Homework 4

Daniele Ferrarelli

Contents

1 Analisi Iniziale

- Processor: x86
- Endian: Little Endian
- Format Portaable Executable

Si esegue su una macchina virtuale appositamente preparata e scollegata da internet. Il malware in questione è un ransomware, poiché andrà a criptare file della macchina virtuale se eseguito. Successivamente mostrerà una pagina HTML ed una immagine contenente le istruzioni per ottenere il programma di decodifica. Si può notare che viene generato un ID del sistema corrente da utilizzare per contattare il sito malevolo. Nelle informazioni mostrate dal malware si vede che indica l'utilizzo di RSA-2048 ed AES-128. Inoltre il malware si cancella una volta terminato il lavoro. Al momento dell'avvio del malware, se non si disattiva Windows Defendere, viene rilevato come il ransomware Locky. Tramite una ricerca su internet è possibile ottenere informazioni utili sul funzionamento del malware.

2 Unpack del Malware

Tramite il tool PE Bear si inizia con una analisi del malware e si notano le sezioni UPX0 ed UPX1, questo indica che il malware fa' uso del packer UPX. Si tenta, utilizzando l'eseguibile UPX, di fare l'unpack del malware tuttavia la decompressione non ha avuto successo, questo può indicare l'utilizzo di modalità di packing diverse da quelle originali di UPX.

Si cerca allora di utilizzare OllyDBG per generare un dump del programma unpackato. Si prova a trovare un long jump su Ghidra, la ricerca dei jump produce 8 possibili jump da analizzare, tuttavia solamente il jump ad indirizzo 0x004a20d0 con target 0x0040ea13 sembra soddisfare le richieste.

Si prova ad utilizzare i plugin di OllyDbg per trovare l'OEP, tuttavia non si ottengono risultati. Utilizzando un breakpoint sull'accesso in memoria all'indirizzo 0x0040ea13 si può vedere come venga modificata la memoria tramite l'algoritmo di compressione utilizzato da UPX. Questo fa' pensare che effettivamente il jump sia il jump verso l'Original Entry Point. Si posiziona un breakpoint prima del jump e poi si produce un dump della memoria, successivamente tramite ImpRec si ricostruisce l'IAT. Successivamente si passa su Ghidra e lo si analizza.

Si notano all'interno dell'applicazione la presenza di due stringhe "lwpqabklbeiutlcti" e "vickhgfqkhpdvlnva" e l'utilizzo della chiamata IsBadStringPtrW per vedere se il pointer è valido e se si hanno diritti alla lettura. Tramite analisi statica si può vedere come vengano eseguite una serie di operazioni su queste due stringhe. Al termine di queste operazioni verrà utilizzata una chiamata a VirtualAlloc per preparare una zona di memoria dove verranno copiati dei byte. Verrà eseguito un salto all'interno di questa area di memoria e tramite debugger si può notare come venga cambiato l'address space originale. Questo

può indicare un ulteriore meccanismo di packing che maschera il malware come una applicazione innocua.

Al momento del salto in questa area di memoria viene passato sullo stack il riferimento ad Kernel32.GetModuleHandleA in modo da non dover risolvere i riferimenti al IAT. Da questo riferimento vengono risolte le altre API necessarie per il funzionamento di questa zona di memoria. Viene chiamata una nuova VirtualAlloc e si copierà il contenuto della zona di memoria in quella nuova, per poi continuare ad eseguire nella nuova zona. Al suo interno verranno utilizzate le chiamate a VirtualProtect per cambiare i permessi di accesso alla memoria e riscrivere i byte dell'address space utilizzando le istruzioni REP STOS e REP MOVS. Successivamente si andrà ad reimpostare i livelli di protezione del segmento appena modificato. Infine viene deallocata la memoria utilizzata in precedenza e si andrà a saltare nell'address space appena modificato. (MAGARI AGGIUNGI PIU DETTAGLI)

Il nuovo entry point dell'applicazione è all'indirizzo 0x00402D8F, come fatto in precedenza si crea un dump dell'applicazione tramite OllyDump per poi risolvere l'IAT tramite ImpRec. A questo punto analizzando il file su Ghidra si possono notare un numero maggiore di API utilizzate, inoltre le API sono legate al funzionamento del malware, come per esempio ADVAPI.dll che include le API per criptare i file. Da questo punto si può iniziare l'analisi del malware effettivo. Inoltre per facilitare l'analisi si disattiva l'ASLR di Windows.

