




UNITÀ 3 SETTIMANA 2 COMPITO 5



LIBRERIE

1) `KERNEL32.DLL` è una libreria di sistema essenziale nei sistemi operativi Microsoft Windows. Questa DLL (Dynamic Link Library) contiene una vasta gamma di funzioni di basso livello che sono utilizzate dai programmi in esecuzione su piattaforme Windows. Le funzioni contenute in `KERNEL32.DLL` includono operazioni di gestione della memoria, gestione dei file, gestione dei processi e delle thread, gestione delle risorse, gestione degli errori e altre operazioni di sistema di base.

Gli sviluppatori di software Windows spesso fanno riferimento a questa libreria per eseguire operazioni di sistema essenziali, come la creazione di file, l'allocazione di memoria, la sincronizzazione delle thread e altro ancora. `KERNEL32.DLL` è parte integrante del sistema operativo Windows e viene caricato in memoria quando il sistema operativo avvia un'applicazione Windows



2) **WS2_32.dll** è una libreria di sistema in ambienti Windows che fornisce l'implementazione delle API di rete per la comunicazione su Internet. Il termine "WS" sta per "Winsock", che è l'interfaccia di programmazione delle applicazioni (API) di Windows per la comunicazione di rete. Il suffisso "_32" indica che si tratta di una libreria a 32 bit, anche se su sistemi operativi moderni è probabile che sia disponibile anche in versione a 64 bit per supportare entrambe le architetture. Questa libreria offre una vasta gamma di funzioni per la creazione, la gestione e la comunicazione attraverso socket, che sono fondamentali per lo sviluppo di applicazioni di rete su piattaforma Windows. Le funzionalità fornite da WS2_32.dll includono la creazione e l'utilizzo di socket TCP/IP e UDP, la risoluzione di nomi di dominio in indirizzi IP, la gestione degli eventi di rete e molto altro ancora. In sostanza, WS2_32.dll fornisce agli sviluppatori di software le funzionalità necessarie per creare applicazioni che possono comunicare su reti di computer, come ad esempio applicazioni web, client di posta elettronica, giochi online, e molti altri.



sezioni

SEZIONE .TEXT

1) Questa sezione contiene il codice eseguibile del programma. È la parte del file eseguibile che contiene le istruzioni che vengono eseguite dalla CPU. Nel caso di malware, la sezione .text conterrà il codice che svolge le azioni dannose o indesiderate. Questo potrebbe includere istruzioni per la raccolta di informazioni, la manipolazione dei file, l'esecuzione di azioni di rete, e così via.



SEZIONE .RDATA

2) Questa sezione contiene i dati di sola lettura (read-only data). Spesso, questa sezione contiene dati costanti utilizzati dal programma, come stringhe di testo, tabelle di lookup, costanti numeriche e così via. Nei malware, la sezione .rdata potrebbe contenere informazioni statiche utilizzate dal codice dannoso, come stringhe di comando, indirizzi IP, chiavi di crittografia e simili.



SEZIONE .DATA

3) Questa sezione contiene i dati variabili utilizzati dal programma. Può includere variabili globali, strutture dati dinamiche allocate durante l'esecuzione e altri dati modificabili. Nei malware, la sezione .data potrebbe contenere variabili utilizzate per tracciare lo stato dell'infezione, memorizzare informazioni rubate o configurazioni specifiche del malware.

Identificazione dei costrutti noti

```
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
```

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

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

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



Ecco i costrutti noti che possono essere identificati:

- 1)Push/Pop: Questi sono utilizzati per spostare dati sullo stack. push inserisce un valore nello stack, mentre pop estrae un valore dallo stack.
- 2)Mov: Questa istruzione viene utilizzata per spostare dati tra registri e memoria.
- 3)Call/Retn: Utilizzato per chiamare e restituire dalle funzioni.
- 4)Cmp/Jz: cmp compara due operandi, mentre jz salta all'etichetta specificata se il risultato dell'ultima operazione di confronto è zero (indicando che i due operandi erano uguali).
- 5)Jmp: Salta incondizionatamente a un'etichetta specificata.
- 6)Add/Sub/Xor: Operazioni aritmetiche e logiche.
- 7)Commenti: I commenti iniziano con ;. Sono utilizzati per documentare o annotare il codice.
- 8)Accesso alla memoria: [ebp+var_4] indica l'accesso alla memoria utilizzando l'indirizzo di base del puntatore di base dello stack (ebp) con un offset.
- 9)Etichette: Le etichette come loc_40102B sono punti di riferimento nel codice, spesso utilizzati per salti condizionali o incondizionali.
- 10)Manipolazione dello stack: esp e ebp sono utilizzati per gestire lo stack. esp è il registro dello stack pointer, mentre ebp è utilizzato come base del puntatore dello stack in molti casi.



COMPORTAMENTO DELLA FUNZIONALITÀ IMPLEMENTATA

Connessione a Internet:

- La funzione `InternetGetConnectedState` viene chiamata per verificare lo stato della connessione a Internet.
- Viene controllato il valore restituito per determinare se c'è una connessione attiva (cmp, jz).
- Se c'è una connessione, viene visualizzato il messaggio "Success: Internet Connection".
- Se non c'è una connessione, viene visualizzato il messaggio di errore "Error 1.1: No Internet".

Manipolazione dello stack:


- Viene effettuata una serie di operazioni push e pop per manipolare lo stack e gestire i valori dei registri.

Ritorno dalla funzione:

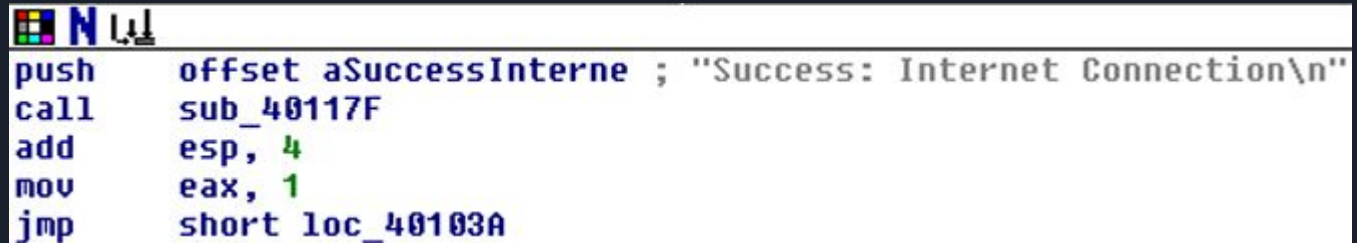
- Alla fine della funzione, viene eseguito un ritorno (ret) per tornare al chiamante.

Gestione dei puntatori di stack:


- `ebp` e `esp` vengono utilizzati per gestire i puntatori di stack all'interno della funzione



Grazie al costrutto if se c'è una connessione, viene visualizzato il messaggio "Success: Internet Connection".



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



Grazie sempre al costrutto if se non c'è una connessione, viene visualizzato il messaggio di errore "Error 1.1: No Internet".



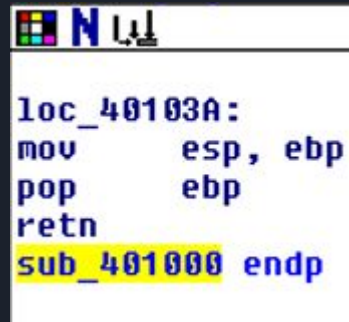
```
loc_40102B:                ; "Error 1.1: No Internet\n"
push    offset aError1_1NoInte
call    sub_40117F
add     esp, 4
xor     eax, eax
```




loc_40102B, offset aError1_1NoInte: Queste etichette sembrano essere punti di riferimento nel codice per gestire i casi in cui la connessione a Internet non è riuscita.

esp, ebp, ebp: Queste istruzioni sembrano coinvolgere la manipolazione dei registri esp ed ebp, che potrebbero essere utilizzati per gestire il frame dello stack prima di restituire il controllo al chiamante.

mou, pop, retn: Queste istruzioni sembrano essere utilizzate per terminare la funzione correttamente e restituire il controllo al chiamante.



