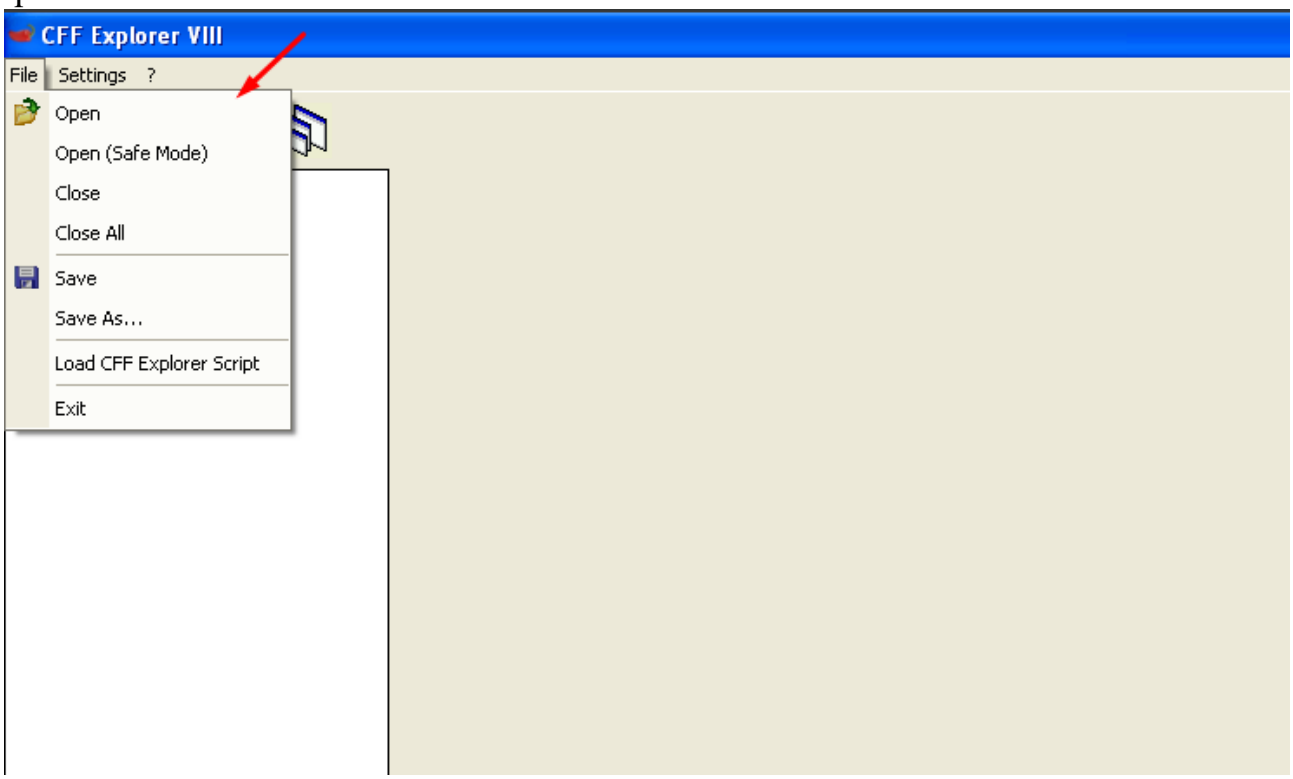
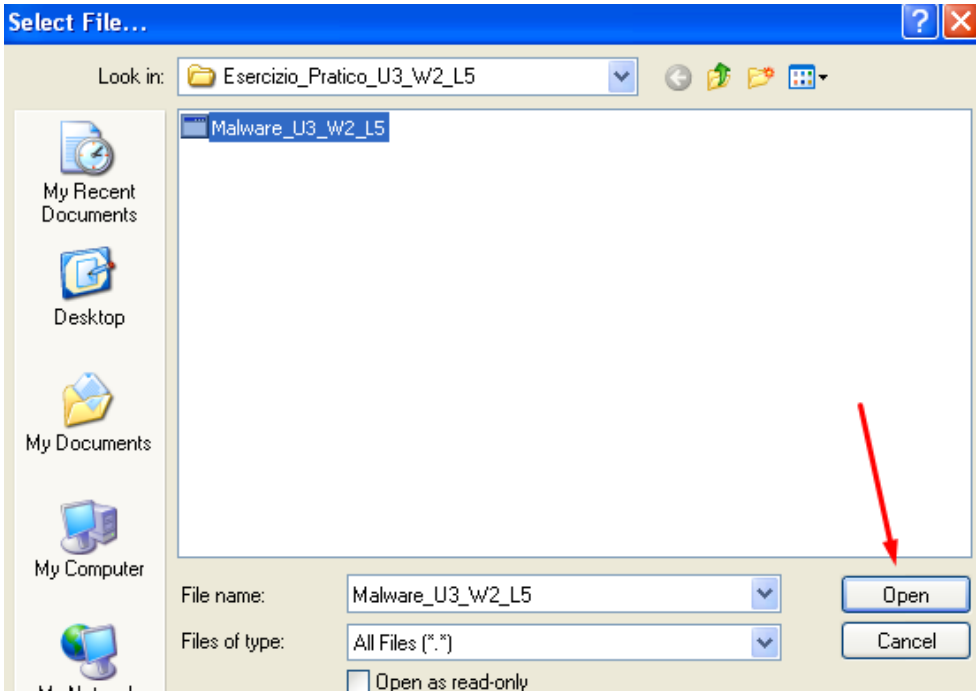
Alaimo Alessandro
13/01/2023

