



BUILD WEEK 3

Malware Analysis

Pablo Ballesteros
Simone Cozzolino
Giulia Salani
Manuel Perelli
Andrea Pace
Milto Ferro
Gabriele Giubilo



INDICE

● Introduzione	
○ Preparazione ambiente	2
○ Tool utilizzati	6
● Giorno 1	
○ Parametri e variabili di Main	11
○ Sezioni del Malware	13
○ Librerie importate dal Malware	23
● Giorno 2	
○ Spiegazione di istruzioni Assembly	27
○ Conclusioni del giorno	29
● Giorno 3	
○ Valore del parametro ResourceName.....	31
○ Identificazione di funzionalità implementate	33
○ Identificazione di funzionalità tramite analisi statica basica	35
○ Flusso della funzione	37
● Giorno 4	
○ Analisi dinamica basica con Procmon	39
○ Analisi del registro con Regshot	40
○ Verifica delle modifiche su Regedit	42
● Giorno 5	
○ Cosa succede se il .dll lecito viene sostituito con il .dll malevolo	44
○ Diagramma di flusso del funzionamento del malware	45
● Osservazioni extra e conclusioni	
○ Dove si salvano le credenziali rubate ?	47
○ Incident response	48
○ Conclusioni	49

INTRODUZIONE

L'obiettivo di questo documento è la comprensione attraverso l'analisi basica e avanzata di un file eseguibile che si sospetta sia un malware. Il file da analizzare ha il nome **Malware_Build_Week_U3**. Verranno utilizzati vari metodi e strumenti per esaminarlo e garantire la migliore comprensione possibile.

Esistono 4 modi per eseguire una scansione malware:

- Statica basica
- Dinamica basica
- Statica avanzata
- Dinamica avanzata

Con strumenti come **CFFexplorer**, **IDA pro**, **OllyDbg** analizzeremo il contenuto ASCII, le stringhe e il codice Assembly del file stesso.

Mentre con strumenti come **Process Monitor** e **Regshot** andremo a vedere il comportamento del malware nel sistema una volta eseguito, come la creazione di processi e la modifica delle chiavi di registro.

Una volta analizzato il file con tutti gli strumenti disponibili, possiamo unire i punti e le prove trovate per trarre conclusioni sul funzionamento e sullo scopo del malware.

Preparazione ambiente

Nel corso della Build Week 3 utilizzeremo un ambiente (la macchina virtuale) che contiene veri e propri malware, dunque la corretta preparazione di un ambiente sicuro è un passaggio fondamentale. Occorre innanzitutto prendere una serie di misure per tutelare la macchina ospite.

Apriamo **Oracle VM Virtualbox**, selezioniamo la nostra macchina virtuale e poi “*Settings*” come in figura 0.1.

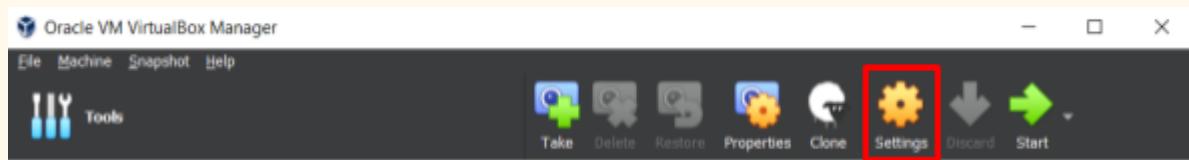


Figura 0.1 Impostazioni virtualbox

1. Network

Il primo punto su cui intervenire sono le interfacce di rete. Occorre isolare la macchina per evitare che il malware possa infettare il sistema ospite tramite la rete. Ci spostiamo sulla tab Network e disabilitiamo tutte le interfacce di rete, così da isolare la macchina virtuale:

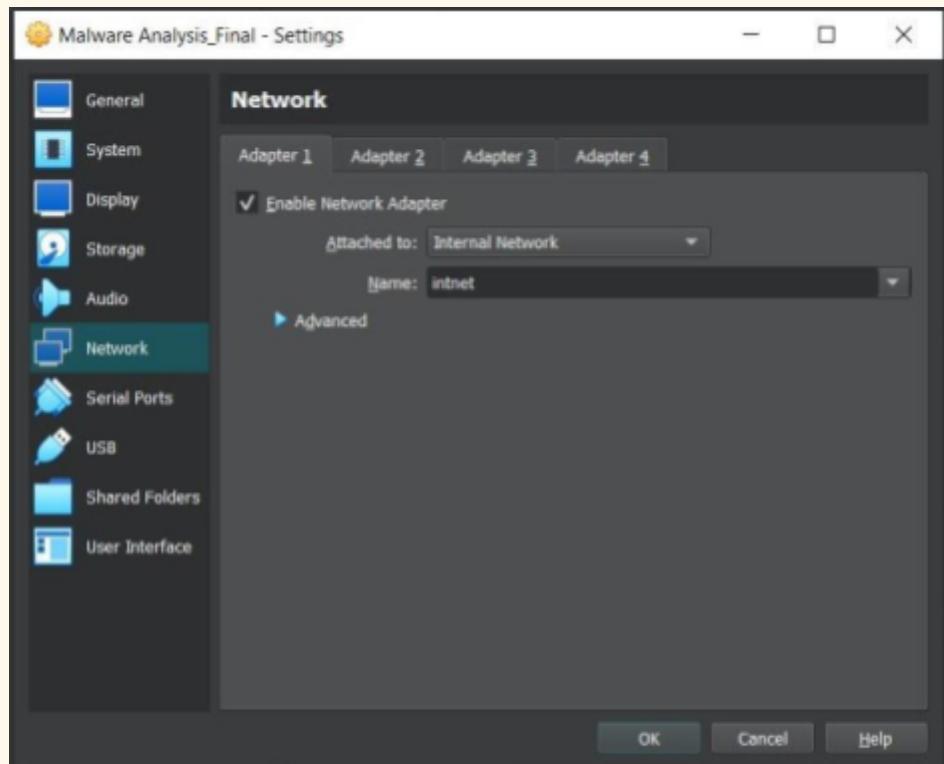


Figura 0.2 Impostazione Network

2. USB Controller

Il secondo punto su cui intervenire è l'USB Controller.

Spostiamoci nella relativa sezione e assicuriamoci che l'opzione sia disabilitata, per evitare che il malware possa utilizzare questa opzione per propagarsi su eventuali chiavi USB inserite nella macchina ospite:

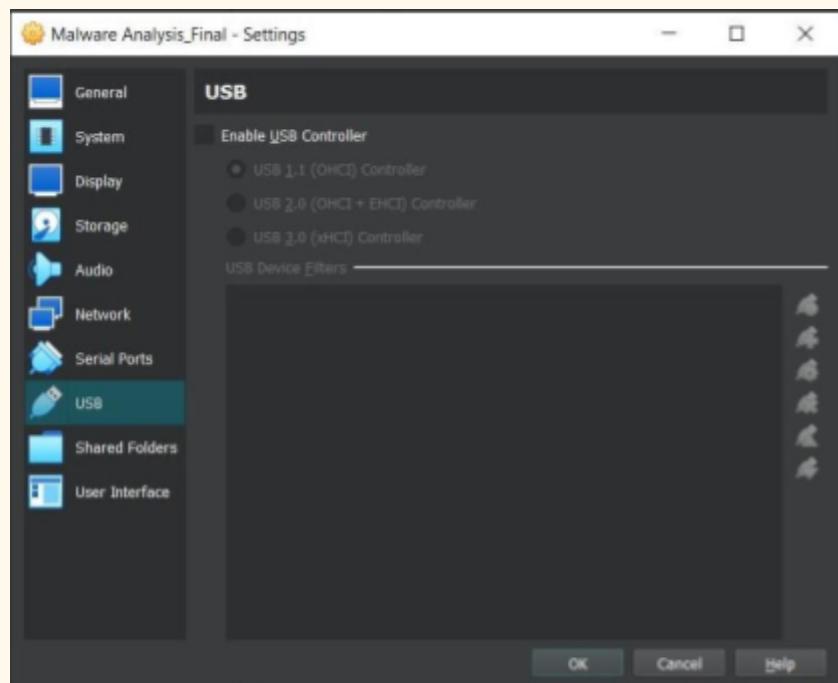


Figura 0.3 Impostazione USB

3. Shared Folders

Il terzo punto di possibile contatto fra la macchina ospite e la macchina virtuale sono le cartelle condivise.

Spostiamoci nella relativa sezione e assicuriamoci che non vi siano cartelle condivise, che il malware potrebbe utilizzare per diffondersi:

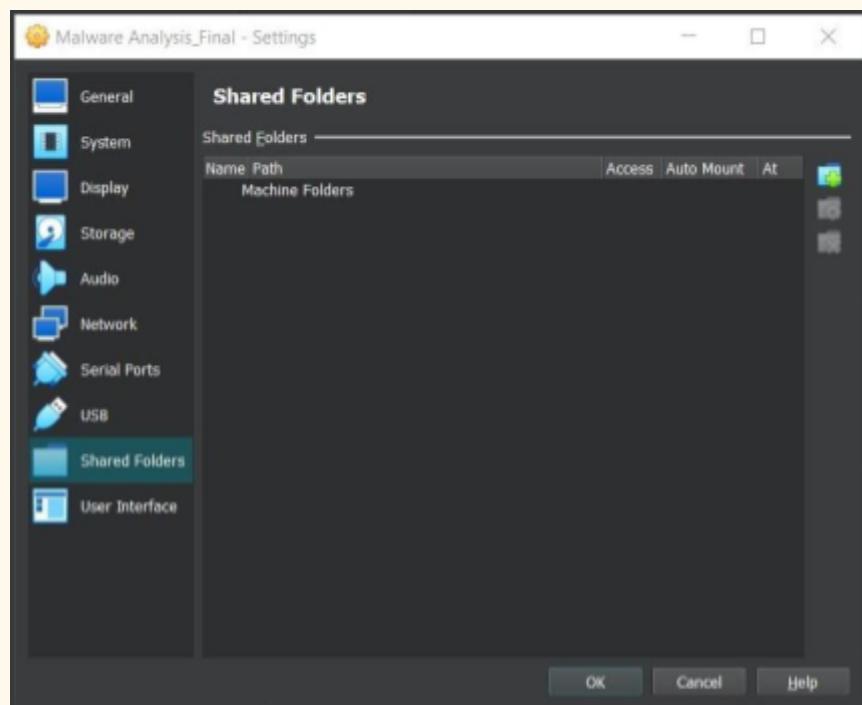


Figura 0.4 Impostazione Folders

Ma non occorre soltanto tutelare la macchina ospite. Eseguendo l'analisi dinamica dei malware sulla macchina virtuale, infatti, potremmo compromettere quel sistema in maniera irreversibile. Per evitare di doverla installare da zero, prima di avviare la macchina virtuale creiamo una instantanea del suo stato attuale.

Nei tool di Oracle VM VirtualBox Manager cerchiamo “*Snapshots*” e lo clicchiamo:

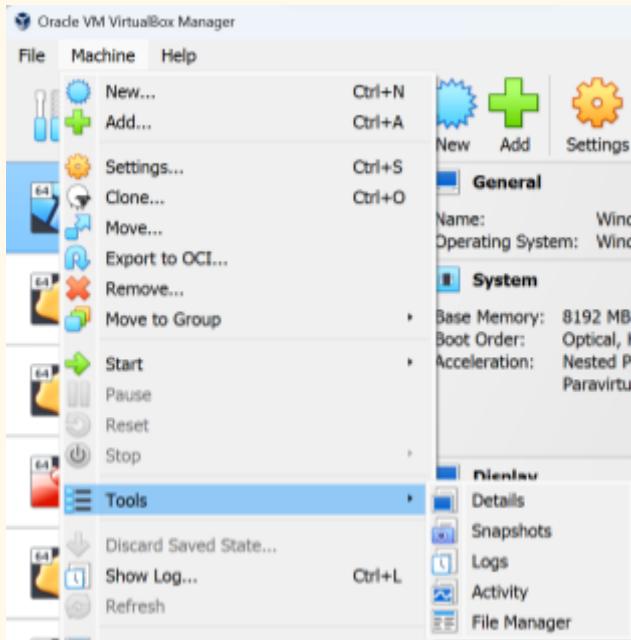


Figura 0.5 Tools tab

A questo punto selezioniamo l'opzione “*Take*” per scattare l'istantanea:

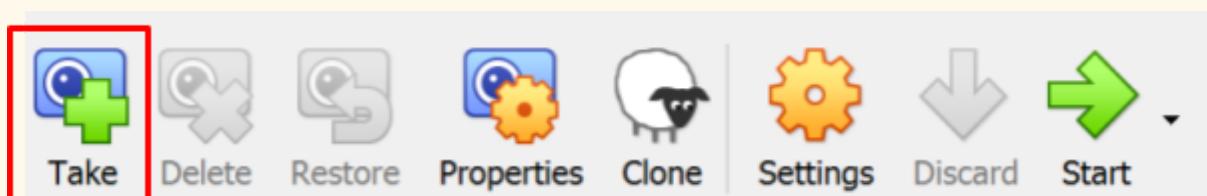


Figura 0.6 Fare snapshot

Oracle VM VirtualBox Manager ci chiede che nome vogliamo dare all'istantanea e possiamo anche inserire una breve descrizione. Compiliamo i campi e clicchiamo su OK:

Al termine delle esercitazioni, una volta spenta la macchina virtuale, è sempre buona norma ripristinare l'istantanea che abbiamo salvato. Lo facciamo con il comando “*Restore*” come si vede in figura 0.8:

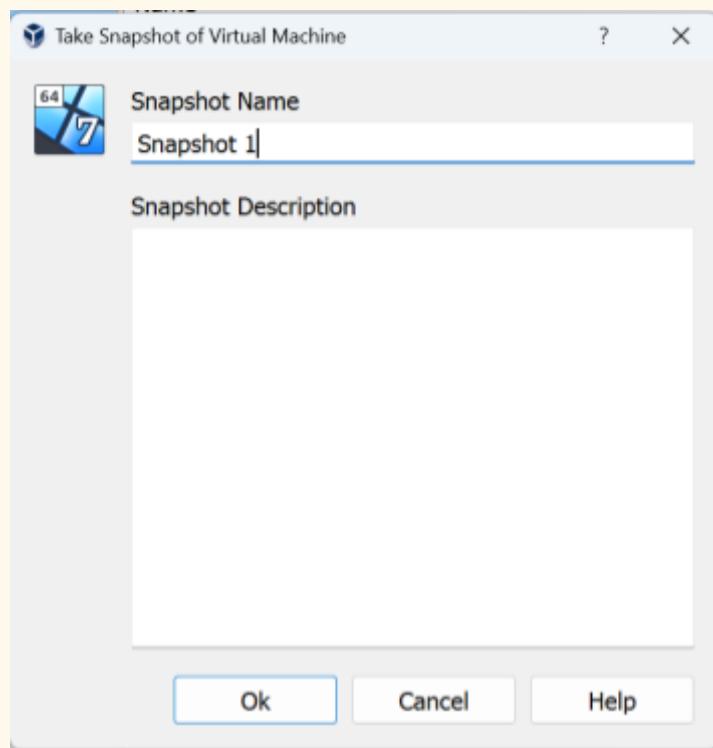


Figura 0.7 Nominazione Snapshot

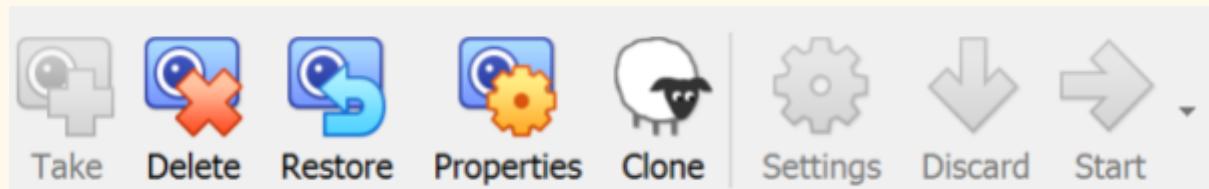


Figura 0.8 Restore Snapshot

Queste semplici e veloci accortezze ci permettono di 1) proteggere la macchina ospite da un eventuale contagio con i malware che di volta in volta analizziamo e 2) salvaguardare la macchina virtuale, risparmiando tempo prezioso.

