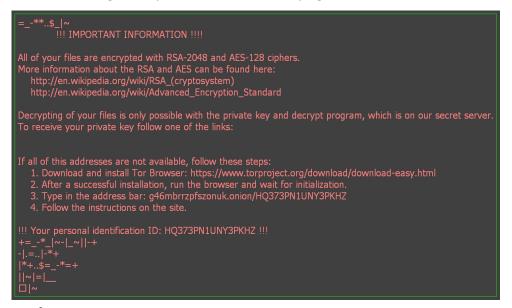
Homework 4

Pierpaolo Spaziani - 0316331

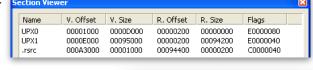
Introduzione

Sapendo che il programma sotto analisi è un malware, dopo aver effettuato uno snapshot della macchina virtuale, è stato lanciato e ne è stato osservato il comportamento, sia dal punto di vista dell'utente che con programmi di analisi, come *Regshot* e *ProcessMonitor*. Il programma è un **Ransomware - Trojan** che utilizza un metodo di criptazione asimmetrica per codificare i file dell'utente, lasciando inalterato il funzionamento della macchina. I target sono quindi solo file con determinate estensioni. Dopo il termine dell'esecuzione, ai file criptati viene assegnata l'estensione *asasin* e viene comunicato all'utente che è stato vittima di un attacco di tipo ransomware, visualizzando una foto con il procedimento per il recupero della chiave di decriptazione. La stessa foto inoltre viene impostata come sfondo e le stesse informazioni vengono riportate tramite una pagina sul browser di default.



Analisi preliminare

Aprendo il file con **PEID** è possibile notare che è stato impacchettato con *UPX*. Tuttavia non è stata utilizzata la versione base/standard ma una custom, infatti non ci sono le 3 classiche



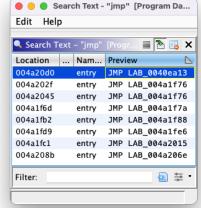
sezioni UPX0, UPX1 e UPX2, ma solo le prime due.

Risulta quindi inutile provare a spacchettarlo con UPX standard.

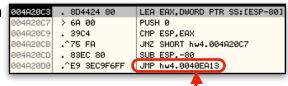
Aprendolo con *PE Explorer*, nonostante tenti e riesca ad effettuare uno spacchettamento valido, si giunge alle stesse conclusioni. Viene però suggerito come *Entry Point* **40EA13**.

Si procede alla ricerca dell'*Entry Point* con *Ghidra*, in particolare andando a cercare tra i jump, l'unico che risulta essere un salto lontano è proprio quello in direzione di **40EA13**.

Bisogna procedere quindi dal debugger.



Da *OllyDbg* si vede che la prima istruzione eseguita è PUSHAD, quindi vengono salvati i registri sullo stack. Da qui è semplice proseguire per arrivare all'*Original Entry Point*.



Come ci si aspettava si arriva proprio all'indirizzo 40EA13.

È possibile quindi effettuare un dump utilizzando il plugin *OllyDump* ed iniziare la prima analisi statica.

Prima di passare a Ghidra, risulta utile analizzare anche i log dei programmi di analisi lanciati in background. In particolare sul report di *ProcessMonitor* è possibile vedere che vengono lanciati 3 processi:



Con le relative specifiche:

Command line: "C:\Program Files\Internet Explorer\iexplore.exe" -nohome
Command line: "rundl|32.exe" C:\WINDOWS\system32\shimgvw.dll,ImageView_Fullscreen C:\Documents and Settings\Administrator\Desktop\asasin.bmp
Command line: cmd.exe /C del /Q /F "\\Mac\Shared Windows\hw4.exe"

- Il browser viene lanciato senza la homepage;
- La rundll32.exe viene utilizzata per aprire l'immagine descritta precedentemente, infatti: rundll32.exe" C:\windows\system32\shimgvw.dll,ImageView_Fullscreen <file_path>
 è il comando standard per aprire un'immagine con "Windows Picture and Fax Viewer".
- Utilizza cmd.exe per avviare una routine di auto-delete, infatti:

con:

/C Esegue il comando specificato e si arresta.

del Elimina il file specificato.

/Q Permette di eliminare senza chiedere conferma.

/F Eliminazione di forza dei file di sola lettura.

