

Immagine che contiene testo, linea, schermata, software

Descrizione generata automaticamente

TRACCIA 1

Utilizzando CFF Explorer, vediamo dalla sezione import directory che il malware U3\_W2\_L5 importa 2 librerie:

**Librerie:**

1. **kernel32.dll. - Questa è una libreria di sistema di Windows che fornisce varie funzioni di basso livello necessarie per l'interazione con il sistema operativo, come la gestione della memoria, l'accesso ai file, la creazione di processi**
2. **wininet.dll - Questa libreria fornisce funzionalità per l'accesso a Internet. È spesso utilizzata per operazioni come il download di file da Internet, l'invio di richieste http.**

Immagine che contiene testo, software, numero, schermata

Descrizione generata automaticamente

TRACCIA 2

**Sezioni:**

1. **.text -** Questa sezione contiene il codice eseguibile, ovvero le istruzioni di programma.
2. **.rdata - Questa sezione contiene dati di sola lettura, come stringhe di testo, costanti o altre informazioni che il programma utilizza ma non modifica durante l'esecuzione.**
3. **.data - Questa sezione contiene dati inizializzati che possono essere modificati durante l'esecuzione del programma.**

Inoltre, alcune funzioni specifiche presenti nella libreria **kernel32.dll**, come **GetEnvironmentVariableA**, **HeapDestroy**, **HeapCreate**, **VirtualFree**, **HeapFree**, **RtlUnwind**, **GetOEMCP**, **GetProcAddress**, **LoadLibraryA**, **GetLastError**, **FlushFileBuffers**, **SetFilePointer**, **CloseHandle**.

1. **GetEnvironmentVariableA**: Questa funzione viene utilizzata per recuperare il valore di una variabile d'ambiente specificata.
2. **HeapDestroy**: Questa funzione viene utilizzata per eliminare un gestore di heap specificato.
3. **HeapCreate**: Viene utilizzata per creare un nuovo heap.
4. **VirtualFree**: Questa funzione libera la memoria virtuale allocata da una chiamata precedente alla funzione **VirtualAlloc**.
5. **HeapFree**: Questa funzione libera un blocco di memoria all'interno di un heap specificato.
6. **RtlUnwind**: Questa funzione esegue un'operazione di "unwind" (scivolamento all'indietro) nello stack, annullando le chiamate di funzione finché non viene trovata una destinazione specificata.
7. **GetOEMCP**: Questa funzione restituisce il codice pagina OEM (Original Equipment Manufacturer) corrente.
8. **GetProcAddress**: Viene utilizzata per ottenere l'indirizzo di una funzione esportata da un modulo DLL.
9. **LoadLibraryA**: Viene utilizzata per caricare una DLL specificata.
10. **GetLastError**: Restituisce il codice dell'ultimo errore.
11. **FlushFileBuffers**: Forza la scrittura dei buffer di un file su disco.
12. **SetFilePointer**: Imposta la posizione del puntatore di file per un file specificato.
13. **CloseHandle**: Chiude un handle di file, una handle di modulo o una handle di comunicazione.

