

DOCUMENTO SUL QUARTO HOMEWORK – MATTEO FANFARILLO

Il quarto homework consiste nel determinare ogni informazione possibile riguardo le funzionalità del programma hw4.ex_. Per far ciò, ho seguito i passaggi qui riportati:

- 1) Descrizione preliminare del programma.
- 2) Formalizzazione dell'obiettivo.
- 3) Ottenimento del codice macchina.
- 4) Analisi statica di base.
- 5) Analisi dinamica di base.
- 6) Disassemblaggio del codice macchina.
- 7) Localizzazione dei frammenti assembly di interesse.
- 8) Analisi gray box del codice di interesse in modo da determinare le informazioni riguardanti le funzionalità del programma.
- 9) Riepilogo delle informazioni ottenute (questo passaggio lo sto portando a termine ora mentre scrivo il qui presente documento).

1) DESCRIZIONE PRELIMINARE DEL PROGRAMMA

Prima di iniziare l'analisi dell'eseguibile, ero a conoscenza solamente delle seguenti informazioni:

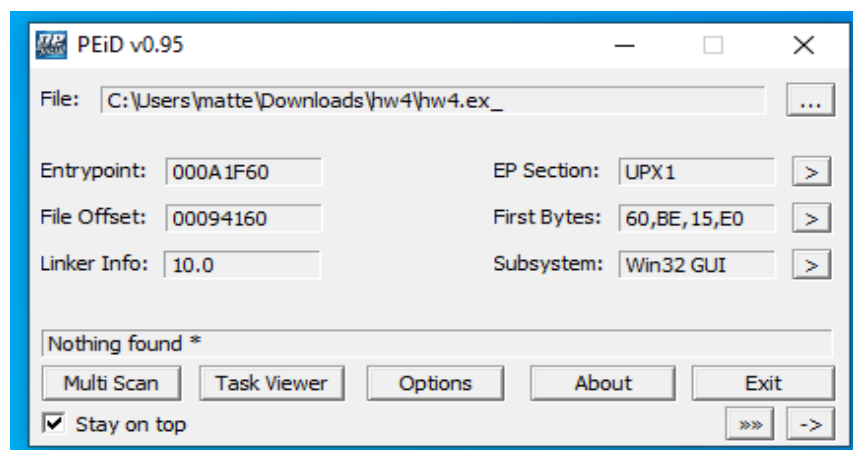
- Si tratta di un programma Windows a 32 bit scritto con il linguaggio C e/o C++.
- È un malware, come riportato da un avviso allegato all'homework. Per questo motivo, prima di cominciare l'homework, ho preparato un ambiente sicuro in cui effettuare l'analisi del malware; in altre parole, ho catturato uno snapshot alla mia macchina virtuale Windows (in modo da poter ripristinare lo stato "clean" dopo che avrò terminato l'analisi) e ho avviato la VM, in cui ho disattivato le protezioni fornite da Windows Defender e ho scaricato hw4.ex_.

2) FORMALIZZAZIONE DELL'OBIETTIVO

L'obiettivo dell'homework è analizzare il programma con gli strumenti di analisi che ho a disposizione, in modo tale da ottenere quante più informazioni possibili concernenti il funzionamento e lo scopo del malware.

3) OTTENIMENTO DEL CODICE MACCHINA

Per prima cosa è opportuno verificare se hw4.ex_ è impacchettato (ovvero è stato dato in pasto a un packer), per cui ho aperto l'eseguibile su PEiD, ottenendo il seguente risultato:



Apparentemente PEiD non ha trovato alcun packer applicato all'eseguibile ma, se prestiamo attenzione, possiamo osservare che la sezione dell'entry point si chiama UPX1: questo è un forte indizio del fatto che hw4.ex_ sia stato impacchettato tramite UPX.

Di fatto, PeStudio conferma tale teoria. Di seguito sono riportate alcune informazioni fornite da PeStudio tramite l'analisi della testata dell'eseguibile:

pestudio 9.22 - Malware Initial Assessment - www.winitor.com

file settings about

c:\users\matte\downloads\hw4\hw4.ex_

Indicators (37)

- virustotal (wait...)
- dos-header (64 bytes)
- dos-stub (64 bytes)
- rich-header (n/a)
- file-header (Jun.2015)
- optional-header (GUI)
- directories (4)
- sections (files)
- libraries (5) *
- functions (8)
- exports (n/a)
- tls-callbacks (n/a)
- .NET (n/a)
- resources (unknown) *
- strings (5949)
- debug (n/a)
- manifest (n/a)
- version (n/a)
- certificate (n/a)
- overlay (n/a)

indicator (37)	detail	level
The file references string(s)	type: blacklist, count: 3	1
The file is scored by virustotal	score: 43/68	1
The size of a resource is suspicious	resource: DCXA.1	1
The size of a resource is suspicious	resource: DCXA.1	1
The size of a resource is suspicious	resource: DCXA.2	1
The size of a resource is suspicious	resource: DCXA.2	1
The file references functions(s)	type: blacklist, count: 1	1
The file contains a blacklist section	section: UPX0	1
The file contains a blacklist section	section: UPX1	1
The first section is writable	section: UPX0	1
The location of the entry-point is suspicious	section: UPX1:0x000A1F60	1
The file contains self-modifying executable section(s)	status: yes	1
The file contains writable and executable section(s)	count: 2	1
The file contains another file	signature: unknown, location: UPX1, offset: 0x000892...	2
The file contains another file	signature: unknown, location: UPX1, offset: 0x000896...	2
The file references blacklist library(ies)	count: 1	2
The file contains a virtualized section	section: UPX0	2
The file checksum is invalid	checksum: 0x00000000	3
The file references a group of API	type: file, count: 2	3
The file references a group of API	type: execution, count: 3	3
The file references a group of API	type: dynamic-library, count: 5	3
The file references a group of API	type: memory, count: 3	3
The file references a group of hint	type: utility, count: 3	3
The file references a group of hint	type: file, count: 8	3
The file references a group of hint	type: format-string, count: 4	3
The file references a group of hint	type: function, count: 4	3

sha256: 7B572046DF86D9DFAD31E6FACC4DEBD2E3D6877A38338BFC1B085903DC3958F2C cpu: 32-bit file-type: executable subsystem: GUI entry-point: 0x000A1F60

Da qui emerge che esistono almeno due sezioni sospette nel file: UPX0 e UPX1, che certamente non si presentano in un'applicazione non impacchettata.

Inoltre, sempre su PeStudio, ho cercato le stringhe presenti nel programma, e le uniche interessanti sono le seguenti (tutte le altre sono apparentemente randomiche e prive di senso, il che fa pensare che l'eseguibile possa essere stato criptato con qualche algoritmo di cifratura):

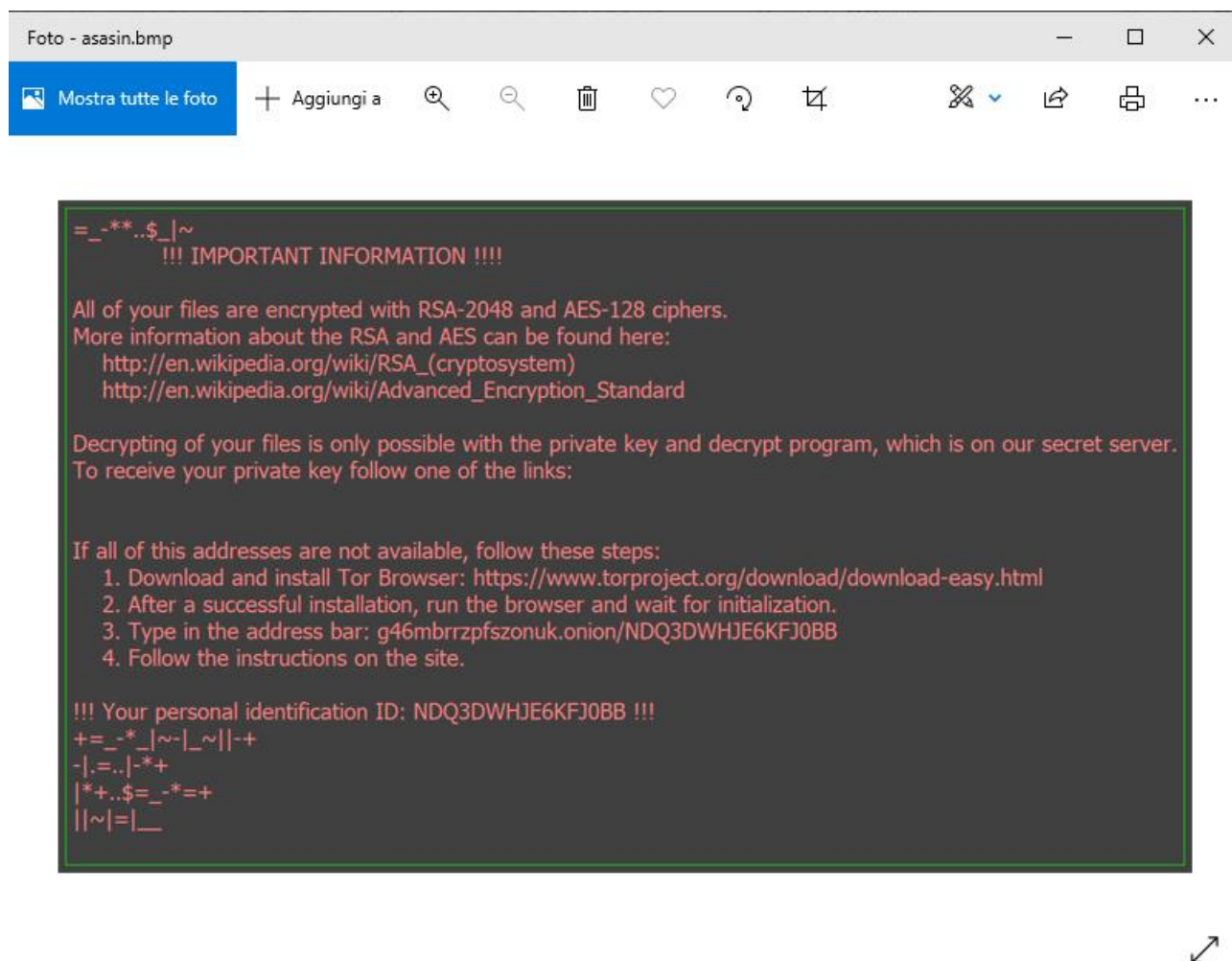
```

value (5949)
UPX0
UPX1
Process emory
ExitProcess
GetProcAddress
VirtualProtect
SE_IsShimDll
:a%S
{ %s1V'
4g%S6
1^o%su|
xK.h
Hz:C
q:t.H
cmutil.dll
KERNEL32.DLL
shell32.dll
shimeng.dll
user32.dll
!This program cannot be run in DOS mode.
E3yU
.rsrc
3.96
UPX!

```

Dopo aver constatato che hw4.ex_ è impacchettato, ho effettuato i miei tentativi di “unpacking” a partire dall’utilizzo dei tool automatici.

1) **PEiD**: ho eseguito il comando “Find OEP” di PEiD che, in linea teorica, prevede l’esecuzione dello stub dell’applicazione; tuttavia, l’esecuzione è andata fuori controllo e il malware è stato eseguito per intero. Infatti, lo sfondo del desktop è cambiato, si è aperta una pagina del browser, si è aperta un’immagine e si è verificato un grosso cambiamento all’interno del file system; in particolare, sia lo sfondo del desktop, sia la pagina del browser, sia l’immagine recitavano così:



Ho approfittato di questo incidente di percorso per stabilire che questo malware non è altro che un ransomware; inoltre, fortunatamente, non ha arrecato alcun danno alla macchina fisica, per cui ogni sua esecuzione all’interno di un ambiente protetto è in qualche modo legittima. Certo è che, durante la fase centrale dell’analisi, sarà necessario farsi periodicamente un backup di tutti i file contenenti informazioni utili all’interno della macchina virtuale (come ad esempio il file del progetto Ghidra relativo all’analisi di hw4.ex_).

In ogni caso, come ogni volta in cui il malware verrà eseguito, mi è stato necessario spegnere la macchina virtuale, ripristinare lo stato “clean” ed effettuare il reboot.

Per quanto riguarda l’original entry point del programma, PEiD non è riuscito a rilevarlo, per cui, a maggior ragione, non sarebbe in grado di effettuare l’unpacking per intero.

2) **UPX**: poiché l’eseguitibile sembrerebbe impacchettato tramite UPX, è sensato tentare di effettuare il procedimento inverso tramite questo tool. Apparentemente l’esito dello spaccettamento è positivo e UPX ha generato la sua versione di eseguibile contrassegnata come “unpacked”. Ho dunque caricato e

analizzato su Ghidra il nuovo eseguibile (che nel frattempo ho ridenominato hw4-upx.ex_), e ho notato che:

- È stato generato del codice assembly apparentemente relativo a un eseguibile spaccettato, con una funzione entry e svariate altre funzioni.

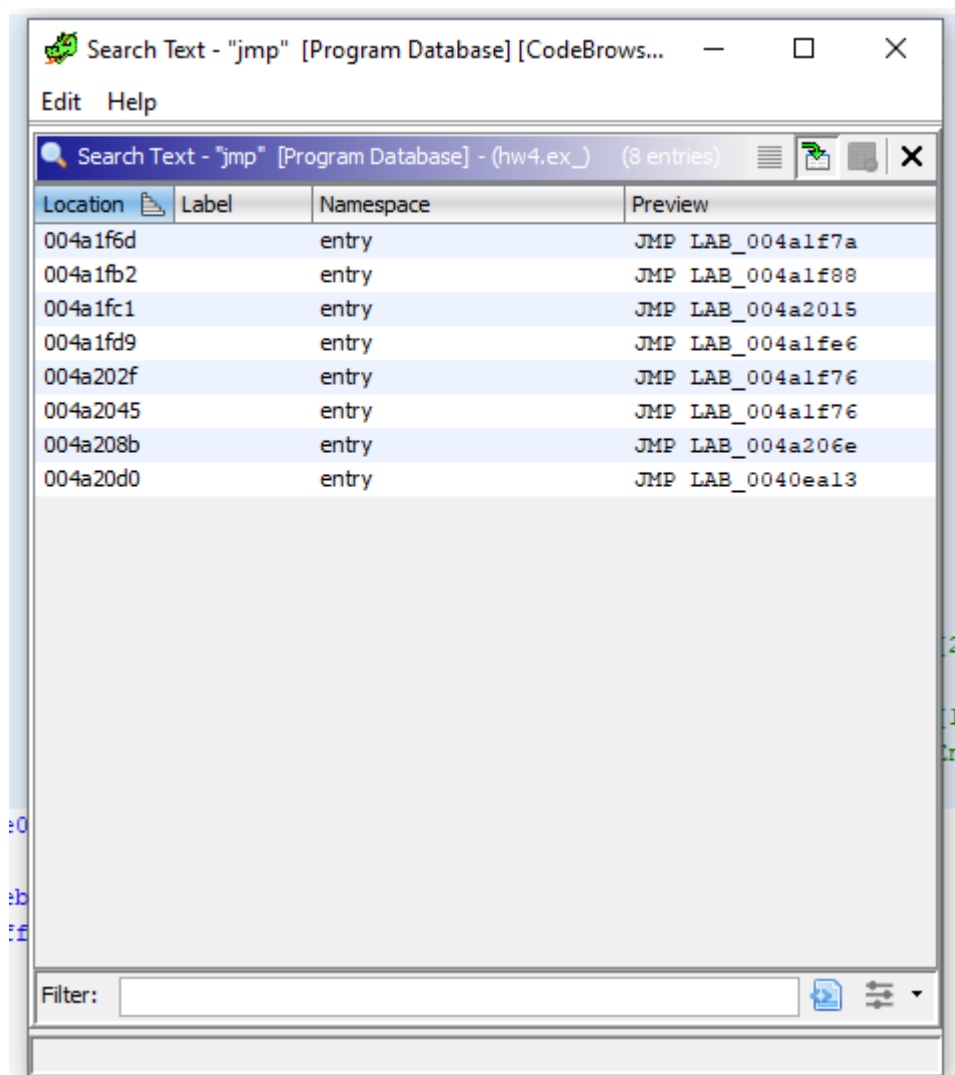
- Gli import sono apparentemente risolti: figurano varie DLL importate in ciascuna delle quali risulta utilizzato un numero congruo di API.

Per avere conferma sul successo dello spaccettamento, ho provato a eseguire hw4-upx.ex_; purtroppo è andato in crash. Su OllyDbg ho visto che il crash è dovuto a un accesso non valido in memoria (scrittura su 0x00494040) che avviene all'istruzione di indirizzo 0x7761e189. Inoltre, la voce LastErr in OllyDbg è posta uguale a ERROR_FILE_NOT_FOUND (00000002). Ciò potrebbe voler dire che la Import Address Table dell'applicazione non sia stata ricostruita completamente e che qualche riferimento sia saltato.

Per questo motivo, ho deciso di effettuare altri tentativi con strumenti diversi per fare l'unpacking di hw4.ex_, nella speranza di ottenere esiti migliori.

3) **PE Explorer:** qui l'esito dello spaccettamento è stato pressoché identico a UPX.

4) **Ricerca dell'OEP tramite un salto (jmp) lontano:** ho caricato hw4.ex_ su Ghidra per vedere se sarei riuscito a trovare l'istruzione di tail jump, che non è altro che il salto tra l'ultima istruzione dello stub e la prima istruzione del programma originale. Ho dunque cercato su Ghidra la lista di tutte le istruzioni di jmp presenti in hw4.ex_ e ho ottenuto la seguente tabella:



The screenshot shows the 'Search Text' window in Ghidra, with the search term 'jmp' and the file 'hw4.ex_'. The results are displayed in a table with four columns: Location, Label, Namespace, and Preview. There are 8 entries listed, all of which are 'entry' type instructions jumping to various labels.

Location	Label	Namespace	Preview
004a1f6d		entry	JMP LAB_004a1f7a
004a1fb2		entry	JMP LAB_004a1f88
004a1fc1		entry	JMP LAB_004a2015
004a1fd9		entry	JMP LAB_004a1fe6
004a202f		entry	JMP LAB_004a1f76
004a2045		entry	JMP LAB_004a1f76
004a208b		entry	JMP LAB_004a206e
004a20d0		entry	JMP LAB_0040ea13

Tenendo conto che la prima istruzione dello stub si trova all'indirizzo 0x004a1f60, i primi 7 jump dell'elenco non sono interessanti: sono salti vicini che avvengono all'interno della medesima funzione. È invece particolare l'ultima istruzione di jmp, che ha come indirizzo sorgente 0x004a20d0 e come indirizzo destinazione 0x0040ea13: probabilmente è lui il tail jump che cercavamo.

Tale supposizione è stata immediatamente smentita dal fatto che all'indirizzo di 0x0040ea13, in Ghidra, ho trovato del codice assembly, il che vuol dire che anche lì è presente un po' di codice dello stub.

Possiamo concludere che neanche questa tecnica ci ha portati a un risultato.

5) Comando Find OEP by Section Hop di OllyDump: eseguendo questo comando, sia con l'opzione Trace Into che con l'opzione Trace over, l'esecuzione dell'applicazione su OllyDbg rimane bloccata nello stato di tracing per svariato tempo, per cui ho concluso che neanche questa tecnica mi è d'aiuto.

6) Inserimento di un breakpoint sullo stack: la prima istruzione dello stub è una PUSHAD. L'ho eseguita tramite una Step Into, ho inserito un breakpoint hardware in accesso all'indirizzo dello stack puntato dal registro ESP e ho avviato l'esecuzione, nella speranza che tale indirizzo dello stack venga riacceduto solo al termine dello stub (subito prima della tail jump). Purtroppo però non è così: mi sono infatti ritrovato nell'istruzione situata all'indirizzo 0x0040ea13, che abbiamo già appurato non corrispondere all'OEP.

7) Esecuzione step-by-step dello stub con l'inserimento di un breakpoint alla fine di ogni ciclo: in mancanza d'altro, ho deciso di ripercorrere le istruzioni dello stub passo dopo passo (a partire dall'indirizzo 0x0040ea13) finché non avrei trovato il codice relativo all'eseguibile originale (spacchettato). Chiaramente, man mano ho inserito un breakpoint hardware alla fine di ogni loop e agli indirizzi dello stack corrispondentemente alle istruzioni PUSHAD, in modo tale da non rimanere bloccato per troppo tempo all'interno dei cicli.

Qui ho potuto vedere come l'eseguibile originale sia stato caricato in memoria a partire dall'indirizzo 0x00400000 tramite delle chiamate a VirtualProtect, VirtualAlloc e le successive istruzioni REP MOVSB (un altro indizio è dato dal registro EDI che, in una fase dell'esecuzione, assumeva, nell'ordine, i valori ".text", ".rdata", ".data", ".reloc", ".pdata", che sono delle stringhe indicanti le sezioni dell'address space di un eseguibile non impacchettato); successivamente, sono state caricate anche le DLL utilizzate dall'eseguibile originale.

A un tratto ho incontrato l'istruzione JMP ESI, dove il registro ESI conteneva l'indirizzo 0x00402d8f: questo ha tutta l'aria di essere il tail jump. L'ho eseguito con una Step Into e ho salvato il dump del processo così ottenuto mediante il comando Dump Debugged Process del plugin OllyDump:

- Dapprima ho eseguito tale comando con l'opzione Rebuild Import attivata e col metodo 1 per la ricostruzione della IAT (Search JMP[API] | CALL[API] in memory image), generando così un nuovo eseguibile sperabilmente "unpacked" che ho ridenominato hw4-olly1.exe.

- In secondo luogo l'ho eseguito con l'opzione Rebuild Import attivata e col metodo 2 per la ricostruzione della IAT (Search DLL & API name string in dumped file), generando così un nuovo eseguibile sperabilmente "unpacked" che ho ridenominato hw4-olly2.exe.

- Infine l'ho eseguito con l'opzione Rebuild Import disattivata, generando così un ulteriore eseguibile che ho ridenominato hw4-olly0.exe. Chiaramente esso non può funzionare così com'è perché certamente non ha la IAT ricostruita. Probabilmente è un eseguibile che non servirà ma, comunque, non è svantaggioso tenerlo in sordina.

