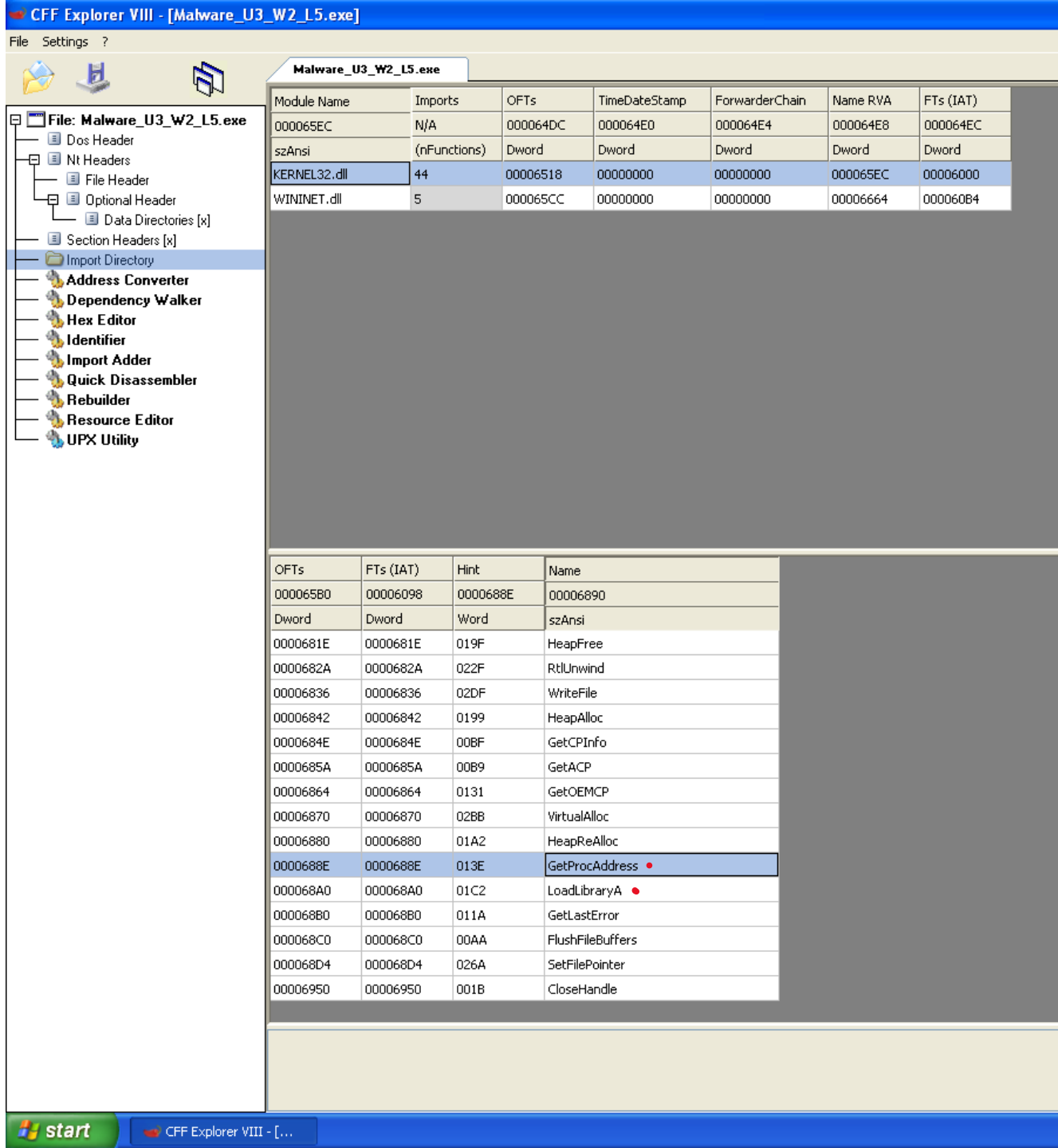
In questo esercizio andremo a fare un'analisi statica basica di un malware su windows XP.