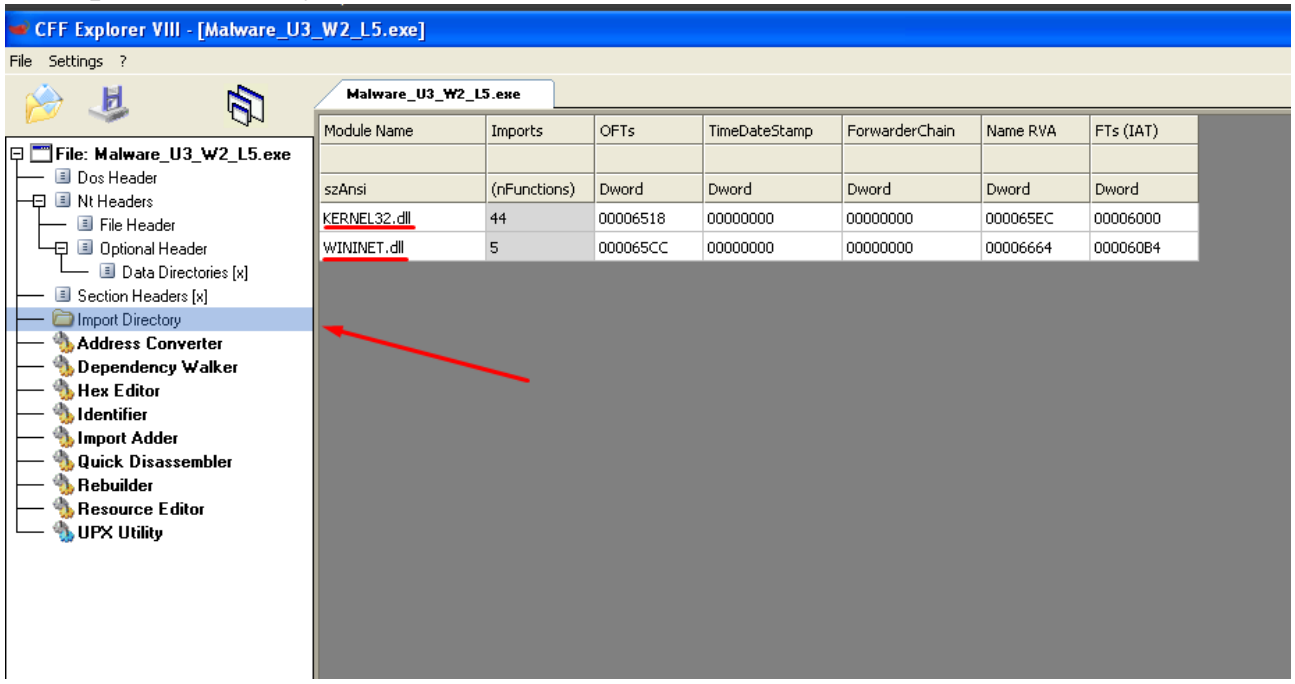
Report Weekend 10

Una volta controllato se la nostra macchina virtuale sia pressoché isolata, andando a disattivare eventuali schede di rete, controller USB ed assicurarci che non vi siano presenti cartelle condivise, possiamo iniziare ad analizzare il nostro Malware.
Andiamo ad avviare CFF Explorer ed andiamo ad aprire al suo interno il file in questione:





Fatto ciò andiamo a controllare quale librerie importa il Malware andando a cliccare su **Import Directory:**



Possiamo difatti notare le due librerie Kernel32.dll e Wininet.dll:

- KERNEL32.dll: È una libreria piuttosto comune che contiene le funzioni principali per interagire con il sistema operativo, nel nostro caso dalla libreria vengono importate 44 funzioni;

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
0000681E	0000681E	019F	HeapFree
0000682A	0000682A	022F	RtlUnwind
00006836	00006836	02DF	WriteFile
00006842	00006842	0199	HeapAlloc
0000684E	0000684E	00BF	GetCPInfo

0000685A	0000685A	00B9	GetACP
00006864	00006864	0131	GetOEMCP
00006870	00006870	02BB	VirtualAlloc
00006880	00006880	01A2	HeapReAlloc
0000688E	0000688E	013E	GetProcAddress
000068A0	000068A0	01C2	LoadLibraryA
000068B0	000068B0	011A	GetLastError
000068C0	000068C0	00AA	FlushFileBuffers
000068D4	000068D4	026A	SetFilePointer
00006950	00006950	001B	CloseHandle

- WININET.dll: Libreria che contiene le funzioni per l'implementazione di alcuni protocolli di rete come HTTP, FTP, NTP, andando ad importare 5 funzioni.

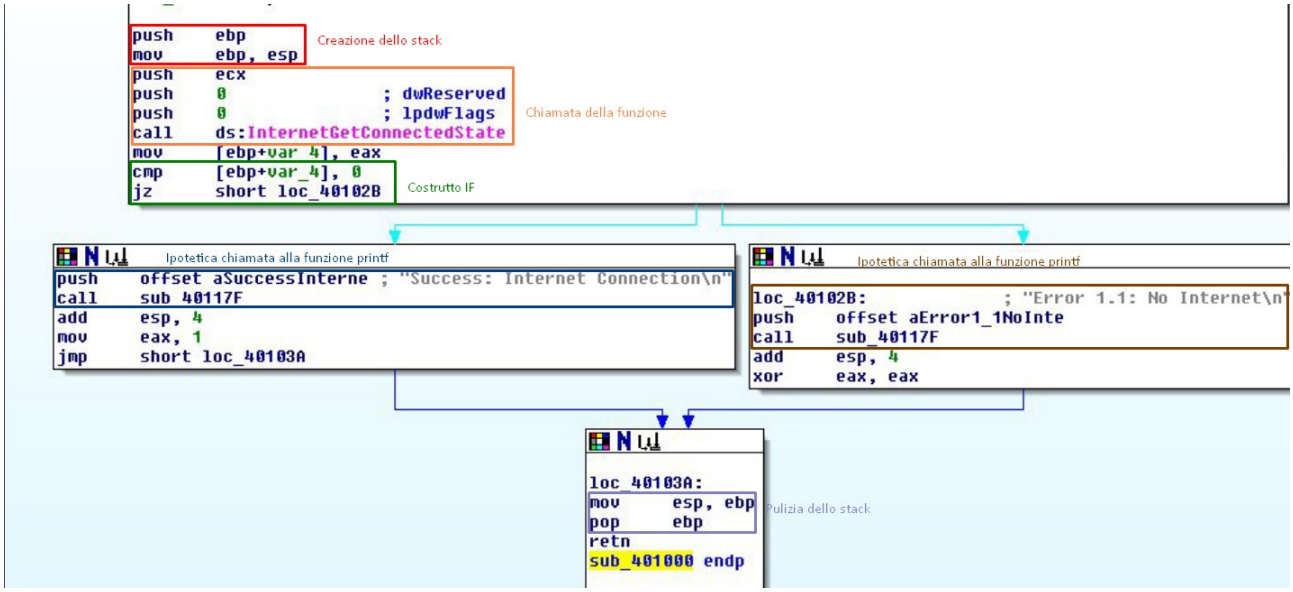
00006640	00006640	0071	InternetOpenUrlA
0000662A	0000662A	0056	InternetCloseHandle
00006616	00006616	0077	InternetReadFile
000065FA	000065FA	0066	InternetGetConnectedState
00006654	00006654	006F	InternetOpenA

Ora andiamo a visualizzare le sezione del Malware andando a cliccare su **Section Headers [x]** le quali sono **.text** , **.rdata** , **.data:**

Malware_U3_W2_L5.exe

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00004A78	00001000	00005000	00001000	00000000	00000000	0000	0000	60000020
.rdata	0000095E	00006000	00001000	00006000	00000000	00000000	0000	0000	40000040
.data	00003F08	00007000	00003000	00007000	00000000	00000000	0000	0000	C0000040

.text: Questa sezione contiene tutte le righe di codice che la CPU eseguirà;
.rdata: Questa è la sezione che contiene le variabili locali;
.data: Questa sezione contiene le variabili globali.



