

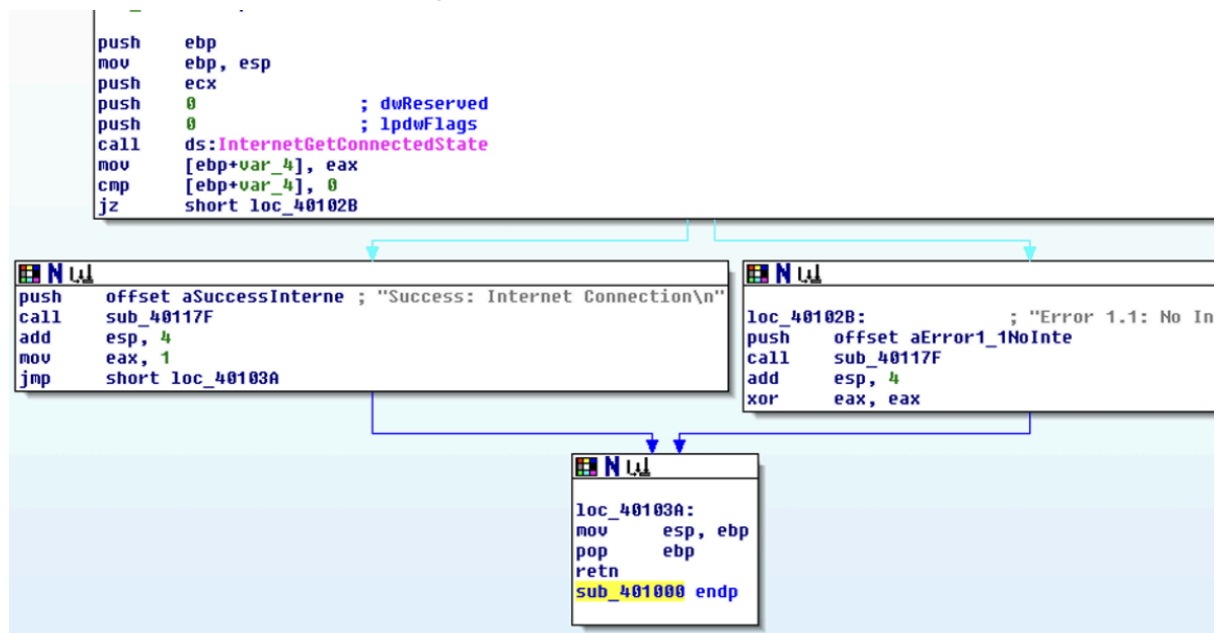
S10_L5 Assembly Hacking: Connessioni e Controllo

Traccia:

Con riferimento al file Malware_U3_W2_L5 presente all'interno della cartella «Esercizio_Pratico_U3_W2_L5 » sul desktop della macchina virtuale dedicata per l'analisi dei malware, rispondere ai seguenti quesiti:

1. Quali librerie vengono importate dal file eseguibile?
2. Quali sono le sezioni di cui si compone il file eseguibile del malware? Di entrambi dare una breve spiegazione Con riferimento alla figura in slide 3, risponde ai seguenti quesiti:
3. Identificare i costrutti noti (creazione dello stack, eventuali cicli, altri costrutti)
4. Ipotezzare il comportamento della funzionalità implementata
5. fare tabella con significato delle singole righe di codice assembly

Fase 1: Codice Assembly



Descrizione Generale:

L'immagine mostra un diagramma di flusso di codice assembly, che rappresenta una funzione per verificare la connettività Internet. Questo diagramma può essere scomposto in tre blocchi principali:

1: Blocco di Inizializzazione e Verifica Connessione Internet

```
push    ebp
mov     ebp, esp
push    ecx
push    0          ; dwReserved
push    0          ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

In questo blocco, vengono eseguite le seguenti operazioni:

- Salvataggio del base pointer corrente (push ebp) e impostazione del base pointer per il frame corrente (mov ebp, esp).
- Salvataggio del registro ecx sullo stack (push ecx).
- Preparazione dei parametri 0 per lpdwReserved e dwFlags e chiamata della funzione InternetGetConnectedState per verificare la connessione Internet.
- Salvataggio del risultato della funzione in una variabile locale (mov [ebp+var_4], eax).
- Confronto del risultato con 0. Se il risultato è zero (indicando nessuna connessione), il flusso salta al blocco di gestione del fallimento (jz short loc_40102B).

2: Blocco di Successo della Connessione

