

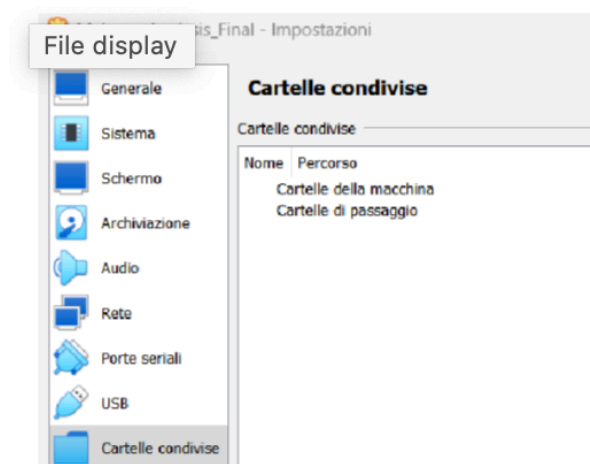
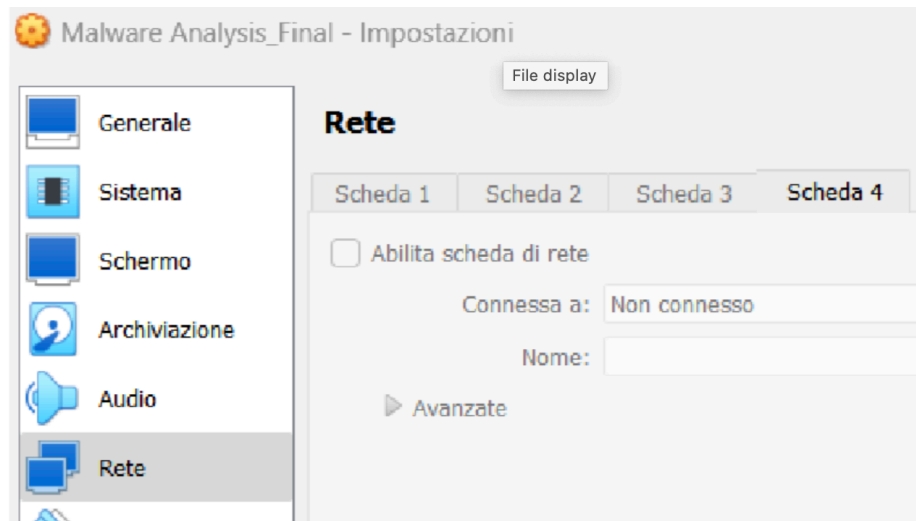
Malware Analysis - Introduzione

Oggi analizzeremo il malware: **Malware Build Week U3.**

Quest'ultimo verrà analizzato sulla macchina virtuale Windows XP "Malware Analysis_Final" installata su VirtualBox.

Affinché la nostra malware analysis vada a buon fine, ci assicureremo che durante le analisi, specialmente in quella dinamica, ci ritroveremo a lavorare in un ambiente isolato, in modo tale da non permettere al malware di propagarsi sul nostro pc o che quest'ultimo si propaghi sulla nostra rete. Inoltre, per precauzione, creeremo anche delle istantanee della macchina, così da avere la nostra macchina allo stato iniziale in caso qualcosa andasse storto.

Quindi assicuriamoci che sulla nostra macchina virtuale che non ci siano schede di rete abilitate, non ci siano cartelle condivise e che non ci siano dispositivi usb collegati.