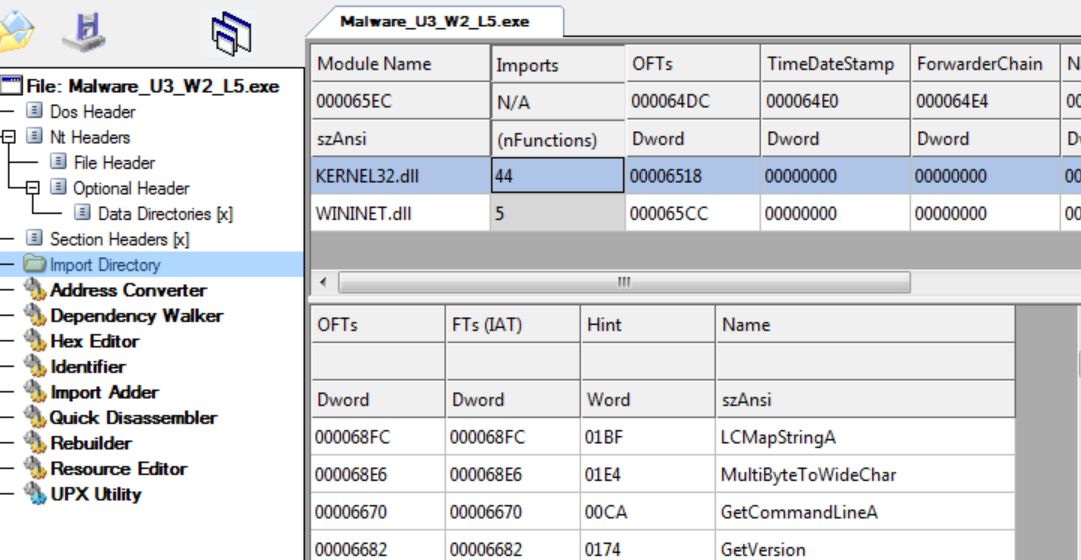
Dal contesto fornito, possiamo trarre alcune ipotesi su ciò che il malware potrebbe fare:

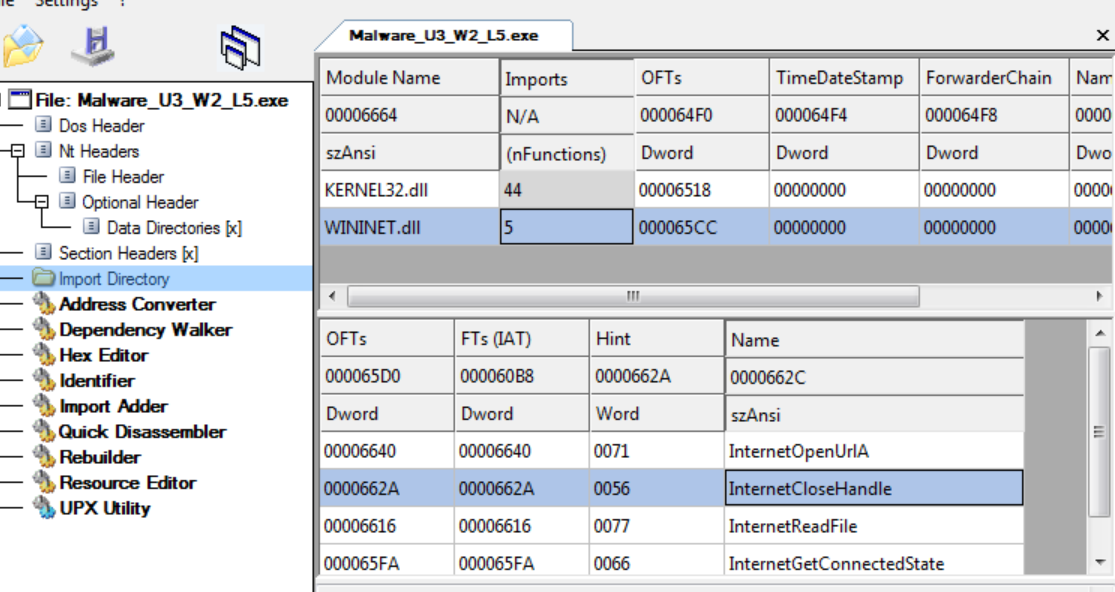
1. **Interazione con il sistema operativo**: La presenza di funzioni come **GetLastError**, **FlushFileBuffers**, **SetFilePointer**, **CloseHandle** e altre funzioni di basso livello di **kernel32.dll** suggerisce che il malware potrebbe essere coinvolto in attività di gestione di file, manipolazione di memoria o interazione con il sistema operativo. Questo potrebbe includere la manipolazione dei file, la creazione di processi maligni o altre attività dannose.
2. **Comunicazione in rete**: L'importazione di **wininet.dll** e la potenziale utilizzo di funzioni come **InternetOpenUrlA** suggerisce che il malware potrebbe essere coinvolto in attività di comunicazione in rete, come il download di file dannosi, l'invio di dati a server remoti o il controllo da parte di un server di comando e controllo (C&C).
3. **Struttura del file eseguibile**: La presenza delle sezioni **.text**, **.rdata** e **.data** indica la struttura tipica di un file eseguibile Windows. La sezione **.text** contiene il codice eseguibile, **.rdata** contiene dati di sola lettura e **.data** contiene dati inizializzati.