```
push    offset aSuccessInterne ; "Success: Internet Connection\n"
call    sub_40117F
add     esp, 4
mov     eax, 1
jmp     short loc_40103A
```

In questo blocco, vengono eseguite le seguenti operazioni in caso di connessione Internet attiva:

- Caricamento dell'indirizzo della stringa "Success: Internet Connection\n" sullo stack.
- Chiamata di una subroutine (sub_40117F) per gestire la stampa del messaggio di successo.
- Aggiustamento dello stack pointer (add esp, 4).
- Impostazione del registro eax a 1, probabilmente per indicare il successo.
- Salto alla fine della funzione (jmp short loc_40103A).

3: Blocco di Fallimento della Connessione

```
loc_40102B:                                ; "Error 1.1: No Internet\n"
push     offset aError1_1NoInte
call     sub_40117F
add     esp, 4
xor     eax, eax
```

In questo blocco, vengono eseguite le seguenti operazioni in caso di mancata connessione Internet:

- Caricamento dell'indirizzo della stringa "Error 1.1: No Internet\n" sullo stack.
- Chiamata della stessa subroutine (sub_40117F) per gestire la stampa del messaggio di errore.
- Aggiustamento dello stack pointer (add esp, 4).
- Azzeramento del registro eax (xor eax, eax), probabilmente per indicare un errore.