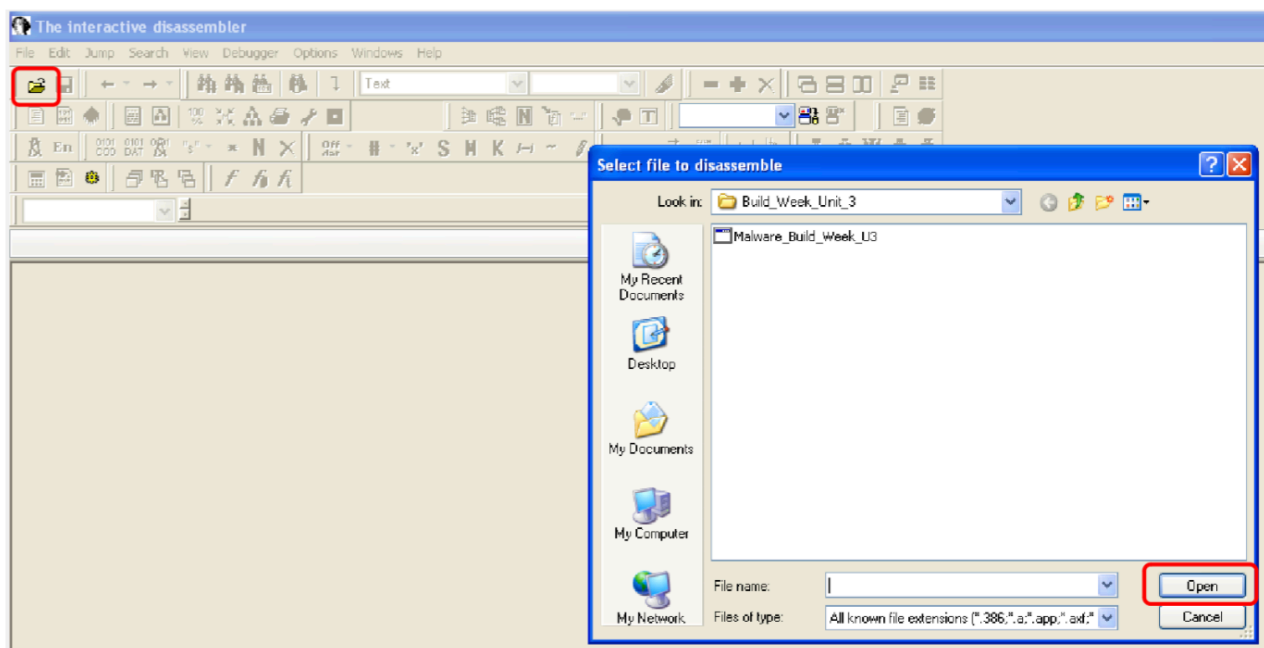


Analisi statica

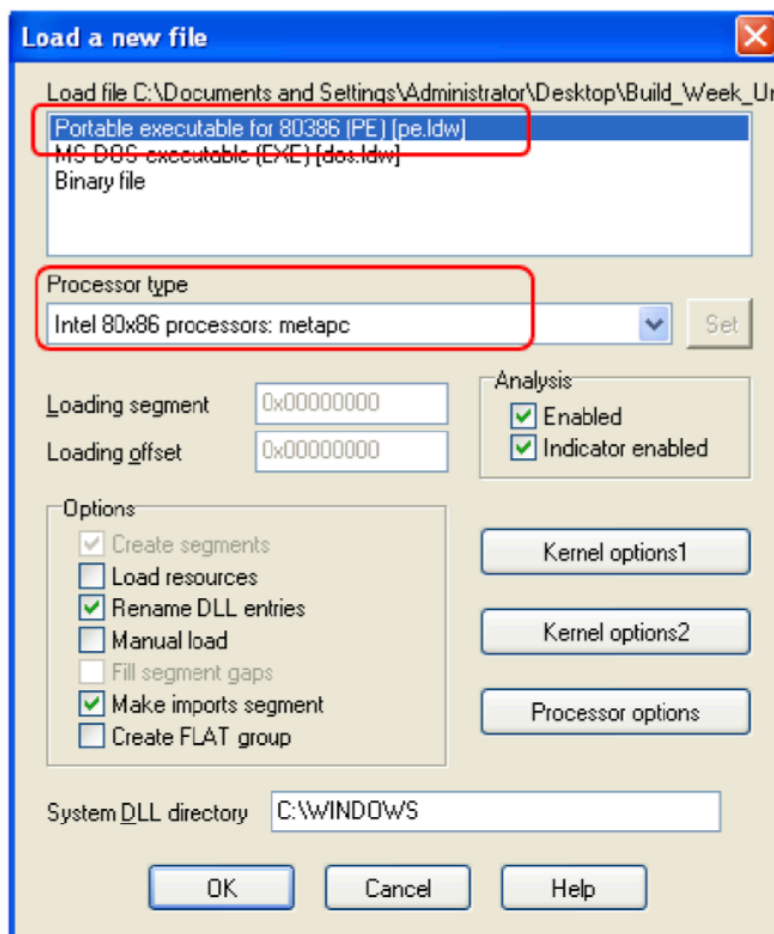
In questa parte ci concentreremo nell'analisi delle variabili locali, dei parametri e della funzione Main() .

Un tool fondamentale per questo tipo di analisi è IDA (Interactive DisAssembler) è uno strumento di disassemblaggio usato nell'ambito cyber per l'analisi dei malware. Questo software ci consente di esaminare in linguaggio Assembly il codice binario di un programma o file eseguibile.

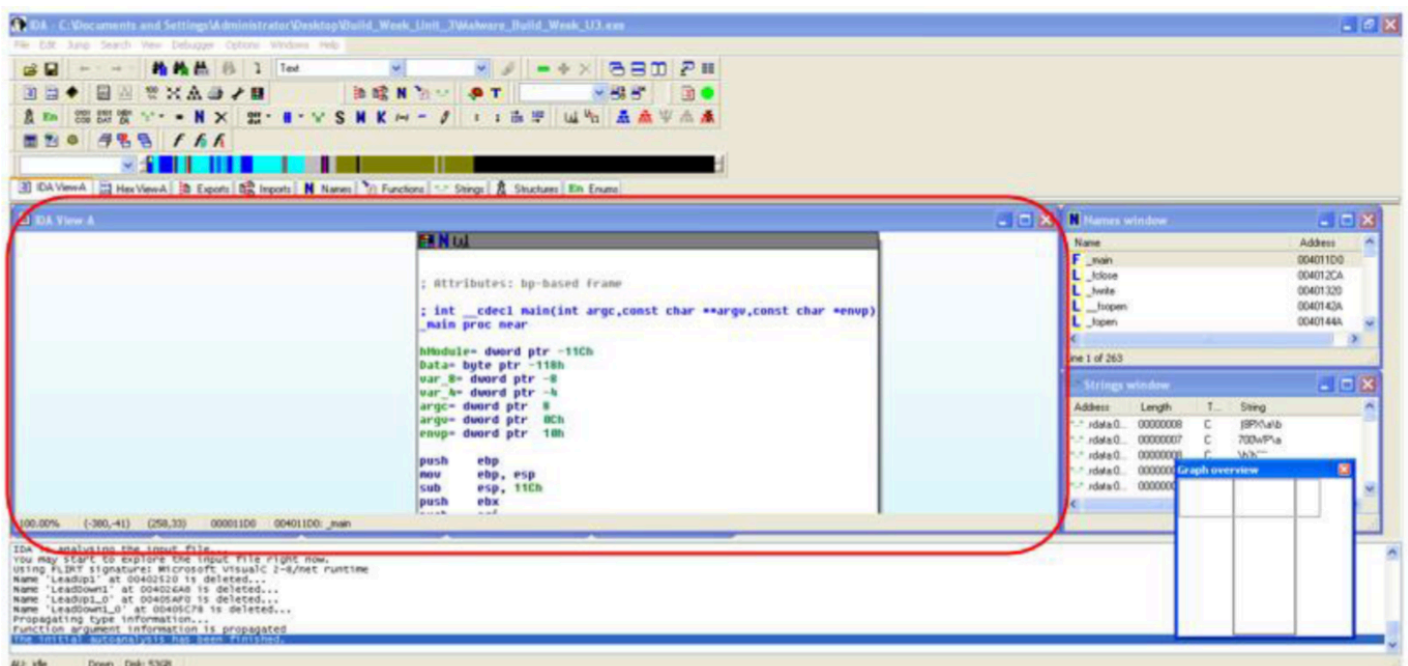
Quindi avviamo IDA e clicchiamo sull'icona in alto a sinistra a forma di cartella aperta e cerchiamo il nostro file da analizzare.



Successivamente Ida ci presenterà una finestra dove ci farà selezionare l'architettura del processo e il formato del file. In questo caso è stato identificato sia il formato corretto che il topo di processore. Quindi clicchiamo su “OK”.