3 Analisi Malware

Iniziando l'analisi tramite Ghidra si nota la presenza di chiamate a funzioni con la convenzione thiscall, questo può indicare l'utilizzo di un linguaggio ad oggetti all'interno del malware. Durante l'analisi si nota la presenza di un gran numero di NOP, JUMP e LEA che vengono utilizzate per ostacolare la fase di analisi statica e dinamica andando a complicare il flusso delle istruzioni. Il passaggio di molti parametri alle funzioni scritte dal creatore del malware viene oscurato utilizzando determinati offset rispetto ad EBP, questo ha reso l'analisi più ardua poiché per risolvere molti parametri si è reso necessario utilizzare il debugger e seguire i vari valori in memoria. Durante l'analisi del malware si fa molto uso del decompilatore di Ghidra poiché rende più leggibile il flusso delle esecuzioni rispetto al disassemblatore considerando la grande presenza di NOP e JUMP.

Una volta deoffuscato il malware si nota l'entry point originale dell'applicazione, in questo entry point si andrà a trovare la funzione che conterrà il main del malware all'indirizzo 0x0042C820. Questa funzione rinominata a mainWrapper chiamerà l'effettivo main del malware (0x00429EA0). Il main del malware è riconoscibile per il numero e tipologia dei parametri passati essendo int, char* e char*. Questa struttura risulta essere diversa dal classico main utilizzato nel linguaggio C, ma, seguendo la documentazione Windows, si nota come corrisponda alla segnatura della funzione main nel linguaggio C++. Varie funzioni presenti all'interno del malware non vengono analizzate, poiché sono funzioni legate alla gestione degli oggetti. Inoltre sono presenti molte funzioni che incap-

sulano la chiamata all'api vera e propria, come per esempio FUN_0x0041F680 che incapsula GetModuleFileNameW.

3.1 MainWrapper

All'interno del mainWrapper si nota la chiamata della funzione 0x00477050 prima dell'esecuzione del main, questa funziona risulta essere interessante nell'analisi del malware poiché dopo aver iniziato ad analizzarlo, si è notata la presenza di un area di memoria utilizzata dalla funzione main. Questa area di memoria viene allocata proprio da questa funzione, per questo motivo viene rinominata a prepareMemoryArea e si passa ad analizzarla. Dopo aver chiamato la funzione memoryWrapper verrà chiamato il main effettivo del malware.

3.1.1 PrepareMemoryArea

All'interno di questa funzione verrà allocata una porzione di memoria utilizzata poi dalla funzione main. Inoltre viene fatto un controllo sulla PEB per controllare la presenza di un debugger in esecuzione. Questo controllo viene eseguito accedendo, tramite il segmento FS, alla flag BeingDebugged presente nella PEB. Se il malware esegue sotto debugger verranno utilizzate le chiamate ad GetModuleHandle e GetProcAddress per ottenere il riferimento all'API della DLL kernel32 AllocConsole, che viene copiato nella zona di memoria precedentemente allocata.

Se non è presente il debugger in questa zona verranno inseriti dei byte di controllo ed una serie 0. Controllando il contenuto del resto dell'area di memoria, si può notare una serie di byte che corrispondono al linguaggio HTML, questo può indicare che verranno utilizzati per creare la pagina HTML mostrata quando il computer viene infettato. Viene inoltre trovata una sequenza di byte che potrebbe essere utilizzata per creare la bitmap mostrata insieme alla pagina HTML. All'interno è presente anche il testo mostrato all'utente, contenente le istruzioni per decriptare i file nel sistema. Si può notare l'ID placeholder che verrà sostituito con un ID generato dal malware per identificare la macchina infetta. In questa zona di memoria andranno ad essere inserite informazioni necessarie anche per successive funzioni del malware, un esempio può essere l'indirizzo ip con cui il malware comunicherà, la pagina htm da utilizzare ed alcuni comportamenti del malware (utilizzo di internet, scritture su registri di sistema). Queste informazioni sono presenti all'interno dell'applicazione, poiché non vengono accessi file esterni o servizi di comunicazione con internet, questo fa pensare che siano offuscate in qualche modo.

