

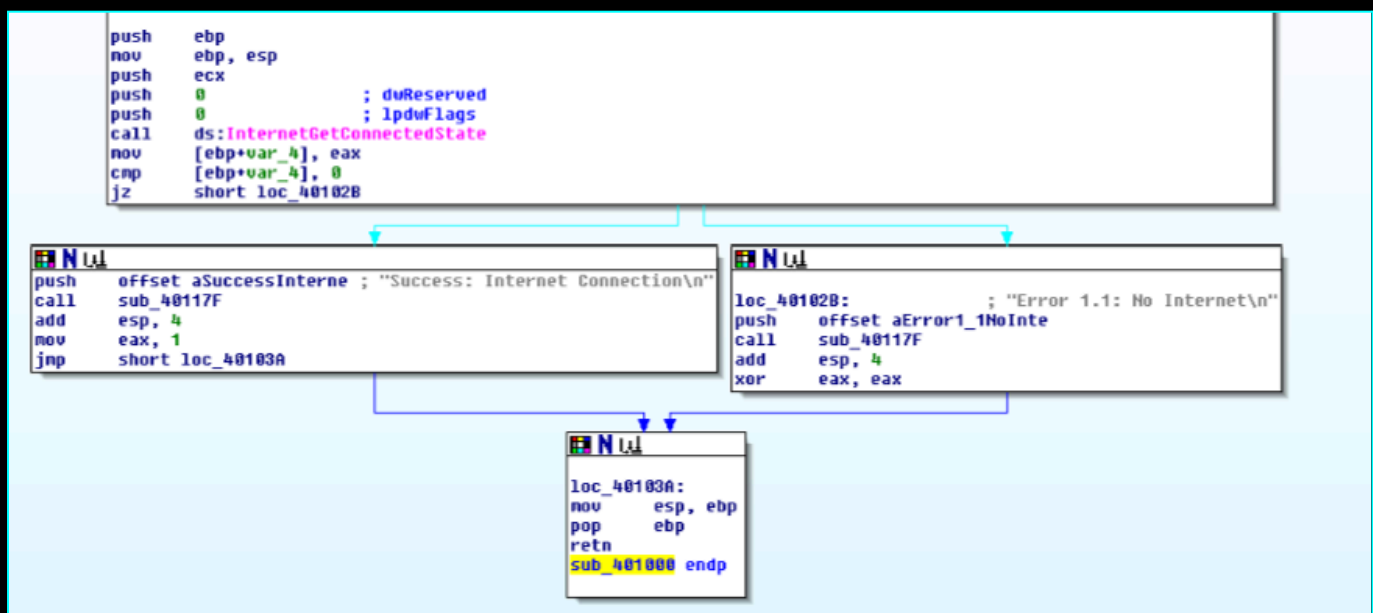
Malware analysis



Richiesta: Dato il Malware_U3_W2_L5 all'interno della VM windows 7, analizzarlo e 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?

Con riferimento al seguente estratto di codice in assembly, un linguaggio di programmazione a basso livello che comunica direttamente con l'hardware tramite istruzioni specifiche della CPU:



Rispondere ai seguenti quesiti:

- 1. Identificare i costrutti noti.
- 2. Ipotizzare il comportamento della funzionalità implementata.
- 5. BONUS: Spiegare ogni singola riga di codice.

Analisi malware

Effettuiamo l'analisi del malware attraverso CFF explorer, uno strumento software capace di analizzare file eseguibili in formato PE (.exe, .dll).

Verifichiamo le librerie importate:

Malware_U3_W2_L5.exe					
Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC
WININET.dll	5	000065CC	00000000	00000000	00006664

KERNEL32.dll: Una delle librerie fondamentali di Windows. Fornisce funzioni di base per la gestione della memoria, la creazione e la gestione dei processi e dei thread, e altre funzioni del sistema operativo.

WININET.dll: Libreria che gestisce funzioni relative ad internet, Fornisce API per la comunicazione con protocolli come HTTP, FTP e HTTPS, implementando funzioni come la navigazione web, il download e l'upload di file, la gestione delle sessioni e dei cookie ecc...

Adesso effettuiamo l'analisi delle sezioni del malware, ovvero parti del file che contengono diversi tipi di dati e codice. Queste sezioni organizzano l'eseguibile in blocchi con specifici scopi e permessi.

Sezioni riportate:

Malware_U3_W2_L5.exe						
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbr
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword
.text	00004A78	00001000	00005000	00001000	00000000	00000000
.rdata	0000095E	00006000	00001000	00006000	00000000	00000000
.data	00003F08	00007000	00003000	00007000	00000000	00000000

.text: Contiene il codice eseguibile del programma. È dove risiedono le istruzioni che la CPU esegue.

.rdata: 'read-only data', questa sezione contiene dati costanti utilizzati dal programma, come stringhe e import tables. È marcata come leggibile e non scrivibile.

.data: Questa sezione contiene dati inizializzati, il che significa variabili globali e statiche che sono inizializzate dal programmatore.

Costrutti noti

L'estratto di codice assembly corrisponde a una struttura condizionale che verifica se c'è una connessione a Internet e agisce di conseguenza. Questa strutta è comparabile al costrutto if-else.

Parte di codice interessata:

Nel contesto di un costrutto if-else, la parte "if" è implicita; *se il risultato del cmp non è zero*, il codice dopo cmp e prima di jz viene eseguito (quindi il flusso di esecuzione continua linearmente). *Se il risultato è zero*, il flusso di esecuzione salta a loc_40182B, che rappresenta la parte "else".

