

MALWARE ANALYSIS



WORK DELIVERIES

**Quali librerie
vengono
importate?**

**Quali sono le
sezioni di cui si
compone il file
eseguibile del
malware?**

**Identificare i
costrutti noti**

**ipotizzare il
comportamento
delle funzionalità
integrate**

COSA SONO LE LIBRERIE

Le librerie sono insiemi di funzioni e procedure predefinite che possono essere utilizzate da programmi software. Nell'ambito dei malware, il termine "libreria" può assumere un significato diverso. In questo contesto, i creatori di malware possono sfruttare librerie dannose o malevoli per eseguire attività dannose sui sistemi compromessi. Queste librerie possono contenere codice dannoso che sfrutta vulnerabilità di sicurezza, ruba informazioni sensibili o esegue altre attività dannose. In pratica, le librerie malware possono essere utilizzate per rendere il codice dannoso più modulare, riducendo il rischio di rilevamento da parte degli antivirus e semplificando l'aggiornamento del malware stesso.



1

Quali sono le librerie importate?

answer

A seguito di una scansione eseguita tramite il programma: CFF Explorer, abbiamo individuato che il malware importa 2 librerie:

- **KERNEL32.dll**: contiene le funzioni principali per interagire con il sistema operativo, ad esempio: manipolazione dei file, la gestione della memoria.
- **WININET.dll**: contiene le funzioni per l'implementazione di alcuni protocolli di rete come HTTP, FTP, NTP.

The screenshot shows the CFF Explorer interface. On the left, a tree view displays the file structure of 'Malware_U3_W2_L5.exe', including sections like Dos Header, Nt Headers, File Header, Optional Header, Data Directories [x], Section Headers [x], and Import Directory. Below the tree are various tools: Address Converter, Dependency Walker, Hex Editor, Identifier, Import Adder, Quick Disassembler, Rebuilder, Resource Editor, and UPX Utility. On the right, a table titled 'Malware_U3_W2_L5.exe' lists the imported modules and their details:

Module Name	Imports	OFTs	TimeStamp	ForwarderChain	Name
szAnsi	(nFunctions)	Dword	Dword	Dword	Dwo
KERNEL32.dll	44	00006518	00000000	00000000	00000000
WININET.dll	5	000065CC	00000000	00000000	00000000

COSA SONO LE SEZIONI DI UN MALWARE

Le sezioni di un malware si riferiscono alle diverse parti o componenti che costituiscono il codice malevolo. Queste sezioni sono organizzate in modo da consentire al malware di eseguire le sue funzioni in modo efficace e di eludere la rilevazione da parte delle soluzioni di sicurezza. Bisogna notare che queste sezioni possono variare a seconda del tipo specifico di malware e degli obiettivi degli attaccanti. Gli autori di malware possono adottare diverse strategie per complicare l'analisi e la rilevazione, utilizzando tecniche avanzate di programmazione e crittografia.