Tool utilizzati

CFFexplorer:

CFF Explorer è un programma informatico utilizzato principalmente per esaminare e modificare file eseguibili su piattaforma Windows. Questi file, noti come file eseguibili portatili (PE), includono eseguibili come .exe e .dll. CFF sta per "Common File Format" (Formato File Comune).



Il software consente agli sviluppatori e agli esperti di sicurezza informatica di visualizzare informazioni dettagliate sui file PE, come l'header del file, le sezioni, le risorse, le importazioni, le esportazioni e molto altro. Queste informazioni sono fondamentali per comprendere come funzionano i programmi e possono essere utili per l'analisi della sicurezza e lo sviluppo del software.

Inoltre, CFF Explorer fornisce strumenti per eseguire alcune operazioni avanzate sui file PE, come l'estrazione e l'iniezione di risorse, la modifica delle intestazioni del file e la visualizzazione e modifica delle tabelle di esportazione e importazione.

In generale, è uno strumento potente per esaminare e manipolare file eseguibili su piattaforma Windows.

IDA Pro:

IDA Pro (Interactive DisAssembler Professional) è un software di disassemblaggio e analisi del codice binario. È ampiamente utilizzato nell'analisi di malware, nel reverse engineering e nella comprensione del codice eseguibile.



Disassembler:

IDA Pro traduce il codice binario di un eseguibile in un formato leggibile, mostrando il codice assembly corrispondente.

Analisi statica:

Permette di esaminare il codice sorgente assembly, identificare funzioni, variabili e flussi di controllo per comprendere la struttura del programma.

Graph View:

Fornisce una visualizzazione grafica del flusso di controllo del programma, semplificando la comprensione della logica di esecuzione.

IDA Pro è ampiamente utilizzato nel campo della sicurezza informatica e dell'analisi malware. Gli analisti possono utilizzare questo strumento per comprendere il comportamento di malware, identificare vulnerabilità di sicurezza e analizzare il funzionamento interno di applicazioni.

OllieDbG:

OllieDBG è un debugger a livello di codice macchina ampiamente utilizzato nell'analisi del software e nella ricerca sulla sicurezza informatica. Le sue principali funzioni includono:



-Tracciamento dell'esecuzione del codice: OllieDBG consente agli utenti di esaminare il flusso di esecuzione del codice macchina di un programma. Questo include il tracciamento delle istruzioni eseguite, l'analisi del contenuto della memoria e la gestione dei registri della CPU.

-Analisi dei punti di interruzione: Gli utenti possono impostare punti di interruzione in determinate istruzioni o aree di memoria per controllare l'esecuzione del programma. Ciò consente di interrompere l'esecuzione del programma in un punto specifico per esaminare lo stato del sistema in quel momento.

-Esplorazione del codice assembly: OllieDBG fornisce un'interfaccia per esaminare il codice assembly del programma in esecuzione. Gli utenti possono visualizzare le istruzioni assembly che corrispondono al codice sorgente e analizzare il comportamento del programma a livello di istruzioni di basso livello.

-Analisi delle chiamate di funzione: Il debugger consente agli utenti di tracciare le chiamate di funzione all'interno del programma e di esaminare i parametri passati alle funzioni, nonché i valori restituiti.

ProcessMonitor (Procmon):

Procmon, o Process Monitor, è uno strumento software utilizzato principalmente su sistemi Windows per monitorare e registrare le attività del sistema in tempo reale. Possiamo paragonarlo ad un registro dettagliato di tutte le azioni che i programmi eseguono sul computer.



Queste azioni includono l'apertura di file, la scrittura nel registro di sistema, l'avvio dei processi e molte altre attività. Con Procmon, puoi vedere esattamente cosa sta succedendo sul tuo computer in un dato momento e individuare eventuali problemi o comportamenti sospetti dei programmi.

Regshot:

REGSHOT è uno strumento utilizzato per confrontare le modifiche nel registro di sistema di un computer prima e dopo l'installazione di un'applicazione o l'esecuzione di determinate azioni. In altre parole, regista e confronta lo stato del registro di sistema prima e dopo un'operazione.



Le sue funzionalità principali includono:

-Creazione di snapshot del registro di sistema: REGSHOT crea uno snapshot del registro di sistema in un determinato momento. Questo snapshot rappresenta lo stato corrente del registro, inclusi tutti i valori, le chiavi e le sottochiavi presenti.

-Esecuzione di un'azione: Dopo la creazione del primo snapshot, l'utente esegue un'azione desiderata, come l'installazione di un'applicazione o la modifica delle impostazioni di sistema.

-Creazione di un secondo snapshot: Una volta completata l'azione, REGSHOT crea un secondo snapshot del registro di sistema.

-Confronto dei due snapshot: REGSHOT confronta i due snapshot e identifica le differenze tra di essi. Le differenze possono includere l'aggiunta, la modifica o la rimozione di valori, chiavi o sottochiavi nel registro di sistema.

È ampiamente utilizzato nell'analisi dei problemi di configurazione, dell'installazione del software e nell'analisi dei malware.

GIORNO 1

Parametri e variabili della funzione Main

Il Malware analizzato contiene un lungo codice Assembly, dove ogni funzione è spiegata in modo granulare riga per riga, inclusa una delle più importanti: La funzione Main. In questa funzione viene chiamata un'altra coppia di funzioni, che necessitano di **variabili e parametri**.

- **Variabili:** Le variabili nel linguaggio Assembly possono rappresentare vari tipi di dati, come numeri interi, caratteri, matrici e strutture. Vengono utilizzate per contenere valori utilizzati nei calcoli, nei confronti e in altre operazioni all'interno del programma. In uno stack le funzioni sono contrassegnate con un offset negativo rispetto all'EBP (Extended Base Pointer). A differenza dei linguaggi di programmazione di alto livello, in Assembly ogni variabile deve dichiarare una dimensione, **Byte** per 8 bit, **Word** per 16 bit e **Double Word (Dword)** per 32 bit.
- **Parametri:** Nel linguaggio Assembly, i parametri sono valori che vengono passati a una funzione o subroutine. Consentono il trasferimento dei dati tra diverse parti di un programma. I parametri possono essere utilizzati per fornire input ad una funzione o per ricevere output da una funzione. In uno stack i parametri sono indicati con un offset positivo rispetto all'EBP.

Giorno 1

In figura 1.1 è possibile vedere il codice Assembly della **funzione Main**, dove all'inizio sono opportunamente indicate le funzioni, le variabili ed i parametri che verranno utilizzati, con i rispettivi offset positivi e negativi.



The screenshot shows the assembly code for the `main` function in a debugger. The code is color-coded to highlight different components: blue for labels and function names, green for registers and memory addresses, and pink for comments. The assembly instructions are in Intel syntax. The code starts with a bp-based frame setup, followed by variable declarations, and then the main logic involving `GetModuleHandleA` and `GetModuleFileNameA` calls.

```
; 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
mov     ebp, esp
sub    esp, 11Ch
push    ebx
push    esi
push    edi
mov     [ebp+var_4], 0
push    0          ; lpModuleName
call    ds:GetModuleHandleA
mov     [ebp+hModule], eax
mov     [ebp+Data], 0
mov     ecx, 43h
xor     eax, eax
lea     edi, [ebp-117h]
rep stosd
stosb
mov     eax, [ebp+hModule]
push    eax          ; hModule
call    sub_401080
add     esp, 4
mov     [ebp+var_4], eax
push    10Eh        ; nSize
lea     ecx, [ebp+Data]
push    ecx          ; lpFilename
push    0            ; hModule
call    ds:GetModuleFileNameA
push    5Ch          ; int
lea     edx, [ebp+Data]
```

Figura 1.1 Codice Assembly Main

```
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
```

Figura 1.2 Variabili e parametri Main

Nella figura 1.2 possiamo vedere le variabili ed i parametri più in dettaglio. Le prime 4 righe sono le variabili (delimitate dal rettangolo rosso), come indicato dal loro offset negativo, una delle quali è la variabile **hModule**, che funge da puntatore ai moduli e può accedere alle loro risorse e funzioni.

Le successive 3 linee (delimitate dal rettangolo blu) indicano i parametri che verranno utilizzati, come indicato dal loro offset positivo.

Sezioni del Malware

Per verificare di quante sezioni si compone il malware, ci spostiamo sul tool **CFF Explorer**. Apriamo il programma e carichiamo il malware cliccando l'icona a forma di cartella in alto a sinistra. Dopodiché, selezioniamo **Malware_Build_Week_U3**:

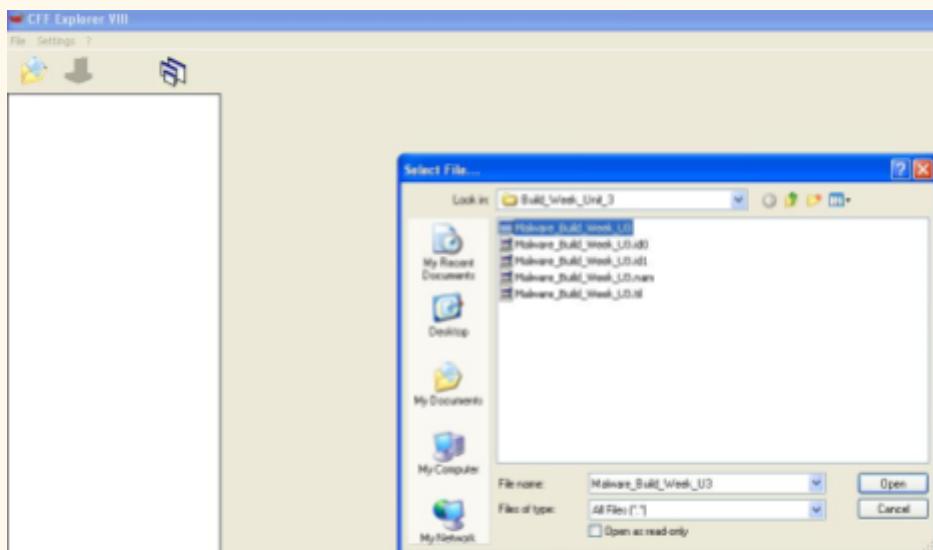


Figura 1.3 Aprire File in CFF

Dall'elenco a sinistra, selezioniamo la riga *Section Headers* e notiamo che le sezioni sono quattro, come indicato in figura 1.4 : **.text**, **.rdata**, **.data**, **.rsrc**.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber ...	Characteristics
00000250	0000025C	00000260	00000264	00000268	0000026C	00000270	00000272	00000274	
Byte[8]	Dword	Dword	Dword	Dword	Dword	Word	Word	Word	Dword
.text	00005646	00001000	00005600	00001000	00000000	00000000	0000	0000	60000020
.rdata	000009AE	00007000	00001000	00007000	00000000	00000000	0000	0000	40000040
.data	00003EA8	00008000	00003000	00008000	00000000	00000000	0000	0000	C0000040
.rsrc	00001A70	0000C000	00002000	00008000	00000000	00000000	0000	0000	40000040

Figura 1.4 Sezioni in CFF

Prima di aprirle, partendo dai loro nomi ripercorriamo a livello teorico che cosa potrebbe contenere ciascuna di esse:

- **.text**: Contiene il codice eseguibile del programma, ovvero le istruzioni di Assembly che vengono eseguite durante l'esecuzione del programma. Qui si trova il cuore del codice malevolo, inclusi i segmenti di codice responsabili delle azioni specifiche del malware, come la replicazione, l'infezione di file o la comunicazione con server remoti.
- **.rdata**: Contiene dati di sola lettura, spesso stringhe costanti, tabelle di lookup o altre costanti necessarie all'esecuzione del programma. Questa sezione potrebbe contenere stringhe di testo utilizzate nel codice, indirizzi IP, chiavi di crittografia o qualsiasi altra costante necessaria per il funzionamento del malware. Queste informazioni possono essere cifrate o oscurate per rendere più difficile l'analisi.
- **.data**: Contiene dati globali inizializzati prima dell'esecuzione del programma, come variabili globali o dati inizializzati a runtime. Questa sezione potrebbe includere variabili globali utilizzate dal malware per tenere traccia dello stato, memorizzare configurazioni, o scambiare informazioni tra funzioni.
- **.rsrc**: Contiene risorse come icone, immagini, stringhe localizzate e altri dati non eseguibili utilizzati dall'applicazione. Alcuni malware utilizzano questa sezione per nascondere dati o codice malevolo, approfittando del fatto che le risorse sono spesso trascurate durante

Giorno 1

l'analisi. Ad esempio, possono cifrare il loro payload e nasconderlo tra le risorse.

