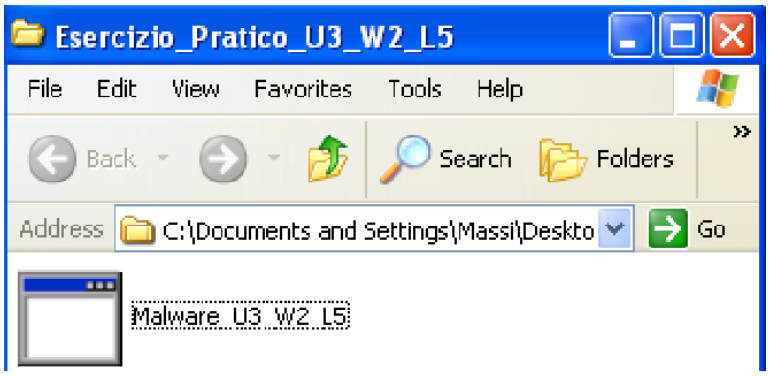
Il progetto di oggi prevede l'analisi del Malware presente sulla macchina Windows XP.



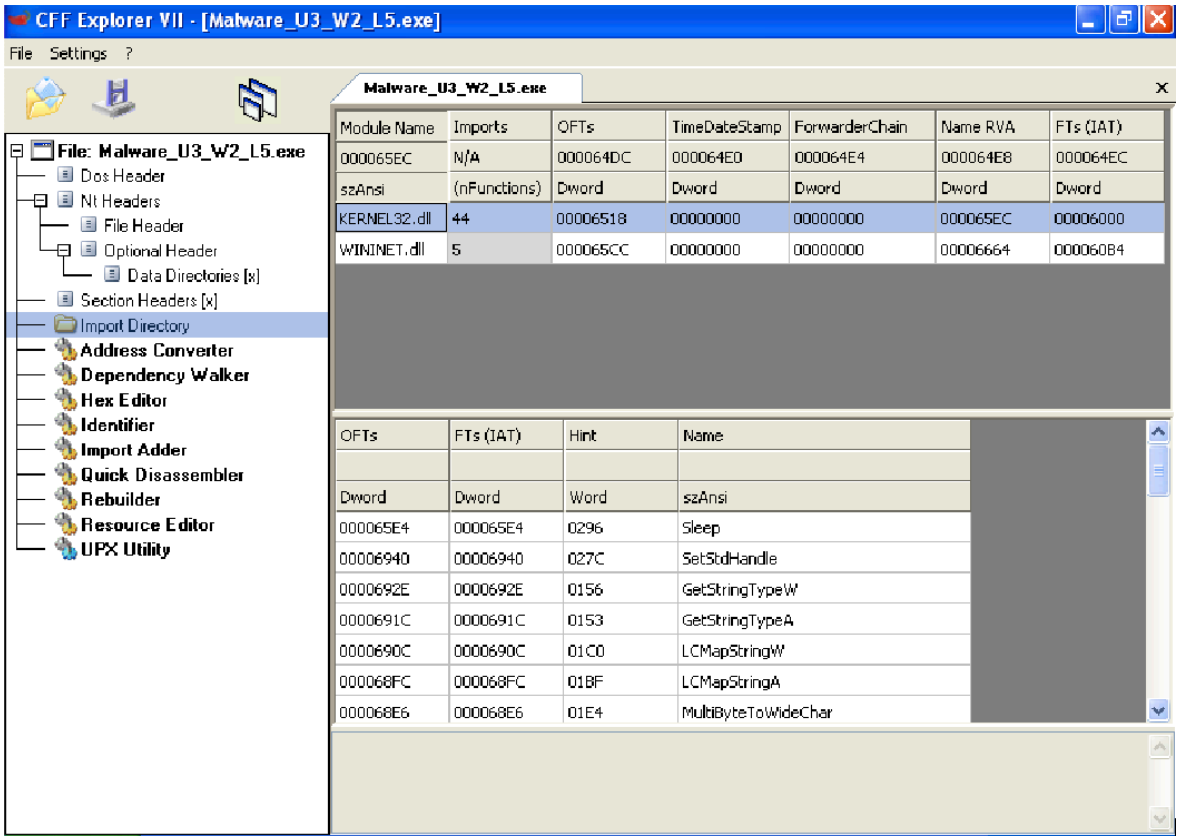
Nello specifico il nostro compito è di verificare quanto segue:

