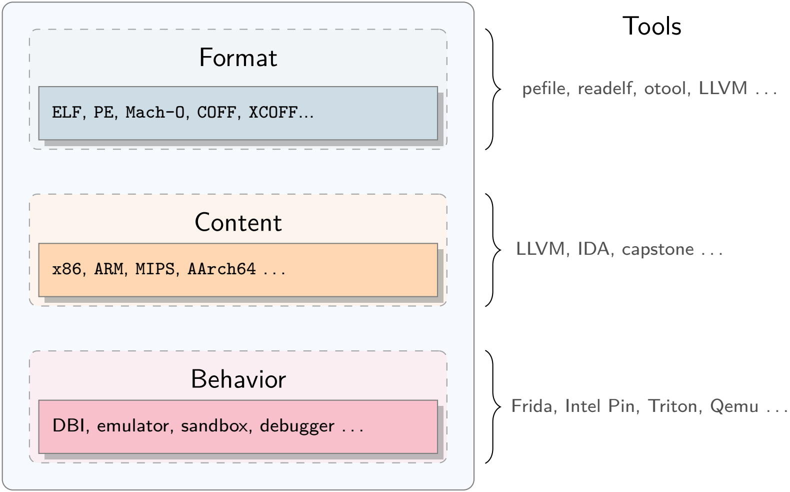
x86, ARM, MIPS, AArch64 . . .

LLVM, IDA, capstone . . .

Behavior

DBI, emulator, sandbox, debugger . . .

Layers of information



Tools

Format

ELF, PE, Mach-O, COFF, XCOFF...

pefile, readelf, otool, LLVM . . .

Content

x86, ARM, MIPS, AArch64 . . .

LLVM, IDA, capstone . . .

Behavior

DBI, emulator, sandbox, debugger . . .

Frida, Intel Pin, Triton, Qemu . . .

Howto?

- Get assembly code?
- Get symbols?
- Get imported functions?
- Get entry point?

Executable Formats

What is an executable format ?

Executable File Formats in a Nutshell

Executable file format gives information such as:

- First instruction address to execute

Executable File Formats in a Nutshell

Executable file format gives information such as:

- First instruction address to execute
- Libraries used

Executable File Formats in a Nutshell

Executable file format gives information such as:

- First instruction address to execute
- Libraries used
- Target architecture (`x86`, `ARM` . . .)

Executable File Formats in a Nutshell

The three mainstream formats:

- **ELF**: Linux, Android . . .
- **PE**: Windows
- **Mach-O**: OS-X, iOS, . . .

Purpose of LIEF

- Provide a **cross-platform** library to parse ELF, PE and Mach-O formats

Purpose of LIEF

- Provide a **cross-platform** library to parse ELF, PE and Mach-O formats
- Abstract common features from the different formats (section, header, entry point, symbols . . .)

Purpose of LIEF

- Provide a **cross-platform** library to parse ELF, PE and Mach-O formats
- Abstract common features from the different formats (section, header, entry point, symbols . . .)
- Enable format modifications

Purpose of LIEF

- Provide a **cross-platform** library to parse ELF, PE and Mach-O formats
- Abstract common features from the different formats (section, header, entry point, symbols . . .)
- Enable format modifications
- Provide an API for different languages (Python, `C++`, `C` . . .)

Purpose of LIEF

- Provide a **cross-platform** library to parse ELF, PE and Mach-O formats
- Abstract common features from the different formats (section, header, entry point, symbols . . .)
- Enable format modifications
- Provide an API for different languages (Python, `C++`, `C` . . .)

Provide an *all-in-one* library to deal with executable formats

Table of Contents

Introduction

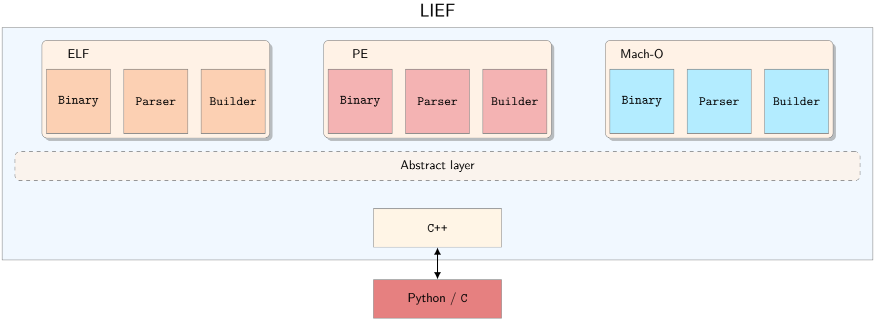
Project Overview
- Architecture
- Abstract Layer
- Tests and CI