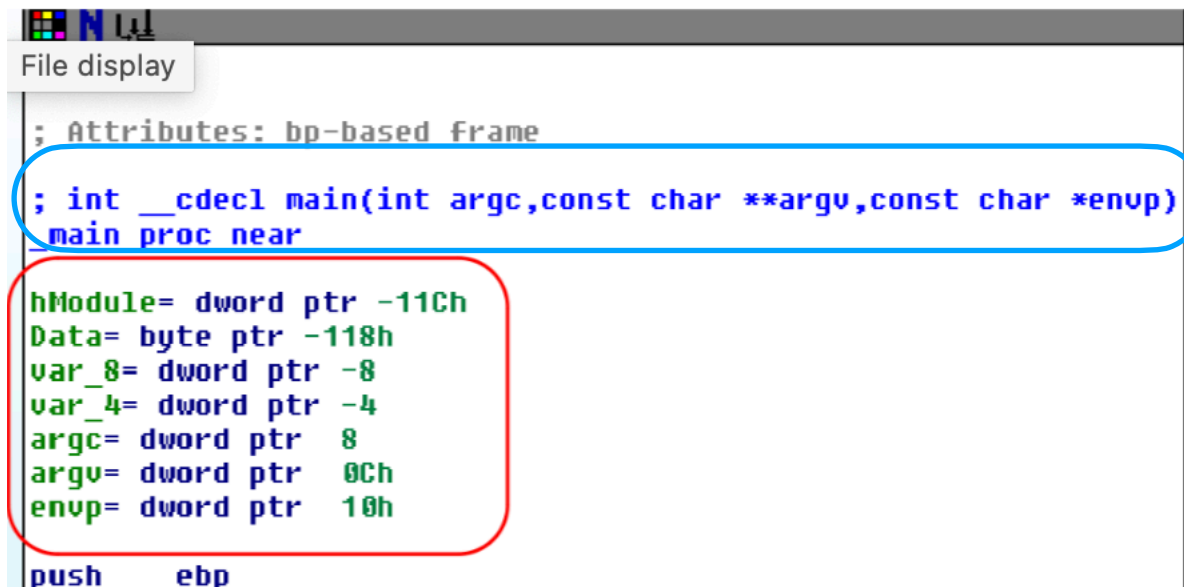
Vediamo quindi l'interfaccia dell'analisi di IDA. Il pannello grande al centro si chiama Disassembly Panel e ci mostra le funzioni principali del file selezionato in linguaggio Assembly come i salti, le variabili locali, funzioni definite e i parametri.



Analisi della funzione Main()

Ora concentriamoci sull'analizzare la nostra funzione Main();

Vediamo che questa funzione è stata individuata come “`int Main()`” e sotto di essa vediamo le variabili **locali**.



```

; Attributes: bp-based frame

; int __cdecl main(int argc,const char **argv,const char *envp)
main proc near

hModule= dword ptr -11Ch
Data= byte ptr -118h
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push    ebp

```

Facciamo chiarezza su come sono strutturate le variabili e i parametri su IDA;

Nome variabile/parametro= formato -offset rispetto al registro EBP

Il registro **EBP** o **Extender Base Pointer** viene utilizzato per la creazione di un punto di riferimento alla base dello stack della funzione, mentre l'offset rispetto a quest'ultimo, serve per indicare la distanza in byte rispetto al registro in esadecimale.

Inoltre per distinguere le variabili locali dai parametri facciamo una distinzione: i parametri si trovano in un offset positivo rispetto al registro EBP, viceversa per le variabili locali.

Date le affermazioni dette qui sopra, capiamo che 3 parametri passano per la nostra funzione Main(): **argc**, **argv** ed **envp**.

E le variabili dichiarate all'interno della funzione Main() sono 4: **hModule**, **Data**, **var_8** e **var_4**.

```

File display tes: bp-based frame

; int __cdecl main(int argc,const char **argv,const char *envp)
_main proc near

hModule= dword ptr -11Ch
Data= byte ptr -118h
var_8= dword ptr -8
var_4= dword ptr -4

argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push    ebp

```

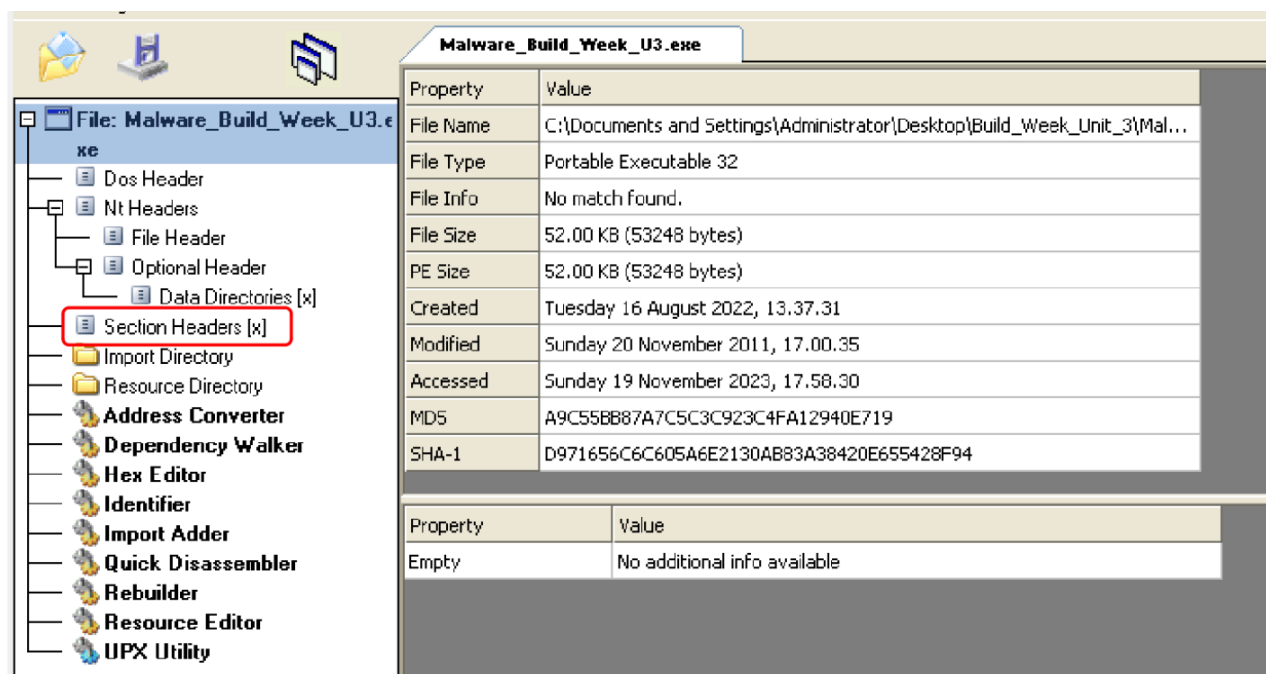
Analisi delle sezioni del file eseguibile

Passiamo all'analisi del file eseguibile.