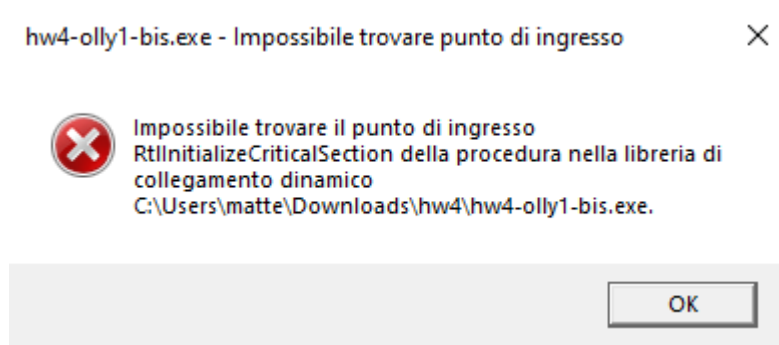
A questo punto ho dato in pasto a Ghidra hw4-olly1.exe e ho notato che:

- Anche stavolta è stato generato del codice assembly apparentemente relativo a un eseguibile spacchettato, con una funzione entry e svariate altre funzioni.

- Gli import sono apparentemente risolti: figurano varie DLL importate in ciascuna delle quali risulta utilizzato un numero congruo di API. In particolare, le DLL risultanti sono: combase.dll, dsrole.dll, kernel32.dll, mpr.dll e urlmon.dll.

Successivamente ho caricato su Ghidra anche l'applicazione hw4-olly2.exe e ho ottenuto un risultato del tutto analogo rispetto a hw4-olly1.exe. Tuttavia, ho notato che qui le DLL importate sono un po' diverse, ovvero: advapi32.dll, gdi.dll, kernel32.dll e mpr.dll. Sembrerebbe dunque che il metodo 2 della ricostruzione della IAT di OllyDump abbia individuato alcune DLL ed API importate ma ne abbia tralasciate altre, così come il metodo 1. Per avere conferma di ciò, ho provato a lanciare entrambi gli eseguibili:

- hw4-olly1.exe ha mostrato il seguente messaggio di errore:



- hw4-olly2.exe, invece, è crashato immediatamente senza mostrare messaggi di errore (il che diventa palese con l'utilizzo di Process Explorer).

L'ultima spiaggia per ottenere un eseguibile unpacked funzionante consiste nell'utilizzo del tool ImpREC (Import REConstructor). Questo tool, per effettuare l'unpacking, richiede che l'applicazione in questione (hw4.ex_) sia nello stato running ma senza l'utilizzo di OllyDbg. Nel nostro caso ciò è possibile, perché ho potuto osservare che, quando il malware viene lanciato, rimane in esecuzione per oltre un minuto prima che cifri i miei file, apra il browser, apra l'immagine, cambi lo sfondo del desktop e si chiuda.

L'unico problema è che non sono in grado di lanciare un'applicazione con estensione ".ex_" tramite un semplice doppio click. Fortunatamente, mi è stato sufficiente modificare l'estensione in ".exe" in modo tale da ottenere un eseguibile identico ma facilmente avviabile tramite il doppio click.

A questo punto, ho avviato hw4.exe, su ImpREC ho selezionato il processo di hw4.exe, ho inserito l'OEP (che ho assunto essere 0x00402d8f) e ho cliccato sul bottone IAT AutoSearch. Sfortunatamente, ImpREC ha mostrato un messaggio di errore che indicava l'invalidità dell'OEP inserito. Perciò, il massimo che ho potuto fare è stato ammazzare il processo relativo a hw4.exe tramite Process Explorer.

Poiché non ho in sordina ulteriori strategie per ottenere un eseguibile spaccettato funzionante, si conclude qui la fase dell'analisi relativa all'ottenimento del codice macchina.

Comunque sia, continuo ad assumere che l'OEP dell'eseguibile sia con una certa probabilità 0x00402d8f, anche perché ho appurato tramite Ghidra che hw4-olly1.exe e hw4-olly2.exe contengono il codice macchina dell'applicazione originale in chiaro. Per quanto riguarda ImpREC, sicuramente non è un tool infallibile!

4) ANALISI STATICA DI BASE

Prima di buttarci a capofitto sull'utilizzo del disassemblatore e del debugger, proviamo a raccogliere quante più informazioni possibili tramite gli strumenti di analisi di base, a partire da quelli di analisi statica.

Per prima cosa vediamo quali sono le **stringhe** che figurano nella sezione Defined Strings di Ghidra per quanto riguarda hw4-olly1.exe. Qui emergono:

- Le stringhe relative alle dll e alle API importate (approfondiremo tra breve questo aspetto).
- Le stringhe indicanti i giorni della settimana, i mesi e i formati delle date (come "MM/dd/yy", "dddd, MMMM dd, yyyy", "HH:mm:ss").

- Alcuni messaggi di errore, per lo più riguardanti allocazione di memoria e inizializzazione di strutture dati: purtroppo non hanno fornito informazioni interessanti.
- La stringa “asasin”, che avevo già visto quando il malware mi era sfuggito dal controllo:



- Poche altre stringhe momentaneamente poco significative.

Concentriamoci sulle **API importate** perlustrando la sezione Imports di Ghidra per hw4-olly1.exe e hw4-olly2.exe: ce ne sono a iosa, per cui non è conveniente analizzarle tutte qui a una a una. Piuttosto, spenderemo qualche parola su quelle che sembrano più interessanti dal punto di vista degli scopi del malware:

- OpenProcessToken, LookupPrivilegesA e AdjustTokenPrivileges: servono per effettuare un’escalation dei privilegi. Il malware, dunque, per portare a termine il lavoro di cifratura dei file, potrebbe aver bisogno di acquisire qualche privilegio particolare.
- CryptGenRandom, CryptHashData, CryptEncrypt, CryptDestroyKey e CryptReleaseContext: hanno tutta l’aria di essere delle funzioni atte a generare e distruggere delle chiavi di cifratura e a criptare delle informazioni (presumibilmente i file che risiedono nella macchina in cui gira il ransomware).
- InitializeSecurityDescriptor e SetSecurityDescriptorDacl: servono rispettivamente a inizializzare un security descriptor di un oggetto (e.g. un file) e a ridefinirne la DACL (Discretionary Access Control List), in modo da modificare i permessi di accesso per quel particolare oggetto.
- AccessCheck: determina se un security descriptor prevede che un determinato client abbia determinati diritti di accesso a un particolare oggetto.
- MapGenericMask: mappa i diritti di accesso a un oggetto su una maschera di 32 bit.
- CreateProcessW e CreateThread: probabilmente, il ransomware lancia altri processi e/o altri thread.
- CreateFileA, CreateFileW, CopyFileW, MoveFileExW, ReadFile, WriteFile e DeleteFileW: anche da queste API è evidente che il malware manipola dei file all’interno del sistema.

Dopodiché ho aperto il malware su **PeStudio** e ho ricavato un altro paio di informazioni interessanti:

- Il campo “signature” è pari a Microsoft Visual C++ 8.
- Il compiler-stamp indica che l’eseguibile è stato compilato il 7 giugno 2008: si tratta di un malware vecchio, per cui non dovrebbe essere eccessivamente sofisticato, e la sua firma hash dovrebbe essere ben nota.

5) ANALISI DINAMICA DI BASE

Consideriamo ora qualche informazione di base riguardante il comportamento del malware.

Abbiamo avuto già modo di constatare durante la fase di ottenimento del codice macchina, per via di un

incidente di percorso, che abbiamo a che fare con un ransomware (ovvero un malware che cifra i file che risiedono all'interno della macchina vittima). Sappiamo anche che nascondersi non è una priorità del malware: al termine della sua esecuzione, infatti, mostra il messaggio che ho precedentemente riportato all'interno del presente documento mediante ben tre mezzi: una pagina del browser, un'immagine e lo sfondo del Desktop. In particolare, tale avviso riporta che tutti i file vengono criptati coi cifrari RSA-2048 e AES-128.

Dopo la cifratura, i file si presentano così e non possono essere aperti dall'utente vittima:

	Nome	Ultima modifica	Tipo
Accesso rapido			
Desktop			
Download			
Documenti			
Immagini			
Download per M			
hw4			
	NDQ3DWHJ-E6KF-J0BB-8F438F4B-EE54F3DAC047.asasin	11/01/2022 21:54	File ASASIN
	NDQ3DWHJ-E6KF-J0BB-461D43B8-97EC568E8C3E.asasin	11/01/2022 21:54	File ASASIN
	NDQ3DWHJ-E6KF-J0BB-F5DB6E93-53B2636E219A.asasin	11/01/2022 21:55	File ASASIN

Qui possiamo notare una cosa: "asasin" è anche la nuova estensione dei file a seguito della loro cifratura.

Per provare a estrapolare qualche informazione in più, ho lanciato il ransomware e, mediante Process Monitor, ho catturato gli eventi che esso ha scaturito:

Operation	Path	Result	Detail
RegOpenKey	HKLM\Software\WOW6432Node\Microsoft\LanguageOverlay\OverlayPackages\it-IT	REPARSE	Desired Access: Read
RegOpenKey	HKLM\SOFTWARE\Microsoft\LanguageOverlay\OverlayPackages\it-IT	SUCCESS	Desired Access: Read
RegSetInfoKey	HKLM\SOFTWARE\Microsoft\LanguageOverlay\OverlayPackages\it-IT	SUCCESS	KeySetInformationClass: KeySetHandleTagsInformat
RegQueryValue	HKLM\SOFTWARE\Microsoft\LanguageOverlay\OverlayPackages\it-IT\Latest	SUCCESS	Type: REG_SZ, Length: 210, Data: C:\Program Files
RegCloseKey	HKLM\SOFTWARE\Microsoft\LanguageOverlay\OverlayPackages\it-IT	SUCCESS	
CreateFile	C:\Program Files\WindowsApps\Microsoft.LanguageExperiencePackit-IT_19041.35.108.0_neutral_...	SUCCESS	Desired Access: Read Data/List Directory, Synchron
CreateFileMapping	C:\Program Files\WindowsApps\Microsoft.LanguageExperiencePackit-IT_19041.35.108.0_neutral_...	FILE LOCKED WITH ONLY READE...	Sync Type: SyncTypeCreateSection, PageProtection
QueryStandardInfor...	C:\Program Files\WindowsApps\Microsoft.LanguageExperiencePackit-IT_19041.35.108.0_neutral_...	SUCCESS	AllocationSize: 1.208.320, EndOfFile: 1.1656, NumberO
CreateFileMapping	C:\Program Files\WindowsApps\Microsoft.LanguageExperiencePackit-IT_19041.35.108.0_neutral_...	SUCCESS	Sync Type: SyncTypeOther
CreateFile	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	Desired Access: Read Attributes, Disposition: Open,
QueryBasicInformati...	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	CreationTime: 28/07/2021 17:30:16, LastAccessTim
CloseFile	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	
CreateFile	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	
CreateFileMapping	C:\Windows\SysWOW64\VBBoxMRXNP.dll	FILE LOCKED WITH ONLY READE...	Desired Access: Read Data/List Directory, Execute/
QueryStandardInfor...	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	Sync Type: SyncTypeCreateSection, PageProtection
ReadFile	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	AllocationSize: 1.208.320, EndOfFile: 1.205.432, Nui
ReadFile	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	Offset: 0, Length: 4.096, I/O Flags: Non-cached, Pa
RegOpenKey	HKLM\System\CurrentControlSet\Control\CI	SUCCESS	Offset: 261.632, Length: 16.384, I/O Flags: Non-cac
RegOpenKey	HKLM\System\CurrentControlSet\Control\CI	REPARSE	Desired Access: Read
RegOpenKey	HKLM\System\CurrentControlSet\Control\CI	SUCCESS	Desired Access: Read
RegQueryValue	HKLM\System\CurrentControlSet\Control\CI\Disable26178932	NAME NOT FOUND	Length: 20
RegCloseKey	HKLM\System\CurrentControlSet\Control\CI	SUCCESS	
RegOpenKey	HKLM\System\CurrentControlSet\Control\CI	REPARSE	Desired Access: Query Value
RegOpenKey	HKLM\System\CurrentControlSet\Control\CI	SUCCESS	Desired Access: Query Value
RegQueryValue	HKLM\System\CurrentControlSet\Control\CI\Disable26178932	NAME NOT FOUND	Length: 80
RegCloseKey	HKLM\System\CurrentControlSet\Control\CI	SUCCESS	
RegOpenKey	HKLM\System\CurrentControlSet\Control\CI	REPARSE	Desired Access: Read
RegOpenKey	HKLM\System\CurrentControlSet\Control\CI	SUCCESS	Desired Access: Read
RegQueryValue	HKLM\System\CurrentControlSet\Control\CI\Disable26178932	NAME NOT FOUND	Length: 20
RegCloseKey	HKLM\System\CurrentControlSet\Control\CI	SUCCESS	
QueryStandardInfor...	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	AllocationSize: 1.208.320, EndOfFile: 1.205.432, Nui
CreateFileMapping	C:\Windows\SysWOW64\VBBoxMRXNP.dll	FILE LOCKED WITH ONLY READE...	Sync Type: SyncTypeCreateSection, PageProtection
QueryStandardInfor...	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	AllocationSize: 1.208.320, EndOfFile: 1.205.432, Nui
CreateFileMapping	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	Sync Type: SyncTypeOther
ReadFile	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	Offset: 1.171.456, Length: 32.768, I/O Flags: Non-ci
ReadFile	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	Offset: 1.204.224, Length: 1.208, I/O Flags: Non-ca
RegOpenKey	HKLM\System\CurrentControlSet\Control\CI	REPARSE	Desired Access: Read
RegOpenKey	HKLM\System\CurrentControlSet\Control\CI	SUCCESS	Desired Access: Read
RegQueryValue	HKLM\System\CurrentControlSet\Control\CI\Disable26178932	NAME NOT FOUND	Length: 20
RegCloseKey	HKLM\System\CurrentControlSet\Control\CI	SUCCESS	
ReadFile	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	Offset: 1.089.024, Length: 16.384, I/O Flags: Non-ci
ReadFile	C:\Windows\SysWOW64\VBBoxMRXNP.dll	SUCCESS	Offset: 1.122.304, Length: 16.384, I/O Flags: Non-ci

Qui ho riportato solo una frazione piccolissima di eventi ma, di fatto, essi non sono particolarmente eterogenei: la stragrande maggioranza riguarda le letture delle chiavi di registro e alcune operazioni sui file di sistema (per lo più DLL), come Create, Read, Query Information e Close. È bene sottolineare che questi eventi avvengono prima della cifratura vera e propria dei file.

Durante la fase finale dell'esecuzione, invece, gli eventi predominanti sono quelli riportati di seguito:

```
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Sync Data\LevelDB\LOCK SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Sync Data\LevelDB\LOG SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Site Characteristics Database... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Site Characteristics Database... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Site Characteristics Database... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Subresource Filter\Indexed Rules\34... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Subresource Filter\Indexed Rules\34... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\GPUCache\data_2 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\GPUCache\data_2 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Service Worker\Database\... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Service Worker\Database\... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Extension State\LOG SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Extension State\LOCK SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Extension State\MANIFEST-... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Extension State\000003.log SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\EdgeCoupons\coupons_dat... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Windows\System32\it-IT\kernel32.dll.mui SUCCESS Name: \Windows\System32\it-IT\kernel32.dll.mui
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\EdgeCoupons\coupons_dat... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\data_reduction_proxy_level... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\load_statistics.db SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Program Files\WindowsApps\Microsoft.LanguageExperiencePack\it-IT_19041.35.108.0_neutral... SUCCESS Name: \Program Files\WindowsApps\Microsoft.Langu
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\History\journal SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\heavy_ad_intervention_opt_... SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\shared_proto_db\LOG SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\GPUCache\data_0 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Collections\collections.SQLite SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\Default\Local Storage\leveldb\LOCK SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Windows\System32\it-IT\KernelBase.dll.mui SUCCESS Name: \Windows\System32\it-IT\KernelBase.dll.mui
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\ShaderCache\GPUCache\index SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\GrShaderCache\GPUCache\data_0 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\GrShaderCache\GPUCache\data_3 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\GrShaderCache\GPUCache\index SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\GrShaderCache\GPUCache\index SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\GrShaderCache\GPUCache\index SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\ShaderCache\GPUCache\data_0 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\ShaderCache\GPUCache\data_0 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\ShaderCache\GPUCache\data_1 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\ShaderCache\GPUCache\data_1 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\ShaderCache\GPUCache\data_2 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\ShaderCache\GPUCache\data_2 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
QueryNameInformationFile C:\Users\matte\AppData\Local\Microsoft\Edge\User Data\GrShaderCache\GPUCache\data_0 SUCCESS Name: \Users\matte\AppData\Local\Microsoft\Edge
```

Come si può notare, sono innumerevoli eventi di tipo QueryNameInformationFile che coinvolgono svariati file.

6) DISASSEMBLAGGIO DEL CODICE MACCHINA

Nonostante nella fase di ottenimento del codice macchina non sia riuscito a ottenere un eseguibile unpacked funzionante, non ritengo di aver avuto un fallimento; di fatto, l'analisi gray box del malware è assolutamente fattibile:

- È possibile effettuare un'esecuzione controllata del malware hw4.ex_ (o, equivalentemente, hw4.exe) su OllyDbg.
- È possibile effettuare l'analisi statica del codice che caratterizza effettivamente il malware aprendo su Ghidra hw4-olly1.exe e/o hw4-olly2.exe; tra l'altro, combinando l'analisi di questi due eseguibili, è verosimile che si riesca a risalire a (quasi) tutte le DLL e le API importate dal malware.

Di fatto, scegliendo questa strada, mi ritrovo il codice macchina di interesse già disassemblato nella fase 3 dell'analisi (ottenimento del codice macchina). Dando uno sguardo veloce alle varie funzioni che compongono il programma, sembrerebbe che non siano state applicate particolari tecniche anti-disassembler, per cui posso reputare questa fase dell'analisi già conclusa. Se alcune tecniche anti-disassembler dovessero emergere successivamente, me ne occuperò a tempo debito.

7) LOCALIZZAZIONE DEI FRAMMENTI ASSEMBLY DI INTERESSE

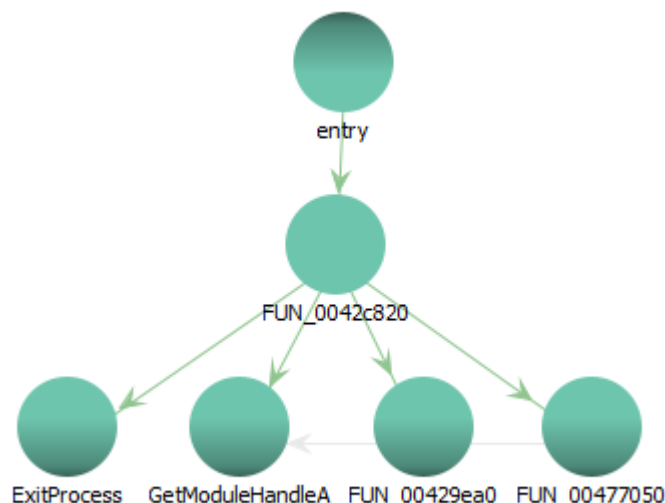
L'obiettivo di questa fase è trovare il codice scritto dal programmatore. Qui mi sono fatto aiutare da Ghidra, in cui ho dato uno sguardo alla funzione entry (che inizia in prossimità dell'indirizzo 0x00402d8f).

In particolare, all'interno di entry vengono invocate le seguenti funzioni:

FUNZIONE	TIPOLOGIA FUNZIONE
__security_init_cookie	Funzione di libreria (Visual Studio 2010 Release)
__SEH_prolog4	Funzione di libreria (Visual Studio)
GetStartupInfoW	API (Kernel32.dll)
HeapSetInformation	API (Kernel32.dll)
__heap_init	Funzione di libreria (Visual Studio 2010 Release)
__fast_error_exit	Funzione di libreria (Visual Studio 2010 Release)
__mtinit	Funzione di libreria (Visual Studio 2010 Release)
__RTC_Initialize	Funzione di libreria (Visual Studio 2010 Release)
__ioinit	Funzione di libreria (Visual Studio 2010 Release)
__amsg_exit	Funzione di libreria (Visual Studio 2010 Release)
GetCommandLineA	API (Kernel32.dll)
__crtGetEnvironmentStringsA	Funzione di libreria (Visual Studio 2010 Release)
__setargv	Funzione di libreria (Visual Studio 2010 Release)
__setenvp	Funzione di libreria (Visual Studio 2010 Release)
__cinit	Funzione di libreria (Visual Studio 2010 Release)
__wincmdln	Funzione di libreria (Visual Studio 2010 Release)
FUN_0042c820	Funzione che sembrerebbe essere scritta dal programmatore
_exit	Funzione di libreria (Visual Studio 2010 Release)
__cexit	Funzione di libreria (Visual Studio 2010 Release)

Dalla tabella qui sopra, può sembrare che FUN_0042c820 sia candidata a essere la funzione main, poiché dovrebbe essere l'unica funzione scritta dal programmatore invocata da entry.

Osserviamo inoltre il Function Call Graph relativo a FUN_0042c820:



Da qui emerge anche che entry è l'unica funzione che invoca FUN_0042c820.

Infine, il decompilatore contrassegna FUN_0042c820 come una funzione senza parametri di input (il che è ammissibile per un main).

Di conseguenza, per il momento assumo che FUN_0042c820 sia effettivamente la funzione main; se successivamente dovessi accorgermi di aver commesso uno sbaglio, correggerò di conseguenza.

8) ANALISI GRAY BOX DEL CODICE DI INTERESSE

Poiché le funzioni presenti all'interno dell'eseguibile sono numerosissime (e alcune di loro sembrano anche piuttosto corpose), è illogico pensare di analizzarle tutte nel dettaglio. Piuttosto, dobbiamo essere bravi nel concentrarci nelle funzioni che sono il cuore del ransomware e contribuiscono maggiormente a causare danni ai sistemi informatici.

Il main è semplice: dopo aver recuperato l'handle di hw4.ex_ attraverso una chiamata a GetModuleHandleA, invoca la funzione FUN_00477050. Poiché quest'ultima non è facilissima da leggere neanche col decompilatore, mi sono fatto aiutare dal debugger.

