

DEP & ASLR

chchao

來測試一下shellcode

```
1
2 char shellcode[]="1\x0cPh//shh/bin\x89\xe3PYPZ\xb0\x0b\xcd\x80\xff";
3
4 int main(){
5     ((void (*)(()))shellcode)();
6 }
7
```

欸？

```
vmvm@ubuntu:~/test$  
vmvm@ubuntu:~/test$ vim shellcode.c  
vmvm@ubuntu:~/test$ gcc shellcode.c  
vmvm@ubuntu:~/test$ ./a.out  
Segmentation fault (core dumped)  
vmvm@ubuntu:~/test$
```

Data Execution Prevention (DEP)

- 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)
- 資料段不可以當作程式執行
- 程式段不可以寫入資料
- 2004年開始用在Linux (kernel 2.6.8)上
- 別名
 - NX (No-eXecute)
 - W^X

看一下權限

- \$ cat /proc/(pid)/maps

```
vmvm@ubuntu:~$ cat /proc/3680/maps
08048000-08049000 r-xp 00000000 08:01 1321219 /home/vmvm/test/a.out
08049000-0804a000 r--p 00000000 08:01 1321219 /home/vmvm/test/a.out
0804a000-0804b000 rw-p 00001000 08:01 1321219 /home/vmvm/test/a.out
b7e17000-b7e18000 rw-p 00000000 00:00 0
b7e18000-b7fc0000 r-xp 00000000 08:01 525301 /lib/i386-linux-gnu/libc-2.19.so
b7fc0000-b7fc2000 r--p 001a8000 08:01 525301 /lib/i386-linux-gnu/libc-2.19.so
b7fc2000-b7fc3000 rw-p 001aa000 08:01 525301 /lib/i386-linux-gnu/libc-2.19.so
b7fc3000-b7fc6000 rw-p 00000000 00:00 0
b7fdb000-b7fdd000 rw-p 00000000 00:00 0
b7fdd000-b7fde000 r-xp 00000000 00:00 0 [vdso]
b7fde000-b7ffe000 r-xp 00000000 08:01 525313 /lib/i386-linux-gnu/ld-2.19.so
b7ffe000-b7fff000 r--p 0001f000 08:01 525313 /lib/i386-linux-gnu/ld-2.19.so
b7fff000-b8000000 rw-p 00020000 08:01 525313 /lib/i386-linux-gnu/ld-2.19.so
bffd000-c0000000 rw-p 00000000 00:00 0 [stack]
```

關掉DEP

- \$ gcc -z execstack shellcode.c

```
vmvm@ubuntu:~/test$ gcc -z execstack shellcode.c
vmvm@ubuntu:~/test$ ./a.out
$
```

- \$ execstack

```
vmvm@ubuntu:~/test$ execstack -c a.out
vmvm@ubuntu:~/test$ ./a.out
Segmentation fault (core dumped)
vmvm@ubuntu:~/test$ execstack -s a.out
vmvm@ubuntu:~/test$ ./a.out
$
```

Code reuse attack

- ret2text
 - return到已有的code
- ret2libc
 - 程式通常都不會有 `system("/bin/sh")` 給攻擊者用
 - 不過大多數程式都會dynamic link libc
 - 可以去用libc中的 `system`，即使程式本身沒有用`system`
- Return Orient Programing (ROP)
 - 當沒有libc可用時的大絕招

Address Space Layout Randomization

- 位址空間配置隨機載入
- => 程式在記憶體中放的位置是隨機的
- 2005年開始用在Linux (kernel 2.6.12)上
- 可以random
 - stack
 - heap
 - library
 - code (需要PIE)

開關ASLR

- `/proc/sys/kernel/randomize_va_space`
- 0: disable
- 1: randomize stack, vdso, libraries
- 2: 1 + heap

有了ASLR之後

- \$ cat /proc/(pid)/maps

vmvm@ubuntu:~\$ cat /proc/5219/maps							
08048000-08049000	r-xp	00000000	08:01	1321219		/home/v	
08049000-0804a000	r--p	00000000	08:01	1321219		/home/v	
0804a000-0804b000	rw-p	00001000	08:01	1321219		/home/v	
08301000-09007000	rw-p	00000000	00:00	0		[heap]	
b7525000-b754a000	rw-p	00000000	00:00	0			
b7526000-b754b000	r-xp	00000000	08:01	525301		/lib/i3	
b76ce000-b76f3000	r--p	001a8000	08:01	525301		/lib/i3	
b76d0000-b76f5000	rw-p	001aa000	08:01	525301		/lib/i3	
b76d1000-b76f6000	rw-p	00000000	00:00	0			
b76e9000-b770e000	rw-p	00000000	00:00	0			
b76eb000-b7710000	r-xp	00000000	00:00	0		[vdso]	
b76ec000-b7711000	r-xp	00000000	08:01	525313		/lib/i3	
b770c000-b7731000	r--p	0001f000	08:01	525313		/lib/i3	
b770d000-b7732000	rw-p	00020000	08:01	525313		/lib/i3	
bfd59000-bfcdb000	rw-p	00000000	00:00	0		[stack]	