Anche all'interno di IDA Pro, nella subview “Segmentation” possiamo esplorare le sezioni. Per aprire questa tab ci basterà cliccare “View”, “Open subviews” e “Segments” come descritto in figura 1.5:

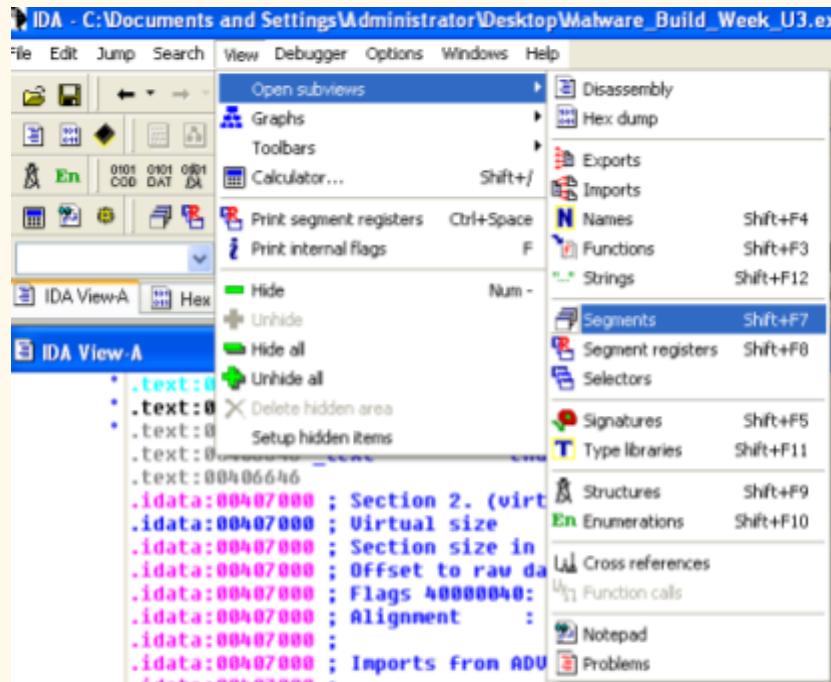


Figura 1.5 Aprire sezioni in IDA

SEZIONE .TEXT:

Grazie a CFF Explorer, notiamo che la sezione .txt è la più pesante. In IDA Pro, dalla sezione *Segmentation*, con un doppio clic sul nome di sezione veniamo reindirizzati alla tab IDA View, dove possiamo analizzare il contenuto della sezione. In figura 1.6, per ragioni di spazio presentiamo soltanto l'inizio della sezione, che poi prosegue:

The screenshot shows the IDA Pro interface. On the left, there is a large assembly code window with several lines of assembly code highlighted in yellow. On the right, there is a smaller window titled "Graph overview" which displays a call graph of the program's control flow.

```

hObject= dword ptr -4
lpData= dword ptr 8
cbData= dword ptr 0Ch

push    ebp
mov    ebp, esp
push    ecx
push    0           ; lpdwDisposition
lea    eax, [ebp+hObject]
push    eax          ; phkResult
push    0           ; lpSecurityAttributes
push    0@03Fh        ; samDesired
push    0           ; dwOptions
push    0           ; lpClass
push    0           ; Reserved
push    offset SubKey ; "SOFTWARE\Microsoft\Windows NT\CurrentVer...
push    0@0000002h    ; hKey
call    ds:RegCreateKeyExA
test   eax, eax
jz     short loc_401032

```

Figura 1.6 Prima sezione del malware: .text

Come ci aspettavamo, la sezione .text contiene il codice eseguibile del programma. Scorrendo le istruzioni in Assembly, notiamo il caricamento di variabili e parametri (istruzione “*push*”), le chiamate di funzione (istruzione “*call*”) e diversi salti condizionali (istruzione “*jz*”) o non condizionali (istruzione “*jmp*”).

SEZIONE .RDATA:

La seconda sezione contiene i dati di sola lettura e da CFF Explorer possiamo notare come sia la sezione meno pesante del malware. Andiamo ad approfondirla con **IDA Pro** e, scorrendo le sue righe, notiamo che nella prima parte della sezione le righe iniziano con la dicitura **.idata**, mentre da un certo punto in avanti le righe iniziano con la dicitura **.rdata**. Vediamo la differenza:

- .idata (Import Directory Section):** Questa sezione contiene informazioni sull'importazione di funzioni da librerie dinamiche (DLL) nell'eseguibile. In sostanza, qui vengono elencate le funzioni o simboli che il programma utilizza da librerie esterne.
- .rdata (Read-only Data Section):** Questa sezione contiene dati che sono dichiarati come costanti e che non possono essere modificati durante l'esecuzione del programma. Può includere costanti stringa, tabelle e altri dati che il programma può leggere ma non scrivere.

Giorno 1

La parte .idata si presenta così:

```
.text:00400640 _text ends  
.idata:00407000 ; Section 2. (virtual address 00007000)  
.idata:00407000 ; Virtual size : 000009AE ( 2478.)  
.idata:00407000 ; Section size in file : 00001000 ( 4096.)  
.idata:00407000 ; Offset to raw data for section: 00007000  
.idata:00407000 ; Flags 40000040: Data Readable  
.idata:00407000 ; Alignment : default  
.idata:00407000 ;  
.idata:00407000 ; Imports From ADVAPI32.dll  
.idata:00407000 ;  
.idata:00407000 ;  
.idata:00407000 ;  
.idata:00407000 ; Segment type: Externs  
.idata:00407000 ; _idata  
* .idata:00407000 : LONG __stdcall RegSetValueExA(HKEY hKey,LPCSTR lpValueName,DWORD Reserved,DWORD dwType,const BYTE *lpData  
extra RegSetValueExA:dword ; DATA XREF: sub_401000+67Tr  
* .idata:00407000 : LONG __stdcall RegCreateKeyExA(HKEY hKey,LPCSTR lpSubKey,DWORD Reserved,LPSTR lpClass,DWORD dwOptions,RE  
extra RegCreateKeyExA:dword ; DATA XREF: sub_401000+21Tr  
* .idata:00407000 ;  
.idata:00407000 ;  
.idata:00407000 ; Imports From KERNEL32.dll  
.idata:00407000 ;  
.idata:00407000 ;  
.idata:00407000 ; DWORD __stdcall SizeofResource(HMODULE hModule,HRSRC hResInfo)  
extra SizeofResource:dword ; DATA XREF: sub_401000+98Tr  
* .idata:00407010 : LPVOID __stdcall LockResource(HGLOBAL hResData)  
extra LockResource:dword ; DATA XREF: sub_401000+7FTr  
* .idata:00407010 : HGLOBAL __stdcall LoadResource(HMODULE hModule,HRSRC hResInfo)  
extra LoadResource:dword ; DATA XREF: sub_401000+67Tr  
* .idata:00407010 : LPVOID __stdcall VirtualAlloc(LPHANDLE lpAddress,DWORD dwSize,DWORD dwType,DWORD dwProtect)  
extra VirtualAlloc:dword ; DATA XREF: sub_401000+98Tr  
.idata:00407010 ;  
.idata:00407010 ; DWORD __stdcall GetModuleFileNameA(HMODULE hModule,LPSTR lpfilename,DWORD nSize)  
extra GetModuleFileNameA:dword ; DATA XREF: sub_401000+50Tr  
.idata:00407010 ;  
extra GetModuleFileNameA:dword ; DATA XREF: sub_401000+50Tr  
; __setargs+23Tr ...
```

Figura 1.8 Seconda sezione del malware

Le prime righe sono dedicate ai metadati che avevamo già potuto osservare in CFF Explorer, come per esempio il numero di sezione e le dimensioni:

```
.text:00400640  
.idata:00407000 ; Section 2. (virtual address 00007000)  
.idata:00407000 ; Virtual size : 000009AE ( 2478.)  
.idata:00407000 ; Section size in file : 00001000 ( 4096.)  
.idata:00407000 ; Offset to raw data for section: 00007000  
.idata:00407000 ; Flags 40000040: Data Readable  
.idata:00407000 ; Alignment : default  
.idata:00407000 ;
```

Figura 1.9 Intestazione della sezione .rdata

Subito sotto i metadati, entriamo nel vivo della sezione e notiamo una serie di *Import*, prima da ADVAPI32.dll e successivamente da KERNEL32.dll, due librerie:

```
.idata:00407000 ;  
.idata:00407000 ; Imports from ADVAPI32.dll  
.idata:00407000 ;  
.idata:00407000 ;  
.idata:00407000 ;  
.idata:00407000 ; Segment type: Externs  
.idata:00407000 ; _idata  
* .idata:00407000 : LONG __stdcall RegSetValueExA(HKEY hKey,LPCSTR lpValueName,DWORD Reserved,DWORD dwType,const BYTE *lpData  
extra RegSetValueExA:dword ; DATA XREF: sub_401000+67Tr  
* .idata:00407000 : LONG __stdcall RegCreateKeyExA(HKEY hKey,LPCSTR lpSubKey,DWORD Reserved,LPSTR lpClass,DWORD dwOptions,RE  
extra RegCreateKeyExA:dword ; DATA XREF: sub_401000+21Tr  
* .idata:00407000 ;  
.idata:00407000 ;  
.idata:00407000 ; Imports from KERNEL32.dll  
.idata:00407000 ;  
.idata:00407000 ;  
.idata:00407000 ; DWORD __stdcall SizeofResource(HMODULE hModule,HRSRC hResInfo)  
extra SizeofResource:dword ; DATA XREF: sub_401000+98Tr  
* .idata:00407010 : LPVOID __stdcall LockResource(HGLOBAL hResData)  
extra LockResource:dword ; DATA XREF: sub_401000+7FTr  
* .idata:00407010 : HGLOBAL __stdcall LoadResource(HMODULE hModule,HRSRC hResInfo)  
extra LoadResource:dword ; DATA XREF: sub_401000+67Tr  
.idata:00407010 ;  
.idata:00407010 ;
```

Figura 1.10 Import dalle funzioni

Giorno 1

Scorrendo la sezione, possiamo riconoscere in rosa le diverse funzioni che vengono utilizzate dal malware (solo a titolo di esempio **RegSetValueExA** e **RegCreateKeyExA**, da cui possiamo già iniziare a dedurre che il malware ha l'obiettivo di intervenire sui registri di Windows).

In blu invece notiamo sia i tipi di dato (come **DWORD**, che rappresenta un numero intero a 32 bit senza segno, o **LPVOID**, che rappresenta un puntatore a un tipo di dato non specificato, o ancora **HGLOBAL**, che rappresenta un handle), sia le convenzioni di chiamata, ovvero **__stdcall**. **__stdcall** specifica sia come vengono passati i parametri delle funzioni allo stack, sia come viene gestita la pulizia dello stack dopo una chiamata di funzione.

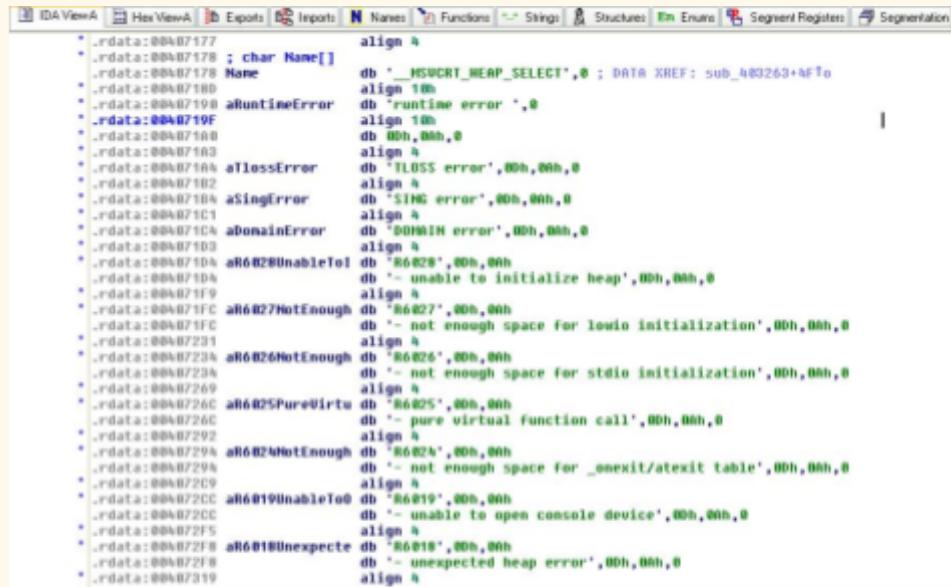
Scendendo incontriamo la parte di **.rdata**, che anche dal punto di vista della formattazione si presenta diversa:

```
IDAViewA HexViewA Exporta Info N Names Funzioni Strings Structure Enums Segment Registers Segmentation
.rdata:004070DC ; Segment type: Pure data
.rdata:004070DC ; Segment permissions: Read
.rdata:004070DC _rdata segment para public 'DATA' use32
assume cs:_rdata
.org 4070DCh
align 10h
.rdata:004070DC unk_4070E8 db 0Fh ; DATA XREF: start+50
.rdata:004070E0 db 0Fh
.rdata:004070E1 db 0Fh
.rdata:004070E2 db 0Fh
.rdata:004070E3 db 0Fh
.rdata:004070E4 db 47h ; G
.rdata:004070E5 db 15h
.rdata:004070E6 db 40h ; @
.rdata:004070E7 db 0
.rdata:004070E8 db 50h ; [
.rdata:004070E9 db 15h ; ]
.rdata:004070EA db 40h ; @
.rdata:004070EB db 0
.rdata:004070EC byte_4070EC db 6 ; DATA XREF: sub_401079:loc_4010DETr
.rdata:004070ED db 0
.rdata:004070EE db 0
.rdata:004070EF db 6
.rdata:004070F0 db 0
.rdata:004070F1 db 1
.rdata:004070F2 db 0
.rdata:004070F3 db 0
.rdata:004070F4 db 10h
.rdata:004070F5 db 0
.rdata:004070F6 db 3
.rdata:004070F7 db 6
.rdata:004070F8 db 0
.rdata:004070F9 db 6
.rdata:004070FA db 2
.rdata:004070FB db 10h
.rdata:004070FC db 4
```

Figura 1.11 Le righe **.rdata** nella seconda sezione

Giorno 1

Continuando a scorrere la sezione, incontriamo diverse stringhe:



The screenshot shows the IDA View window with the 'Strings' tab selected. The second section contains various error messages and initialization-related strings. Some examples include:

- .rdata:004B7177 ; char Name[] align 4 db '_MSVCRT_HEAP_SELECT',0 ; DATA XREF: sub_483263+8To
- .rdata:004B7178 Name align 10h db 'runtime error ',0 align 10h db 0Bh,0Bh,0
- .rdata:004B719F .rdata:004B71A0 align 4 db 'LOSS error',0Bh,0Bh,0
- .rdata:004B71A0 align 4 db 'SING error',0Bh,0Bh,0
- .rdata:004B71C1 align 4 db 'DOMAIN error',0Bh,0Bh,0
- .rdata:004B71D3 align 4 db 'R6028UnableToI' db 'R6028',0Bh,0Bh
- .rdata:004B71D4 align 4 db '- unable to initialize heap',0Bh,0Bh,0
- .rdata:004B71F9 align 4 db 'R6027',0Bh,0Bh
- .rdata:004B71FC align 4 db '- not enough space for Lewis initialization',0Bh,0Bh,0
- .rdata:004B7231 align 4 db 'R6026',0Bh,0Bh
- .rdata:004B7234 align 4 db '- not enough space for stdio initialization',0Bh,0Bh,0
- .rdata:004B7234 align 4 db 'R6025',0Bh,0Bh
- .rdata:004B726C align 4 db '- pure virtual function call',0Bh,0Bh,0
- .rdata:004B7292 align 4 db 'R6024',0Bh,0Bh
- .rdata:004B7294 align 4 db '- not enough space for _onexit/atexit table',0Bh,0Bh,0
- .rdata:004B7299 align 4 db 'R6019UnableToO' db 'R6019',0Bh,0Bh
- .rdata:004B72CE align 4 db '- unable to open console device',0Bh,0Bh,0
- .rdata:004B72F5 align 4 db 'R6018',0Bh,0Bh
- .rdata:004B72F8 align 4 db '- unexpected heap error',0Bh,0Bh,0
- .rdata:004B7319 align 4

