EPICODE-CS0124
S10/L5 - Pratica

Flaviano Sedici

Indice

Riferimenti e versioni

Responsabile del documento: Flaviano Sedici

Versionamento

Versione	Descrizione	Riferimento	Data
1.0	Redazione documento	Responsabile	29/03/2024

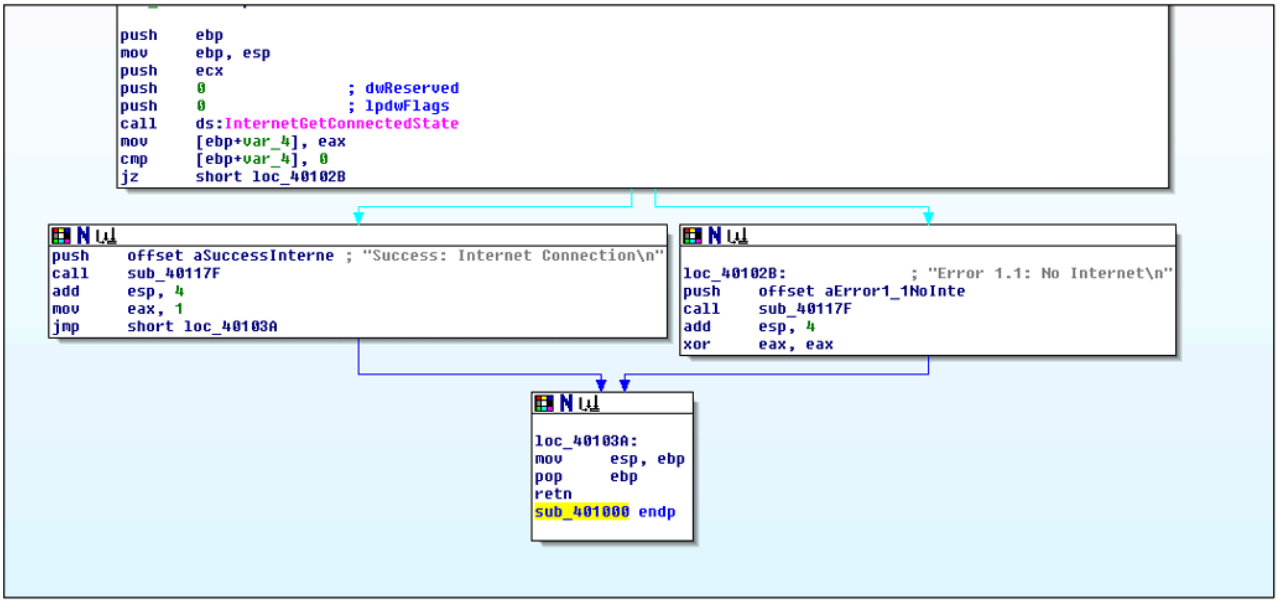
1. Traccia

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:

- Quali **librerie** vengono importate dal file eseguibile?
- Quali sono le **sezioni** di cui si compone il file eseguibile del malware?

Con riferimento alla figura in slide 3, risponde ai seguenti quesiti:

- Identificare i **costrutti** noti (creazione dello stack, eventuali cicli, altri costrutti).
- **Ipotizzare il comportamento della funzionalità implementata.**
- BONUS fare tabella con significato delle singole righe di codice assembly.



Definizioni

Librerie: le Dynamic Link Libraries (DLL) sono file contenenti codice eseguibile e dati, progettati per essere utilizzati da più programmi contemporaneamente.

Sezioni: le sezioni all'interno di un file eseguibile sono blocchi di dati che contengono informazioni specifiche per il caricamento e l'esecuzione del programma.

1.1. Librerie importate

1.1.1. KERNEL32.dll

Libreria di sistema di Windows che fornisce funzioni essenziali per la gestione della memoria, la gestione dei file, l'accesso ai dispositivi hardware e altre operazioni di basso livello. È fondamentale per il funzionamento stabile del sistema operativo Windows.

Vengono invocate 44 funzioni.

1.1.2. WININET.dll

Libreria di sistema di Windows che fornisce funzionalità per l'accesso a Internet dalle applicazioni Windows. Include funzioni per la gestione delle connessioni HTTP, HTTPS e FTP, l'invio e la ricezione di dati su Internet e altre operazioni di rete. È essenziale per le applicazioni web e Internet.

