BUILD WEEK 3 – PROGETTO UNIT 3

Malware Analysis and Reverse Engineering in Practice

Alessandro Alari

Adrian Amarfi

Alessio Bartolocci

Giacomo De Conti

Gabriele Di giampietro

Domenico Faccilongo

Nicolas Piletti

MALWARE ANALYSIS

Giorno 1:

Con riferimento al file eseguibile Malware_Build_Week_U3, rispondere ai seguenti quesiti utilizzando i tool e le tecniche apprese nelle lezioni teoriche:

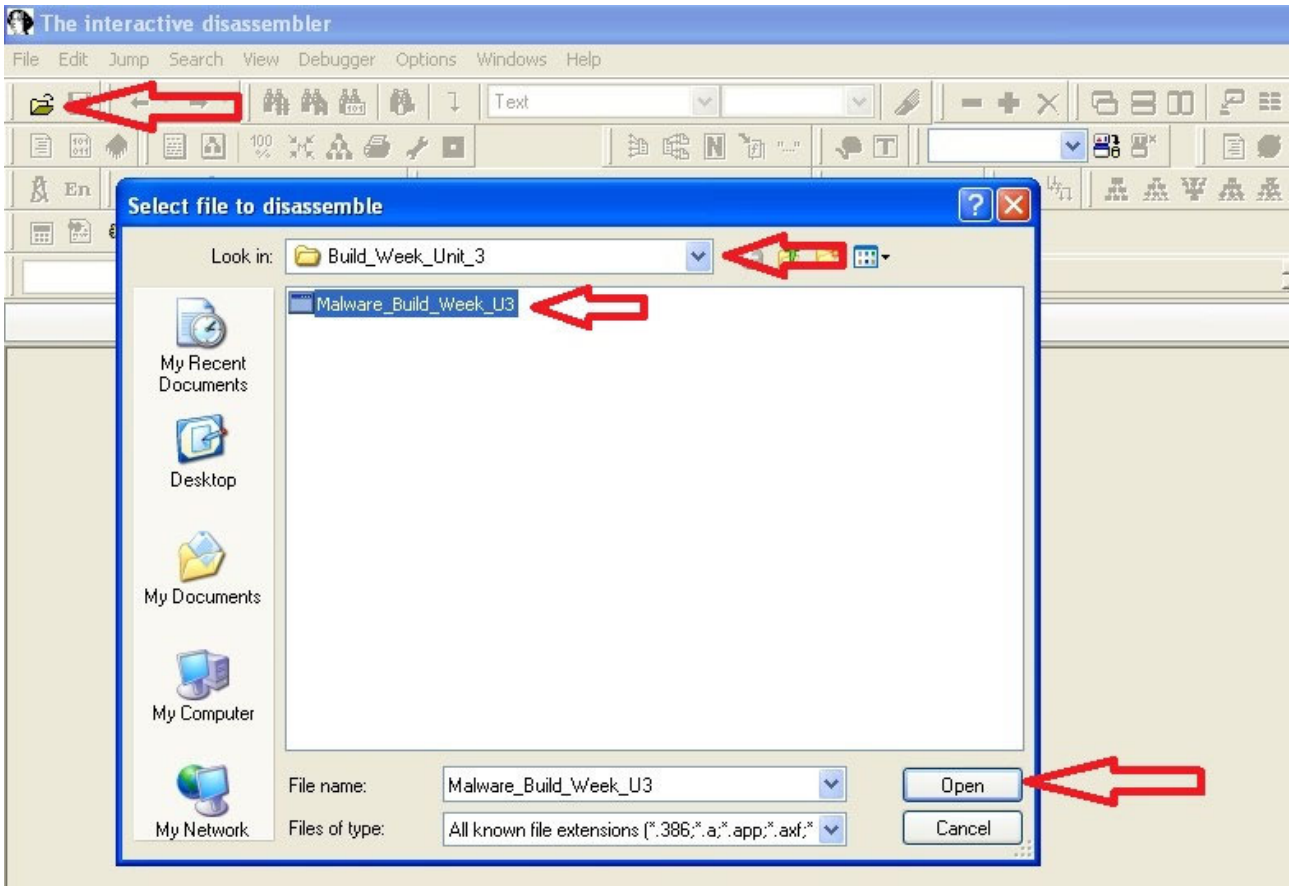
1. Quanti parametri sono passati alla funzione Main()?
2. Quante variabili sono dichiarate all'interno della funzione Main()?
3. Quali sezioni sono presenti all'interno del file eseguibile? Descrivete brevemente almeno 2 di quelle identificate
4. Quali librerie importa il Malware? Per ognuna delle librerie importate, fate delle ipotesi sulla base della sola analisi statica delle funzionalità che il Malware potrebbe implementare. Utilizzate le funzioni che sono richiamate all'interno delle librerie per supportare le vostre ipotesi.

Come da traccia siamo andati ad avviare la nostra macchina Windows XP per esaminare il nostro malware.

Per rispondere ai primi questi abbiamo dovuto recuperare il codice Assembly del malware, questo si può recuperare avendo a disposizione l'eseguibile e un disassembler che traduce le istruzioni da linguaggio macchina a linguaggio Assembly. Siamo quindi andati ad aprire il tool **IDA Pro** presente sulla nostra macchina.
IDA è un potente disassembler, tra i più utilizzati dagli analisti di sicurezza, che supporta molti formati di file eseguibili. IDA riesce a mettere a disposizione degli analisti una serie di caratteristiche intuitive per semplificare le attività. Infatti, oltre alla traduzione completa del linguaggio macchina di un eseguibile in linguaggio assembly, IDA identifica:

- Funzioni/chiamate di funzione;
- Analisi dello stack Variabili;
- Variabili locali e parametri.

Una volta avviato il tool siamo andati ad aprire il nostro malware cliccando l'icona della cartella e seguendo il path dell'eseguibile per poi cliccare su "Open".



Una volta aperto il malware, IDA Pro ci presenterà la sua interfaccia di analisi. In questa interfaccia nel pannello centrale troveremo il "Disassembly Panel" in modalità grafica, passeremo quindi alla modalità testuale cliccando sulla barra spaziatrice, qui è dove il tool ci mostrerà il codice Assembly dell'eseguibile, qui potremo identificare le funzioni (funzione Main() nel riquadro verde), le variabili locali (nel riquadro rosso) ed i parametri (nel riquadro giallo).

```
.text:004011D0 ; ||||||||||||||| S U B R O U T I N E |||||||||||||||||||||||||||||||||||||||
.text:004011D0
.text:004011D0 ; Attributes: bp-based frame
.text:004011D0
.text:004011D0 ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:004011D0 _main           proc near               ; CODE XREF: start+AF↓p
.text:004011D0 hModule         = dword ptr -11Ch
.text:004011D0 Data            = byte ptr -118h
.text:004011D0 var_8           = dword ptr -8
.text:004011D0 var_4           = dword ptr -4
.text:004011D0 argc            = dword ptr  8
.text:004011D0 argv            = dword ptr  0Ch
.text:004011D0 envp            = dword ptr  10h
.text:004011D0
```

1. **Quanti parametri sono passati alla funzione Main()?**
   IDA indica le variabili ed i parametri utilizzando come riferimento il puntatore EBP; i parametri si troveranno ad un offet positivo rispetto ad EBP, dove per offset si intende la differenza rispetto ad un valore di riferimento (EBP).

```
.text:004011D0 argc            = dword ptr  8
.text:004011D0 argv            = dword ptr  0Ch
.text:004011D0 envp            = dword ptr  10h
```

   Come potremo vedere dal riquadro in giallo troveremo **3 parametri** che verranno passati alla funzione:

a. **Argc:** (argument count) è un intero che contiene il numero di argomenti passati al programma.
b. **Argv:** (argument vector) è un array di puntatori di caratteri che contiene gli argomenti effettivi passati al programma. Ogni elemento dell'array è un puntatore a una stringa con terminazione null che rappresenta un argomento della riga di comando.
c. **Envp**: (environment variable pointer) è un array di puntatori di caratteri che contiene le variabili di ambiente che vengono passate al programma. Ogni elemento dell'array è un puntatore a una stringa con terminazione null nel formato "nome=valore" che rappresenta una variabile di ambiente.