Position-independent executables

- 讓code也可以ASLR
- \$ gcc -fpie -pie test.c

```
b759b000- b7523000-b7524000 rw-p 00000000 00:00 0
b759c000- b7524000-b76cc000 r-xp 00000000 08:01 525301 /lib/i386-linux-gnu/libc-2.19.so
b7744000- b76cc000-b76ce000 r--p 001a8000 08:01 525301 /lib/i386-linux-gnu/libc-2.19.so
b7746000- b76ce000-b76cf000 rw-p 001aa000 08:01 525301 /lib/i386-linux-gnu/libc-2.19.so
b7747000- b76cf000-b76d2000 rw-p 00000000 00:00 0
b775f000- b76e7000-b76e9000 rw-p 00000000 00:00 0
b7761000- b76e9000-b76ea000 r-xp 00000000 00:00 0 [vdso]
b7762000- b76ea000-b770a000 r-xp 00000000 08:01 525313 /lib/i386-linux-gnu/ld-2.19.so
b7782000- b770a000-b770b000 r--p 0001f000 08:01 525313 /lib/i386-linux-gnu/ld-2.19.so
b7783000- b770b000-b770c000 rw-p 00020000 08:01 525313 /lib/i386-linux-gnu/ld-2.19.so
b7784000- b770c000-b770d000 r-xp 00000000 08:01 1321219 /home/vmvm/test/a.out
b7785000- b770d000-b770e000 r--p 00000000 08:01 1321219 /home/vmvm/test/a.out
b7786000- b770e000-b770f000 rw-p 00001000 08:01 1321219 /home/vmvm/test/a.out
b956c000- b92f0000-b9311000 rw-p 00000000 00:00 0 [heap]
bffa1000- bfa1e000-bfa3f000 rw-p 00000000 00:00 0 [stack]
```

Conquer ASLR

- Information leak
 - 先leak出某些訊息後再算offset
- fork
 - fork不會再次進行ASLR
 - 可以brute-force

Ret2libc

- 你需要
 - 被攻擊的有弱點程式
 - 該程式所用的libc
- 步驟
 - Leak GOT
 - 計算offset
 - exploit

Leak GOT

- Global Offset Tables (GOT)
 - 一個寫著library function實際位置的表格
- 如果可以leak當中的訊息的話，就可以算出其他function的offset
- 用 `objdump -R` 可以看程式中有哪些GOT及其位置

```
vmvm@ubuntu:~/test$ objdump -R a.out
a.out:          file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET      TYPE          VALUE
08049ffc    R_386_GLOB_DAT    __gmon_start__
0804a038    R_386_COPY        stdin
0804a00c    R_386_JUMP_SLOT   printf
0804a010    R_386_JUMP_SLOT   __isoc99_fscanf
0804a014    R_386_JUMP_SLOT   free
0804a018    R_386_JUMP_SLOT   fgets
0804a01c    R_386_JUMP_SLOT   sleep
0804a020    R_386_JUMP_SLOT   malloc
0804a024    R_386_JUMP_SLOT   __gmon_start__
0804a028    R_386_JUMP_SLOT   __libc_start_main
0804a02c    R_386_JUMP_SLOT   fopen
```

計算offset

- 用 `objdump -d` 可以看libc中各function的位置
- 先算出libc的base address，接著可以跳到任意function

```
vmvm@ubuntu:~$ objdump -d /lib/i386-linux-gnu/libc-2.19.so | grep "malloc>:"  
000766b0 <__libc_malloc>:  
vmvm@ubuntu:~$ objdump -d /lib/i386-linux-gnu/libc-2.19.so | grep "system>:"  
00040190 <__libc_system>:
```

Exploit

abcd0000	variable 1
abcd0004	saved ebp
abcd0008	return address
abcd000c	arg1
abcd0010	arg2
abcd0014

abcd0000	AAAA
abcd0004	AAAA
abcd0008	address of "system"
abcd000c	ret addr after "system"
abcd0010	address of "/bin/sh"
abcd0014

Practice

- nc 140.113.194.89 5566
- <http://140.113.194.89:8000>

Question