2020년 2학기

Assignment #4

과목 : 컴퓨터SW시스템개론

담당 교수 : 김종

학과 : 컴퓨터공학과

학번 : 20190439

이름 : 오승훈

povis id : sho0927

|  |
| --- |
| < 명예서약(Honor code) >  “나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.” |

**Problem : Attack Lab**

1. 문제의 개요

본 프로그램은 다음의 내용을 포함한다.

buffer overflow를 이용하여 Smashing attack, Code Injection attack, ROP attack 등을 직접 구현해본다. 현재 이 프로그램에서는 getbuf라는 함수에서 return address를 변경하여 attack을 하는 방법을 직접 사용해본다.

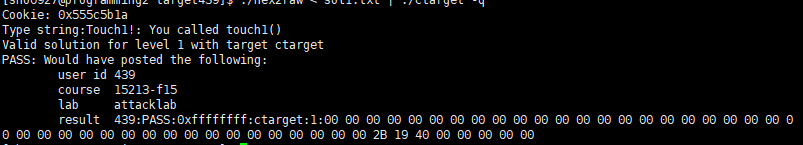
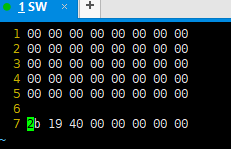
2. 프로그램 구조 및 설명

**1) Phase 1**

Getbuf라는 함수에서는 현재 0x28 즉 40byte의 stack memory를 사용하고 있다는 것을 확인할 수 있다. 그러므로 return address로 접근하기 위해서 padding을 넣어주고, return address를 넣어주면되는 문제가 된다.

1. padding을 넣어주는 이유는 현재 스택에서는 40byte가 할당이 되어 있는데 그것보다 주소값이 큰 곳에 return address가 저장이 되어있고, 이를 변경을 해주기 위해서는 padding이 40byte만큼 필요하게 된다.
2. Return address의 값은 touch1 함수의 address로 보내버리면 getbuf가 끝나고, return address로 touch1 함수의 address로 돌아가기 때문에 touch1 함수의 address인 0x40192b를 넣어주면 된다.

그러므로 다음과 같은 값을 넣어주면 되고, 그 값이 touch1을 호출한다는 것을 다음 사진을 통해 볼 수 있다.



현재 여기서 return address 부분을 2b 19 40 00 00 00 00 00으로 채워준 이유는 LSB가 낮은 주소값을 가지는 Little Endian 이기 때문에 다음과 같이 나타나고, touch1에 도달했다는 것을 알 수 있다.

**2) Phase\_2**

touch2함수를 호출하며, 인자로 %rdi에 cookie 값을 담아서 넘겨야 한다. 그렇기 때문에 rip 값을 현재 buffer의 값으로 넣어주고, 그 후 buffer에서 필요한 rdi 값을 넣어주고, return address를 또 다시 touch2로 향하게 해주면 된다는 것을 알 수 있다.

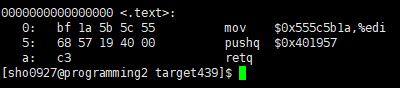
먼저 buffer의 위치를 알기 위해서 getbuf 함수에 break point를 걸어둔 후 rsp register의 메모리 위치를 찾아보면 0x55642ab0이 되고, 그렇다면 buffer의 시작 위치는 0x28을 뺀 0x55642a88이 된다. 그렇다면 padding 이후 들어갈 값에 대해선 88 2a 64 55 00 00 00 00이 return address 자리에 들어가면 된다는 것을 알 수 있다.

이후에는 rdi에 0x555c5b1a 값을 mov 시켜준 후 return address를 stack에 push 시켜주고 return을 시켜주면 된다는 것을 알 수 있다.

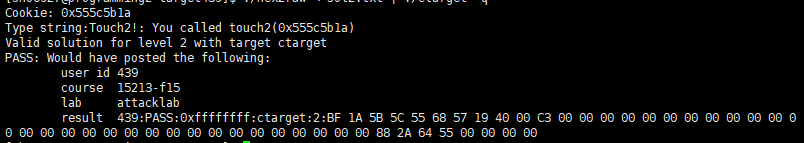
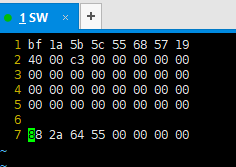
mov $0x555c5b1a, %rdi

pushq 0x401957

ret 을 machine code로 바꿔주면 다음과 같은 결과를 얻을 수 있고,



이를 코드에 padding을 포함하여 넣어주면 다음과 같은 답과 touch2에 도달하는 모습을 볼 수 있다.