Demo

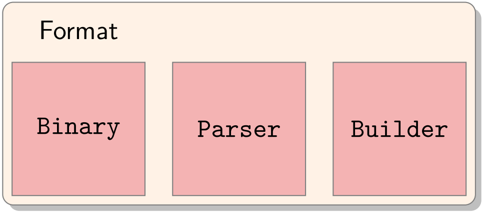
Conclusion

Architecture

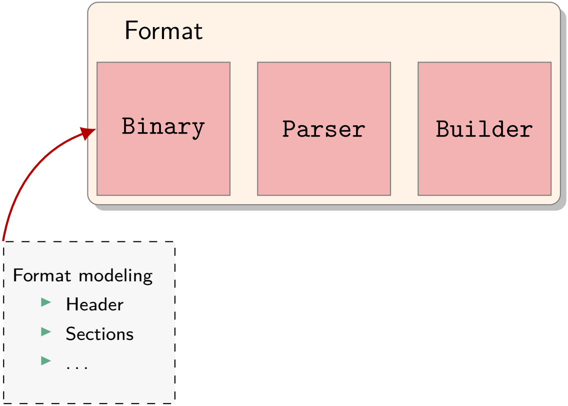
Architecture



Architecture



Architecture



Format

Binary

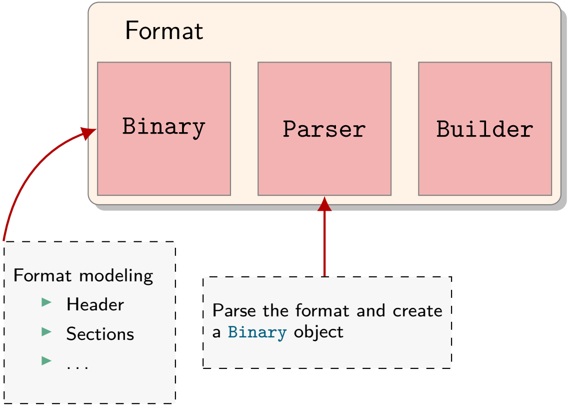
Parser

Builder

Format modeling

- Header
- Sections
- . . .

Architecture



Format

Binary

Parser

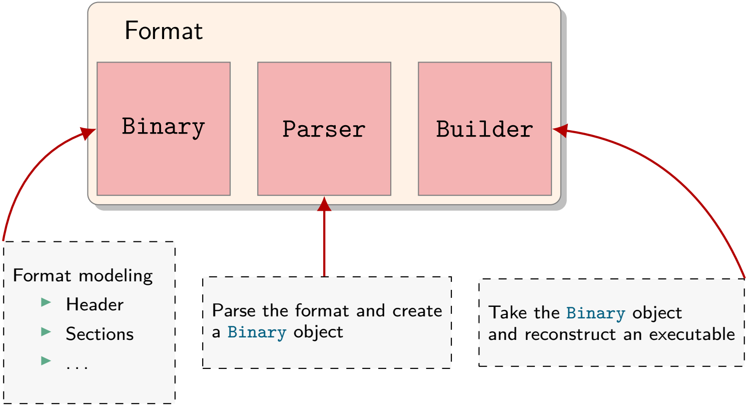
Builder

Format modeling

- Header
- Sections
- . . .

Parse the format and create a Binary object

Architecture



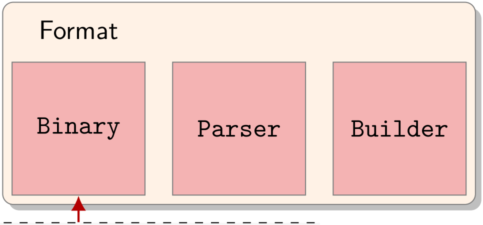
Format modeling

- Header
- Sections
- ...