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

Parte corrispondente a if:

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

Se il risultato del cmp non è 0, il codice segue il suo flusso lineare, mostrando un messaggio in cui fa riferimento che il sistema è connesso a internet.

Parte corrispondente a else:

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

Quando l'istruzione jz viene eseguita, se il flag zero è impostato (cioè se il risultato del cmp è zero), il programma salterà all'indirizzo etichettato come loc_40182B.

Qui, viene mostrato un messaggio di errore (No Internet), viene chiamata un'altra subroutine (per gestire l'errore o stamparlo) e vengono effettuate delle operazioni di pulizia.

Esempio del costrutto ad alto livello in C:

```
1
2 int main() {
3     // Presumiamo che InternetGetConnectedState ritorni 1 per "connesso" e 0 per "non connesso"
4     int isConnected = InternetGetConnectedState();
5
6     if (isConnected) {
7         printf("Success: Internet Connection\n");
8         // operazioni in caso di connessione
9     } else {
10        printf("Error 1.1: No Internet\n");
11        // operazioni in caso di assenza di connessione
12    }
13
14 }
```

Comportamento funzionalità implementata

Questo estratto di codice è un esempio di come un malware verifica la presenza di una connessione a Internet prima di procedere con ulteriori azioni.

La gestione dei casi di successo e di errore suggerisce che il malware è progettato per operare in maniera condizionale a seconda dello stato della connessione internet della macchina vittima, in caso di successo, il programma continua con il suo normale andamento, al contrario invece se fallisce.

Bonus: Spiegazione codice

Flusso principale:

```
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
```

1. push ebp: Salva il base pointer attuale nello stack per preservare il contesto della funzione chiamante.

2. mov ebp, esp: Imposta il base pointer al valore dello stack pointer per preparare il frame dello stack della funzione corrente.

3. push ecx: Salva il registro ecx (Extended Counter Register) nello stack. Potrebbe essere utilizzato per preservare un valore attraverso le chiamate di funzione.

4. push 0: Mette un valore zero nello stack, si tratta di un valore per la funzione InternetGetConnectedState.

5. call ds:InternetGetConnectedState: Chiama la funzione InternetGetConnectedState (presa da l'API di windows). Il segmento ds indica che l'indirizzo della funzione è preso dal segmento dati.

6. mov [ebp+var_4], eax: Memorizza il valore restituito dalla funzione (in eax) in una variabile locale nello stack (var_4).

7. cmp [ebp+var_4], 0: Confronta il valore nella variabile locale con 0 per vedere se c'è una connessione internet.

8. jz short loc_40182B: Salta all'etichetta loc_40182B se il risultato del confronto è zero (se non c'è connessione internet).

Flusso internet connesso

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

1. push offset aSuccessInterne: Mette l'indirizzo della stringa "Success: Internet Connection\n" nello stack, come argomento per la successiva chiamata alla funzione.

2. call sub_40117F: Chiama una subroutine, per stampare il messaggio o eseguire altre operazioni legate al successo della connessione.

3. **add esp, 4**: Pulisce lo stack di 4 byte, che è la dimensione dell'argomento pushato precedentemente (la stringa).

4. **mov eax, 1**: Imposta il valore di eax a 1, che potrebbe indicare il successo dell'operazione.

5. **jmp short loc_40183A**: Effettua un salto incondizionato all'etichetta loc_40183A per continuare l'esecuzione dopo il blocco condizionale.

Flusso internet non connesso:



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

1. **push offset aError1_1NoInte**: Mette l'indirizzo della stringa "Error 1.1: No Internet\n" nello stack, come argomento per la successiva chiamata alla funzione.

2. **call sub_40117F**: Chiama una subroutine, per stampare il messaggio o eseguire altre operazioni legate al fallimento della connessione.

3. **add esp, 4**: Pulisce lo stack di 4 byte.

4. **xor eax, eax**: Azzerà il registro eax (mette eax a zero), che potrebbe indicare il fallimento dell'operazione.

Flusso finale:

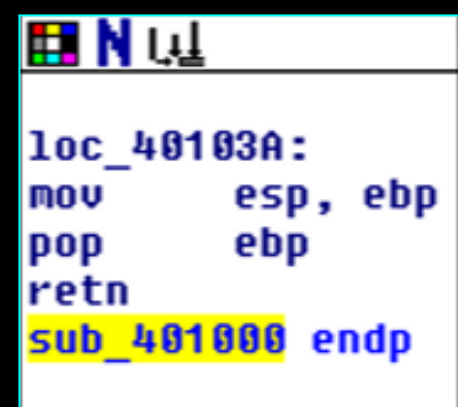
1. **loc_40183A::** Etichetta usata per il salto incondizionato; potrebbe segnare l'inizio del codice successivo al blocco condizionale.

2. **mov esp, ebp**: Ripristina lo stack pointer al valore del base pointer, preparandosi a uscire dalla funzione.

3. **pop ebp**: Ripristina il valore originale di ebp dallo stack.

4. **retn**: Ritorna dalla subroutine. La parola chiave retn prende il valore dall'inizio dello stack (l'indirizzo di ritorno) e salta a quell'indirizzo, continuando l'esecuzione dalla funzione chiamante.

5. **sub_401000 endp**: Questa è un'etichetta che indica la fine di una procedura (subroutine) iniziata con sub_401000. Il endp sta per "end procedure"



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