L'analisi statica basica consente di analizzare un eventuale malware senza la necessità di eseguirlo, dandoci comunque delle informazioni importanti a riguardo. E' sicuramente la tecnica di analisi più rapida ed intuitiva, anche se la meno efficace per quanto riguarda i malware più avanzati.

Iniziamo verificando le librerie importate dal malware, possiamo notare subito **KERNEL32.dll** e **WININET.dll**.

Kernel32.dll è una libreria che contiene le funzioni principali per interagire col sistema operativo o per gestire la memoria.

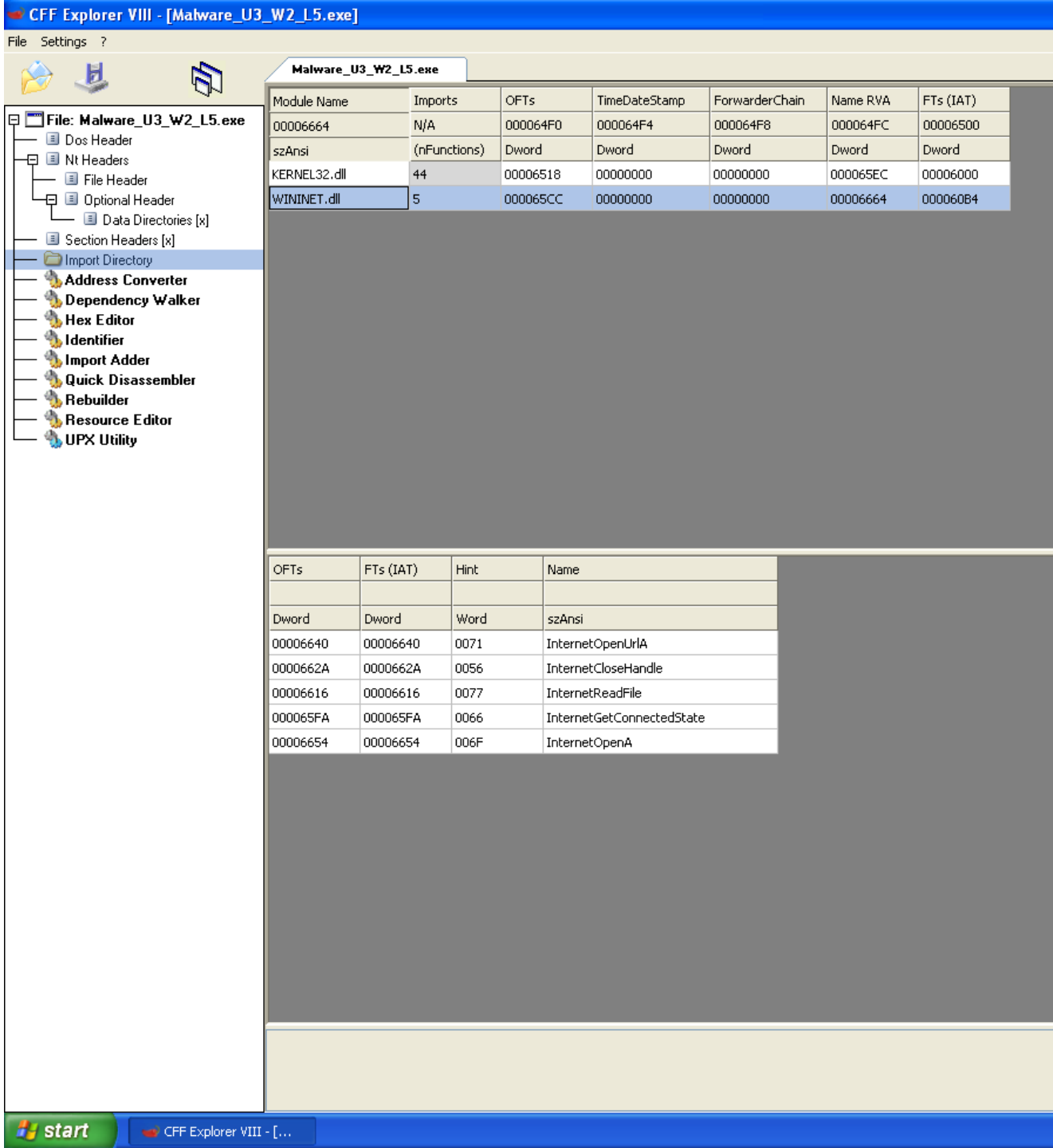
In questo caso possiamo notare due funzioni in particolare; GetProcAddress e LoadLibraryA, queste due ci fanno capire che la libreria in questione viene caricata dinamicamente, ovvero vengono chiamate soltanto quando il programma (o malware) in questione ha bisogno di utilizzarle.
Caricando le librerie dinamicamente permette ad un eventuale malware di essere meno invasivo e meno rilevabile.