Parse the format and create a `Binary` object

Take the `Binary` object and reconstruct an executable

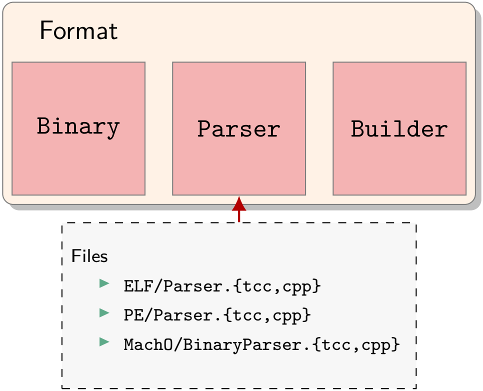
Architecture



Files

- `ELF/{Binary Header Section ...}.cpp`
- `PE/{Binary DosHeader Section ...}.cpp`
- `MachO/{Binary Header LoadCommand ...}.cpp`

Architecture



Format

`Binary`

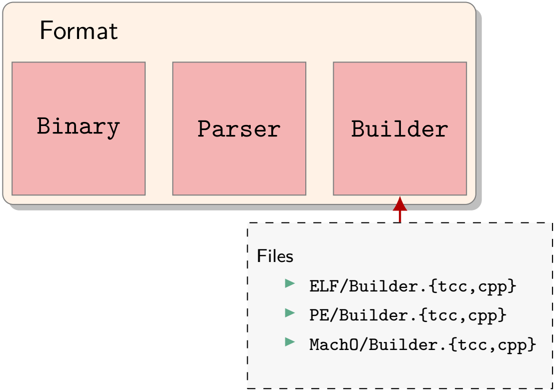
`Parser`

`Builder`

Files

- `ELF/Parser.{tcc,cpp}`
- `PE/Parser.{tcc,cpp}`
- `MachO/BinaryParser.{tcc,cpp}`

Architecture



Format

Binary

Parser

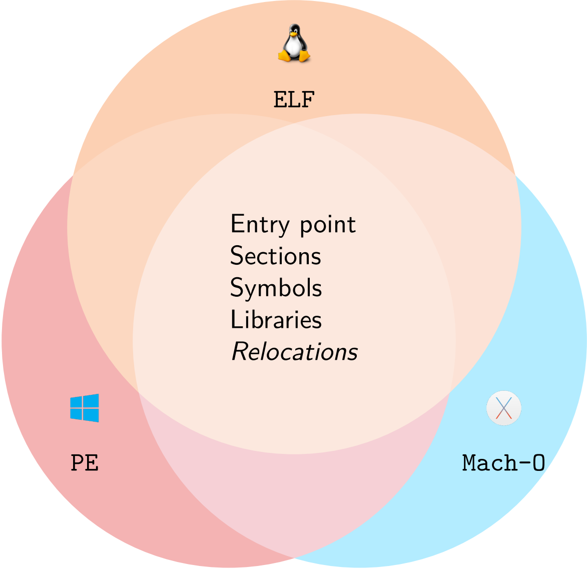
Builder

Files

- ELF/Builder.{tcc,cpp}
- PE/Builder.{tcc,cpp}
- MachO/Builder.{tcc,cpp}

Abstract Layer

Abstract Layer



ELF

PE

Mach-O

Entry point
Sections
Symbols
Libraries
Relocations

What is abstracted - Binary

Binary level

- Imported functions
- Exported functions
- Patch value(s) from a given address
- Retrieve value(s) from a given address

What is abstracted - Header

Header:

- Type
- Entry point
- Architecture
- Modes
- Endianness

What is abstracted - Header

Header:

- Type
  - `LIEF::OBJECT_TYPES::TYPE_EXECUTABLE`
  - `LIEF::OBJECT_TYPES::TYPE_LIBRARY`
  - ...
- Entry point
- Architecture
- Modes
- Endianness

What is abstracted - Header

Header:

- Type
- Entry point
- Architecture
  - `LIEF::ARCHITECTURES::ARCH_ARM`
  - `LIEF::ARCHITECTURES::ARCH_X86`
  - `LIEF::ARCHITECTURES::ARCH_ARM64`
  - ...
- Modes
- Endianness

What is abstracted - Header

Header:

- Type
- Entry point
- Architecture
- Modes
  - `LIEF::MODES::MODE_64`
  - `LIEF::MODES::MODE_THUMB`
  - `LIEF::MODES::MODE_V9`
  - ...
- Endianness

What is abstracted - Header

Header:

- Type
- Entry point
- Architecture
- Modes
- Endianness
  - `LIEF::ENDIANNESS::ENDIAN_BIG`
  - `LIEF::ENDIANNESS::ENDIAN_LITTLE`

