

3.4 格式化字符串漏洞

```
#include <stdio.h>

int main(void)
{
    printf("%s", "Hello world!");
    return 1;
}
```

3.4 格式化字符串漏洞

□ 格式化字符串

- 在printf()系列函数中按照一定的格式对数据进行输出，并输出到标准输出，也可以输出到文件句柄、字符串中等，对应的函数有fprintf,sprintf,snprintf,vprintf,vfprintf,vsprintf,vsprintf等。

```
printf("%s", "Hello world!"); printf("Hello world!");
```



格式化字符串 (Format string)

3.4 格式化字符串漏洞

□ include <stdio.h>

- int printf(const char *format, ...);
- int fprintf(FILE *stream, const char *format, ...);
- int dprintf(int fd, const char *format, ...);
- int sprintf(char *str, const char *format, ...);
- int snprintf(char *str, size_t size, const char *format, ...);

printf(buf);?

3.4 格式化字符串漏洞

- ❑ `#include <stdarg.h>`
- `int vprintf(const char *restrict format, va_list arg);`
- `int vfprintf(FILE *restrict fp, const char *restrict format, va_list arg);`
- `int vdprintf(int fd, const char *restrict format, va_list arg);`
- `int vsprintf(char *restrict buf, const char *restrict format, va_list arg);`
- `int vsnprintf(char *restrict buf, size_t n, const char *restrict format, va_list arg);`

3.4 格式化字符串漏洞

```
1.  /*模仿snprintf()函数*/
2.  int my_snprintf(char *s, int size, const char *fmt, ...)
3.  {
4.      va_list ap;
5.      int n=0;
6.      va_start(ap, fmt);          /*获得可变参数列表*/
7.      n=vsnprintf(s, size, fmt, ap); /*写入字符串*/
8.      va_end(ap);                /*释放资源*/
9.      return n;                  /*返回写入的字符个数*/
10. }
11. int main() {
12.     char str[256];
13.     my_snprintf(str, sizeof(str), "%s:%d,%d,%d,%d", "my_snprintf", 5, 6, 7, 8);
14.     printf("%s\n", str);
15.     return 0;
16. }
```

```
int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int dprintf(int fd, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);
```



```
int vprintf(const char *restrict format, va_list arg);
int vfprintf(FILE *restrict fp, const char *restrict format, va_list arg);
int vdprintf(int fd, const char *restrict format, va_list arg);
int vsprintf(char *restrict buf, const char *restrict format, va_list arg);
int vsnprintf(char *restrict buf, size_t n, const char *restrict format, va_list arg);
```

3.4 格式化字符串漏洞

```
int main(int argc, char** argv)
{
    int iNuma, iLen, iAuthenticated=0;
    int iThreshold=12345678;
    iLen=argc;
    scanf("%d",&iNuma);
    iAuthenticated=Hashcmp(iNuma,iThreshold);
    printf("%x,%x,%x,%x",iNuma,iLen,iAuthenticated);
    if (iAuthenticated)
        printf("The login is OK");
    else
        printf("The login is FAIL");
    return iLen;
}
```

```
int Hashcmp(int a, int b)
{
    if (a==b) return 1;
    else return 0;
}
```

```
D:\course\scode>v032
12345678
bc614e,1,1,bc614e
The login is OK
```

```
D:\course\scode>v032
64
40,1,0,40
The login is FAIL
```

3.4 格式化字符串漏洞

```
int iNuma;  
int iLen;  
int iAuthenticated=0;  
int iThreshold=12345678
```

```
iLen=argc;
```

```
scanf("%d",&iNuma);
```

```
iAuthenticated=Hashcmp(iNuma,iThreshold);
```

```
printf("%x,%x,%x,%x",iNuma,iLen,iAuthenticated);
```

```
if (iAuthenticated)
```

```
{
```

```
    printf("The login is OK\r\n");
```

```
}
```

```
D:\course\scode>v032  
64  
40,1,0,40
```