Potremo trovare riscontro nella definizione della funzione Main() nel riquadro verde.

2. **Quante variabili sono dichiarate all'interno della funzione Main()?**

Come per i parametri IDA permetterà di identificare le funzioni locali rispetto al valore di riferimento EBP, queste infatti si troveranno ad un offset negativo rispetto ad EBP.

```
.text:004011D0 hModule          = dword ptr -11Ch
.text:004011D0 Data             = byte ptr -118h
.text:004011D0 var_8            = dword ptr -8
.text:004011D0 var_4            = dword ptr -4
```

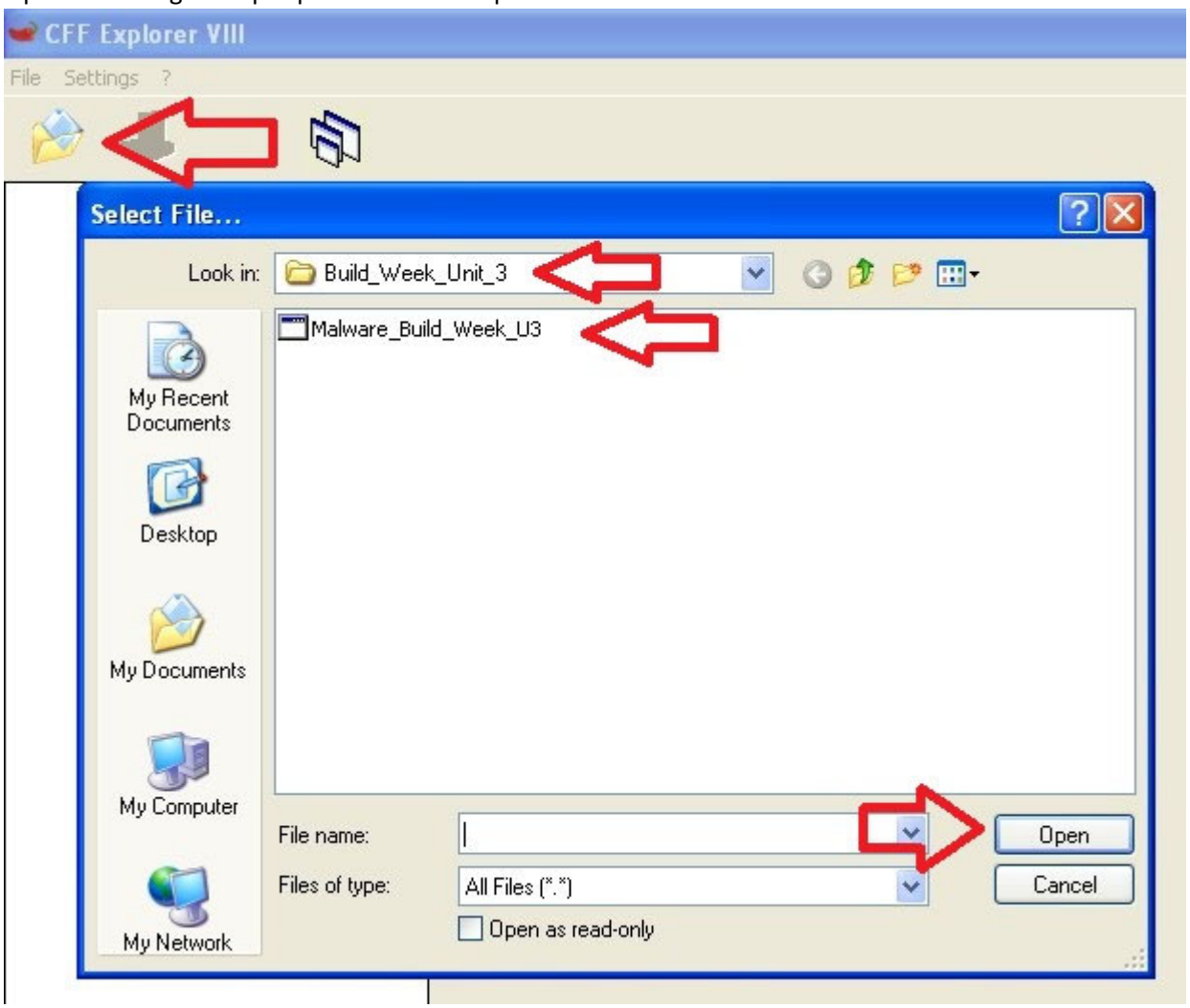
Come potremo vedere dal riquadro in rosso troveremo **4 variabili** dichiarate all'interno della funzione:

a. **hModule:** un tipo di variabile utilizzata in Windows per rappresentare un handle per un modulo. Un modulo è un file che contiene codice o dati che possono essere utilizzati da un programma. Un handle è un valore che può essere utilizzato per fare riferimento al modulo.
b. **Data:** variabile che potrebbe essere utilizzata per memorizzare qualsiasi tipo di dato.
c. **Var_8:** variabile che potrebbe essere utilizzata per memorizzare qualsiasi tipo di dato.
d. **Var_4:** variabile che potrebbe essere utilizzata per memorizzare qualsiasi tipo di dato.

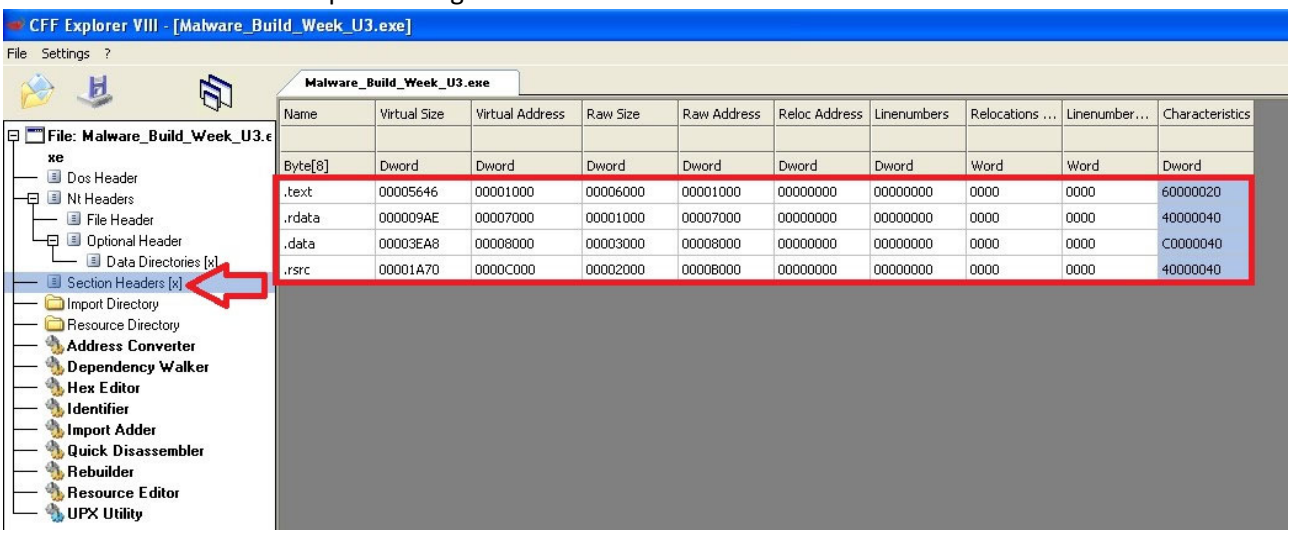
Per rispondere ai successivi quesiti siamo andati ad usufruire di un altro tool presente sulla nostra macchina Windows XP, **CFF Explorer**. Windows utilizza per la maggior parte dei file eseguibili il formato PE (Portable Executable), questo formato al suo interno contiene delle informazioni necessarie al sistema operativo per capire come gestire il codice del file, come ad esempio le librerie. Le informazioni circa le librerie e le funzioni richieste dall'eseguibile sono contenute nell'header del formato PE. Controllare quali sono le librerie e le funzioni importate ed esportate è fondamentale per capire lo scopo del malware.