**3) Phase\_3**

phase 2와 마찬가지로 rdi에 인자를 넣어준 후 touch3를 호출해야 한다. 하지면 여기서 touch2와 다른 점은 인자를 char형과 비교를 한다는 것이다. 그렇기 때문에 이번에는 인자로 문자열의 포인터를 주어야 하므로, address 정보를 주고, 그 address에 이 char 값들을 넣어두면 된다는 것이다.

mov $0x555c5b1a, %rdi

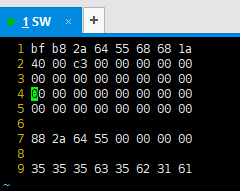
pushq 0x401957

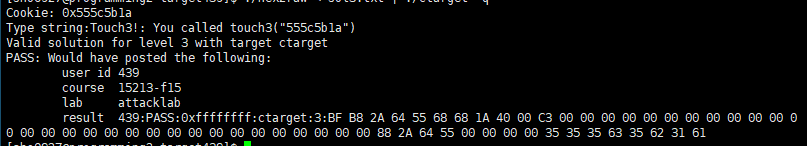
ret 과 비슷하지만 rdi의 값으로 두 번째 address 영역의 주소를 이용할 것이다. 그렇기 때문에 rsp 값이 0x55642ab0 였고, 여기서 두 번째 address를 이용할 것이기 때문에 여기에 0x8을 추가한 값을 rdi 값에 mov 시켜주면 된다. 그러므로

mov $0x55642ab8, %rdi

pushq 0x401a68

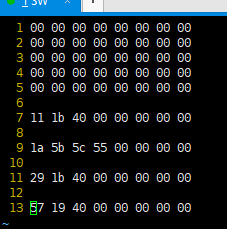
ret을 machine code로 변환을 해주면 다음과 같은 code injection을 해주면 된다는 것을 알 수 있고, padding을 넣어주면 다음과 같은 결과가 나온다는 것을 알 수 있다.

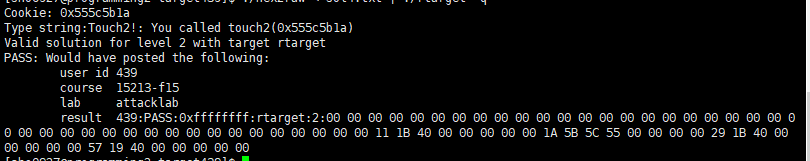




**4) Phase\_4**

Phase4부터는 stack 영역이 executable하지 못하도록 막아뒀기 때문에 ROP를 이용하여 문제를 해결하라는 것이다. 그렇다면 우리는 phase2와 같은 작업을 ROP를 통해서 해결하면 된다. 여러가지 gadget들을 모아서 필요한 연산을 수행할 수 있도록 만들면 되는데, 현재 필요한 연산은 mov rdi를 해주는 연산과 pushq를 해줄 수 있는 연산이 있으면 된다는 것을 알 수 있다. 현재 gadget들 중 setval\_453 이라는 함수에서 58 c3이라는 연산이 존재하는 것을 볼 수 있다. 이는 popq rax 연산인데 그러면 stack에 cookie 값을 넣어둔 후 rax에 pop을 통해 넣어주고, 그 후 rax -> rdi로 mov를 해주는 연산이 존재하면 된다는 것을 알 수 있다. 48 89 c7이 있는 연산을 찾아보면 setval\_314 함수에 48 89 c7 c3가 존재하는 것을 볼 수 있고, 그 후 최종적으로 돌아가려는 touch2의 주소를 넣으면 된다. 그렇다면 패딩을 40 byte를 넣어주고, 그 후 우리가 모은 gadget들을 하나씩 쌓으면 되기 때문에

 옆과 같은 결과가 나오고 9번째 줄은 쿠키 값을 pop해줄 값을 stack에 넣어주는 것이다. 이 값을 실행해보면 다음과 같은 결과를 초래한다는 것을 알 수 있다.



**5) Phase\_5**

phase5는 ROP를 이용하여 phase3와 같은 작업을 수행해야 한다. 그러기 위해서는 우리가 쓸만한 연산들이 어떤 것들이 존재하는지를 먼저 gadget을 파악을 해봐야하는데 우리가 필요한 연산은 stack의 return address 부분에 존재하는 cookie의 memory 주소를 찾을 수 있게 해주는 연산과 그 memory 주소를 rdi에 넣어줄 수 있는 연산이 최종적으로 필요한 것이다.