最后一个
0x40是什么

3.4 格式化字符串漏洞

- ❑ printf函数
- 参数：变长。
- 调用函数调用printf时，把实参从右往左压入栈中。
- printf会自动从栈顶中寻找形参的数据并输出。
- 当实参个数少于格式化串要求的变量个数(%)时，从而出现信息泄露。

```
printf("%x,%x,%x,%x\n", \
iNuma,iLen,iAuthenticated);
```

```
mov     DWORD PTR [esp+0x1c], 0x0
mov     DWORD PTR [esp+0x18], 0xbc614e

mov     eax, DWORD PTR [ebp+0x8]
mov     DWORD PTR [esp+0x14], eax
lea     eax, [esp+0x10]
mov     DWORD PTR [esp+0x4], eax
mov     DWORD PTR [esp], 0x403024
call    401c10 <_scanf>
mov     eax, DWORD PTR [esp+0x10]
mov     edx, DWORD PTR [esp+0x18]
mov     DWORD PTR [esp+0x4], edx
mov     DWORD PTR [esp], eax
call    4013ce <Hashcmp>
mov     DWORD PTR [esp+0x1c], eax
mov     eax, DWORD PTR [esp+0x10]
mov     edx, DWORD PTR [esp+0x1c]
mov     DWORD PTR [esp+0xc], edx
mov     edx, DWORD PTR [esp+0x14]
mov     DWORD PTR [esp+0x8], edx
mov     DWORD PTR [esp+0x4], eax
mov     DWORD PTR [esp], 0x403027
call    401c18 <printf>
```


3.4 格式化字符串漏洞

```
int iNuma;  
int iLen;  
int iAuthenticated=0;  
int iThreshold=12345678;  
  
iLen=argc;  
  
scanf("%d",&iNuma);  
iAuthenticated=Hashcmp(iNuma,iThreshold);  
printf("%x,%x,%x,%x",iNuma,iLen,iAuthenticated);  
if (iAuthenticated)  
{  
    printf("The login is OK\r\n");  
}
```

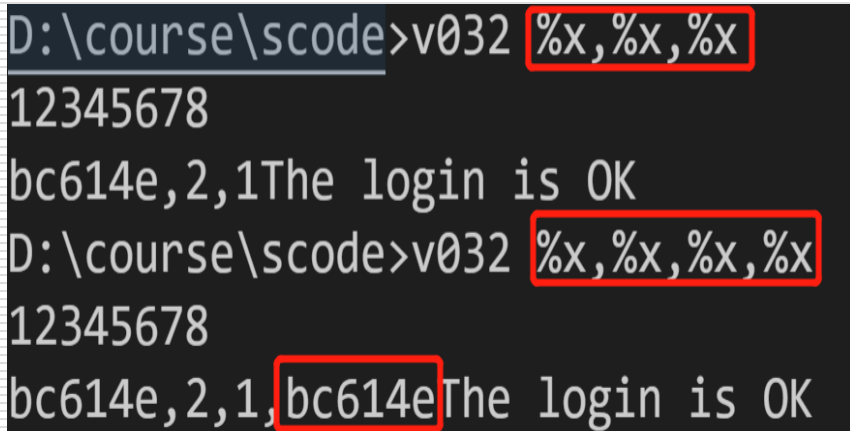
```
mov     DWORD PTR [esp+0x1c], 0x0  
mov     DWORD PTR [esp+0x18], 0xbc614e  
  
mov     eax, DWORD PTR [ebp+0x8]  
mov     DWORD PTR [esp+0x14], eax  
lea     eax, [esp+0x10]  
mov     DWORD PTR [esp+0x4], eax  
mov     DWORD PTR [esp], 0x403024  
call    401c10 <_scanf>  
mov     eax, DWORD PTR [esp+0x10]  
mov     edx, DWORD PTR [esp+0x18]  
mov     DWORD PTR [esp+0x4], edx  
mov     DWORD PTR [esp], eax  
call    4013ce <Hashcmp>  
mov     DWORD PTR [esp+0x1c], eax  
mov     eax, DWORD PTR [esp+0x10]  
mov     edx, DWORD PTR [esp+0x1c]  
mov     DWORD PTR [esp+0xc], edx  
mov     edx, DWORD PTR [esp+0x14]  
mov     DWORD PTR [esp+0x8], edx  
mov     DWORD PTR [esp+0x4], eax  
mov     DWORD PTR [esp], 0x403027  
call    401c18 <printf>
```