```

loc_40103A:
mov     esp, ebp
pop     ebp
retn
sub_401000 endp

```

4: Blocco di Chiusura della Funzione

In questo blocco finale, vengono eseguite le operazioni per chiudere la funzione e ripristinare il contesto precedente:

- Ripristino dello stack pointer al valore del base pointer (mov esp, ebp).
- Ripristino del base pointer precedente (pop ebp).
- Ritorno dal chiamante (retn), terminando l'esecuzione della subroutine.

Conclusioni

L'analisi del diagramma di flusso mostra un malware che verifica la connettività Internet e reagisce in base al risultato. Utilizza funzioni critiche di sistema, manipola lo stack per passare parametri e gestisce feedback condizionale (successo o fallimento). Questa struttura condizionale e adattativa è tipica dei malware sofisticati, che cercano di operare furtivamente e adattarsi alle condizioni del sistema per eseguire attività dannose.

Fase 2: Librerie

Libreria 1:

Utilizzando il tool CFF Explorer, si è scoperto che il malware richiama le seguenti librerie:

- **KERNEL32.dll**: Questa libreria contiene numerose funzioni critiche per il funzionamento del sistema operativo Windows e delle sue applicazioni. Le funzioni presenti includono la gestione della memoria, dei processi e dei thread, operazioni di input/output, gestione dei file, gestione dei tempi e delle date, nonché gestione delle risorse e altre operazioni fondamentali.
- **WININET.dll**: Questa libreria include funzioni per l'implementazione di vari protocolli di rete come HTTP, FTP e NTP. Fornisce un'API che consente alle applicazioni Windows di effettuare richieste e gestire operazioni di rete, quali il download di file da Internet, l'invio di richieste HTTP a server web e l'accesso a servizi FTP.

| Module Name | Imports | OFTs | TimeDateStamp | ForwarderChain | Name RVA | FTs (IAT) |
|--------------|--------------|----------|---------------|----------------|----------|-----------|
| szAnsi | (nFunctions) | Dword | Dword | Dword | Dword | Dword |
| KERNEL32.dll | 44 | 00006518 | 00000000 | 00000000 | 000065EC | 00006000 |
| WININET.dll | 5 | 000065CC | 00000000 | 00000000 | 00006664 | 000060B4 |

Libreria 2:

Sleep: una funzione di sistema di Windows che mette in pausa l'esecuzione di un thread per un determinato periodo di tempo, consentendo ad altri thread di essere eseguiti nel sistema durante quel periodo.

CloseHandle: un'API utilizzata per chiudere un handle (un tipo di riferimento o puntatore) a un oggetto kernel. In sostanza, questa funzione libera le risorse associate a un handle dopo che è stato utilizzato.

GetProcAddress: una funzione di libreria dinamica che permette ai programmatori di ottenere un puntatore a una funzione all'interno di una DLL caricata in memoria. Questo è spesso utilizzato quando si desidera chiamare dinamicamente una funzione esportata da una DLL durante l'esecuzione del programma, anziché linkarla staticamente durante la compilazione.

VirtualAlloc: una funzione di programmazione utilizzata principalmente su piattaforme Windows per riservare o allocare memoria virtuale per un processo. Consente ai programmatori di riservare una porzione di memoria virtuale senza necessariamente allocare memoria fisica corrispondente nello spazio di archiviazione fisica (RAM o disco rigido).

VirtualFree: una funzione dell'API di Windows utilizzata per liberare la memoria allocata dinamicamente da un processo.

| OFTs | FTs (IAT) | Hint | Name |
|----------|-----------|------|----------------|
| | | | |
| Dword | Dword | Word | szAnsi |
| 000065E4 | 000065E4 | 0296 | Sleep |
| 00006940 | 00006940 | 027C | SetStdHandle |
| 0000692E | 0000692E | 0156 | GetStringTypeW |
| 0000691C | 0000691C | 0153 | GetStringTypeA |
| 0000690C | 0000690C | 01C0 | LCMapStringW |
| 000068FC | 000068FC | 01BF | LCMapStringA |

Libreria 3:

- **InternetOpenUrl:** utilizzata per aprire una connessione Internet e recuperare il contenuto di una risorsa identificata da un URL specifico. Questa funzione è ampiamente usata nelle applicazioni Windows per scaricare dati da risorse online, come pagine web e file. In sostanza, permette alle applicazioni di effettuare richieste HTTP o FTP per ottenere contenuti da Internet.
- **InternetCloseHandle:** utilizzata per chiudere un handle di risorsa Internet precedentemente aperto o creato da altre funzioni della libreria WinINet. È fondamentale usare questa funzione per rilasciare correttamente le risorse allocate e

liberare la memoria associata, garantendo il corretto funzionamento del programma e prevenendo perdite di memoria.

- **InternetReadFile:** utilizzata per leggere dati da una risorsa Internet identificata da un handle. Consente alle applicazioni di recuperare dati da risorse online, come pagine web o file su server FTP.
- **InternetGetConnectedState:** una funzione utilizzata per determinare lo stato della connessione di rete del sistema. Consente alle applicazioni di verificare se il sistema è connesso a Internet o a una rete locale.
- **InternetOpen:** crea un handle che rappresenta una sessione di comunicazione, fornendo un punto di ingresso per l'accesso alla rete. Tramite questa sessione, l'applicazione può effettuare operazioni di rete come il download di risorse da Internet, l'invio di richieste HTTP e altre attività di comunicazione.

| OFTs | FTs (IAT) | Hint | Name |
|----------|-----------|------|---------------------------|
| Dword | Dword | Word | szAnsi |
| 00006640 | 00006640 | 0071 | InternetOpenUrlA |
| 0000662A | 0000662A | 0056 | InternetCloseHandle |
| 00006616 | 00006616 | 0077 | InternetReadFile |
| 000065FA | 000065FA | 0066 | InternetGetConnectedState |
| 00006654 | 00006654 | 006F | InternetOpenA |

Fase 3: Composizione Malware

| Name | Virtual Size | Virtual Address | Raw Size | Raw Address | Reloc Address | Linenumbers | Relocations N... | Linenumbers ... | Characteristics |
|---------|--------------|-----------------|----------|-------------|---------------|-------------|------------------|-----------------|-----------------|
| Byte[8] | Dword | Dword | Dword | Dword | Dword | Dword | Word | Word | Dword |
| .text | 00004A78 | 00001000 | 00005000 | 00001000 | 00000000 | 00000000 | 0000 | 0000 | 60000020 |
| .rdata | 0000095E | 00006000 | 00001000 | 00006000 | 00000000 | 00000000 | 0000 | 0000 | 40000040 |
| .data | 00003F08 | 00007000 | 00003000 | 00007000 | 00000000 | 00000000 | 0000 | 0000 | C0000040 |

.text: questa sezione contiene le istruzioni (le righe di codice) che la CPU eseguirà quando il software viene avviato. In genere, è l'unica sezione di un file eseguibile che viene eseguita dalla CPU, poiché tutte le altre sezioni contengono dati o informazioni di supporto.

.rdata: include generalmente le informazioni riguardanti le librerie e le funzioni importate ed esportate dall'eseguibile, informazioni che, come abbiamo visto, possono essere ottenute tramite CFF Explorer.

.data: contiene tipicamente i dati o le variabili globali del programma eseguibile, che devono essere accessibili da qualsiasi parte del programma. Una variabile è considerata globale quando non è definita all'interno del contesto di una funzione, ma è dichiarata globalmente e quindi accessibile da qualsiasi funzione all'interno dell'eseguibile.

Fase 4: Analisi costrutti noti

Immagine1 :

