

CTF线下赛中常用的PWN题patch方法

2018-07-02

CTF Linux PWN

在国赛以后，突然发现对PWN题中的patch方法了解不太深入，尤其是不够优雅，经常就用IDA直接手改了，或者就是用加一个section的方法，导致patch后的文件改动很大，尤其是在国赛中，被主办方打电话过来问是不是加了通防，本文就简单介绍一下常用的patch方法。

IDA

IDA Pro是一个非常强大的工具，其中包含了对汇编指令修改的功能。

以国赛华北赛区的半决赛为例，其中有一道PWN2是一个栈溢出，代码是这样的。

```
unsigned __int64 __fastcall play(int a1)
{
    char s; // [rsp+10h] [rbp-140h]
    unsigned __int64 v3; // [rsp+148h] [rbp-8h]

    v3 = __readfsqword(0x28u);
    bzero(&s, 0x12CuLL);
    write(a1, "Just have fun~~~~~\n", 0x13uLL);
    write(a1, ">", 1uLL);
    read(a1, &s, 0x1CCuLL);
    write(a1, &s, 0x144uLL);
    return __readfsqword(0x28u) ^ v3;
}
```

很显然，在read这里有一个明显的栈溢出，修复漏洞的方法也和容易，将这个值改小成0x138就好了，下面的write也一样的改法。

P4nda

search...

文章目录

隐藏目录

- 1. IDA
- ▼ 2. lief
 - 2.1. 增加segment
 - 2.2. 增加library
 - 2.3. 修改程序.eh_frame段

P4nda

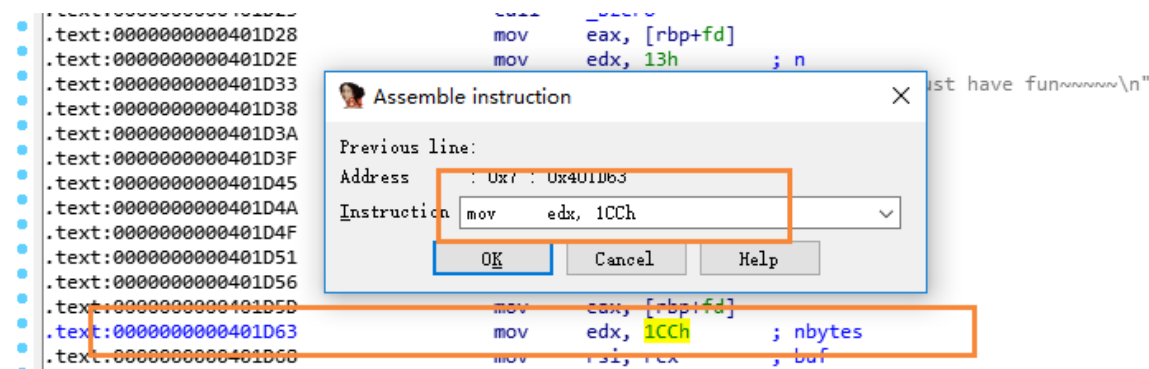
search...

文章目录

隐藏目录

- 1. IDA
- ▼ 2. lief
 - 2.1. 增加segment
 - 2.2. 增加library
 - 2.3. 修改程序.eh_frame段

这里使用IDA默认的修改插件来改，在Edit-Patch Program目录下，首先切换到IDA View-A这个汇编指令界面，并选中要改的汇编指令行：



选择Assemble/Change byte/Change word都可以，以Assemble为例在Instruction窗口，将mov edx, 1cch改为mov edx, 138h。

此时，切换到类C语言窗口可以看到该行已经被修改为了read(a1, &s, 0x138uLL);

```
unsigned __int64 __fastcall play(int a1)
{
    char s; // [rsp+10h] [rbp-140h]
    unsigned __int64 v3; // [rsp+148h] [rbp-8h]

    v3 = __readfsqword(0x28u);
    bzero(&s, 0x12CuLL);
    write(a1, "Just have fun~~~~~\n", 0x13uLL);
    write(a1, ">", 1uLL);
    read(a1, &s, 0x138uLL);
    write(a1, &s, 0x144uLL);
    return __readfsqword(0x28u) ^ v3;
}
```

但并没有完，这仅仅修改了IDA对于该文件的数据库，并没有应用到文件中去，同样在Edit-Patch Program目录下，选择Apply patches into file...，将修改写入文件，就完

P4nda

search...