%x%x...	esp
iNuma	esp+4
iLen	esp+c
iAuth...	
?	esp+10
iLen	
iThreshold	esp+18
iAuth...	

3.4 格式化字符串漏洞

- ❑ 格式化字符串漏洞是指其格式化字符串来自不可信的外部输入，会引发信息泄露、变量修改或者控制劫持的缺陷。
- ❑ 1) 参数个数不固定造成栈中的数据被越界访问泄露

```
int main(int argc, char** argv)
{int iNuma, iLen, iAuthenticated=0;
 int iThreshold=12345678;
 iLen=argc;
 scanf("%d",&iNuma);
 iAuthenticated=Hashcmp(iNuma,iThreshold);
 printf(argv[1], iNuma,iLen,iAuthenticated);
 if (iAuthenticated)
     printf("The login is OK");
 else
     printf("The login is FAIL");
 return iLen;}
```



```
D:\course\scode>v032 %x,%x,%x
12345678
bc614e,2,1The login is OK
D:\course\scode>v032 %x,%x,%x,%x
12345678
bc614e,2,1,bc614eThe login is OK
```

3.4 格式化字符串漏洞

□ 2) 利用**%n**格式符写入跳转地址，从而改写栈中地址的值。

```
printf(argv[1],iNuma,iLen,&iAuthenticated);  
printf("\n%d",iAuthenticated);  
//printf(argv[1],iNuma,iLen,iAuthenticated);  
//printf("%x,%x,%x,%x\n",iNuma,iLen,iAuthent  
icated);  
if (iAuthenticated)  
    printf("\nThe login is OK");  
else  
    printf("\nThe login is FAIL");
```

```
D:\course\score>v032 %x,%x:%n  
64  
40,2:  
0  
The login is FAIL
```

```
jmfu@HW-WORK:/mnt/d/course/score$ ./v032 %x,%x:%n  
64  
40,2:  
5  
The login is OK
```

Why?

3.4 格式化字符串漏洞

There are **724 CVE Records** that match your search.