Andando ad analizzare il codice in Assembly della slide 3, possiamo andare ad identificare 6 costrutti noti, e sono:

- push ebp
  mov ebp, esp
  Con queste due stringhe di comando andiamo a creare lo stack.

- push ecx
  push 0 ;dwReserved
  push 0 ;lpdwFlags
  call ds:InternetGetConnectedState
  Inserendo 3 parametri, viene chiamata la funzione **InternetGetConnectedState** per controllare se il dispositivo è connesso ad internet.

- cmp [ebp+var_4], 0
  jz short loc_40102B
  Queste due stringhe rappresentano un IF, il quale controlla se il risultato è 0 ed in caso salta fino all'indirizzo di memoria scritto per continuare il codice.

- push offeset aSuccessInterne ;"Success: Internet Connection\n"
  call sub_40117F
  Tramite questa chiamata possiamo ipotizzare che chiama la funzione printf per stampare la stringa **Success: Internet Connection.**

- push offeset aError1_1NoInte ;"Success: Internet Connection\n"
  call sub_40117F
  Tramite questa chiamata possiamo ipotizzare che chiama la funzione printf per stampare la stringa **Error 1.1: Internet.**

- mov esp, ebp
  pop ebp
  Tramite queste 2 stringhe di comando si svuota lo stack.

Possiamo dire con certezza che il nostro Malware controlla se la macchina ha accesso alla connessione ad internet, stampando l'esito del controllo.

Infine, come richiesta bonus, andiamo ad analizzare tutte le righe del codice Assembly.

push ebp: Passare i parametri EBP in cima allo stack;

mov ebp, esp: Copia il valore della sorgente (ESP), nel destinatrio (EBP);

push exc: Passare i paramentri exc in cima allo stack;

push 0 ; dwReserved: Passare il valore dei parametri (valore 0) di una variabile;

push 0 ; lpdwFlags: Passare il valore dei parametri (valore0) di una variabile;

call ds:InternetGetConnectedState: chiamata di funzione tramite ds: che crea un nuovo EIP per la funzione chiamata;

(https://stackoverflow.com/questions/3819699/what-does-ds40207a-mean-in-assembly)

mov [ebp+var_4], eax: Copia il valore della sorgente (eax), nel destinatrio (ebp+var_4);

cmp [ebp+var_4], 0: Mette a confronto destinatario e sorgente per restituire un valore binario;

jz short loc_40102B: *(salta alla locazione di memoria specificata se ZF=1):* Salta alla locazione di memoria se ZF = 1 riferito al cmp precedente;

push offset aSuccesInterne   ; "Succes: Internet Connection\n": Passa l'alloggio di memoria aSuccesInterne in cima allo stack;

call sub_40105F: Chiamata di funzione;

add esp, 4: Somma il valore della sorgente (4) con quello del destinatario (ESP);

mov eax, 1: Copia il valore della sorgente (1), nel destinatario (EAX);

jmp shot loc_40103A: Fa un salto all'istruzione nella locazione data (40103A);

loc_40102B: Label nel quale la stringa "**jz short loc_40102B:**" arriva;

push offset_aError1_1NoInter: Passa l'alloggio di memoria offset_aError1_1NoInter in cima allo stack;

call sub_40117F: Chiamata di funzione;

add ESP, 4: Somma il valore della sorgente (4) con quello del destinatario (ESP);

xor eax,eax: Inizializza il valore di EAX a zero tramite operatore logico XOR (se i valori sono uguali restituisce 0);

mov esp,ebp: Copia il valore della sorgente (EBP), nel destinatrio (ESP);

pop ebp: Pulizia dello stack;

retn: Pone fine ad una procedura;

(https://stackoverflow.com/questions/1396909/ret-retn-retf-how-to-use-them)

sub_401000 endp: Segna la fine procedura precedentemente nominata (sub_401000).

https://learn.microsoft.com/en-us/cpp/assembler/masm/endp?view=msvc-170