L'esecuzione di tale funzione è abbastanza farraginosa poiché inizia con un'esecuzione molto intensiva di jump. Dopo un po' ho realizzato di essere finito in un loop, poiché stavo eseguendo dei salti verso indirizzi che si ripetevano ciclicamente. In particolare tali indirizzi sono:

0x004778d8
0x004779f6
0x004777af
0x004774a1
0x00477979
0x004773d0
0x00477334
0x00477894
0x004778a3
0x004774af
0x004775d1
0x0047798f
0x00477398
0x004770c6
0x0047728a
0x00477610
0x00477728
0x004770b6
0x004776c3

Andando a vedere il compilatore, ho avuto conferma del fatto che mi trovavo all'interno di un ciclo while. In particolare, nel momento in cui due valori (ottenibili dal module handle – che nel debugger ho visto corrispondere all'indirizzo 0x00400000 – tramite deferenza e spiazzamento) soddisfano una particolare condizione, si esce dal ciclo: ragionevolmente si stanno cercando dei campi specifici all'interno dell'header dell'eseguibile spaccettato.

All'interno del loop di jump ho cercato dunque dei salti condizionali, che possono essere visti come bivi, e ne ho trovati 3:

- JC LAB_00477979; JMP LAB_00477a0a

Di questi due salti, quello che conduce fuori dal loop è JMP LAB_00477a0a; tuttavia, porta subito a una return, per cui non va bene.

- JZ LAB_00477728; JMP LAB_004775d1

Nessuno di questi due salti conduce fuori dal loop.

- JNZ LAB_00477728; JMP LAB_0047716b

Di questi due salti, quello che conduce fuori dal loop è JMP LAB_0047716b, ed è effettivamente quello che vogliamo prendere.

Di conseguenza, ho fissato un breakpoint all'indirizzo 0x0047716b e ho proseguito con l'esecuzione del programma.

Successivamente viene invocata la VirtualAlloc ma il punto interessante lo si ha a partire dall'indirizzo 0x00477573, da cui, omettendo le varie jump e le varie NOP, si hanno le seguenti istruzioni:

```
MOV  EAX, FS:[0x18]
MOV  EAX, dword ptr [EAX + 0x30]
MOVZX EAX, byte ptr [EAX + 0x2]
MOV  dword ptr [EBP + local_8], EAX
CMP  dword ptr [EBP + local_8], EDX      ; dove EDX = 0
JZ   LAB_00477a28
JMP  LAB_00477311
```

Qui, tramite le prime due istruzioni, viene effettuato un accesso alla PEB (Process Environment Block); dopodiché si ha uno spiazzamento di due byte per leggere il campo BeingDebugged della PEB e memorizzarlo all'interno della variabile locale local_8; infine, viene valutato il valore di local_8 (ovvero il valore di BeingDebugged): se è pari a 0, si esegue il salto verso 0x00477a28, che porta alla prosecuzione dell'esecuzione del malware; altrimenti, si esegue il salto verso 0x00477311, che porta all'invocazione di GetModuleHandleA, GetProcAddress e memcpy, per poi avere praticamente la terminazione del processo.

Di fatto, siamo appena incappati in una tecnica anti-debugger. Per superarla, è opportuno effettuare una patch a hw4.ex_ tramite OllyDbg, in modo tale da sostituire l'istruzione JZ LAB_00477a28 con JMP LAB_00477a28. Purtroppo non è possibile salvare un nuovo eseguibile modificato: la patch riguarda un'istruzione che viene caricata in memoria dallo stub solo a run-time, per cui è impossibile renderla persistente su disco. Poco male: il lavoro può comunque proseguire senza problemi, dato che il OllyDbg tiene conto anche delle patch non salvate in modo persistente.

Successivamente, come riporta anche il decompilatore, c'è un secondo ciclo do-while, anch'esso composto da una serie di jump che si ripetono ciclicamente. In particolare, gli indirizzi attraversati all'interno del loop sono:

```
0x004777ef
0x004777de
0x0047779b
0x00477562
0x0047788b
0x00477526
0x004770b2
0x004770d3
0x004771c4
0x004777c4
0x00477609
0x00477308
0x004775f9
```

0x0047787a
0x00477123
0x004775e2
0x00477511
0x00477544
0x00477695
0x004777aa
0x00477807
0x0047791f
0x004776b9
0x00477466
0x004776b2
0x004779df
0x00477782
0x00477357
0x004771f2
0x0047751e
0x0047761c
0x0047724e
0x0047749c

Stavolta, il salto condizionale presente all'interno del ciclo è solo uno, ed è posto all'indirizzo 0x0047724e. Infatti qui si hanno le seguenti due istruzioni: JC LAB_0047749c; JMP LAB_00477837.

Il salto condizionale verso l'indirizzo 0x0047749c viene preso ogni volta che si hanno ancora ulteriori iterazioni da effettuare all'interno del loop.

Ho dunque fissato un breakpoint all'indirizzo 0x00477837 e ho proseguito con l'esecuzione del programma.

Dopodiché, all'interno della funzione FUN_00477050, c'è qualche altro controllo, un'invocazione a VirtualAlloc, una chiamata a un altro paio di funzioni (FUN_0046e870, FUN_0046e940) e un'invocazione a VirtualFree.

Ho dato un'occhiata a FUN_0046e870, FUN_0046e940 e le relative funzioni che esse chiamano a loro volta e, apparentemente, non sembra esserci un granché di interessante. Perciò, ho deciso di lasciarle almeno momentaneamente da parte e di verificare se il comando Step Over di OllyDebug in prossimità della loro invocazione desse problemi: poiché sembra essere andato tutto liscio, posso concludere che non sono state prese ulteriori misure anti-debugger all'interno di queste funzioni.

La funzione FUN_00477050, che abbiamo appena ripercorso, non ha implementato la funzionalità principale del ransomware ma è risultata interessante per due motivi:

- Tra quelle che abbiamo incontrato, è stata la prima funzione sufficientemente grande a contenere più istruzioni di salto e NOP che altre istruzioni assembly. Questo costringe l'analista a prestare molta attenzione durante l'analisi, poiché non ha davanti una sequenza di istruzioni chiara e lineare, bensì del codice in cui è molto facile rimanere disorientati e perdere di vista le istruzioni importanti che sembrano inserite casualmente tra tante NOP e tante jump (all'inizio della pagina seguente ho inserito un estratto di codice che mostra per bene questo aspetto).

In realtà moltissime altre funzioni (tra cui lo stesso main che, essendo di dimensioni ridotte, non rende perfettamente il senso di "disorientamento") hanno una struttura di questo tipo: ne terremo conto e le affronteremo di conseguenza, senza ovviamente trascrivere di nuovo alcuni dettagli come ho fatto per FUN_00477050.

```

0047726d 90          NOP
0047726e e9 78 06    JMP         LAB_004778eb
          00 00

          LAB_00477273
00477273 90          NOP
00477274 aa      STOSB      ES:EDI
00477275 e9 65 02    JMP         LAB_004774df
          00 00

          LAB_0047727a
0047727a 90          NOP
0047727b c9      LEAVE
0047727c 90          NOP
0047727d e9 bd 07    JMP         LAB_00477a3f
          00 00

          LAB_00477282
00477282 90          NOP
00477283 8d 09      LEA        ECX, [ECX]
00477285 e9 f5 02    JMP         LAB_0047757f
          00 00

          LAB_0047728a
0047728a 90          NOP
0047728b 39 70 04    CMP         dword ptr [EAX + 0x4],ESI
0047728e e9 7d 03    JMP         LAB_00477610
          00 00

          LAB_00477293
00477293 56      PUSH      ESI
00477294 90          NOP
00477295 e9 19 06    JMP         LAB_004778b3
          00 00

```

- Il secondo aspetto interessante di FUN_00477050 consiste nella misura anti-debugger che sfrutta il campo BeingDebugged del Process Environment Block. Poiché è effettivamente questo il tratto caratteristico della funzione, ho deciso di ridenominare quest'ultima anti_debugger_PEB.

Dopo la terminazione di anti_debugger_PEB, il main invoca la funzione FUN_00429ea0. Poiché si tratta dell'ultima funzione chiamata dal main, rappresenta con ogni probabilità il cuore del ransomware: l'ho dunque ridenominata (almeno per il momento) ransomware_clue. Notiamo anche che non sembra avere parametri in input.

La prima cosa che fa anti_debugger_PEB è invocare la funzione FUN_00401018, che è molto piccola:


```

00401018 6a ff      PUSH     -0x1
0040101a 50         PUSH     EAX
0040101b 64 a1 00   MOV      EAX,FS:[0x0]
           00 00 00
00401021 50         PUSH     EAX
00401022 8b 44 24 0c MOV      EAX,dword ptr [ESP + local_res0]
00401026 64 89 25   MOV      dword ptr FS:[0x0],ESP
           00 00 00 00
0040102d 89 6c 24 0c MOV      dword ptr [ESP + local_res0],EBP
00401031 8d 6c 24 0c LEA      EBP=>local_res0,[ESP + 0xc]
00401035 50         PUSH     EAX
00401036 c3         RET

```

Qui sono particolarmente d’interesse le seguenti istruzioni:

```

PUSH  EAX
MOV   EAX, FS:[0x0]
PUSH  EAX
MOV   dword ptr FS:[0x0], ESP

```

È abbastanza evidente che si sta aggiungendo un nuovo elemento alla SEH chain (Structured Exception Handler chain), che è una lista collegata i cui nodi puntano a dei gestori di eccezione dell’applicazione. Questo tipo di operazione potrebbe, anche se non è detto, essere sfruttato come meccanismo anti-debugger o anti-disassembler. Per questo motivo, teniamo conto della presenza di tale funzione e la ridenominiamo `append_SEH_chain`.

Comunque sia, su OllyDbg ho potuto constatare che, inizialmente, il registro EAX conteneva l’indirizzo `0x0042c70b`: è qui che inizia la funzione del gestore appena inserito. Sono andato a darvi un’occhiata e c’è qualche jump seguito dall’invocazione di una funzione di libreria (Visual Studio 2005 Release): `FID_conflict:___CxxFrameHandler3`.

Tornando a `ransomware_clue`, dopo la chiamata ad `append_SEH_chain`, effettua un controllo sulla variabile globale `DAT_004831f8`, che era stata inizializzata dal main al valore di ritorno di `anti_debugger_PEB`. Così, sono tornato a dare uno sguardo ad `anti_debugger_PEB` e ho notato che restituisce l’indirizzo ritornato dall’ultima `VirtualAlloc`. Ricordiamo che `VirtualAlloc` è un’API che alloca della memoria e restituisce l’indirizzo base dell’area di memoria in questione. Inoltre, accetta i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPVOID	lpAddress	Indirizzo base della regione di memoria da allocare.	NULL
2	SIZE_T	dwSize	Dimensione in byte della regione di memoria da allocare.	25748
3	DWORD	flAllocationType	Tipo di allocazione di memoria.	MEM_COMMIT MEM_RESERVE
4	DWORD	flProtect	Protezione che si vuole applicare alla memoria da allocare.	PAGE_READWRITE

Riassumendo, `DAT_004831f8` assume il valore restituito dall’ultima `VirtualAlloc` di `append_SEH_chain`, per cui ridenominò questa variabile globale in `GLOBAL_ALLOCATED_MEM`.

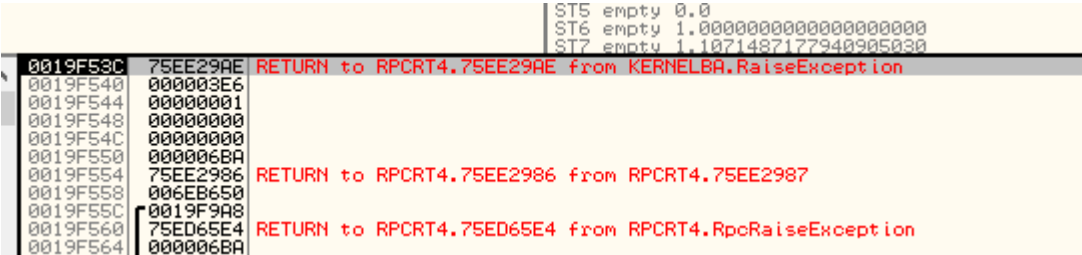
Tornando a `ransomware_clue`, si ha un controllo su se `GLOBAL_ALLOCATED_MEM` è pari a NULL: se sì, la funzione termina immediatamente, altrimenti si procede a invocare `SetErrorMode` (che controlla se il sistema o il processo è in grado di gestire un particolare tipo di errore serio) e `SetUnhandledExceptionFilter` (che fa in modo che, se si verifica un’eccezione non gestita e non c’è alcun debugger, venga invocata la

funzione passata come parametro che, nel nostro caso, inizia all’indirizzo 0x0041f820). Successivamente, ransomware_clue invoca FUN_0042cf00, che non sembra di particolare interesse, per cui, almeno per il momento, la saltiamo. Seguono le invocazioni a GetSystemDefaultLangID(), GetUserDefaultLangID e GetUserDefaultUILanguage, che recuperano rispettivamente il language identifier del sistema, il language identifier dell’utente corrente e il language identifier della lingua UI dell’utente corrente.

Poi si entra in un ciclo do-while colossale, in cui viene invocato un gran numero di funzioni: proviamo a vedere quelle che, a occhio, sembrano più rilevanti.

La prima funzione che mi ha colpito è FUN_00431440 poiché, verso la fine, chiama CryptReleaseContext, che è una delle API che mi erano balzate all’occhio durante l’analisi statica di base. Con l’aiuto del decompilatore, ho controllato velocemente le varie funzioni invocate da FUN_00431440, e tutte quelle che precedono FUN_0042cdd0 non hanno attirato la mia attenzione.

Ho dunque provato a eseguire il programma col debugger fino all’invocazione di FUN_0042cdd0, ma si è sollevata un’eccezione:



Ho dovuto riavviare il debugger ed eseguire FUN_00431440 in maniera più puntuale (ovvero tramite degli Step Into e/o Step Over) per capire qual è la causa dell’eccezione. Da qui ho capito che il problema nasce dall’API GetVolumeNameForVolumeMountPointA che viene chiamata da FUN_0042f2e0 che a sua volta viene invocata da FUN_0042f430.

GetVolumeNameForVolumeMountPointA è una funzione che recupera il path GUID di un volume (che è una stringa del tipo “\\?\Volume{26a21bda-a627-11d7-9931-806e6f6e6963}\”), e restituisce zero in caso di fallimento, un altro valore altrimenti; inoltre, accetta i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPCSTR	lpszVolumeMountPoint	Puntatore a una stringa che contiene il path di un mounted folder (che è un’associazione tra un volume e una directory in un altro volume) o di un drive letter (come “C:”).	“C:Windows\”
2	LPSTR	lpszVolumeName	Puntatore a un buffer che riceverà una stringa relativa al GUID path.	0x0019fafc
3	DWORD	cchBufferLength	Lunghezza del buffer di output.	260

Di fatto questa API restituisce 0, con last error pari a ERROR_NOT_A_REPARSE_POINT. Di conseguenza, come si può anche vedere dal decompilatore, viene sollevata un’eccezione:

In particolare, eseguendo il comando Step Over sulla chiamata a `__CxxThrowException@8`, mi ritrovo nella seguente situazione:

Da qui, passando l'eccezione all'applicazione con SHIFT+F9, essa sembra essere gestita correttamente, dato che si ritorna al di fuori della logica di eccezione. Perciò, l'esecuzione della funzione FUN_0042f2e0, e quindi di FUN_0042f2e0, termina senza problemi.

Finalmente possiamo dare un'occhiata a FUN_0042cdd0: è una funzione molto piccola contiene al suo interno una chiamata a CryptHashData. Un'altra caratteristica curiosa di questa funzione è che, almeno secondo il decompilatore, accetta il parametro this: questo fa pensare che, probabilmente, il malware, almeno in parte, sia stato scritto in C++. Ma, a prescindere da ciò, spendiamo due parole su CryptHashData: la documentazione della Microsoft riporta che, prima di tale API, deve essere chiamata CryptCreateHash. È abbastanza verosimile che CryptCreateHash venga invocata poco prima di CryptHashData e, quindi, di FUN_0042cdd0: la funzione chiamata subito prima di FUN_0042cdd0 all'interno di FUN_00431440 è FUN_0042cb70. Guardando all'interno di quest'ultima funzione, possiamo notare che c'è un'invocazione a un'altra procedura che però non è ben riconosciuta da Ghidra:

```
Decompile: FUN_0042cb70 - (hw4-olly2-poker.exe)

1
2 /* WARNING: Globals starting with '_' overlap smaller symbols at the same address */
3
4 undefined ** __thiscall
5 FUN_0042cb70(void *this,undefined4 *param_1,undefined4 param_2,undefined4 param_3)
6
7 {
8     int iVar1;
9     undefined **local_c;
10    void *local_8;
11
12    *(undefined4 *)this = 0;
13    local_c = (undefined **)this;
14    local_8 = this;
15    iVar1 = (*DAT_0047a04c)(*param_1,param_2,0,param_3,this);
16    if (iVar1 == 0) {
17        local_8 = (void *)GetLastError();
18        local_c = &PTR_LAB_0047c80c;
19        /* WARNING: Subroutine does not return */
20        __CxxThrowException@8(&local_c,&DAT_0047c868);
21    }
22    return (undefined **)this;
23 }
24
```

Proviamo a eseguire l'applicazione col debugger fino al punto in cui avviene l'invocazione di questa funzione (ovvero fino all'indirizzo 0x0042cbdf) per verificare se si tratta effettivamente di una CryptCreateHash.

```

> 90      NOP
0042CBDF  . FF15 4CA04700  CALL DWORD PTR DS:[47A04C]  ADVAPI32.CryptCreateHash
0042CBE5  . vEB 69      JMP SHORT hw4-copi.0042CC50
0042CBE7  > 51          PUSH ECX
0042CBE8  . 90          NOP
0042CBE9  . vEB 75      JMP SHORT hw4-copi.0042CC60
0042CBEB  > 90          NOP
```

Ed effettivamente è così! Su Ghidra ho dunque identificato DAT_0047a04c col nome CryptCreateHash. Tra l'altro, l'esecuzione col debugger fino all'istruzione di indirizzo 0x0042cbdf è andata a buon fine, il che vuol dire, ancora una volta, non dovremmo essere incappati in tecniche anti-debugger (d'ora in poi ometterò questa osservazione a meno di incontrare effettivamente un altro meccanismo anti-debugger).

CryptCreateHash, stando alla documentazione della Microsoft, inizializza una funzione hash per uno stream di dati. Restituisce TRUE se ha successo, FALSE altrimenti. I parametri che prende in input sono i seguenti:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HCRYPTPROV	hProv	Handle a un CSP (cryptographic service provider, che è un modulo software che esegue algoritmi crittografici per l'autenticazione e la cifratura) creato da CryptAcquireContext.	0x004c7960
2	ALG_ID	AlgId	Identificatore dell'algoritmo da usare.	CALG_MD5
3	HCRYPTKEY	hKey	Chiave della funzione hash (se l'algoritmo usato prevede l'utilizzo di una chiave).	0
4	DWORD	dwFlags	Flag	0

5	HCRYPTHASH*	phHash	Indirizzo in cui verrà copiato l'handle al nuovo oggetto hash.	[out] this
---	-------------	--------	--	------------

Possiamo osservare che CryptCreateHash, a sua volta, richiede che precedentemente sia stata invocato CryptAcquireContext. Per cercare tale API, ho adottato lo stesso metodo di prima: ho cercato nella funzione invocata subito prima di FUN_0042cb70 da parte di FUN_00431440, che sarebbe FUN_00416570.

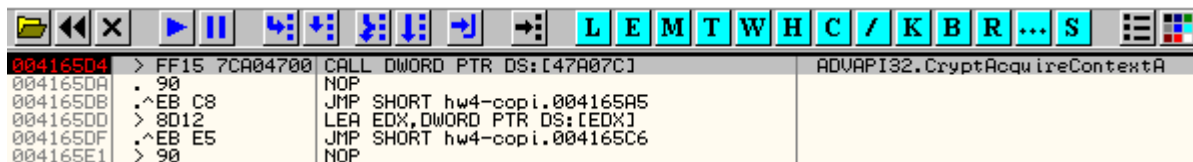
All'interno di FUN_00416570, come prima, c'è una chiamata a una procedura che Ghidra non riconosce:

```

4  undefined ** __thiscall FUN_00416570(void *this,undefined4 param_1,undefined4 param_2)
5
6  {
7      int iVar1;
8      undefined **local_c;
9      void *local_8;
10
11     *(undefined4 *)this = 0;
12     local_c = (undefined **)this;
13     local_8 = this;
14     iVar1 = (*_DAT_0047a07c)(this,0,0,param_1,param_2);
15     if (iVar1 == 0) {
16         local_8 = (void *)GetLastError();
17         local_c = &PTR_LAB_0047c80c;
18         /* WARNING: Subroutine does not return */
19         __CxxThrowException@8(&local_c,&DAT_0047c868);
20     }
21     return (undefined **)this;
22 }

```

Anche stavolta il debugger ha confermato le mie aspettative: DAT_0047a07c corrisponde all'API CryptAcquireContextA.



Tale API serve ad acquisire un handle a un particolare key container (che è una sezione del key database che contiene tutte le coppie di chiavi appartenenti a uno specifico utente) all'interno in un CSP.

Restituisce TRUE se ha successo, FALSE altrimenti. I parametri che prende in input sono i seguenti:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HCRYPTPROV*	phProv	Indirizzo in cui verrà copiato il puntatore al CSP.	[out] this
2	LPCSTR	szContainer	Nome del key container.	NULL
3	LPCSTR	szProvider	Nome del CSP da usare.	NULL
4	DWORD	dwProvType	Tipo di provider da acquisire.	24
5	DWORD	dwFlags	Flag	0xF0000000

Tornando a CryptCreateHash, c'è da osservare anche che il secondo parametro che gli viene passato è CALG_MD5, il che implica l'utilizzo di MD5 come funzione hash. Non siamo dunque giunti alla definizione delle funzioni di cifratura dei file del sistema (che sappiamo essere RSA-2048 e AES-128): probabilmente, in questa specifica sede, le API della famiglia Crypt hanno uno scopo differente.