Name	Description
CVE-2020-7241	The WP Database Backup plugin through 5.5 for WordPress stores downloads by default locally in the directory wp-content/uploads/db-backup/. This might allow attackers to read ZIP archives by guessing random ID numbers, guessing date strings with a 2020_{0..1}{0..2}_{0..3}{0..9} format, guessing UNIX timestamps, and making HTTPS requests with the complete guessed URL.
CVE-2020-5215	In TensorFlow before 1.15.2 and 2.0.1, converting a string (from Python) to a tf.float16 value results in a segmentation fault in eager mode as the format checks for this use case are only in the graph mode. This issue can lead to denial of service in inference/training where a malicious attacker can send a data point which contains a string instead of a tf.float16 value. Similar effects can be obtained by manipulating saved models and checkpoints whereby replacing a scalar tf.float16 value with a scalar string will trigger this issue due to automatic conversions. This can be easily reproduced by tf.constant("hello", tf.float16), if eager execution is enabled. This issue is patched in TensorFlow 1.15.1 and 2.0.1 with this vulnerability patched. TensorFlow 2.1.0 was released after we fixed the issue, thus it is not affected. Users are encouraged to switch to TensorFlow 1.15.1, 2.0.1 or 2.1.0.
CVE-2020-5204	In ftpd before 2.11, there is a buffer overflow vulnerability in handle_PORT in ftpcmd.c that is caused by a buffer that is 16 bytes large being filled via sprintf() with user input based on the format specifier string %d.%d.%d.%d. The 16 byte size is correct for valid IPv4 addresses (len('255.255.255') == 16), but the format specifier %d allows more than 3 digits. This has been fixed in version 2.11
CVE-2020-35869	An issue was discovered in the rusqlite crate before 0.23.0 for Rust. Memory safety can be violated because rusqlite::trace::log mishandles format strings.
CVE-2020-3118	A vulnerability in the Cisco Discovery Protocol implementation for Cisco IOS XR Software could allow an unauthenticated, adjacent attacker to execute arbitrary code or cause a reload on an affected device. The vulnerability is due to improper validation of string input from certain fields in Cisco Discovery Protocol messages. An attacker could exploit this vulnerability by sending a malicious Cisco Discovery Protocol packet to an affected device. A successful exploit could allow the attacker to cause a stack overflow, which could allow the attacker to execute arbitrary code with administrative privileges on an affected device. Cisco Discovery Protocol is a Layer 2 protocol. To exploit this vulnerability, an attacker must be in the same broadcast domain as the affected device (Layer 2 adjacent).
CVE-2020-29018	A format string vulnerability in FortiWeb 6.3.0 through 6.3.5 may allow an authenticated, remote attacker to read the content of memory and retrieve sensitive data via the redir parameter.

3.4 格式化字符串漏洞

标识	提交时间	漏洞等级	漏洞名称	漏洞状态	人气
CVE-2020-15634	2020-08-20	—	NETGEAR R6700 格式化字符串错误漏洞	    	15
CVE-2020-13160	2020-06-09	—	AnyDesk格式化字符串错误漏洞	    	79
CVE-2020-1992	2020-04-08	—	Palo Alto Networks PAN-OS 格式化字符串错误漏洞	    	11
CVE-2020-1979	2020-03-11	—	Palo Alto Networks PAN-OS 格式化字符串错误漏洞	    	12
CVE-2019-5143	2020-02-25	—	Moxa AWK-3131A 格式化字符串错误漏洞	    	21
CVE-2014-6262	2020-02-12	—	Zenoss Core RRDtool 格式化字符串错误漏洞	    	25
CVE-2020-3118	2020-02-05	—	Cisco IOS XR 格式化字符串错误漏洞	    	146
CVE-2018-10388	2019-12-23	—	Open TFTP Server SP 格式化字符串错误漏洞	    	17
CVE-2018-10389	2019-12-23	—	Open TFTP Server MT 格式化字符串错误漏洞	    	12
CVE-2012-0824	2019-11-19	—	gnusound 格式化字符串错误漏洞	    	27

1 2 3 4 5 6 ... 28 29 共 289 条

<http://cve.scap.org.cn//vulns?view=global&keyword=格式化>

3.4 格式化字符串漏洞

□ CVE-2020-27853

- 发布时间：2020-10-27
- 危害等级：超危
- 漏洞详情：

Wire是个人开发者的一款聊天软件。Wire 2020-10-16之前版本存在安全漏洞，该漏洞攻击者造成**拒绝服务**（应用程序崩溃）或可能通过**格式字符串执行任意代码**。

- 影响版本：

Wire AVS (Audio, Video, and Signaling) 5.3版本至6.x系列6.4之前版本, Wire Secure Messenger application Android 3.49.918版本, Wire Secure Messenger application iOS 3.61版本。