Figura 1.12 Stringhe di testo nella seconda sezione

Ad un certo punto si fa riferimento ad una **MessageBoxA**, cosa che suggerisce che il malware potrebbe mostrare all'utente una finestra di dialogo con un messaggio ingannevole:



The screenshot shows the IDA View window with the 'Strings' tab selected. The third section contains the string for the **MessageBoxA** function. The string is:

```
a____ db '...',0 ; DATA XREF: sub_483611+8To
aProgramNameLink db '(program name unknown)',0 ; DATA XREF: sub_483611+20To
aGetLastActiveP[] align 10h
aGetLastActiveP db 'GetLast&ActivePopup',0 ; DATA XREF: ___crtMessageBox+30To
aGetLastActiveW[] align 4
aGetActiveWindow db 'GetActiveWindow',0 ; DATA XREF: ___crtMessageBox+35To
aProcName[] align 4
ProcName db 'MessageBoxA',0 ; DATA XREF: ___crtMessageBox+25To
LibFileName[] align 4
LibFileName db 'user32.dll',0 ; DATA XREF: ___crtMessageBox+8To
byte_407ABC align 4
byte_407ABC db A dup(0) ; DATA XREF: ___crtLCMapString+57To
byte_407ABC ; DATA XREF: ___crtGetStringType+52To
```

Figura 1.13 La stringa MessageBoxA

SEZIONE .DATA:

Segue la terza sezione, che contiene dati globali inizializzati prima dell'esecuzione del programma e si presenta come in figure 1.14 e 1.15:

Giorno 1

```

.IDA ViewA HexViewA Exports Inputs Names Functions Strings Structures Enums Segmentation
.00000000 ; Section 3, virtual address 00000000
.00000000 ; Virtual size : 00003EAB ( 16048L)
.00000000 ; Section size in file : 00003000 ( 12288L)
.00000000 ; Offset to raw data for section: 00000000
.00000000 ; Flags C0000000: Data Readable Writable
.00000000 ; Alignment : default
.00000000 ;
.00000000 ;
.00000000 ; Segment type: Pure data
.00000000 ; Segment permissions: Read/Write
.00000000 _data segment para public 'DATA' use32
assume cs:_data
.00000000 org 400000h
.00000000 unk_400000 db 0 ; DATA XREF: __cinit+tFts
.00000001 db 0
.00000002 db 0
.00000003 db 0
.00000004 unk_400004 db 0 ; DATA XREF: __cinit+tata
.00000005 db 0
.00000006 db 0
.00000007 db 0
.00000008 unk_400008 db 0 ; DATA XREF: __cinit+tata
.00000009 db 0
.0000000a db 0
.0000000b db 0
.0000000c db 0 ; a
.0000000d db 1Ch
.0000000e db 40h ; @
.0000000f db 0
.00000010 db 6Ah ; d
.00000011 db 5Ah ; T
.00000012 db 40h ; @
.00000013 db 0
.00000014 unk_400014 db 0 ; DATA XREF: __cinit+loc_402900E0
.00000015 db 0

```

Figura 1.14 Terza sezione del malware, .data

```

.IDA ViewA HexViewA Exports Inputs Names Functions Strings Structures Enums Segmentation
.00000000 ; LPCSTR lpType
.00000000 lpType dd offset aBinary
.00000000 aBinary ; DATA XREF: sub_401000:loc_h01000Tr
.00000000 ; "BINARY"
.00000000 ; LPCSTR lpName
.00000000 lpName dd offset aTgad
.00000000 aTgad ; DATA XREF: sub_401000+3ETr
.00000000 ; "TGAD"
.00000000 aTgad align 1M
.00000000 aBinary db 'BINARY',0 ; DATA XREF: .data:lpTypeTo
.00000000 align 4
.00000000 aTl db 'R1',00h,0 ; DATA XREF: sub_401000:loc_h01002Tr
.00000000 char ValueName[]
.00000000 ValueName db 'GinaDLL',0 ; DATA XREF: sub_401000+3ETr
.00000000 char SubKey[]
.00000000 SubKey db 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon',0 ; DATA XREF: sub_401000+17Tr
.00000000 align 4
.00000000 aBr db 'DB',00h,0 ; DATA XREF: sub_401000+11Tr
.00000000 char aMsina32_dll[]
.00000000 aMsina32_dll db 'msina32.dll',0 ; DATA XREF: sub_401000+E6Tr
.00000000 align 1M
.00000000 char aWb[]
.00000000 aWb db 'ub',0 ; DATA XREF: sub_401000+E1Tr
.00000000 align 4
.00000000 aMsina32_dll db 'msina32.dll',0 ; DATA XREF: _main+28Tr
.00000000 align 1M
.00000000 off_h000C4 dd offset __exit ; DATA XREF: __anq_exit+10Tr
.00000000 off_h000C4 dd 1 ; DATA XREF: _FF_HSCANNER+ETr
.00000000 sub_h01679+400Tr ; DATA XREF: sub_401000+400Tr
.00000000 sub_h01679+400Tr ; "(null)"
.00000000 off_h000C8 dd offset aNull ; DATA XREF: sub_401000+26Tr
.00000000 off_h000C8 dd 0 ; DATA XREF: sub_401000+26Tr
.00000000 sub_h01679+400Tr ; "(null)"

```

Figura 1.15 Presenza di una subkey in terza sezione

Notiamo in particolare il riferimento alla subkey **SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon**. Tale subkey è una chiave del Registro di sistema di Windows associata alle impostazioni di login e di avvio del sistema operativo. Quando un malware modifica o aggiunge voci in questa subkey, potrebbe cercare di influenzare il comportamento di avvio del sistema o intercettare informazioni di login.

Notiamo inoltre la presenza di **GinaDLL**: in alcune versioni di Windows precedenti a Windows Vista, la chiave "GinaDLL" specificava la DLL per la Graphical Identification and Authentication (GINA), che gestisce

Giorno 1

l'interfaccia utente di accesso. Modifiche a questa chiave potrebbero indicare un tentativo di sostituire o intercettare il processo di autenticazione.

SEZIONE .RSRC:

Sappiamo che .rsrc contiene le risorse: come icone, immagini, stringhe localizzate e altri dati non eseguibili utilizzati dall'applicazione. Tale sezione non viene caricata di default su IDA Pro, se vogliamo che sia visibile dobbiamo spuntare la casella di Load Resources all'apertura del malware:

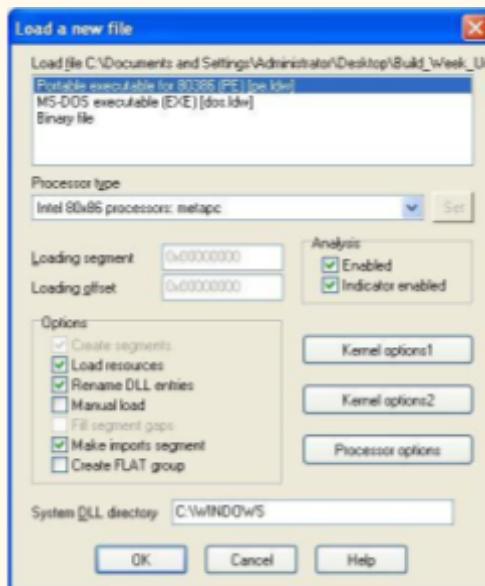


Figura 1.16 Il flag “Load Resources”

Così vediamo che compare nella sezione “Segments”:

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es	ss	ds	fs	gs
.text	00401000	00407000	R	.	X	.	L	para	0001	public	CODE	32	0000	0000	0003	FFF...	FFF...
.idata	00407000	004070DC	R	L	para	0002	public	DATA	32	0000	0000	0003	FFF...	FFF...
.rdata	004070DC	00408000	R	L	para	0002	public	DATA	32	0000	0000	0003	FFF...	FFF...
.data	00408000	0040BEA8	R	W	L	para	0003	public	DATA	32	0000	0000	0003	FFF...	FFF...
.rsrc	0040C000	0040E000	R	L	para	0004	public	DATA	32	0000	0000	0003	FFF...	FFF...	

Figura 1.17 Le sezioni del malware visualizzate da IDA Pro

Giorno 1

Ci spostiamo sulla sezione e vediamo come si presenta:

```
File View A Review D Exports I Imports N Names Functions S Strings Structures E Enums Segmentation  
.data:00400E04  
.FNC:0040C000 ; Section 4. (virtual address 0000C000)  
.FNC:0040C000 ; Virtual size : 00001A70 ( 6768.)  
.FNC:0040C000 ; Section size in File : 00002000 ( 8192.)  
.FNC:0040C000 ; Offset to raw data for section: 00000000  
.FNC:0040C000 ; Flags 40000040: Data Readable  
.FNC:0040C000 ; Alignment : default  
.FNC:0040C000 ;  
.FNC:0040C000 ; Segment type: Pure data  
.FNC:0040C000 ; Segment permissions: Read  
.FNC:0040C000 _FNC segment para public 'DATA' use32  
.FNC:0040C000 assume CS:_FNC  
.FNC:0040C000 org 40C000h  
* .FNC:0040C000 db 0  
* .FNC:0040C001 db 0  
* .FNC:0040C002 db 0  
* .FNC:0040C003 db 0  
* .FNC:0040C004 db 0  
* .FNC:0040C005 db 0  
* .FNC:0040C006 db 0  
* .FNC:0040C007 db 0  
* .FNC:0040C008 db 0  
* .FNC:0040C009 db 0  
* .FNC:0040C00A db 0  
* .FNC:0040C00B db 0  
* .FNC:0040C00C db 0  
* .FNC:0040C00D db 1  
* .FNC:0040C00E db 0  
* .FNC:0040C00F db 0  
* .FNC:0040C010 db 0  
* .FNC:0040C011 db 0  
* .FNC:0040C012 db 0  
* .FNC:0040C013 db 0  
* .FNC:0040C014 db 0
```

Figura 1.18 La quarta sezione del malware, .rsrc

Procedendo nella sezione, possiamo osservare la presenza in tutto di 3 subroutine. Ricordiamo che la subroutine è una sequenza di istruzioni di programma raggruppate insieme per eseguire una specifica operazione o compito all'interno di un programma più grande. In altri termini, una subroutine è una porzione di codice che può essere chiamata (o invocata) da altre parti del programma per eseguire un'operazione specifica.

In figura 1.18, per esempio, ne vediamo una:

```

    .srcrc:BB4BCACB
    .srcrc:BB4BCACB ; ===== SUBROUTINE =====
    .srcrc:BB4BCACB
    .srcrc:BB4BCACB
    .srcrc:BB4BCACB sub_ABCACB proc near ; CODE XREF: .srcrc:BB4BCBF81p
    .srcrc:BB4BCACB
    .srcrc:BB4BCACB var_21C = dword ptr -21Ch
    .srcrc:BB4BCACB var_218 = dword ptr -218h
    .srcrc:BB4BCACB var_20C = dword ptr -20Ch
    .srcrc:BB4BCACB arg_0 = dword ptr 8
    .srcrc:BB4BCACB arg_4 = dword ptr 0
    .srcrc:BB4BCACB
    .srcrc:BB4BCACB mov eax, [esp+arg_4]
    .srcrc:BB4BCACB sub esp, 208h
    .srcrc:BB4BCACB cmp eax, 1
    .srcrc:BB4BCACB jnz short loc_A80527
    .srcrc:BB4BCACB push esi
    .srcrc:BB4BCACB mov esi, [esp+208h+arg_0]
    .srcrc:BB4BCACB push esi
    .srcrc:BB4BCACB call dword ptr ds:100002010h
    .srcrc:BB4BCACB lea eax, [esp+218h+var_20C]
    .srcrc:BB4BCACB push eax
    .srcrc:BB4BCACB
    .srcrc:BB4BCACB push eax
    .srcrc:BB4BCACB
    .srcrc:BB4BCACB mov ds:100003F0h, esi
    .srcrc:BB4BCACB call dword ptr ds:100002010h
    .srcrc:BB4BCACB lea ecx, [esp+218h+var_214]
    .srcrc:BB4BCACB push 100002010h
    .srcrc:BB4BCACB push ecx
    .srcrc:BB4BCACB call dword ptr ds:100002010h
    .srcrc:BB4BCACB lea edx, [esp+220h+var_21C]
    .srcrc:BB4BCACB push edx
    .srcrc:BB4BCACB call dword ptr ds:100002020h
    .srcrc:BB4BCACB xor ecx, ecx
    .srcrc:BB4BCACB mov ds:10000324h, eax
    .srcrc:BB4BCACB test eax, eax

```

Figura 1.18 Esempio di subroutine nella sezione .rsrc

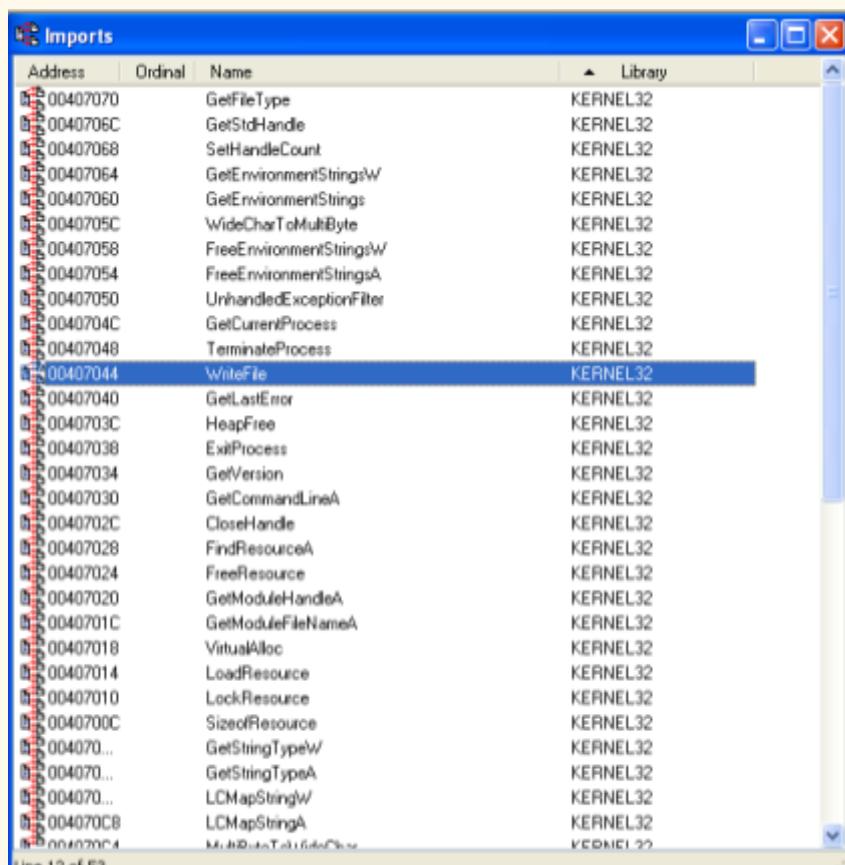
Librerie importate dal malware

Con CFF possiamo vedere le librerie/moduli importati dal malware, che sono solo 2, KERNEL32.dll con 51 funzioni e ADVAPI32.dll con 2 funzioni.

