Добавление кода в бинарный файл

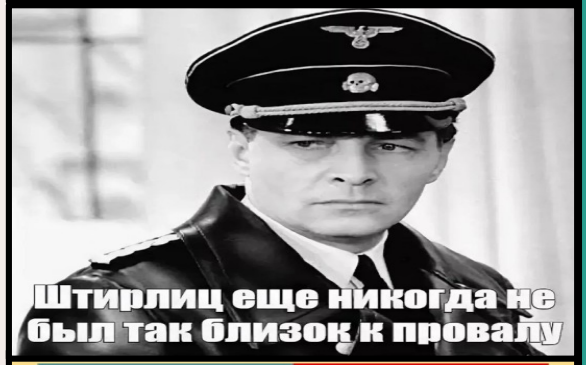
Данилов Александр ККСО-01-20

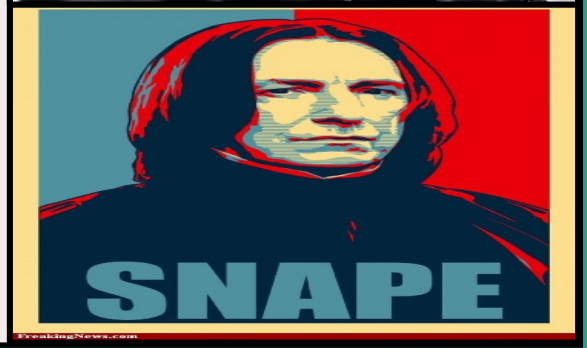
План доклада:

1) Метод Гая Юлия Цезаря

2) Метод Штирлица

3) Метод Северуса Снейпа





Метод Гая Юлия Цезаря
или
...

Суть: ...

Метод Гая Юлия Цезаря
или
Замена команд

Суть: меняй нужную команду на свою.