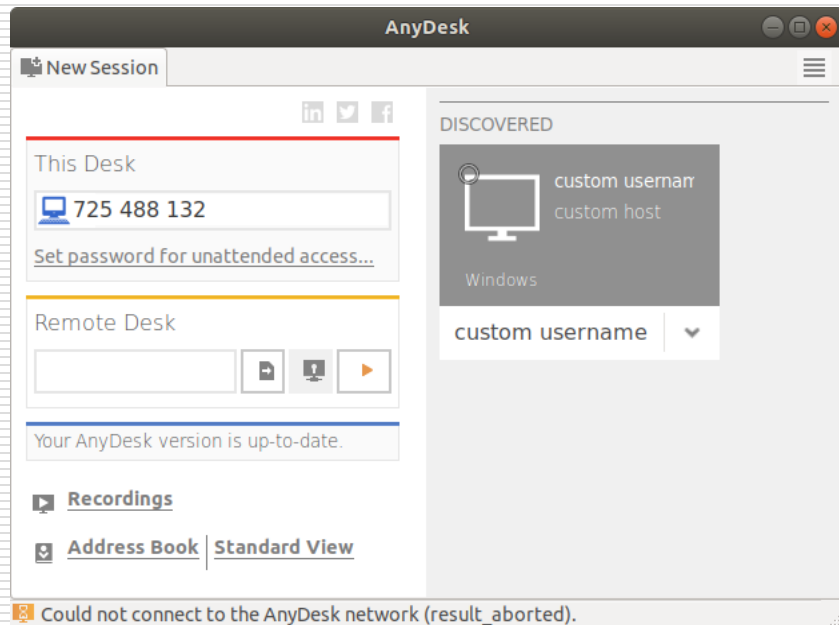
3.4 格式化字符串漏洞

❑ CVE-2020-13160

- 发布时间：2020-02-21
- 危害等级：超危
- 漏洞详情： AnyDesk是一款免费的远程连接、远程桌面控制软件，拥有先进的视频压缩技术DeskRT，AnyDesk 可用于多种操作系统，包括Windows，Linux，Android 和 iOS。远程发送单个UDP数据包，会造成拒绝服务或远程代码执行。
- 影响版本：AnyDesk Linux 版本 5.5.2 。

3.4 格式化字符串漏洞

□ AnyDesk Linux 版本 5.5.2



```
p =  
gen_discover_pack  
et(4919, 1,  
'custom host',  
'custom username',  
'ad', 'main')  
s =  
socket.socket(sock  
et.AF_INET,  
socket.SOCK_DGR  
AM)
```


3.4 格式化字符串漏洞

```
p = gen_discover_packet(4919, 1,  
'\x85\xfeHOSTNAME %165$p',  
'\x85\xfe%18472249x%93$ln', 'ad',  
'main')
```



Thread 1 "anydesk" received signal
SIGSEGV, Segmentation fault.



0x7f4918657932

<_IO_vfprintf_internal+9634>:
mov DWORD PTR [rax],r13d



```
gdb-peda$ bt  
#0  0x00007f4918657932 in _IO_vfprintf_internal (s=s@entry=0x7ffe12cbb5b0,  
      format=format@entry=0x7ffe12cbb800 "Failed to set text from markup due to error p  
#1  0x00007f4918682910 in _IO_vsnprintf (  
      string=0x7ffe12cbb800 "Failed to set text from markup due to error parsing markup  
      format=0x7ffe12cbb800 "Failed to set text from markup due to error parsing markup  
#2  0x000000000008ab34b in ?? ()  
#3  0x000000000008aba98 in ?? ()  
#4  0x00000000000434395 in ?? ()  
#5  0x00007f491d0b11cd in g_logv () from /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0  
#6  0x00007f491d0b133f in g_log () from /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0
```

3.4 格式化字符串漏洞

- ❑ 构造特制的格式化字符串，触发格式化字符串漏洞
 - 内存损害
 - 信息泄露
 - 越权写
 - 远程代码执行
- ❑ 检测和防御
 - 审核格式化字符串的字符
 - 禁用“%n”

3.5 整数溢出漏洞

□ 常见的整数类型

- unsigned char
- char
- unsigned short int
- short int
- unsigned int
- int
- unsigned long int
- long int

```
Size of char,short,int,long
is: 1, 2, 4, 4
uc=128, c=-128
Transfer to large domain
ui=4294966996;c=-128
```