Per facilitare l'analisi si è disattivata la misura anti-debug modificando l'eseguibile, in questo modo è possibile analizzare il comportamento del malware senza che rilevi la presenza di un debugger. Inoltre considerando la complessità del codice molto spesso si deve derivare il funzionamento di molte funzioni dalle API utilizzate e dal comportamento osservato tramite debugger. Successivamente si passa all'analisi della funzione main del malware (0x00429EA0).

3.2 Main

Iniziando l'analisi della funzione main si nota la presenza di una funzione (0x00401018) che scrive su FS:[0], questa funzione viene chiamata all'inizio di molte altre funzioni, questo fa pensare che si possa trattare di una funzione inserita dal compilatore, per questo non viene analizzata.

Si passa ad analizzare la nuova funzione main dell'applicazione, tramite una veloce analisi Ghidra si può notare la presenza di funzioni che potrebbero essere relative al funzionamento osservato del malware, per esempio la cancellazione dell'eseguibile del malware.

Successivamente si nota la presenza della chiamata a SetErrorMode, questo viene fatto utilizzando le flag SEM_FAILCRITICALERRORS, SEM_NOGPFAULTERRORBOX, SEM_NOOPENFILERRORBOX. Questo set della gestione di errore permette di non mostrare dialoghi all'utente in caso di errore (file non trovato, Windows Error Reporting, critical-error-handler), questi errori vengono direttamente passati al processo. In questo modo il malware può nascondersi anche in caso di errore. Successivamente viene usata l'api SetUnhandledExceptionFilter per cambiare il gestore di eccezioni top-level, impostando la funzione FUN_0041F820 (malwareExceptionHandler).

3.2.1 Gestione delle eccezioni

Questo gestore di eccezioni chiuderà due handle (handle che fanno riferimento a mutex dichiarati successivamente) utilizzando la chiamata a CloseHandle. Poi utilizzerà una chiamata a GetModuleFileNameW, questa funzione permette di ottenere il path del file che contiene il modulo indicato da un handle passato alla funzione. Se questo handle è NULL, come nel caso di questa chiamata, allora questa funzione otterrà il path del processo corrente. Una volta ottenuto il path dell'eseguibile associato con il processo corrente verrà chiamata la funzione FUN_0042EC10 (rinominata a spawnProcess). All'interno di questa funzione verrà chiamata la API CreateProcessW che permette di creare un processo., come input a questa chiamata viene passato il path ottenuto in precedenza. Questo fa' pensare che se il malware incontra una eccezione andrà a lanciare una nuova istanza del malware per poi chiudere l'istanza corrente.

3.2.2 Settaggio Token

Successivamente viene chiamata la funzione FUN_0042CF00 , questa funzione andrà, tramite le API GetCurrentProcess e OpenProcessToken, ad ottenere un token per il processo corrente. Infine andrà ad impostare, tramite SetTokenInformation, la flag TokenVirtualizationEnabled.

3.2.3 Ricerca della lingua del sistema

All'interno del malware si notano chiamate ad una serie di API per ottenere gli identificativi della lingua usata per UI e per il sistema. Una volta ottenuti gli identificativi viene applicato un AND con 0x3ff e si controlla se il risultato

corrisponde ad 0x19. Se il risultato è diverso da 0x19 di andrà a saltare in un'altra porzione di codice, altrimenti si eseguirà la funzione FUN_00421DE0. Considerando la provenienza di molti malware si prova con l'id della lingua russa e si vede che in questo caso si esegue la funzione sopracitata. Si passa ad analizzare la funzione FUN_00421DE0 (rinominata a deleteMalware).