What is abstracted - Section

Section:

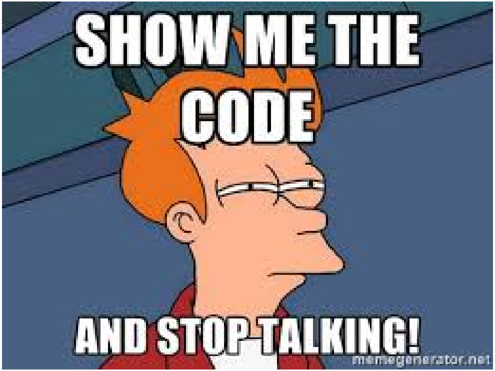
- Name
- Offset
- Size
- Virtual Address
- Raw content
- Entropy

What is abstracted - Symbol

Symbol:

- Name

Architecture



Abstract Layer

```python
import lief

def get_abstract_binary(binary):
    return super(binary.__class__, binary)

pe_exe    = get_abstract_binary(lief.parse("PE64_x86-64_HelloWorld.exe"))
macho_exe = get_abstract_binary(lief.parse("MachO64_x86-64_ls.bin"))
elf_exe   = get_abstract_binary(lief.parse("ELF64_x86-64_ls.bin"))

binaries = [pe_exe, macho_exe, elf_exe]

assert(all(
    binary.header.object_type  == lief.OBJECT_TYPES.EXECUTABLE
    for binary in binaries))
```

Abstract Layer

```python
import lief

def get_abstract_binary(binary):
    return super(binary.__class__, binary)

pe_exe    = get_abstract_binary(lief.parse("PE64_x86-64_HelloWorld.exe"))
macho_exe = get_abstract_binary(lief.parse("MachO64_x86-64_ls.bin"))
elf_exe   = get_abstract_binary(lief.parse("ELF64_x86-64_ls.bin"))

binaries = [pe_exe, macho_exe, elf_exe]

assert(all(
    binary.header.architecture == lief.ARCHITECTURES.X86
    for binary in binaries))
```

Abstract Layer

```python
import lief

def get_abstract_binary(binary):
    return super(binary.__class__, binary)

pe_exe    = get_abstract_binary(lief.parse("PE64_x86-64_HelloWorld.exe"))
macho_exe = get_abstract_binary(lief.parse("MachO64_x86-64_ls.bin"))
elf_exe   = get_abstract_binary(lief.parse("ELF64_x86-64_ls.bin"))

binaries = [pe_exe, macho_exe, elf_exe]

assert(all(
    lief.MODES.M64 in binary.header.modes
    for binary in binaries))
```

Abstract Layer

```python
import lief

def get_abstract_binary(binary):
    return super(binary.__class__, binary)

pe_exe    = get_abstract_binary(lief.parse("PE64_x86-64_HelloWorld.exe"))
macho_exe = get_abstract_binary(lief.parse("MachO64_x86-64_ls.bin"))
elf_exe   = get_abstract_binary(lief.parse("ELF64_x86-64_ls.bin"))

binaries = [pe_exe, macho_exe, elf_exe]

assert(all(
    binary.header.endianness == lief.ENDIANNESS.LITTLE
    for binary in binaries))
```

nm

nm utility

GNU nm lists the symbols from object files . . .

nm

Binutils/BFD Version:

nm

```
/* Print a single symbol.  */

static void
print_symbol (bfd *abfd, asymbol *sym, bfd_vma ssize, bfd *archive_bfd)
{
  symbol_info syminfo;
  struct extended_symbol_info info;

  PROGRESS (1);

  format->print_symbol_filename (archive_bfd, abfd);

  bfd_get_symbol_info (abfd, sym, &syminfo);
  info.sinfo = &syminfo;
  info.ssize = ssize;
  if (bfd_get_flavour (abfd) == bfd_target_elf_flavour)
    info.elfinfo = (elf_symbol_type *) sym;
  else
    info.elfinfo = NULL;
  format->print_symbol_info (&info, abfd);

  if (line_numbers)
    {
      static asymbol **syms;
      static long symcount;
      const char *filename, *functionname;
      unsigned int lineno;

  ...
```