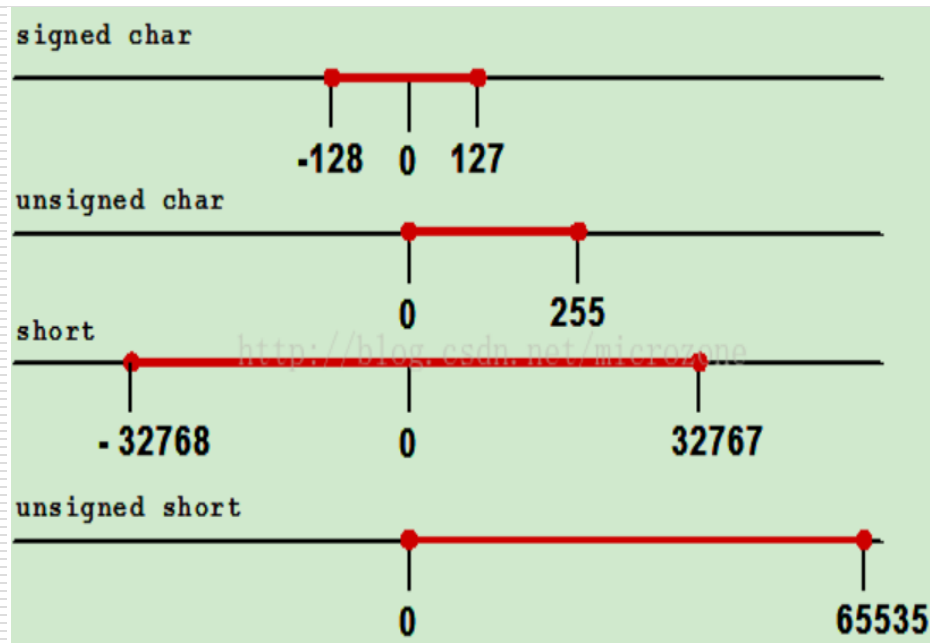
```
unsigned short int usi; short int si;
unsigned int ui; int ii;
unsigned char uc; char c;
printf("The size of char,short,int,long\n\tis:
%2d,%5d,%4d,%4d",sizeof(char), sizeof(short int),
sizeof(int),sizeof(long int));
uc=127; uc++; c=127;c++;
printf("\nuc=%d, c=%d",uc,c);
ii=-300; ui=300;
if (ii>ui) printf("\nTransfer to large domain");
if (ii>(int)ui) printf("\ncomparison with int");
ui=ii;
uc=128; c=uc;
printf("\nui=%u;c=%d",ui,c);
```

3.5 整数溢出漏洞

- 有符号整数-高位为符号位
- 无符号整数-高位为有效位

char	1Byte
short int	2Byte
int	4Byte
long int	4Byte

- uc++没有溢出 (255)
- c++溢出 (127)



`unsigned char uc; char c; uc=127; uc++; c=127; c++;`

3.5 整数溢出漏洞

- 1) 上溢 (Overflow) : 整数运算结果超过了该类型表示的上限, 结果变成一个很小的数 (无符号数) 或负数 (有符号数)。

例: `unsigned short usi = 65535;`

`usi = usi+5`的计算结果为4。



分析:

$$\begin{aligned}(65535)_{10} + (5)_{10} &= (1111\ 1111\ 1111\ 1111)_2 + (101)_2 \\ &= (1\ 0000\ 0000\ 0000\ 0100)_2\end{aligned}$$

`unsigned short` 长度为16位, 最高位的1被丢弃, 只保留低16位, 即 $(000000000000000100)_2 = (4)_{10}$

