Homework 4

Daniele Ferrarelli

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 "lwpqabkl-beiutlcti" 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.

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 lin-

guaggio C, ma, seguendo la documentazione Windows, si nota come corrisponda alla segnatura della funzione main nel linguaggio C++.

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 verranno applicate delle misure anti-debug

3.2 Analisi

Si inizia l'analisi del malware vero e prorio e si vede una funzione simile ad un entry point classico. Si cerca il main, vedendo la struttura delle istruzioni si può supporre che la funzione 0x00429ea0 sia il main. Mentre si analizza si nota la presenza di un grande numero di jump e NOP all'interno dell'applicazione per complicare il flusso delle istruzioni. ???Dalle chiamate fatte si può notare l'utilizzo di handler di eccezioni.

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. Si continua l'analisi supponendo che questa funzione faccia le operazioni del malware.

All'inizio dell'esecuzione in questa funzione si vede la chiamata alla funzione FUN_00401018 (changeFirstSE), e si passa ad analizzarla. Si può notare l'accesso al segmento FS con offset 0, questo segmento a quell'offset contiene il TEB. In questo TEB è contenuto il gestore di eccezioni, questo viene salvato sullo stack e viene sostituito dalla funzione 0042C70B.

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, viene inserita la funzione FUN_0041F820 (malwareExceptionHandler).

Successivamente si passa alla funzione FUN_0042CF00 (lookupVirtualizationEnbaled), in questa funzione viene prelevato l'handle del processo tramite

GetCurrentProcess, per poi utilizzare openProcessToken con l'andle appena ottenuto, con accesso TOKEN_ADJUST_DEFAULT (utilizzato per cambiare l'ownerm gruppo) per prelevare un security token. Successivamente viene utilizzata la chiamata a SetTokenInformation utilizzando il security token appena ottenuto utilizzando il token TokenVirtualizationEnabled, in questo modo si può ottenere se la virtualizzazione è abilitata, per poi chiudere l'Handle del processo e ritornare.

All'interno del main poi viene fatta la chiamata a GetSystemDefaultLangID, GetUserDefaultLangID e GetUserDefaultUILanguage per ottenere il linguaggio di defult del sistema, dell'utente e dell'UI. Successivamente tramite debugger si vede che si va in sleep, questo può indicare la presenza di misure anti-debug.

3.3 Ricerca meccanismi anti analisi

Questa misura consiste nel andare in sleep se un determinato dato in memoria è diverso da 0, utilizzando il debugger si può vedere che questa zona di memoria fa parte delle zone allocate in precedenza, quindi al loro interno si potrebbe usare qualche meccanismo anti -debug. In questo indirizzo di memoria vengono copiato CCCCCCCC, che viene controllato e porta il sleep il malware. Zona di memoria allocata separatamente. Questa zona di memoria viene preparata dalla funzione FUN_0042c820 (mainWrapper)che è un wrapper della funzione main, chiamato dalla funzione entryFromDeobfs. Questo wrapper si occuperà di chiamare funzioni che deallocano le aree di memoria utilizzate per mutare il malware. Poi allocherà una nuova zona di memoria. Si nota l'accesso alla struttura TEB ad offset FS:[0x18], per poi accedere la PEB e controllare il byte BeingDebugged. Se durante l'analisi si andrà a cambiare il risultato del registro è possibile continuare con l'analisi dinamica, senza che il programma termini.

Si nota la presenza di function pointer assegnati dalla funzione FUN_00476b50, si va ad analizzare questa funzione con il debugger per capire che api vengono usate.

3.4 Analisi meccanismi malware

Dopo aver superato il meccanismo antidebug si va ad analizzare la funzione successiva. Utilizzando GetModuleFileNameW ottiene il filename del processo corrente(eseguibile corrispondente). func3.

Poi si andranno a confrontare due stringhe utilizzando la funzione FUN_0042E4C0 (checkStringEqual). Le stringhe confrontate sono (che dovrebbero contenere la locazione).

3.5 Servizi

FORSEEEEEE Tramite un regshot è possibile confrontare le chiavi del sistema, essendo presenti api di operazioni su registri. (controllare regshot). Sembrerebbe aggiungere un servizio al gruppo Boot Bus Extender. Si vede un servizio fxedzgcv.sys comparire nella directory dei drivers di windows.

FUN_0041f560 Successivamente si va ad analizzare la funzione FUN_0041f560, in questa funzione si utilizzerà GetTempPath per ottenere la directory contenente i file temporanei.

Analisi thread 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. Successivamente si possono notare chiamate a funzioni che preparano aree di memoria e utilizzano operazioni su file come FindFirstFileW.

Grr All'interno delle funzioni utilizzate si possono notare riferimenti simili ad una programmazione ad oggetti. Ghidra riconosce molte delle funzioni associate ad una determinata struttura, questo perché è presente il parametro this in molte delle funzioni analizzate.

Considerando la complessità del codice a disposizione si sceglie di passare ad analizzare partendo dalle chiamate API, si conosce il funzionamento del malware quindi si passa ad analizzare chiamate Http e ad funzioni di crittografia. Da come vengono utilizzate le funzioni del codice si cerca capire l'utilizzo delle varie funzioni.

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.

securelist.it - locky