Замена с помощью дизассемблера IDA

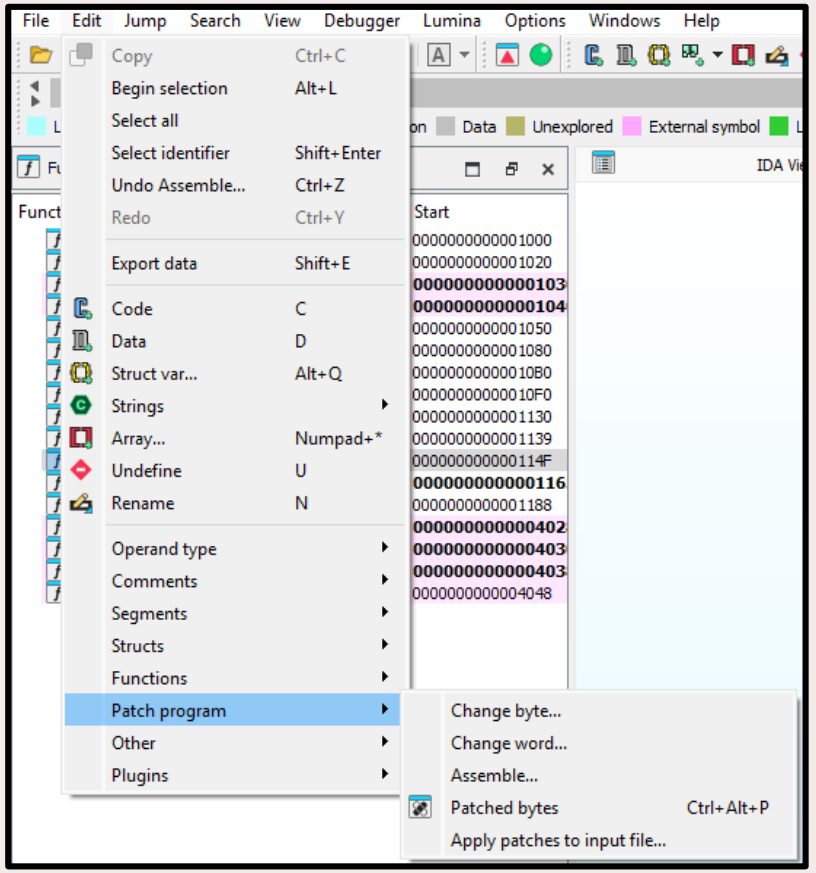
```
$ ./m_1.elf
Hello, World!
$
```

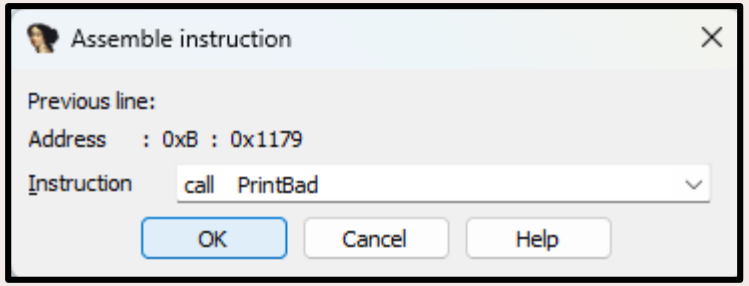
```
; Attributes: bp-based frame

; int __fastcall main(int argc, const char **argv, const char **envp)
public main
main proc near

var_10= qword ptr -10h
var_4= dword ptr -4

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_4], edi
mov     [rbp+var_10], rsi
mov     eax, 0
call    PrintHello
mov     eax, 0
leave
retn
; } // starts at 1165
main endp

_text ends
```

Functions

Function name	Segment	Start
_init_proc	.init	0000000000001000
sub_1020	.plt	0000000000001020
_puts	**.plt**	**000000000000103**
__cxa_finalize	**.plt.got**	**000000000000104**
_start	.text	0000000000001050
deregister_tm_clones	.text	0000000000001080
register_tm_clones	.text	00000000000010B0
__do_global_dtors_aux	.text	00000000000010F0
frame_dummy	.text	0000000000001130
PrintHello	.text	0000000000001139
PrintBad	.text	000000000000114F
main	**.text**	**000000000000116**
_term_proc	.fini	0000000000001188
__libc_start_main	**extern**	**000000000000402**
puts	**extern**	**000000000000403**
__imp___cxa_finalize	**extern**	**000000000000403**
__gmon_start__	extern	0000000000004048

Замена с помощью дизассемблера IDA






```
$ ./m_1.elf
А ты сделал лабы?!
$
```

Замена с помощью дизассемблера Radare2

```
$ ./m_1.elf
Hello, World!
$ r2 -A -w ./m_1.elf
```

```
[0x00001050]> afl
0x00001030    1      6 sym.imp.puts
0x00001040    1      6 sym.imp.__cxa_finalize
0x00001050    1     33 entry0
0x00001080    4     34 sym.deregister_tm_clones
0x000010b0    4     51 sym.register_tm_clones
0x000010f0    5     54 entry.fini0
0x00001130    1      9 entry.init0
0x00001139    1     22 sym.PrintHello
0x0000114f    1     22 sym.PrintBad
0x00001188    1      9 sym._fini
0x00001165    1     32 main
0x00001000    3     23 sym._init
[0x00001050]>
```

```
[0x00001050]> s main
[0x00001165]> pdf
            ; ICOD XREF from entry0 @ 0x1064(r)
┌ 32: int main (int argc, char **argv);
└ `- args(rdi, rsi) vars(2:sp[0xc..0x18])
            0x00001165      55              push rbp
            0x00001166      4889e5          mov rbp, rsp
            0x00001169      4883ec10        sub rsp, 0x10
            0x0000116d      897dfc          mov dword [var_4h], edi     ; argc
            0x00001170      488975f0        mov qword [var_10h], rsi    ; argv
            0x00001174      b800000000      mov eax, 0
            0x00001179      e8bbffffff      call sym.PrintHello
            0x0000117e      b800000000      mov eax, 0
            0x00001183      c9              leave
            0x00001184      c3              ret
[0x00001165]> s 0x1179
[0x00001179]> wa call 0x114f
INFO: Written 5 byte(s) (call 0x114f) = wx e8d1ffffff @ 0x00001179
```

```
$ ./m_1.elf
А ты сделал лабы?!
$
```

Метод Штирлица
или
...