Useremo un altro tool: CFF Explorer.

Questo strumento avanzato di analisi e disassemblaggio ci aiuta ad esaminare i file eseguibili (es .exe .dll...) in binario per poi comprenderli.

Apriamo quindi il programma, selezioniamo in alto a sinistra l'icona di una cartella per poi selezionare il nostro file da analizzare.



Selezioniamo a sinistra la voce “Section Headers” e a destra vedremo un pannello dove visualizziamo le sezioni del file. In questo pannello vediamo delle tabelle, queste sono delle sezioni del malware.

Vediamo la colonna “name” dove riporta il nome della sezione e le altre tabelle ci danno altre informazioni su di esso. Selezionando una di queste tabelle ci verranno restituite le informazioni dettagliate rispetto alla loro sezione.

The screenshot shows a software interface for analyzing a file named "Malware_Build_Week_U3.exe". The top section displays a table of section headers. Below this, the details for the selected ".text" section are shown, including its virtual size, address, and entry point. At the bottom, a memory dump is displayed with hexadecimal and ASCII representations.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Cha
000001D8	000001E0	000001E4	000001E8	000001EC	000001F0	000001F4	000001F8	000001FA	00C
File display	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dw
.text	00005646	00001000	00006000	00001000	00000000	00000000	0000	0000	60C
.rdata	000009AE	00007000	00001000	00007000	00000000	00000000	0000	0000	40C
.data	00003EA8	00008000	00003000	00008000	00000000	00000000	0000	0000	CO
.rsrc	00001A70	0000C000	00002000	0000B000	00000000	00000000	0000	0000	40C

This section contains:

Code Entry Point: 00001487

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	55	8B	EC	51	6A	00	8D	45	FC	50	6A	00	68	3F	00	0F	U iQj.. EuPj.h?..
00000010	00	6A	00	6A	00	6A	00	68	54	80	40	00	68	02	00	00	.j.j.j.hT!@.h ..
00000020	80	FF	15	04	70	40	00	85	C0	74	07	B8	01	00	00	00	!y!lp@.. At ..
00000030	EB	49	8B	4D	0C	51	8B	55	08	52	6A	01	6A	00	68	4C	èI M Q U Rj j.hL
00000040	80	40	00	8B	45	FC	50	FF	15	00	70	40	00	85	C0	74	!@.. EuPy ..p@.. At
00000050	11	8B	4D	FC	51	FF	15	2C	70	40	00	B8	01	00	00	00	! MuQy ..p@.. ..
00000060	EB	19	68	48	80	40	00	E8	2D	02	00	00	83	C4	04	8B	è hH @.è-
00000070	55	FC	52	FF	15	2C	70	40	00	33	C0	8B	E5	5D	C3	CC	UuRy ..p@..3A à A
00000080	55	8B	EC	83	EC	18	56	57	C7	45	EC	00	00	00	00	C7	U i i VWÇEi...Ç
00000090	45	E8	00	00	00	00	C7	45	F8	00	00	00	00	C7	45	F0	Eè...ÇEø...ÇEè
000000A0	00	00	00	00	C7	45	F4	00	00	00	00	83	7D	08	00	75	...ÇEó... }!u

Analizziamo quindi le sezioni delle tabelle all'interno del file.

- `.text` contiene il codice eseguibile del programma. inoltre è riportato l'indirizzo dell'entry point del file, ovvero la prima istruzione eseguita dalla CPU. L'entry point dell'eseguibile si trova all'indirizzo 00001487.
- `.rdata` contiene dati di lettura che il programma può leggere ma non modificare durante l'esecuzione. Il programma indica che:
- `.rdata` inizia all'indirizzo 00007000 .
- l'indirizzo di memoria della Import Directory (directory delle librerie e funzioni richieste dal programma durante l'esecuzione) è 000074EC .
- l'indirizzo di memoria della Address Table Directory (tabella che contiene gli indirizzi delle funzioni specifiche all'interno delle librerie dinamiche (DLL) è 00007000 .

This section contains:

File display

Data: 00007000

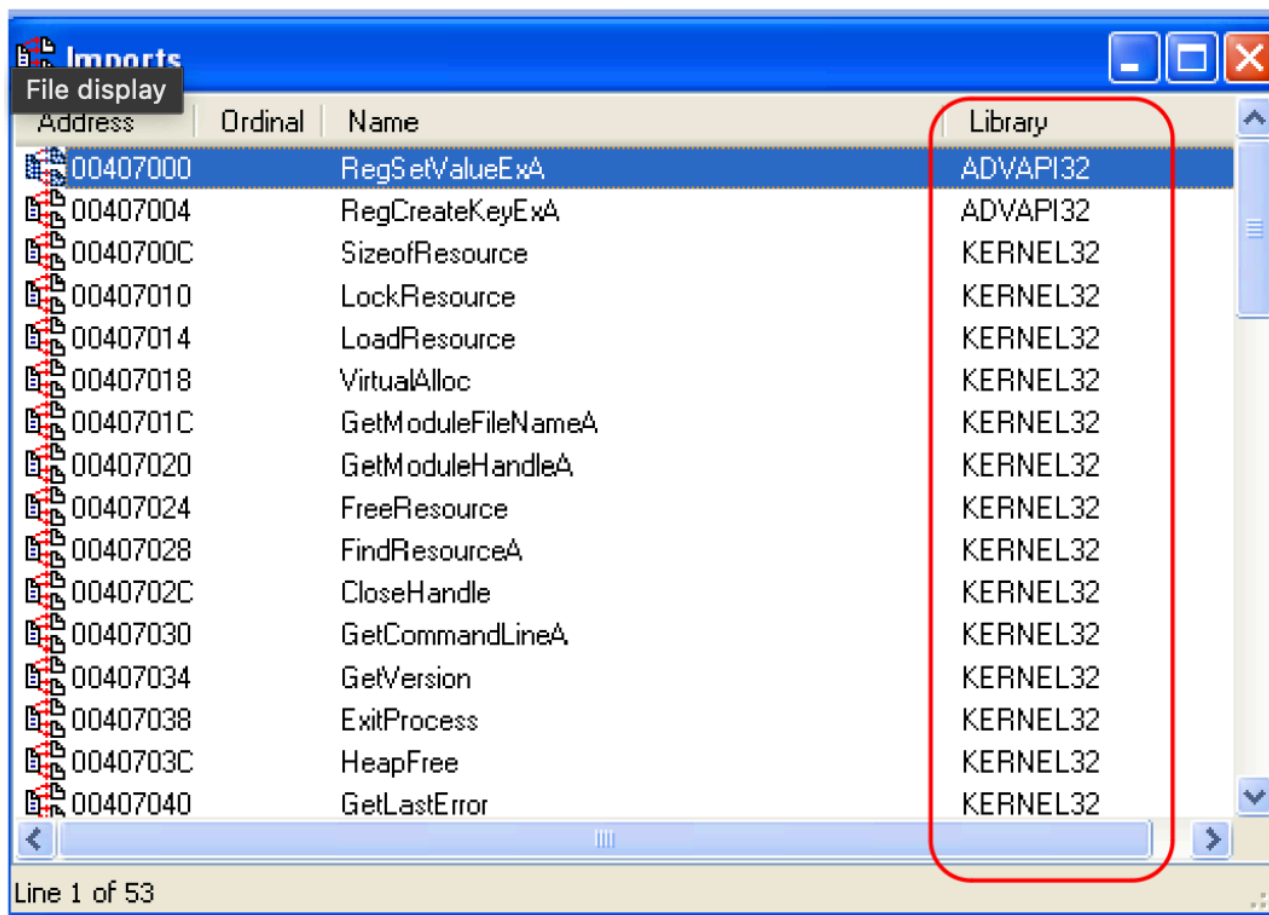
Import Directory: 000074EC

Import Address Table Directory: 00007000

- `.data` contiene dati variabili inizializzate che il programma legge e modifica durante l'esecuzione. In particolare, contiene le variabili globali definite nel codice eseguibile.
- `.rsrc` contiene risorse come dati propri del programma che quest'ultimo utilizza durante l'esecuzione del programma. La Resource Directory si trova all'indirizzo di memoria 0000C000.

Analisi delle librerie importate dal malware

Adesso analizziamo con CFF Explorer le librerie importate dal malware. Andiamo nella scheda Imports e analizziamola.



Vediamo che il malware importa le librerie ADVAPI32 e KERNEL32.

Possiamo fare delle ipotesi su quali siano le funzioni del malware.

Sappiamo che: **ADVAPI32.dll** si occupa dei servizi avanzati di Windows e delle sue funzioni quali la gestione degli account utente, la gestione dei servizi, della sicurezza e altri servizi avanzati come manipolare il registro di sistema, funzioni di crittografia e autenticazione e autorizzazione.

Sopra infatti vediamo le funzioni richiamate all'interno di questa libreria,

RegCreateKeyExA crea o apre la chiave del registro di sistema specifica.

RegSetValueExA imposta il tipo e i dati di uno specifico valore in una chiave del registro di sistema.

Mentre **KERNEL32.dll** è una delle librerie di sistema principali di Windows, contiene funzioni di basso livello che coinvolgono direttamente l'hardware nella gestione dei processi. Tra le sue funzioni troviamo quella di allocazione

della memoria, creazione dei processi e gestione dei file ecc... in sintesi è essenziale per l'esecuzione delle applicazioni Windows.

Ipotesi sulle funzioni del malware

Possiamo fare quindi due **ipotesi**:

La prima è che il malware usi la libreria **ADVAPI32.dll** per modificare le chiavi di registro al fine di ottenere la persistenza, ovvero che il malware venga avviato all'avvio del sistema.

La seconda è che il malware sia un dropper, cioè un malware che contiene dentro di sé un altro malware. Se così fosse il malware che viene rilasciato si trova nella sezione risorse (.rsrc) del file.

Infatti nella libreria **KERNEL32.dll** le funzione richiamate all'interno di quest'ultima, sono utilizzate tipicamente dai dropper per l'estrazione di un malware contenuto nella sezione delle risorse. Infine notiamo anche le funzioni "CreateFileA" e "WriteFile", quindi molto probabilmente il dropper dopo aver rilasciato il malware sarebbe in grado di salvarlo nel disco.

Name	Library	Name	Library
File display w	KERNEL32	SetEndOfFile	KERNEL32
GetStringTypeA	KERNEL32	SetFilePointer	KERNEL32
GetStringTypeW	KERNEL32	SetHandleCount	KERNEL32
SizeofResource	KERNEL32	SetStdHandle	KERNEL32
LockResource	KERNEL32	SizeofResource	KERNEL32
LoadResource	KERNEL32	TerminateProcess	KERNEL32
VirtualAlloc	KERNEL32	UnhandledExceptionFilter	KERNEL32
GetModuleFileNameA	KERNEL32	VirtualAlloc	KERNEL32
GetModuleHandleA	KERNEL32	VirtualFree	KERNEL32
FreeResource	KERNEL32	WideCharToMultiByte	KERNEL32
FindResourceA	KERNEL32	WriteFile	KERNEL32
CloseHandle	KERNEL32	CloseHandle	KERNEL32
GetCommandLineA	KERNEL32	CreateFileA	KERNEL32
GetVersion	KERNEL32	ExitProcess	KERNEL32
		FindResourceA	KERNEL32

Malware analysis

Ritornando su IDA nella sezione del Disassembly Panel di IDA, premendo la barra spaziatrice visualizziamo la versione testuale del codice

```
.text:00401011      push     0                ; dwOptions
.text:00401013      push     0                ; lpClass
.text:00401015      push     0                ; Reserved
.text:00401017      push     offset SubKey    ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"
.text:00401019      push     00000002h        ; hKey
.text:00401021      call     ds:RegCreateKeyExA
.text:00401027      test     eax, eax
.text:00401029      jz       short loc_401032
.text:0040102B      mov      eax, 1
.text:00401030      jmp      short loc_40107B
.text:00401032      ; -----
.text:00401032      loc_401032:               ; CODE XREF: sub_401000+29↑j
.text:00401032      mov      ecx, [ebp+cbData]
.text:00401033      push     ecx                ; cbData
.text:00401035      mov      edx, [ebp+lpData]
.text:00401036      push     edx                ; lpData
.text:00401039      push     1                ; dwType
.text:0040103A      push     0                ; Reserved
.text:0040103C      push     offset ValueName ; "GinaDLL"
.text:0040103E
```

Dettagli della funzione alla locazione di memoria 00401021

Notiamo subito che viene richiamata la funzione RegCreateKeyEaX a quell'indirizzo di memoria.