1) Quali librerie vengono importate dal file eseguibile.
2) Quali sono le sezioni di cui si compone il file eseguibile del malware.
3) Identificare i costrutti noti.
4) Ipotizzare il comportamento della funzionalità implementata.

ANALISI PRATICA

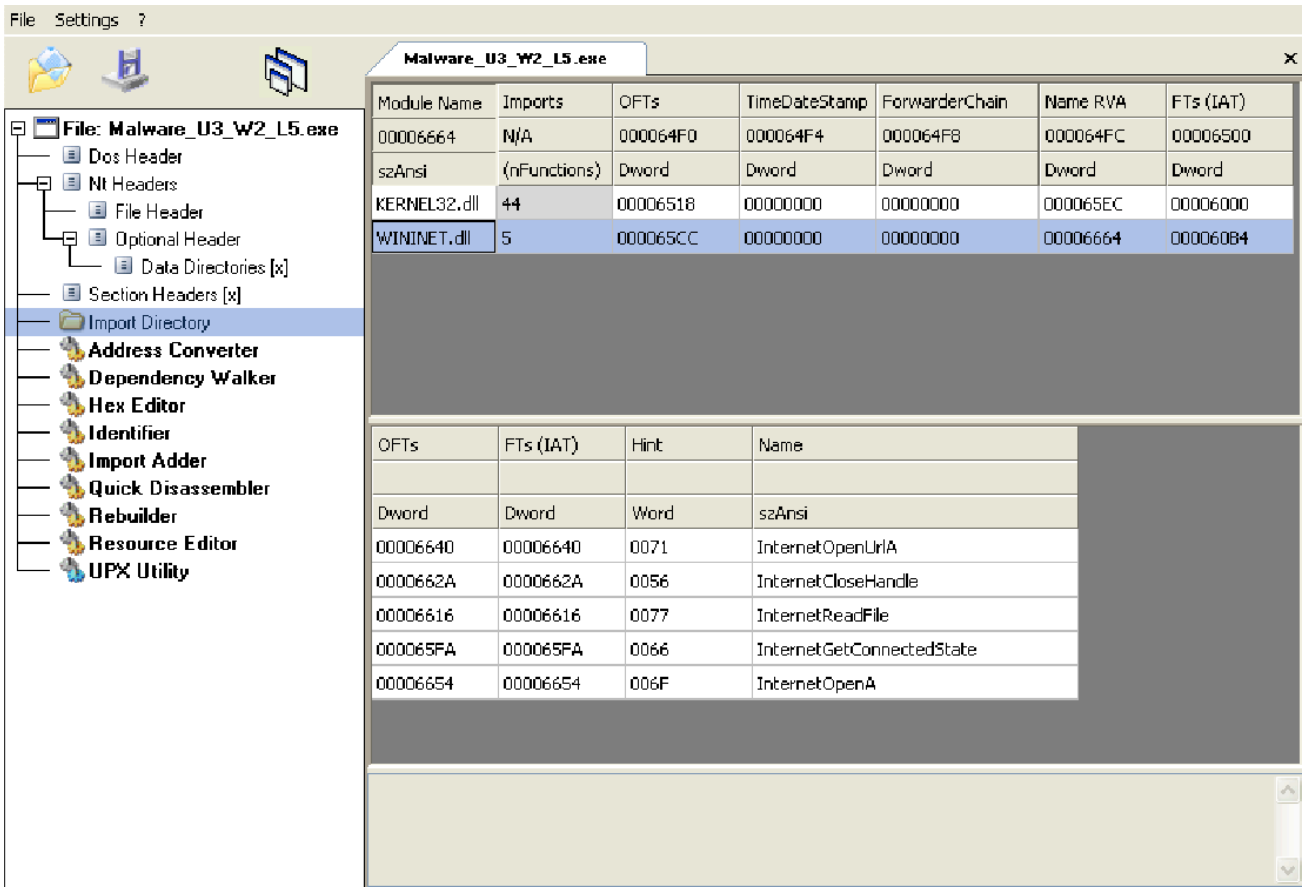
La tecnica di analisi utilizzata è la seguente;
Analisi Statica Basica.
Questa tecnica consiste nell'esaminare un eseguibile al fine di ottenere informazioni generiche circa le sue funzionalità e se un dato file è malevolo o meno.