文章目录

隐藏目录

- 1. IDA
- ▼ 2. lief
 - 2.1. 增加segment
 - 2.2. 增加library
 - 2.3. 修改程序.eh_frame段

成了一道简单题目的patch。

```
unsigned __int64 __fastcall play(int a1)
{
    char s; // [rsp+10h] [rbp-140h]
    unsigned __int64 v3; // [rsp+148h] [rbp-8h]

    v3 = __readfsqword(0x28u);
    bzero(&s, 0x12CuLL);
    write(a1, "Just have fun~~~~~\n", 0x13uLL);
    write(a1, ">", 1uLL);
    read(a1, &s, 0x138uLL);
    write(a1, &s, 0x144uLL);
    return __readfsqword(0x28u) ^ v3;
}
```

这种方法完全依靠手动，而且不能修改文件结构，可以供手动修改的位置也很少，一旦出现如UAF等悬垂指针的问题基本就很难解决了，还得依靠其他更有力的方法来解决。

lief

lief是一个开源的跨平台的可执行文件修改工具，链接如下：

```
1 https://github.com/lief-project/LIEF
```

对外提供了Python、C++、C的接口。

对于Python库安装可以使用pip，如

```
1 sudo pip install lief
```

对于lief的API和用法就不介绍了，RTFM。

P4nda

search...

文章目录

隐藏目录

1. IDA

▼ 2. lief

2.1. 增加segment

2.2. 增加library

2.3. 修改程序.eh_frame段

1 <https://lief.quarkslab.com/doc/latest/api/python/index.htm>

以下是几种可行的patch方法

—— 增加segment ——

这个方法的目的是增加一个程序段，在这个程序段中加入一个修复漏洞的程序代码，一般程序会在call某个函数时触发漏洞，一般语句为call 0x8041234，可以劫持这句话的逻辑，改成call我们定义的修复函数。

首先我们的代码程序如下：

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main(int argc, char** argv) {
4     printf("/bin/sh%d",102);
5     puts("let's go\n");
6     printf("/bin/sh%d",102);
7     puts("let's gogo\n");
8     return EXIT_SUCCESS;
9 }
```

我们想把第一处printf修改掉，改成我们自己的逻辑，首先需要编译一个包含实现patch函数的静态库，比如：

```
1 void myprintf(char *a,int b){
2     asm(
3         "mov %rdi,%rsi\n"
4         "mov $0,%rdi\n"
5         "mov $0x20,%rdx\n"
6         "mov $0x1,%rax\n"
7         "syscall\n"
```

P4nda

search...

文章目录

隐藏目录

1. IDA

▼ 2. lief

2.1. 增加segment

2.2. 增加library

2.3. 修改程序.eh_frame段

```
8      , ,
9  }
10 void myputs(char *a){
11     asm(
12         "push $0x41414141\n"
13         "push $0x42424242\n"
14         "push %rsp\n"
15         "pop  %rsi\n"
16         "mov  $0,%rdi\n"
17         "mov  $0x20,%rdx\n"
18         "mov  $0x1,%rax\n"
19         "syscall\n"
20         "pop  %rax\n"
21         "pop  %rax\n"
22     );
23 }
24 //gcc -Os -nostdlib -nodefaultlibs -fPIC -Wl,-shared hook
```

如上，将printf改成了write(0,"/bin/sh%d",0x20)，利用注释的gcc命令将其编译。

patch程序的流程是首先将代码段加入到binary程序中，然后修改跳转逻辑，将call printf@plt，改成call myprintf。

lief中提供了add参数可以用于为二进制文件增加段：