L'altra libreria in questione è **Wininet.dll**, una libreria che permette al malware di importare funzioni per l'utilizzo di protocolli di rete (ad esempio HTTP ed FTP).

Questo ci dà parecchie info su ciò che potrebbe fare il malware, molto probabilmente potrebbe connettersi ad internet, per scaricare altro materiale malevolo oppure per condividere informazioni sensibili dal dispositivo ad un server remoto o anche una sorta di backdoor.

Analizzeremo nel dettaglio il programma in assembly più avanti, durante l'esercizio, per ottenere qualche informazione in più sul malware in questione.

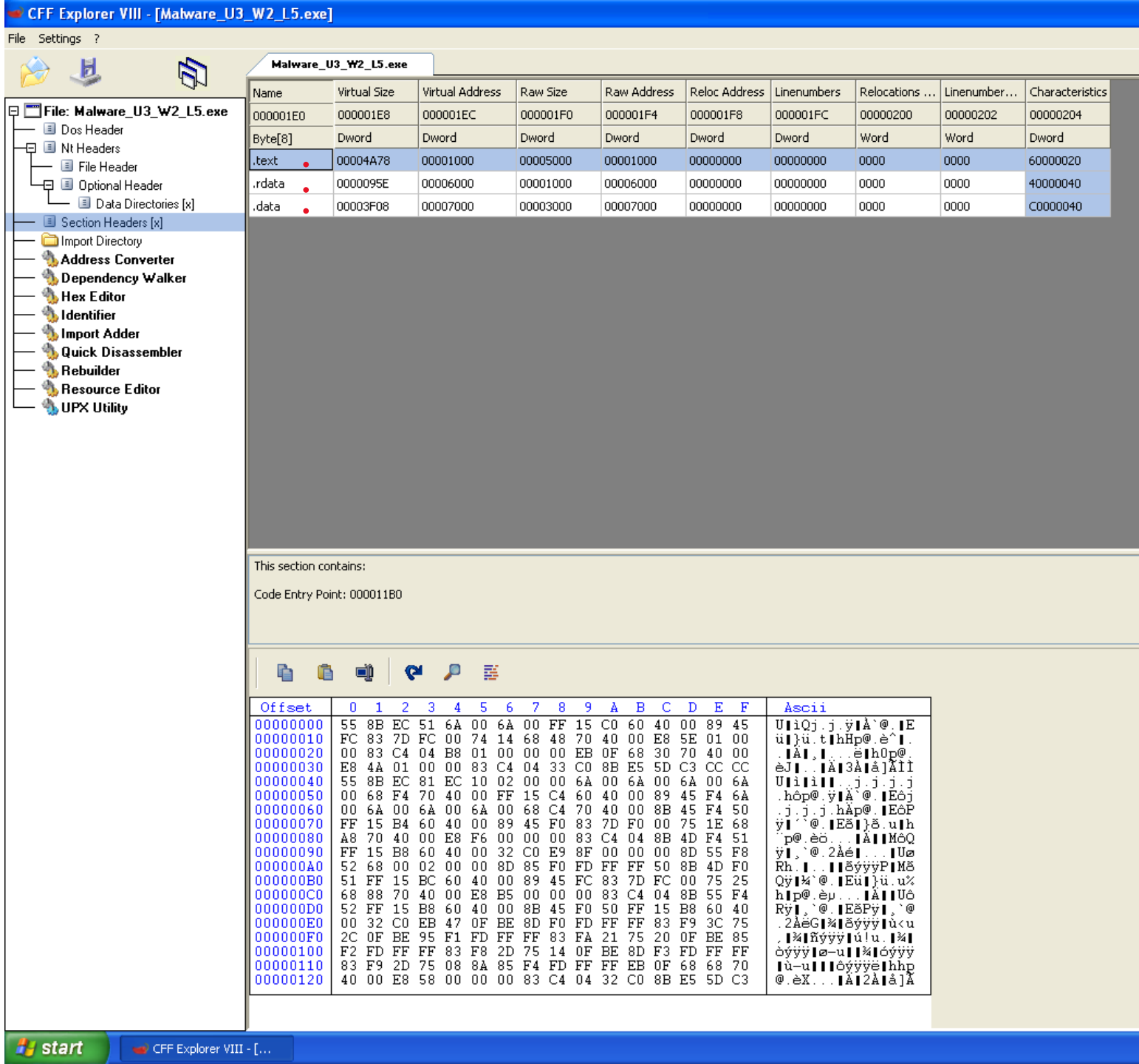


Il secondo passaggio che andremo a fare è verificare di che "sezioni" è composto il file. Le sezioni ci danno un'ulteriore idea su come funziona il malware, andiamo ad analizzarle.

.text è la sezione di file che contiene tutte le informazioni e le istruzioni che la CPU dovrà eseguire a malware avviato.

.rdata include tutte le informazioni sulle librerie importate ed esportate dal programma (cosa che possiamo appunto verificare con CFF Explorer, come negli screenshot precedenti).

.data invece include le variabili globali dell'eseguibile, sono globali perché devono essere disponibili a qualsiasi funzione all'interno dell'eseguibile.



Il programma qui a fianco è scritto in assembly x86.

Il suo scopo è verificare un'eventuale connessione ad internet utilizzando la funzione "InternetGetConnectedState".