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	51	00007534	00000000	00000000	0000769E	0000700C
ADVAPI32.dll	2	00007528	00000000	00000000	000076D0	00007000

Figura 1.19 Librerie importate dal malware

Con IDA pro possiamo analizzare più in dettaglio le librerie importate. Nella finestra vista in figura 1.20 possiamo vedere le funzioni che ciascuno porta con sé, che possono darci una buona idea di come funziona il malware. Molte funzioni sono legate alla gestione della memoria, tuttavia alcune altre potrebbero portare a comportamenti dannosi.



Address	Ordinal	Name	Library
00407070		GetFileType	KERNEL32
0040706C		GetStdHandle	KERNEL32
00407068		SetHandleCount	KERNEL32
00407064		GetEnvironmentStringW	KERNEL32
00407060		GetEnvironmentStrings	KERNEL32
0040705C		WideCharToMultiByte	KERNEL32
00407058		FreeEnvironmentStringW	KERNEL32
00407054		FreeEnvironmentStringsA	KERNEL32
00407050		UnhandledExceptionFilter	KERNEL32
0040704C		GetCurrentProcess	KERNEL32
00407048		TerminateProcess	KERNEL32
00407044		WriteFile	KERNEL32
00407040		GetLastError	KERNEL32
0040703C		HeapFree	KERNEL32
00407038		ExitProcess	KERNEL32
00407034		GetVersion	KERNEL32
00407030		GetCommandLineA	KERNEL32
0040702C		CloseHandle	KERNEL32
00407028		FindResourceA	KERNEL32
00407024		FreeResource	KERNEL32
00407020		GetModuleHandleA	KERNEL32
0040701C		GetModuleFileNameA	KERNEL32
00407018		VirtualAlloc	KERNEL32
00407014		LoadResource	KERNEL32
00407010		LockResource	KERNEL32
0040700C		SizeofResource	KERNEL32
004070...		GetStringTypeW	KERNEL32
004070...		GetStringTypeA	KERNEL32
004070...		LCMapStringW	KERNEL32
004070C8		LCMapStringA	KERNEL32
004070C4		MessageBoxA	KERNEL32

Figura 1.20 Funzioni importate dal malware

Giorno 1

Quando si analizza il malware, è necessario essere consapevoli dell'uso di determinate funzioni che potrebbero essere utilizzate per creare processi, esfiltrare o modificare dati e allocare memoria per inserire codice in processi legittimi. Dalla libreria KERNEL32 possiamo trovare le seguenti funzioni:

- **TerminateProcess:** Come indica il nome, TerminateProcess termina i processi attualmente in esecuzione sul sistema, che potrebbero essere processi relativi alla sicurezza o processi che interferiscono con l'ottenimento della persistenza del malware.
- **WriteFile:** Fa quello che dice il nome, scrive un file, questo file potrebbe includere dati sensibili raccolti dal sistema infetto e poi inviati all'aggressore. WriteFile potrebbe essere utilizzato anche per molte altre funzioni come ottenere la persistenza o modificare file di sistema importanti per creare una backdoor, eliminare prove o danneggiare il funzionamento.
- **GetVersion:** Ottiene la versione del sistema infetto.
- **VirtualAlloc:** Questa funzione viene utilizzata per allocare memoria nello spazio degli indirizzi del processo chiamante. Il malware utilizza spesso questa funzione per allocare memoria per il proprio codice o per eseguire l'inserimento di codice in processi legittimi.
- **LoadResource:** Il malware può utilizzare LoadResource per estrarre codice dannoso incorporato o risorse di dati dal proprio file eseguibile o da altri eseguibili legittimi. Questa tecnica consente al malware di nascondere il suo carico utile all'interno delle risorse di un file apparentemente innocuo, rendendone più difficile il rilevamento.
- **ReadFile:** ReadFile può leggere altri file di sistema, quindi scriverli con WriteFile e quindi esfiltrare.
- **LoadLibraryA:** Può richiamare una libreria che contiene ancora più funzioni potenzialmente dannose.
- **GetStartupInfoA:** Il malware può abusare di questa funzione per raccogliere informazioni sul sistema su cui è in esecuzione, inclusi dettagli sugli input standard, sull'output e sugli handle di errore del

Giorno 1

processo. Queste informazioni possono essere utilizzate per manipolare la comunicazione dei processi, adattarsi all'ambiente e potenzialmente eludere il rilevamento o l'analisi da parte di ricercatori e strumenti di sicurezza.

Quelle sopra riportate sono alcune delle funzioni importate da KERNEL32.dll sospette, tuttavia le 2 funzioni importate da ADVAPI32.dll sono le più evidentemente dannose.

- **RegCreateKeyExA:** Il malware può utilizzare questa funzione per creare o modificare entry di registro per ottenere la persistenza, eludere il rilevamento, archiviare dati di configurazione o eseguire altre attività dannose.
- **RegSetValueExA:** Viene utilizzato per impostare i dati e il tipo di valori all'interno dei registri.

GIORNO 2

Spiegazione di istruzioni assembly

1) Alla locazione **00401027** viene chiamata la funzione **RegCreateKeyExA** la quale è una funzione dell'API di Windows che viene utilizzata per creare o aprire una chiave di registro. Questa funzione è parte delle API di Windows fornite per interagire con il registro di sistema di Windows.

```
.text:00401021          call    ds:RegCreateKeyExA
.text:00401027          test    eax, eax
.text:00401029          jz     short loc_401032
```

Figura 2.1 Funzione RegCreateKeyExA

2) Il parametro presente alla locazione **0040101C** viene passato alla locazione **00401021** sullo stack tramite comando push.

```
.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
```

Figura 2.2 Creazione registro

3) L'oggetto rappresentato alla locazione **00401017** è una sottochiave a cui viene assegnato il valore di una stringa, in questo caso è “SOFTWARE\Microsoft\Windows NT\CurrentVe”. Alla locazione **0040101C** viene pushato il valore della chiave di registro.

4) All'indirizzo **00401027** è presente l'istruzione condizionale test (con sintassi istruzione destinazione, sorgente), la quale effettua un'operazione AND senza modificare il contenuto degli operandi. Tuttavia, questa istruzione modifica il flag ZF (zero flag) del registro EFLAGS, che viene settato ad 1 se e solo se il risultato dell'AND è 0. Viene di fatto utilizzato per controllare se un valore è zero o meno.

```
.text:00401027          test    eax, eax
.text:00401029          jz     short loc_401032
```

Figura 2.3 Salto condizionale

All'indirizzo **00401029** è presente un salto condizionale jz (jump if zero con sintassi istruzione locazione di memoria). Se l'istruzione precedente ha impostato il registro eax a zero, allora il salto viene eseguito, portando l'esecuzione a **loc_401032**.

Giorno 2

In figura 2.4 si può vedere la rappresentazione grafica del suddetto salto condizionale. In rosso i salti non effettuati, viceversa in verde.



Figura 2.4 Diagramma dei salti

Possiamo affermare che le istruzioni che abbiamo analizzato sono riconducibili a un **ciclo if**.

5) Da questo codice assembly è possibile ricavare il codice in linguaggio C originale:

```

if (eax == 0) { // Testa se eax è uguale a zero

    goto loc_401032; // Se eax è zero, salta al codice etichettato come loc_401032
} else {
    eax = 1; // Se eax non è zero, imposta eax a 1
    goto loc_40107B; // Salta al codice etichettato come loc_40107B
}

```

Figura 2.5 Traduzione in codice C

6) Il valore del parametro `ValueName` è **GinaDLL**.

```
.text:0040103E          push    offset ValueName ; "GinaDLL"
```

Figura 2.6 Modifica a registro

Conclusioni del giorno

Sembra che il malware stia cercando di manipolare il registro di sistema di Windows per influenzare il comportamento del sistema operativo. In particolare, sta creando una chiave di registro specifica e assegnando un valore a una sottochiave.

La chiave di registro che viene manipolata è “SOFTWARE\Microsoft\Windows NT\CurrentVersion” e il valore della sottochiave è impostato su “GinaDLL”.

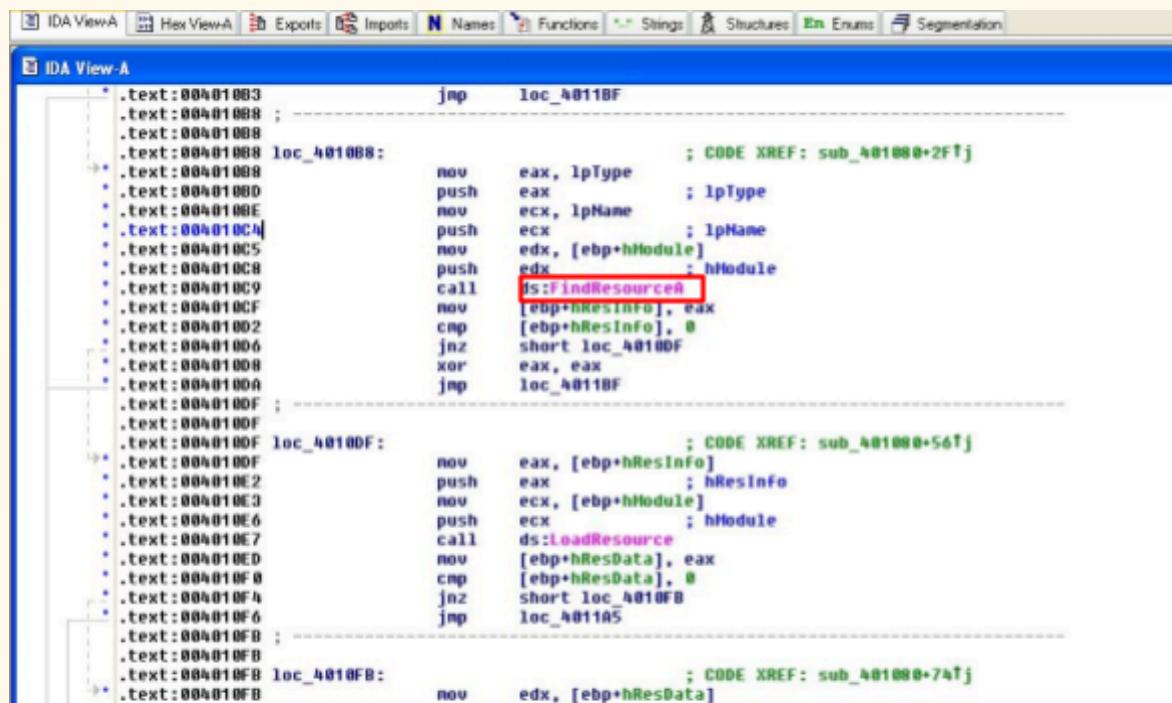
Gina.dll, che sta per Graphical Identification and Authentication Dynamic Link Library, è un file DLL impiegato nei sistemi operativi Windows per abilitare le funzioni di autenticazione e riconoscimento grafico. Questo file ha il compito di gestire l’interfaccia di autenticazione per gli utenti che si connettono a un computer Windows XP. Gina.dll viene caricata da Winlogon ed entra in funzione prima dell’inizio del processo di login di Windows, prima cioè che vengano richieste le credenziali dell’utente.

Un malware potrebbe cercare di sostituire GinaDLL con una propria versione per intercettare le credenziali degli utenti.

GIORNO 3

Valore del parametro “ResourceName”

Per rispondere al quesito in oggetto, lanciamo il software IDA Pro. Dalla tab IDA View, con la barra spaziatrice ci spostiamo in modalità testuale. A questo punto scorriamo le righe di codice fino a raggiungere la locazione di memoria **00401080** e individuiamo la funzione **FindResourceA**:



The screenshot shows the assembly view in IDA Pro. The function **FindResourceA** starts at address **00401080**. The assembly code is as follows:

```
text:00401080        jmp    loc_4011BF
text:00401088 ; -----
text:00401088 loc_401088:
    mov    eax, lpType           ; CODE XREF: sub_401080+2Fj
    push   eax                 ; lpType
    mov    ecx, lpName          ; lpName
    push   ecx                 ; lpName
    mov    edx, [ebp+hModule]   ; hModule
    push   edx                 ; hModule
    call   ds:FindResourceA
    mov    [ebp+hResInfo], eax
    cmp    [ebp+hResInfo], 0
    jnz    short loc_4010DF
    xor    eax, eax
    jmp    loc_4011BF

text:0040109F ; -----
text:0040109F loc_40109F:
    mov    eax, [ebp+hResInfo]   ; CODE XREF: sub_401080+56Tj
    push   eax                 ; hResInfo
    mov    ecx, [ebp+hModule]   ; hModule
    push   ecx                 ; hModule
    call   ds:LoadResource
    mov    [ebp+hResData], eax
    cmp    [ebp+hResData], 0
    jnz    short loc_4010FB
    jmp    loc_4011A5

text:004010FB ; -----
text:004010FB loc_4010FB:
    mov    edx, [ebp+hResData]   ; CODE XREF: sub_401080+74Tj
```

Figura 3.1 Funzione *FindResourceA* visualizzata in IDA Pro

Notiamo che uno dei parametri caricati dalla funzione è **lpName**. Con un doppio click su **lpName** veniamo reindirizzati alla sezione **.data** dove possiamo notare che **lpName** è una stringa (LPCSTR) e che il valore della stringa è “**TGAD**”:

Giorno 3

```

    .data:00400B2E db 0
    .data:00400B2F db 0
    .data:00400B30 ; LPCSTR lpType
    .data:00400B30 dd offset aBinary ; DATA XREF: sub_401000:loc_A010007r
    .data:00400B30
    .data:00400B34 ; LPCSTR lpName
    .data:00400B34 dd offset aTgad ; DATA XREF: sub_401000+3ETr
    .data:00400B34
    .data:00400B38 aTgad db 'TGAD',0 ; DATA XREF: .data:lpNameTo
    .data:00400B38 align 10h
    .data:00400B40 aBinary db 'BINARY',0 ; DATA XREF: .data:lpTypeTo
    .data:00400B40 align 4
    .data:00400B44 aRI db 'RI',0Ah,0 ; DATA XREF: sub_401000:loc_A010027r
    .data:00400B4C ; char ValueName[]
    .data:00400B4C ValueName db 'GinaDLL',0 ; DATA XREF: sub_401000+3ETo
    .data:00400B54 ; char SubKey[]
    .data:00400B54 SubKey db 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon',0 ; DATA XREF: sub_401000+17To
    .data:00400B54
    .data:00400B8A align 4
    .data:00400B8C aDr db 'DR',0Ah,0 ; DATA XREF: sub_401000+11BTo
    .data:00400B90 ; char aHsgina32_dll_0[]
    .data:00400B90 aHsgina32_dll_0 db 'Hsgina32.dll',0 ; DATA XREF: sub_401000+E6To
    .data:00400B90 align 10h
    .data:00400B9D ; char aWb[]
    .data:00400B9D aWb db 'WB',0 ; DATA XREF: sub_401000+E1To
    .data:00400B9D align 4
    .data:00400B9E aHsgina32_dll db '\Hsgina32.dll',0 ; DATA XREF: _main+78To
    .data:00400B9E align 10h
    .data:00400BC0 off_400B0C0 dd offset __exit ; DATA XREF: __amsq_exit+1CTr
    .data:00400BC4 duord_400B0C4 dd 1 ; DATA XREF: __FF_HSGANNER+ETr
    .data:00400BC4
    .data:00400BC8 off_400B0C8 dd offset aNull ; DATA XREF: sub_401679:loc_A019F27r

```

Figura 3.2 Il nome della risorsa (TGAD)

Possiamo giungere alla stessa conclusione anche caricando il malware sul debugger OllyDBG. Infatti raggiungendo lo stesso indirizzo vediamo che ResourceType è “BINARY”, mentre ResourceName è “TGAD”:

```

00401000 > RI 00004000 HOU ERX,DMORD PTR DS:[400000]
00401000 . 60 PUSH ERX
00401000 . 8B0D 34004000 HOU ECX,DMORD PTR DS:[400004]
00401000 . 51 PUSH ECX
00401000 . 8B55 08 MOV EDX,DMORD PTR SS:[EBP+8]
00401000 . 52 PUSH EDX
00401000 . FF15 28704000 CALL DMORD PTR DS:[<&KERNEL32.FindResourceA]
00401000 . 8945 EC HOU DWORD PTR SS:[EBP-14],ERX
00401000 . 837D EC 00 CMP DWORD PTR SS:[EBP-14],0
00401000 . 75 07 JNZ SHORT Malware_.0040100F
00401000 . 33C0 XOR ERX,ERX
00401000 . E9 E0000000 JMP Malware_.0040110F
00401000 > 8B45 EC HOU ERX,DMORD PTR SS:[EBP-14]
00401000 . 50 PUSH ERX
00401000 . 8B4D 08 MOV ECX,DMORD PTR SS:[EBP+8]
00401000 . 51 PUSH ECX
00401000 . FF15 14704000 CALL DMORD PTR DS:[<&KERNEL32.LoadResource]
00401000 . 8945 EB HOU DWORD PTR SS:[EBP-18],ERX
00401000 . 837D EB 00 CMP DWORD PTR SS:[EBP-18],0
00401000 . 75 05 JNZ SHORT Malware_.004010FB
00401000 > E9 A0000000 JMP Malware_.00401105
00401000 . 8B55 EB MOV EDX,DMORD PTR SS:[EBP-18]
00401000 . 52 PUSH EDX
00401000 . FF15 18704000 CALL DMORD PTR DS:[<&KERNEL32.LockResource]
00401000 . 8945 FS HOU DWORD PTR SS:[EBP-8],ERX
00401000 . 837D FS 00 CMP DWORD PTR SS:[EBP-8],0
00401000 . 75 05 JNZ SHORT Malware_.00401113
00401000 > E9 92000000 JMP Malware_.00401145
00401000 . 8B45 EC HOU ERX,DMORD PTR SS:[EBP-14]
00401000 . 50 PUSH ECX

```

Figura 3.3 La funzione FindResourceA visualizzata in OllyDBG

Identificazione di funzionalità implementate

Le funzioni che si susseguono in questa parte di codice sono quattro:

1. FindResourceA
2. LoadResource

3. LockResource
4. SizeofResource

```

.text:004010B8 loc_4010B8:           ; CODE XREF: sub_401080+2FTj
    .text:004010B8
    .text:004010BD push    eax      ; lpType
    .text:004010BE mov     ecx      ; lpName
    .text:004010C4 push    edx      ; lpName
    .text:004010C5 mov     edx, [ebp+hModule]
    .text:004010C8 push    edx      ; hModule
    .text:004010C9 call    ds:FindResourceA
    .text:004010CF mov     [ebp+hResInfo], eax
    .text:004010D2 cmp     [ebp+hResInfo], 0
    .text:004010D8 jnz    short loc_4010DF
    .text:004010DA xor    eax, eax
    .text:004010DF jmp    loc_4011BF

.text:004010DF loc_4010DF:           ; CODE XREF: sub_401080+56↑j
    .text:004010DF
    .text:004010E2 push    eax      ; hResInfo
    .text:004010E3 mov     ecx, [ebp+hModule]
    .text:004010E6 push    edx      ; hModule
    .text:004010E7 call    ds:LoadResource
    .text:004010ED mov     [ebp+hResData], eax
    .text:004010F0 cmp     [ebp+hResData], 0
    .text:004010F4 jnz    short loc_4010FB
    .text:004010F6 jmp    loc_4011A5

.text:004010FB loc_4010FB:           ; CODE XREF: sub_401080+74Tj
    .text:004010FB
    .text:004010FE push    edx      ; hResData
    .text:004010FF call    ds:LockResource
    .text:00401105 mov     [ebp+var_8], eax
    .text:00401108 cmp     [ebp+var_8], 0
    .text:0040110C jnz    short loc_401113
    .text:0040110E jmp    loc_4011A5

.text:00401113 loc_401113:           ; CODE XREF: sub_401080+8CTj
    .text:00401113
    .text:00401116 push    eax      ; hResInfo
    .text:00401117 mov     ecx, [ebp+hModule]
    .text:0040111A push    edx      ; hModule
    .text:0040111B call    ds:SizeofResource
    .text:00401121 mov     [ebp+dwSize], eax
    .text:00401124 cmp     [ebp+dwSize], 0
    .text:00401128 ja     short loc_40112C
    .text:0040112A jmp    short loc_4011A5

```

Figura 3.4 Le funzionalità del Dropper

Queste quattro funzioni ci riconducono ad un Dropper. Vediamo nel dettaglio che cosa fa ciascuna di esse:

FindResourceA:

Questa funzione viene utilizzata per individuare una risorsa specifica all'interno di un file eseguibile (o di un modulo) in base al nome e al tipo.

LoadResource:

Dopo aver individuato la risorsa con FindResourceA, la funzione LoadResource viene utilizzata per caricare la risorsa in memoria.

LockResource:

Dopo aver caricato la risorsa con LoadResource, LockResource viene utilizzata per ottenere un puntatore ai dati della risorsa.

SizeofResource:

Questa funzione restituisce la dimensione, in byte, di una risorsa specifica.

Queste quattro funzioni sono spesso utilizzate in combinazione da un dropper per individuare, caricare, ottenere un puntatore e determinare la dimensione di una risorsa contenente il payload malevolo. Questo tipo di approccio consente al dropper di operare con le informazioni necessarie per eseguire il suo payload dannoso.

Identificazione di funzionalità tramite analisi statica basica

Proveremo ora ad identificare la funzionalità Dropper tramite analisi statica basica con i tool CFF Explorer e exeinfope.

Apriamo **CFF Explorer** e ci spostiamo su *Section Headers*. Procediamo poi con l'analisi di ciascuna sezione. Nel riquadro in basso del software, dove viene visualizzata la versione in HEX/Ascii del contenuto del malware, ricerchiamo le quattro funzioni che hanno catturato la nostra attenzione. Le troviamo nella sezione **.rdata** come visto in figura 3.5 e 3.6:

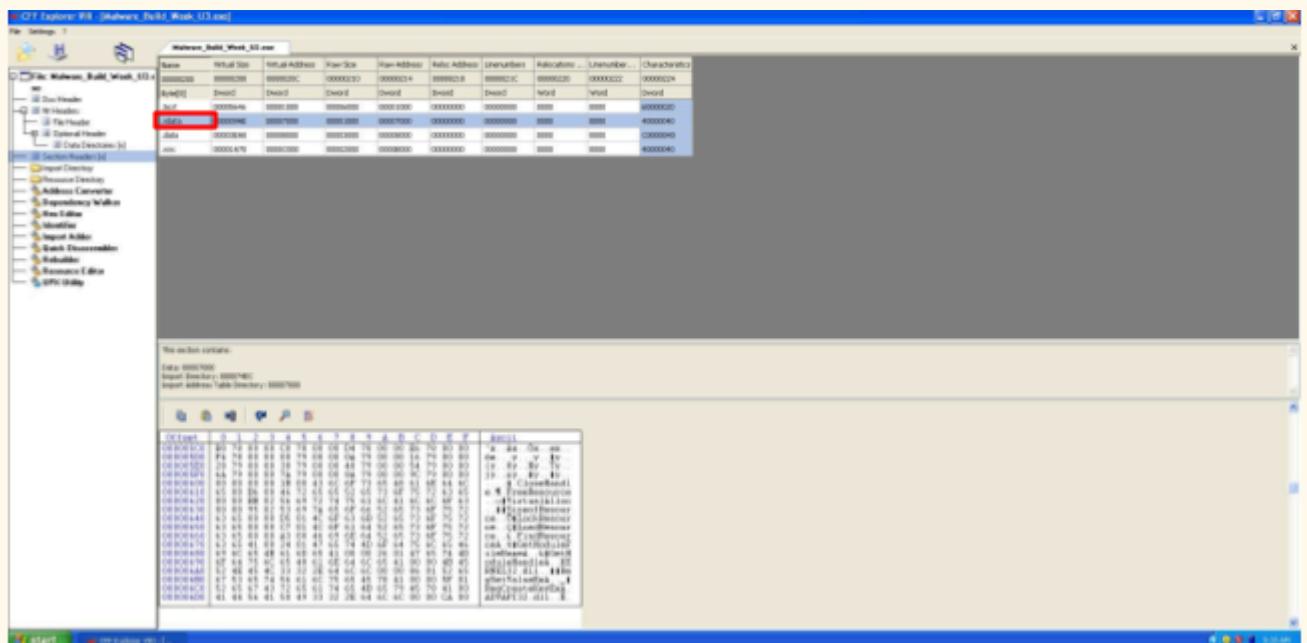


Figura 3.5 Ricerca delle funzionalità del dropper in CFF Explorer

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
000005C0	B0	78	00	00	C0	78	00	00	D4	78	00	00	E6	78	00	00	x..Ax..Cx..ax..
000005D0	F4	78	00	00	00	79	00	00	0A	79	00	00	16	79	00	00	Cx..y..y..Iy..
000005E0	28	79	00	00	38	79	00	00	48	79	00	00	54	79	00	00	(y..By..Ty..
000005F0	6A	79	00	00	7A	79	00	00	8A	79	00	00	9C	79	00	00	3y..zy..Iy..
00000600	00	00	00	00	1B	00	43	6C	6F	73	65	48	61	6E	64	6C	I CloseHandle
00000610	65	00	B6	00	46	72	65	65	52	65	73	6F	75	72	63	65	@.t!FreeResource
00000620	00	00	BB	02	56	69	72	74	75	61	6C	41	6C	6F	63		..!VirtualAlloc
00000630	00	00	95	02	53	69	7A	65	6F	66	52	65	73	6F	75	72	..!SizeofResour
00000640	63	65	00	00	D6	01	4C	6F	63	6B	52	65	73	6F	75	72	ce..O!LockResour
00000650	63	65	00	00	C7	01	4C	6F	61	64	52	65	73	6F	75	72	ce..Q!LoadResour
00000660	63	65	00	00	A3	00	46	69	6E	64	52	65	73	6F	75	72	ce..I FindResour
00000670	63	65	41	00	24	01	47	65	74	4D	6F	64	75	6C	65	46	cen..#!GetModuleF
00000680	69	6C	65	4E	61	6D	65	41	00	00	26	01	47	65	74	4D	i!leNameA ..!GetM
00000690	68	64	75	6C	65	49	61	6E	64	6C	65	41	00	04	4B	45	cdoleHandleA ..!Re
000006A0	52	4E	45	4C	33	32	2E	64	6C	6C	00	00	86	01	52	65	RNEL32.dll ..!Re
000006B0	67	53	65	74	56	61	6C	75	65	45	78	41	00	00	5F	01	g!SetValueExA ..!
000006C0	52	65	67	43	72	65	61	74	65	4B	65	79	45	78	41	00	RegCreateKeyExA ..!
000006D0	41	44	56	41	50	49	33	32	2E	64	6C	6C	00	00	CA	00	ADVAPI32.dll ..!E

Figura 3.6 Zoom sul contenuto HEX/Ascii

Giorno 3

Proviamo a fare lo stesso utilizzando un secondo tool di analisi statica basica, **exeinfope**. Lo apriamo, carichiamo il malware e ci spostiamo sulla seconda sezione. Anche qui analizzando il contenuto in HEX/Ascii riusciamo ad individuare le quattro funzioni:

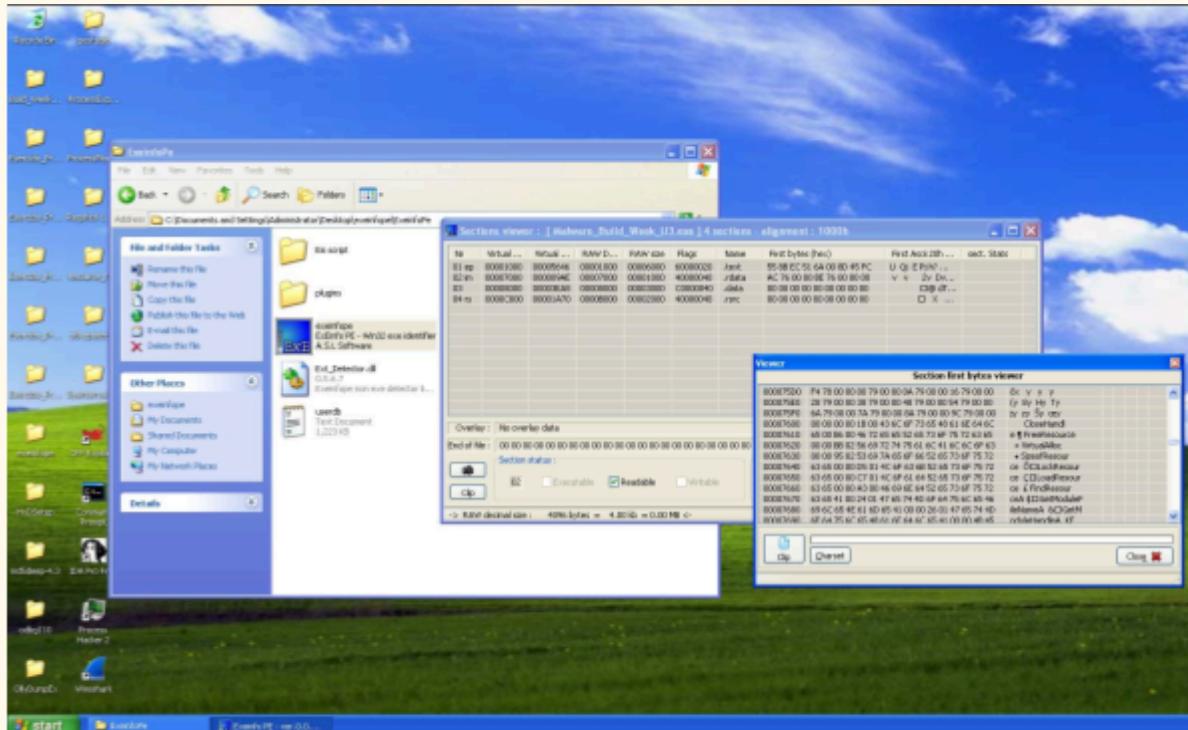


Figura 3.7 Ricerca delle funzionalità del dropper in exefinfope

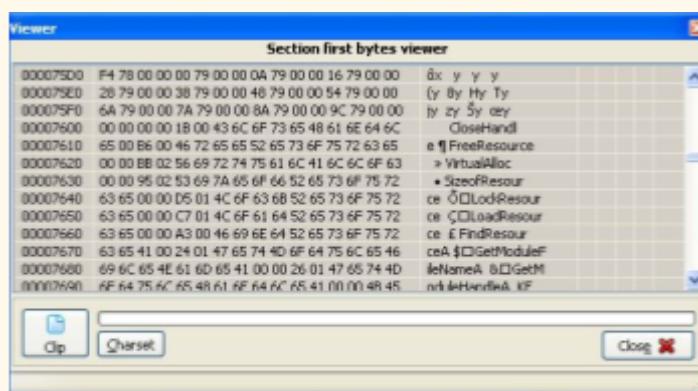


Figura 3.8 Zoom del contenuto HEX/Ascii

Possiamo quindi concludere che le quattro funzionalità del Dropper possono essere individuate anche con tool di analisi statica basica.