3.2.4 Delete malware

In questa funzione verranno prelevati i path dell'eseguibile corrente e della cartella temporanea di Windows. Successivamente si utilizzerà la chiamata MoveFileEx per spostare il malware nella cartella temp, viene anche cambiato il nome dell'eseguibile generandone uno casuale, utilizzando GetTempFileNameW, a cui viene aggiunto il prefisso "sys". Successivamente si utilizzerà di nuovo MoveFileExW con la flag MOVEFILE_DELAY_UNTIL_REBOOT per spostare il file temporaneo appena generato in una destinazione nulla. Utilizzando questa destinazione e la flag è possibile cancellare un file al momento del reboot del sistema. Infine si nota la presenza di una stringa offuscata, analizzando con ollyDbg è possibile vedere che questa stringa corrisponde al comando "cmd.exe /C del /Q /F". Questo comando verrà lanciato utilizzando la funzione spawnProcess analizzata in precedenza. In questo modo si lancerà una istanza di cmd.exe che eseguirà il comando del in maniera quiet (con la flag Q) e forced (con la flag F), cancellando il file temporaneo contenente il malware. Da questo si può capire che questa funzione andrà a rimuovere il malware dalla macchina corrente. Questa funzione verrà utilizzata sia nel caso in cui il malware ha terminato la sua esecuzione sia nel caso si deve cancellare poiché la lingua del sistema è in russo. Questa funzione si trova all'interno di un while loop insieme al resto delle istruzioni del main, questo poiché al termine dell'esecuzione delle istruzioni all'interno si andrà a saltare in questa funzione, cancellando il malware.

Andando a vedere la mappa nel mondo delle infezioni del ransomware Locky, su cui è basato il malware corrente, si può notare come le infezioni non siano presenti in russia. Questo perché il malware viene cancellato se la lingua del sistema è il russo.

3.2.5 Sleep del malware

Continuando l'analisi del malware si nota la presenza di uno sleep per un numero di millisecondi dipendente dai dati scritti in memoria precedentemente. Nel caso non sia presente il debugger lo sleep è di 12 secondi, mentre se è presente lo sleep può essere di un tempo dettato dal valore dei byte che rappresentano il riferimento ad AllocConsole presenti in memoria. Per continuare l'analisi del malware senza dover aspettare lo sleep il malware verrà modificato in modo da utilizzare un tempo di 0s modificando il valore di una PUSH prima della chiamata alla funzione Sleep.

4 Comportamento con debugger

Continuando l'analisi del malware, dopo la fase di sleep c'è un if in cui si andranno a controllare dei byte presenti nella zona di memoria settata all'inizio dell'esecuzione. Se in questa zona alcune porzioni non sono 0 significa che il malware ha rilevato la presenza di un debugger. In questo caso il malware andrà ad eseguire una serie di istruzioni che sono diverse dal comportamento normale del malware. Andando ad analizzare questa serie di istruzioni si può notare la presenza di una stringa offuscata, tramite debugger è possibile vedere che questa stringa è "svhost.exe". Questo fa pensare che il malware possa ripartire mascherandosi come il programma svchost.exe. Successivamente il malware andrà a copiarsi, tramite CopyFileW, nella cartella temp di Windows con il nome svchost.exe. Si nota inoltre che dopo il malware andrà a cancellare il file nella cartella temp svchost.exe:ZoneIdentifier, questo perché l'estensione :Zone.Identifier comprende dei metadati che gestiscono alcune feature di sicurezza di WIndows per file scaricati da internet. Poi verrà chiamata la funzione spawnProcess che farà partire il svchost.exe dalla cartella temp, questo eseguibile è in realtà quello del malware appena copiato. Infine il malware ritornerà all'inizio della funzione main, poiché è presente un ciclo while, e si cancellerà utilizzando la funzione deleteMalware.

