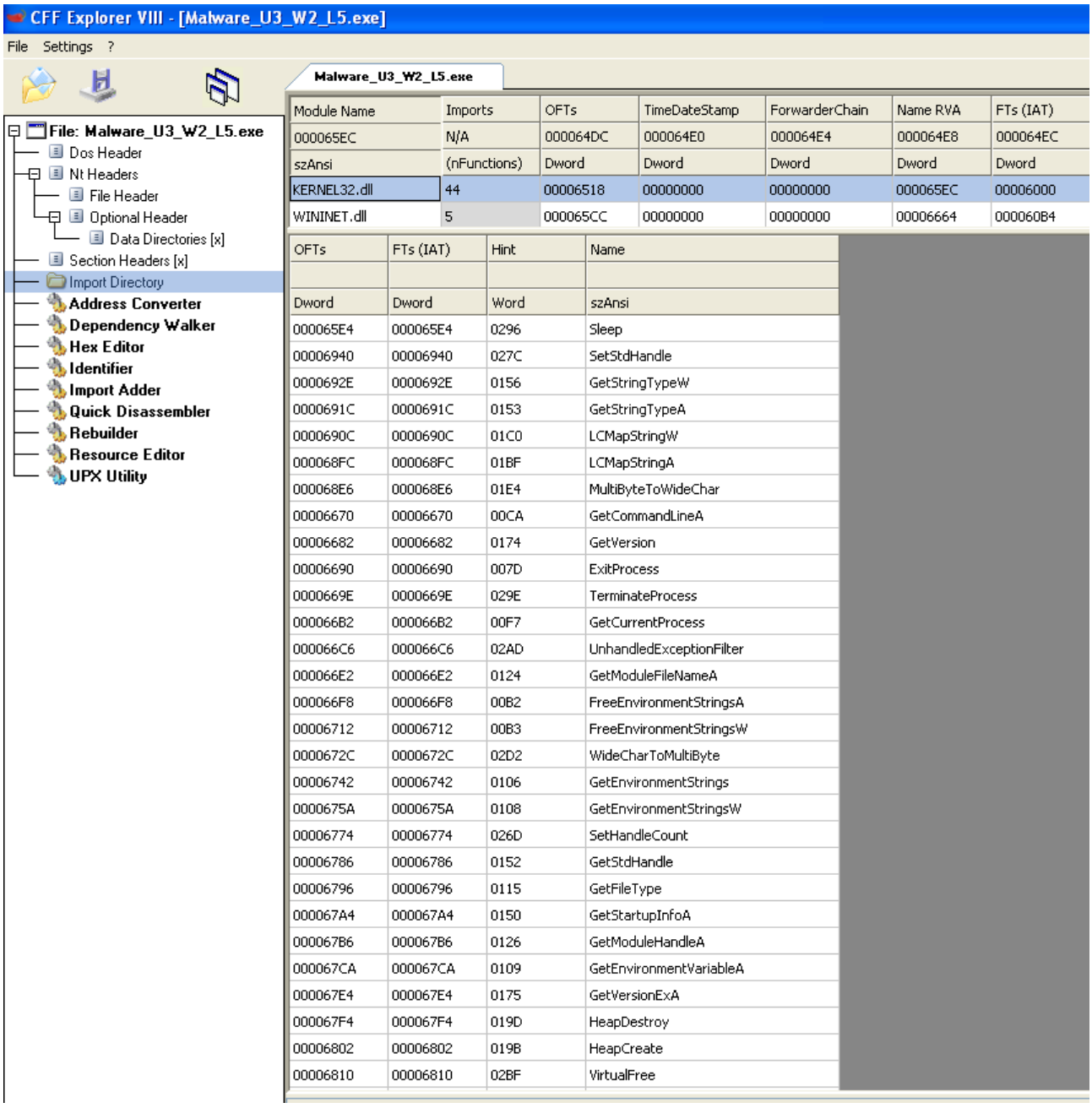
S10L5 LUCA DANELLI

Quali librerie vengono importate dal file eseguibile?