LIEF Version:

nm

```python
import lief
import sys

binary = lief.parse(sys.argv[1])
for symbol in binary.symbols:
    print(symbol)
```

nm

```
$ python nm.py winhello64-mingw.exe
__mingw_invalidPa... 0    1   NULL FUNCTION  STATIC
pre_c_init           10   1   NULL FUNCTION  STATIC
.rdata$.refptr.mi... 470  3   NULL NULL      STATIC
...
```

nm

```
$ python nm.py FAT_libc++abi.dylib
___bzero           EXT 100 0
___maskrune        EXT 100 0
___stack_chk_fail  EXT 100 0
___stack_chk_guard EXT 100 0
___stderrp         EXT 100 0
_fputc             EXT 100 0
_free              EXT 100 0
_fwrite            EXT 100 0
_malloc            EXT 100 0
_memcmp            EXT 100 0
_memcpy            EXT 100 0
_memmove           EXT 100 0
...
```

nm

```
$ python nm.py /bin/ls
getenv          FUNC GLOBAL 0 0 GLIBC_2.2.5(3)
cap_to_text     FUNC GLOBAL 0 0 * Local *
sigprocmask     FUNC GLOBAL 0 0 GLIBC_2.2.5(3)
raise           FUNC GLOBAL 0 0 GLIBC_2.2.5(3)
localtime       FUNC GLOBAL 0 0 GLIBC_2.2.5(3)
__mempcpy_chk   FUNC GLOBAL 0 0 GLIBC_2.3.4(4)
...
```

Sectionless binary

nm

With LIEF, we removed the sections from the `ls` binary.

```
$ readelf -S ls_no_sections
There are no sections in this file.

$ nm ls_no_sections
nm: ls_no_sections: File format not recognized
```

nm

```
$ python nm.py ls_no_sections
getenv         FUNC GLOBAL 0 0 GLIBC_2.2.5(3)
cap_to_text    FUNC GLOBAL 0 0 * Local *
sigprocmask    FUNC GLOBAL 0 0 GLIBC_2.2.5(3)
raise          FUNC GLOBAL 0 0 GLIBC_2.2.5(3)
localtime      FUNC GLOBAL 0 0 GLIBC_2.2.5(3)
__mempcpy_chk  FUNC GLOBAL 0 0 GLIBC_2.3.4(4)
...
```

Howto? (answers)

Howto #1

Get assembly code?

Howto? (answers)

Howto #1

Get assembly code?

```
import lief
binary = lief.parse("C:\\Windows\\explorer.exe") # PE
asm = binary.get_section(".text")
```

Howto? (answers)

Howto #2

Get symbols?

Howto? (answers)

Howto #2

Get symbols?

```
import lief
binary = lief.parse("/bin/ls") # ELF
for symbol in binary.symbols:
  print(symbols)
```

Howto? (answers)

Howto #3

Get imported functions?

Howto? (answers)

Howto #3

Get imported functions?

```python
import lief
binary = lief.parse("/usr/lib/libc++abi.dylib") # Mach-O
for function in binary.imported_functions:
  print(function)
```

Tests and CI

Test suite

- Unit tests

Test suite

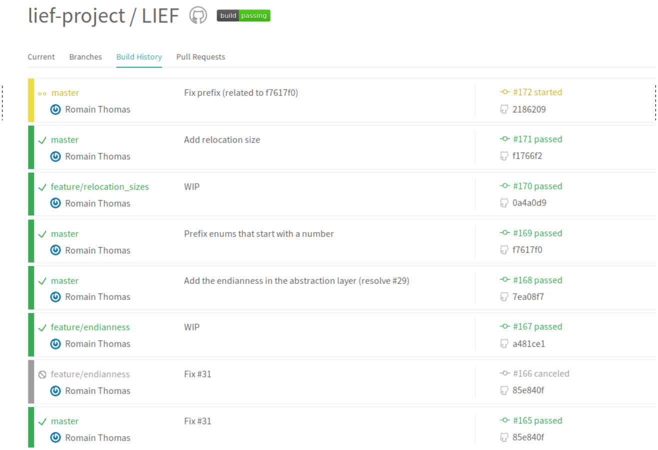
- Unit tests
- ELF parser is fuzzed with Melkor

Test suite

- Unit tests
- ELF parser is fuzzed with Melkor
- `Builder` tests: We run the (reconstructed) binary and check that it doesn't crash

Continuous Integration

Every commits are tested on Linux, OSX and Windows:



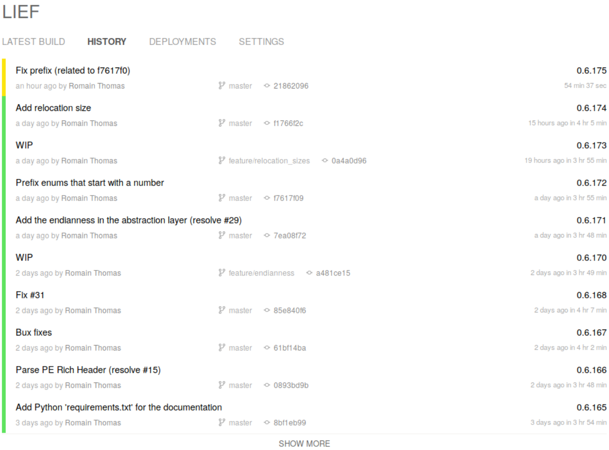
lief-project / LIEF build passing

Current | Branches | Build History | Pull Requests

○○ master	Romain Thomas	Fix prefix (related to f7617f0)	#172 started 2186209
✓ master	Romain Thomas	Add relocation size	#171 passed f1766f2
✓ feature/relocation_sizes	Romain Thomas	WIP	#170 passed 0a4a0d9
✓ master	Romain Thomas	Prefix enums that start with a number	#169 passed f7617f0
✓ master	Romain Thomas	Add the endianness in the abstraction layer (resolve #29)	#168 passed 7ea08f7
✓ feature/endianness	Romain Thomas	WIP	#167 passed a481ce1
⦸ feature/endianness	Romain Thomas	Fix #31	#166 canceled 85e840f
✓ master	Romain Thomas	Fix #31	#165 passed 85e840f

Continuous Integration

Every commits are tested on Linux, OSX and Windows:



LIEF

LATEST BUILD HISTORY DEPLOYMENTS SETTINGS

Fix prefix (related to f7617f0) — 0.6.175
an hour ago by Romain Thomas — master — 21862096 — 54 min 37 sec

Add relocation size — 0.6.174
a day ago by Romain Thomas — master — f1766f2c — 15 hours ago in 4 hr 5 min

WIP — 0.6.173
a day ago by Romain Thomas — feature/relocation_sizes — 0a4a0d96 — 19 hours ago in 3 hr 55 min

Prefix enums that start with a number — 0.6.172
a day ago by Romain Thomas — master — f7617f09 — a day ago in 3 hr 55 min

Add the endianness in the abstraction layer (resolve #29) — 0.6.171
a day ago by Romain Thomas — master — 7ea08f72 — a day ago in 3 hr 48 min

WIP — 0.6.170
2 days ago by Romain Thomas — feature/endianness — a481ce15 — 2 days ago in 3 hr 49 min

Fix #31 — 0.6.168
2 days ago by Romain Thomas — master — 85e840f6 — 2 days ago in 4 hr 7 min

Bux fixes — 0.6.167
2 days ago by Romain Thomas — master — 61bf14ba — 2 days ago in 4 hr 2 min

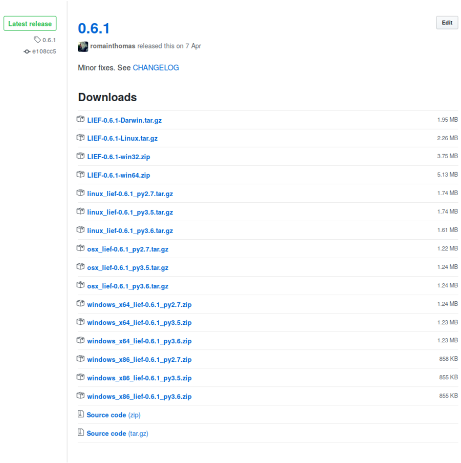
Parse PE Rich Header (resolve #15) — 0.6.166
2 days ago by Romain Thomas — master — 0893bd9b — 2 days ago in 3 hr 48 min

Add Python 'requirements.txt' for the documentation — 0.6.165
3 days ago by Romain Thomas — master — 8bf1eb99 — 3 days ago in 3 hr 54 min