Vengono importate 5 funzioni.

1.2. Descrizione sezioni

1.2.1. .text

La sezione contiene il codice eseguibile del malware.

Come previsto analizzandone il contenuto non si riesce a rintracciare alcuna riga di codice scritta in chiaro.

1.2.2. .rdata

La sezione contiene le librerie o le chiamate alle librerie e alle funzioni.

Riportiamo di seguito alcune parti del codice che riteniamo siano interessanti per l'analisi del malware.

```
__GLOBAL_HEAP_SELECTED
__MSVCRT_HEAP_SELECT
[...]
HeapDestroy
HeapCreate
VirtualFree
HeapFree
[...]
VirtualAlloc
HeapReAlloc
```

Questi codici potrebbero essere utilizzati internamente dal runtime di C della libreria Microsoft (MSVCRT) per gestire l'allocazione e la deallocazione della memoria nella heap globale o

selezionare il tipo di heap da utilizzare durante l'esecuzione del programma. La libreria MSVCRT non sembra però essere richiamata direttamente nelle librerie elencate precedentemente.

Ne deduciamo che probabilmente il malware sta cercando di inserire del codice eseguibile nella memoria Heap.

user32.dll

E' una libreria di sistema di Windows che fornisce funzionalità per la creazione e la gestione dell'interfaccia utente grafica (GUI).

InternetGetConnectedState
InternetReadFile
InternetCloseHandle
InternetOpenUrlA
InternetOpenA

Con questi comandi il malware presumibilmente tenta di gestire una connessione ad internet utilizzandola per accedere a degli URL. Potrebbe quindi scaricare nuovi componenti dannosi o connettersi ad un server per operazioni di Command & Control[1].

GetProcAddress
LoadLibraryA

Con questi comandi solitamente il malware cerca di caricare delle librerie a tempo di esecuzione (runtime), ovvero fa caricare le librerie al sistema operativo solo quando richiesto dal codice.

LCMapStringA
LCMapStringW
GetStringTypeA
GetStringTypeW
SetStdHandle
CloseHandle

Con questi comandi sembra identificare la mappatura per la conversione dei caratteri, il reperimento e la gestione delle stringhe.

1.2.3. .data

La sezione contiene le variabili globali.

Error 1.1: No Internet

[1] *Command & Control (C2 o C&C) è un'infrastruttura utilizzata da attaccanti o malware per gestire e controllare le operazioni di hacking o attività malevole, consentendo loro di inviare comandi, ricevere dati e coordinare le azioni degli agenti di malware o risorse compromesse.*

Success: Internet Connection

Error 2.3: Fail to get command

Error 2.2: Fail to ReadFile

Error 2.1: Fail to OpenUrl

http://www.practicalmalwareanalysis.com/cc.htm

Le variabili globali gestiscono evidentemente degli errori e il link al portale:

http://www.practicalmalwareanalysis.com/cc.htm

1.2.4. Considerazioni finali e analisi Virus Total

Ad una analisi superficiale e un confronto dell'hash su Virus Total, il malware potrebbe essere associato ad un Trojan, con capacità di creare/caricare una backdoor o un keylogger.

1.3. Identificazione dei costrutti

1.3.1. Creazione di Stack

Creazione di uno stack

rappresenta la creazione di uno stack. Il comando **push** inserisce un nuovo stack sopra il pointer **EBP**, poi con il comando **mov** il valore del pointer **ESP** viene copiato nel pointer **EBP**. Non viene però creato un spazio preallocato tramite il comando **sub ESP, 0xYY** per le variabili locali.

1.3.2. Call

Costrutto Call

Nelle prime tre righe, tramite il comando **push**, vengono caricati 3 valori che serviranno per il corretto funzionamento della chiamata di della funzione **InternetGetConnectedState**[2] che solitamente prevede il passaggio di soli due parametri per verificare lo stato della connessione ad internet, ma in questo caso, avendone passati tre, possiamo aspettarci che ci restituirà un valore booleano per comunicarci se il sistema è connesso ad internet e in quale modo è connesso (LAN, modem, etc.). La call dovrebbe ripulire i parametri aggiunti allo stack tramite i push in autonomia identificandola come **STDCALL**.