Su *Regshot* è possibile vedere che vengono sia modificate che aggiunte diverse chiavi, ad esempio si può vedere come cambia lo sfondo:

 $HKU\S-1-5-21-725345543-2025429265-1417001333-500\Control\ Panel\ \Desktop\Wallpaper: "C:\Documents and Settings\Administrator\Desktop\asasin.bmp".$

Analisi statica

Aprendo il codice ottenuto con *Ghidra*, non sembra usare tecniche di anti-disassemblaggio note, tuttavia la parte leggibile è molto scarsa. È facile ipotizzare che il programma si modifichi a run-time, ma per fare questo ha bisogno di usare DLL ed API al momento non caricate. Risultano invece semplici da individuare le funzioni LoadLibrary e GetProcAddress. Il target di analisi è quindi l'elenco di stringhe riconosciute dal disassemblatore, infatti contiene diverse funzioni e DLL importate dal programma durante l'esecuzione passandole per nome.

Analisi dinamica

Passando ad *OllyDbg* si cominciano ad incontrare le prime tecniche di intralcio per l'analisi.

Infatti, poco dopo l'inizio del programma, è possibile trovare un ciclo di 30 (1E esadecimale) esecuzioni, contenente la funzione WaitForSingleObject con

```
PUSH 1E
          6A 00
                           PUSH 0
          68 E8030000
                           PUSH 3E8
00407307
304073D0
                           PUSH -1
004073DE
          8D05 8C404900
                           LEA EAX,DWORD PTR DS:[<&kernel32.WaitForSingleObjectEx>]
004073E4
          FF10
                           CALL DWORD PTR DS:[EAX]
                           DEC DWORD PTR SS:[ESP]
004073E6
          FF0C24
                           JNZ SHORT hw4_dump.004073D5
          83C4 04
                           ADD ESP,4
```

timeout pari a 1000 millisecondi. In tutti i casi osservati, il valore di ritorno è stato sempre 102, ovvero WAIT_TIMEOUT. È stato quindi possibile patchare l'eseguibile per rimuovere questo ciclo (lasciando solo l'istruzione PUSH 0) ed evitare di aspettare 30 secondi ad ogni esecuzione (è stato inoltre lanciato l'eseguibile per conferma ed effettivamente era del tutto funzionante).

Proseguendo si incontra la prima VirtualAlloc all'indirizzo 40699A.

I parametri passati permettono di decidere al sistema dove allocare i 1672 byte che, come il flag di protezione specifica, dovranno contenere codice eseguibile. Il valore di ritorno di

```
Address = NULL
Size = 688 (1672.)
AllocationType = MEM_COMMIT
Protect = PAGE_EXECUTE_READWRITE
```

questa funzione è proprio l'indirizzo di base della memoria allocata, in questo caso **940000**. Facendo 'Follow in Dump' sul valore presente nel registro EAX, è quindi possibile arrivare in questa zona e vedere come viene popolata nel ciclo successivo, ovvero quello compreso tra gli indirizzi **4069AC** e **4069CF**. Successivamente l'esecuzione prosegue su questo "nuovo" codice. Qui sono presenti diversi un loop in cui vengono caricate dinamicamente le funzioni di interesse delle DLL.

All'indirizzo **940065** c'è una nuova VirtualAlloc con valore di ritorno **950000**. Anche qui, il codice con cui viene riempita la memoria contiene caricamenti dinamici delle API. Proseguendo è possibile vedere come il programma continua ad allocare ed a modificarsi. All'indirizzo **9501BE** viene effettuato un jump all'indirizzo **402D8F**, tornando quindi al "codice originale", tuttavia è completamente diverso poiché è stato modificato a run-time. Arrivando all'istruzione **402d37** si nota che l'esecuzione si arresta. Quindi eseguendo '*Step into*' su di essa si arriva su una zona di codice non analizzata dal debugger. Facendo analizzare il nuovo codice al debugger con *Ctrl+A*, è possibile vedere il vero e proprio malware. Effettuare un dump a questo punto, permette di procedere parallelamente con l'analisi statica del codice.

Anti-debugging

In tutto il codice, compreso quello già analizzato, sono presenti sistemi intralcio all'analisi. Diverse volte viene chiamata GetTickCount insieme a QueryPerformanceCounter per valutare le performance d'esecuzione del codice. IsDebuggerPresent è nel codice, ma, invece che far terminare l'esecuzione, induce a comportamenti differenti da quelli standard. Il codice è stato analizzato istruzione per istruzione a causa di questi controlli, infatti se lanciato con 'Run' o 'Animate over', il debugger si arresta ma non termina.

Il nuovo dump ottenuto risulta impossibile da seguire dal debugger poiché il flusso è interrotto ogni 2/3 istruzioni da jump. Questo fa si che il codice non sia lineare ed è difficile mantenere il filo logico d'esecuzione.

La stessa difficoltà viene fuori anche aprendolo con *Ghidra*. Tuttavia a supporto c'è il *Decompilatore* che, per quanto non completamente affidabile, permette di comprendere la

logica del codice. Inoltre questo dump contiene DLL ed API già caricate, quindi è possibile vedere come e quando utilizza le funzioni d'interesse.

