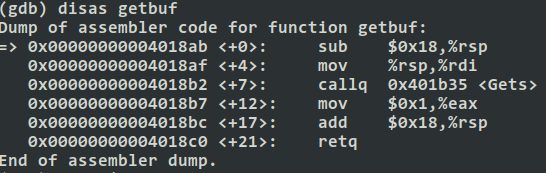
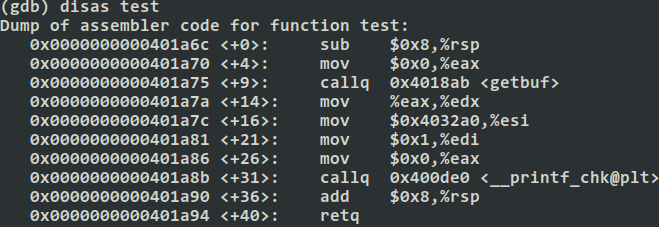
이름 : 김창민

학번 : 201329119

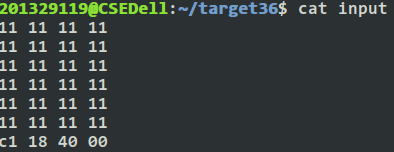
1. phase\_1



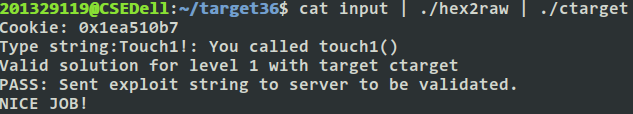
phase\_1의 목표는 getbuf함수가 리턴하면서 0x00401a7a가 아니라 touch1을 실행하게끔 하는것이다.



return address까지 0x18 = 24바이트의 여유 공간을 채워넣은 후에 touch1의 주소인 0x004018c1를 삽입하면 된다.

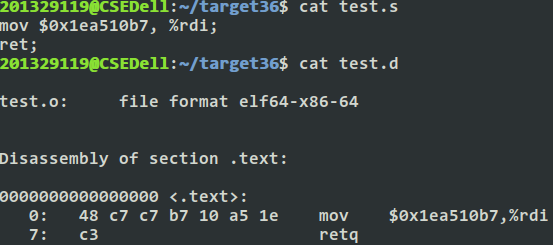


little-endian 방식에 따라 input 파일을 위와 같이 구성하면,

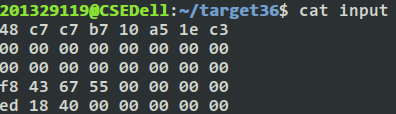


2. phase\_2

여전히 getbuf 리턴 후의 주소는 0x401a7a이고 버퍼의 크기는 0x18 = 24바이트인 상황에서, 원하는 함수를 호출하는것 뿐만 아니라 원하는 인자를 전달하려면 레지스터에 값 이동 연산을 하는 코드를 버퍼에 injection 해야한다. 인자 전달 후, return instruction을 수행하여 rip가 touch2함수를 가리키도록 해야한다. getbuf함수의 return address를 주입한 코드의 주소가 되도록 하고, 주입하는 코드를 레지스터 전달 및 리턴문으로 만들며, 스택 포인터가 가리키는 다음 위치에 touch2의 주소가 오게 하면 된다.



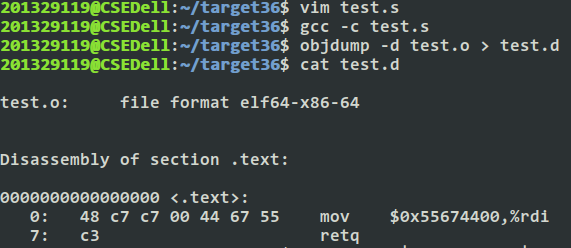
위와 같은 코드를 생성하고, 아래와 같이 input파일을 작성한다.



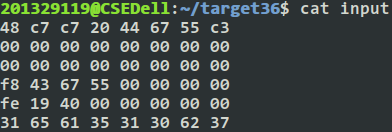
input 파일만 놓고 봤을 때 아래쪽이 높은 주소이고, 위쪽이 낮은 주소이다. 첫 줄 부터 세번째 줄까지 버퍼의 크기이며, 4번째 줄이 원래 getbuf함수의 return address가 존재해야 하는 곳이며, 직접 생성한 instruction을 가리키도록 getbuf함수에서 버퍼를 할당했을 때의 %rsp의 값으로 정했다. getbuf가 리턴할 때 injection code가 수행되는데, 삽입한 코드 중 앞의 7바이트가 cookie값을 %rdi로 전달하고, 8번째바이트의 c3이 수행되면 5번째 줄의 touch2함수를 가리키는 주소가 %rip에 할당될 것이다.

