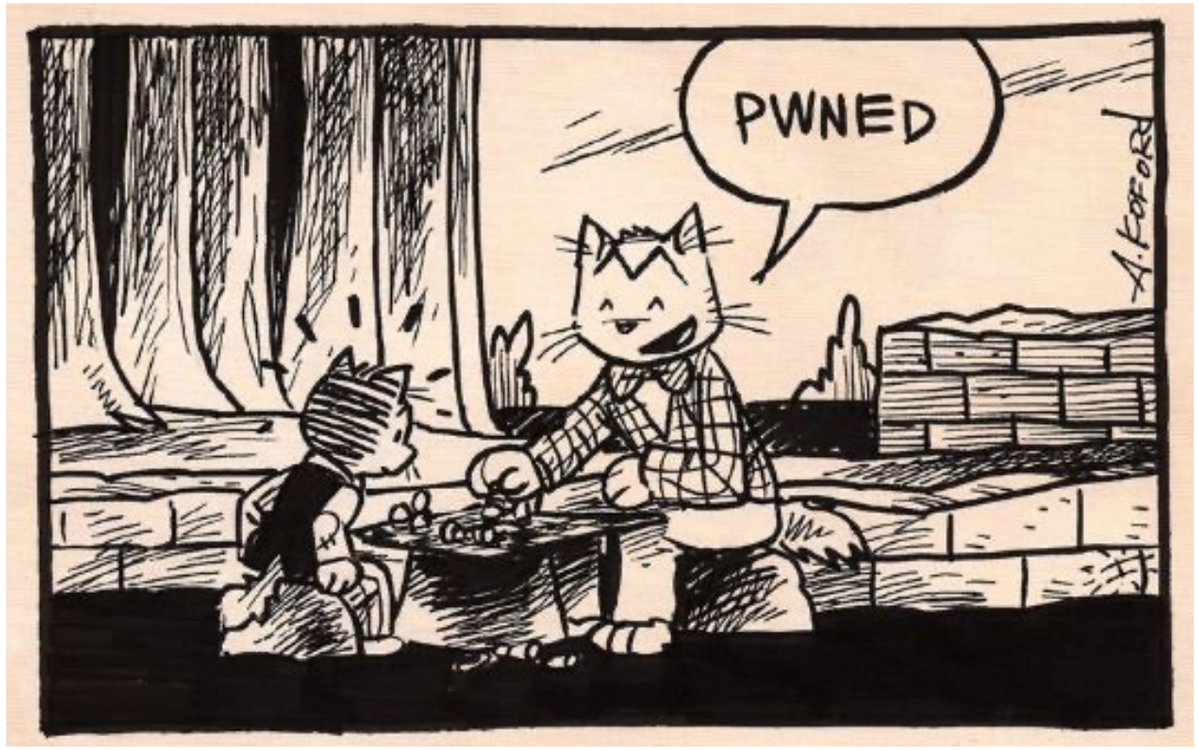
pwn 专题 02 – ROP (Return-Oriented Programming



Acknowledgement: Some slides are based on Zhi Wang (FSU)'s slides.

OOP 都还没学就要学 ROP？

有关于面向 XX 编程，关键是考虑 **侧重点**

如大家熟悉的 C 语言，号称 面向 ***过程*** 编程
其含义是指你在写 C 程序时主要是 **考虑实现功能的过程**

- 怎么写循环、怎么写判断才能达到预期功能

未来学习面向对象（OOP）时候，就要考虑如何设计类和对象，设计不同类之间的关系，等

而 ROP，其核心在于考虑如何组装 gadgets，如何利用 ret 指令来编程

Outline

- ret2text
- ret2sc
- well, what are gadgets
- ret2plt
- ret2libc
- ret2csu
- stack pivot