```
push    ebp
mov     ebp, esp
push    ecx
push    0          ; dwReserved
push    0          ; lpdwFlags
call    ds:InternetGetConnectedState
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_40102B
```

Il costrutto assembly mostrato nell'immagine rappresenta una parte di una funzione che verifica la connettività Internet

Istruzioni di Setup:

```
push    ebp
mov     ebp, esp
push    ecx
```

push ebp: Salva il valore del registro ebp (base pointer) sullo stack. Questo è fatto per preservare il contesto del chiamante.

mov ebp, esp: Imposta ebp all'attuale valore di esp (stack pointer), creando un nuovo frame dello stack per la funzione corrente.

push ecx: Salva il registro ecx sullo stack. Questo potrebbe essere fatto per preservare il valore di ecx o per preparare spazio sullo stack per variabili locali.

Preparazione e Chiamata della Funzione

```
push    0          ; lpdwReserved
push    0          ; dwFlags
call    ds:InternetGetConnectedState
```

push 0 (dwReserved): Passa 0 come argomento per dwReserved. Questo è un parametro riservato e non utilizzato nella chiamata alla funzione InternetGetConnectedState.

push 0 (dwFlags): Passa 0 come argomento per dwFlags. Questo indica che non ci sono flag specifici per la chiamata.

call ds:InternetGetConnectedState: Chiama la funzione InternetGetConnectedState attraverso l'offset nel segmento dei dati (ds). Questa funzione verifica se il sistema è connesso a Internet.

Elaborazione del Risultato