5 Comportamento normale

Se il malware non rileva la presenza di debugger avrà un comportamento diverso al caso precedente. In questo caso la memoria contenente i byte di controllo non è più occupata dal riferimento ad AllocConsole ma contiene informazioni importanti per il funzionamento del malware. Viene controllato se all'offset 0xC il valore è 0, se sì si andrà ad eseguire questa serie di istruzioni. Per prima cosa verrà generato l'ID che andrà ad identificare la macchina una volta che è stata infettata. Questo viene fatto utilizzando la funzione FUN_00431440 rinominata a generateHash. Successivamente si utilizzerà la funzione FUN_00426A40 (rinominata a prepareMutex) per creare due semafori per la sincronizzazione di thread, da questo si può capire che il malware utilizzerà più thread per il suo funzionamento.

5.1 GenerateHash

All'interno di questa funzione verrà generato l'hash che identifica la macchina, questo viene fatto tramite una serie di informazioni del sistema. Per prima cosa si preleverà la Windows direcotry (in questo caso C:\Windows), con questa informazione si preleverà il GUID della partizione utilizzando la funzione GetVolumeNameForVolumeMountPointA. Una volta prelevato il GUID, si preparerà una CSP (cryptographic service provider), con tipologia PROV_RSA_AES e flag CRYPT_VERIFYCONTEXT. Con queste flag si preparerà un provider di funzioni di crittografia per hashare il GUID. Successivamente si creerà un hash con

la funzione CryptCreateHash, questa utilizzerà l'algoritmo CALG_MD5 specificato tramite un parametro della funzione. Infine si utilizzerà la funzione CryptHashData per creare un hash MD5 del GUID. Questo poi viene convertito ad una stringa che contiene solo valori esadecimali per poi essere tagliato in modo da contenere 16 caratteri. Questo ID della macchina verrà utilizzato poi per dare parte del nome dei file criptati e per identificarla se manderà dei dati ad un server remoto. Queste informazioni vengono trovate andando ad analizzare lo stack durante l'esecuzione passo passo del malware. Tuttavia si nota che questo ID è diverso da quello finale utilizzato per identificare la macchina. Questo si otterrà successivamente nell'esecuzione del malware.

5.2 Ricerca Atom

Si nota la presenza della funzione FUN_004270F0, questa funzione (rinominata a searchAtoms) viene chiamata dal main, dopo aver generato l'hash, all'interno di un IF. Andandola ad analizzare si nota la presenza di chiamate a funzioni per trovare Atom nel sistema. La tabella atom è una tabella definita dal sistema che mantiene stringhe ed identificatori. Tramite debugger si può notare che viene ricercato il valore "˜˜˜ID ˜˜˜", dove ID è l'identificativo della macchina. Viene ricercata sia la tabella globale del sistema che quella specifica all'applicazione tramite le API GlobalFindAtom e FindAtom. Questo atom potrebbero essere aggiunti più avanti nell'esecuzione e potrebbero indicare se la macchin è già stata infettata. Se vengono trovati si terminerà l'esecuzione del malware.

5.3 Comunicazione internet disattivata

Il malware nell'esecuzione normale può seguire due strade per il funzionamento. Nel caso nei byte di configurazione il byte ad offset 0xF sia zero il malware eseguirà senza connettersi ad internet. Questo si può capire poiché nelle parti successive all'analisi si notano funzioni che inviano richieste HTTP, queste funzioni vengono eseguite solamente se un byte è settato (rinominato a withInternet). Nel caso sia disattivata la connessione si andrà ad eseguire la funzione FUN_00423740 (rinominata a generateHash2). All'interno di questa funzione si vedono query ad informazioni di sistema utilizzando GetUserDefaultUILanguage, GetVersionEx, GetSystemMetrics e DsRoleGetPrimaryDomainInformation. Si nota anche la presenza della stringa offuscata "YBNDRFG8EJKMCPQX0T1UWISZA345H769". Eseguendo questa funzione si vede che viene ritornata sul registro EAX il riferimento ad una area dello stack contenente l'ID finale della macchina. Questo fa' pensare che qui viene terminata la generazione dell'ID della macchina utilizzando le informazioni prelevate in precedenza.

Infine si eseguiranno una serie di funzioni per impostare classi usate successivamente dal sistema, per poi andare ad eseguire la parte fuori dall'IF comune ai due modi di esecuzione.