1.3.3. IF

Notiamo come a seguito della funzione **cmp**, ci sia una funzione **jz (Jump Zero)** che salta alla riga indicata se e solo se i due parametri confrontati sono uguali tra loro (ovvero ZF=1)

---

[2] *https://learn.microsoft.com/en-us/windows/win32/api/wininet/nf-wininet-internetgetconnectedstate*

La riga a cui il programma salta è contrassegnata dalla locazione **loc_40102B**.

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

Costrutto IF

1.3.4. Call

```
push    offset aError1_1NoInte
call    sub_40117F
add     esp, 4
```

Costrutto Call

Il costrutto effettua un passaggio di parametro alla funzione tramite il **push** poi invoca una **subroutine 40117F** eliminando il dato aggiunto allo stack tramite il push appena effettuato (**add esp,4**).

1.3.5. Call

```
push    offset aSuccessInterne ; "Suc
call    sub_40117F
add     esp, 4
```

Costrutto Call

Il costrutto effettua un passaggio di parametro alla funzione tramite il **push** poi invoca una **subroutine 40117F** eliminando il dato aggiunto allo stack tramite il push appena effettuato (**add esp,4**).

1.3.6. Eliminazione dello stack

```
mov     esp, ebp
pop     ebp
```

Eliminazione stack

Il codice elimina lo **stack** creato all'inizio dell'esecuzione del programma.

1.4. Ipotesi funzionamento

Il codice sembra appartenere ad una funzione/subroutine chiamata **sub_401000** che implementa la verifica della connettività ad internet del sistema infettato restituendo anche possibili informazioni sul tipo di connettività presente (LAN, modem, etc.).

Se il sistema è connesso ad internet chiama la **sub_40117F** con un parametro di "Controllo avvenuto con successo" e poi imposta il valore di EAX a 1.

Al contrario qualora non ci sia una connessione ad internet, passa questa informazione alla stessa subroutine **sub_40117F** impostando poi il valore di EAX a 0.

Riassumendo, questa subroutine effettua un controllo sulla connessione ad internet e restituendo il risultato booleano 1 o 0 a seconda se sia o non sia correttamente funzionante, riportando anche i valori: **Success: Internet Connection** o **Error 1.1: No Internet**

1.5. Tabella codice assembly

Push ebp	effettua il push di un nuovo stack sopra EBP
Mov ebp, esp	sposta il valore di ESP in EBP
Push ecx	effettua il push nello stack del valore contenuto in ECX
Push 0 ;dwReserved	effettua il push nello stack del valore immediato 0
Push 0 ;lpdwFlags	effettua il push nello stack del valore immediato 0
Call ds:InternetGetConnectedState	effettua la call della funzione GetConnectedState passando i tre parametri precedentemente inviati tramite i push
Mov [ebp+var_4], eax	muove il valore di EAX sulla variabile ebp+var_4
Cmp [ebp+var_4], 0	effettua il compare tra il valore 0 e il valore contenuto in ebp+var_4
Jz short loc_40102B	Jump Zero, salta alla locazione indicata se i due valori comparati precedentemente sono uguali (ovvero ZF = 1)
Push offest aSuccessInterne ; “Success: Internet Connection\n”	effettua il push nello stack del valore “Success: Internet Connection”
Call sub_40117F	effettua la call della subroutine sub_40117F
Add esp, 4	aggiunge 4 bytes al registro ESP al fine di eliminare il push offset precedente
Mov eax, 1	muove il valore 1 nel registro EAX
Jmp short loc_40103A	salta incondizionatamente alla locazione di memoria indicata
loc_40102B: ; “Error 1.1 : No Internet\n”	indica la locazione in cui il programma salta nel caso in cui, a valle della comparazione precedente, ZF sia pari a 1
Push offset aError1_1NoInte	effettua il push del valore da passare alla funzione/subroutine successiva
Call Sub_40117F	effettua il call della subroutine indicata passando il parametro precedentemente inviato tramite il push
Add esp, 4	aggiunge 4 bytes al registro ESP al fine di eliminare il push offset precedente
Xor eax, eax	effettua un operazione XOR che compara i due valori EAX e EAX trasferendo il risultato (in questo caso 0/FALSE) nel valore EAX.
Loc_40103A:	identifica una locazione di memoria
Mov esp, ebp	muove il valore di EBP in ESP
Pop ebp	effettua il pop dello stack sopra EBP
Retn	termina una subroutine e tornare al punto di chiamata della subroutine
sub_401000 endp	indica la fine della subroutine