```
mov [ebp+var_4], eax
cmp [ebp+var_4], 0
jz short loc_40102B
```

mov [ebp+var_4], eax: Salva il valore di ritorno della funzione InternetGetConnectedState (che si trova nel registro eax) in una variabile locale [ebp+var_4].

cmp [ebp+var_4], 0: Confronta il valore della variabile locale [ebp+var_4] con 0.

jz short loc_40102B: Se il confronto risulta in zero (indicando che non c'è connessione a Internet), il flusso del programma salta all'etichetta loc_40102B, che gestisce il caso di errore o mancata connessione.

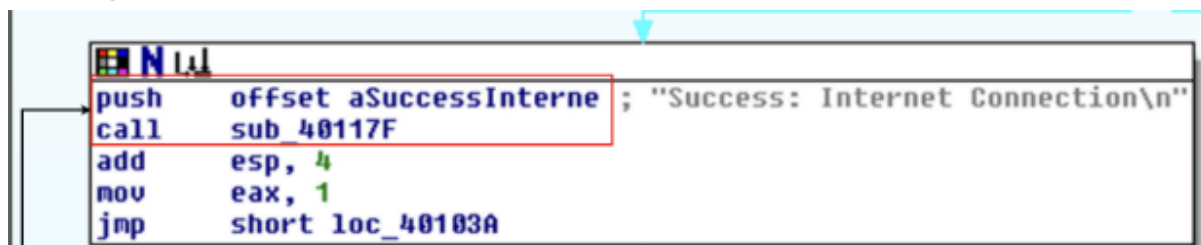
Significato Complessivo

Questo costrutto assembly prepara lo stack e chiama la funzione InternetGetConnectedState per verificare la connessione a Internet. Successivamente, il risultato della chiamata viene salvato in una variabile locale e confrontato con 0. Se il sistema non è connesso a Internet (eax = 0), il flusso del programma viene diretto a gestire la situazione di errore o di mancanza di connessione.

Applicazioni nel Contesto Malware

Nel contesto dell'analisi del malware, questo costrutto potrebbe essere usato per determinare se il malware può comunicare con un server di comando e controllo (C&C). Se non c'è connessione a Internet, il malware potrebbe decidere di terminare la propria esecuzione o di intraprendere altre azioni alternative per evitare il rilevamento.

Immagine2 :



Il costrutto assembly nell'immagine mostra una sequenza di istruzioni che gestisce il caso di successo nella verifica della connessione Internet.

1. push offset aSuccessInternetC:

- Questa istruzione spinge l'offset della stringa "Success: Internet Connection\n" sullo stack. La stringa è probabilmente definita in un segmento dati del programma e viene usata per fornire un feedback di successo.

2. call sub_40117F:

- Chiama una subroutine situata all'indirizzo 40117F. Questa subroutine potrebbe essere responsabile della gestione e visualizzazione del messaggio di successo, come la stampa su una console o la registrazione in un log.

3. **add esp, 4:**

- Aggiusta il puntatore dello stack (esp) incrementandolo di 4 byte per ripulire lo stack dal parametro passato alla subroutine (offset aSuccessInternetC). Questo passo è necessario per mantenere lo stack bilanciato.

4. **mov eax, 1:**

- Imposta il registro eax a 1. Questo valore potrebbe essere usato per indicare il successo della verifica della connessione Internet. Nei programmi, eax è spesso usato per restituire valori di stato o risultati delle funzioni.

5. **jmp short loc_40103A:**

- Salta all'etichetta loc_40103A, che si trova alla fine della funzione. Questo salto evita l'esecuzione delle istruzioni successive non rilevanti per il caso di successo e termina la gestione della connessione.

Significato Complessivo

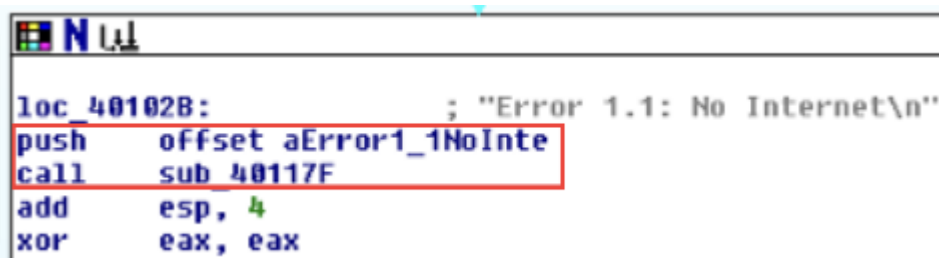
Questo blocco di codice gestisce il caso in cui la connessione a Internet è stata verificata con successo:

- Passa e stampa un messaggio di successo.
- Ripristina lo stack dopo la chiamata della subroutine.