还记得上周的 example3 么

example2 单纯覆盖返回地址 crash 程序有啥意思，但如果 example3 里栈上没有指针的话，我们还能劫持控制流么

```c
#include <stdio.h>
#include <stdlib.h>

void backdoor()
{
    printf("Hi Backdoor\n");
    system("/bin/sh");
}

void normal()
{
    printf("Hi Normal\n");
    return;
}

int main(int argc, char *argv[])
{
    // void (*ptr)(void) = normal;
    char buffer[32];
    gets(buffer);
    puts(buffer);
    // ptr();
    return 0;
}
```

ret2text - demo1

栈上保留的返回地址就是 code pointer

ret2text

overflow ret to text

劫持返回地址 到 text段上已有的函数

```c
#include <stdio.h>
#include <stdlib.h>

void backdoor()
{
    printf("Hi Backdoor\n");
    system("/bin/sh");
}

void vul()
{
    char buffer[32];
    gets(buffer);
}

int main(int argc, char *argv[])
{
    vul();
}
```

PIE 保护简介

What you need to know:

- 整个程序 image 加载地址随机，所以各个段（.text 如函数地址、.data 全局变量地址等）都变成未知的了（除非有手段可以泄露，如上节课的 fsb）

```c
#include <stdio.h>

int main()
{
    printf("main address is %p\n", main);
    return 0;
}
```

基础课没讲的 stack canary (stack cookie)

其实在分析汇编的时候已经有发现

```
000000000000066a <func>:
 66a:   55                          push   rbp
 66b:   48 89 e5                    mov    rbp,rsp
 66e:   48 83 ec 50                 sub    rsp,0x50
 .......
 67b:   64 48 8b 04 25 28 00        mov    rax,QWORD PTR fs:0x28
 682:   00 00
 684:   48 89 45 f8                 mov    QWORD PTR [rbp-0x8],rax
 688:   31 c0                       xor    eax,eax
 .......
 6d1:   48 8b 75 f8                 mov    rsi,QWORD PTR [rbp-0x8]
 6d5:   64 48 33 34 25 28 00        xor    rsi,QWORD PTR fs:0x28
 6dc:   00 00
 6de:   74 05                       je     6e5 <func+0x7b>
 6e0:   e8 5b fe ff ff              call   540 <__stack_chk_fail@plt>
 6e5:   c9                          leave
 6e6:   c3                          ret
```

ret2sc - demo2

okay，那现在假设没有 backdoor 函数呢（多数时候是没有的）

```c
#include <stdio.h>
#include <stdlib.h>

void vul()
{
    char buffer[0x10];

    printf("buffer address: %p\n", buffer);

    gets(buffer);
}

int main(void)
{
    vul();

    return 0;
}
```

well 那如果栈不可执行呢

- DEP (W^X)
  Data Execution Prevention

攻与防的演替 ☺
至此，想简单的往 target 中注入 malicious code 的路给堵上了

那就复用代码中已有的“合法”片段组装攻击

也是 ROP 往往在实战中的操作

what are gadgets

在 ROP 中，gadgets 是用来构造 ROP chain 的，一些定义

- ROP gadgets are **sequences of CPU instructions** that are already present in the program being exploited or its loaded shared libraries and can be used to execute almost any **arbitrary code**;
- ROP gagdgets **most often end with the ret instruction;**
- ROP gadgets **bypass the DEP (NX bit protection),** since there is no executable code being injected to and executed from the stack, instead existing executable code is used to achieve the same malicious intent.