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



Tuttavia, per ottenere una comprensione completa del comportamento del programma, sarebbe necessario esaminare più codice, in particolare la parte di codice che chiama questa funzione e come viene utilizzata la funzione in un contesto più ampio. La presenza di altre funzioni e variabili globali avrebbe un impatto significativo sul comportamento complessivo del programma.



Tabella con significato delle singole righe di codice assembly

Righe di Codice Assembly	Significato
<code>push</code>	Mette un valore nello <code>stack</code> .
<code>mov</code>	Muove dati da una sorgente a una destinazione.
<code>push</code>	Mette un valore nello <code>stack</code> .
<code>push</code>	Mette un valore nello <code>stack</code> .
<code>push</code>	Mette un valore nello <code>stack</code> .
<code>call</code>	Chiama una funzione.
<code>mov</code>	Muove dati da una sorgente a una destinazione.
<code>cmp</code>	Confronta due operandi.
<code>ja</code>	Salta se l'ultima operazione di confronto è zero.
<code>ebp</code>	Registra del frame dello <code>stack</code> .
<code>ebp, esp</code>	Muove il registro <code>ebp</code> nel registro <code>esp</code> .
<code>ecx</code>	Registro per uso generale.
<code>h: InternetGetConnectedState</code>	Accesso alla funzione <code>InternetGetConnectedState</code> .
<code>[ebp+var_4], eax</code>	Salva il valore di <code>eax</code> all'indirizzo <code>[ebp+var_4]</code> .
<code>[ebp+var_4], 0</code>	Salva il valore 0 all'indirizzo <code>[ebp+var_4]</code> .
<code>jmp</code>	Etichetta del codice.
<code>push</code>	Mette un valore nello <code>stack</code> .
<code>call</code>	Chiama una funzione.
<code>add</code>	Aggiunge due operandi.
<code>mov</code>	Muove dati da una sorgente a una destinazione.
<code>jmp</code>	Salta incondizionatamente a una destinazione.
<code>offset aSuccessInterno</code>	Ottiene l'offset della stringa "Success: Internet Connection\n".
<code>sub_40117F</code>	Chiama la funzione <code>sub_40117F</code> .
<code>esp, 4</code>	Modifica il registro <code>stack pointer</code> <code>esp</code> .
<code>eax, 1</code>	Assegna il valore 1 al registro <code>eax</code> .
<code>loc_40103A</code>	Etichetta del codice.
<code>push</code>	Mette un valore nello <code>stack</code> .
<code>call</code>	Chiama una funzione.
<code>add</code>	Aggiunge due operandi.
<code>xor</code>	Esegue un'operazione logica XOR.
<code>jmp</code>	Etichetta del codice.
<code>loc_40102B</code>	Etichetta del codice.
<code>offset aError1_NoInte</code>	Ottiene l'offset della stringa "Error 1.1: No Internet\n".
<code>sub_40117F</code>	Chiama la funzione <code>sub_40117F</code> .
<code>esp, 4</code>	Modifica il registro <code>stack pointer</code> <code>esp</code> .
<code>eax, ecx</code>	Assegna il valore di <code>ecx</code> a <code>eax</code> .
<code>loc_40103A</code>	Etichetta del codice.
<code>mov</code>	Probabile errore di battitura, corretto potrebbe essere <code>mov</code> .

Righe di Codice Assembly	Significato
<code>pop</code>	Estrae un valore dallo <code>stack</code> .
<code>ret</code>	Restituisce il controllo al chiamante.
<code>sub_401000_end</code>	Marca la fine della funzione.
<code>esp, ebp</code>	Muove il registro <code>stack pointer</code> <code>esp</code> nel registro base pointer <code>ebp</code> .
<code>ebp</code>	Registro del frame dello <code>stack</code> .