Utilizza un costrutto IF (lo analizzeremo nella prossima slide), nel caso la funzione venga verificata correttamente, il programma userà la parte "Success: Internet Connection", in caso contrario, la parte "Error 1.1: No internet".

Possiamo correlare questo codice in assembly alla libreria "wininet.dll" vista in precedenza, confermando il fatto che il malware vada ad utilizzare una componente che si connette ad internet.

Analizziamo nel dettaglio i blocchi di codice.

```
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
```

"Success: Internet Connection"

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

"Error 1.1: No internet"

```
Loc_40102B:
push offset aError1_NoInte
call sub_40117F
add esp, 4
xor eax, eax

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

Il programma crea lo stack	push ebp mov ebp, esp push ecx
Chiama la funzione "InternetGetConnectedState"	push 0 ; dwReserved push 0 ; lpdwFlags call ds:InternetGetConnectedState
Copia il valore in [ebp+var_4]	mov [ebp+var_4], eax
Confronta il risultato con 0.	cmp [ebp+var_4], 0
Se uguale a 0, fa un jump all'indirizzo loc_40102B (costrutto IF, connessione fallita)	jz short loc_40102B

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

Connessione fallita	loc_40102B: push offset aError1_NoInte ; "Error 1.1: No Internet\n" call sub_40117F add esp, 4 xor eax, eax
Resetta lo stack e ritorna (retn) alla funzione	loc_40103A: mov esp, ebp pop ebp retn