ROPgadget or ropper

- https://github.com/JonathanSalwan/ROPgadget

suppose pwntools 安装的时候已经顺便装好 ROPgadget 了，敲一下 ROPgadget 试试

```
ROPgadget --binary XXX
```

```
0x0000000000440608 : mov dword ptr [rdx], ecx ; ret
0x00000000004598b7 : mov eax, dword ptr [rax + 0xc] ; ret
0x0000000000431544 : mov eax, dword ptr [rax + 4] ; ret
0x000000000045a295 : mov eax, dword ptr [rax + 8] ; ret
0x00000000004a3788 : mov eax, dword ptr [rax + rdi*8] ; ret
0x0000000000493dec : mov eax, dword ptr [rdx + 8] ; ret
0x00000000004a36f7 : mov eax, dword ptr [rdx + rax*8] ; ret
0x0000000000493dc8 : mov eax, dword ptr [rsi + 8] ; ret
0x000000000043fbeb : mov eax, ebp ; pop rbp ; ret
0x00000000004220fa : mov eax, ebx ; pop rbx ; ret
0x0000000000495b90 : mov eax, ecx ; pop rbx ; ret
0x0000000000482498 : mov eax, edi ; pop rbx ; ret
0x0000000000437c11 : mov eax, edi ; ret
0x000000000042cfa1 : mov eax, edx ; pop rbx ; ret
0x000000000047d484 : mov eax, edx ; ret
0x000000000043de7e : mov ebp, esi ; jmp rax
0x0000000000499461 : mov ecx, esp ; jmp rax
0x00000000004324fb : mov edi, dword ptr [rbp] ; call rbx
0x0000000000443f34 : mov edi, dword ptr [rdi + 0x30] ; call rax
0x00000000004607e2 : mov edi, dword ptr [rdi] ; call rsi
0x000000000045c71e : mov edi, ebp ; call rax
0x0000000000491e33 : mov edi, ebp ; call rdx
0x00000000004a7a2d : mov edi, ebp ; nop ; call rax
0x000000000045c4c1 : mov edi, ebx ; call rax
```

with gadgets, lets ret2plt - demo3

prepare arguments and return to system plt

```c
#include <stdio.h>
#include <stdlib.h>

char string[] = "/bin/sh";

void prepare()
{
    system("echo hello");
}

void vul()
{
    char buffer[32];
    gets(buffer);
}

int main(int argc, char *argv[])
{
    vul();

    return 0;
}
```

当然，如果 system 压根就不在 plt 的时候呢

ret2libc - demo4

无论如何，system函数就在 libc 里面，不离不弃

- 需要泄露 libc (demo里假设已经完成泄露)
- prepare arguments and return to system libc

```c
#include <stdio.h>
#include <stdlib.h>

char string[] = "/bin/sh";

void vul()
{
    char buffer[32];

    printf("gets address: %p\n", printf);

    gets(buffer);
}

int main(int argc, char *argv[])
{
    vul();

    return 0;
}
```

呜呜呜参数准备好麻烦

- 假设 binary 中没有 /bin/sh
  - 那还需要泄露栈地址自行布局字符串

有没有一个地址像后门 backdoor 一样，跳过去就能启shell完成利用 ???

onegadget

OneGadget

When playing ctf pwn challenges we usually need the one-gadget RCE (remote code execution), which leads to call `execve('/bin/sh', NULL, NULL)`.

一个地址，完成所有梦想

concerns

- 结合前面的例子 demo 一下
- 为啥 libc 里会有这样的“危险”代码
- one gadget 使用的限制

参数控制的技巧

跳到后门不需要参数，跳到 system 只需要准备一个参数
那如果需求是准备更多参数呢?

- 比如程序通过沙箱保护，无法启shell (execve系统调用被禁止)
  - ORW
- 比如需要从 ROP 转化为 shellcode，需要修改页面属性
  - mprotect

of course, 常规而言就多找几个 pop gadgets 咯

```
pop rdi; ret;
 -> pop rsi; ret;
  -> pop ......
```

ret2csu - demo5

`__libc_csu_init`: 几乎所有未阉割的 ELF 都存在这么一个看似没用的函数

```c
#include <stdio.h>
#include <stdlib.h>

void prepare()
{
    system("echo hello");
}

void vul()
{
    char buffer[32];
    read(0, buffer, 0x200);
}

int main(int argc, char *argv[])
{
    prepare();
    vul();

    return 0;
}

```

溢出不够？stack pivot - demo6

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

char string[0x100];

void prepare()
{
    system("echo who are you?");
    read(0, string, 0x100);
}

void vul()
{
    char buffer[16];
    read(0, buffer, 0x20);
}

int main(int argc, char *argv[])
{
    prepare();
    vul();
}
```