```
1 binary    = lief.parse(binary_name)
2 lib       = lief.parse(lib_name)
3 segment_add = binary.add(lib.segments[0])
```

在修改跳转语句部分，由程序的call执行寻址方法是相对寻址的，即call addr = EIP + addr

因此需要计算写入的新函数距离要修改指令的偏移，计算方法如下：

P4nda

search...

文章目录

隐藏目录

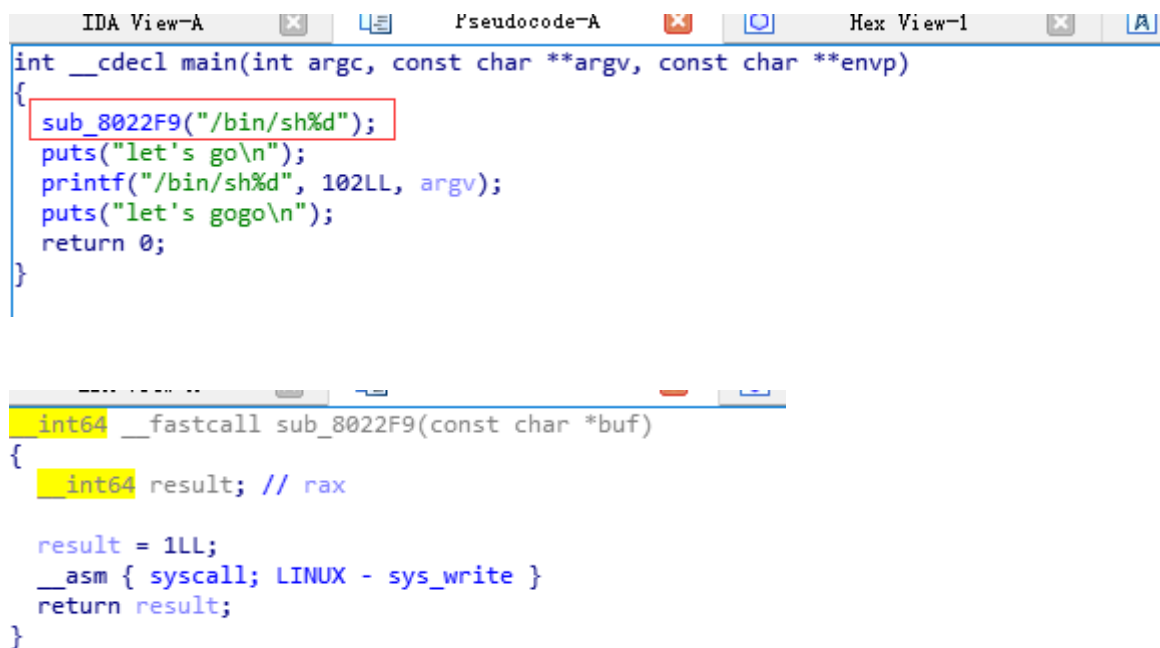
- 1. IDA
- ▼ 2. lief
 - 2.1. 增加segment
 - 2.2. 增加library
 - 2.3. 修改程序.eh_frame段

```
1 call xxx =(addr of new segment + offset function ) - (a
```

由于偏移地址是补码表示的，因此在用python计算时需要对结果异或0xffffffff，最终patch计算函数如下：

```
1 def patch_call(file,where,end,arch = "amd64"):  
2     print hex(end)  
3  
4     length = p32((end - (where + 5 )) & 0xffffffff)  
5     order = '\xe8'+length  
6     print disasm(order,arch=arch)  
7     file.patch_address(where,[ord(i) for i in order])
```

执行之后可以看到patch成功了，



```
IDA View-A | Pseudocode-A | Hex View-1  
int __cdecl main(int argc, const char **argv, const char **envp)  
{  
    sub_8022F9("/bin/sh%d");  
    puts("let's go\n");  
    printf("/bin/sh%d", 102LL, argv);  
    puts("let's gogo\n");  
    return 0;  
}  
  
int64 __fastcall sub_8022F9(const char *buf)  
{  
    int64 result; // rax  
  
    result = 1LL;  
    __asm { syscall; LINUX - sys_write }  
    return result;  
}
```

P4nda

search...

文章目录

隐藏目录

- 1. IDA
- ▼ 2. lief
 - 2.1. 增加segment
 - 2.2. 增加library
 - 2.3. 修改程序.eh_frame段

但是一个重大的问题是patch前后文件大小改动很大：