1) Quali librerie vengono importate dal file eseguibile.
   Per far ciò ci siamo avvalsi del tool CFF Explorer che ci ha permesso di analizzare l'header del PE (Portable Executable), un formato che contiene delle informazioni necessarie al sistema operativo per capire come gestire il codice del file.



Come prima cosa ci siamo spostati nella sezione Import Directory. Qui abbiamo recuperato le informazioni sulle librerie importate dall'eseguibile con le sue funzioni. Nello specifico della prima slide abbiamo preso in esame la libreria:

KERNEL.32.dll; questa è una libreria piuttosto comune che contiene le funzioni principali per interagire con il sistema operativo, che permette al malware di manipolare i file e gestire la memoria



In questa seconda slide abbiamo visionato la presenza della libreria:

WININET.dll; questa libreria contiene le funzioni per l'implementazione di alcuni protocolli di rete come HTTP, FTP, NTP

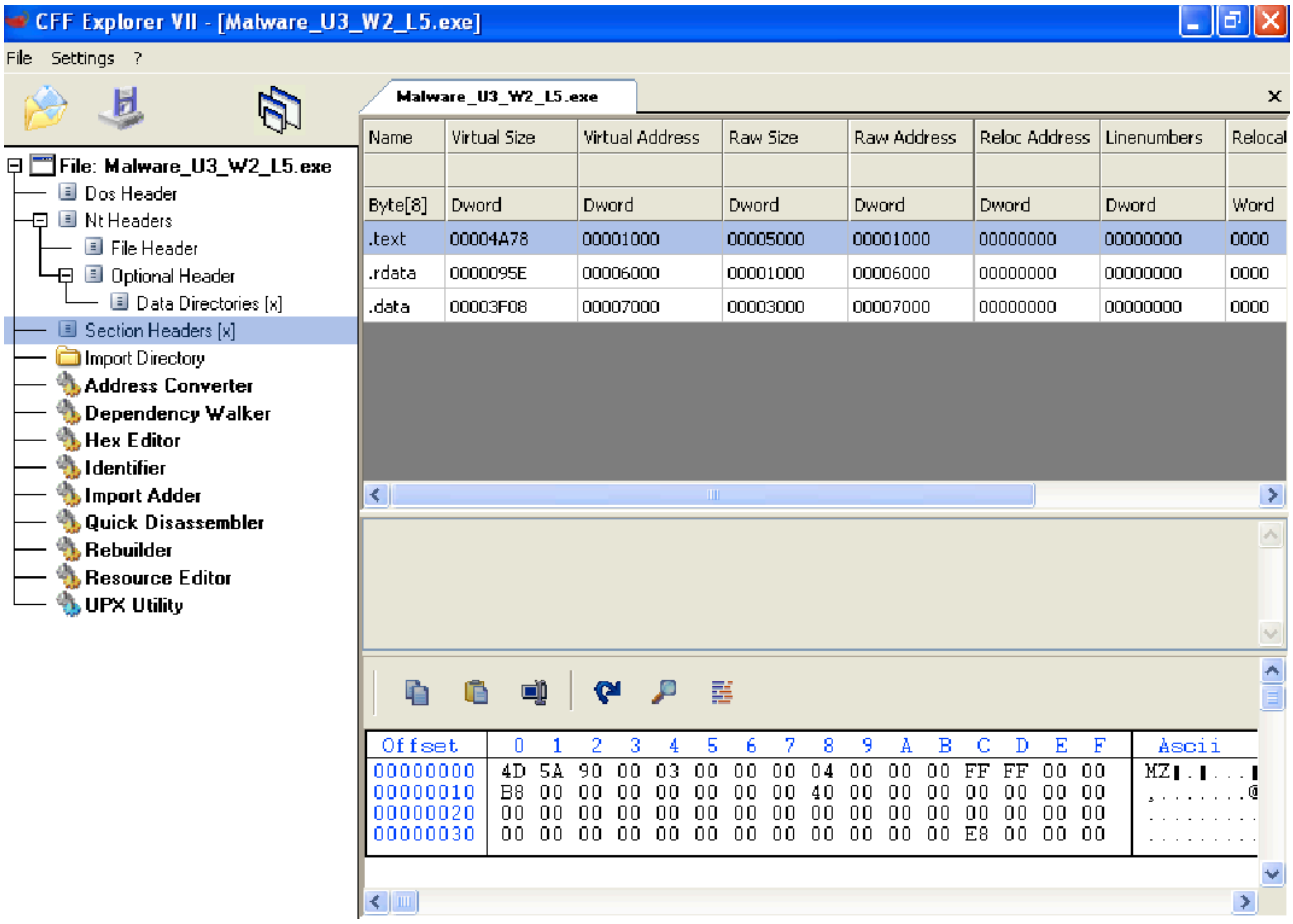
2) Quali sono le sezioni di cui si compone il file eseguibile del malware.