Tornando a FUN_00431440, dopo la chiamata a FUN_0042cb70, viene invocata la funzione FUN_0042cdd0 che, come accennavo precedentemente, contiene una chiamata a CryptHashData, che è un'API che aggiunge dati all'oggetto hash creato tramite CryptCreateHash.

Anche CryptHashData restituisce TRUE se ha successo, FALSE altrimenti. I parametri che prende in input sono i seguenti:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HCRYPTHASH	hHash	Handle all'oggetto hash.	*this
2	const BYTE*	pbData	Puntatore a un buffer che contiene i dati che dovranno essere aggiunti all'oggetto hash.	0x0019FC8C
3	DWORD	dwDataLen	Numero di byte di dati che dovranno essere aggiunti.	3
4	DWORD	dwFlags	Flag	0

Successivamente, FUN_00431440 invoca FUN_00430010, che a sua volta chiama FUN_0042fa00, la quale di interessante ha una chiamata a FUN_0042ccb0.

FUN_0042ccb0 è interessante poiché ha la medesima struttura delle ultime funzioni viste. In particolare, contiene un'invocazione a CryptGetHashParam come OllyDbg può confermare.

CryptGetHashParam recupera i dati che governano le operazioni di un oggetto hash (tra cui il valore hash effettivo). Restituisce TRUE se ha successo, FALSE altrimenti. I parametri che prende in input sono i seguenti:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HCRYPTHASH	hHash	Handle all'oggetto hash.	*this
2	DWORD	dwParam	Tipo di query.	2
3	BYTE*	pbData	Puntatore a un buffer che riceverà i dati richiesti.	0x0019fb64
4	DWORD*	pdwDataLen	Puntatore a un intero che specifica le dimensioni del buffer pbData. Dopo l'invocazione della funzione, l'intero sarà pari al numero di byte memorizzati nel buffer.	0x0019fc00
5	DWORD	dwFlags	Parametro riservato che deve essere pari a 0.	0

Successivamente, all'interno di FUN_00431440, si ha un'invocazione a una procedura non riconosciuta da Ghidra, ma che OllyDbg è riuscito a identificare CryptDestroyHash. Tale API non fa altro che distruggere l'oggetto hash.

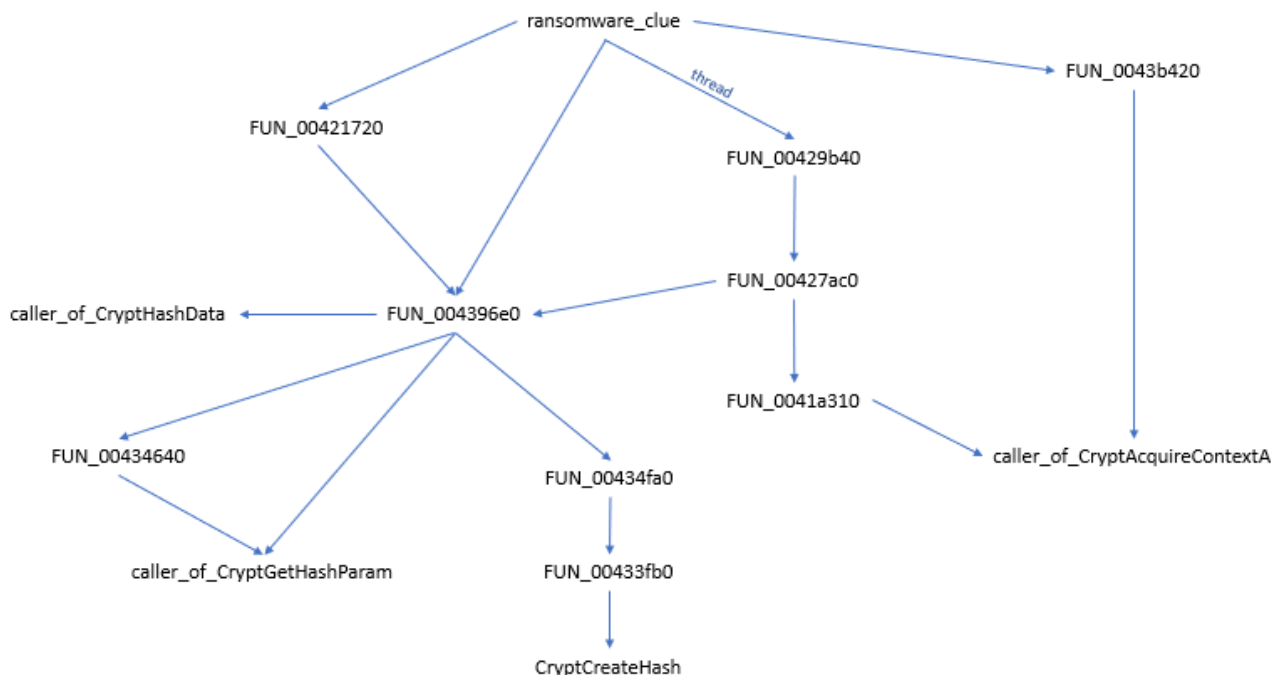
Dopodiché FUN_00431440 invoca anche CryptReleaseContext, che rilascia l'handle del CSP e del key container.

A valle di questa breve speculazione, mi sento di effettuare le seguenti ridenominazioni di funzioni:

- FUN_00431440 → crypt_MD5
- FUN_00416570 → caller_of_CryptAcquireContextA
- FUN_0042cb70 → caller_of_CryptCreateHash
- FUN_0042cdd0 → caller_of_CryptHashData
- FUN_00430010 → containing_caller_of_CryptGetHashParam
- FUN_0042ccb0 → caller_of_CryptGetHashParam

Ho poi eseguito per intero la funzione crypt_MD5 su OllyDbg e ho dato un'occhiata ai file della mia macchina virtuale: come immaginavo, non sembra essere avvenuto nulla di rilevante.

Così, tramite Ghidra, ho cercato altri riferimenti alle API CryptAcquireContextA, CryptCreateHash, CryptHashData e CryptGetHashParam, ottenendo il seguente schema di invocazioni:



Dal grafico sembra centrale il ruolo della funzione FUN_004396e0, che viene invocata da tre altre funzioni (ransomware_clue, FUN_00421720 e FUN_00427ac0) e contiene al suo interno delle chiamate a CryptCreateHash, caller_of_CryptHashData e caller_of_CryptGetHashParam.

La prima a invocare FUN_004396e0 è FUN_00421720, la quale, a giudicare da come appare nel decompilatore, ha una struttura abbastanza curiosa, per cui ho pensato che possa valere la pena darvi uno sguardo.

Tuttavia, dando un'occhiata più accurata al decompilatore di Ghidra, ho notato che le funzioni FUN_0043b420, FUN_00421720 e FUN_004396e0 vengono invocate all'interno di un ramo else e, più specificatamente, se la condizione `*(char*)(GLOBAL_ALLOCATED_MEM + 15) == '\0'` non viene rispettata. Eseguendo ransomware_clue sul debugger tramite il comando Step Over, ho realizzato che tale ramo else non viene preso, per cui le tre funzioni sopra elencate non dovrebbero mai essere chiamate.

A questo punto, potrebbe essere utile fare due cose:

- Capire meglio in cosa consiste la condizione `*(char*)(GLOBAL_ALLOCATED_MEM + 15) == '\0'`.
- Verificare se all'interno del ramo if (corrispondente alla condizione verificata) viene eseguita qualche operazione interessante (al di là, dunque, delle API di crittografia che abbiamo incontrato precedentemente).

Per quanto riguarda il punto (a), ricordiamo che la variabile locale GLOBAL_ALLOCATED_MEM contiene l'indirizzo di un'area di memoria che è stata allocata e inizializzata all'interno della funzione anti_debugger_PEB. Nel momento in cui tale indirizzo viene caricato di cui sopra, ho EAX per effettuare il controllo di cui sopra, ho usato il comando Follow in Dump per vedere il contenuto dell'area di memoria:

Il corrispondente ramo else (che non verrà eseguito dal nostro malware) è il seguente:

```
Decompile: ransomware_clue - (hw4-olly2-poker.exe)
289     else {
290         *(undefined2 *) (unaff_EBP + -0x88) = 0x73;
291         *(undefined2 *) (unaff_EBP + -0x86) = 0x76;
292         *(undefined2 *) (unaff_EBP + -0x84) = 99;
293         *(undefined2 *) (unaff_EBP + -0x82) = 0x68;
294         *(undefined2 *) (unaff_EBP + -0x80) = 0x6f;
295         *(undefined2 *) (unaff_EBP + -0x7e) = 0x73;
296         *(undefined2 *) (unaff_EBP + -0x7c) = 0x74;
297         *(undefined2 *) (unaff_EBP + -0x7a) = 0x2e;
298         *(undefined2 *) (unaff_EBP + -0x78) = 0x65;
299         *(undefined2 *) (unaff_EBP + -0x76) = 0x78;
300         *(undefined2 *) (unaff_EBP + -0x74) = 0x65;
301         *(undefined2 *) (unaff_EBP + -0x72) = 0;
302         FUN_004213b0();
303         lpNewFileName = *(LPCWSTR *) (unaff_EBP + -0x2c);
304         *(undefined *) (unaff_EBP + -4) = 3;
305         if (*(uint *) (unaff_EBP + -0x18) < 8) {
306             lpNewFileName = (LPCWSTR) (unaff_EBP + -0x2c);
307         }
308         lpExistingFileName = *(LPCWSTR *) (unaff_EBP + -0xac);
309         if (*(uint *) (unaff_EBP + -0x98) < 8) {
310             lpExistingFileName = (LPCWSTR) (unaff_EBP + -0xac);
311         }
312         BVar4 = CopyFileW(lpExistingFileName, lpNewFileName, 0);
313         if (BVar4 == 0) {
314 code_r0x0042a6df:
315             *(undefined *) (unaff_EBP + -4) = 2;
316             FUN_0040b940((void *) (unaff_EBP + -0x2c), '\x01', (void *) 0x0);
317             goto code_r0x0042a036;
318         }
319         *(undefined2 *) (unaff_EBP + -100) = 0x3a;
320         *(undefined2 *) (unaff_EBP + -0x62) = 0x5a;
321         *(undefined2 *) (unaff_EBP + -0x60) = 0x6f;
322         *(undefined2 *) (unaff_EBP + -0x5e) = 0x6e;
```

La quarta volta che GLOBAL_ALLOCATED_MEM viene acceduta all'interno di ransomware_clue è quella in cui ci siamo imbattuti poc'anzi: qui viene controllato il byte all'offset 0xf = 15 dell'area di memoria e, poiché è nullo, si entra all'interno del ramo if (che è indicato dalla freccia verde all'interno dell'immagine situata nella pagina precedente).

Il corrispondente ramo else, invece, è il seguente:

```
Decompile: ransomware_due - (hw4-olly2-poker.exe)

114     else {
115         *(undefined4 *) (unaff_EBP + -0x18) = 0xf;
116         *(undefined4 *) (unaff_EBP + -0x1c) = 0;
117         *(undefined *) (unaff_EBP + -0x2c) = 0;
118         FUN_0041fd90((void *) (unaff_EBP + -0x2c), puVar9);
119         puVar9 = (undefined4 *) (GLOBAL_ALLOCATED_MEM + 0x3f);
120         *(undefined *) (unaff_EBP + -4) = 5;
121         *(undefined4 *) (unaff_EBP + -0x78) = 0xf;
122         *(undefined4 *) (unaff_EBP + -0x7c) = 0;
123         *(undefined *) (unaff_EBP + -0x8c) = 0;
124         FUN_0041fd90((void *) (unaff_EBP + -0x8c), puVar9);
125         *(undefined *) (unaff_EBP + -4) = 6;
126         FUN_0043b420();
127         skippedl_1((void *) (unaff_EBP + -0x8c), '\x01', (void *) 0x0);
128         *(undefined *) (unaff_EBP + -4) = 2;
129         skippedl_1((void *) (unaff_EBP + -0x2c), '\x01', (void *) 0x0);
130         iVar11 = FUN_004351a0();
131         *(undefined4 *) (unaff_EBP + -0x34) = 0x454c4544;
132         *(undefined2 *) (unaff_EBP + -0x30) = 0x4554;
133         *(undefined *) (unaff_EBP + -0x2e) = 0;
134         *(int *) (unaff_EBP + -0x40) = iVar11 * 2;
135         *(undefined *) (unaff_EBP + -4) = 7;
136         puVar9 = (undefined4 *) FUN_00421720();
137         *(undefined *) (unaff_EBP + -4) = 8;
138         puVar9 = FUN_004203a0(&DAT_00481fd0, puVar9);
139         uVar16 = FUN_0041abc0(puVar9, (byte *) (unaff_EBP + -0x34));
140         *(bool *) (unaff_EBP + -0x39) = (int) uVar16 == 0;
141         *(undefined *) (unaff_EBP + -4) = 7;
142         skippedl_1((void *) (unaff_EBP + -0xfc), '\x01', (void *) 0x0);
143         if (*(char *) (unaff_EBP + -0x39) != '\0') goto code_r0x0042a821;
144         *(undefined2 *) (unaff_EBP + -0x38) = 0x6469;
145         *(undefined *) (unaff_EBP + -0x36) = 0x3d;
146         *(undefined *) (unaff_EBP + -0x35) = 0;
```

Notiamo che qui, come dicevo, si hanno le invocazioni alle funzioni FUN_0043b420 e FUN_00421720; poco più avanti, anche se non si vede nell'immagine, viene direttamente chiamata anche FUN_004396e0.

Ora resta da capire se esistono operazioni abbastanza rilevanti svolte all'interno del ramo if corrispondente alla condizione `*(char *) (GLOBAL_ALLOCATED_MEM + 15) == '\0'`.

Al solito, ho dato un'occhiata veloce alle funzioni che vengono invocate all'interno di questo blocco di codice: in realtà non sembra che ce ne sia alcuna di attraente. Solo FUN_00423740 è apparentemente più complessa ma, poiché non sembra neanche lei contenere il cuore del ransomware, per ora preferisco saltarla; se dovesse servire, ci torneremo più avanti.

A questo punto, ho eseguito il programma col debugger fino all'istruzione precedente all'invocazione alla funzione FUN_0046c640 (ho dunque eseguito tutto il ramo if): è andato tutto a buon fine e ancora non sembra che ci siano state ripercussioni nel file system e nella macchina virtuale in generale.

Ricordiamo che avevamo adocchiato la funzione FUN_004396e0, che abbiamo appurato essere invocata in quattro punti diversi, di cui tre non raggiungibili perché appartenenti a un ramo else che non viene preso.

Abbiamo inoltre visto che la quarta invocazione a tale funzione avviene a partire da FUN_00429b40, che in realtà è una funzione eseguita da qualche thread:

```
202     FUN_0046c640();
203     iVar11 = GLOBAL_ALLOCATED_MEM;
204     *(undefined *) (unaff_EBP + -4) = 0x15;
205     if (*(char *) (iVar11 + 0x6493) != '\0') {
206         FUN_00476b50();
207     }
208     for (lpParameter = *(LPVOID *) (unaff_EBP + -0xe0);
209         lpParameter != *(LPVOID *) (unaff_EBP + -0xdc);
210         lpParameter = (LPVOID) ((int)lpParameter + 0x1c)) {
211         in_stack_fffffde4 = (undefined4 *) 0x42a279;
212         pvVar5 = CreateThread((LPSECURITY_ATTRIBUTES) 0x0, 0, (LPTHREAD_START_ROUTINE) LAB_00429b40,
213                               lpParameter, 0, (LPDWORD) (unaff_EBP + -0x30));
214         *(undefined4 *) (unaff_EBP + -0x34) = 0;
```

Questo vuol dire che FUN_004396e0 verrà eseguita da dei thread, e ciò avverrà sicuramente perché l'API CreateThread si trova in un blocco di codice che viene eseguito dal malware.

Possiamo dunque riprendere in considerazione la possibilità di analizzare delle funzioni che contengono delle invocazioni alle API della famiglia Crypt (come CryptCreateHash, CryptHashData e così via); fra l'altro, come possiamo vedere nell'immagine qui sopra riportata, tra il punto a cui siamo arrivati con l'analisi e il ciclo for in cui viene invocata CreateThread ci separano solo due chiamate a funzione: FUN_0046c640 e FUN_00476b50. Queste due funzioni sembrano molto interessanti, per cui nel frattempo occupiamoci di loro.

La prima cosa che fa FUN_0046c640, a parte chiamare append_SEH_chain, è invocare per ben tre volte consecutive la funzione FUN_0046c150.

FUN_0046c150 parte con l'invocazione a WNetOpenEnumW, che è un'API che inizia un'enumerazione di una risorsa contenitore di rete; il suo valore di ritorno è NO_ERROR in caso di successo, un codice di errore di sistema altrimenti. I parametri che accetta in input sono i seguenti:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	DWORD	dwScope	Ambito / scopo dell'enumerazione.	RESOURCE_CONNECTED
2	DWORD	dwType	Tipi di risorse che devono essere enumerate.	RESOURCETYPE_DISK
3	DWORD	dwUsage	Tipi di utilizzo delle risorse che devono essere enumerate.	0 (=all resources)
4	LPNETRESOURCEW	lpNetResource	Puntatore a una risorsa NETRESOURCE che specifica il contenitore da enumerare.	NULL
5	LPHANDLE	lpEnum	Puntatore a un handle di enumerazione che può essere utilizzato in una successiva chiamata a WnetEnumResource.	[out] 0x00482008

Invece, verso la fine, FUN_0046c150 contiene una chiamata a WNetAddConnection2W, che effettua una connessione a una risorsa di rete e può redirezionare un dispositivo locale a quella risorsa di rete.

Tutto sommato, FUN_0046c150 lavora con le risorse di rete: la ridenominiamo network_resource_fun e, almeno momentaneamente, passiamo oltre.

Tornando alla funzione FUN_0046c640, un'operazione interessante che esegue a seguito delle chiamate a network_resource_fun è una chiamata a GetLogicalDrives, che è un'API che restituisce una maschera di bit

indicante le unità disco attualmente disponibili (in particolare, il bit 0 corrisponde all'unità A, il bit 1 corrisponde all'unità B, il bit 2 corrisponde all'unità C e così via). Dopodiché si entra in un grande ciclo do-while che inizia con il seguente controllo su tale maschera di bit:

```
if (maschera_di_bit & 1 << ((EBP-0x14) & 0x1f)) != 0
```

dove EBP-0x14 era stato inizialmente posto uguale a 1, per cui la condizione diventa:

```
if (maschera_di_bit & 1 << (1 & 0x1f)) != 0
```

Che è equivalente a:

```
if (maschera_di_bit & 1 << 1) != 0
```

Ovvero:

```
if (maschera_di_bit & 2) != 0
```

E cioè:

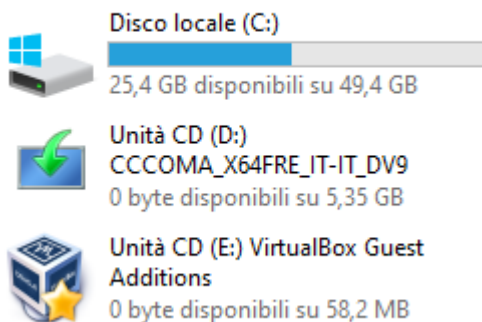
```
if (l'unità B è stata rilevata)
```

In particolare, se l'unità B viene rilevata, vengono invocate altre funzioni come GetDriveTypeW e/o GetDiskFreeSpaceExW e/o GetVolumeInformationW:

- GetDriveTypeW determina se un drive di disco è un drive removibile, un drive fisso, un CD-ROM, un RAM disk oppure un drive di rete.
- GetDiskFreeSpaceExW recupera le informazioni riguardo la quantità di spazio totale di un volume del disco, la quantità di spazio libera del volume e la quantità di spazio libera per lo specifico utente associato al thread chiamante.
- GetVolumeInformationW recupera le informazioni sul file system e sul volume associati alla root directory specificata.

Nel mio caso specifico, queste ultime chiamate a funzione non avvengono alla prima iterazione del ciclo do-while perché nella mia macchina virtuale non vi è alcuna unità B:

▼ Dispositivi e unità (3)



Infatti, il valore di ritorno dell'API GetLogicalDrives per me è stato 0x1C, che corrisponde alla maschera di bit i cui bit pari a 1 sono solo quelli nelle posizioni 2, 3 e 4.

A ogni iterazione del loop, il valore di EBP-0x14 viene incrementato di 1: questo comporta banalmente che, ogni volta che viene effettuato il controllo `if (maschera_di_bit & 1 << ((EBP-0x14) & 0x1f)) != 0`, si verifica la presenza di un'unità disco differente (dopo l'unità B si controllo dunque l'unità C, poi l'unità D, e così via).

Possiamo dunque concludere che la funzione FUN_0046c640 ottiene alcune informazioni riguardanti le unità disco presenti all'interno del sistema in cui il malware è in esecuzione. Perciò, ridenominiamo tale funzione `get_logical_drives_info`.