SHOW MORE

Release

For each tagged versions we provide prebuilt SDK and Python packages



Latest release

0.6.1

e108cc5

Edit

0.6.1

romainthomas released this on 7 Apr

Minor fixes. See CHANGELOG

Downloads

LIEF-0.6.1-Darwin.tar.gz	1.95 MB
LIEF-0.6.1-Linux.tar.gz	2.26 MB
LIEF-0.6.1-win32.zip	3.75 MB
LIEF-0.6.1-win64.zip	5.13 MB
linux_lief-0.6.1_py2.7.tar.gz	1.74 MB
linux_lief-0.6.1_py3.5.tar.gz	1.74 MB
linux_lief-0.6.1_py3.6.tar.gz	1.61 MB
osx_lief-0.6.1_py2.7.tar.gz	1.22 MB
osx_lief-0.6.1_py3.5.tar.gz	1.24 MB
osx_lief-0.6.1_py3.6.tar.gz	1.24 MB
windows_x64_lief-0.6.1_py2.7.zip	1.24 MB
windows_x64_lief-0.6.1_py3.5.zip	1.23 MB
windows_x64_lief-0.6.1_py3.6.zip	1.23 MB
windows_x86_lief-0.6.1_py2.7.zip	858 KB
windows_x86_lief-0.6.1_py3.5.zip	855 KB
windows_x86_lief-0.6.1_py3.6.zip	855 KB
Source code (zip)	
Source code (tar.gz)	

Table of Contents

Introduction

Project Overview

Demo
- PE Hooking
- Petya signature
- ELF obfuscation

Conclusion

PE Hooking

Petya signature

ELF obfuscation

Table of Contents

Introduction

Project Overview

Demo

Conclusion
- Format modifications
- Documentation
- Version `0.7`

Format modifications

Modifications

Format modifications can be a starting point to:

- Packing
- Watermarking
- Hooking: Perform interposition on functions
- Persistent code injection
- Malware analysis (static unpacking . . .)

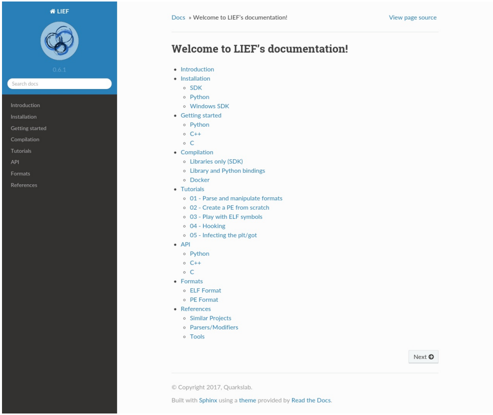
Documentation

Documentation

LIEF documentation includes:

- Tutorials
- API: Python, `C++` and `C`
- References: Existing projects that deals with executable formats
- Installation and compilation guide

Documentation



Welcome to LIEF's documentation!

- Introduction
- Installation
  - SDK
  - Python
  - Windows SDK
- Getting started
  - Python
  - C++
  - C
- Compilation
  - Libraries only (SDK)
  - Library and Python bindings
  - Docker
- Tutorials
  - 01 - Parse and manipulate formats
  - 02 - Create a PE from scratch
  - 03 - Play with ELF symbols
  - 04 - Hooking
  - 05 - Infecting the plt/got
- API
  - Python
  - C++
  - C
- Formats
  - ELF Format
  - PE Format
- References
  - Similar Projects
  - Parsers/Modifiers
  - Tools

Built with Sphinx using a theme provided by Read the Docs.

See: `https://lief.quarkslab.com/doc`

Version 0.7

Version 0.7

What's new ?

- Function hooking through the *IAT*
- Icons, Manifest . . . modification with the *ResourcesManager*
- Serialize PE object into JSON
- Parse Rich Header

Version 0.7

What's new ?

- Fully handle section-less binaries
- Parse notes: `.note.ABI-tag`, `.note.gnu.build-id`, ...
- Parse SYSV hash table

Version 0.7

Full changelog

`https://lief.quarkslab.com/doc/changelog.html#july-3-2017`

- Source code is available on GitHub: `https://github.com/lief-project` (**Apache 2.0** license)
- Website: `https://lief.quarkslab.com`

- Source code is available on GitHub: `https://github.com/lief-project` (**Apache 2.0** license)
- Website: `https://lief.quarkslab.com`

Missing feature or bug?

- Source code is available on GitHub: `https://github.com/lief-project` (**Apache 2.0** license)
- Website: `https://lief.quarkslab.com`

Missing feature or bug?

lief@quarkslab.com
or
Open an issue / pull request

Thank you!

Twitter: `@rh0main`

quarkslab

SECURING EVERY BIT OF YOUR DATA