1. **[整数溢出](#_第三章_整数溢出_&)**

# 第三章 整数溢出

1.原理

每种数据类型都有各自的大小范围

|  |  |  |
| --- | --- | --- |
| 类型 | 字节 | 范围 |
| short int | 2byte(word) | 0~32767(0~0x7fff)  -32768~-1(0x8000~0xffff) |
| unsigned short int | 2byte(word) | 0~65535(0~0xffff) |
| int | 4byte(dword) | 0~2147483647(0~0x7fffffff) -2147483648~-1(0x80000000~0xffffffff) |
| unsigned int | 4byte(dword) | 0~4294967295(0~0xffffffff) |
| long int | 8byte(qword) | 正: 0~0x7fffffffffffffff  负: 0x8000000000000000~0xffffffffffffffff |
| unsigned long int | 8byte(qword) | 0~0xffffffffffffffff |

add 0x7fff, 1 == 0x8000， 这种上界溢出对无符号整型就没有影响， 但是在有符号短整型中 ， 0x7fff 表示的是 32767， 但是 0x8000 表示的是 -32768， 用数学表达式来表示就是在有符号 短整型中 32767+1 == -32768

比如上面的有符号型加法的汇编代码是 add eax, 1， 因为 eax=0xffff， 所以 add eax, 1 == 0x10000， 但是无符号的汇编代码是对内存进行加法运算 add word ptr [rbp - 0x1a], 1 == 0x0000。

再从数字层面看看这种溢出的结果， 在有符号短整型中， 0xffff==-1， -1 + 1 == 0， 从有符号看 这种计算没问题。

但是在无符号短整型中， 0xffff == 65535, 65535 + 1 == 0。

2.例子

#include<stddef.h>

int main(void)

{

int len;

int data\_len;

int header\_len;

char \*buf;

header\_len = 0x10;

scanf("%uld", &data\_len);

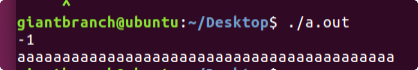
len = data\_len+header\_len

buf = malloc(len);

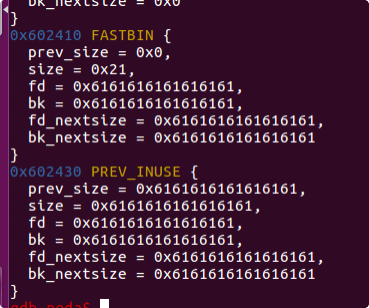
read(0, buf, data\_len);

return 0;

}



只申请 0x20 大小 ( 0xf+0x10+0x1) 的堆(实测0x21,可能是对齐之类的原因)， 但是却能输入 0xffffffff 长度的数据， 从整型溢出到堆溢出



$ cat test3.c

int main(void)

{

int len, l;

char buf[11];

scanf("%d", &len);

if (len < 10) {

l = read(0, buf, len);

\*(buf+l) = 0;

puts(buf);

} else

printf("Please len < 10");

}

$ gcc test3.c

$ ./a.out

-1

aaaaaaaaaaaa

aaaaaaaaaaaa

从表面上看， 我们对变量 len 进行了限制， 但是仔细思考可以发现， len 是有符号整型， 所以 len 的长 度可以为负数， 但是在 read 函数中， 第三个参数的类型是 size\_t， 该类型相当于 unsigned long int， 属于无符号长整型

[回到目录](#_top)