Adesso è il turno di FUN_00476b50: la prima cosa che fa questa funzione invocare per quattro volte di seguito FUN_00474820:


```

34 FUN_00474820(s_SeDebugPrivilege_0047ef28);
35 FUN_00474820(s_SeTakeOwnershipPrivilege_0047ef08);
36 FUN_00474820(s_SeBackupPrivilege_0047eef0);
37 FUN_00474820(s_SeRestorePrivilege_0047eed8);

```

Già dal parametro che FUN_00474820 accetta in input si intuisce bene che si tratta di una funzione che si occupa dell'escalation dei privilegi. E infatti si presenta così:

```

4  hint __cdecl FUN_00474820(LPCSTR param_1)
5
6  {
7      HANDLE pvVar1;
8      uint uVar2;
9      BOOL BVar3;
10     DWORD DVar4;
11     uint uVar5;
12     undefined4 uVar6;
13     HANDLE *ppvVar7;
14     HANDLE local_8;
15
16     ppvVar7 = &local_8;
17     uVar6 = 0x28;
18     uVar5 = 0;
19     pvVar1 = GetCurrentProcess();
20     uVar2 = (*_OpenProcessToken)(pvVar1,uVar6,ppvVar7);
21     if (uVar2 != 0) {
22         BVar3 = LookupPrivilegeValueA((LPCSTR)0x0,param_1,(PLUID)&DAT_0048206c);
23         if (BVar3 != 0) {
24             BVar3 = AdjustTokenPrivileges
25                 (local_8,0,(PTOKEN_PRIVILEGES)&DAT_00482068,0,(PTOKEN_PRIVILEGES)0x0,
26                 (PDWORD)0x0);
27             if (BVar3 != 0) {
28                 DVar4 = GetLastError();
29                 uVar5 = 0;
30                 if (DVar4 == 0) {
31                     uVar5 = 1;
32                 }
33             }
34         }
35         uVar2 = CloseHandle(local_8);
36     }
37     return uVar2 & 0xffffffff00 | uVar5;

```

Le API GetCurrentProcess, OpenProcessToken, LookupPrivilegeValueA e AdjustTokenPrivileges non lasciano alcun dubbio sugli scopi di questa funzione, che nel frattempo ho ridenominato privileges_escalation. Concentriamoci sul parametro che, a ogni invocazione, privileges_escalation prende in input, in modo tale che comprendiamo quali sono i privilegi che il malware vuole ottenere:

- **SeDebugPrivilege**: acquisizione dei privilegi a livello di sistema.
- **SeTakeOwnershipPrivilege**: acquisizione della proprietà dei file.
- **SeBackupPrivilege**: acquisizione della possibilità di effettuare il backup di file e directory.
- **SeRestorePrivilege**: acquisizione della possibilità di ripristinare file e directory.

A seguito dell'escalation dei privilegi, FUN_00466b50 invoca GetModuleHandleA, che recupera un module handle per il modulo specificato come parametro (che in questo caso è ntdll.dll); tale module handle viene salvato nella variabile globale DAT_004831c8, che ho ridenominato subito NTDLL_MODULE.

Se la chiamata a GetModuleHandleA è andata a buon fine, ed è questo il caso, allora si entra in un ramo if in cui vengono invocate alcune GetProcAddress, che restituiscono gli indirizzi delle API specificate; tali indirizzi vengono poi salvati in delle variabili locali:

```

43  NTDLL_MODULE = hModule;
44  if (hModule != (HMODULE)0x0) {
45      local_58 = 0x7551744e;
46      local_54 = 0x53797265;
47      local_50 = 0x65747379;
48      local_4c = 0x666e496d;
49      local_48 = 0x616d726f;
50      local_44 = 0x6e6f6974;
51      local_40 = 0;
52      _DAT_004831f0 = GetProcAddress(hModule, (LPCSTR)&local_58);
53      local_3c = 0x7544744e;
54      local_38 = 0x63696c70;
55      local_34 = 0x4f657461;
56      local_30 = 0x63656a62;
57      local_2c = 0x74;
58      local_2b = 0;
59      _DAT_004831e8 = GetProcAddress(NTDLL_MODULE, (LPCSTR)&local_3c);
60      local_1c = 0x7551744e;
61      local_18 = (undefined **)0x4f797265;
62      local_14 = 0x63656a62;
63      local_10 = 0x74;
64      local_f = 0;
65      hModule = (HMODULE)GetProcAddress(NTDLL_MODULE, (LPCSTR)&local_1c);
66      _DAT_004831ec = hModule;
67  }

```

Con l'ausilio di OllyDbg, ho potuto appurare che:

- In DAT_004831f0 viene salvato l'indirizzo della funzione NtQuerySystemInformation.
- In DAT_004831e8 viene salvato l'indirizzo della funzione NtDuplicateObject.
- In DAT_004831ec viene salvato l'indirizzo della funzione NtQueryObject.

Dopodiché, se tutte e tre le chiamate a GetProcAddress sono andate a buon fine, viene invocata una CreateThread coi seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPSECURITY_ATTRIBUTES	lpThreadAttributes	Puntatore a una struttura SECURITY_ATTRIBUTES che determina se l'handle al thread che viene creato potrà essere ereditato dai processi figli (se è pari a NULL, l'handle non potrà essere ereditato).	NULL
2	SIZE_T	dwStackSize	Dimensione iniziale dello stack (0 = default).	0

3	LPTHREAD_START_ROUTINE	lpStartAddress	Puntatore alla funzione che dovrà essere eseguita dal thread.	&LAB_004768d0
4	__drv_aliasesMem LPVOID	lpParameter	Puntatore a una variabile che verrà passata al thread.	NULL
5	DWORD	dwCreationFlags	Flag che controllano la creazione del thread.	0 (= il thread eseguirà immediatamente)
6	LPDWORD	lpThreadId	Puntatore a una variabile che riceverà l'identificativo del thread.	[out] 0x0019fcc4

Se l'invocazione a CreateThread non va a buon fine (ovvero se non viene restituito l'handle al thread che viene creato), allora viene lanciata un'eccezione. Comunque sia, FUN_00476b50 termina chiudendo l'handle al thread creato tramite una CloseHandle e restituisce il valore di ritorno di quest'ultima API. In conclusione, ho rinominato FUN_00476b50 in privileges_and_thread.

Proviamo a vedere anche la funzione eseguita dal thread creato, che ho chiamato thread_fun1: in poche parole contiene un ciclo infinito che, ogni due secondi, invoca FUN_00476150.

FUN_00476150 inizialmente chiama GetModuleFileNameA, un'API che recupera il percorso completo per il file contenente il modulo specificato e che accetta i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HMODULE	hModule	Handle al modulo il cui path viene richiesto.	NTDLL_MODULE
2	LPSTR	lpFilename	Puntatore a un buffer che riceverà il percorso completo del modulo.	[out] 0x0258fd28
3	DWORD	nSize	Dimensione del buffer lpFilename.	520

Se quest'API ha successo, si passa a invocare una CreateFileA, che prende i seguenti parametri in input:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPCSTR	lpFilename	Nome del file da creare o da aprire.	"C:\Windows\SYSTEM32\ntdll.dll"
2	DWORD	dwDesiredAccess	L'accesso richiesto al file.	GENERIC_READ
3	DWORD	dwShareMode	La modalità di condivisione richiesta del file.	FILE_SHARE_READ
4	LPSECURITY_ATTRIBUTES	lpSecurityAttributes	Puntatore a una struttura SECURITY_ATTRIBUTES.	NULL
5	DWORD	dwCreationDisposition	Azione da intraprendere sul file.	OPEN_EXISTING
6	DWORD	dwFlagsAndAttributes	Flag e attributi del file.	0
7	HANDLE	hTemplateFile	Se il file viene creato, è l'handle a un file modello che fornisce gli attributi estesi per il file.	NULL

In caso di successo, viene invocata FUN_004749c0 che serve ad allocare dinamicamente della memoria. Se anche questa va a buon fine, viene recuperato l'identificatore del processo tramite GetCurrentProcessId.

Successivamente si hanno due loop annidati in cui vengono effettuate alcune operazioni e vengono chiamate altre funzioni. Tra queste ultime, ce ne sono tre su cui posso spendere un paio di parole:

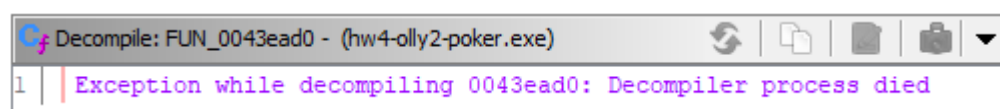
- FUN_004750e0 contiene, in particolar modo, delle CreateEventA, una SetEvent e una CreateThread. Il thread che viene creato qui, a sua volta, esegue un loop infinito contenente un'ulteriore chiamata a SetEvent.

- FUN_00475540 contiene una GetLogicalDriveStringsW, che riempie un buffer con delle stringhe che specificano le unità valide del sistema, e una QueryDosDeviceW, che recupera informazioni sui nomi dei dispositivi MS-DOS (che sono collegamenti simbolici nel gestore oggetti con un nome nella forma "\\DosDevices\\DosDeviceName").

- FUN_0043ead0 dovrebbe essere una funzione grande e complessa in una maniera spropositata, tant'è vero che, secondo Ghidra, contiene una quantità di variabili automatiche mai vista prima:

```
undefined4      Stack[-0x2d4... local_2d4c
undefined4      Stack[-0x2d5... local_2d50
undefined4      Stack[-0x2d5... local_2d54
undefined4      Stack[-0x2d5... local_2d58
undefined4      Stack[-0x2d5... local_2d5c
undefined4      Stack[-0x2d6... local_2d60
undefined4      Stack[-0x2d6... local_2d64
undefined4      Stack[-0x2d6... local_2d68
undefined4      Stack[-0x2d6... local_2d6c
undefined4      Stack[-0x2d7... local_2d70
undefined4      Stack[-0x2d7... local_2d74
undefined4      Stack[-0x2d7... local_2d78
undefined4      Stack[-0x2d7... local_2d7c
undefined4      Stack[-0x2d8... local_2d80
undefined4      Stack[-0x2d8... local_2d84
undefined4      Stack[-0x2d8... local_2d88
undefined4      Stack[-0x2d8... local_2d8c
undefined4      Stack[-0x2d9... local_2d90
undefined4      Stack[-0x2d9... local_2d94
undefined4      Stack[-0x2d9... local_2d98
undefined4      Stack[-0x2d9... local_2d9c
undefined4      Stack[-0x2da... local_2da0
undefined4      Stack[-0x2da... local_2da4
undefined4      Stack[-0x2da... local_2da8
undefined4      Stack[-0x2da... local_2dac
undefined4      Stack[-0x2db... local_2db0
undefined4      Stack[-0x2db... local_2db4
undefined4      Stack[-0x2db... local_2db8
undefined4      Stack[-0x2db... local_2dbc
undefined4      Stack[-0x2dc... local_2dc0
undefined4      Stack[-0x2dc... local_2dc4
undefined4      Stack[-0x2dc... local_2dc8
undefined4      Stack[-0x2dc... local_2dcc
FUN_0043ead0
```

Tra l'altro, il decompilatore di Ghidra è andato in crash:



Poiché, a maggior ragione senza il codice decompilato, sarebbe troppo oneroso provare ad analizzare FUN_0043ead0, ci ritroviamo costretti a saltare questa funzione a pie' pari. Oltre a ridenominare tale funzione too_long, ciò che si può fare è eseguirla col debugger: sembra che non sia accaduto nulla di rilevante, per cui possiamo mettere too_long da parte.

Un'osservazione che può valere la pena fare è che il nostro thread, insieme a quello che crea all'interno di FUN_004750e0, invoca tutte e tre le API della libreria ntdll.dll ottenute da GetProcAddress all'interno della funzione privileges_and_thread: potrebbe essere anche questo il motivo per cui all'inizio il nostro thread apre il file relativo a ntdll.dll mediante una CreateFileA.

Ora spostiamo nuovamente la nostra attenzione sul main thread del processo e, in particolare, torniamo sulla funzione ransomware_clue. Qui abbiamo appena oltrepassato l'invocazione a privileges_and_thread: subito dopo c'è un ciclo for che inizia con un'altra CreateThread che, se ci ricordiamo, è la CreateThread che volevamo raggiungere, perché dovrebbe condurci alla fatidica FUN_004396e0.

Intanto vediamo con quali parametri stavolta è stata invocata tale API:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPSECURITY_ATTRIBUTES	lpThreadAttributes	Puntatore a una struttura SECURITY_ATTRIBUTES che determina se l'handle al thread che viene creato potrà essere ereditato dai processi figli (se è pari a NULL, l'handle non potrà essere ereditato).	NULL
2	SIZE_T	dwStackSize	Dimensione iniziale dello stack (0 = default).	0
3	LPTHREAD_START_ROUTINE	lpStartAddress	Puntatore alla funzione che dovrà essere eseguita dal thread.	&LAB_00429b40
4	__drv_aliasesMem LPVOID	lpParameter	Puntatore a una variabile che verrà passata al thread.	0x02201308 (indirizzo che contiene la stringa "c:")
5	DWORD	dwCreationFlags	Flag che controllano la creazione del thread.	0 (= il thread eseguirà immediatamente)
6	LPDWORD	lpThreadId	Puntatore a una variabile che riceverà l'identificativo del thread.	[out] 0x0019feac

Di fatto, nel nostro caso specifico, questo specifico thread è l'unico a essere creato all'interno del loop, poiché viene eseguita un'unica iterazione nel ciclo for stesso. Infatti, come argomenti del ciclo abbiamo:

```
for (lpParameter = *(LPVOID *) (unaff_EBP + -0xe0);
    lpParameter != *(LPVOID *) (unaff_EBP + -0xdc);
    lpParameter = (LPVOID) ((int) lpParameter + 0x1c)) {
```

EBP-0xe0 è pari a 0x0019fdcf e punta all'indirizzo 0x02201308, che contiene la stringa "c:". Dunque, alla prima iterazione, lpParameter è pari all'indirizzo 0x02201308 e punta alla stringa "c:".

Per la seconda iterazione, lpParameter viene incrementato di 0x1c e diviene uguale a 0x02201324, che è l'indirizzo puntato da 0x19fe00, che è proprio pari a EBP-0xdc. Ciò sancisce l'uscita dal loop dopo la prima

iterazione.

La stringa "c:" che viene passata al thread che viene creato mi ha fatto pensare che, con ogni probabilità, viene lanciato un thread per ogni unità disco interessante che è stata trovata precedentemente in `get_logical_drives_info`. Ricordiamo che sulla nostra macchina virtuale sono state rilevate le unità C, D, E, dove D ed E certamente non contengono i file dell'utente; le informazioni sulla natura delle unità disco sono state estrapolate sempre in `get_logical_drives_info`, come avevamo potuto vedere.

Andiamo ora ad analizzare la funzione che parte dall'indirizzo 0x00429b40, che è la funzione in cui il nuovo thread inizia la sua esecuzione, e chiamiamola `thread_fun2`.

`Thread_fun2` esordisce recuperando lo pseudo-handle al thread tramite una `GetCurrentThread`, e poi invoca `SetThreadPriority`, che è un'API atta a impostare un determinato valore di priorità per il thread; il suo valore di ritorno è nullo in caso di fallimento, non nullo altrimenti; i parametri che prende in input sono:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HANDLE	hThread	(Pseudo-)handle al thread la cui priorità deve essere impostata.	<code>GetCurrentThread()</code>
2	int	nPriority	Valore della priorità per il thread.	<code>THREAD_PRIORITY_LOWEST</code>

Dopodiché, a partire da questo `SetThreadPriority`, ho eseguito alcune istruzioni su OllyDbg mediante il comando `Step Into` e mi sono ritrovato direttamente all'invocazione della funzione `FUN_0046bf70`. L'unica cosa che mi ha attirato all'interno di `FUN_0046bf70` è la funzione `FUN_0046b310`: vediamo la insieme.

Per rendere più lineare il lavoro, all'interno di `FUN_0046b310` ho contrassegnato come "skipped" le funzioni che non sembrano svolgere un lavoro di particolare rilievo.

Comunque sia, verso l'inizio, `FUN_0046b310` invoca l'API `FindFirstFileW`, che cerca all'interno di una directory un file o una sottodirectory con un nome specifico; il suo valore di ritorno è un handle di ricerca in caso di successo, `INVALID_HANDLE_VALUE` altrimenti, e i parametri che accetta in input sono i seguenti:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPCWSTR	lpFileName	Pathname del file o della directory cercata.	"c:*"
2	LPWIN32_FIND_DATAW	lpFindFileData	Puntatore a una struttura <code>WIN32_FIND_DATA</code> che riceverà le informazioni riguardanti il file o la directory trovata.	[out] 0x025efc10

Se l'invocazione a `FindFirstFileW` va a buon fine viene chiamata `FUN_00462b10`.

`FUN_00462b10` inizia la sua esecuzione con un ciclo `while` contenente una chiamata a `GetFileSecurityW`, che è un'API che ottiene determinate informazioni sulla sicurezza di un file o directory; il suo valore di ritorno è 0 in caso di fallimento, un valore differente altrimenti; i suoi parametri sono:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPCWSTR	lpFileName	Puntatore a una stringa che specifica il file o directory per cui le informazioni di sicurezza verranno recuperate.	"c:"
2	SECURITY_INFORMATION	RequestedInformation	Valore che identifica le informazioni di sicurezza richieste.	7*
3	PSECURITY_DESCRIPTOR	pSecurityDescriptor	Puntatore a un buffer che riceverà una copia del security descriptor	[out] NULL

			dell'oggetto specificato dal parametro lpFileName.	
4	DWORD	nLength	Dimensione in byte del buffer puntato da pSecurityDescriptor.	0
5	LPDWORD	lpLengthNeeded	Puntatore a una variabile che riceverà il numero di byte necessari per memorizzare il security descriptor.	[out] 0x025efbf0

*OWNER_SECURITY_INFORMATION || GROUP_SECURITY_INFORMATION || DACL_SECURITY_INFORMATION

C'è da specificare che i parametri qui riportati sono relativi alla prima iterazione nel ciclo while. Si esce dal loop nel momento in cui l'API non genera l'errore ERROR_INSUFFICIENT_BUFFER; se invece tale errore si presenta, viene allocata dinamicamente della memoria con una malloc e viene assegnata al parametro pSecurityDescriptor; la quantità di memoria da allocare viene presa dal valore all'indirizzo 0x025efbf0.

Una volta uscita dal loop, FUN_00462b10 invoca una GetCurrentThread per recuperare lo pseudo-handle relativo al thread chiamante e, successivamente una OpenThreadToken, un'API che apre l'access token associato al thread; anche OpenThreadToken restituisce un valore diverso da zero in caso di successo; inoltre, accetta i seguenti parametri in input:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HANDLE	ThreadHandle	Pseudo-handle al thread il cui access token deve essere aperto.	GetCurrentThread()
2	DWORD	DesiredAccess	Access mask che specifica i tipi di accesso richiesti all'access token.	STANDARD_RIGHTS_READ TOKEN_DUPLICATE TOKEN_IMPERSONATE TOKEN_QUERY
3	BOOL	OpenAsSelf	TRUE se il controllo di accesso deve essere eseguito rispetto al security context a livello di processo. FALSE se il controllo di accesso deve essere eseguito rispetto al security context a livello di thread.	TRUE
4	PHANDLE	TokenHandle	Puntatore a una variabile che riceverà l'handle all'access token.	[out] 0x025efbf8

Poiché, nel nostro caso, questa chiamata fallisce, viene eseguita anche una OpenProcessToken, che ha uno scopo analogo a OpenThreadToken, con la differenza che agisce a livello di processo anziché a livello di thread; tra l'altro questa API la conosciamo già dal contesto dell'escalation dei privilegi avvenuta tramite le funzioni privileges_escalation. Comunque sia, i permessi d'accesso richiesti (DesiredAccess) all'interno di OpenProcessToken sono i medesimi di quelli richiesti in OpenThreadToken.

L'operazione successiva è l'invocazione di DuplicateToken, un'API che duplica un access token esistente, creandone uno nuovo; restituisce un valore diverso da zero solo in caso di successo e prende i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HANDLE	ExistingTokenHandle	Handle a un access token esistente.	0x348
2	SECURITY_IMPERSONATION_LEVEL	ImpersonationLevel	Livello di rappresentazione del nuovo token.	SecurityImpersonation

3	PHANDLE	DuplicateTokenHandle	Puntatore a una variabile che riceverà un handle al token duplicato.	[out] 0x025efbf4
---	---------	----------------------	--	------------------

Se DuplicateToken ha successo, viene invocata anche MapGenericMask, che mappa i diritti di accesso generici in un'access mask su diritti d'accesso specifici. In particolare, i diritti d'accesso impostati all'interno della struttura GENERIC_MAPPING passata come parametro all'API sono i seguenti:

```
local_2c.GenericRead = 0x120089;
local_2c.GenericWrite = 0x120116;
local_2c.GenericExecute = 0x1200a0;
local_2c.GenericAll = 0x1f01ff;
```

L'ultima API particolare invocata all'interno di FUN_00462b10 è AccessCheck, che determina se un security descriptor concede un determinato insieme di diritti di accesso al client identificato dall'access token; restituisce un valore non nullo solo in caso di successo e accetta i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	PSECURITY_DESCRIPTOR	pSecurityDescriptor	Puntatore a una struttura SECURITY_DESCRIPTOR rispetto a cui viene verificato l'accesso.	La struttura pSecurityDescriptor usata per la (seconda) invocazione a GetFileSecurityW
2	HANDLE	ClientToken	Handle a un impersonation token che rappresenta il client che sta tentando di ottenere l'accesso.	0x2e4
3	DWORD	DesiredAccess	Access mask che specifica i diritti di accesso da verificare.	0x00120116 (indirizzo fornito dall'API MapGenericMask)
4	PGENERIC_MAPPING	GenericMapping	Puntatore alla struttura GENERIC_MAPPING associata all'oggetto al cui accesso deve essere verificato.	0x025efbd4
5	PPRIVILEGE_SET	PrivilegeSet	Puntatore a una struttura PRIVILEGE_SET che riceverà i privilegi usati per eseguire la validazione dell'accesso.	[out] 0x025efbc0
6	LPDWORD	PrivilegeSetLength	Puntatore alla dimensione in byte del buffer puntato da PrivilegeSet.	0x025efbe4 (che punta al valore 0x14 = 20)
7	LPDWORD	GrantedAccess	Puntatore a un access mask che riceverà i diritti di accesso consentiti.	[out] 0x025efbe8
8	LPBOOL	AccessStatus	Puntatore a un booleano il cui valore sarà: - TRUE se il security descriptor consente i diritti di accesso richiesti. - FALSE altrimenti.	[out] 0x025efbec

Se l'invocazione di quest'ultima API ha successo, allora FUN_00462b10 restituisce il booleano puntato dall'indirizzo AccessStatus; altrimenti, restituisce FALSE.

In poche parole, FUN_00462b10 si occupa di gestire i privilegi e i diritti di accesso per il thread chiamante e, se tutto va secondo i piani, restituisce TRUE, altrimenti restituisce FALSE. Per questo motivo, l'ho ridenominata privileges_and_access_rights.

Tornando a FUN_0046b310, a seguito della chiamata a privileges_and_access_rights, entra in un ciclo do-while in cui esegue un po' di operazioni e invoca l'API FindNextFileW, che non fa altro che continuare la ricerca di file / directory intrapresa nella precedente chiamata a FindFirstFileW.