3.5 整数溢出漏洞

- 2) 下溢 (Underflow) : 整数运算结果低于该类型能表示的下限, 结果变成一个很大的整数。

例: `char c = -128;`

`c = c-1;`

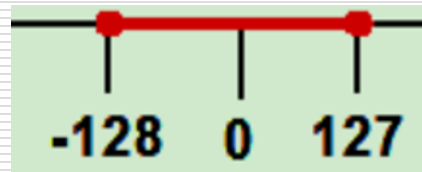
实际结果为: `c = 127`

分析:

$$(-128)_{10} = (-1000\ 0000)_2 = (0111\ 1111 + 1)_2 = (1000\ 0000)_2$$

$$(-1)_{10} = (-0000\ 0001)_2 = (1111\ 1110 + 1)_2 = (1111\ 1111)_2$$

$$(-128)_{10} + (-1)_{10} = (0111\ 1111)_2 = (127)_{10}$$



3.5 整数溢出漏洞

- 3) 宽带转换 (Width Conversion) : 把长类型变量赋值给短类型变量时, 出现截断。

例: `int ii = 65537;`
 `short si = ii;`

实际结果为: `si = 1`

分析:

$(65537)_{10} = (1 \text{ } 0000 \text{ } 0000 \text{ } 0000 \text{ } 0001)_2$

截断后: $si = (0000 \text{ } 0000 \text{ } 0000 \text{ } 0001)_2 = 1$

3.5 整数溢出漏洞

- 4) 符号转换(signed/unsigned conversion): 当无符号整数和有符号整数进行比较或运算时, 注意符号变化。

例: `int ii = -300;`
 `unsigned int ui = 300;`

预期比较结果: `ii < ui`

实际比较结果: `ii > ui`

分析:

-300转化成无符号整数后等于4294966996, 远远大于300.

`ii=-300; ui=300;`

`if (ii>ui) printf("\nTransfer to large domain");`

Transfer to large domain

3.5 整数溢出漏洞

□ 阅读代码

```
char* mystrcat(char *s1, unsigned int len1, char *s2, unsigned int len2)
{
    if(len1 + len2 + 1 > 4096 )
    {
        return FALSE;
    }
    char *pBuf = new char[len1 + len2 + 1];
    memcpy(pBuf, s1, len1);
    memcpy(pBuf + len1, s2, len2);
    return pBuf ;
}
```

```
char* allocBuf(unsigned int cbSize)
{
    if( cbSize > 4096 )
    {
        return FALSE;
    }
    char *pBuf = new char[cbSize-1];
    memset(pBuf, 0x90, cbSize-1);
    return pBuf;
}
```

□ 安全缺陷？

3.5 整数溢出漏洞

There are **2680 CVE Records** that match your search.

Name	Description
CVE-2021-27219	An issue was discovered in GNOME GLib before 2.66.6 and 2.67.x before 2.67.3. The function <code>g_bytes_new</code> has an integer overflow on 64-bit platforms due to an implicit cast from 64 bits to 32 bits . The overflow could potentially lead to memory corruption.
CVE-2021-26825	An integer overflow issue exists in Godot Engine up to v3.2 that can be triggered when loading specially crafted TGA image files. The vulnerability exists in <code>ImageLoaderTGA::load_image()</code> function at line: <code>const size_t buffer_size = (tga_header.image_width * tga_header.image_height) * pixel_size</code> ; The bug leads to Dynamic stack buffer overflow. Depending on the context of the application, attack vector can be local or remote, and can lead to code execution and/or system crash.
CVE-2021-21036	Acrobat Reader DC versions versions 2020.013.20074 (and earlier), 2020.001.30018 (and earlier) and 2017.011.30188 (and earlier) are affected by an Integer Overflow vulnerability. An unauthenticated attacker could leverage this vulnerability to achieve arbitrary code execution in the context of the current user. Exploitation of this issue requires user interaction in that a victim must open a malicious file.
CVE-2021-1059	NVIDIA vGPU manager contains a vulnerability in the vGPU plugin, in which an input index is not validated , which may lead to integer overflow, which in turn may cause tampering of data, information disclosure, or denial of service. This affects vGPU version 8.x (prior to 8.6) and version 11.0 (prior to 11.3).
CVE-2021-0355	In <code>kisd</code> , there is a possible out of bounds write due to an integer overflow. This could lead to local escalation of privilege with System execution privileges needed. User interaction is not needed for exploitation. Product: Android; Versions: Android-11; Patch ID: ALPS05425581.
CVE-2021-0354	In <code>ged</code> , there is a possible out of bounds write due to an integer overflow. This could lead to local escalation of privilege with System execution privileges needed. User interaction is not needed for exploitation. Product: Android; Versions: Android-11; Patch ID: ALPS05431161.
CVE-2021-0312	In <code>WAVSource::read</code> of <code>WAVExtractor.cpp</code> , there is a possible out of bounds write due to an integer overflow. This could lead to remote information disclosure with no additional execution privileges needed. User interaction is needed for exploitation. Product: Android; Versions: Android-8.1, Android-9, Android-10, Android-11, Android-8.0; Android ID: A-170583712.
CVE-2020-9875	An integer overflow was addressed through improved input validation. This issue is fixed in iOS 13.6 and iPadOS 13.6, macOS Catalina 10.15.6, tvOS 13.4.8, watchOS 6.2.8, iTunes 12.10.8 for Windows, iCloud for Windows 11.3, iCloud for Windows 7.20. Processing a maliciously crafted image may lead to arbitrary code execution.