Andando avanti con con l'analisi dinamica, si incontrano diversi punti morti che interrompono l'analisi. Staticamente è possibile vedere che sono state inserite alcune *Sleep* che, come già notato in precedenza, vengono eseguite solo a seguito di determinati controlli che certificano la presenza di un debugger. Tuttavia, essendo arrivati nel codice vero e proprio, è possibile patchare il codice per evitare di rimanere bloccati e proseguire.

Continuando quindi l'esecuzione dal debugger, con **ProcessMonitor** avviato in background, è possibile vedere che l'esecuzione non segue quella vista in precedenza.

Dei 3 processi che venivano creati in precedenza, solo *cmd.exe* rimane.

```
Process Create C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\svchost.exe Process Create C:\WINDOWS\system32\cmd.exe
```

In aggiunta viene lanciato *svchost.exe* e, filtrando le operazioni svolte da quest'ultimo, è possibile vedere che ha lo stesso comportamento del codice lanciato inizialmente. Si può quindi pensare che il malware, avendo riconosciuto di essere nel debugger e quindi sotto analisi, incarica il nuovo processo di eseguire il codice per lui per nascondersi. Crea infatti un nuovo file, chiamato per l'appunto svchost.exe, in cui copia ste stesso per poi lanciarlo.

<u></u> CreateFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	Desired Access: Generic Write, F
<u></u> CreateFile	C:\Documents and Settings\Administrator\Local Settings\Temp	SUCCESS	Desired Access: Synchronize, Di
<u></u> CloseFile	C:\Documents and Settings\Administrator\Local Settings\Temp	SUCCESS	
	eC:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	FileSystemAttributes: Case Prese
■ QueryBasicInformationFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	CreationTime: 1/2/2021 4:57:52
■QueryAttributeInformationVolum	e\Device\PrlMiniRdr\Mac\Shared\Windows\nuovo_dump_patch.copia 5.exe	SUCCESS	FileSystemAttributes: Case Prese
QueryDeviceInformationVolume	\Device\PrlMiniRdr\Mac\Shared\Windows\nuovo_dump_patch copia 5.exe	SUCCESS	DeviceType: Disk, Characteristic
SetEndOfFileInformationFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	EndOfFile: 540,672
ReadFile	\Device\PrlMiniRdr\Mac\Shared Windows\nuovo_dump_patch copia 5.exe	SUCCESS	Offset: 0, Length: 61,440
S WriteFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	Offset: 0, Length: 61,440
ReadFile	\Device\PrlMiniRdr\Mac\Shared Windows\nuovo_dump_patch copia 5.exe	SUCCESS	Offset: 61,440, Length: 61,440
S WriteFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	Offset: 61,440, Length: 61,440
ReadFile	\Device\PrlMiniRdr\Mac\Shared Windows\nuovo_dump_patch copia 5.exe	SUCCESS	Offset: 122,880, Length: 61,440
S WriteFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	Offset: 122,880, Length: 61,440
ReadFile	\Device\PrlMiniRdr\Mac\Shared Windows\nuovo_dump_patch copia 5.exe	SUCCESS	Offset: 184,320, Length: 61,440
S WriteFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	Offset: 184,320, Length: 61,440
ReadFile	\Device\PrlMiniRdr\Mac\Shared Windows\nuovo_dump_patch copia 5.exe	SUCCESS	Offset: 245,760, Length: 61,440
S WriteFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	Offset: 245,760, Length: 61,440
ReadFile	\Device\PrlMiniRdr\Mac\Shared Windows\nuovo_dump_patch copia 5.exe	SUCCESS	Offset: 307,200, Length: 61,440
S WriteFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	Offset: 307,200, Length: 61,440
ReadFile	\Device\PrlMiniRdr\Mac\Shared Windows\nuovo_dump_patch copia 5.exe	SUCCESS	Offset: 368,640, Length: 61,440
S WriteFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	Offset: 368,640, Length: 61,440
ReadFile	\Device\PrlMiniRdr\Mac\Shared Windows\nuovo_dump_patch copia 5.exe	SUCCESS	Offset: 430,080, Length: 61,440
S WriteFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	Offset: 430,080, Length: 61,440
ReadFile	\Device\PrlMiniRdr\Mac\Shared Windows\nuovo_dump_patch copia 5.exe	SUCCESS	Offset: 491,520, Length: 49,152
■ WriteFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	Offset: 491,520, Length: 49,152
■ ReadFile	\Device\PrlMiniRdr\Mac\Shared Windows\nuovo_dump_patch copia 5.exe	END OF FILE	Offset: 540,672, Length: 61,440
SetBasicInformationFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	CreationTime: 1/1/1601 1:00:00
■ CloseFile	\Device\PrlMiniRdr\Mac\Shared Windows\nuovo_dump_patch copia 5.exe	SUCCESS	
■ CloseFile	C:\Documents and Settings\Administrator\Local Settings\Temp\svchost.exe	SUCCESS	

Inoltre usando DeleteFileW su: UNICODE "C:\DOCUME"1\ADMINI"1\LOCALS"1\Temp\svchost.exe:Zone.Identifier" cancella quindi, se presenti, i metadati sulle zone di sicurezza associate al file.