3. phase\_3

ASCII 표를 참조하여 쿠키문자열을 16진수로 변환한 값을 두 개의 return address 이후에 삽입하고, 그 문자열이 시작하는 주소(삽입한 주소)를 %rdi에게 인자로 넘겨주면 된다.



getbuf함수에서 버퍼공간을 할당하고 난 이후의 %rsp는 0x556743f8을 가리키고 있었고, 버퍼의 크기는 24바이트, 그리고 두 개의 return address를 위한 공간 16바이트가 있으므로 %rsp + 40 = 0x55674400이 문자열이 시작하는 주소가 될 것이다. 먼저 위 명령 코드를 버퍼의 최하위에 삽입한다. 첫 번째 return address는 이 명령을 가리키는 주소가 올 것이고, 두 번째 return address는 touch3을 가리키는 주소가 올 것이다.



첫 번째 줄 : 문자열의 시작주소를 %rdi에 전달하는 명령

두 번째, 세 번째 줄 : 버퍼 패딩

네 번째 줄 : 첫 번째 줄의 코드를 가리키는 주소

다섯 번째 줄 : touch3을 가리키는 주소

여섯 번째 줄 : cookie 문자열을 나타내는 16진수

위와 같이 입력한 결과 3단계를 통과할수 있었다.

4. phase\_4

phase\_4는 ROP를 활용하여 phase\_2와 같은 목표를 달성하기만 하면 된다. 먼저 encoding table을 참조했고 rtarget을 objdump하여 code farm을 살펴보았다.



살펴본 후 가젯의 실행순서를 다음과 같이 정했다.

1. pop %rax; ret; -> 삽입된 cookie를 pop해서 %rax에 저장하고 리턴
2. movq %rax, %rdi; ret; -> %rax에 있는 cookie를 %rdi로 옮김
3. touch2 실행

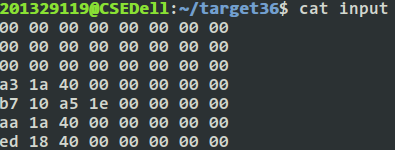
a는 58 c3으로부터 인코딩 됨을 참조하고 다음으로부터 얻어왔다.



b는 48 89 c7 c3으로부터 인코딩 됨을 참조하고 다음으로부터 얻어왔다.



그리고 input을 다음과 같이 셋팅하면



1~3 line : 패딩 버퍼 공간

4 line : a의 가젯 주소

5 line : cookie

6 line : b의 가젯 주소

7 line : touch2의 주소

5. phase\_5

cookie string을 return oriented programming 시퀀스가 끝난 이후에 넣는다고 가정하자. 그렇게 제일 높은 주소에 저장된 cookie를 참조하기 위해서 %rsp를 참조하고 %rsp를 참조하는 시점에서 cookie string까지의 offset을 알아낼 필요가 있다. 따라서 phase\_5에서는 원하는 목표를 달성하기 위해서 가젯들을 어떤식으로 배치할지에 대한 설계가 굉장히 중요하다.

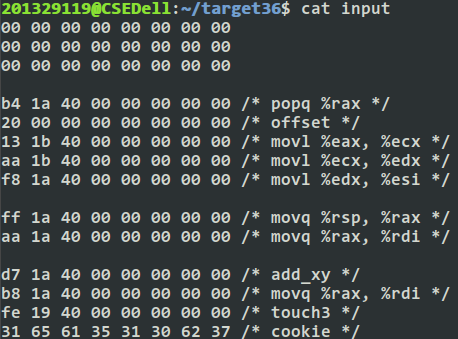
%rsp + offset이 cookie string을 가리키는 주소이므로, 둘을 더한 주소를 %rdi에 넘겨주는 것이 목표라고 할 수 있다. code farm에서 이러한 목표를 달성하기 위한 함수를 하나 골라보았다.



위의 함수를 사용하여 %rdi와 %rsi에 %rsp와 offset값을 넣어주어야 한다. offset은 임의의 값이라 생각하고 모든 설계가 끝난 후 %rsp값을 꺼내는 gadget instruction과 cookie string 사이의 거리를 넣어주면 된다.

%rdi와 %rsi에 각각의 값들을 옮겨주기 위해서 필요한 code segment를 code farm으로부터 찾을 때 기존의 movq를 사용하기에는 farm의 자원이 부족했고, movl의 code segment들을 찾을 수 있었다. 하지만 ROP를 위해서는 movl 이후에 바로 ret을 의미하는 c3이 와야하는데, 그런 코드들을 찾을수 없었다. 하지만 다행히도 movl instruction과 ret 사이에 레지스터의 값을 변경하지 않고 condition flag만 변경하는 비교 명령들이 존재하는 code segment가 있었기 때문에, CPU자원을 쓸데없는 곳에 낭비하더라도 원하는 목적을 달성하기 위해 이를 exploiting 할수 있었다.

완성된 input은 다음과 같다.



4~8 line : 9번째와 14번째 라인의 거리인 0x20만큼의 offset을 pop하여 %rax에 저장하고 %ecx, %edx, %esi에 최종적으로 옮김

9~10 line : %rsp의 값을 %rdi로 옮김

11~14 line : %rdi와 %rsi를 더하여 %rax에 저장하고, %rax를 %rdi로 옮김, touch3 호출

이로써 cookie string의 주소를 %rdi에 성공적으로 전달하고 touch3을 실행하는 목표를 이룰 수 있었다.