Intuiamo che questa funzione ha come scopo quello di creare una chiave specifica del registro di sistema o se già esiste, chiede semplicemente di aprirla. Qui sotto vi mostro i parametri che richiede la funzione. Mentre analizzando il codice Assembly su IDA vediamo che quest'ultimi sono passati usando l'istruzione "push"

```
LSTATUS RegCreateKeyExA(
    [in]          HKEY          hKey,
    [in]          LPCSTR        lpSubKey,
    DWORD         Reserved,
    [in, optional] LPSTR        lpClass,
    [in]          DWORD         dwOptions,
    [in]          REGSAM        samDesired,
    [in, optional] const LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    [out]          PHKEY         phkResult,
    [out, optional] LPDWORD      lpdwDisposition
);
```

Locazione di memoria 00401017

```

.text:00401003      push     ecx
.text:00401004      push     0             ; lpdwDisposition
.text:00401006      lea      eax, [ebp+hObject]
.text:00401009      push     eax             ; phkResult
.text:0040100A      push     0             ; lpSecurityAttributes
.text:0040100C      push     0F003Fh        ; samDesired
.text:00401011      push     0             ; dwOptions
.text:00401013      push     0             ; lpClass
.text:00401015      push     0             ; Reserved
.text:00401017      push     offset SubKey   ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"..
.text:0040101C      push     80000002h       ; hKey
.text:00401021      call     ds:RegCreateKeyExA

```

Analizziamo anche questa locazione di memoria 00401017.

Sappiamo che viene passato il parametro **lpSubKey** della funzione **RegCreateKeyExA**, questa specifica la sottochiave di registro che si viene a creare o che va a modificare.

In breve una sottochiave o subkey di registro, è una cartella del registro che è contenuta nella cartella principale di registro. In questo caso notiamo che il parametro sta dicendo alla funzione di creare o modificare una chiave di registro dell'area HKLM o HKEY_LOCAL_MACHINE, in essa sono contenute le configurazioni della macchina.

La chiave che va a modificare è nella cartella di registro HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion, questa contiene informazioni sulla versione corrente del sistema operativo Windows.

Analisi locazioni di memoria 00401027 - 00401029

Nelle locazioni di memoria 00401027 - 00401029 il codice in Assembly sembrerebbe riproduce un costrutto if in linguaggio C.

Tradotto il codice in C è il seguente:

```
Int a; #eax
```

```
Int c; #ecx
```

```
Int d; #[ebp+cbData]
```

qui ci sarebbe una parte di codice che assegna i valori alle variabili

```
If    (a == 0)    {
        c = d;
}    else        {
        a = 1;
}
```

Questa prima istruzione **test eax, eax**, esegue semplicemente l'operatore logico AND con se stesso al valore del registro.

Distinguiamo invece l'operatore test, che invece di modificare il valore del registro eax, modifica il flag ZF o zero flag del registro EFLAGS che verrà impostato a 1 se la somma di AND è 0, e dato che l'operatore AND fa un operazione con se stesso, il risultato sarà 0 solo se quest'ultimo è 0.

Mentre l'altra istruzione **jz short loc_401032** fa un salto condizionale.

Questo farà un salto di memoria solo se soddisferà una certa condizione, in questo caso l'operatore jz esegue il salto se il flag ZF del registro EFLAGS è uguale a 1.

Quindi quando il valore di `eax` è uguale a 0 il salto viene eseguito all'indirizzo di memoria 00401032 ed esegue l'istruzione “`mov ecx, [ebp+cbData]`”, che sposterà il contenuto della memoria all'indirizzo: `[ebp+cbData]` nel registro `ecx`. Mentre quando il valore sarà diverso da 0 eseguirà la prossima istruzione `mov eax, 1`, e imposta il valore del registro `eax` ad 1.

```

.text:00401029      jz      short loc_401032
.text:0040102B      mov     eax, 1
.text:00401030      jmp     short loc_40107B
;-----
.text:00401032      loc_401032:  mov     ecx, [ebp+cbData] ; CODE XREF: sub_401000+29↑j
.text:00401032

```

Uguale a 0

```

.text:00401027      test    eax, eax
.text:00401029      jz      short loc_401032
.text:0040102B      mov     eax, 1

```

Diverso da 0

Vediamo poi che alla locazione di memoria 00101047 `RegSetValueExA`. questa funzione permette di dettare i dati e anche il tipo di un valore specificato in una delle chiavi del registro di sistema.

Locazione di memoria 00401047

```

.text:00401032      mov     ecx, [ebp+cbData]
.text:00401035      push    ecx ; cbData
.text:00401036      mov     edx, [ebp+lpData]
.text:00401039      push    edx ; lpData
.text:0040103A      push    1 ; dwType
.text:0040103C      push    0 ; Reserved
.text:0040103E      push    offset ValueName ; "GinaDLL"
.text:00401043      mov     eax, [ebp+hObject]
.text:00401046      push    eax ; hKey
.text:00401047      call    ds:RegSetValueExA

```

Vediamo qui sotto che la funzione sta cercando di creare o modificare il nome del valore da impostare. Lo capiamo dal parametro `lpValueName`, e il valore assegnato è “GinaDLL”.

```

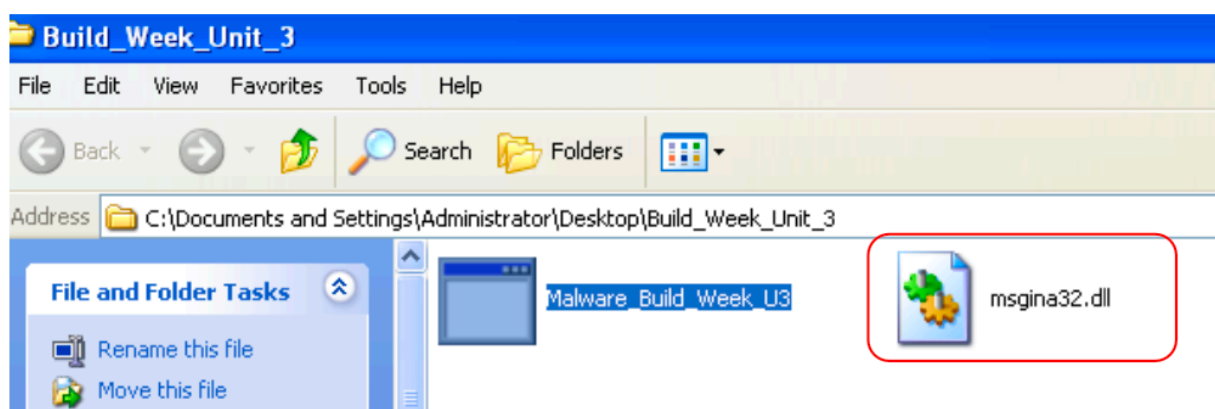
LSTATUS RegSetValueExA(
    [in] HKEY hKey,
    [in, optional] LPCSTR lpValueName,
    DWORD Reserved,
    [in] DWORD dwType,
    [in] const BYTE *lpData,
    [in] DWORD cbData
);

```

Analisi dinamica

Per l'analisi dinamica useremo il tool di microsoft Process Monitor, questo tool monitora e registra le attività di sistema operativo in live. Inoltre il tool ci da informazioni sulle operazione dei processi, inclusi i processi di sistema e le app utente.