Una volta avviato il tool siamo andati ad aprire il nostro malware cliccando l'icona della cartella e seguendo

il path dell'eseguibile per poi cliccare su "Open".



3. **Quali sezioni sono presenti all'interno del file eseguibile? Descrivere due di quelle identificate.**
   Una volta aperto il malware sul nostro tool CFF Explorer, dovremo spostarci nel pannello di sinistra cliccando su "Section Headers [x]", il pannello a destra andrà quindi a mostrarci le informazioni circa le sezioni di cui si compone l'eseguibile.



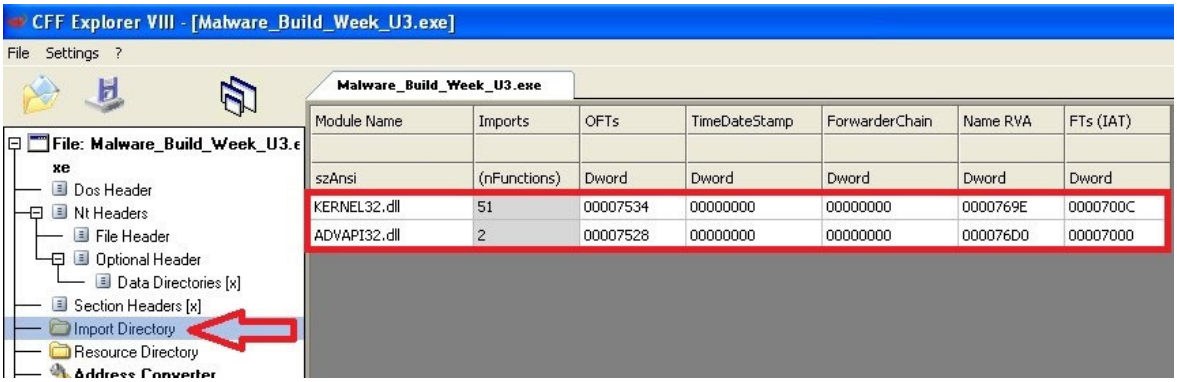
   Troveremo quindi **quattro sezioni** nelle quali si compone il software. Ogni sezione ha un preciso scopo:

a. **.text:** contiene le istruzioni (le righe di codice) che la CPU eseguirà una volta che il software sarà avviato. Generalmente questa è l'unica sezione di un file eseguibile che viene eseguita dalla CPU, in quanto tutte le altre sezioni contengono dati o informazioni a supporto.
b. **.rdata:** include generalmente le informazioni circa le librerie e le funzioni importate ed esportate dall'eseguibile, informazione che come abbiamo visto possiamo ricavare con CFF Explorer.
c. **.data:** contiene tipicamente i dati / le variabili globali del programma eseguibile, che devono essere disponibili da qualsiasi parte del programma. Ricordate che una variabile si dice globale quando non è definita all'interno di un contesto di una funzione, ma bensì è globalmente dichiarata ed è di conseguenza accessibile da qualsiasi funzione dell'eseguibile.
d. **.rsrc**: include le risorse utilizzate dall'eseguibile come ad esempio icone, immagini, menu e stringhe che non sono parte dell'eseguibile stesso.

CFF Explorer riporta non solo il nome delle sezioni ma anche altre importanti informazioni, come Virtual Size, che indica lo spazio allocato per la sezione durante il processo di caricamento dell'eseguibile in memoria, e Raw Size, che indica lo spazio occupato dalla sezione quando è sul disco.

4. **Quali librerie importa il Malware? Fare delle ipotesi sulle funzionalità del malware in base alle librerie importate.**

Per controllare le librerie e le funzioni importate, andremo a spostarci su "import directory" nel menù a sinistra.



Il pannello a destra mostrerà informazioni sulle librerie importate dall'eseguibile:

a. **KERNEL32.dll:** è una libreria di sistema di Windows che fornisce una serie di funzioni di base per la gestione dei processi, thread, memoria, file, eccezioni e risorse. È una delle librerie di sistema più importanti di Windows e viene utilizzata da molti programmi per accedere alle funzionalità di base del sistema operativo. È presente in tutte le versioni di Windows.
b. **ADVAPI32.dll:** è una libreria di sistema di Windows che fornisce funzioni di sicurezza e gestione delle autorizzazioni, utilizzate da molte applicazioni Windows per accedere alle funzionalità di sicurezza del sistema operativo, come la creazione e gestione degli account utente e delle autorizzazioni, la crittografia dei dati e la gestione dei certificati digitali. È presente in tutte le versioni di Windows.

Mentre per ognuna delle librerie, il pannello inferiore mostrerà una lista delle funzioni richieste all'interno della libreria selezionata.

Per KERNEL32.dll troveremo 51 funzioni:

Malware_Build_Week_U3.exe

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
0000769E	N/A	000074EC	000074F0	000074F4	000074F8	000074FC
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	51	00007534	00000000	00000000	0000769E	0000700C
ADVAPI32.dll	2	00007528	00000000	00000000	000076D0	00007000

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00007632	00007632	0295	SizeofResource
00007644	00007644	01D5	LockResource
00007654	00007654	01C7	LoadResource
00007622	00007622	02BB	VirtualAlloc
00007674	00007674	0124	GetModuleFileNameA
0000768A	0000768A	0126	GetModuleHandleA
00007612	00007612	00B6	FreeResource
00007664	00007664	00A3	FindResourceA
00007604	00007604	001B	CloseHandle
000076DE	000076DE	00CA	GetCommandLineA
000076F0	000076F0	0174	GetVersion
000076FE	000076FE	007D	ExitProcess
0000770C	0000770C	019F	HeapFree
00007718	00007718	011A	GetLastError
00007728	00007728	02DF	WriteFile
00007734	00007734	029E	TerminateProcess
00007748	00007748	00F7	GetCurrentProcess
0000775C	0000775C	02AD	UnhandledExceptionFilter
00007778	00007778	00B2	FreeEnvironmentStringsA
00007792	00007792	00B3	FreeEnvironmentStringsW
000077AC	000077AC	02D2	WideCharToMultiByte
000077C2	000077C2	0106	GetEnvironmentStrings
000077DA	000077DA	0108	GetEnvironmentStringsW
000077F4	000077F4	026D	SetHandleCount
00007806	00007806	0152	GetStdHandle
00007816	00007816	0115	GetFileType
00007824	00007824	0150	GetStartupInfoA
00007836	00007836	0109	GetEnvironmentVariableA
00007850	00007850	0175	GetVersionExA

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00007860	00007860	019D	HeapDestroy
0000786E	0000786E	019B	HeapCreate
0000787C	0000787C	02BF	VirtualFree
0000788A	0000788A	022F	RtlUnwind
00007896	00007896	0199	HeapAlloc
000078A2	000078A2	01A2	HeapReAlloc
000078B0	000078B0	027C	SetStdHandle
000078C0	000078C0	00AA	FlushFileBuffers
000078D4	000078D4	026A	SetFilePointer
000078E6	000078E6	0034	CreateFileA
000078F4	000078F4	00BF	GetCPInfo
00007900	00007900	00B9	GetACP
0000790A	0000790A	0131	GetOEMCP
00007916	00007916	013E	GetProcAddress
00007928	00007928	01C2	LoadLibraryA
00007938	00007938	0261	SetEndOfFile
00007948	00007948	0218	ReadFile
00007954	00007954	01E4	MultiByteToWideChar
0000796A	0000796A	01BF	LCMapStringA
0000797A	0000797A	01C0	LCMapStringW
0000798A	0000798A	0153	GetStringTypeA
0000799C	0000799C	0156	GetStringTypeW

Mentre per ADVAPI32.dll troveremo due funzioni:

CFF Explorer VIII - [Malware_Build_Week_U3.exe]

File Settings ?