Attua successivamente dei sistemi di *Privilege Escalation* manipolando il descrittore di sicurezza utilizzando le funzioni:

```
GetCurrentProcess;
```

- OpenProcessToken;
- LookupPrivilegesValueA;
- AdjustTokenPrivileges.

```
FUN_00474820("SeDebugPrivilege");
FUN_00474820("SeTakeOwnershipPrivilege");
FUN_00474820("SeBackupPrivilege");
FUN_00474820("SeRestorePrivilege");
```

```
ProcessHandle = GetCurrentProcess();

uVar1 = OpenProcessToken(ProcessHandle,DVar4,TokenHandle);

if (uVar1 != 0) {

BVar2 = LookupPrivilegeValueA((LPCSTR)0x0,param_1,(PLUID)&DAT_0048206c);

if (BVar2 != 0) {

BVar2 = AdjustTokenPrivileges

(local_8,0,(PTOKEN_PRIVILEGES)&DAT_00482068,0,(PTOKEN_PRIVILEGES)0x0,

(PDWORD)0x0);
```

Probabilmente per ulteriore sicurezza, MoveFileExW(pWVar4,(LPCWSTR)0x0,MOVEFILE_DELAY_UNTIL_REBOOT); usa su se stesso MoveFileExW con flag MOVEFILE_DELAY_UNTIL_REBOOT e senza directory di destinazione. Quindi come suggerisce la documentazione, questa istruzione permette di essere eliminato al successivo riavvio del sistema.

È stato inoltre notato che il malware non attacca il sistema

LVar3 = GetSystemDefaultLangID();
if ((((LVar3 & 0x3ff) != 0x19) && (LVar3 = GetUserDefaultLangID(), (LVar3 & 0x3ff) != 0x19)) &&
 (LVar3 = GetUserDefaultUILanguage(), (LVar3 & 0x3ff) != 0x19)) goto LAB_0042a02e;

se impostato con linguaggio russo, ovvero il valore 19 esadecimale.

Data Encryption

Tra l'elenco di funzioni trovate, risaltano quelle relative alla criptazione:

CryptAcquireContextA: Usato con CRYPT VERIFYCONTEXT. Questa opzione è

concepita per le applicazioni che utilizzano chiavi temporanee o per le applicazioni che non richiedono

l'accesso a chiavi private persistenti, come le applicazioni che

eseguono solo l'hashing, la crittografia e la verifica della

firma digitale.

CryptCreateHash : Usa CALG MD5 ed avvia l'hashing di un flusso di dati. Crea e

restituisce un handle ad un CSP (Cryptograpic Service

Provider) usato in CryptHashData.

CryptDestroyHash : Distrugge l'hash creato precedentemente.

CryptDestroyKey : Rilascia l'handle riferita al CSP distruggendo la chiave publica.

CryptEncrypt : Cripta i dati usando la chiave detenuta dal CSP.

CryptGenRandom : Utilizzata per aggiungere byte crittograficamente casuali.

CryptGetHashParam : Recupera i dati che regolano le operazioni di un hash.

CryptGetKeyParam : Recupera i dati che regolano le operazioni di una chiave.

CryptHashData : Aggiunge dati ad un hash.

CryptImportKey : Trasferisce la chiave crittografica al CSP. La chiave è esportata

come key BLOB da CryptExportKey in modo sicuro, in questo modo può essere salvata o inviata senza far trapelare

alcuna informazione.

CryptReleaseContext : Rilascia l'handle del CSP.

CryptSetHashParam : Personalizza la configurazione sull'hash.

Conclusioni

Nonostante nel malware siano presenti funzioni e settaggi per utilizzare il protocollo HTTP, dall'analisi non risulta che vengano inviati pacchetti. Non viene escluso quindi che potrebbero esserci altre funzionalità nascoste o modalità di esecuzione differenti. Non aver trovato traccia di una chiave e l'utilizzo di CryptImportKey, porta a pensare che la chiave pubblica per la criptazione asimmetrica è stata salvata nel file (in modo sicuro come detto precedentemente). Quindi non sembra esserci un sistema di generazione in loco di una chiave random (valida sia per codificare che decodificare), criptata ed inviata ad un server utilizzando criptazione asimmetrica.