40 1b 11 popq rax (58)

40 1b d4 movl eax, edx (89 c2)

40 1b a0 movl edx, ecx (89 d1)

40 1b 93 movl ecx, esi (89 ce)

40 1b 7f movq rsp, rax (48 89 e0)

40 1b 29 movq rax, rdi (49 89 c7)

40 1b a0 lea (rdi, rsi, 1), rax

40 19 b8 movq rax, rdi (48 89 c7)

먼저 40 1b a0의 값을 보면 lea(rdi,rsi,1),rax가 있는 것을 확인 할 수 있다. 그렇다면 우리가 필요한 것은 return address 값들 사이에 넣어둔 cookie의 char형 memory 주소를 rsp + offset을 해준 값에 대한 연산을 해줄 수 있어야하는 것이고, rdi 또는 rsi에 각각의 rsp 값과 offset을 넣어줄 수 있다면 모든게 해결되는 문제이다. 그러고 난 후 gadget들을 살펴보면

40 1b 11 popq rax (58)

40 1b d4 movl eax, edx (89 c2)

40 1b a0 movl edx, ecx (89 d1)

40 1b 93 movl ecx, esi (89 ce)

여기서 rax에 pop을 해온 후 esi 까지 값을 옮길 수 있다는 것을 볼 수 있으므로 여기에 offset의 값을 stack에 넣어준 후 pop을 시켜 rax에 넣어주고, esi까지 값을 옮기면 된다는 것을 볼 수 있다. 그렇다면 rsp register의 값을 옮겨주는 연산에는

40 1b 7f movq rsp, rax (48 89 e0)

40 1b 29 movq rax, rdi (49 89 c7)

이 있으므로 rdi로 rsp 값을 옮길 수 있다는 것 역시 알 수 있다. 그후 rax 값을 다시 rdi로 옮겨주는 연산인

40 19 b8 movq rax, rdi (48 89 c7) || 40 1b 29를 통해서 rax 값을 rdi로 옮겨주면 된다는 것을 알 수 있다. 그렇다면 현재 모아둔 gadget들의 함수 address들을 padding한 후의 return address stack 영역에 넣어주면 되고, 이때 필요한 것은 offset 계산이 필요하게 되는데 40 1b 7f를 해줄 때의 rsp에 비해 우리가 가장 마지막에 넣어줄 cookie의 값의 address가 0x20만큼 증가를 하는 것을 볼 수 있기 때문에 우리의 offset은 0x20이 되고, 그렇다면 다음과 같은 순서로 넣어주면 된다.

Padding

40 1b 11 popq rax (58)

Offset 0x20

40 1b d4 movl eax, edx (89 c2)

40 1b a0 movl edx, ecx (89 d1)

40 1b 93 movl ecx, esi (89 ce)

40 1b 7f movq rsp, rax (48 89 e0)

40 1b 29 movq rax, rdi (49 89 c7)

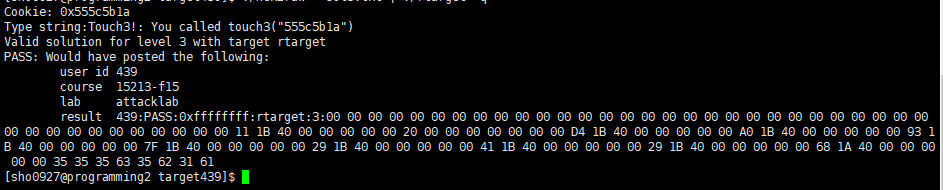
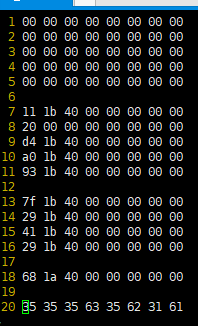
40 1b a0 lea (rdi, rsi, 1), rax

40 19 b8 movq rax, rdi (48 89 c7)

Touch3의 주소 0x401a68

Cookie의 character 값

그러므로 다음과 같은 코드를 넣어주면 된다.



다음과 같은 결과가 나온다는 것 역시 볼 수 있다.

3. 토론

Code injection attack에 대해서 개념을 잘 이해하고 있다고 생각했었는데 rip 값이 return address 값을 가리킨다는 개념을 잘 이해를 못하고 있었던 것 같다. 그래서 return address에 buffer의 주소를 넣어준다는 것을 이해하는데 까지 오랜 시간이 걸렸던 것 같다. 또한 touch3는 문제를 이해하는데 상당한 시간이 걸렸던 것 같고, character 형과 int형의 비교를 할 때 왜 그런 차이가 있는지에 대한 이해를 하는데도 상당한 시간이 걸렸던 것 같다.

또한 ROP에서는 gadget들을 farming 하는 것이 눈에 잘 들어오지 않아 machine code들과 하나하나 비교를 하는 것이 힘들었던 과제인 것 같다. 하지만 ROP의 gadget farming이라는 개념과 rip 값이 return address를 가리킨다는 내용 그리고 stack의 구조가 어떻게 형성이 되어있는지 등에 대해서는 확실하게 이해를 하는데 도움이 된 과제라고 생각한다.