Суть: ...

Метод Штирлица
или
Внедрение команд

Суть: расширяем файл командами.

Немного теории, правда ...

Секция — это именованный участок памяти с набором прав

Сегмент — это набор секций

Можно выдохнуть, снова практика 🎉

```
section .text
    global _start

_start:
    xor rax, rax

    mov rax, 0x0A2183D1B1D0B0D0
    push rax
    mov rax, 0xBBD020BBD0B0D0BB
    push rax
    mov rax, 0xD0B5D0B4D081D120
    push rax
    mov rax, 0x8FD1202CB0D094D0
    push rax

    mov rdx, 32

    ; output
    mov rax, 1
    mov rdi, 1
    mov rsi, rsp
    syscall

    add rsp, 32

    ; exit
    mov rax, 60
    xor rdi, rdi
    syscall
```

```
section .text
    global _start

_start:
    add rsp, 16
    mov rax, [rsp]
    sub rsp, 16
    ret
```

```
nasm -f elf64 good1.nasm -o good1.o
ld -m elf_x86_64 good1.o -o good1.bin
nasm -f elf64 good2.nasm -o good2.o
ld -m elf_x86_64 good2.o -o good2.bin
```

```
ldd ./good1.bin
        not a dynamic executable
ldd ./good2.bin
        not a dynamic executable
```

Добавление секции

```python
import lief
import typing
from sys import argv

if len(argv) < 4:
  print(f"Usage: {argv[0]} FROM_BIN TO OUT")

from_path = argv[1] # binary program
to_path = argv[2] # program or shared library
output = argv[3] # result

from_lief = lief.ELF.parse(from_path)
to_lief = lief.ELF.parse(to_path)

if isinstance(from_lief, type(None)) or isinstance(to_lief, type(None)):
  print("Error reading files")
  exit(1)

print("ELF mode")
print(f"Segments count: {len(from_lief.segments)}")

section = lief.ELF.Section(f".test.good", lief.ELF.Section.TYPE.PROGBITS)
section.type = lief.ELF.Section.TYPE.PROGBITS
section += lief.ELF.Section.FLAGS.EXECINSTR
section += lief.ELF.Section.FLAGS.ALLOC
section += lief.ELF.Section.FLAGS.WRITE
section.content  = from_lief.get_section(".text").content
section.alignment = 0x1000
section = to_lief.add(section, loaded=True)

print(f"Virtual address: {hex(section.virtual_address)}")
to_lief.patch_pltgot("GetHelloWorld", section.virtual_address)

to_lief.write(output)
```

```c
#include <stdio.h>

extern char* GetHelloWorld();

int main() {
  const char* nuclear_code = "662607015\n";
  printf(GetHelloWorld());
}
```

```c
char* GetHelloWorld() {
  return "Hello, World!\n";
}
```

```
$ LD_LIBRARY_PATH=$(realpath .) ./m_2.elf
Hello, World!
```

```
$ python add_section.py good2.bin m_2.elf m_2h.elf
ELF mode
Segments count: 2
Virtual address: 0xd000
```

```
$ LD_LIBRARY_PATH=$(realpath .) ./m_2h.elf
662607015
```