In generale, la presenza di queste librerie e funzioni, insieme alla struttura del file eseguibile, suggerisce che il malware potrebbe essere sofisticato e potenzialmente pericoloso. Tuttavia, per ottenere una comprensione completa delle sue capacità e del suo impatto potenziale, è necessaria un'analisi più approfondita, comprese le analisi dinamiche

Immagine che contiene testo, numero, linea, Carattere

Descrizione generata automaticamente





TRACCIA 3

Ecco una disamina della sequenza di codice assembly fornita:

1. **Creazione dello stack**:
   * Le istruzioni **push ebp** e **mov ebp, esp** vengono utilizzate per creare un nuovo frame di stack. L'indirizzo base dello stack viene salvato nel registro **ebp**, e **esp** viene utilizzato come punto di riferimento per lo stack locale a questa funzione.
2. **Chiamata alla funzione InternetGetConnectedState**:
   * Le istruzioni **push ecx**, **push 0**, **push 0** preparano gli argomenti da passare alla funzione **InternetGetConnectedState**.
   * La chiamata alla funzione è effettuata con l'istruzione **call ds:internetGetConnectedState**.
   * Il risultato della chiamata viene memorizzato in **[ebp - var\_4]** utilizzando l'istruzione **mov [ebp - var\_4], eax**.
3. **Gestione del risultato della chiamata**:
   * Viene eseguito un controllo sul valore restituito dalla funzione. Se il risultato è non zero, viene eseguita la sezione "Success"; altrimenti, viene eseguita la sezione "Error".
4. **Fine della funzione**:
   * L'istruzione **mov esp, ebp** ripristina il puntatore dello stack allo stato precedente.
   * L'istruzione **pop ebp** ripristina il registro di base dello stack.
   * L'istruzione **ret** indica la fine della funzione.

TRACCIA 4

**Ipotizzare il comportamento della funzionalità implementata:**

* Questo frammento di codice sembra verificare se il sistema è connesso a Internet utilizzando la funzione **InternetGetConnectedState**.
* Se la connessione a Internet è attiva, viene stampato un messaggio di successo, altrimenti viene stampato un messaggio di errore.
* Dopo l'esecuzione della stampa del messaggio, la funzione restituisce il valore 1.

TRACCIA 5

**Significato delle singole righe di codice assembly:**

| **Righe di Codice Assembly** | **Significato** |
| --- | --- |
| **push ebp** | Salva il valore corrente di EBP nello stack |
| **mov ebp, esp** | Inizializza il registro EBP con ESP |
| **push ecx** | Salva il valore corrente di ECX nello stack |
| **push 0** | Push del valore 0 nello stack |
| **push 0** | Push del valore 0 nello stack |
| **call ds:internetGetConnectedState** | Chiama la funzione InternetGetConnectedState |
| **mov [ebp - var\_4], eax** | Memorizza il risultato della funzione in [ebp - var\_4] |
| **push offset aSuccessInterne** | Push dell'offset della stringa di successo nello stack |
| **call sub\_40117F** | Chiama la subroutine sub\_40117F |
| **add esp, 4** | Aggiunge 4 byte a ESP |
| **mov eax, 1** | Imposta EAX a 1 |
| **push offset aError1\_1Nointe** | Push dell'offset della stringa di errore nello stack |
| **call sub\_40117F** | Chiama la subroutine sub\_40117F |
| **add esp, 4** | Aggiunge 4 byte a ESP |
| **xor eax, eax** | Esegue un'operazione XOR su EAX con se stesso |
| **mov esp, ebp** | Ripristina ESP dal valore di EBP |
| **pop ebp** | Ripristina EBP dallo stack |
| **ret** | Restituisce il controllo al chiamante |