Nella slide seguente abbiamo riportato le sezioni delle quali si compone il software:

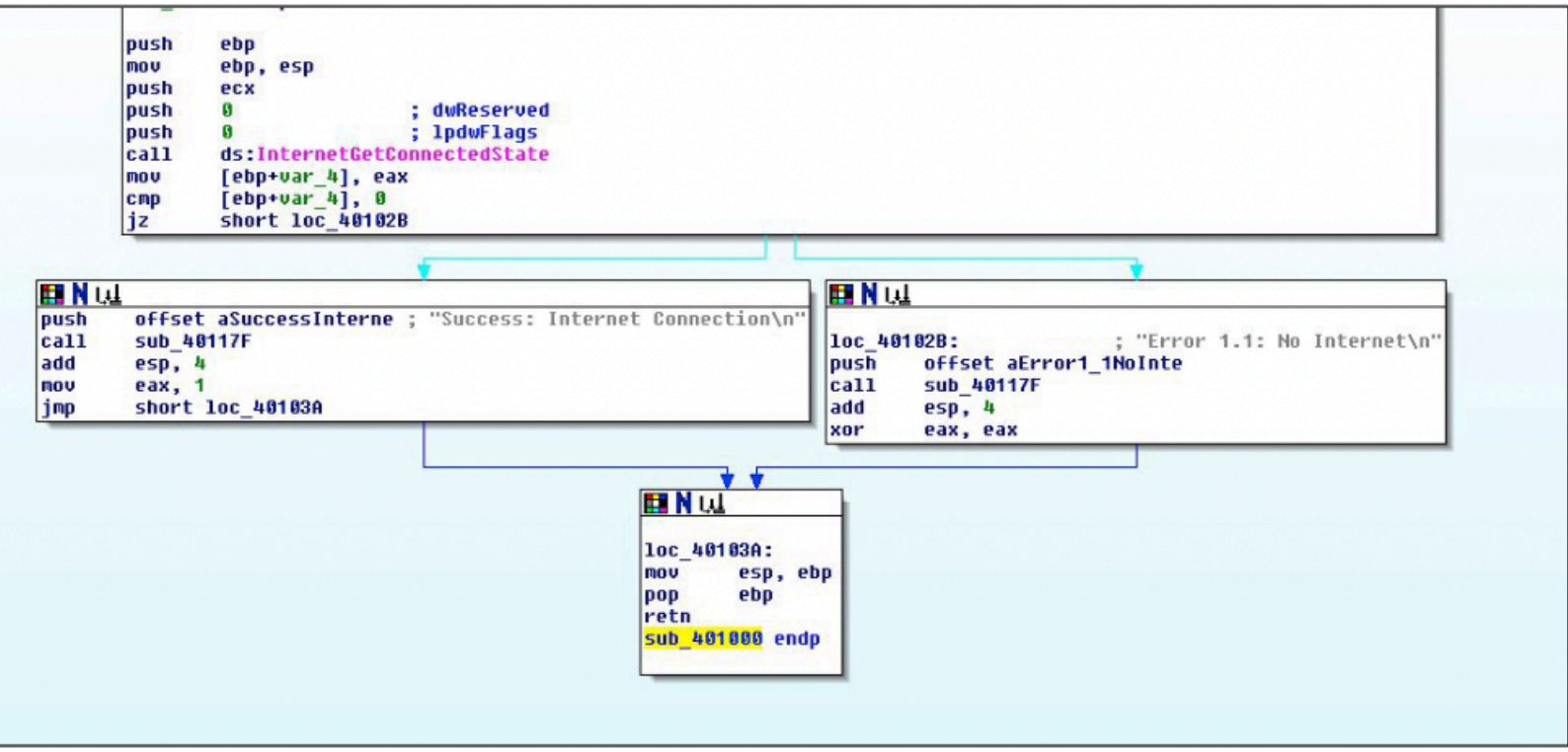
.text = Questa sezione contiene le righe di codice che la CPU eseguirà una volta avviato il software.