```
1 [p4nda@p4nda-virtual-machine] - [~/Desktop/pwn/patch] -  
2 [ ] <> python 1.py  
3 0x8022f9  
4 0: e8 70 1d 40 00          call    0x401d75  
5 [+] ori size 8656  
6 [+] patch size 15885  
7 [+] Seccessful patched in adding segment
```

这样在一些线下赛中很容易由于修改过大和被判定为通防或者宕机。

—— 增加library ——

这是借鉴LD_preload的一种思路，当程序中加载两个库时，在调用某一函数在两个库内同名存在时，是有一定查找顺序的，也就是可以实现，在不修改程序正常代码的前提下，对全部libc函数进行hook。如下例：

```
1 int __cdecl main(int argc, const char **argv, const char >  
2 {  
3     printf("/bin/sh%d", 102LL, envp, argv);  
4     return 0;  
5 }
```

编译一个动态链接库

```
1 // #include "/home/p4nda/linux-4.17.3/lib/syscall.c"  
2  
3 #define _GNU_SOURCE  
4 // #include <stdio.h>  
5 #include <sys/stat.h>  
6 #include <unistd.h>
```

P4nda

search...

文章目录

隐藏目录

1. IDA

▼ 2. lief

2.1. 增加segment

2.2. 增加library

2.3. 修改程序.eh_frame段

```
7 // #include <stdio.h>
8 // gcc -nostdlib -nodefaultlibs -fPIC -Wl,-shared patch.
9
10
11 int printf(char *a,int b) {
12     char str[] = "hacked by me\n ";
13     //puts(a);
14     if(strstr(a,"/bin/sh")){
15         puts("find dangerous str~");
16     }
17     int (*old_printf)(char *,int);
18     old_printf =(int (*)(char *,int)) dlsym(RTLD_NEXT,
19     old_printf(a,b);
20     puts("\n");
21
22 }
```

编译命令在注释中，则每次printf时都会先执行上述库中的函数，达到hook的目的。

```
sample.write('sample_add_lib')%
[p4nda@p4nda-virtual-machine] - [~/Desktop/pwn/patch] - [一 7月 02, 22
[$] <> ./sample_add_lib
find dangerous str~
/bin/sh102
```

优势很明显，可以执行任意libc内函数代码，让编程更容易。

不过缺点也很明显，首先程序变得巨大，并且当不存在这个静态链接库的时候，程序跑不起来... 有些线下赛都是本地check的，比如*网杯，很容易就判断宕机了...

P4nda

search...

文章目录

隐藏目录

- 1. IDA
- ▼ 2. lief
 - 2.1. 增加segment
 - 2.2. 增加library
 - 2.3. 修改程序.eh_frame段



—— 修改程序.eh_frame段 ——

在TSCTF 2018 Final时，我在NeSE战队的binary文件中找到了通防工具，但是程序改动并没有特别大，当时感觉很好奇，在赛后调试了一下，发现他们把通防的shellcode写在了一个叫.eh_frame的段中，这个段会加载到程序中来，并且自身带有可执行权限，在查找这个段用处时，发现该段对程序执行影响不大，故可以将patch代码写在这个段中，再用跳转的方法将程序逻辑劫持到这里来。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    sub_400698("/bin/sh%d");
    sub_4006B3("let's go\n", 102LL);
    printf("/bin/sh%d", 102LL, argv);
    sub_4006B3("let's gogo\n", 102LL);
    return 0;
}
```

P4nda

search...