Per rispondere alla domanda sono andato ad avviare il programma CFF Explorer dalla macchina Windows XP, andando ad aprire il file eseguibile. Mi sono poi spostato sulla tab denominata “Import Directory”, che contiene appunto le librerie importate dall'eseguibile una volta avviato. Come si vede nelle figure sotto, le librerie importate sono **KERNEL32.dll** e **WININET.dll.**

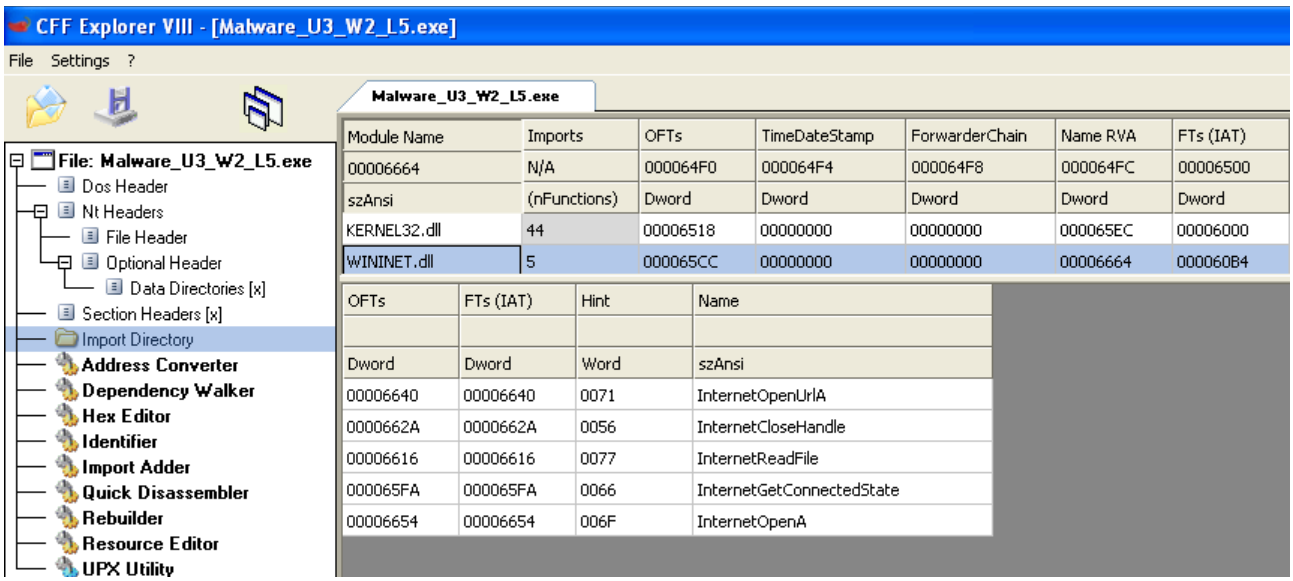
Kernel.dll viene utilizzata per interagire con il sistema operativo, manipolarne i file e la memoria. Wininet.dll viene invece utilizzata per utilizzare i protocolli di rete come HTTP, FTP e NTP

Nelle immagini possiamo vedere anche le funzioni importate corrispondenti per ciascuna libreria:



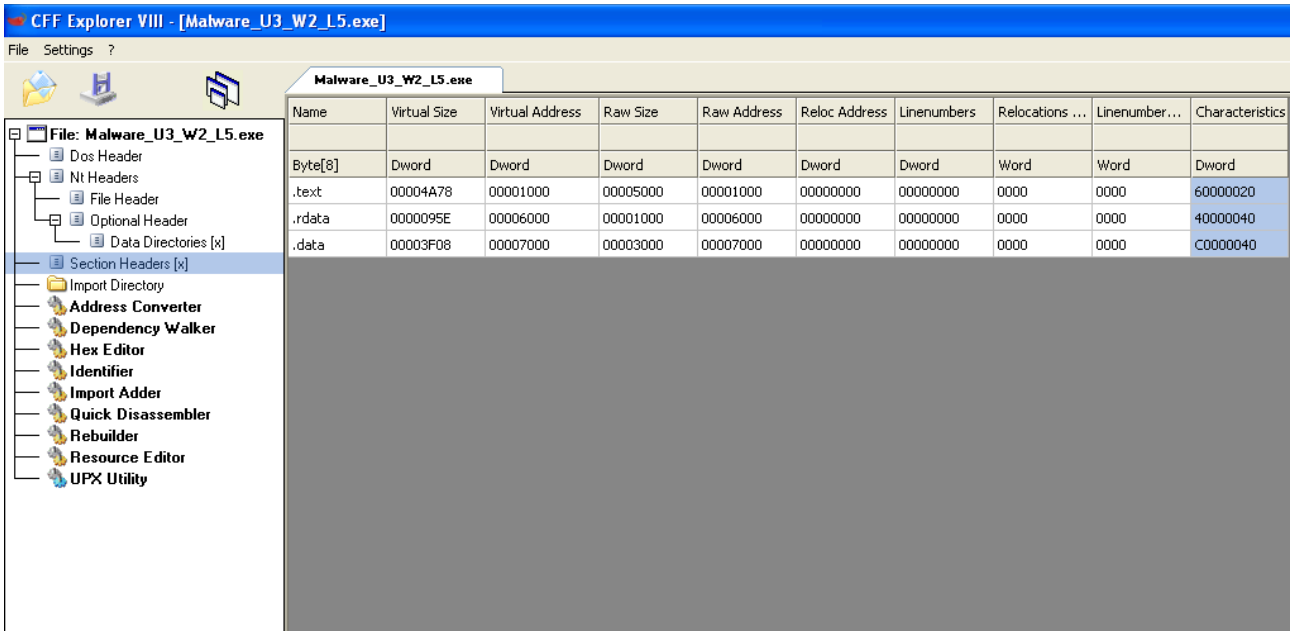
Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
000065EC	N/A	000064DC	000064E0	000064E4	000064E8	000064EC
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	44	00006518	00000000	00000000	000065EC	00006000
WININET.dll	5	000065CC	00000000	00000000	00006664	000060B4

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
000065E4	000065E4	0296	Sleep
00006940	00006940	027C	SetStdHandle
0000692E	0000692E	0156	GetStringTypeW
0000691C	0000691C	0153	GetStringTypeA
0000690C	0000690C	01C0	LCMapStringW
000068FC	000068FC	01BF	LCMapStringA
000068E6	000068E6	01E4	MultiByteToWideChar
00006670	00006670	00CA	GetCommandLineA
00006682	00006682	0174	GetVersion
00006690	00006690	007D	ExitProcess
0000669E	0000669E	029E	TerminateProcess
000066B2	000066B2	00F7	GetCurrentProcess
000066C6	000066C6	02AD	UnhandledExceptionFilter
000066E2	000066E2	0124	GetModuleFileNameA
000066F8	000066F8	00B2	FreeEnvironmentStringsA
00006712	00006712	00B3	FreeEnvironmentStringsW
0000672C	0000672C	02D2	WideCharToMultiByte
00006742	00006742	0106	GetEnvironmentStrings
0000675A	0000675A	0108	GetEnvironmentStringsW
00006774	00006774	026D	SetHandleCount
00006786	00006786	0152	GetStdHandle
00006796	00006796	0115	GetFileType
000067A4	000067A4	0150	GetStartupInfoA
000067B6	000067B6	0126	GetModuleHandleA
000067CA	000067CA	0109	GetEnvironmentVariableA
000067E4	000067E4	0175	GetVersionExA
000067F4	000067F4	019D	HeapDestroy
00006802	00006802	019B	HeapCreate
00006810	00006810	02BF	VirtualFree



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

Sempre da CFF Explorer, mi sposto sul tab Section Headers [x]:



Da qui possiamo vedere le sezioni componenti il malware. Di seguito una breve spiegazione per ciascuna:

.text qui è presente il codice del malware che verrà eseguito
.rdata qui sono contenute le librerie e le funzioni importate
.data qui risiedono le variabili globali di cui il codice e le funzioni importate faranno uso

Con riferimento alla figura slide 3, rispondere ai seguenti quesiti:

Identificare i costrutti noti

1)Il primo costrutto riconoscibile è la **creazione dello stack**, che sarà l'area di memoria RAM all'interno della quale verrà eseguita la funzione successiva. E' identificato dalle seguenti istruzioni:

```
push    ebp
mov     ebp, esp
```

Alla fine del codice troviamo la sua chiusura, identificata dalle istruzioni:

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

2)Il secondo costrutto riconoscibile è la chiamata della funzione **InternetGetConnectedState**. Prima della chiamata vengono aggiunte in cima allo stack i tre parametri di cui la funzione ha bisogno. Di seguito le istruzioni:

```
push    ecx
push    0    ; dwReserved
push    0    ; lpdwFlags
call    ds:InternetGetConnectedState
```