.rdata = Sezione che include le informazioni relative alle librerie e le loro funzioni, importate prima ed esportate poi dall'eseguibile

.data = Quest'ultima contiene i dati e le variabili globali del programma eseguibile. E' importante sapere che essendo variabili globali , esse sono accessibili da qualsiasi funzione dell'eseguibile.



3) Identificare i costrutti noti



In questa slide abbiamo un'analisi del malware tradotto in linguaggio Assembly. Questo linguaggio fa parte della tecnica di Analisi Statica Avanzata.
L'Assembly è un linguaggio univoco per una data architettura di un PC, nel nostro caso un architettura X86 quindi a 32 bit.

Abbiamo dunque identificato i costrutti noti cosi come segue:

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

In queste due righe avviene la creazione dello stack

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

In queste quattro righe vi è il passaggio dei parametri sullo stack tramite le istruzioni push e la chiamata della funzione

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

In queste due righe viene eseguito il ciclo If. Ciclo che verificherà se la connessione internet sarà attiva o meno

Queste quattro righe sono il risultato del controllo if con risposta di connessione attiva

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

Queste cinque righe sono il risultato del controllo if con risposta di connessione non attiva

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

Per quanto riguarda quest'ultimo blocco avviene la pulizia dello stack dopo che lo stato della connessione è stato verificato.

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

4) Ipotizzare il comportamento della funzionalità implementata

Ipotesi di comportamento:

In seguito alle analisi basiche, possiamo ipotizzare che per prima cosa dichiara la variabile var_4, subito dopo chiama la funzione InternetGetConnctedState, dopodiché si trova di fronte alla scelta data dall'IF e se il valore della funzione è 0, salta al loc_40102B altrimenti esegue il programma senza saltare loc ritornando una connessione attiva. Infine per entrambe le strade avviene la pulizia dello stack.

Ipotesi di scopo:

Caricando delle librerie che implementano protocolli di rete, nel mio caso ho preso in esame il protocollo NTP. Utilizzato spesso per attacchi DDoS.
L'attaccante produce requests verso i server utilizzando non il proprio IP ma quello del server bersaglio. In questo modo il server entrerà in un loop tentando di ritrasmettere a sé stesso i pacchetti mancanti.