Flusso delle funzione

All'interno della funzione Main vengono eseguite diverse funzioni, che abbiamo schematizzato nella figura 3.9. Come si può vedere, i due processi rispettivamente carcano il payload malevolo e intervengono sulla chiave di registro:

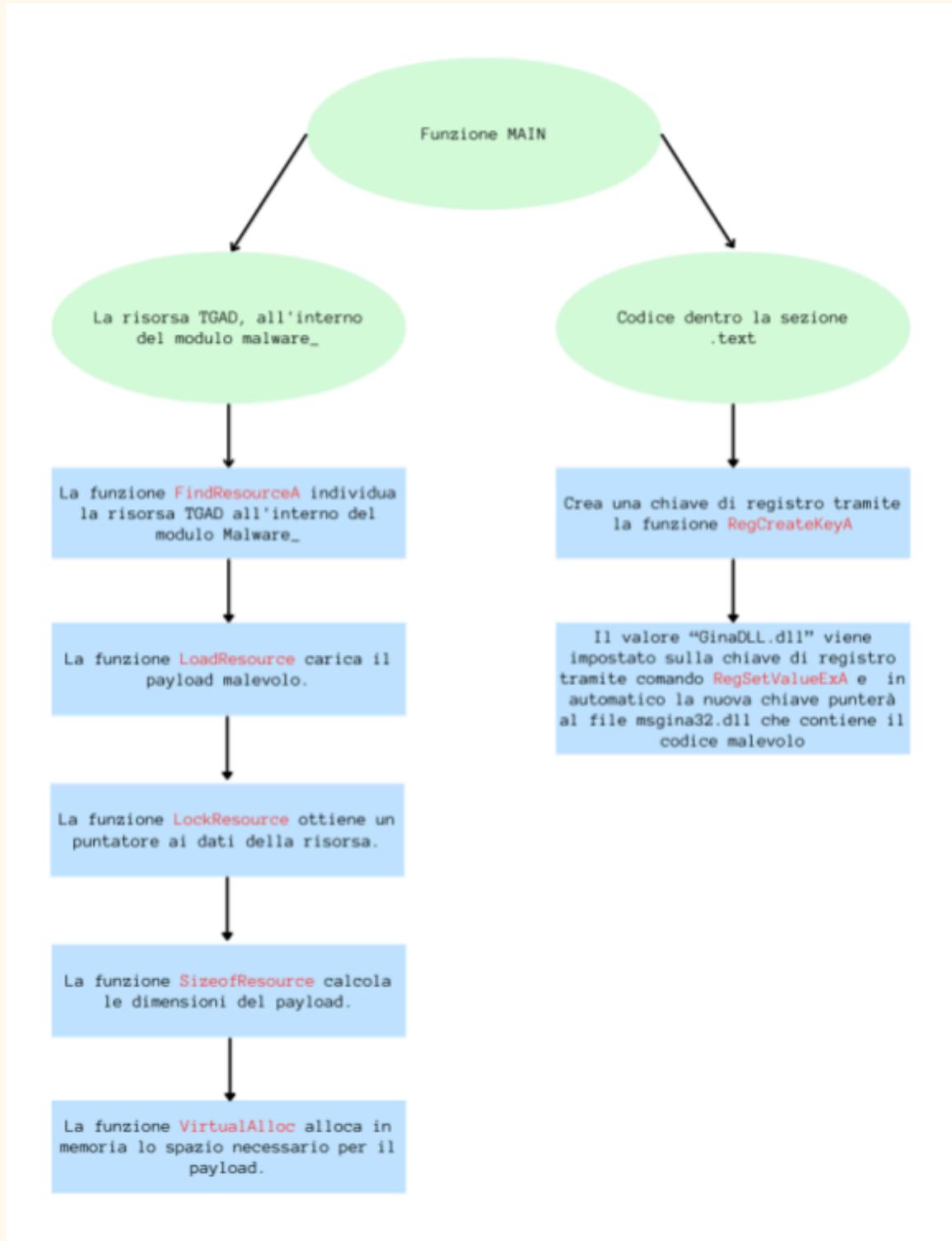


Figura 3.9 Schema delle funzioni principali

GIORNO 4

Analisi dinamica basica con procmon

In questo giorno, per l'esecuzione della task, effettueremo un'analisi dinamica basica che ci permetterà di analizzare il malware mentre è in attività tramite il tool ProcessMonitor il quale consente di visualizzare in tempo reale l'attività sul file system, sul registro, sui processi e sulle attività di rete.

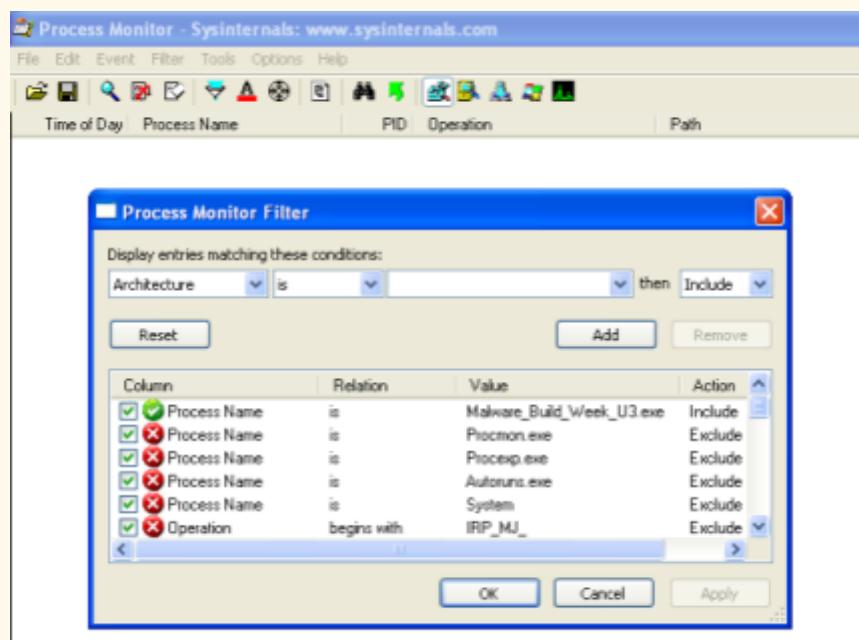


Figura 4.1 Filtri di Procmon

Visualizzando **ProcessMonitor** ci assicuriamo che sia in ascolto semplicemente osservando che la lente d'ingrandimento non sia barrata di rosso, e selezioniamo il filtro sul processo **Malware_Build_Week.U3.exe**. Successivamente eseguiamo il malware e notiamo comparire il file **msgina32.dll** una libreria malevola che ci fa comprendere la possibile natura del virus identificandola come Dropper.

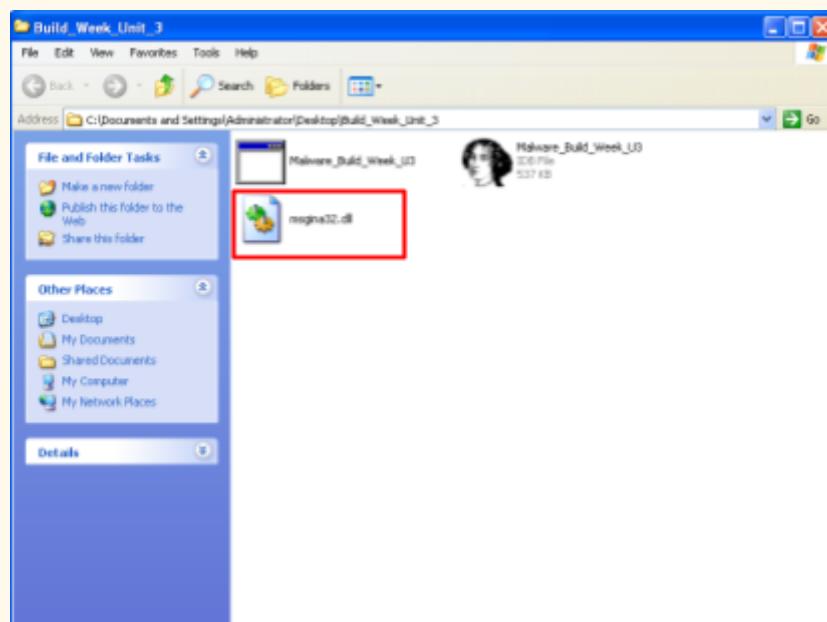


Figura 4.2 msgina.dll

Contemporaneamente alla creazione di msgina32.dll ProcessMonitor cattura le attività del malware segnalando la creazione della chiave di registro identificata come hklm\software\microsoft\windowsnt\current version\winlogon che poi setta subito sotto denominandola **GinaDLL**.

Malware_Build_Week_U3...	292	CloseFile	C:\Documents and Settings\Administrator\Desktop\Build\Week_Unit_3\msgina32.dll	SUCCESS
Malware_Build_Week_U3...	292	RegCreateKey	HKLMSOFTW\Microsoft\Windows NT\CurrentVersion\WinLogon	SUCCESS
Malware_Build_Week_U3...	292	RegSetValue	HKLMSOFTW\Microsoft\Windows NT\CurrentVersion\WinLogon\GinaDLL	SUCCESS
Malware_Build_Week_U3...	292	RegCloseKey	HKLMSOFTW\Microsoft\Windows NT\CurrentVersion\WinLogon	SUCCESS
Malware_Build_Week_U3...	292	ThreadExit		SUCCESS

Figura 4.3 Creazione chiave di registro

Analisi del registro con Regshot

Per avere un quadro più chiaro e completo della situazione abbiamo avviato **regshot**, un utile tool che permette di tenere traccia delle modifiche apportate alle chiavi di registro che i malware comunemente alterano. Effettuiamo la prima istantanea cliccando su **1st shot**.



Figura 4.4 1st shot

Successivamente avviamo il malware. Aspettiamo qualche secondo e clicchiamo su **2nd shot** per avere una seconda istantanea del registro.

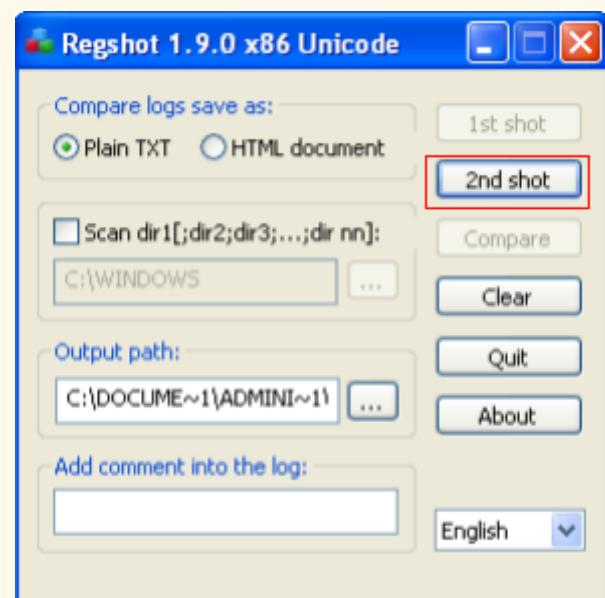


Figura 4.5 2nd shot 40

Giorno 4

Infine clicchiamo su “*compare*” per rilevare le modifiche apportate al sistema.

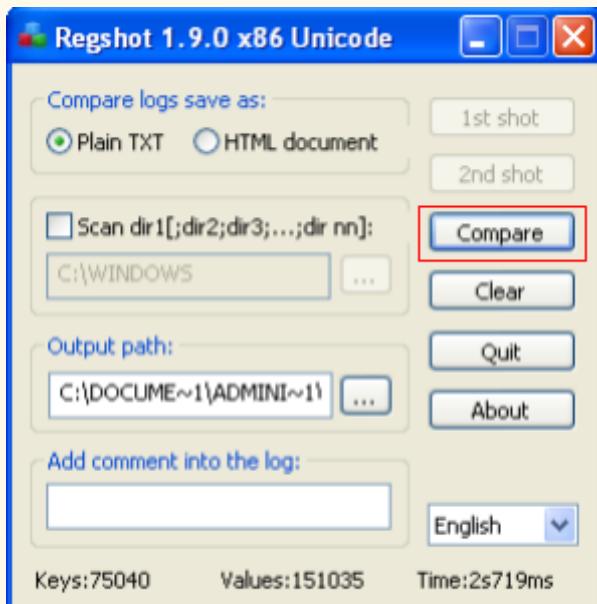


Figura 4.6 Comparazione Scatti

Regshot ci mostra dettagliatamente le differenze delle chiavi e dei valori di registro modificati:

Figura 4.7 Modificazione di valori

Possiamo notare che viene creata la chiave di registro HKLM(Local Machine)\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDll.