5.4 Comunicazione internet attiva

Nel caso com sAaaaaaAAAAaaaaAAAAAAASDFGHJKJHGFDSA. Test con Wireshark ed indirizzo http trovato.

5.5 Zona comune

Dopo aver eseguito la zona di codice relativa alle azioni fatte con o senza internet si andrà ad eseguire la funzione FUN_0046C640 (rinominata a getAllDrives). In questa funzione si nota la presenza delle chiamate GetLogicalDrives, GetDriveType, GetDiskFreeSpaceExW e GetVolumeInformationW. Alcune di queste funzioni vengono eseguite all'interno di un ciclo, questo fa pensare che si stanno raccogliendo delle informazioni sui drive della macchina virtuale. Eseguendo questa funzione si nota come venga prodotta una stringa contenente "c:". Questa stringa può indicare il drive principale di WIndows, quindi si suppone che questa funzione vada a cercare i drive della macchina virtuale corrente, per questo viene rinominata a getAllDrives.

Successivamente verrà chiamata la funzione FUN_00476B50, questa funzione viene rinominata a CreateMalwareThread.

5.5.1 Create Malware Thread

Al suo interno si notano numerose chiamate alla funzione FUN_00474820. Al suo interno si andrà ad ottenere un security token tuilizzando GetCurrentProcess e OpenProcessToken per poi utilizzare LookupPrivilegeValueA per risolvere una stringa ad un determinato id del permesso. Successivamente si userà la chiamata AdjustTokenPrivileges per settare il token precedentemente ottenuto con il permesso appena trovato. Questa funzione, rinominata a setupPrivilege, viene chiamata 4 volte andando a settare i privilegi di "SeDebugPrivilege", "SeTakeOwnershipPrivilege", "SeBackupPrivilege", "SeRestorePrivilege". Successivamente si nota la chiamata a GetModuleHandleA con parametro la stringa "ntdll.dll", poi si utilizzeranno chiamata a GetProcAddress per risolvere 3 API di ntdll.dll. Le api risolte sono NtQueryInformation, NtDuplicateObject e NtQueryObject, questi riferimenti verranno mantenuti come variabili globali. Infine si farà partire un thread con la chiamata CreateThread che eseguirà dalla funzione FUN_004768D0 (rinominata a threadFunction).

Analisi threadFunction All'interno di questa funzione si nota un ciclo while infinito con la chiamata della funzione FUN_00476150 (rinominata a closeOtherHandles) e uno sleep di 2 secondi.

Si passa ad analizzare la funzione closeOtherHandles. All'interno di questa funzione verrà chiamata la funzione FUN_004749C0 (rinominata a getSystemHandleInfo). Al suo interno verrà chiamata la API NtQuerySystemInformation che a seconda dei parametri può ritornare diverse tipologie di strutture contenenti informazioni sul sistema. In questo caso viene passato il parametro 0x10 che corrisponde a SystemHandleInformation. Con questo parametro la chiamata

andrà a produrre la struttura SYSTEM_HANDLE_INFORMATIONS. Questa struttura tuttavia non è documentata nel manuale ufficiale di Windows, quindi alcuni dettagli potrebbero non essere correttamente analizzati.

```
struct SYSTEM_HANDLE_INFORMATION {
ULONG NumberOfHandles;
SYSTEM_HANDLE_TABLE_ENTRY_INFO Handles [ANYSIZE_ARRAY];
}
```

Le entry presenti nell'array sono del tipo SYSTEM_HANDLE_TABLE_ENTRY_INFO. Questa struttura contiene informazioni sulle handle, come per esempio UniqueProcessorId, HandleAttributes, HandleValue. Queste strutture non sono documentate da Windows, quindi alcuni campi non sono conosciuti.