- Imposta un valore di successo nel registro eax.
- Salta alla fine della funzione per terminare l'esecuzione.

Applicazioni nel Contesto Malware

Nel contesto dell'analisi del malware, questa sezione di codice potrebbe essere usata per confermare che il sistema è connesso a Internet e pronto per ulteriori azioni dannose. La stampa del messaggio di successo può essere usata per il debug durante lo sviluppo del malware o per fornire informazioni di stato ai creatori del malware quando questo viene eseguito in ambienti controllati o di test.

Immagine3 :



Il costrutto assembly mostrato nell'immagine rappresenta la gestione del caso di errore, ovvero quando non è presente una connessione Internet.

1. **loc_40102B::**

- Questa è un'etichetta che indica l'inizio del blocco di codice per la gestione dell'errore. L'etichetta loc_40102B viene usata come punto di salto in caso di errore nella connessione Internet.

2. **push offset aError1_1NoInte:**

- Questa istruzione spinge l'offset della stringa "Error 1.1: No Internet\n" sullo stack. Questa stringa è probabilmente definita in un segmento dati del programma e viene usata per fornire un messaggio di errore.

3. **call sub_40117F:**

- Chiama una subroutine situata all'indirizzo 40117F. Questa subroutine potrebbe essere responsabile della gestione e visualizzazione del messaggio di errore, come la stampa su una console o la registrazione in un log.

4. **add esp, 4:**

- Aggiusta il puntatore dello stack (esp) incrementandolo di 4 byte per ripulire lo stack dal parametro passato alla subroutine (offset aError1_1NoInte). Questo passo è necessario per mantenere lo stack bilanciato.

5. **xor eax, eax:**

- Azzeramento del registro eax utilizzando l'operazione XOR. xor eax, eax è un modo efficiente per impostare il valore di eax a 0. Questo valore potrebbe essere usato per indicare il fallimento della verifica della connessione Internet.

Significato Complessivo

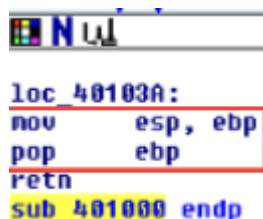
Questo blocco di codice gestisce il caso in cui la connessione a Internet non è stata verificata con successo:

- Passa e stampa un messaggio di errore.
- Ripristina lo stack dopo la chiamata della subroutine.
- Imposta il registro eax a 0 per indicare il fallimento.
- Questo flusso di codice potrebbe continuare con altre operazioni di gestione dell'errore, ma non sono visibili in questa porzione di codice.

Applicazioni nel Contesto Malware

Nel contesto dell'analisi del malware, questa sezione di codice potrebbe essere usata per notificare al malware che non è possibile connettersi a Internet. La stampa del messaggio di errore può essere utile per il debug durante lo sviluppo del malware o per fornire informazioni di stato ai creatori del malware quando questo viene eseguito in ambienti controllati o di test. Azzerare il registro eax è una pratica comune per indicare che un'operazione non ha avuto successo, e in questo contesto, potrebbe interrompere ulteriori azioni dannose che richiedono una connessione a Internet.

Immagine4 :