Verifica delle modifiche su regedit

Aprendo l'editor di registro di Windows è possibile vedere che la nuova chiave è effettivamente presente e punta alla libreria malevola esportata dal malware.

DefaultDomainName	REG_SZ	MALWARE_TEST
DefaultUserName	REG_SZ	Administrator
ForceLogon	REG_DWORD	0x00000000 (0)
GinaDLL	REG_SZ	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll
HibernationPreviouslyEnabled	REG_DWORD	0x00000001 (1)
LegalNoticeCaption	REG_SZ	
LegalNoticeText	REG_SZ	
LogonType	REG_DWORD	0x00000001 (1)

Figura 4.8 Nuova chiave di registro su winlogon

In ultima analisi siamo poi passati alla visualizzazione dell'attività sul file system che ci ha permesso di osservare più in dettaglio la creazione del file msgina32.dll.

Malware_Build_Week_U3...	316	CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS
Malware_Build_Week_U3...	316	CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS
Malware_Build_Week_U3...	316	CloseFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS
Malware_Build_Week_U3...	316	WriteFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS
Malware_Build_Week_U3...	316	WriteFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS
Malware_Build_Week_U3...	316	CloseFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS

Figura 4.9 Nuovo file msgina32.dll

In conclusione, l'analisi dinamica basica ci ha permesso di comprendere al meglio la natura del malware, il suo comportamento e le modifiche che apporta alle impostazioni del sistema operativo.

GIORNO 5

Cosa succede se il .dll lecito viene sostituito con il .dll malevolo

GINA (Graphical Identification and Authentication) è un componente di Windows utilizzato per la gestione dell'autenticazione degli utenti tramite interfaccia grafica. Se un file GINA legittimo viene sostituito da un file malevolo, possono verificarsi varie conseguenze indesiderate.

Il file DLL malevolo può intercettare e catturare le credenziali dell'utente, come username e password, durante il processo di autenticazione.

Una volta che l'attaccante ha ottenuto le credenziali, può utilizzarle per ottenere accesso non autorizzato al sistema ed eseguire azioni dannose: **accedere a dati sensibili, alterare il flusso di autenticazione**, ad esempio consentendo l'accesso a utenti non autorizzati o impedendo l'accesso agli utenti legittimi, e **compromettere la sicurezza complessiva del sistema**.

Inoltre, l'eventuale utilizzo di una backdoor potrebbe consentire all'attaccante di accedere al sistema in futuro senza la necessità di ulteriori credenziali.



Figura 5.1 Schermata di login falsa

Le attività svolte nei cinque giorni della Build Week 3 ci hanno permesso di analizzare i singoli elementi del malware, giungendo ad una visione d'insieme sul suo funzionamento. Questa visione d'insieme è stata descritta nel diagramma riportato in figura 5.2.

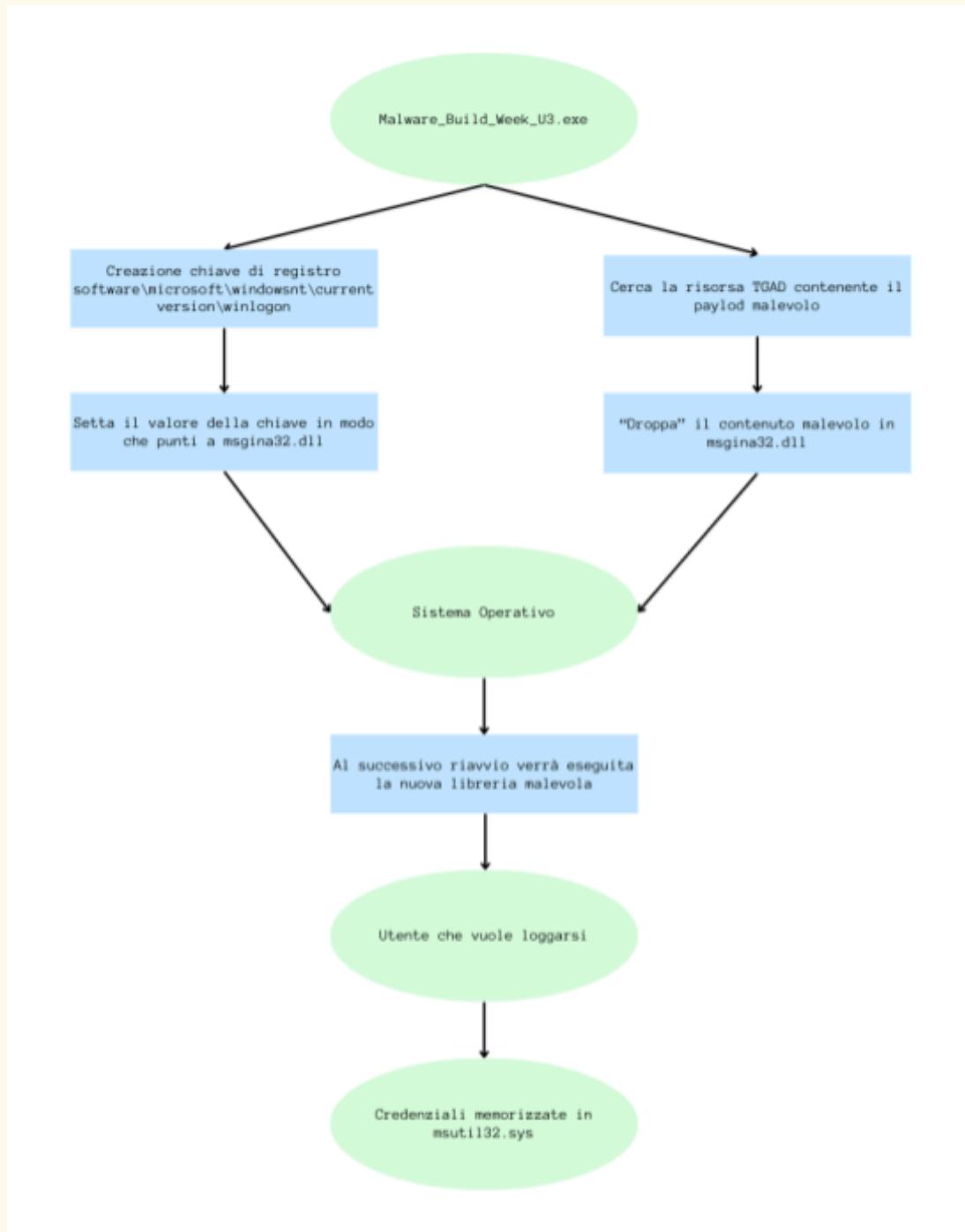


Figura 5.2 Flusso globale del malware

Osservazioni extra e conclusioni

Il malware analizzato dove salva le credenziali rubate?

Analizzando le stringhe della libreria malevola siamo riusciti a capire dove il malware salva le credenziali.

All'interno di un file possiamo trovare delle stringhe. Queste stringhe possono essere utilizzate per vari scopi, come mostrare messaggi all'utente, dettagliare l'uso del software o specificare URL a cui connettersi.

L'utility da riga di comando chiamata “*strings*” è uno strumento utile per estrarre tutte le stringhe presenti all'interno di un file. Questo ci consente di analizzare il contenuto del file e individuare eventuali informazioni rilevanti.

Avviamo l'utility da cmd con il comando *strings path del malware*:

```
C:\Documents and Settings\Administrator\Desktop\SysinternalsSuite>strings "C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll"
```

Figura 6.1 Comando cmd

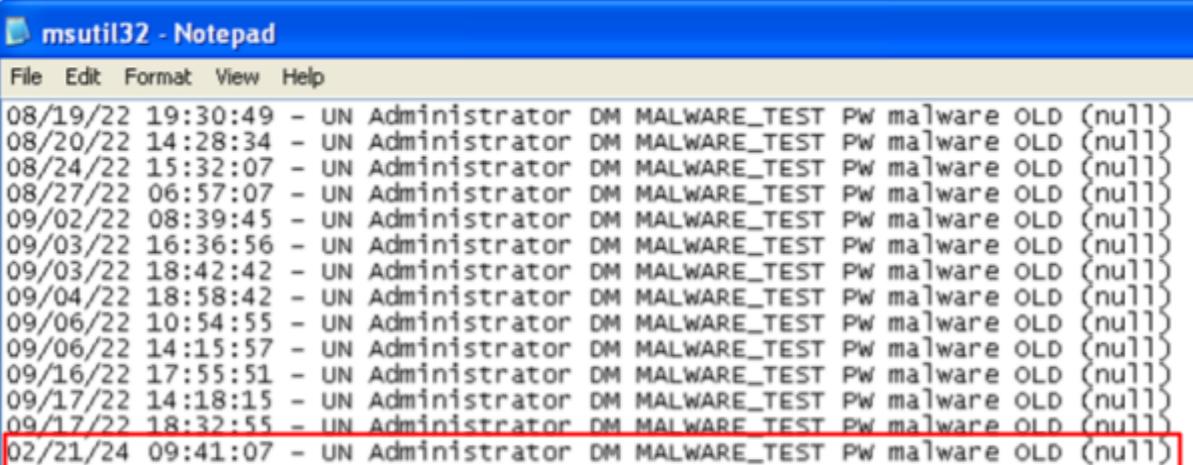
Oltre alle stringhe di funzioni analizzate nei giorni precedenti, abbiamo notato la presenza della stringa **msutil32.sys**:

```
ErrorCode:%d ErrorMessage:%s.  
%s %s - %s  
msutil32.sys  
0&0-080j0z0  
3!313A3Q3a3q3  
4#4*474B4Y4  
4Z5c5  
6!686b6h6n6t6z6  
??"?"<7J7\?7
```

Figura 6.2 Stringhe presenti in msgina.dll

Poiché il file è .sys, estensione tipica dei file di sistema, ci spostiamo all'interno della directory di sistema C:\WINDOWS\system32, dove notiamo la presenza del suddetto file generato dal malware.

Aprendo il file con il blocco note possiamo notare che è un log delle credenziali immesse nella interfaccia malevola creata dal malware.



Date	Time	User	Type	Action	Details
08/19/22	19:30:49	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
08/20/22	14:28:34	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
08/24/22	15:32:07	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
08/27/22	06:57:07	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
09/02/22	08:39:45	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
09/03/22	16:36:56	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
09/03/22	18:42:42	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
09/04/22	18:58:42	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
09/06/22	10:54:55	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
09/06/22	14:15:57	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
09/16/22	17:55:51	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
09/17/22	14:18:15	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
09/17/22	18:32:55	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}
02/21/24	09:41:07	-	UN	Administrator	DM MALWARE_TEST Pw malware OLD {null}

Figura 6.3 Stringhe presenti in msutil32.sys

Incident response

Presupponiamo che questa analisi l'abbiamo effettuata su un file presente in un PC dell'azienda per cui lavoriamo. Secondo un buon piano di incident response dovremmo eseguire le seguenti azioni:

- 1. Isolare il sistema infetto:** il PC infetto dovrebbe essere immediatamente isolato dalla rete aziendale per prevenire la diffusione del malware ad altri sistemi.
- 2. Rimozione del malware:** utilizzare un software di rimozione specializzato per rimuovere il malware dal sistema infetto.
- 4. Aggiornare le macchine:** assicurarsi che tutte le macchine aziendali siano aggiornate con le ultime patch di sicurezza. I malware spesso sfruttano vulnerabilità note che possono essere prevenute con gli aggiornamenti di sicurezza.
- 5. Formazione dei dipendenti:** formare i dipendenti su come riconoscere e evitare potenziali minacce di sicurezza, come phishing, download sospetti e siti web non sicuri.
- 6. Implementare sistemi di sicurezza avanzati:** come un NGFW (next generation firewall) in cui sono integrate funzionalità quali:
 - IPS (sistema di rilevamento e prevenzione di intrusioni), che identifica i pattern noti degli attaccanti e li blocca.

Extra e conclusioni

- Web filtering: può bloccare l'accesso a URL presenti in database di URL malevoli noti
- Antivirus, analizza il payload del pacchetto per verificare la presenza di malware
- Ispezione di traffico SSL, i malware potrebbero sfruttare la crittografia SSL per nascondersi e sfuggire ai controlli. Questa funzionalità permette di decriptare il traffico crittografato, analizzarlo e crittografarlo nuovamente per inviarlo all'endpoint

Conclusioni

L'oggetto della Build Week 3 era l'analisi di un malware presente nella macchina virtuale dedicata. Tale analisi era strutturata su cinque giorni.

Nel corso della prima giornata, ci siamo dedicati all'analisi statica del malware, partendo dalle attività di base del Malware Analyst, come l'individuazione delle sezioni e delle librerie. Da queste prime riflessioni abbiamo potuto già formulare alcune ipotesi, ovvero che il malware andasse ad intervenire sulle chiavi di registro, che l'intervento avesse a che fare con **GINA.dll** e che vi fosse una parte di interazione con l'utente a causa della presenza della stringa **MessageBoxA**.

Il secondo giorno è proseguita l'analisi statica. Ci siamo focalizzati in particolare sulle funzioni con cui il malware interviene sul registro di Windows. Abbiamo notato che esso modifica una chiave di registro legata all'interfaccia di autenticazione per gli utenti che si connettono al sistema infetto, deducendo così che il malware tenta di intercettare le credenziali degli utenti legittimi.

Il terzo giorno abbiamo approfondito la parte del malware in cui quest'ultimo accede ad una propria risorsa ('TGAD') ed estrae un payload malevolo. Così, con l'aiuto di IDA Pro e OllyDBG, abbiamo potuto identificare la natura *dropper* del malware e illustrare le sue funzioni principali in un diagramma di flusso.

Il quarto giorno è stata la volta della vera e propria analisi dinamica del malware. L'analisi dei processi, delle modifiche ai registri, ma anche

Extra e conclusioni

l'osservazione della stessa cartella in cui era salvato l'eseguibile del malware ci hanno permesso di validare le nostre ipotesi: il malware effettivamente crea la chiave di registro HKLM(Local Machine)\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDll, che punta alla libreria malevola **msgina32.dll**, anch'essa manipolata dal malware.

Il quinto giorno avevamo dunque tutti gli elementi per concludere che il malware ha due flussi di lavoro: da un lato interviene sul registro di Windows affinché la chiave dedicata al login degli utenti all'avvio del sistema punti ad una libreria malevola, dall'altro, grazie alla propria natura di dropper, carica in questa libreria malevola il payload contenuto nella risorsa 'TGAD'. Tutto ciò è stato riportato in un diagramma.

In conclusione, possiamo affermare che il malware ha il fine di catturare le credenziali degli utenti legittimi; cosa gravissima, che può portare all'accesso a dati sensibili, all'alterazione del flusso di autenticazione e in generale alla compromissione della sicurezza del sistema.

Per completare la nostra riflessione, anche se non richiesto, ci siamo chiesti dove il malware salvasse tali credenziali. Abbiamo quindi fatto ulteriori approfondimenti e trovato il file con i log delle credenziali degli utenti legittimi: msutil32.sys.