Nel momento in cui FindNextFileW restituisce zero (ovvero fallisce), si esce dal loop e si invoca FindClose, un'API che chiude l'handle di ricerca di file aperto dalla FindFirstFileW. Dopodiché FUN_0046b310 termina.

Riassumendo, FUN_0046b310 invoca la funzione privileges_and_access_rights e cerca i file o le sottodirectory che si trovano all'interno della directory con percorso "c:" della nostra macchina virtuale. Ho dunque ridenominato tale funzione look_for_files.

Tra l'altro, per comodità, ho ridenominato FUN_0046bf70 in caller_of_look_for_files.

È il momento ora della funzione FUN_00427ac0, invocata da thread_fun2, che dovrebbe essere una funzione di gran lunga più interessante.

FUN_00427ac0 per prima cosa, oltre a invocare la solita append_SEH_chain, chiama FUN_0041a310. Quest'ultima funzione, finalmente, contiene qualcuna delle API della famiglia Crypt e, in particolare, CryptAcquireContextA (invocata da caller_of_CryptAcquireContextA) e CryptReleaseContext.

Per quanto riguarda CryptAcquireContextA, viene invocata con gli stessi parametri della prima volta che l'abbiamo incontrata:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HCRYPTPROV*	phProv	Indirizzo in cui verrà copiato il puntatore al CSP.	[out] this
2	LPCSTR	szContainer	Nome del key container.	NULL
3	LPCSTR	szProvider	Nome del CSP da usare.	NULL
4	DWORD	dwProvType	Tipo di provider da acquisire.	24
5	DWORD	dwFlags	Flag	0xF0000000

La CryptReleaseContext, che rilascia l'handle al CSP (cryptographic service provider, che ricordiamo essere un modulo software che esegue algoritmi di crittografia per l'autenticazione e la cifratura), viene invocata se tale handle rimane pari a NULL a seguito della chiamata a CryptAcquireContextA.

Il secondo passo a mio avviso cruciale per FUN_00427ac0 è l'invocazione di FUN_00413be0, che è una funzione piuttosto complessa appartenente a un ciclo for (per cui può essere chiamata più volte): cerchiamo, come al solito, di intuirne le funzionalità tramite le API invocate.

La prima tra queste è GetFileAttributesExW, che recupera gli attributi per un file o una directory specificata; il suo valore di ritorno è diverso da zero solo in caso di successo e i parametri che prende in input sono i seguenti:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPCWSTR	lpFileName	Nome del file o directory.	"c:\Users\...\45-mckenzie.pptx"
2	GET_FILEEX_INFO_LEVELS	flInfoLevelId	Classe delle informazioni sugli attributi da recuperare.	GetFileExInfoStandard

3	LPVOID	lpFileInformation	Puntatore a un buffer che riceverà le informazioni sugli attributi.	[out] 0x263f9d0 (= EBP-0x2d0)
---	--------	-------------------	---	----------------------------------

*È con ogni probabilità il path di uno dei file che verranno cifrati dal ransomware.

Dopo la chiamata a GetFileAttributesExW, lpFileInformation ha il seguente contenuto:

Address	Hex	dump
0263F9D0	20 20 00 00	CE 28 66 57
0263F9D8	11 03 08 01	9F 31 6A 57
0263F9E0	11 03 08 01	9F 31 6A 57
0263F9E8	11 03 08 01	00 00 00 00
0263F9F0	4D EE 01 00	18 00 00 00

Questi byte, così come sono, non ci dicono in granché, ma comunque ce li teniamo da parte.

Più avanti nella funzione c'è anche una SetFileAttributesW, che però si trova all'interno di un blocco if che non sembra mai essere preso.

Piuttosto, viene chiamata la funzione FUN_00411da0, che contiene una CreateFileW a cui vengono passati i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPCWSTR	lpFilename	Nome del file da creare o da aprire.	"c:\Users\...\45-mckenzie.pptx"
2	DWORD	dwDesiredAccess	L'accesso richiesto al file.	GENERIC_READ GENERIC_WRITE
3	DWORD	dwShareMode	La modalità di condivisione richiesta del file.	FILE_SHARE_DELETE
4	LPSECURITY_ATTRIBUTES	lpSecurityAttributes	Puntatore a una struttura SECURITY_ATTRIBUTES.	NULL
5	DWORD	dwCreationDisposition	Azione da intraprendere sul file.	OPEN_EXISTING
6	DWORD	dwFlagsAndAttributes	Flag e attributi del file.	0
7	HANDLE	hTemplateFile	Se il file viene creato, è l'handle a un file modello che fornisce gli attributi estesi per il file.	NULL

All'interno di FUN_00413be0, segue un'invocazione a MoveFileExW, che è un'API che sposta un file o directory insieme ai suoi children; restituisce un valore diverso da zero in caso di successo, e ha i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPCWSTR	lpExistingFileName	Nome corrente del file o directory.	"c:\Users\...\45-mckenzie.pptx"
2	LPCWSTR	lpNewFileName	Nuovo nome del file o directory.	"c:\Users\...\86RB6EG6-BUPI-X9AI-FFA68F35-B8A21FD0CBCB.asasin"
3	DWORD	dwFlags	Flag.	REPLACE_EXISTING MOVEFILE_WRITE_THROUGH

Appare così evidente che MoveFileExW è l'API che si occupa di ridenominare i file in modo che vengano anche convertiti nel formato ".asasin".

Successivamente viene invocata FUN_00410d90, che contiene una CryptGenRandom, che è un'API che riempie un buffer con byte crittograficamente casuali, restituisce TRUE in caso di successo (FALSE altrimenti) e accetta i seguenti parametri in input:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HCRYPTPROV	hProv	Handle al CSP creato precedentemente da CryptAcquireContext.	0x00507bf8
2	DWORD	dwLen	Numero di byte di dati casuali da generare.	16
3	BYTE	*pbBuffer	Buffer che riceverà i dati restituiti.	[out] 0x025df6c4

I dati casuali generati da CryptGenRandom sono:

```
025DF6C4 | 2633B633
025DF6C8 | 2467AB20
025DF6CC | 7B2552F4
025DF6D0 | A0BB0320
```

Dopodiché abbiamo l'invocazione della funzione FUN_00410fe0 che invece contiene CryptEncrypt, un'API che cifra i dati; restituisce TRUE in caso di successo (FALSE altrimenti), e prende i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HCRYPTKEY	hKey	Handle alla chiave di cifratura.	0x004d0b98
2	HCRYPTHASH	hHash	Handle all'oggetto hash utilizzato per eseguire l'hashing di messaggi o chiavi di sessione.	NULL
3	BOOL	Final	Booleano che specifica se questa è l'ultima sezione nella serie che sta per essere criptata.	FALSE
4	DWORD	dwFlags	Flag.	0
5	BYTE *	pbData	Puntatore a un buffer che contiene il plaintext che deve essere cifrato.	0x25df6c4
6	DWORD *	pdwDataLen	Puntatore a un DWORD che, inizialmente, contiene la lunghezza in byte del plaintext; dopo l'esecuzione dell'API, conterrà la lunghezza in byte del testo cifrato.	0x25df574
7	DWORD	dwBufLen	Dimensione totale del buffer pbData.	160

- Contenuto di pbData prima della cifratura:

Address	Hex dump	ASCII
025DF6C4	33 B6 33 26 20 AB 67 24	3A3% %g\$
025DF6CC	F4 52 25 7B 20 03 BB A0	qR%C ♡! à

- Contenuto di pbData dopo la cifratura:

Address	Hex dump	ASCII
025DF6C4	FC EA 4A 9E 75 21 18 C4	?0Jxu†↑
025DF6CC	F5 0A 6D A1 77 82 6B 20	S.miwek
025DF6D4	BF 0A D3 21 13 A6 F1 7C	7.É! ±!
025DF6DC	55 27 7D BB 21 AC FA A7	U".q!%."?
025DF6E4	18 AD 98 37 77 E7 22 DC	†+guwE"■
025DF6EC	59 C2 03 D4 B3 E2 04 A9	Y-#E 0#9
025DF6F4	13 1C F2 71 14 ED 52 90	ll-q!YRÉ
025DF6FC	99 80 01 C1 AB CF 8D AF	0C0-!%S>
025DF704	77 A8 D2 87 06 6C 40 2D	w2EÇ@l@-
025DF70C	91 C6 B2 16 41 BC 49 78	æ33-FFIx
025DF714	FA 2C 60 EB 2C 7A D6 CE	. "ü, z i††
025DF71C	12 E0 AE C6 26 46 37 CE	#0<<S&F7†
025DF724	77 02 48 67 45 31 9E 48	w0HgE1×H
025DF72C	E6 CB B8 79 2C 6F C7 F9	y††0y, o&--
025DF734	D2 25 71 AF B2 93 15 F1	E%q>33S±
025DF73C	70 85 B0 5E 53 02 24 83	p33-^S0\$ã
025DF744	CE FA A3 A0 8B 94 D7 22	††.üá!oi"†
025DF74C	79 CC 85 6B D7 CE 3E 01	y††ák I††>0
025DF754	67 D6 1C 57 D5 CA D4 34	gILW!±E4
025DF75C	5B 77 70 21 FA 17 CE 86	Lwp†!·\$††ã

Dopodiché, all'interno di FUN_00413be0, viene effettuato un controllo sulla lunghezza del testo cifrato presente in pdwDataLen: se essa è pari a 160 byte, si entra in un ramo if, ed effettivamente le cose vanno così.

All'interno di questo ramo if c'è subito una funzione molto particolare, FUN_00473830, che, tra i vari parametri, prende in input anche due locazioni di memoria, di cui una contiene il vecchio nome del file che era stato preso di mira nelle API precedenti ("45-mckenzie.pptx"):

Address	Hex dump	ASCII
025DF590	79 FF D2 69 49 31 30 19	y ëiI10+
025DF598	F7 B1 7E F2 8E 37 2A 76	·¸"=À7*¸
025DF5A0	E2 1A EA 70 AB 2B DA 69	0+Ûp%+ri
025DF5A8	5C 9A A4 9B D2 AD 8E ED	\0K¸E+AY
025DF5B0	75 03 BF C5 DE 28 65 AC	u¸+i(e%¸
025DF5B8	82 B2 C1 37 50 1F 4F DA	e¸+7P*0r
025DF5C0	B1 87 E8 96 6F AF 8D 3A	¸¸p¸o>i:
025DF5C8	ED 1D 4C 0D BD 02 03 D7	¸+L.c¸¸i
025DF5D0	CE FC E6 EC A1 53 6B D6	¸¸p¸iSk i
025DF5D8	4C 4E 27 DB F1 4C 24 0C	LN'¸:L¸.
025DF5E0	F7 CA 18 4D 56 99 73 9B	·+¸MU0s¸
025DF5E8	1A D7 54 40 EB 9B 70 4C	+iT@Û¸pL
025DF5F0	C3 9B 31 A4 95 02 42 3F	¸¸1K00B?
025DF5F8	8F D5 16 7F 64 4E 66 33	A' .¸dNf3
025DF600	AC A8 F2 E7 39 AA B0 D8	%¸=¸9-¸¸i
025DF608	B6 7F A6 A7 D2 31 C0 94	¸¸00E1¸0
025DF610	EB 12 D0 52 D2 B8 60 8A	Û+¸DR0
025DF618	64 C7 C6 2D B6 F6 06 B9	dA¸-A+¸i
025DF620	B2 7D 86 1C 60 C5 E6 96	¸¸)¸L A¸
025DF628	04 02 20 BB B2 F4 26 02	¸¸0 ¸¸¸¸¸0
025DF630	3B 8A F1 2B 5B 4F 17 BD	;e+L0¸¸
025DF638	5F 4D 37 06 ED B9 11 04	_M7+¸i¸¸
025DF640	70 00 00 00 00 00 55 00	¸

Address	Hex dump	ASCII
025DF7C4	2A A1 1B D4 34 00 35 00	*i+E4.5.
025DF7CC	2D 00 6D 00 63 00 6B 00	-.m.c.k.
025DF7D4	65 00 6E 00 7A 00 69 00	e.n.z.i.
025DF7DC	65 00 2E 00 70 00 70 00	e...p.p.
025DF7E4	74 00 78 00 00 00 00 00	t.x.....

La particolarità di FUN_00473830 risiede nel fatto che contiene un'innumerabile quantità di chiamate all'istruzione AESENC, nonché alcune chiamate ad altre istruzioni curiose, come AESENCLAST, PADDQ, PSUBQ e PSHUFB.

Guardando sulla documentazione della Intel, ho visto che:

- PSHUFB esegue uno shuffle (mescolamento) di byte nell'operando destinazione a seconda del cosiddetto shuffle control mask nell'operando sorgente.
- PADDQ esegue una somma SIMD (Single Instruction Multiple Data) tra interi compressi.
- PSUBQ esegue una sottrazione SIMD tra interi compressi.
- AESENC esegue una singola iterazione dell'algoritmo di cifratura AES.
- AESENCLAST esegue l'ultima iterazione dell'algoritmo di cifratura AES.

Per farla breve, FUN_00473830 si occupa di cifratura AES e, in particolare, intacca il buffer che conteneva il nome del file, che diventa:

Address	Hex dump	ASCII
025DF7C0	66 CD 09 5C 73 AA 1A 46	f=.\s-+F
025DF7C8	1C 55 97 1F EF 05 37 FD	LÜü?¸7²
025DF7D0	5B F9 A3 5D AA D6 E3 76	[~ü]-i0v
025DF7D8	B5 70 94 40 2B 93 A1 35	¸p00+0i5
025DF7E0	A9 9A 40 C8 0A 0A 9B 64	000¸.¸.¸d
025DF7E8	22 A5 B0 96 4C F7 B2 E9	"Kc¸L+¸¸ü
025DF7F0	DC B1 77 A6 55 8B 20 47	¸¸w0U i G
025DF7F8	B8 A0 8C 71 AF F1 5D FD	0¸i¸q>+¸j²
025DF800	07 12 86 A8 FF 00 17 3D	+¸¸¸¸.¸=
025DF808	B7 79 E7 FF E1 F2 16 65	¸y¸¸¸=¸.e
025DF810	52 4A D1 54 B7 6B 9B 09	RJ0TAKÿ.
025DF818	F5 E6 BA 26 6B 44 22 41	¸p¸¸¸kD"¸A
025DF820	CD C1 E0 3E 14 7C 32 44	==¸0>¸i!2D
025DF828	C9 61 67 6E 65 5B 83 AB	¸¸agneX¸¸¸k
025DF830	4A 4C 57 8D DF 1E 08 54	JLÜi¸=A¸T
025DF838	C8 9C 28 E8 D0 84 41 33	¸¸E(¸¸¸¸A3
025DF840	AB 30 81 03 21 59 CC CA	¸¸00¸+¸V¸¸¸=
025DF848	D2 BA 79 28 FF 75 D3 B6	ë¸¸y(uE¸A
025DF850	04 13 2D F9 08 5C D3 B1	¸¸!-¸¸¸¸\E¸¸
025DF858	EE 54 17 91 64 B5 34 3C	"T¸¸edA4<
025DF860	7F E4 32 06 14 0F 54 E5	0¸2¸¸¸¸T0
025DF868	69 4D 0E 81 4C 96 94 30	iM¸¸üLü00
025DF870	DE 82 0B D1 3D 0F 9B E9	¸¸¸¸00+¸¸¸¸ü

Ho così rinominato la funzione encrypt_AES.

Più avanti in FUN_00413be0 c'è un ciclo while in cui a grandi linee:

- 1) Viene invocata FUN_004106e0, che contiene una ReadFile (per cui l'ho rinominata caller_of_ReadFile).
- 2) Viene invocata nuovamente encrypt_AES.
- 3) Eventualmente viene invocata FUN_004109f0, che contiene una SetFilePointer (per cui l'ho rinominata

caller_of_SetFilePointer).

4) Viene invocata FUN_004107e0, che contiene una WriteFile (per cui l'ho rinominata caller_of_WriteFile).

1) Per quanto riguarda ReadFile, viene chiamata coi seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HANDLE	hFile	Handle al file da leggere.	0x33c (corrisponde a c:\Users\...\86RB6EG6-BUPI-X9AI-FFA68F35-B8A21FD0CBCB.asasin)
2	LPVOID	lpBuffer	Puntatore al buffer che riceverà i dati da leggere.	[out] 0x03104020
3	DWORD	nNumberOfBytesToRead	Massimo numero di byte da leggere.	126451
4	LPDWORD	lpNumberOfBytesRead	Puntatore alla variabile che riceverà il numero di byte che verranno effettivamente letti.	[out] 0x025df580
5	LPOVERLAPPED	lpOverlapped	Puntatore a una struttura OVERLAPPED.	NULL

2) Encrypt_AES, stavolta, riceve in input proprio il buffer lpBuffer popolato dalla precedente ReadFile: è qui che il file in questione viene cifrato.

3) SetFilePointer, ove necessario, memorizza il file pointer in due valori long.

4) Infine, la WriteFile, che a questo punto riporterà sul file le modifiche apportate da Encrypt_AES tramite un'operazione di scrittura, viene invocata coi seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HANDLE	hFile	Handle al file su cui scrivere.	0x33c (corrisponde a c:\Users\...\86RB6EG6-BUPI-X9AI-FFA68F35-B8A21FD0CBCB.asasin)
2	LPCVOID	lpBuffer	Puntatore al buffer che contiene i dati da scrivere.	0x03104020
3	DWORD	nNumberOfBytesToWrite	Massimo numero di byte da scrivere sul file.	126451
4	LPDWORD	lpNumberOfBytesWritten	Puntatore alla variabile che riceverà il numero di byte che verranno effettivamente scritti.	[out] 0x025df580
5	LPOVERLAPPED	lpOverlapped	Puntatore a una struttura OVERLAPPED.	NULL

Se ci facciamo caso, i parametri passati a WriteFile sono identici a quelli passati alla precedente ReadFile: ciò è coerente con l'idea secondo cui, all'interno del file, verranno scritti gli stessi dati letti tramite la ReadFile a seguito della loro cifratura con l'algoritmo AES.

Inoltre, osserviamo che ha perfettamente senso che le operazioni qui sopra elencate appartengano a un ciclo: qui sono coinvolte le operazioni di ReadFile e WriteFile, che di base operano su flussi di byte, per cui non si può assumere che operino atomicamente sul contenuto dei file.

Dopo il loop, viene effettuata un'altra scrittura sul file, vengono recuperate data e ora correnti del sistema mediante una `GetSystemTimeAsFileTime` e, con una `SetFileTime`, il timestamp appena ottenuto viene usato per aggiornare data e ora di creazione, ultimo accesso e ultima modifica del nostro file.

Ho a questo punto toccato i punti cruciali della funzione `FUN_00413be0` invocata da `FUN_00427ac0`; ho ridenominato tale `FUN_00413be0` in `encrypt_file_system_AES`.

Tornando a `FUN_00427ac0`, il terzo passaggio che sembra rilevante, dopo la chiamata a `encrypt_file_system_AES`, è l'invocazione di `FUN_0042e7d0`, che comunque appare come una funzione abbastanza semplice; comunque sia, appartiene al medesimo ciclo for di `encrypt_file_system_AES`. Quando l'istruzione pointer punta alla prima istruzione di `FUN_0042e7d0`, sia nel registro `EAX` che sullo stack è presente la seguente stringa UNICODE:

"c:\Users\matte\Downloads\Git\trunk\Slide\CNS\asasin-2ae2.htm"

Ho notato che tale stringa fa riferimento alla medesima directory in cui era presente il file che è stato cifrato ("45-mckenzie.pptx").

