DEP & ASLR

ddaa

Review shellcode

```
 1 #include <stdio.h>
 2
 3 char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73"
 4                   "\x68\x68\x2f\x62\x69\x6e\x89"
 5                   "\xe3\x89\xc1\x89\xc2\xb0\x0b"
 6                   "\xcd\x80\x31\xc0\x40\xcd\x80";
 7
 8 int main()
 9 {
10   fprintf(stdout,"Lenght: %d\n",strlen(shellcode));
11   (*(void  (*)()) shellcode)();
12 }
```

But……

```
Program received signal SIGSEGV, Segmentation fault.
0x0804a024 in shellcode ()
(gdb) x/10i 0x0804a024
=> 0x804a024 <shellcode>:       xor    %eax,%eax
   0x804a026 <shellcode+2>:     push   %eax
   0x804a027 <shellcode+3>:     push   $0x68732f2f
   0x804a02c <shellcode+8>:     push   $0x6e69622f
   0x804a031 <shellcode+13>:    mov    %esp,%ebx
   0x804a033 <shellcode+15>:    mov    %eax,%ecx
   0x804a035 <shellcode+17>:    mov    %eax,%edx
   0x804a037 <shellcode+19>:    mov    $0xb,%al
   0x804a039 <shellcode+21>:    int    $0x80
   0x804a03b <shellcode+23>:    xor    %eax,%eax
(gdb)
```

Check memory maps

- cat /proc/$pid/maps

```
08048000-08049000 r-xp 00000000 fc:00 4200031   /home/dada/a.out
08049000-0804a000 r--p 00000000 fc:00 4200031   /home/dada/a.out
0804a000-0804b000 rw-p 00001000 fc:00 4200031   /home/dada/a.out
f7e23000-f7e24000 rw-p 00000000 00:00 0
f7e24000-f7fca000 r-xp 00000000 fc:00 4587531   /lib32/libc-2.19.so
f7fca000-f7fcc000 r--p 001a5000 fc:00 4587531   /lib32/libc-2.19.so
f7fcc000-f7fcd000 rw-p 001a7000 fc:00 4587531   /lib32/libc-2.19.so
f7fcd000-f7fd1000 rw-p 00000000 00:00 0
f7fd9000-f7fdb000 rw-p 00000000 00:00 0
f7fdb000-f7fdc000 r-xp 00000000 00:00 0         [vdso]
f7fdc000-f7ffc000 r-xp 00000000 fc:00 4587532   /lib32/ld-2.19.so
f7ffc000-f7ffd000 r--p 0001f000 fc:00 4587532   /lib32/ld-2.19.so
f7ffd000-f7ffe000 rw-p 00020000 fc:00 4587532   /lib32/ld-2.19.so
fffdd000-ffffe000 rw-p 00000000 00:00 0         [stack]
```

Data Execution Prevention

- It marks areas of memory as either “executable” or “nonexecutable”, and allows only data in an “executable” area to be run by programs, services, device drivers, etc. (Wikipedia)
- 資料執行防止 (Microsoft)
- WX 兩種權限不會同時存在
- NonExecutable => NX，又稱 NX protection

- gdb -z execstack test.c

```
dada@ubuntu:~$ gcc -m32 -z execstack test.c
test.c: In function ‘main’:
test.c:10:33: warning: incompatible implicit declaration of built-in
   fprintf(stdout,"Lenght: %d\n",strlen(shellcode));
                                 ^
dada@ubuntu:~$ ./a.out
Lenght: 28
$
```

Bypass DEP

- ret2text
  - Return to existing code
  - 想想 hw0 的 magic 應該就懂了
- ret2libc
  - 程式裡面沒有使用 system(“/bin/sh”) 是非常正常的一件事
  - 不過可能用到其他的 function，還是會 link glibc
  - 可以直接跳到 library 的位置執行
  - 就好像在 call function ，要先把 stack 放置好參數
- => Return Orient Program

ret2libc

ffff5000	char a
ffff5004	ebp
ffff5008	ret
ffff500c	arg1
ffff5010	????
ffff5014	????

```
void foo (char *arg1)
{
    char a;
    strcpy(&a,arg1);
    return;
}
```

ret2libc

ffff5000	AAAA
ffff5004	AAAA
ffff5008	addr of func
ffff500c	after lib call
ffff5010	ptr to “/bin/sh”
ffff5014	????

```
void foo (char *arg1)
{
    char a;
    strcpy(&a,arg1);
    return;
}
```

ret2libc demo

```
dada@ubuntu:~/example/ret2lib$ ldd ret2lib
        linux-gate.so.1 =>  (0xf770e000)
        libc.so.6 => /lib32/libc.so.6 (0xf7557000)
        /lib/ld-linux.so.2 (0xf770f000)
dada@ubuntu:~/example/ret2lib$ ldd ret2lib
        linux-gate.so.1 =>  (0xf77d3000)
        libc.so.6 => /lib32/libc.so.6 (0xf761c000)
        /lib/ld-linux.so.2 (0xf77d4000)
dada@ubuntu:~/example/ret2lib$ ldd ret2lib
        linux-gate.so.1 =>  (0xf76f0000)
        libc.so.6 => /lib32/libc.so.6 (0xf7539000)
        /lib/ld-linux.so.2 (0xf76f1000)
dada@ubuntu:~/example/ret2lib$
```

Address Space Layout Randomization

- 位址空間配置隨機載入
  - Random stack
  - Random heap
  - Radnom libraries
- /proc/sys/kernel/randomize_va_space
  - 0: Disable ASLR.
  - 1: Randomize the positions of the stack, VDSO page, and shared memory regions.
  - 2: 1 + data segment

Position Independent Executable

- position-independent code
- gcc -fpie -pie test.c

```
00000636 <main>:
 636:   55                      push   %ebp
 637:   89 e5                   mov    %esp,%ebp
 639:   53                      push   %ebx
 63a:   83 e4 f0                and    $0xfffffff0,%esp
 63d:   83 ec 10                sub    $0x10,%esp
 640:   e8 9b fe ff ff          call   4e0 <__x86.get_pc_thunk.bx>
 645:   81 c3 bb 19 00 00       add    $0x19bb,%ebx
 64b:   8d 83 00 e7 ff ff       lea    -0x1900(%ebx),%eax
 651:   89 04 24                mov    %eax,(%esp)
 654:   e8 b2 ff ff ff          call   60b <foo>
```

Under ASLR & PIE …

- ret2shellcode X
- ret2text X
- ret2libc X

- Because we must know where to return at first.

Conquer ASLR

1. Information leak
   - Then we can calculate offset.
2. fork
   - Everything won’t change, we can try to guess the correct address.