Successivamente la funzione andrà a prendere l'ID attuale del processo, per poi utilizzare la funzione FUN_00474C30 rinominata openAndSwap, per ogni handle contenuta nella struttura precedente. All'interno della funzione si utilizzeranno le chiamate ad OpenProcess, con permesso di PROCESS_DUP_HANDLE, GetCurrentProcess e NtDuplicateObject (funzione non documentata, tuttavia viene chiamata dall'API DuplicateHandle). Quello che andrà a fare il thread è duplicare il riferimento delle handle presenti nel sistema all'interno di questo processo. La chiamata NtDuplicateObject permettere di duplicare un handle di un determinato processo in modo da poterla utilizzare anche nel processo target. In questo caso verranno aperti i riferimenti ai processi che hanno le handle per poi duplicare i riferimenti anche con il processo corrente. Poi verrà fatto partire un altro thread che utilizzerà NtQueryObject per ottenere il nome dei file che hanno come riferimento le varie handle, una volta che questo thread ha completato la sua ricerca viene terminato.

Successivamente fuori da questa funzione verrà controllato che per ogni file aperto con una handle appartenga alla whitelist. Per fare ciò vengono utilizzate le funzioni che verrano analizzate successivamente whiteListDirectory e whitelist. Se ci sarà un math all'interno di queste liste si andrà ad eseguire nuovamente la funzione openAndSwap. Questa volta si passerà il parametro DUPLICATE_CLOSE_SOURCE in modo da chiudere le handle. Questo viene fatto poiché lasciare handle aperte da altri processi potrebbe portare a qualche problema nel momento in cui i file verranno criptati e rinominati.

5.5.2 Creazione thread

For per creare i thread

Analisi threadMalwareCrypt

Ricerca dei file

Funzione di criptaggio

Connessione ad internet attiva

5.5.3 Richieste RPC

5.5.4 Scrittura su chiavi di registro

Per l'encrypt dei dati sono presenti istruzioni AESENC dell'SEE. HTML presente nella zona virtualallocata.

Se è debuggato forse parte su svchost, lo fa partire da C:\Windows\User\AppData\Local\Temp\svchost.exe poiché farà una copia dell'eseguibile utilizzando CopyFileW.

Usa chiavi di registri per cambiare il wallpaper all'interno di launch something.

Si controlla il traffico utilizzando Wireshark. Si notano query DNS a WPAD (Web Proxy Auto-Discovery) ma dovrebbe essere proxy di Windows. Inoltre DNS a sb.scorecardresearch.com, img-s-msn-com.akamized.net, api.msn.com, c.msn.com, c.bing.com, assets.msn.com (credo siano tracker). Si nota trasmissione di dati con un indirizzo 204.79.197.219 probabilmente per ottenere la chiave di cifratura per il malware.

securelist.it - locky

Analisi thread di cifratura Continuando l'analisi si nota la presenza all'interno della funzione main di uno spawn di un thread, questo thread andrà ad eseguire subito dopo la creazione nella funzione threadFunction (FUN_00429b40). In questa funzione si andranno a modificare le priorità del thread corrente utilizzando le chiamate a GetCurrentThread e SetThreadPriority, viene impostata una bassa priorità (THREAD_PRIORITY_LOWEST).

Sucessivamente si andrà a chiamare la funzione PREPARA WHITE LIST di direcotry e file e si preleva tutte le classi, poi chiama quella che cripta. In prepareCSP si andrà a prendere una chiave RSA che veniva mantenuta nella memoria allocata all'inizio dell'esecuzione del malware, chiave RSA1 (probabilemte pubblica usata per criptare la chiave vera). Vengono usate funzioni criptografiche che generano dati casuali per il nome del nuovo file. Usa la chiave RSA1.

encryptFile Possono essere usate anche le istruzioni AESENC con accellerazione, tuttavia queste istruzioni non vengono riconosciute da OllyDBG. Questa funzione encryptWithAESENC andrà a criptare i dati presi dal file. Inoltre imposterà la data di creazione del file al tempo corrente.

Forse il malware ha disattivato la comunicazione con internet, questo perché la funzione che cripta non andrà a comunicare con internet(??????).

5.6 Comunicazione con Internet

Comunicazione disattivata, si vede un URL non completo. I Thread che criptano probabilemente mandano informazioni alla centrale di controllo.

5.7 CoFunctions

5.8

Vengono create 1 asasin per cartella, il main thread lo crea nel desktop

6 Riassunto