Кот убежал, но обещал вернуться

Добавление сегмента

```python
import lief
import typing
from sys import argv

if len(argv) < 4:
  print(f"Usage: {argv[0]} FROM_BIN TO OUT")

from_path = argv[1] # binary program
to_path = argv[2] # program or shared library
output = argv[3] # result

from_lief = lief.ELF.parse(from_path)
to_lief = lief.ELF.parse(to_path)

if isinstance(from_lief, type(None)) or isinstance(to_lief, type(None)):
  print("Error reading files")
  exit(1)

print("ELF mode")
print(f"Segments count: {len(from_lief.segments)}")

for seq in from_lief.segments:
  segment = to_lief.add(seq)
  segment.alignment = 0x1000

  print(f"Sections count: {len(seq.sections)}")
  for s in seq.sections:
    if s.name == ".text":
      addr = segment.virtual_address + (s.virtual_address - seq.virtual_address)
      print(f"Added .text, virtual address : {hex(addr)}")
      to_lief.patch_pltgot("GetHelloWorld", addr)

to_lief.write(output)
```

```
$ python add_segment.py good1.bin m_2.elf m_2h.elf
ELF mode
Segments count: 2
Sections count: 0
Sections count: 1
Added .text, virtual address : 0x16000
```

```
$ LD_LIBRARY_PATH=$(realpath .) ./m_2h.elf
Да, я сделал лабу!
```

Немного корма 🍗

Замена экспортируемой функции

```python
import lief
import typing
from sys import argv

if len(argv) < 4:
  print(f"Usage: {argv[0]} FROM_BIN TO OUT")

from_path = argv[1] # binary program
to_path = argv[2] # program or shared library
output = argv[3] # result

from_lief = lief.ELF.parse(from_path)
to_lief = lief.ELF.parse(to_path)

if isinstance(from_lief, type(None)) or isinstance(to_lief, type(None)):
  print("Error reading files")
  exit(1)

print("ELF mode")
print(f"Segments count: {len(from_lief.segments)}")

section = lief.ELF.Section(f".test.good", lief.ELF.Section.TYPE.PROGBITS)
section.type = lief.ELF.Section.TYPE.PROGBITS
section += lief.ELF.Section.FLAGS.EXECINSTR
section += lief.ELF.Section.FLAGS.ALLOC
section += lief.ELF.Section.FLAGS.WRITE
section.content  = from_lief.get_section(".text").content
section.alignment = 0x1000
section = to_lief.add(section, loaded=True)

print(f"Virtual address: {hex(section.virtual_address)}")

export = to_lief.export_symbol("GetHelloWorld")
export.value = section.virtual_address

to_lief.write(output)
```

```
$ python change_so_addr.py good2.bin libdata.so libdata.so
ELF mode
Segments count: 2
Virtual address: 0xd000
```

```
$ LD_LIBRARY_PATH=$(realpath .) ./m_2.elf
662607015
```

Замена функции инициализации

```python
import lief
import typing
from sys import argv

if len(argv) < 4:
  print(f"Usage: {argv[0]} FROM_BIN TO OUT")

from_path = argv[1] # binary program
to_path = argv[2] # program or shared library
output = argv[3] # result

from_lief = lief.ELF.parse(from_path)
to_lief = lief.ELF.parse(to_path)

if isinstance(from_lief, type(None)) or isinstance(to_lief, type(None)):
  print("Error reading files")
  exit(1)

print("ELF mode")
print(f"Segments count: {len(from_lief.segments)}")

section = lief.ELF.Section(f".test.good", lief.ELF.Section.TYPE.PROGBITS)
section.type = lief.ELF.Section.TYPE.PROGBITS
section += lief.ELF.Section.FLAGS.EXECINSTR
section += lief.ELF.Section.FLAGS.ALLOC
section += lief.ELF.Section.FLAGS.WRITE
section.content  = from_lief.get_section(".text").content
section.alignment = 0x1000
section = to_lief.add(section, loaded=True)

print(f"Virtual address: {hex(section.virtual_address)}")

if to_lief.has(lief.ELF.DynamicEntry.TAG.INIT_ARRAY):
  init_array = to_lief.get(lief.ELF.DynamicEntry.TAG.INIT_ARRAY)
  assert isinstance(init_array, lief.ELF.DynamicEntryArray)
  callbacks = init_array.array
  callbacks[0] = section.virtual_address
  init_array.array = callbacks

if to_lief.has(lief.ELF.DynamicEntry.TAG.INIT):
  init = to_lief.get(lief.ELF.DynamicEntry.TAG.INIT)
  init.value = section.virtual_address

to_lief.write(output)
```

```
$ python change_so_init.py good1.bin libdata.so libdata.so
ELF mode
Segments count: 2
Virtual address: 0x17000
```

```
$ LD_LIBRARY_PATH=$(realpath .) ./m_2.elf
Да, я сделал лабу!
```

Метод Северуса Снейпа
или
...