Guarda caso, `FUN_0042e7d0` effettua una chiamata a `CreateFileW` coi seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPCWSTR	lpFilename	Nome del file da creare o da aprire.	"c:\Users\...\ asasin-2ae2.htm"
2	DWORD	dwDesiredAccess	L'accesso richiesto al file.	GENERIC_READ GENERIC_WRITE
3	DWORD	dwShareMode	La modalità di condivisione richiesta del file.	0
4	LPSECURITY_ATTRIBUTES	lpSecurityAttributes	Puntatore a una struttura SECURITY_ATTRIBUTES.	NULL
5	DWORD	dwCreationDisposition	Azione da intraprendere sul file.	CREATE_ALWAYS
6	DWORD	dwFlagsAndAttributes	Flag e attributi del file.	0
7	HANDLE	hTemplateFile	Se il file viene creato, è l'handle a un file modello che fornisce gli attributi estesi per il file.	NULL

Se la creazione di "asasin-2ae2.htm" va a buon fine, viene invocata anche `DeviceIoControl`, un'API che invia un codice di controllo a un certo driver di dispositivo, in modo tale che il dispositivo esegua l'operazione corrispondente al codice di controllo. Tale API restituisce `TRUE` in caso di successo, `FALSE` altrimenti, e prende in input i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HANDLE	hDevice	Handle al dispositivo su cui l'operazione dovrà essere eseguita.	0x270 (corrisponde a "...asasin-2ae2.htm")
2	DWORD	dwIoControlCode	Codice di controllo per l'operazione.	FSCTL_SET_COMPRESSION
3	LPVOID	lpInBuffer	Puntatore al buffer di input che contiene i dati richiesti per eseguire l'operazione.	0x0260fc8c (qui ci sono i byte 01 00)

4	DWORD	nInBufferSize	Dimensione in byte del buffer di input.	2
5	LPVOID	lpOutBufferSize	Puntatore al buffer di output che riceverà i dati restituiti dall'operazione.	NULL
6	DWORD	nOutBufferSize	Dimensione in byte del buffer di output.	0
7	LPDWORD	lpBytesReturned	Puntatore a una variabile che riceverà le dimensioni dei dati memorizzati nel buffer di output.	0x260fc80
8	LPOVERLAPPED	lpOverlapped	Puntatore a una struttura OVERLAPPED.	NULL

Segue una chiamata a WriteFile coi seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HANDLE	hFile	Handle al file su cui scrivere.	0x270 (corrisponde a "...asasin-2ae2.htm")
2	LPCVOID	lpBuffer	Puntatore al buffer che contiene i dati da scrivere.	0x023f27e8
3	DWORD	nNumberOfBytesToWrite	Massimo numero di byte da scrivere sul file.	8124
4	LPDWORD	lpNumberOfBytesWritten	Puntatore alla variabile che riceverà il numero di byte che verranno effettivamente scritti.	[out] 0x0260fc74
5	LPOVERLAPPED	lpOverlapped	Puntatore a una struttura OVERLAPPED.	NULL

Su OllyDbg è chiaro come il buffer di input per la WriteFile contenga del testo HTML (vedere pagina seguente):

Address	Hex dump	ASCII
023F27E8	3C 68 74 6D 6C 20 63 6C	<html ol
023F27F0	61 73 73 3D 27 77 68 6C	ass='whl
023F27F8	6D 72 79 71 27 20 64 61	mryq' da
023F2800	74 61 2D 61 71 64 6D 6E	ta-aqdmn
023F2808	75 62 3D 27 75 6F 6C 79	ub='uoly
023F2810	76 76 68 64 27 3E 3C 68	uvhd'><h
023F2818	65 61 64 3E 0A 3C 6D 65	ead>.<me
023F2820	74 61 2D 63 68 61 72 73	ta chars
023F2828	65 74 3D 27 75 74 66 2D	et='utf-
023F2830	38 27 3E 0A 3C 73 74 79	8'>.<sty
023F2838	6C 65 3E 2E 78 71 76 63	le>.<xqvc
023F2840	75 77 2D 7B 63 6F 6C 6F	uw (colo
023F2848	72 3A 2D 23 64 65 64 65	r: #dede
023F2850	64 65 3B 6C 65 66 74 2D	de; left
023F2858	3A 2D 2D 39 31 30 70 78	: -910px
023F2860	3B 70 6F 73 69 74 69 6F	;positio
023F2868	6E 2D 3A 2D 61 62 73 6F	n : abso
023F2870	6C 75 74 65 3B 7D 62 6F	lute; }bo
023F2878	64 79 7B 62 61 63 6B 67	dy(backg
023F2880	72 6F 75 6E 64 2D 63 6F	round-co
023F2888	6C 6F 72 3A 23 64 65 64	lor: #ded
023F2890	65 64 65 3B 7D 2E 6F 69	ede; }.oi
023F2898	63 6C 69 7B 63 6F 6C 6F	cli (colo
023F28A0	72 3A 23 64 65 64 65 64	r: #deded
023F28A8	65 7D 3C 2F 73 74 79 6C	e)</styl
023F28B0	65 3E 3C 62 6F 64 79 3E	e)<body>
023F28B8	3C 66 6F 6E 74 2D 73 74	<font st
023F28C0	79 6C 65 3D 27 66 6F 6E	yle='fon
023F28C8	74 2D 66 61 6D 69 6C 79	t-family
023F28D0	3A 2D 74 61 68 6F 6D 61	: tahoma
023F28D8	27 3E EF BB BF 3C 64 69	'> }<di
023F28E0	76 2D 63 6C 61 73 73 3D	v class=
023F28E8	78 71 76 63 75 77 3E 63	xqvcuw>c
023F28F0	69 67 77 6C 64 77 62 68	lgwldwbh
023F28F8	3C 2F 64 69 76 3E 3D 5F	</div>=
023F2900	2D 2A 2A 2E 2E 24 5F 7C	-*...\$_T
023F2908	3C 73 70 61 6E 2D 63 6C	<span ol
023F2910	61 73 73 3D 27 6F 69 63	ass='oic
023F2918	6C 69 27 3E 65 3C 2F 73	li'>e</s
023F2920	70 61 6E 3E 3C 62 72 2D	pan><br
023F2928	2F 3E 3C 73 70 61 6E 2D	/><span
023F2930	63 6C 61 73 73 3D 27 6F	class='o
023F2938	69 63 6C 69 27 3E 61 3C	icli'>a<
023F2940	2F 73 70 61 6E 3E 3C 73	/span><s
023F2948	70 61 6E 2D 63 6C 61 73	pan clas
023F2950	73 3D 27 6F 69 63 6C 69	s='oicli
023F2958	27 3E 26 6E 62 73 70 3B	'> sp;
023F2960	3C 2F 73 70 61 6E 3E 3C	<
023F2968	73 70 61 6E 2D 63 6C 61	span cla
023F2970	73 73 3D 27 6F 69 63 6C	ss='oicli
023F2978	69 27 3E 61 3C 2F 73 70	i'>a</sp
023F2980	61 6E 3E 3C 64 69 76 2D	an><div
023F2988	63 6C 61 73 73 3D 78 71	class=xq
023F2990	76 63 75 77 3E 68 6E 62	vcuw>hnb
023F2998	70 6F 6D 67 3C 2F 64 69	nmq</di

Infine, FUN_0042e7d0 invoca una CloseHandle per il file asasin-2ae2.htm e termina.




























Alla fin dei conti, ho ridenominato tale funzione create_html.

Arrivato a questo punto, ho deciso di uscire dal ciclo for che contiene le invocazioni a encrypt_file_system_AES e create_html, e ho inserito un breakpoint in prossimità della funzione FUN_004396e0, che è l'altra degna di nota. Come mi aspettavo per via dell'esperienza accennata precedentemente, quando ho avviato l'esecuzione su OllyDbg, mi sono ritrovato in uno stato di eccezione dopo la creazione di molti thread. Ho dovuto premere SHIFT+F9 un numero smisurato di volte prima di uscire dalla condizione di eccezione e di raggiungere così il breakpoint.











Qui, se andiamo a farci un giro sullo stack, possiamo anche vedere un qualche riferimento all'RSA, il che conferma l'utilizzo di questo algoritmo dichiarato dal messaggio mostrato dal ransomware stesso a fine esecuzione:

025DF4C8	00000001	
025DF4CC	72449EA0	rsaenh.CPENcrypt
025DF4D0	025DFC94	
025DF4D4	7775AD10	ntdll.7775AD10
025DF4D8	07B951A4	

Dopodiché ho dato un'occhiata alle condizioni della mia macchina virtuale, e ho capito che il ciclo for che ho appena superato ha iterato sui file da cifrare all'interno del file system (e, in particolare, all'interno della directory "c:"). Infatti, la situazione è la seguente:

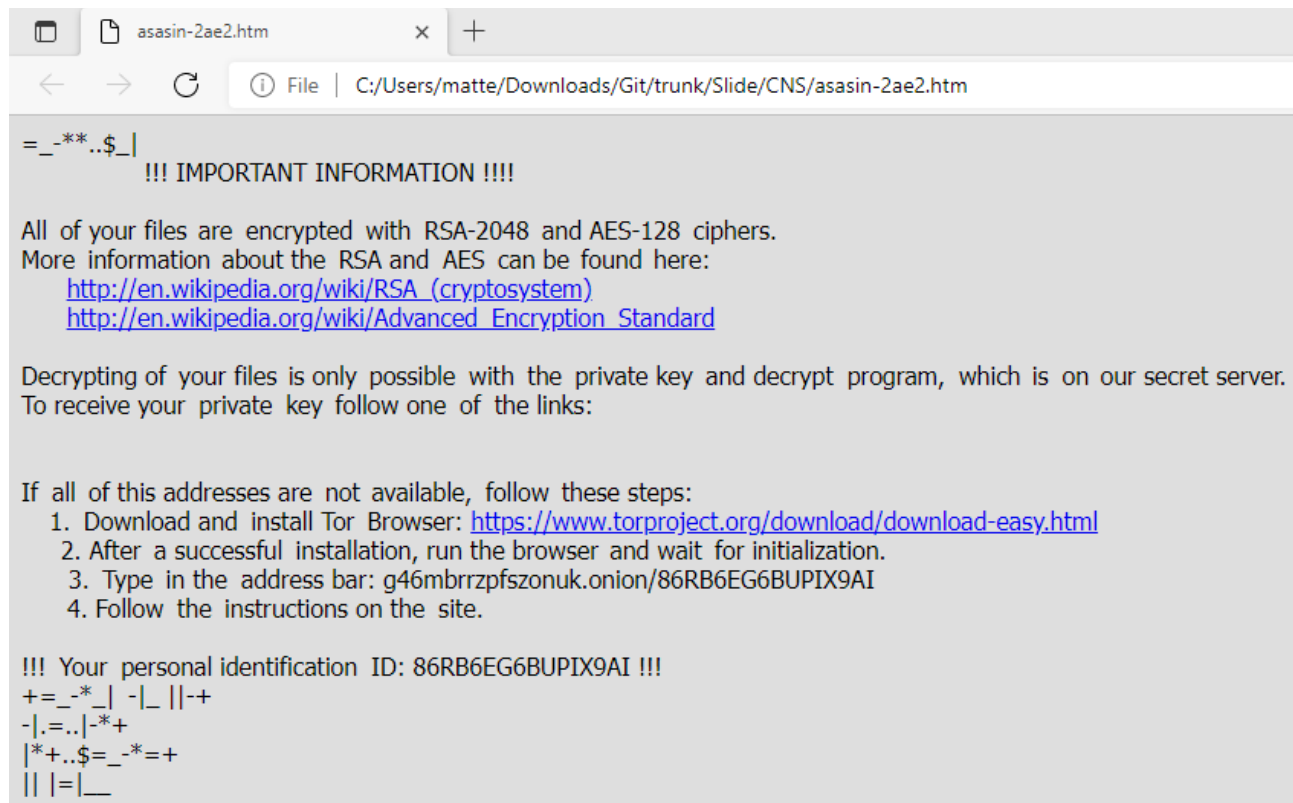
Nome	Ultima modifica	Tipo	Dimensione
 86RB6EG6-BUPI-X9AI-00E98F97-5893E2A...	25/01/2022 18:54	File ASASIN	588 KB
 86RB6EG6-BUPI-X9AI-0A41B505-64F560F...	25/01/2022 18:54	File ASASIN	500 KB
 86RB6EG6-BUPI-X9AI-2AE7A785-7745249...	25/01/2022 17:36	File ASASIN	125 KB
 86RB6EG6-BUPI-X9AI-4C0B0A8A-7887C1...	25/01/2022 19:01	File ASASIN	2.098 KB
 86RB6EG6-BUPI-X9AI-5C58E1B1-9101CB2...	25/01/2022 18:54	File ASASIN	284 KB
 86RB6EG6-BUPI-X9AI-5E8834F3-A4F6464...	25/01/2022 18:54	File ASASIN	253 KB
 86RB6EG6-BUPI-X9AI-6A510BCA-19A9E5...	25/01/2022 18:54	File ASASIN	211 KB
 86RB6EG6-BUPI-X9AI-8A0CBA2F-E6CAD...	25/01/2022 18:54	File ASASIN	402 KB
 86RB6EG6-BUPI-X9AI-8C785651-4EC01C6...	25/01/2022 18:54	File ASASIN	354 KB
 86RB6EG6-BUPI-X9AI-9A983277-053F6D7...	25/01/2022 18:54	File ASASIN	904 KB
 86RB6EG6-BUPI-X9AI-9D40EA21-7A68697...	25/01/2022 18:54	File ASASIN	491 KB
 86RB6EG6-BUPI-X9AI-51E74910-F5AF1FF...	25/01/2022 18:54	File ASASIN	197 KB
 86RB6EG6-BUPI-X9AI-63F7998C-ADD03D...	25/01/2022 18:54	File ASASIN	132 KB
 86RB6EG6-BUPI-X9AI-76FEAE73-6022B2A...	25/01/2022 18:54	File ASASIN	667 KB
 86RB6EG6-BUPI-X9AI-337FADFA-57AA77...	25/01/2022 19:01	File ASASIN	1.326 KB
 86RB6EG6-BUPI-X9AI-706D8C74-C94D3B...	25/01/2022 18:54	File ASASIN	163 KB
 86RB6EG6-BUPI-X9AI-771C2CF2-984690D...	25/01/2022 19:03	File ASASIN	6.600 KB
 86RB6EG6-BUPI-X9AI-1106BD62-FE87B36...	25/01/2022 18:54	File ASASIN	955 KB
 86RB6EG6-BUPI-X9AI-3647EEC0-4428768...	25/01/2022 18:54	File ASASIN	209 KB
 86RB6EG6-BUPI-X9AI-9581B74A-462204A...	25/01/2022 18:54	File ASASIN	169 KB
 86RB6EG6-BUPI-X9AI-AA17619B-4D2C78...	25/01/2022 19:01	File ASASIN	1.805 KB
 86RB6EG6-BUPI-X9AI-B12C0E69-C8F8982...	25/01/2022 18:54	File ASASIN	960 KB
 86RB6EG6-BUPI-X9AI-B79C54F8-403B0B9...	25/01/2022 19:01	File ASASIN	1.420 KB
 86RB6EG6-BUPI-X9AI-BAC2A606-3ED604...	25/01/2022 18:54	File ASASIN	311 KB
 86RB6EG6-BUPI-X9AI-DE6396EE-EA8C572...	25/01/2022 18:54	File ASASIN	362 KB
 86RB6EG6-BUPI-X9AI-F30CB324-D428D4...	25/01/2022 18:54	File ASASIN	532 KB
 86RB6EG6-BUPI-X9AI-F6356FAC-F8F6462...	25/01/2022 18:54	File ASASIN	189 KB
 asasin-2ae2.htm	25/01/2022 18:40	Microsoft Edge H...	8 KB

Una cosa spiacevole è che anche il file GhidraRun.bat, ovvero quello che viene lanciato per avviare Ghidra, è stato cifrato:

Nome	Ultima modifica
 docs	25/01/2022 19:04
 Extensions	31/10/2021 15:03
 Ghidra	31/10/2021 15:07
 GPL	31/10/2021 15:08
 licenses	25/01/2022 19:03
 server	25/01/2022 19:03
 support	25/01/2022 19:03
 86RB6EG6-BUPI-X9AI-AEFBAB23-FDE457...	25/01/2022 19:00
 ghidraRun	28/09/2021 18:45
 LICENSE	28/09/2021 18:45

Ciò potrebbe comportare un qualche problema con l'utilizzo di Ghidra d'ora in avanti. Probabilmente possiamo risolvere installando nuovamente Ghidra a questo punto dell'esecuzione del malware: proveremo a farlo se ci accorgeremo che sarà necessario.

A proposito del file "asasin-2ae2.htm", adesso, quando viene aperto, mostra la seguente pagina del browser:



Arrivato a questo punto, avevo intenzione di proseguire con l'analisi dinamica avanzata della funzione FUN_004396e0, per cui ho messo un breakpoint in prossimità della prima API invocata (CryptGenRandom) per vedere quali parametri accettasse, per poi eseguire uno Step Over. Purtroppo, però, OllyDbg è andato in crash, o meglio: è rimasto in uno stato di running perpetuo mentre la schermata dei registri è sparita; ho fatto più di un tentativo per assicurarmi che non fosse una casualità. Dopodiché, considerando anche il fatto che non ho trovato delle tecniche anti-debugger, mi sono arreso all'idea di non sfruttare OllyDbg per l'analisi di FUN_004396e0: qui mi arrangerò diversamente, sicuramente a partire dal codice disassemblato e decompilato di Ghidra. Anche se dovessi ottenere delle informazioni non esaustive o parziali, non sarebbe un grosso problema.

Come accennavo, inizialmente FUN_004396e0 invoca per due volte l'API CryptGenRandom in modo da riempire due buffer con dati crittograficamente casuali.

La prima volta vengono passati i seguenti parametri in input:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HCRYPTPROV	hProv	Handle al CSP creato precedentemente da CryptAcquireContext.	&DAT_00483654
2	DWORD	dwLen	Numero di byte di dati casuali da generare.	1
3	BYTE	*pbBuffer	Buffer che riceverà i dati restituiti.	[out] EBP-0x11

La seconda volta, invece, abbiamo i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HCRYPTPROV	hProv	Handle al CSP creato precedentemente da CryptAcquireContext.	&DAT_00483654
2	DWORD	dwLen	Numero di byte di dati casuali da generare.	32
3	BYTE	*pbBuffer	Buffer che riceverà i dati restituiti.	[out] EBP-0x128

Ho dunque effettuato le seguenti ridenominazioni su Ghidra:

- DAT_00483654 → CSP
- EBP-0x11 → CryptGenRandom_buffer1
- EBP-0x128 → CryptGenRandom_buffer2

Successivamente, la funzione chiama per due volte FUN_00434fa0. Essa contiene a sua volta tre chiamate a funzione: FUN_00434310, FUN_00433fb0 e FUN_004344e0.

1) FUN_00434310 contiene al suo interno la seguente chiamata a funzione:

```

}
*(undefined4 *)this = 0;
iVar1 = (*_DAT_0047a080) (*param_1,param_2,param_3,0,param_4,this);
uVar2 = (uint3) ((uint) iVar1 >> 8);

```

Ora, per capire di quale API si tratta, è conveniente tenere OllyDbg aperto e con l'esecuzione bloccata a una qualsiasi istruzione del codice inizialmente impacchettato (in modo tale che ci ritroviamo tutti i riferimenti a API e librerie già risolti). Da qui, sono andato all'indirizzo 0x0047a080 e ho visto cosa si trova al suo interno:

0047A074	• E0E38F75	DD ADVAPI32.RegSetValueExW
0047A078	• B0FD8F75	DD ADVAPI32.RegDeleteValueA
0047A07C	• 00F48F75	DD ADVAPI32.CryptAcquireContextA
0047A080	• E0E88F75	DD ADVAPI32.CryptImportKey
0047A084	00	DB 00
0047A085	00	DB 00
0047A086	00	DB 00
0047A087	00	DB 00
0047A088	• C06F6D75	DD GDI32.CreateCompatibleBitmap

Ho quindi concluso che l'API invocata è CryptImportKey, che trasferisce una chiave crittografica da una chiave BLOB a un Cryptographic Service Provider (CSP).

2) FUN_00433fb0 contiene invece un'invocazione a CryptCreateHash che, come già sappiamo, inizializza una funzione hash per uno stream di dati; il suo secondo parametro, che identifica l'algoritmo da usare, è pari a 0x8009, che corrisponde a CALG_HMAC.

3) FUN_004344e0, infine, contiene una chiamata a CryptSetHashParam, un'API che personalizza le operazioni di un oggetto hash, tra cui la configurazione del contenuto hash iniziale e la selezione di uno specifico algoritmo hash.

Alla fin dei conti ho effettuato le seguenti ridenominazioni:

- FUN_00434310 → containing_CryptImportKey
- FUN_00433fb0 → containing_CryptCreateHash
- FUN_004344e0 → containing_CryptSetHashParam
- FUN_00434fa0 → crypt_HMAC_init

Dopodiché FUN_004396e0 prevede un'invocazione a CryptHashData, che ricordiamo essere un'API che aggiunge dati all'oggetto hash creato tramite CryptCreateHash, e un'invocazione a CryptGetHashParam, che recupera i dati che governano le operazioni di un oggetto hash (tra cui il valore hash effettivo) e inserisce tali dati all'interno del buffer puntato da EBP-0x107 (che ho rinominato hash_object_data).

Dopodiché si hanno due chiamate alla funzione FUN_004743f0, che a sua volta contiene FUN_0046fdf0. FUN_0046fdf0 esegue delle operazioni su dei dati (AND, OR, XOR, shift left, shift right, PSHUFB):

```
26     else {
27         uVar2 = *(uint *) (*param_3 + 8);
28         uVar5 = *(uint *) (*param_3 + 0xc);
29         *(uint *) ((int)this + 0xf4) =
30             uVar2 >> 0x18 | (uVar2 & 0xff0000) >> 8 | (uVar2 & 0xff00) << 8 | uVar2 << 0x18;
31         *(uint *) ((int)this + 0xf0) =
32             uVar5 >> 0x18 | (uVar5 & 0xff0000) >> 8 | (uVar5 & 0xff00) << 8 | uVar5 << 0x18;
33         uVar5 = *(uint *) *param_3;
34         uVar2 = *(uint *) (*param_3 + 4);
35         uVar5 = uVar5 >> 0x18 | (uVar5 & 0xff0000) >> 8 | (uVar5 & 0xff00) << 8 | uVar5 << 0x18;
36         *(uint *) ((int)this + 0xf8) =
37             uVar2 >> 0x18 | (uVar2 & 0xff0000) >> 8 | (uVar2 & 0xff00) << 8 | uVar2 << 0x18;
38     }
39     uVar1 = *(uint *) ((int)this + 8);
40     /* WARNING: Load size is inaccurate */
41     uVar3 = *this;
42     *(uint *) ((int)this + 0xfc) = uVar5;
43     uVar4 = *(uint *) ((int)this + 0xc);
44     uVar2 = *(uint *) ((int)this + 4);
45     uVar5 = uVar3 >> 0x18 | (uVar3 & 0xff0000) >> 8 | (uVar3 & 0xff00) << 8 | uVar3 << 0x18;
46     param_3 = (undefined *) [16];
47     (uVar1 >> 0x18 | (uVar1 & 0xff0000) >> 8 | (uVar1 & 0xff00) << 8 | uVar1 << 0x18);
48     *(uint *) ((int)this + 8) =
49         uVar2 >> 0x18 | (uVar2 & 0xff0000) >> 8 | (uVar2 & 0xff00) << 8 | uVar2 << 0x18;
50     *(uint *) this = uVar4 >> 0x18 | (uVar4 & 0xff0000) >> 8 | (uVar4 & 0xff00) << 8 | uVar4 << 0x18;
51     *(uint *) ((int)this + 0xc) = uVar5;
52     *(undefined (**) [16]) ((int)this + 4) = param_3;
53 }
54 else {
55     if (param_3 == (undefined *) [16])0x0) {
56         auVar6 = (undefined [16])0x0;
57     }
58     else {
59         auVar6 = pshufb(*param_3, _DAT_0047e820);
60     },
```

Ho così deciso di ridenominare la funzione FUN_004743f0 in data_operations.

Dopodiché ci ritroviamo nuovamente davanti a encrypt_AES: qui vengono sicuramente cifrati altri dati mediante l'algoritmo AES. Qui, il secondo parametro di encrypt_AES è la variabile automatica pauVar13, ed è quella che contiene i dati che vengono cifrati; perciò rinomino tale variabile encrypted_data_AES. Segue un'invocazione a CryptEncrypt, un'API che, di nuovo, cifra dei dati; questi dati stavolta sono contenuti in CryptGenRandom_buffer2, ovvero nel buffer che era stato inizializzato tramite la seconda invocazione a CryptGenRandom all'interno di FUN_004396e0.