Riassumendo la procedura preliminare, ci verrà presentata una schermata con dei valori e parametri da impostare, lasciamo tutto così, selezioni eventuali filtri, li rimuoviamo e avviamo.



Avviando il malware notiamo che alla cartella è stato creato il file **msgina32.dll**.

Nell'analisi statica abbiamo visto come il malware utilizzi le funzioni RegCreateKeyExA e RegSetValueExA, per creare o modificare una chiave del registro di sistema, e come il primo vadi a creare o modificare una sottochiave del registro nella cartella del registro Software\\Microsoft\\Windows\\CurrentVersion\\, ricordiamo che questo registro contiene le informazioni del sistema operativo Windows installato.

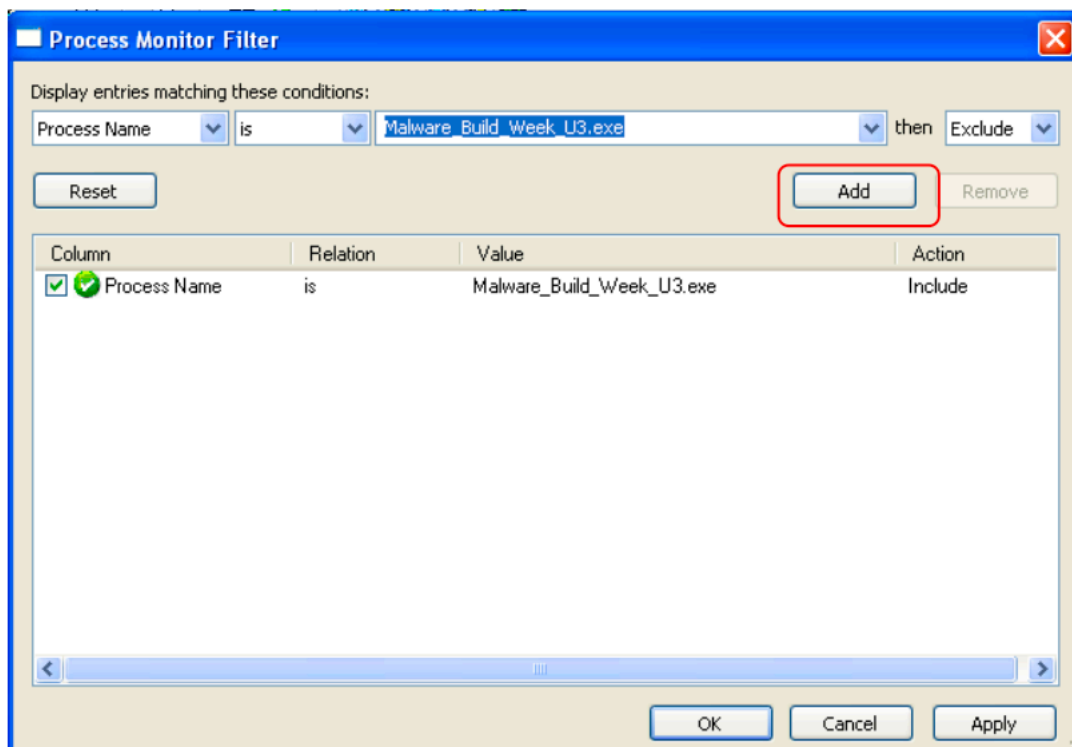
Sappiamo anche che dall'analisi dei parametri è venuto fuori come la funzione RegSetValueExA ci dia il nome del valore della chiave creata o modificata, ovvero GinaDLL.

Mentre per l'analisi del file system abbiamo visto le funzioni utilizzate dal malware: CreateFileA e WriteFile, che vanno a creare un file e a scriverlo.

Riassumendo, sappiamo che il malware probabilmente ha usufruito delle ultime due funzioni sopra citate per creare il file msgina32.dll. questo file sarebbe connesso alla chiave di registro di GinaDLL creata o modificata dal malware e

inoltre potrebbe essere utilizzato per delle azioni malevole sugli accessi, come ottenere privilegi tramite le credenziali degli utenti per l’esecuzione di azioni non consensuali. E infine sappiamo che la chiave del registro di GinaDLL è connessa all’interfaccia di accesso in modo tale da sostituire la procedura di accesso di Windows.

Tornando al Process monitor, impostiamo il filtro che ci darà solo l’analisi delle azioni del malware e avviamo.



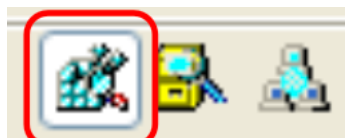
Subito notiamo come i primi risultati ci diano una serie di operazioni specifiche, quelle di mappare i file di sistema e delle cartelle. Queste cartelle contengono operazione come CreateFile, ReadFile, QueryDirectory,

1592	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Malware_Build_Week_U3.exe	NAME NOT FOUND
1592	CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS
1592	FileSystemControl	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS
1592	QueryOpen	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe.Local	NAME NOT FOUND
1592	Load Image	C:\WINDOWS\system32\kernel32.dll	SUCCESS
1592	RegOpenKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
1592	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat	SUCCESS
1592	RegCloseKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
1592	Load Image	C:\WINDOWS\system32\advapi32.dll	SUCCESS
1592	Load Image	C:\WINDOWS\system32\rpcrt4.dll	SUCCESS
1592	Load Image	C:\WINDOWS\system32\secur32.dll	SUCCESS
1592	RegOpenKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
1592	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat	SUCCESS
1592	RegCloseKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
1592	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Secur32.dll	NAME NOT FOUND
1592	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\RPCRT4.dll	NAME NOT FOUND
1592	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ADVAPI32.dll	NAME NOT FOUND
1592	RegOpenKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
1592	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSAppCompat	SUCCESS
1592	RegQueryValue	HKLM\System\CurrentControlSet\Control\Terminal Server\TSUserEnabled	SUCCESS
1592	RegCloseKey	HKLM\System\CurrentControlSet\Control\Terminal Server	SUCCESS
1592	RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon	SUCCESS
1592	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\LeakTrack	NAME NOT FOUND
1592	RegCloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon	SUCCESS