```
loc_40103A:  
mov     esp, ebp  
pop     ebp  
retn  
sub_401000 endp
```

L'immagine mostra una sequenza di codice assembly che gestisce il ripristino dello stack e la terminazione della funzione.

Dettagli del Codice

1. **mov esp, ebp:**

- Questa istruzione ripristina il valore originale del puntatore dello stack (esp) copiando il valore dal registro ebp. Durante l'entrata nella funzione, ebp viene utilizzato per salvare il frame base della funzione chiamante, e ora serve a ripristinare il frame precedente.

2. pop ebp:

- Questa istruzione ripristina il valore originale del registro ebp prelevandolo dallo stack. È l'inverso dell'istruzione push ebp che si trova all'inizio della funzione.

3. retn:

- Questa istruzione termina la funzione e ritorna il controllo al chiamante. Essa preleva l'indirizzo di ritorno dallo stack e salta a tale indirizzo, continuando l'esecuzione del programma chiamante.

Funzione dell'Epilogo

L'epilogo della funzione è un insieme di istruzioni standard che viene utilizzato per:

- Ripristinare il frame precedente della funzione chiamante, assicurando che il contesto del chiamante sia integro.
- Ripulire lo stack per evitare fughe di memoria o corruzione dello stack.
- Rilasciare il controllo al chiamante, permettendo al programma di continuare la sua esecuzione normale.

Importanza nell'Analisi del Malware

Nell'analisi del malware, è fondamentale comprendere queste istruzioni per capire come il malware gestisce le chiamate di funzione e il contesto del sistema. Le sequenze di epilogo standard possono essere modificate da malware per nascondere il comportamento dannoso o per manipolare il flusso di controllo del programma. Capire come queste operazioni avvengono aiuta gli analisti a rilevare e decodificare il comportamento del malware.

Fase Finale: Analisi Codice

| | | | |
|----|-----------------|----------------------------|---|
| 1 | push | ebp | Il contenuto del registro ebp viene caricato sullo stack |
| 2 | mov | ebp, esp | Sposto il contenuto di esp in ebp |
| 3 | push | ecx | Il contenuto del registro ecx viene caricato sullo stack |
| 4 | push | 0 ;dwReserved | Carico il valore 0 sullo stack * |
| 5 | push | 0 ;lpdwFlags | Carico il valore 0 sullo stack ** |
| 6 | call | ds:InternetGetConnectState | Viene chiamata la funzione InternetGetConnectState *** |
| 7 | mov | [ebp+var_4], eax | Sposto il contenuto del registro eax nella variabile locale [ebp+var_4] |
| 8 | cmp | [ebp+var_4], 0 | Viene comparata la variabile locale con 0 |
| 9 | jz | short loc_40102B | Se l'istruzione precedente ha impostato lo ZeroFlag a 1 allora viene effettuato il salto alla locazione indicata, altrimenti continuo l'esecuzione |
| 10 | push | offset aSuccessInterne | Viene caricata la variabile aSuccessInterne di tipo string nello stack |
| 11 | call | sub_40117F | Viene chiamata la funzione all'indirizzo 40117F |
| 12 | add | esp, 4 | Il contenuto del registro esp viene sommato con 4 ed il risultato memorizzato nel medesimo registro. Così facendo libero lo stack dalla push effettuata a riga 10 |
| 13 | mov | eax, 1 | Viene salvato il valore 1 all'interno del registro eax |
| 14 | jmp | short loc_40103A | Viene effettuato un salto incondizionato alla locazione di memoria 40103A, per saltare la porzione di codice che viene eseguita nel caso del salto a riga 9 |
| 15 | loc_40102B | | E' un flag che sta a sottolineare che mi trovo alla locazione 40102B |
| 16 | push | offset aError1_1NoInte | Viene caricata la variabile aError1_1NoInte di tipo string nello stack |
| 17 | call | sub_40117F | Viene chiamata la funzione all'indirizzo 40117F |
| 18 | add | esp, 4 | Il contenuto del registro esp viene sommato con 4 ed il risultato memorizzato nel medesimo registro. Così facendo libero lo stack dalla push effettuata a riga 16 |
| 19 | xor | eax, eax | Viene effettuata l'operazione logica di xor sul contenuto di eax inizializzando a 0 |
| 20 | loc_40103A | | E' un flag che sta a sottolineare che mi trovo alla locazione 40103A |
| 21 | mov | esp, ebp | Viene spostato il contenuto del registro ebp all'interno del registro esp |
| 22 | pop | ebp | Rimuove il contenuto della locazione indicizzata a ebp dallo stack |
| 23 | retn | | Terminare una subroutine o una funzione e ritorna al punto di chiamata |
| 24 | sub_401000 endp | | Viene utilizzato per delimitare il corpo di una subroutine e segnalarne la fine al compilatore o all'assemblatore **** |

Conclusioni Finali e Motivo dell'Analisi

Questa analisi è stata eseguita per approfondire la comprensione delle tecniche di programmazione in assembly e delle metodologie utilizzate per la gestione delle connessioni Internet. Comprendere questi aspetti è cruciale per chi lavora nell'ambito della sicurezza informatica e dell'ethical hacking. In particolare, la conoscenza di come i programmi gestiscono le connessioni e l'epilogo delle funzioni può aiutare a:

- Identificare e analizzare il comportamento dei malware.
- Migliorare le tecniche di rilevamento e prevenzione delle minacce informatiche.
- Sviluppare software sicuro e robusto.

L'analisi del codice assembly, la comprensione delle funzioni di sistema e la gestione delle connessioni Internet sono solo una piccola parte del vasto universo dell'ethical hacking. Queste competenze non solo migliorano la nostra capacità di difesa contro le minacce informatiche, ma rafforzano anche la nostra capacità di costruire un futuro digitale più sicuro.

In conclusione; l'analisi del codice assembly e la comprensione della gestione delle connessioni Internet sono fondamentali per chiunque voglia eccellere nel campo della sicurezza informatica. Grazie a questa analisi, possiamo migliorare le nostre tecniche di rilevamento e difesa contro i malware, contribuendo a un ambiente digitale più sicuro per tutti.