3)Il terzo costrutto riconoscibile è il salto condizionale **jz** in riferimento all'istruzione **cmp**, che è l'equivalente di un costrutto **if/else** in linguaggio C:

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

Viene confrontato il valore 0 con la variabile var_4 tramite l'istruzione **cmp.** Se i due operandi risultano uguali, l'istruzione setterà lo **ZF** a 1. Successivamente il salto condizionale **jz** controllerà se lo **ZF** risulti settato (quindi uguale a 1) e in caso affermativo salterà all'indirizzo di memoria 40102B, ignorando le istruzioni comprese tra tale indirizzo e l'indirizzo corrente.
A livello logico possiamo dire che tale costrutto serve a capire se l'output della funzione **InternetGetConnectedState** sia 0. In tal caso verranno eseguite le seguenti istruzioni:

```
loc_40102B:
push    offset aError1_1NoInte ; “Error1.1: No Internet\n”
call sub_40117F
add esp, 4
xor eax, eax
```

Altrimenti, quindi in caso di connessione presente, verranno eseguite le seguenti istruzioni:

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

Si può dire quindi che sia paragonabile ad uno pseudocodice di questo tipo:

```
if (a=0) {
print(No internet)
}
else {
print(Success: Internet Connection)
}
```

Ipotizzare il comportamento della funzionalità implementata

Dall'analisi statica di base del codice si può ipotizzare che il malware in questione cerchi di connettersi ad internet dall'import della libreria wininet.dll
Dall'analisi dell'assembly si conferma questa ipotesi, dato che nel segmento di codice evidenziato è presente una funzione che va a controllare se la macchina target è connessa o meno ad internet. Non è possibile dal solo tratto di codice evidenziato capire quali altre azioni possa eseguire il malware una volta verificato che la connessione sia presente, ma potenzialmente potrebbe servirsene per scaricare altri malware sulla macchina, creare una backdoor per dare il controllo ad un eventuale attaccante o semplicemente inviargli le informazioni raccolte.

BONUS: fare tabella con il significato delle singole righe del codice assembly:

push ebp	Metto il puntatore EBP in cima allo stack
mov ebp, esp	Copio il valore di ESP in EBP, serve a creare lo stack
push ecx	Metto il registro ECX in cima allo Stack
push 0 ; dwReserved	Metto 0 in cima allo stack. Dal commento ipotizziamo che si riferisca al valore di una dword riservata per la funzione sottostante che farà da parametro alla stessa
push 0 ; lpdwFlags	Metto 0 in cima allo stack. Ipotizzo che si riferisca ad un Flag riservato
call ds:InternetGetConnectedState	Chiamo la funzione InternetGetConnectedState
mov [ebp+var_4], eax	Copio il valore di eax in var_4. Probabilmente eax conterrà il valore di ritorno della funzione precedente
cmp [ebp+var_4], 0	Confronto var_4 con 0
jz short loc_40102B	Se var_4 è zero, salta all'indirizzo di memoria specificato
push offset aSuccessInterne ; "Success: Internet Connection\n"	Metto la stringa specificata in cima allo stack, che verrà utilizzata come argomento
call sub_40117F	Chiamo una subroutine (probabilmente per stampare la stringa dell'istruzione precedente)
add esp, 4	Aggiungo 4 al puntatore ESP (serve per pulire lo stack)
mov eax, 1	Copio il valore 1 in eax
jmp short loc_40103A	Salto incondizionato all'indirizzo specificato

loc_40102B:	Label con indirizzo di memoria, sarà il punto di arrivo del salto condizionato precedente
push offset aError1_1NoInte ; “Error1.1: No Internet\n”	Metto la stringa specificata in cima allo stack, che verrà utilizzata come argomento
call sub_40117F	Chiamo una subroutine (probabilmente per stampare la stringa dell'istruzione precedente)
add esp, 4	Aggiungo 4 al puntatore ESP (serve per pulire lo stack)
xor eax, eax	Pulisco il valore di eax
loc_40103A:	Label con indirizzo di memoria, sarà il punto di arrivo del salto incondizionato precedente
mov esp, ebp	Copio EBP in ESP, serve a chiudere lo stack
pop ebp	Rimuovo EBP dallo stack
retn	Ritorno alla funzione chiamante
sub_401000 endp	Marcatore di fine della subroutine indicata