File: Malware_Build_Week_U3.exe
- Dos Header
- Nt Headers
  - File Header
  - Optional Header
    - Data Directories [x]
- Section Headers [x]
- Import Directory
- Resource Directory
- Address Converter
- Dependency Walker
- Hex Editor

Malware_Build_Week_U3.exe

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
000076D0	N/A	00007500	00007504	00007508	0000750C	00007510
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	51	00007534	00000000	00000000	0000769E	0000700C
ADVAPI32.dll	2	00007528	00000000	00000000	000076D0	00007000

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
000076AC	000076AC	0186	RegSetValueExA
000076BE	000076BE	015F	RegCreateKeyExA

Tra le funzioni caricate dalle librerie potremo trovare funzioni come

c. **LoadLibrary** e **GetProcAddress**: utilizzate per caricare funzioni addizionali durante l'esecuzione.
d. **RegSetValueEx**: funzione che permette di aggiungere un nuovo valore all'interno del registro e di settare i rispettivi dati. Accetta come parametri la chiave, la sottochiave e il dato da inserire
e. **SizeOfResource, LockResource, LoadResource** e **FindResource**: APIs che permettono di localizzare all'interno della sezione "risorse" il malware da estrarre, e successivamente da caricare in memoria per l'esecuzione o da salvare sul disco per esecuzione futura. APIs

caratteristiche dei malware appartenenti alla famiglia dei **Dropper**, programmi malevoli che contengono al loro interno un malware. Nel momento in cui vengono eseguiti, i dropper iniziano la loro esecuzione ed estraggono il malware che contengono per salvarlo sul disco.

f. **CreateFile** e **WriteFile**: usate rispettivamente per creare un file vuoto e scrivere al suo interno il malware appena estratto, in questo scenario il dropper utilizzerà questa coppia di funzioni per il salvataggio del malware sul disco per eventuale utilizzo futuro.

Se il malware importa queste funzioni, potrebbe utilizzarle per manipolare le risorse di sistema, caricare librerie e accedere al registro per eseguire azioni dannose come il furto di informazioni o l'installazione di malware aggiuntivo.
Un dropper prima utilizza il set di APIs di Windows per estrarre il malware contenuto nel segmento risorse. Successivamente può creare un processo per eseguire immediatamente il malware, oppure salvare il malware sul disco, e magari eseguirlo in un secondo momento.

Giorno 2:

Con riferimento al Malware in analisi, spiegare:

1. Lo scopo della funzione chiamata alla locazione di memoria 00401021;
2. Come vengono passati i parametri alla funzione alla locazione 00401021;
3. Che oggetto rappresenta il parametro alla locazione 00401017;
4. Il significato delle istruzioni comprese tra gli indirizzi 00401027 e 00401029;
5. Con riferimento all'ultimo quesito, tradurre il codice Assembly nel corrispondente costrutto C;
6. Valutate ora la chiamata alla locazione 00401047, qual è il valore del parametro "ValueName"?

Nel complesso delle due funzionalità appena viste, spiegate quale funzionalità sta implementando il Malware in questa sezione.

Andiamo ad effettuare un ulteriore analisi statica avanzata del Malware che ci è stato dato. Per fare ciò utilizziamo il tool **IDA pro**, un **disassembler** che come abbiamo visto nel giorno 1 ci permette di analizzare il codice tradotto dal linguaggio macchina in assembly.

1. **Scopo della funzione alla locazione di memoria 00401021.**
   Nel primo punto della nostra analisi troviamo una chiamata di funzione alla locazione di memoria 00401021, con chiamata a **RegCreateKeyExA**.

```
.text:00401017                 push    offset SubKey   ; "SOFTWA
.text:0040101C                 push    80000002h       ; hKey
.text:00401021                 call    ds:RegCreateKeyExA
.text:00401027                 test    eax, eax
```

   RegCreateKeyEx è una funzione API di Windows che viene utilizzata per creare una nuova chiave nel registro di Windows o aprire una chiave esistente. Il registro di Windows è un database gerarchico che memorizza le impostazioni di configurazione e le opzioni per il sistema operativo e il software installato. Questa funzione viene spesso utilizzata da software legittimo per creare nuove chiavi o modificare quelle esistenti per memorizzare impostazioni o dati di configurazione.

   Tuttavia, il malware può anche utilizzare questa funzione per creare nuove chiavi o modificare quelle esistenti nel registro di Windows per eseguire azioni dannose come: Nascondere la sua presenza aggiungendo nuove chiavi al Registro, Impostando la sua persistenza aggiungendo nuove

chiavi che lanciano il malware ogni volta che si avvia il sistema. Modificare le chiavi esistenti per cambiare il comportamento del sistema operativo o del software installato, o rubare informazioni sensibili aggiungendo nuove chiavi al registro che memorizzano i dati rubati

È importante notare che la sola presenza della funzione RegCreateKeyEx non indica che un file è dannoso, in quanto è comunemente utilizzato anche da software legittimo. Per determinare se un file è o meno un Malware, sarebbe necessaria un'ulteriore analisi del comportamento e del codice del file.

2. **Come vengono passati i parametri alla funzione nella locazione 00401021?**
   Riguardo alla funzione RegCreateKeyEx, possiamo trovare i parametri che vengo passati nello Stack per la chiamata di funzione nelle istruzioni sopra. Partiamo dall'istruzione **Push** che mette in cima allo Stack il registro ebp che poi successivamente l'istruzione **mov** copia il registro esp nel registro epb, dopodiché le seguenti istruzioni Push inserisco nello Stack il registro ecx e il parametro 0. Passiamo poi all'istruzione **lea** che copia la variabile locale **hObject** nel registro **eax,** successivamente l'istruzione **Push** continua ad insererire nello Stack e ne passa sopra il registro eax, il codice continua poi sempre attraverso le istruzioni Push successive ad inserire tutti i parametri, come possiamo vedere nella frazione di codice seguente.

```
.text:00401000    push    ebp
.text:00401001    mov     ebp, esp
.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
```

3. **Che oggetto rappresenta il parametro alla locazione 00401017?**
   Nell'allocazione di memoria 00401017 troviamo che il Malware utilizza una chiave di registro per ottenere la persistenza, seguendo il path che viene inserito come parametro, "**Software\Microsoft\Windows NT\Current Versione\ Winlogon".**

   Winlogon (Windows LogOn Process), conosciuto anche come **winlogon.exe** nel task manager, è un processo di sistema di Microsoft Windows che gestisce il logon interattivo e altri aspetti, come l'autenticazione delle credenziali all'utente e altri startup. Winlogon assicura la sicurezza e la stabilità del sistema operativo, quindi non deve essere terminato. Il processo si occupa anche di avviare la sessione utente e la shell. Inoltre, tramite il SAS (Security Attention Sequence), impedisce ai trojan di recuperare informazioni sensibili sull'utente. Talvolta però come nel nostro caso, capita che winlogon.exe in realtà sia un virus: alcuni virus, infatti, si camuffano da winlogon.exe per non essere individuati.

```
text:00401017    push    offset SubKey    ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"...
```

```
db 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon',0
                                ; DATA XREF: sub_401000+17To
```

4. **Il significato delle istruzioni comprese tra gli indirizzi 00401027 e 00401029.**
   Le istruzioni comprese tra gli indirizzi 00401027 e 00401029 forniscono le condizioni per un salto condizionale. Con **l'istruzione test**, il codice simula un **AND logico** tra destinazione e sorgente lasciandoli però inalterati; questa operazione però basta per modificare i flag e favorire i conseguenti salti condizionali.

   Con **jz (jump zero),** si effettuerà il salto condizionale alla locazione di memoria specificata solo se la zero flag verrà settata ad 1, ovvero se l'operazione logica precedente restituisce 0 come risultato.

   AND logico: operatore logico che funge da congiunzione tra due proposizioni creandone una terza che sarà vera solo quando le prime 2 sono vere.