QueryStandardInformationFile, CloseFile. Queste azioni non fanno altro che aprire, leggere, richiedere informazioni e chiudere i file e le directory. Scorrendo, troviamo funzioni come RegOpenKey, RegQueryValue e

1592	CreateFile	C:\WINDOWS\Prefetch\MALWARE_BUILD_WEEK_U3.EXE-0E171D0F.pf	SUCCESS
1592	QueryStandardInformationFile	C:\WINDOWS\Prefetch\MALWARE_BUILD_WEEK_U3.EXE-0E171D0F.pf	SUCCESS
1592	ReadFile	C:\WINDOWS\Prefetch\MALWARE_BUILD_WEEK_U3.EXE-0E171D0F.pf	SUCCESS
1592	CloseFile	C:\WINDOWS\Prefetch\MALWARE_BUILD_WEEK_U3.EXE-0E171D0F.pf	SUCCESS
1592	CreateFile	C:\	SUCCESS
1592	QueryInformationVolume	C:\	SUCCESS
1592	FileSystemControl	C:\	SUCCESS
1592	CreateFile	C:\	SUCCESS
1592	QueryDirectory	C:\	SUCCESS
1592	QueryDirectory	C:\	NO MORE FILES
1592	CloseFile	C:\	SUCCESS
1592	IRP_MJ_CLOSE	C:\	SUCCESS
1592	CreateFile	C:\DOCUMENTS AND SETTINGS	SUCCESS
1592	QueryDirectory	C:\Documents and Settings	SUCCESS
1592	QueryDirectory	C:\Documents and Settings	NO MORE FILES
1592	CloseFile	C:\Documents and Settings	SUCCESS
1592	IRP_MJ_CLOSE	C:\Documents and Settings	SUCCESS
1592	CreateFile	C:\Documents and Settings\ADMINISTRATOR	SUCCESS
1592	QueryDirectory	C:\Documents and Settings\Administrator	SUCCESS
1592	QueryDirectory	C:\Documents and Settings\Administrator	NO MORE FILES
1592	CloseFile	C:\Documents and Settings\Administrator	SUCCESS
1592	IRP_MJ_CLOSE	C:\Documents and Settings\Administrator	SUCCESS
1592	CreateFile	C:\Documents and Settings\Administrator\Desktop	SUCCESS
1592	QueryDirectory	C:\Documents and Settings\Administrator\Desktop	SUCCESS
1592	QueryDirectory	C:\Documents and Settings\Administrator\Desktop	NO MORE FILES
1592	CloseFile	C:\Documents and Settings\Administrator\Desktop	SUCCESS
1592	IRP_MJ_CLOSE	C:\Documents and Settings\Administrator\Desktop	SUCCESS
1592	CreateFile	C:\Documents and Settings\Administrator\Desktop\BUILD_WEEK_UNIT_3	SUCCESS
1592	QueryDirectory	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS
1592	QueryDirectory	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	NO MORE FILES
1592	CloseFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS

RegCloseKey che a sua volta eseguono l'apertura, richiedono informazioni e alla chiusura di cavi di registro di sistema.



Date:	11/19/2023 10:57:57 PM
Thread:	2968
Class:	Registry
Operation:	RegCreateKey
Result:	SUCCESS
Path:	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
Duration:	0.0000120
Desired Access:	All Access

Adesso cercheremo di filtrare le attività del registro lasciando attiva solo questa icona.

Date:	11/19/2023 10:57:57 PM
Thread:	2968
Class:	Registry
Operation:	RegSetValue
Result:	SUCCESS
Path:	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL
Duration:	0.0000084
Type:	REG_SZ
Length:	520
Data:	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll

Ci ricordiamo che il malware effettuava molte operazioni mirate a creare e modificare le chiavi di registro.

Infatti nella lista notiamo queste operazioni RegSetValue e RegSetKey,

forceunlocklogon	REG_DWORD	0x00000000 (0)
GinaDLL	REG_SZ	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll...
HibernationPreviouslyEnabled	REG_DWORD	0x00000001 (1)

analizzando la prima operazione notiamo che il malware ha modificato la chiave



HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

2	Load Image	C:\WINDOWS\system32\secur32.dll
2	CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll
2	CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll
2	WriteFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll
2	WriteFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll
2	WriteFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll
2	CloseFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll

Mentre analizzando la seconda istruzione notiamo che il malware ha sempre modificato la chiave, inserendo un valore nuovo corrispondente al file msgina32.dll e dando come nome al valore “GinaDLL” come visto nell’analisi statica.

La prima operazione serve ad ottenere informazioni per la procedura di accesso degli utenti.

Per confermare che la seconda operazione compia effettivamente ciò, apriamo il registro di sistema con il comando “regedit” e troviamo il suo nuovo valore inserito.

Ora filtriamo le attività lasciando attivo solo questo pulsante.

Ora l’analisi ci dà anche una nuova attività del malware CreateFile nel suo path della sua directory:

C:\DocumentsandSettings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll.

L’operazione è servita per la creazione del file msgina32.dll, che prima dell’avvio del Malware non esisteva. E sotto di esso troviamo anche i file di modifica (scrittura) e chiusura di questo file.

Il malware ha modificato le cartelle dove si trova quest’ultimo.

Conclusioni finali

L'analisi statica e dinamica ci hanno rivelato che: il malware ha creato il file msgina32.dll nel suo percorso: C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\.

Ha creato la chiave di registro denominata GinaDLL nel suo path:
C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll

Nella cartella di registro: HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon.

Questo ci dice che il file msgina32.dll è collegato alla chiave di registro GinaDLL.

Sappiamo che questa chiave è collegata all'interfaccia di accesso Gina, Graphical Identification and Authentication e viene utilizzata per sostituire la procedura di accesso di Windows.

Quindi il file con la sua chiave, se propriamente impostato, potrebbe ottenere un accesso automatico non autorizzato e personalizzato al fine di recuperare credenziali e privilegi degli utenti per scopi malevoli.

Possiamo concludere che le azioni del malware sono: modifica delle impostazioni di configurazione dell'interfaccia di accesso, e dal suo comportamento.

Il Malware è un dropper.