Суть: ...

Метод Северуса Снейпа
или
Манипуляции с библиотеками

Суть: меняем поведение программы через изменение динамических зависимостей.

Подмена динамической библиотеки

```c
#include <stdio.h>

extern char* GetHelloWorld();

int main() {
  printf(GetHelloWorld());
}
```

```c
char* GetHelloWorld() {
  return "Hello, World!\n";
}
```

```c
char* GetHelloWorld() {
  return "Goodbye, World!\n";
}
```

```
$ ldd ./m_3.elf
        linux-vdso.so.1 (0x00007ffea6dee000)
        libdata.so => not found
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f6e841b0000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f6e843a2000)
```

```
make -C libgood
make[1]: Entering directory '/mnt/c/Users/Aleks/Desktop/РПВ/Доклад/RPV-lief/linking/libgood'
gcc data.c -shared -o libdata.so
make[1]: Leaving directory '/mnt/c/Users/Aleks/Desktop/РПВ/Доклад/RPV-lief/linking/libgood'
make -C libbad
make[1]: Entering directory '/mnt/c/Users/Aleks/Desktop/РПВ/Доклад/RPV-lief/linking/libbad'
gcc data.c -shared -o libdata.so
gcc who.c -shared -o libwho.so
```

```
$ LD_LIBRARY_PATH=$(realpath ./libgood) ./m_3.elf
Hello, World!
$ LD_LIBRARY_PATH=$(realpath ./libbad) ./m_3.elf
Goodbye, World!
```

Обязательная загрузка динамической библиотеки

```c
#include <stdio.h>

[[gnu::constructor]] void init() {
  printf("I'm hacker!\n");
}
```

```
make -C libbad
make[1]: Entering directory '/mnt/c/Users/Aleks/Desktop/РПВ/Доклад/RPV-lief/linking/libbad'
gcc data.c -shared -o libdata.so
gcc who.c -shared -o libwho.so
gcc print.c -shared -o libprint.so
```

```
$ (LD_LIBRARY_PATH=$(realpath ./libgood) LD_PRELOAD=$(realpath ./libbad/libprint.so) ./m_3.elf)
I'm hacker!
Hello, World!
```

Полная свобода

```python
import lief
import typing
from sys import argv

if len(argv) < 4:
  print(f"Usage: {argv[0]} FROM_BIN TO OUT")

bin = argv[1] # binary program
shared = argv[2] # import file
output = argv[3] # result

bin = lief.ELF.parse(bin)

if isinstance(bin, type(None)):
  print("Error reading file")
  exit(1)

print("ELF mode")

bin.add_library(shared)

bin.write(output)
```

```c
#include <stdlib.h>

[[gnu::constructor]] void init() {
  system("/usr/bin/whoami");
}
```

```
make -C libbad
make[1]: Entering directory '/mnt/c/Users/Aleks/Desktop/РПВ/Доклад/RPV-lief/linking/libbad'
gcc data.c -shared -o libdata.so
gcc who.c -shared -o libwho.so
gcc print.c -shared -o libprint.so
```

```
$ python add_import.py .\m_3.elf libwho.so ./m_3h.elf
ELF mode
```

```
$ LD_LIBRARY_PATH=$(realpath ./libbad) ./m_3h.elf
alex
Goodbye, World!
```

Не передавайте libwho.so через LD_PRELOAD!

СПАСИБО

ЗА ВНИМАНИЕ !

https://github.com/Dandya/RPV-lief

ВОПРОСЫ?