文章目录

隐藏目录

1. IDA

▼ 2. lief

2.1. 增加segment

2.2. 增加library

2.3. 修改程序.eh_frame段

```
eh_frame:000000000400698 ; __libc_start_main @plt
eh_frame:000000000400698 sub_400698 proc near ; CODE XREF: main+1Efp
eh_frame:000000000400698 mov rsi, rdi ; buf
eh_frame:000000000400698 mov rdi, 0 ; fd
eh_frame:0000000004006A2 mov rdx, 20h ; count
eh_frame:0000000004006A9 mov rax, 1
eh_frame:0000000004006B0 syscall ; LINUX - sys_write
eh_frame:0000000004006B2 ret
eh_frame:0000000004006B2 sub_400698 endp
eh_frame:0000000004006B3 ; ===== S U B R O U T I N E =====
eh_frame:0000000004006B3 sub_4006B3 proc near ; CODE XREF: main+28fp
eh_frame:0000000004006B3 ; main+46fp
eh_frame:0000000004006B3 push 41414141h
eh_frame:0000000004006B8 push 42424242h
eh_frame:0000000004006BD push rsp
eh_frame:0000000004006BE pop rsi ; buf
eh_frame:0000000004006BF mov rdi, 0 ; fd
eh_frame:0000000004006C6 mov rdx, 20h ; count
eh_frame:0000000004006CD mov rax, 1
eh_frame:0000000004006D4 syscall ; LINUX - sys_write
eh_frame:0000000004006D6 pop rax
eh_frame:0000000004006D7 pop rax
eh_frame:0000000004006D8 ret
eh_frame:0000000004006D8 sub_4006B3 endp
eh_frame:0000000004006D8
```

可以看到在patch前后，程序大小保持不变。

```
[p4nda@p4nda-virtual-machine] - [~/Desktop/pwn/patch] - [~]
[+] python test.py
[+] find .eh_frame : 0x400698
0x400698 0: e8 0f 01 00 00 call 0x114
0x4006b3 0: e8 02 01 00 00 call 0x107
0x4006b3 0: e8 20 01 00 00 call 0x125
[+] ori size 8656
[+] patch size 8656
[+] Successful patched in .eh_frame
```

缺点同样明显，.eh_frame的大小是有限的...

综上，似乎没有比较简洁的通用方法，综合着来用吧....

本文标题: CTF线下赛中常用的PWN题patch方法

文章作者: P4nda

发布时间: 2018-07-02, 19:26:13

最后更新: 2018-07-28, 11:54:38

原始链接: <http://p4nda.top/2018/07/02/patch-in-pwn/> 

许可协议: © "署名-非商用-相同方式共享 4.0" 转载请保留原文链接及作者。

P4nda

search...

← [【WCTF 2018】parrot_revenge 题解](#) [【PWNABLE.TW】wannaheap 解题思路](#) →



文章目录

隐藏目录

1. IDA

▼ 2. lief

2.1. 增加segment

2.2. 增加library

2.3. 修改程序.eh_frame段



在 P4NDA 上还有

【PWNABLE.TW】 wannaheap 解题思路

2年前 • 4条评论

一道比较新的题目，暂时不
开放了，有思路欢迎交流。

【KERNEL PWN】强网 杯CTF2018 core题解

2年前 • 8条评论

最近刚开始学习内核PWN相
关的东西， ...

Linux kernel 4.20 BPF 整数溢出漏洞分析

2年前 • 4条评论

Linu

2年前

漏洞
验室
赛中

P4nda

search...

文章目录

隐藏目录

- 1. IDA
- ▼ 2. lief
 - 2.1. 增加segment
 - 2.2. 增加library
 - 2.3. 修改程序.eh_frame段

0条评论

p4nda

🔒 Disqus 隐私政策

♥ 推荐

🐦 推文

f 分享



开始讨论...

通过以下方式登录

或注册一个 DISQUS 帐号 ?

姓名

来做第一个留言的人吧!

📧 订阅

🔒 在您的网站上使用 Disqus添加 Disqus添加

⚠ Do Not Sell My Data

P4nda

search...

文章目录

隐藏目录

- 1. IDA
- ▼ 2. lief
 - 2.1. 增加segment
 - 2.2. 增加library
 - 2.3. 修改程序.eh_frame段

