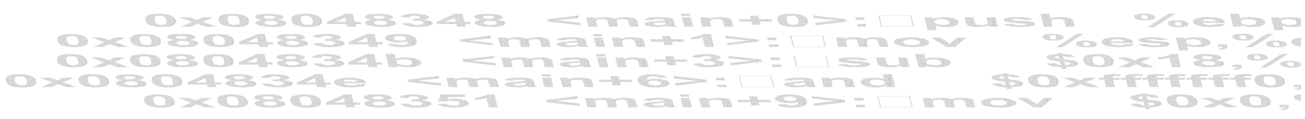
0x08048348 <main+0>: push %ebp
0x08048349 <main+1>: mov %esp,%e
0x0804834b <main+3>: sub $0x18,%e
0x0804834e <main+6>: and $0xfffffff0,
0x08048351 <main+9>: mov $0x0,

Writing Shellcodes in Linux
----

by Amitesh Singh

Introduction

Shellcoding is a skill to write your machine codes in hexadecimal form. Many people lack it. From the view of security it is very important, as hackers use it to exploit vulnerable applications. In this article, we are working on Linux using the IA-32(x86) architecture. Basic knowledge of C, ASM(AT&T style) and working with debuggers (gdb & objdump)is required.

Lets Start

Consider a simple program

```
#####exit.c###########
         int main()
           {
             exit(0);
           }
##########EOP########

$ make exit && ./exit
cc     exit.c   -o exit
$ echo $?
0
$
```

Syntax of exit() is "void exit(int status)".The exit() function causes normal program termination and the the value of status return to the parent. Here in the above program status return to main is 0.you can check it by altering the code

```
int main()
{ exit(1);
}

$ make exit && ./exit
cc     exit.c   -o exit
$ echo $?
1
$
```

Writing ASM code for above program

Since the operating system features are accessed through System calls. These are invoked by setting the registers in special way and issuing the instruction int $0x80.

For function<6 arguments

EAX<<=========function(EBX,ECX,EDX,EDI,ESI)
The syscall number of function stores in EAX and arguments store in EBX,ECX & so on.
You can view Linux SYSCALLS in file ‘unistd.h’. For our function exit(0),the syscall no. of exit is 1 hence EAX is to be loaded with 1 and EBX with the exit

status which is 0 here.

```
######exit.s######
.globl main
main:

     movl $0,%ebx
     movl $1,%eax
     int $0x80
#######EOP###########
```

Now compile it and disassemble it using 'gdb' or 'objdump'

```
$as -o exit.o exit.s && ld -o exit exit.o
$objdump -d a.out

08048314 <main>:
 8048314:   bb 00 00 00 00          mov    $0x0,%ebx
 8048319:   b8 01 00 00 00          mov    $0x1,%eax
 804831e:   cd 80                   int    $0x80
```

Too many null bytes there. Any null byte in the shellcode will be considered the end of string, hence only the first byte of the shellcode to be copied into the buffer. To get the shellcode copy into the buffer properly, all of the null bytes must be eliminated.

Using XOR we can eliminate null bytes.

A	**B**	**A XOR B**
0	0	0
0	1	1
1	0	1
1	1	0

hence XOR a,a gives a=0
movl $0,%ebx =====>>> xorl %ebx,%ebx
Now for second line
"movl $1,%eax"

Let first understand EAX in detail

As here we are only working with 1 byte so there is no need to use %eax. You could just use %ah or %al. %ax is the least-significant half of the %eax and further %ax is divided into 2 parts %ah(most significant byte) & %al(least significant byte).

```
  ______EAX________
 |                 |
 __________________
|   |    |  AH | AL |
|___|____|____|____|

         |____AX___|
```

Loading any value into %eax will wipe out whatever value in %ah & %al (and also %ax).Similarly loading any value into either %ah or %al(also %ax) will corrupt whatever value that was formerly in %eax. Hence it's ok to use a register for either a byte or a word but never both at the same time.

movl $1,%eax =====>> movb $1,%al

Again writing ASM code

```
.globl main
```

```
main:

  xorl %ebx,%ebx
  movl %1,%al
  int $0x80
```

Now disassembling our modified program

```
08048314 <main>:
 8048314:       31 db                   xor    %ebx,%ebx
 8048316:       b0 01                   mov    $0x1,%al
 8048318:       cd 80                   int    $0x80
 804831a:       90                      nop
 804831b:       90                      nop
```

Another way to eliminate null bytes

Well we can write "movl $1,%eax" in different way by eliminating the null bytes.

```
movl %1,%eax =========>> movl %ebx,%eax
                         incl %eax
```

Disasssembling.......

```
08048314 <main>:
 8048314:   31 db                   xor    %ebx,%ebx
 8048316:   89 d8                   mov    %ebx,%eax
 8048318:   40                      inc    %eax
 8048319:   cd 80                   int    $0x80
 804831b:   90                      nop
```

but the size of the shellcode is larger than previous one so later one is better.
Finally it's time to write the shellcode
"\x31\xdb\xb\x01\xcd\x80"

```
######exit.s############
char shellcode[]="\x31\xdb\xb\x01\xcd\x80";
int main(int argc,char *argv[])
{  int *ret;

   *((char**)(&ret+2))=shellcode;

}
##################EOP#############
```

now compiling it

```
$ make exit && ./exit
cc     exit.c   -o exit
$ echo $?
0
$
```

Great it works.....

Author: ***Amitesh Singh***
singh.amitesh@gmail.com
http://amitesh.info