Abbiamo compreso che, in buona parte, la funzione FUN_004396e0 si occupa di cifratura, sfruttando in particolar modo gli algoritmi HMAC e AES. Queste stesse API di cifratura che abbiamo incontrato si ripresentano anche più avanti all'interno della funzione.

L'unica funzione invocata da FUN_004396e0 che ha un'aria tutta diversa è FUN_00436cb0. Quest'ultima contiene infatti delle invocazioni a:

- InternetCrackUrlA, che "rompe" un URL nelle sue parti componenti.
- InternetConnectA, che apre una sessione FTP o HTTP per un dato sito.
- HttpOpenRequestA, che crea un handle di richiesta HTTP.
- InternetQueryOptionA, che effettua una query su un'opzione Internet sull'handle specificato.

- `HttpRequestHeadersA`, che aggiunge una o più intestazioni della richiesta HTTP all'handle di richiesta HTTP.
- `HttpSendRequestA`, che invia una richiesta specifica al server HTTP.
- `InternetWriteFile`, che scrive dati su un file Internet aperto.
- `HttpEndRequestA`, che termina una richiesta HTTP.
- `HttpQueryInfoA`, che recupera le informazioni di intestazione associate a una richiesta HTTP.
- `InternetReadFile`, che legge i dati dall'handle aperto mediante l'API `HttpOpenRequestA`.
- `InternetCloseHandle`, che chiude un Internet handle.

Alla fin dei conti, ho effettuato le seguenti ridenominazioni in Ghidra:

- `FUN_00436cb0` → `Internet_function`
- `FUN_004396e0` → `encrypt_and_connect`
- `FUN_00427ac0` → `thread_clue`

Per quanto riguarda la connessione col server HTTP da parte del ransomware, ho provato a ottenere qualche informazione in più tramite Wireshark: ho dunque lanciato il malware normalmente, ho aperto Wireshark e ho avviato la cattura dei pacchetti: quando l'esecuzione del malware è terminata, ho provato a dare un'occhiata alle centinaia e centinaia di pacchetti che sono stati scambiati. Purtroppo però, con ogni probabilità, i messaggi sono stati sottoposti a cifratura, poiché risulta particolarmente tedioso leggerne il payload:

The image shows a Wireshark packet capture of an Ethernet frame. The packet details pane shows the following structure:

- Frame 1100: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \Device\NPF_{343336D4-E492-4B6A-AE82-74FCD33EADDE}.
- Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_b5:a4:e1 (08:00:27:b5:a4:e1)
- Internet Protocol Version 4, Src: 13.107.213.43, Dst: 10.0.2.15
- Transmission Control Protocol, Src Port: 443, Dst Port: 50092, Seq: 471572, Ack: 1993, Len: 1460
- Source Port: 443
- Destination Port: 50092
- [Stream index: 25]
- [TCP Segment Len: 1460]
- Sequence Number: 471572 (relative sequence number)
- Sequence Number (raw): 1059927573
- [Next Sequence Number: 473032 (relative sequence number)]
- Acknowledgment Number: 1993 (relative ack number)
- Acknowledgment number (raw): 1107383571
- 0101 = Header Length: 20 bytes (5)

The packet bytes pane shows the raw data of the packet. The first few bytes are the Ethernet II header, followed by the IP header, and then the TCP header. The payload of the packet is encrypted, as indicated by the non-readable characters in the packet bytes pane.

Non ci resta che effettuare delle ipotesi su quali dati possano essere scambiati durante la connessione tra il ransomware e il server:

- Nell'avviso che compare alla fine c'è scritto che sul server sono memorizzati la chiave segreta e l'algoritmo di decifratura: questi due oggetti potrebbero costituire un "topic" nello scambio di messaggi.
- Visto che però la comunicazione è stata piuttosto lunga e complessa, è plausibile che siano state scambiate anche informazioni più elaborate, come ad esempio delle statistiche.

Arrivato a questo punto, chiuderei con la trattazione del thread la cui esecuzione è partita dalla funzione `thread_fun2` e, dopo aver ripristinato lo stato "clean" della macchina virtuale, tornerei a concentrarmi sull'esecuzione del main thread.

Il main thread, dopo il ciclo for contenente la CreateThread, invoca la funzione FUN_0040fa10, che è potenzialmente interessante, perché ha al suo interno FUN_00478420 che a sua volta contiene invocazioni a LoadLibraryA e GetProcAddress.

In particolare, la LoadLibraryA carica la libreria vssapi.dll e le GetProcAddress, nell'ordine, forniscono gli indirizzi delle seguenti funzioni:

- 1) CreateVssBackupComponentsInternal, che crea un'interfaccia VssBackupComponents; essa è utilizzata per interrogare i writer sullo stato dei file e per eseguire le operazioni di backup o ripristino.
- 2) VssFreeSnapshotPropertiesInternal, che libera il contenuto di VSS_SNAPSHOT_PROP, una struttura contenente le proprietà di una copia shadow; una copia shadow è un elemento che permette la creazione di copie di backup di un file, una cartella o un volume.

A valle di ciò, ho rinominato la funzione FUN_00478420 backup_files_APIs.

FUN_0040fa10 contiene anche una funzione (FUN_0040f5d0) che invoca GetSystemDirectoryW, un'API che recupera il path della directory di sistema; quest'ultima contiene i file di sistema come le librerie a collegamento dinamico e i driver. Il valore di ritorno dell'API è la lunghezza, in TCHAR, della stringa indicante il path della directory di sistema in caso di successo, zero in caso di fallimento; i parametri che accetta in input sono:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPWSTR	lpBuffer	Puntatore al buffer che riceverà il path.	0x0019f9e4
2	UINT	uSize	Dimensione massima del buffer in TCHAR.	260

```

0019f9e0 00000000 ....
0019f9e4 003a0043 C:.
0019f9e8 0057005c \.w.
0019f9ec 006e0069 i.n.
0019f9f0 006f0064 d.o.
0019f9f4 00730077 w.s.
0019f9f8 0073005c \.s.
0019f9fc 00730079 y.s.
0019fa00 00650074 t.e.
0019fa04 00330060 m.3.
0019fa08 00000032 2...

```

Dallo stack si può capire che la directory di sistema è C:\Windows\system32.

L'ultima funzione chiamata da FUN_0040fa10 è FUN_0040d5c0 che, verso l'inizio, utilizza l'API CoCreateInstance. Quest'ultima crea un singolo oggetto della classe associata a uno specifico CLSID; un CLSID è un identificatore univoco globale messo a punto da Microsoft che identifica un oggetto di classe COM (Component Object Model).

Di fatto, sembrerebbe che FUN_0040d5c0 si occupi quasi per intero degli oggetti della classe COM, a giudicare anche dall'uso ricorrente della funzione _com_issue_error, la quale lancia un'eccezione di tipo com_error:

```

*(undefined4 *) (unaff_EBP + -4)
*extraout_ECX = piVar1;
if (piVar1 == (int *)0x0) {
    _com_issue_error(-0x7ff8fff2)
}
*in_FS_OFFSET = *(undefined4 *)

```

Ho dunque ridenominato FUN_0040d5c0 in COM_class_function e, senza ulteriori indugi, sono andato oltre nell'analisi di ransomware_clue.

Qui la prossima operazione di rilievo che viene eseguita è un'invocazione a WaitForSingleObject che, da come suggerisce OllyDbg, ha come argomento l'handle a una finestra:

```

0019fcc8 000000314 hObject = 000000314 (window)
0019fccc ffffffff Timeout = INFINITE
0019fcd0 00000001

```

Purtroppo, questa WaitForSingleObject lascia il thread nello stato di attesa per un tempo indeterminato (comunque molto di più della durata dell'esecuzione dell'applicazione nel caso in cui venga lanciata senza il debugger); tra l'altro, ho potuto notare che nel frattempo i file non vengono cifrati e apparentemente non accade nulla nel sistema: ho avuto la sensazione di essere entrato in una situazione di deadlock.

La mia idea è stata riavviare l'esecuzione del programma su OllyDbg, eseguire fino alla chiamata a WaitForSingleObject esclusa e fare una patch su questa istruzione di CALL, sostituendola con delle NOP. In tal modo, è possibile proseguire con l'analisi dinamica avanzata, almeno finché non si presenterà un qualche eventuale errore per via della mancata chiamata a WaitForSingleObject.

Procedendo con degli Step Over, sono giunto all'invocazione della funzione FUN_004273c0, che di interessante ha due funzioni di libreria: AddAtomA e GlobalAddAtomA.

AddAtomA aggiunge una stringa alla tabella Atom locale, e restituisce un valore univoco che identifica la stringa; di fatto, le tabelle Atom relazionano delle stringhe definite nel sistema con dei valori Atom univoci. L'unico parametro che qui AddAtomA accetta è la stringa `""NDQ3DWHJE6KFJOB""`.

GlobalAddAtomA, invece, aggiunge una stringa alla tabella Atom global e, come AddAtomA, restituisce un valore univoco che identifica la stringa. A giudicare dal parametro passato in input, si tratta della stessa stringa di prima.

Alla fin dei conti, ho rinominato FUN_004273c0 in adding_atom.

Dentro ransomware_clue segue una chiamata a FUN_00425870, che principalmente invoca altre funzioni definite all'interno dell'applicazione. Vediamo le più salienti.

a) FUN_00420a70 chiama l'API SHGetFolderPathW, che ottiene il path di una cartella identificata da un certo valore CSIDL, prendendo i seguenti parametri in input:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HWND	hwnd	Riservato.	NULL
2	int	csidl	Valore CSIDL che identifica la cartella il cui path dovrà essere recuperato.	16
3	HANDLE	hToken	Access token che può essere usato per rappresentare un particolare utente.	NULL
4	DWORD	dwFlags	Flag che specificano il path che dovrà essere restituito.	SHGP_TYPE_CURRENT
5	LPWSTR	pszPath	Puntatore a una stringa che riceverà il path.	[out] 0x0019f94c

A seguito dell'invocazione all'API, pszPath contiene la stringa `"C:\Users\matte\Desktop"`.

b) FUN_0042e6b0 chiama l'API FindFirstFileW che, come già sappiamo, cerca un file o una sottodirectory all'interno di una directory, accettando i seguenti parametri in input:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPCWSTR	lpFileName	Pathname del file o della directory cercata.	<code>"C:\Users\matte\Desktop\asasin.htm"</code>
2	LPWIN32_FIND_DATAW	lpFindFileData	Puntatore a una struttura WIN32_FIND_DATA che riceverà le informazioni riguardanti il file o la directory trovata.	[out] 0x019f918

Ricordiamo che il thread figlio aveva creato e aggiunto il file asasin-2ae2.htm all'interno della directory `C:\Users\matte\Downloads\Git\trunk\Slide\CNS` ma, con ogni probabilità, potrebbe non essere l'unica directory coinvolta in questa operazione.

Il problema è che, avendo saltato la chiamata a WaitForSingleObject, è possibile che l'esecuzione del main thread stia iniziando ad andare incontro a imprevisti, dato che tale file asasin.htm non è pervenuto (anzi, è come se i thread figli non esistessero proprio, dato che sembra non accadere nulla all'interno del sistema). Quello che ho fatto è stato creare un qualunque documento di testo sul desktop e rinominarlo (modificandone anche l'estensione) asasin.htm. Dopodiché ho eseguito uno Step Over su OllyDbg per eseguire FindFirstFileW: il valore di ritorno è effettivamente un handle (e non INVALID_HANDLE_VALUE), per cui sembrerebbe che la funzione abbia avuto successo.

Comunque sia, mi sono accorto solo successivamente che avrei anche potuto non preoccuparmi per la creazione di asasin.htm perché, se la funzione avesse fallito, sarebbe stata invocata poco dopo una create_html che avrebbe svolto il lavoro al posto mio. Questo, allora, può essere indice del fatto che, in realtà, la creazione del file asasin.htm nel **desktop** è delegata (o può essere delegata) al main thread. Dopodiché, FUN_0042e6b0 viene invocata una seconda volta, in cui FindFirstFileW prende invece i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	LPCWSTR	lpFileName	Pathname del file o della directory cercata.	"C:\Users\matte\Desktop\asasin.bmp"
2	LPWIN32_FIND_DATAW	lpFindFileData	Puntatore a una struttura WIN32_FIND_DATA che riceverà le informazioni riguardanti il file o la directory trovata.	[out] 0x019f918

Come prima, anche il file asasin.bmp, che dovrebbe rappresentare la famosa immagine contenente l'avviso, non è presente nel desktop; stavolta però, non mi preoccupo di creare il file, bensì lo lascio fare al malware. Infatti, poiché stavolta FindFirstFileW è fallita, sono previste le invocazioni a FUN_004248b0 e anche create_html (che altrimenti non ci sarebbero state).

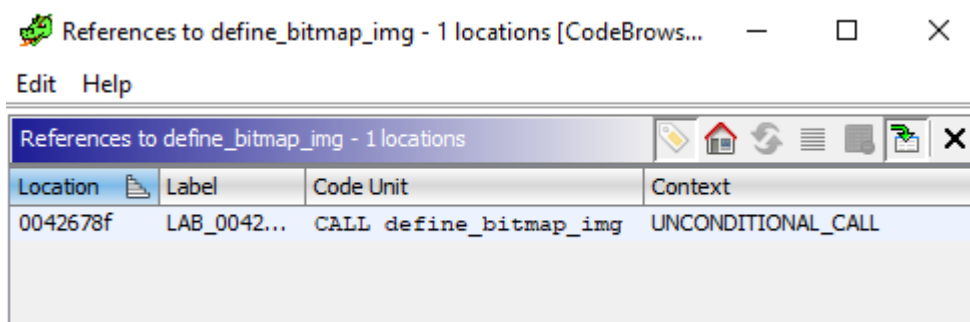
c) FUN_004248b0 è una funzione piuttosto complicata che chiama diverse API e altre funzioni, ma ha tutta l'aria di essere la funzione che crea l'immagine in formato bitmap; lo si capisce anche dai riferimenti a delle strutture dati che danno l'idea dell'utilizzo del formato bitmap:

```
FUN_00416ee0((void *) (unaff_EBP + -0x24), (HBITMAP *) (unaff_EBP + -0x14), 0,
             *(UINT *) (unaff_EBP + -0x60), (LPVOID) ((int)puVar6 + iVar3 + 0x36),
             (LPBITMAPINFO) (puVar10 + 7), 0);
```

Di conseguenza, ho provato a eseguire uno Step Over sull'istruzione CALL FUN_0042a8b0: sul desktop non è comparso nessun file. Però ho realizzato che non avevo tutti i torti, perché dopo, con l'invocazione a create_html, è apparso il fatidico file asasin.bmp. Ho capito dunque che FUN_0042a8b0 si occupa di impostare l'immagine asasin.bmp ma la creazione vera e propria del file viene delegata a create_html. Pertanto, su Ghidra ho effettuato le seguenti ridenominazioni di funzioni:

- FUN_004248b0 → define_bitmap_img
- create_html → create_html_or_bitmap

Per curiosità, ho provato a vedere se la define_bitmap_img viene invocata anche dal thread figlio la cui esecuzione parte da thread_fun2; la ricerca ha avuto esito negativo, poiché Ghidra mi dice che tale funzione ha un solo riferimento (ovvero quello che abbiamo appena incontrato nel main thread):



d) FUN_00416290 chiama l'API RegOpenKeyExA, che apre una specifica registry key, accettando i seguenti parametri in input:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HKEY	hkey	Handle a una registry key già aperta.	HKEY_CURRENT_USER
2	LPCSTR	lpSubKey	Nome della registry subkey da aprire.	"Control Panel\Desktop"
3	DWORD	ulOptions	Opzioni da applicare quando si apre la registry key.	0
4	REGSAM	samDesired	Maschera che indica i diritti di accesso desiderati per la registry subkey da aprire.	KEY_QUERY_VALUE KEY_SET_VALUE KEY_CREATE_SUB_KEY KEY_ENUMERATE_SUB_KEYS KEY_NOTIFY 0x20000
5	PHKEY	phkResult	Puntatore a una variabile che riceverà un handle alla registry subkey da aprire.	[out] 0x0019fcb0

e) FUN_004205a0 contiene a sua volta una chiamata alla funzione FUN_00416390, la quale utilizza l'API RegSetValueExA. Quest'ultima imposta i dati e il tipo di uno specifico valore di una registry key; in caso di successo restituisce ERROR_SUCCESS e prende in input i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	HKEY	hkey	Handle a una registry key già aperta.	3F8
2	LPCSTR	lpValueName	Nome del registry value da settare.	"WallpaperStyle"
3	DWORD	Reserved	Parametro riservato.	0
4	DWORD	dwType	Tipo di dato puntato dal parametro lpData.	REG_SZ (= stringa con carattere di terminazione)
5	const BYTE *	lpData	Dati da memorizzare nel registry value.	0x0019fc5c (è una locazione di memoria contenente la stringa "0")
6	DWORD	cbData	Dimensioni delle informazioni puntate da lpData.	2

Poco dopo, FUN_004205a0 (e quindi RegSetValueExA) viene invocata nuovamente. Stavolta, i valori dei parametri passati alla funzione di libreria sono:

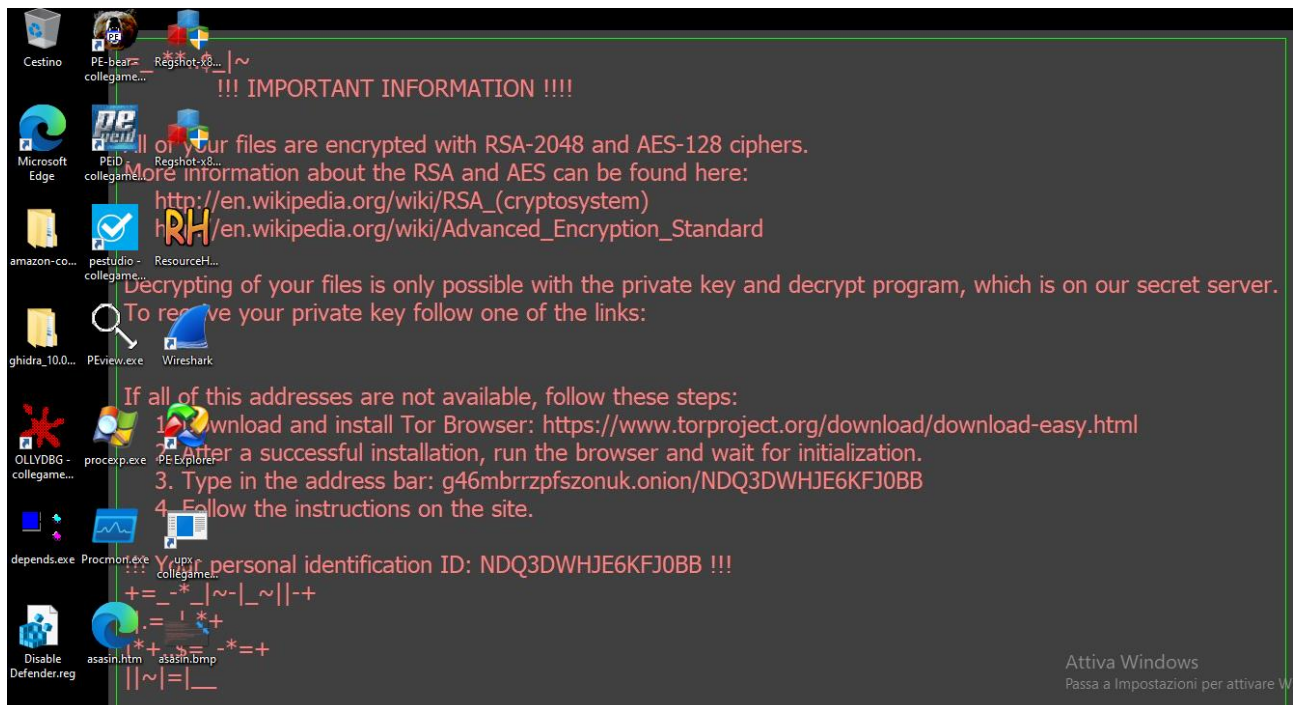
#	TIPO	NOME	DESCRIZIONE	VALORE
1	HKEY	hkey	Handle a una registry key già aperta.	3F8
2	LPCSTR	lpValueName	Nome del registry value da settare.	"TileWallPaper"
3	DWORD	Reserved	Parametro riservato.	0
4	DWORD	dwType	Tipo di dato puntato dal parametro lpData.	REG_SZ (= stringa con carattere di terminazione)

5	const BYTE *	lpData	Dati da memorizzare nel registry value.	0x0019fc5c (è una locazione di memoria contenente la stringa "0")
6	DWORD	cbData	Dimensioni delle informazioni puntate da lpData.	2

f) SystemParametersInfoA è un'API invocata direttamente da FUN_00425870 e ha lo scopo di recuperare o impostare il valore di uno dei parametri di sistema; in caso di successo restituisce un valore non nullo, altrimenti restituisce 0; infine, accetta i seguenti parametri:

#	TIPO	NOME	DESCRIZIONE	VALORE
1	UINT	uiAction	Parametro di sistema da recuperare o impostare.	SPI_SETDESKWALLPAPER
2	UINT	uiParam	Parametro il cui utilizzo dipende dal parametro di sistema richiesto o da impostare.	0
3	PVOID	pvParam	Altro parametro il cui utilizzo dipende dal parametro di sistema richiesto o da impostare.	0x023414d0 (contiene la stringa "C:\Users\matte\Desktop\asasin.bmp")
4	UINT	fWinIni	Se il parametro di sistema deve essere impostato, fWinIni specifica se il profilo utente deve essere aggiornato.	3 (potrebbe essere l'OR tra SPIF_UPDATEINFILE e SPIF_SENDCHANGE)

Abbiamo appena trovato la funzione che cambia lo sfondo del desktop, inserendovi l'immagine asasin.bmp (specificata nel terzo parametro di SystemParametersInfoA)! Infatti, dopo l'esecuzione dell'API, la macchina virtuale si presenta così:



Abbiamo finalmente messo a posto gli ultimi tasselli che ci mancavano per concludere l'analisi del nostro ransomware: è il main thread, verso la fine dell'esecuzione di ransomware_clue, a impostare l'immagine bitmap, a creare (e aprire) i file asasin.htm e asasin.bmp nel desktop e a cambiare lo sfondo del desktop.