3.5 整数溢出漏洞

标识	提交时间	漏洞等级	漏洞名称
CNVD-2020-61109	2020-11-09	—	研华科技WebAccess HMI Runtime存在整数溢出漏洞（CNVD-2020-61109）
CNVD-2020-61117	2020-11-09	—	研华科技WebAccess HMI PanelSim.exe存在整数溢出漏洞
CNVD-2020-38883	2020-07-16	—	Bitcoin wxBitcoin/bitcoind 整数溢出漏洞
CVE-2014-8182	2020-01-02	—	OpenLDAP整数溢出漏洞
CVE-2016-8795	2017-04-02	—	多款Huawei CloudEngine产品整数溢出漏洞
CVE-2015-8982	2017-03-15	—	GNU C Library 远程整数溢出漏洞
CVE-2016-9824	2017-03-01	—	Libav 整数溢出漏洞
CVE-2016-8859	2017-02-13	—	musl libc'tre_tnfa_run_parallel()整数溢出漏洞
CVE-2016-7944	2016-12-13	—	X.Org libXfixes 整数溢出漏洞
CVE-2016-7947	2016-12-13	—	X.Org libXrandr 整数溢出漏洞

1 2 3 4 5 6 ... 87 88 共 873 条

3.5 整数溢出漏洞

□ Adobe Shockwave Player Director文件解析整数溢出漏洞 (CVE-2010-0129)

- 发布时间：2010—5-13
- 危害等级：超危
- 漏洞详情：

Adobe Shockwave Player是美国奥多比（Adobe）公司的一款多媒体播放器产品，它存在多个整数溢出，远程攻击者可通过**触发数组索引**的特制.dir(即Director)文件引发**拒绝服务**(内存破坏)。

3.5 整数溢出漏洞

- 两个整数算术运算（加减乘）、关系运算、赋值运算会触发整数溢出漏洞
 - 运算结果不符合预期：大于，小于，True（False）
 - 运算结果用于内存分配、内存访问、分支选择.....
 - 数组下标、对象计数器
 - 内存分配和拷贝的长度约束（malloc,memcpy, memset, strcpy, strcat.....）
- 整数溢出漏洞的危害
 - 越权读、越权写
 - 远程代码执行
 - 内存错误
- 整数溢出漏洞的检测与防御
 - 审查运算结果
 - 约束运算结果的使用

课后思考

- 如何检测printf的格式化字符串参数个数与实参个数不一致？
- 算术运算容易溢出，有什么措施可以预防溢出？
- 数组和指针容易越界访问，设计一个越界检查的工具。
- 举例说明浮点数的溢出。