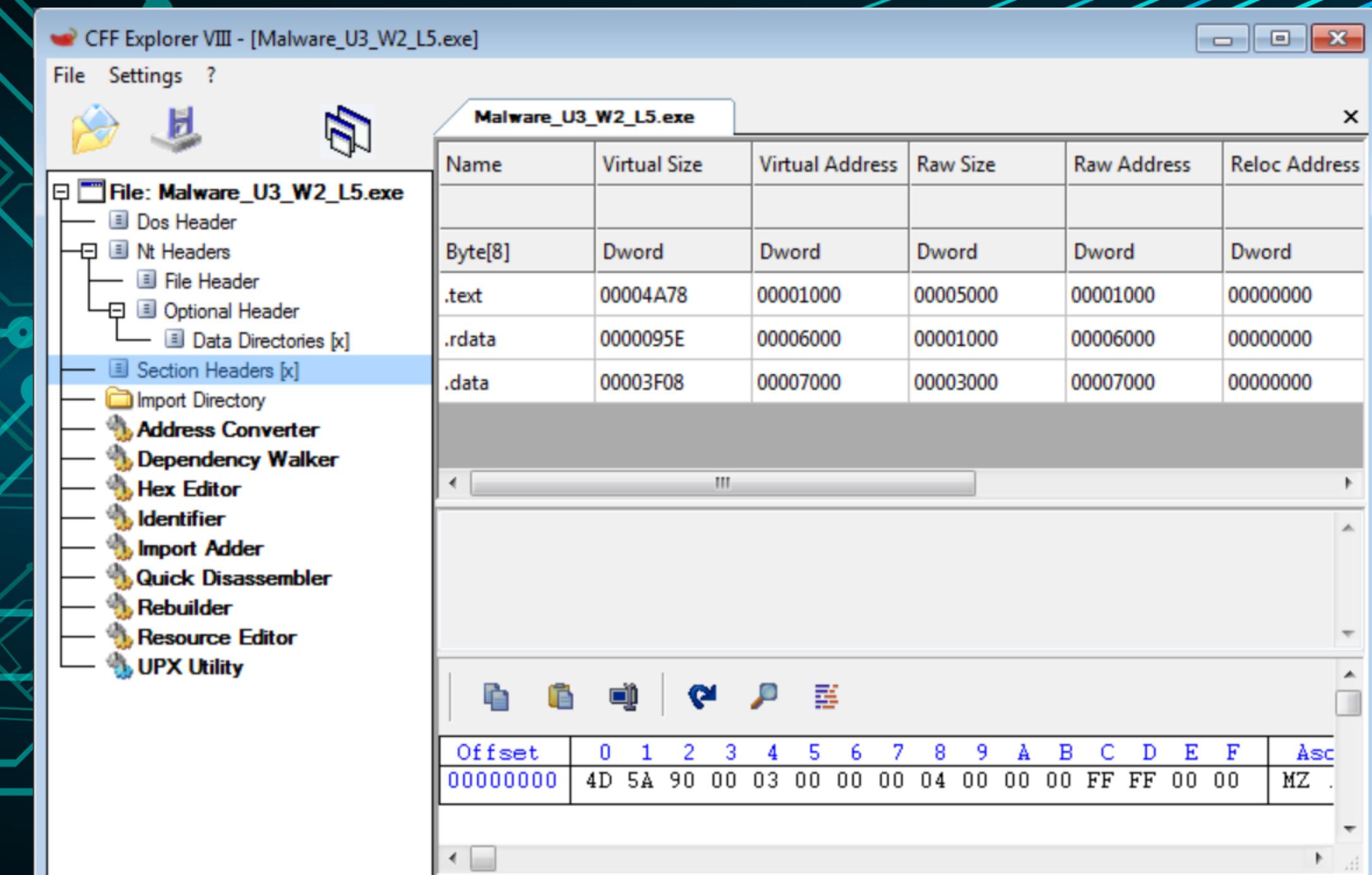
2

question

Quali sono le sezioni di cui si compone il file eseguibile del malware?

Il malware preso in analisi è composto da 3 sezioni differenti:

- **.text:** contiene le rime di codice che la CPU eseguirà una volta che il software sarà avviato. Generalmente questa è l'unica sezione di un file eseguibile che viene eseguita dalla CPU, in quanto tutte le altre sezioni contengono dati o informazioni a supporto.
- **.rdata:** include generalmente le informazioni circa le librerie e le funzioni importate ed esportate dall'eseguibile.
- **.data:** contiene tipicamente i dati e/o le variabili globali del programma eseguibile, che devono essere disponibili da qualsiasi parte del programma.



3

question

Identificare i 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

```

Creazione dello stack

Chiamata della funzione. I parametri passano sullo stack tramite le istruzioni push

```

push    offset aSuccessInternet ; "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

```

Esecuzione del ciclo "if"

```

loc_40103A:
mov    esp, ebp
pop    ebp
retn
sub_401000 endp

```

Chiusura del ciclo if

4

question

Ipotizzare il comportamento delle funzionalità integrate

Il codice assembly verifica la connessione a Internet usando la funzione `InternetGetConnectedState`. Se la connessione è presente, stampa "Success Internet Connection", altrimenti stampa "Error 1.1: No Internet". Il programma utilizza confronti e salti condizionati per gestire i casi di successo e fallimento. Infine, ritorna al chiamante dopo aver stampato il messaggio corrispondente. La presenza o l'assenza di connessione a Internet è determinata dalla funzione `InternetGetConnectedState`. Questa funzione restituisce un valore che indica lo stato della connessione. Nel codice fornito, il risultato della chiamata a `InternetGetConnectedState` viene salvato nella variabile `[ebp+var_4]`.

Successivamente, il codice esegue una comparazione con zero (`cmp [ebp+var_4], 0`) per verificare se il valore restituito dalla funzione è zero o no. Se il confronto dà esito positivo (zero), il programma interpreta ciò come assenza di connessione a Internet e passa al ramo "Error 1.1: No Internet". In caso contrario, il programma interpreta la presenza di una connessione e passa al ramo "Success Internet Connection".

The screenshot shows a debugger interface with three distinct assembly code snippets:

- Top section:** Compares the value at `[ebp+var_4]` with zero using `cmp [ebp+var_4], 0`. If the result is zero (success), it jumps to `loc_40102B`. The assembly code is:

```
mov    [ebp+var_4], eax
cmp    [ebp+var_4], 0
jz     short loc_40102B
```
- Middle section:** Prints "Success: Internet Connection\n" if the connection was found. The assembly code is:

```
push  offset aSuccessInternet ; "Success: Internet Connection\n"
call  sub_40117F
add   esp, 4
mov   eax, 1
jmp   short loc_40103A
```
- Right section:** Prints "Error 1.1: No Internet\n" if no connection was found. The assembly code is:

```
loc_40102B:
push  offset aError1_NoInte ; "Error 1.1: No Internet\n"
call  sub_40117F
add   esp, 4
xor   eax, eax
```

6

bonus

Significato delle singole righe in assembly

Codice	Analisi
.text:00401000 push ebp	Sposta il puntatore ebp nello stack.
.text:00401001 mov ebp, esp	utilizzata per imposta ebp con il valore corrente di esp.
.text:00401003 push ecx	Sposta il registro ecx nello stack
.text:00401004 push 0 ; dwReserved	Sposta valore zero nello stack in una sezione riservata ma vuota
.text:00401006 push 0 ; lpdwFlags	Sposta il valore 0 nello stack, e il commento indica che questo valore potrebbe rappresentare un parametro chiamato
.text:00401008 call ds:InternetGetConnectedState	Chiama il comando InternetGetConnectedState
.text:0040100E mov [ebp+var_4], eax	Salva il valore di eax in una variabile locale (o posizione di memoria) associata al frame del stack corrente
.text:00401011 cmp [ebp+var_4], 0	Verifica se il valore precedentemente salvato in var_4 è uguale a zero o meno
.text:00401015 jz short loc_40102B	Salta se la condizione "zero" è soddisfatta con la conseguente esecuzione del programma all'indirizzo specificato da loc_40102B.
.text:00401017 push offset asuccessInterne ; "Succes Internet Connection\n"	Sposta nello stack l'indirizzo (offset) della stringa "Succes Internet Connection\n"
.text:0040101C call sub_40105F	Chiama la funzione sub_40105F
.text:00401021 add esp, 4	Aggiunge 4 byte al registro esp
.text:00401024 mov eax, 1	Imposta il registro eax a 1
.text:00401029 jmp short loc_40103A	Esegue un salto all'indirizzo loc_40103A
.text:0040102B push offset aError1_1NoInte ; "Error 1.1: No Internet\n" call sub_40117F add esp, 4 xor eax, eax	- push offset aError1_1NoInte mette nello stack l'indirizzo della stringa "Error 1.1: No Internet\n". - call sub_40117F esegue una chiamata alla funzione il cui indirizzo inizia con sub_40117F. Questa funzione probabilmente gestisce la stampa del messaggio di errore. - add esp, 4 libera lo spazio nello stack precedentemente riservato per il parametro della stringa. - xor eax, eax imposta il registro eax a zero. Questa operazione può essere utilizzata per indicare un fallimento o un risultato negativo nel contesto del programma.
.text:0040103A: mov esp, ebp pop ebp retn sub_401000 endp	- mov esp, ebp ripristina il valore originale di esp con il valore contenuto nel registro ebp, liberando lo spazio allocato per le variabili locali e i parametri della funzione nello stack. - pop ebp ripristina il valore originale del registro ebp e chiude il frame del stack della funzione. - retn esegue un'istruzione di ritorno dalla funzione, riportando il flusso di esecuzione al punto in cui è stata chiamata la funzione. - 'etichetta sub_401000 endp indica la fine della funzione denominata sub_40100

GRAZIE DELL'ATTENZIONE