A	B	A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

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

5. **Con riferimento al punto 4, tradurre il codice Assembly nel corrispondente costrutto C.**
   Guardando la porzione di codice analizzata nel punto 4, sembrerebbe si tratti di un costrutto del **ciclo IF**, si può dedurre dal fatto che l'istruzione test è usata similmente all'istruzione cmp, che in questo caso serve per controllare se il contenuto del registro EAX è uguale a 0; con l'istruzione successiva, **jz short loc_401032**, si salterà alla locazione specificata solamente se la ZF (Zero Flag) è settata ad 1.

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

6. **Valutate ora la chiamata alla locazione 00401047, quale è il valore del parametro "ValueName"?**
   Sull'allocazione di memoria 00401047 troviamo la chiamata di funzione **RegSetValueExA,** una funzione API di Windows che imposta i dati e il tipo di un valore specificato in una chiave del registro di sistema. Possiamo notare i parametri che sono passati sullo Stack nelle istruzioni precedenti per la chiamata di funzione, nello specifico nell'allocazione di memoria 0040103E il valore del parametro di **ValueName** è **GinaDLL.**

   GinaDLL: libreria che viene caricata durante l'avvio del processo **Winlogon**, viene usata tra le altre cose per fornire procedure di identificazione e autenticazione dell'utente ed esegue questa operazione delegando il monitoraggio degli eventi SAS a Winlogon; è responsabile dell'impostazione della ricezione degli eventi SAS e della notifica a Winlogon quando questi si verificano. Inoltre GinaDLL gestisce la **Gui (Interfaccia grafica) di Windows.**

```
.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
```

Nella analisi che abbiamo appena svolto abbiamo trovato due funzionalità principali implementate dal Malware:

La funzione **RegCreateKeyExA** che viene usata dal malware per creare una chiave di registro specificata oppure aprirne una già esistente, mentre **RegSetValueExA** aggiunge un nuovo valore all'interno del registro e imposta i rispettivi dati. Queste 2 funzioni vengono usate solitamente dai software malevoli **per ottenere persistenza**, cioè l'esecuzione automatica e permanente dell'eseguibile all'avvio del PC senza che l'utente faccia nessuna azione.

Giorno 3:

Riprendete l'analisi del codice, analizzando le routine tra le locazioni di memoria 00401080 e 00401128:

1. Qual è il valore del parametro "ResourceName" passato alla funzione FindResourceA();
2. Il susseguirsi delle chiamate di funzione che effettua il Malware in questa sezione di codice l'abbiamo visto durante le lezioni teoriche. Che funzionalità sta implementando il Malware?
3. È possibile identificare questa funzionalità utilizzando l'analisi statica basica?
4. In caso di risposta affermativa, elencare le evidenze a supporto.

Entrambe le funzionalità principali del Malware viste finora sono richiamate all'interno della funzione Main().

5. Disegnare un diagramma di flusso (inserite all'interno dei box solo le informazioni circa le funzionalità principali) che comprenda le 3 funzioni.

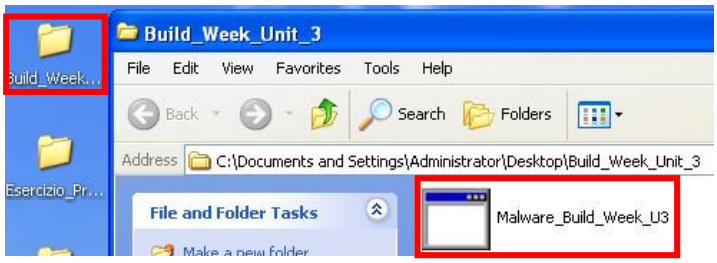
Al fine di rispondere ai quesiti andiamo ad utilizzare i seguenti tool:

- **OLLYDBG**: Debugger con diverse funzionalità: tracciare i registri, riconoscere le funzioni e i parametri passati alle principali librerie standard, le chiamate alle API del sistema operativo, salti condizionali, ecc. Fondamentale per l'Analisi Dinamica Avanzata.

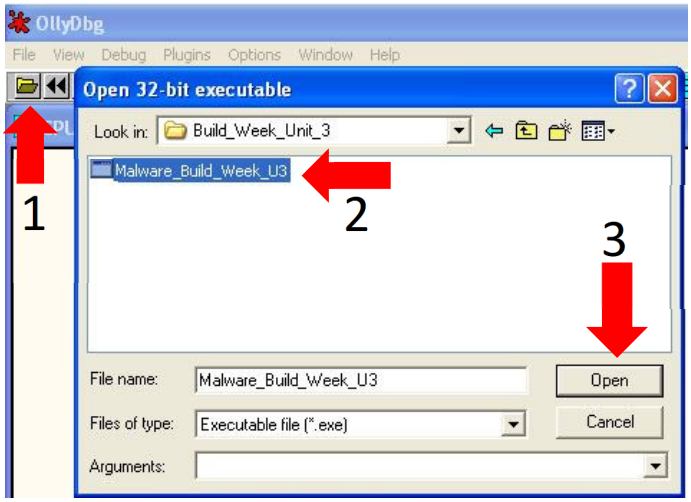
- **IDA PRO**: Disassembler largamente usato per il reverse engineering. Necessario per l'Analisi Statica Avanzata.

- **CFF EXPLORER**: Tool che include una serie di sezioni, fondamentale per lo studio di header del formato PE di un malware. Fondamentale per l'Analisi Statica Basica.

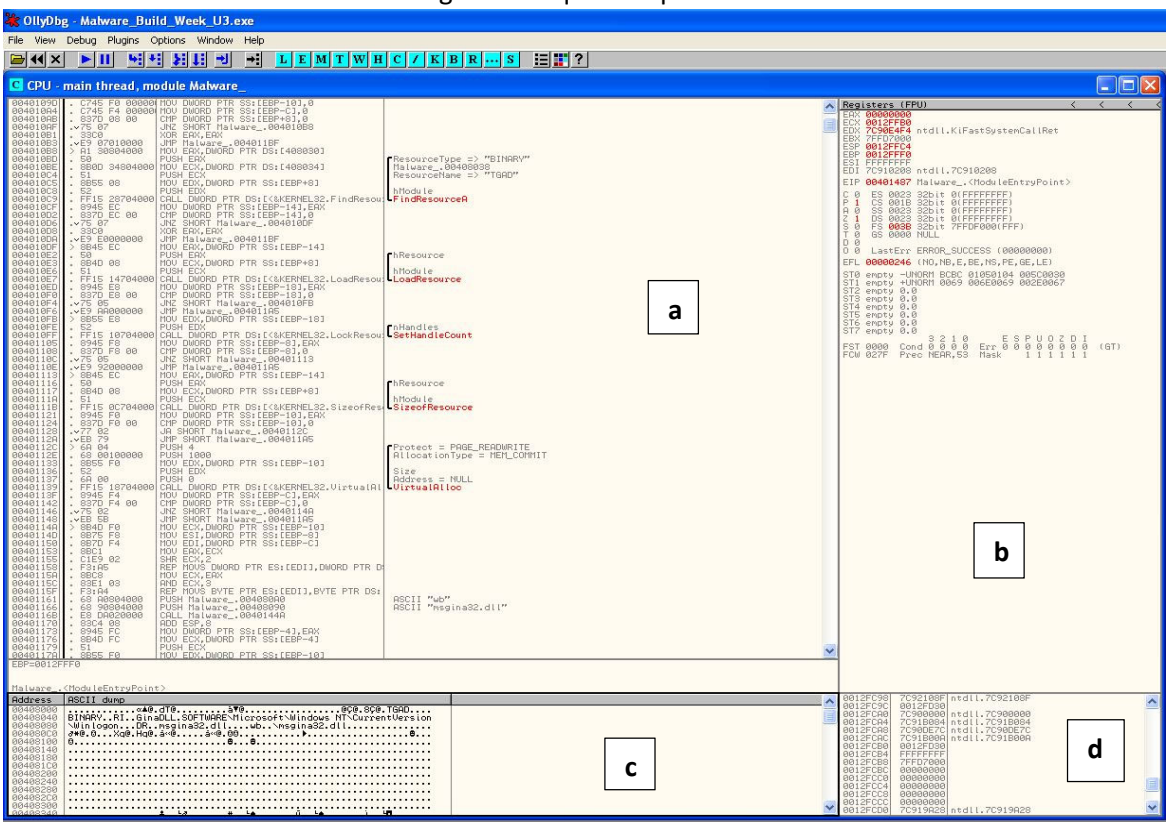
1. **Qual è il valore del parametro "ResourceName" passato alla funzione FindResourceA()?**
   La locazione del malware, come visto precedentemente, è sul Desktop, nella cartella *Build_Week_Unit_3*, come indicato in figura:



In tal caso andiamo ad utilizzare OllyDBG per la risoluzione del quesito. All'apertura del tool selezioniamo il file eseguibile, cliccando sull'icona come mostrato in figura:



Una volta caricato il file avremo una grafica di questo tipo:



Dove:

a. **Disassembler Window**: Una schermata che contiene le istruzioni che sono eseguite dalla CPU, con aggiunta di alcuni commenti inseriti direttamente da OllyDBG in base al suo DB.

b. **Register Window**: Schermata che riporta lo stato dei registri e del loro valore al momento del breakpoint. Ogni qual volta che un valore cambia, esso viene evidenziato in rosso in figura.
c. **Memory Dump Window**: In questa sezione troviamo il contenuto degli indirizzi di memoria del programma in esecuzione.
d. **Stack Window:** Esso mostra lo stato attuale dello stack in memoria. Anche in questo caso OllyDBG aggiunge dei commenti in modo da supportare la comprensione del file eseguibile e il suo comportamento.

Al fine di identificare il valore del parametro **ResourceName**, andiamo nella sezione 1. e dirigiamoci all'intervallo della locazione di memoria indicato, ovvero 00401080 – 00401128. Possiamo infatti notare, come immagine seguente, che il valore del parametro indicato è **TGAD:**

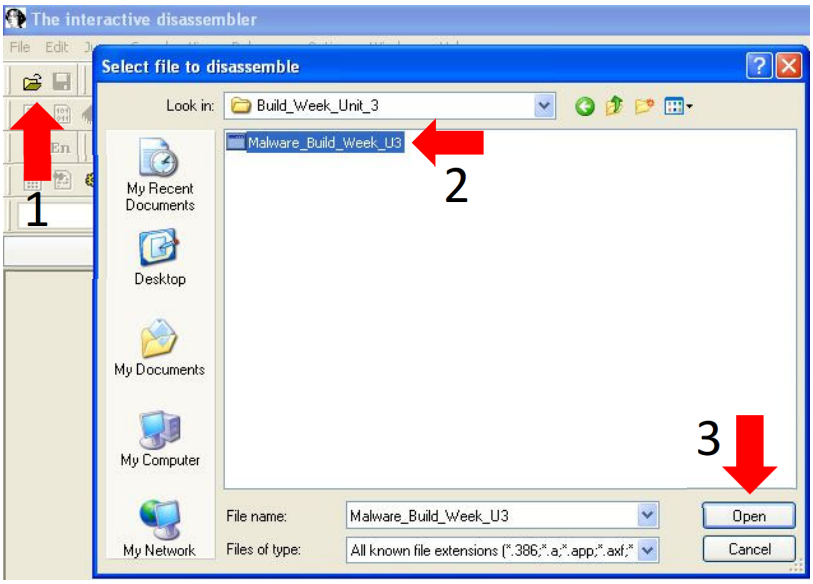
```
004010B1  . 33C0          XOR EAX,EAX
004010B3  .v E9 07010000  JMP Malware_.004011BF
004010B8  > A1 30804000   MOV EAX,DWORD PTR DS:[408030]
004010BD  . 50            PUSH EAX
004010BE  . 8B0D 34804000 MOV ECX,DWORD PTR DS:[408034]
004010C4  . 51            PUSH ECX
004010C5  . 8B55 08       MOV EDX,DWORD PTR SS:[EBP+8]
004010C8  . 52            PUSH EDX
004010C9  . FF15 28704000 CALL DWORD PTR DS:[<&KERNEL32.FindResou
004010CF  . 8945 EC       MOV DWORD PTR SS:[EBP-14],EAX
004010D2  . 837D EC 00    CMP DWORD PTR SS:[EBP-14],0
004010D6  .v 75 07        JNZ SHORT Malware_.004010DF

ResourceType => "BINARY"
ResourceName => "TGAD"
hModule
FindResourceA
```

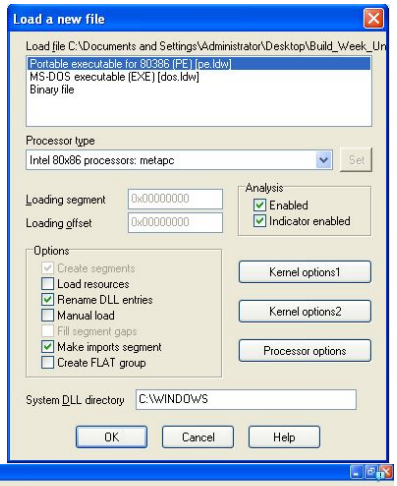
È possibile che il parametro TGAD sia un nome di variabile personalizzato o un parametro utilizzato in un programma o progetto specifico. Senza ulteriore contesto, non è possibile determinare la sua funzionalità specifica o lo scopo del parametro.

2. **Che funzionalità sta implementando il Malware?**

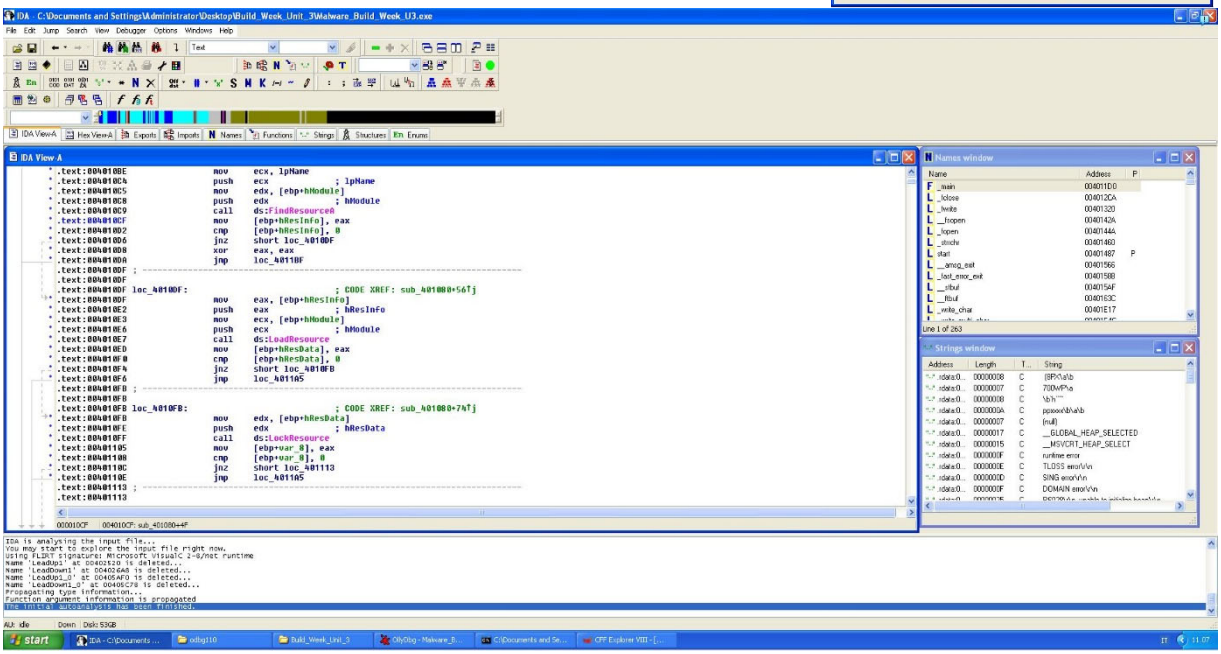
Al fine di rispondere al quesito andiamo innanzitutto ad elencare quali sono le chiamate di funzione che ritroviamo in questo intervallo di locazioni di memoria, e per farlo utilizziamo il tool **IDA PRO**, da cui, all'apertura andiamo a selezionare il malware interessato:



All’apertura del file ci verrà presentato un’interfaccia del tool che ci permette di andare a modificare i criteri di lettura del file eseguibile. In tal caso lasciamo le caratteristiche che ritroviamo e proseguiamo all’analisi del quesito cliccando su OK:

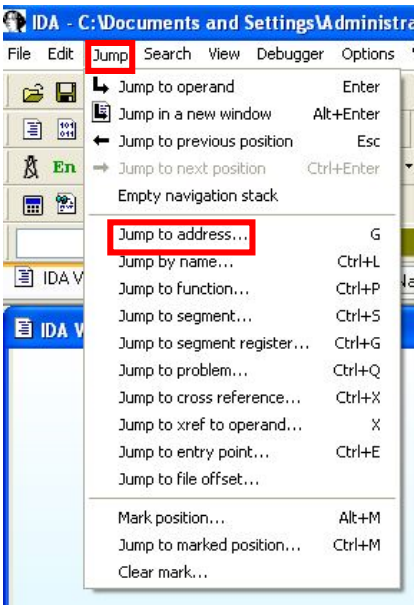


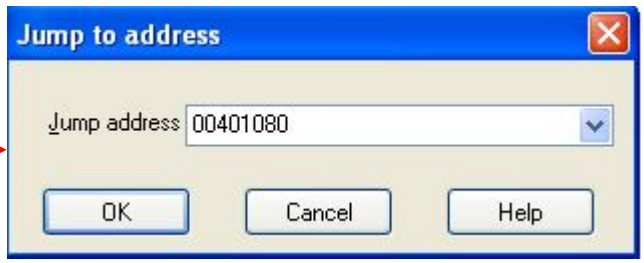
Successivamente avremo un’interfaccia grafica di questo tipo:



La sezione interessata è IDA View-A, dal quale possiamo identificare le chiamate di funzioni note nell’intervallo di memoria 00401080 – 00401128, come da task.

Per andare più velocemente alla locazione interessata utilizziamo il comando *jump to address* dalla sezione *jump* e nella tabella di testo inseriamo la locazione:





Da qui andiamo ad evidenziare le chiamate di funzioni note:

```
.text:004010C8                 push    edx             ; hModule
.text:004010C9                 call    ds:FindResourceA
.text:004010CF                 mov     [ebp+hResInfo], eax
.text:004010D2                 cmp     [ebp+hResInfo], 0
.text:004010D6                 jnz     short loc_4010DF
.text:004010D8                 xor     eax, eax
.text:004010DA                 jmp     loc_4011BF
.text:004010DF ; ---------------------------------------------------------------------------
.text:004010DF
.text:004010DF loc_4010DF:                             ; CODE XREF: sub_401080+56↑j
.text:004010DF                 mov     eax, [ebp+hResInfo]
.text:004010E2                 push    eax             ; hResInfo
.text:004010E3                 mov     ecx, [ebp+hModule]
.text:004010E6                 push    ecx             ; 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 ; ---------------------------------------------------------------------------
.text:004010FB
.text:004010FB loc_4010FB:                             ; CODE XREF: sub_401080+74↑j
.text:004010FB                 mov     edx, [ebp+hResData]
.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 ; ---------------------------------------------------------------------------
.text:00401113
.text:00401113 loc_401113:                             ; CODE XREF: sub_401080+8C↑j
.text:00401113                 mov     eax, [ebp+hResInfo]
.text:00401116                 push    eax             ; hResInfo
.text:00401117                 mov     ecx, [ebp+hModule]
.text:0040111A                 push    ecx             ; hModule
.text:0040111B                 call    ds:SizeofResource
```

Il costrutto in figura ci permette di intuire che il file eseguibile altro non è che un programma malevolo denominato **Dropper** il quale, nel momento in cui viene eseguito, inizia la sua esecuzione ed estrae il malware che contiene per salvarlo sul disco. Al fine di estrarre il malware, il **Dropper** utilizza le APIs che abbiamo identificato:

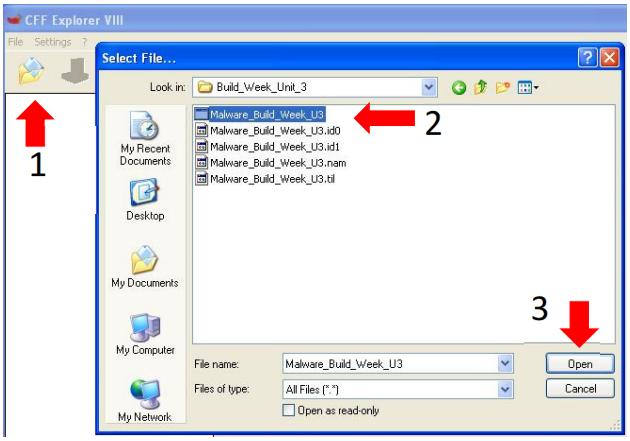
- **FindResource():** Determina la posizione di una risorsa con il tipo e il nome specificato nel modulo.
- **LoadResource():** Recupera un handle che può essere usato per ottenere un puntatore al primo byte della risorsa specificata in memoria.
- **LockResource():** Recupera un puntatore alla risorsa specificata in memoria.
- **SizeOfResource():** Recupera la dimensione, in byte, della risorsa specificata.

In definitiva, queste APIs permettono di localizzare all'interno della sezione "risorse" il malware da estrarre, e successivamente da caricare in memoria per l'esecuzione (creazione di un processo) o da salvare sul disco per esecuzione futura (utilizzando la coppia di funzioni **CreateFile()** e **WriteFile()**.

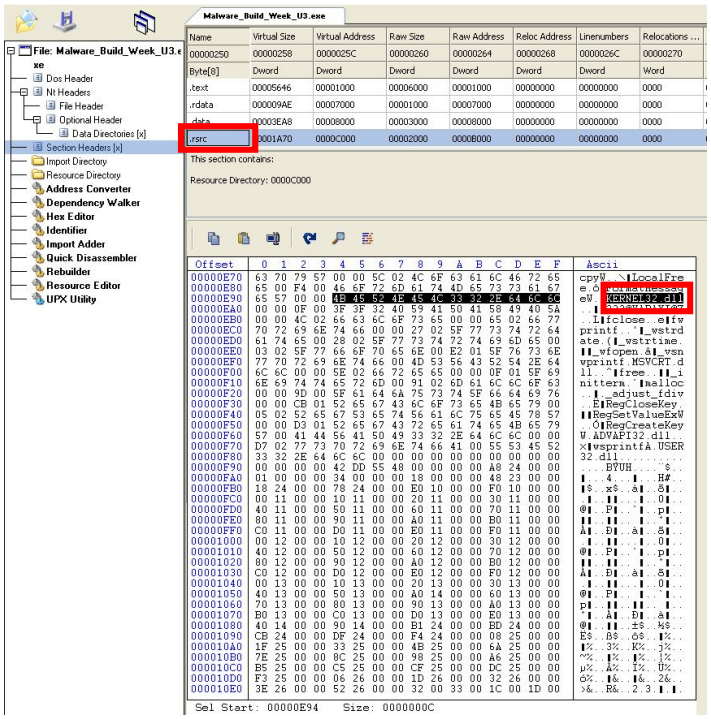
3. **È possibile identificare questa funzionalità utilizzando l'analisi statica basica?**

La risposta a questo quesito è Si è possibile identificare la funzionalità e per farlo utilizziamo il tool **CFF Explorer.**

4. **Elencare le evidenze a supporto:**
Come abbiamo detto in precedenza utilizziamo il tool CFF per identificare il costrutto. All'apertura del programma, selezioniamo il simbolo della cartella per caricare il file a noi interessato, come mostrato in figura:



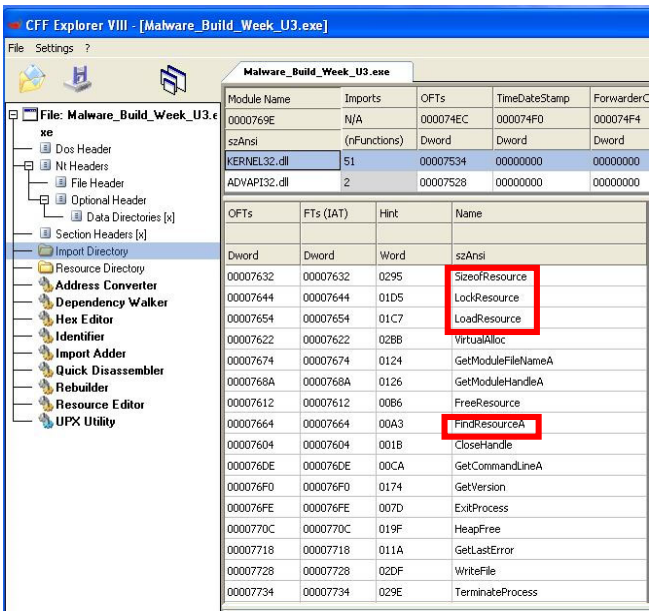
Successivamente dirigiamoci nel riquadro a sinistra, dove troviamo le sezioni necessarie per analizzare il file eseguibile, precisamente nella sezione **Section Headers [x]** da cui vengono evidenziate le sezioni importate dal malware**:**



Generalmente il malware incluso nel **Dropper** è contenuto nella sezione ".rsrc", ovvero sezione risorse, in cui al suo interno si hanno delle istruzioni (**obj** che permettono al compilatore di eseguirlo, anche se non è in grado di elaborarlo realmente).

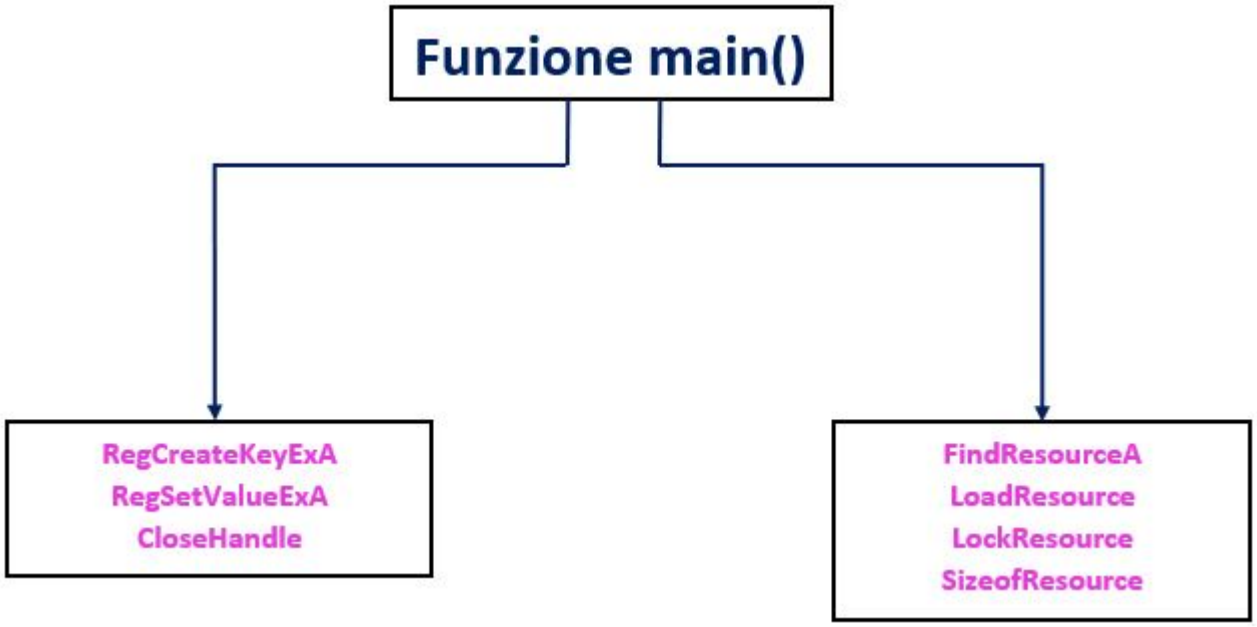
Aprendo la sezione dedicata, noteremmo dall'ASCII (codice per la codifica di caratteri) che il testo è criptato; ma, come evidenziato in figura, ritroviamo la libreria Kernel32.dll che potrebbe esserci utile.

Infatti, andiamo nella sezione **Import Directory** da cui vengono evidenziate le librerie importate dal malware, e apriamo rispettivamente Kernel32**:**



Come possiamo vedere abbiamo ritrovato le stesse funzioni indicate precedentemente, tutte presenti nella libreria Kernell32.dll. Una libreria che contiene le funzioni principali per interagire con il sistema operativo, come ad esempio la manipolazione dei file e gestione della memoria, cosa che il programma effettua al fine di estrarre il malware che dovrà poi salvare su disco o eseguire.

5. **Diagramma di flusso:**
   Entrambe le funzionalità principali del Malware viste finora sono richiamate all'interno della funzione Main(). Disegnare un diagramma di flusso che comprenda le 3 funzioni.



Giorno 4:

Preparate l'ambiente ed i tool per l'esecuzione del Malware (suggerimento: avviate principalmente Process Monitor ed assicurate di eliminare ogni filtro cliccando sul tasto "reset" quando richiesto in fase di avvio). Eseguite il Malware, facendo doppio click sull'icona dell'eseguibile.

1. Cosa notate all'interno della cartella dove è situato l'eseguibile del Malware? Spiegate cosa è avvenuto, unendo le evidenze che avete raccolto finora per rispondere alla domanda

   Analizzate ora i risultati di Process Monitor (consiglio: utilizzate il filtro come in figura sotto per estrarre solo le modifiche apportate al sistema da parte del Malware). Fate click su "ADD" poi su "Apply" come abbiamo visto nella lezione teorica.

Filtrate includendo solamente l'attività sul registro di Windows.

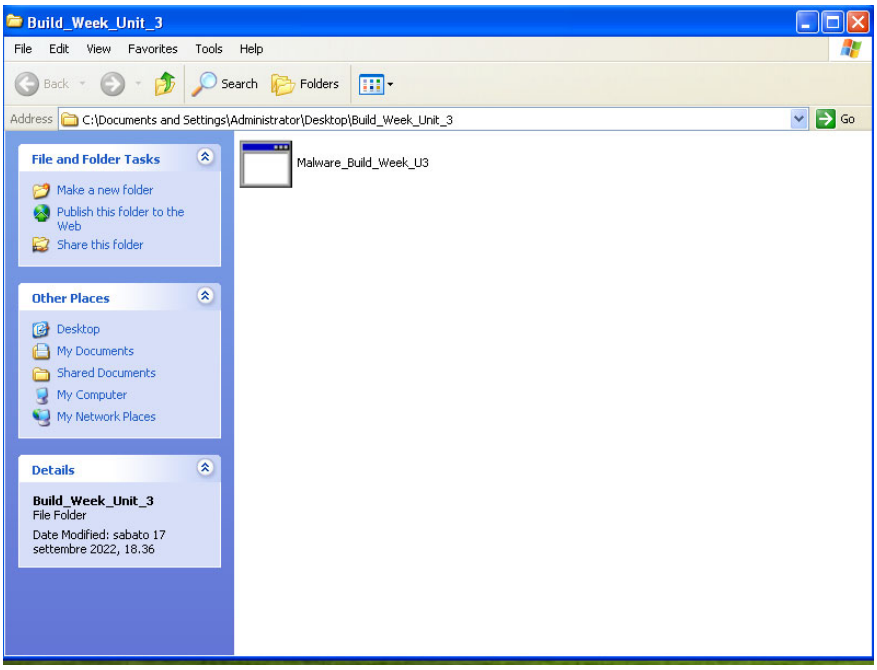
2. Quale chiave di registro viene creata?
3. Quale valore viene associato alla chiave di registro creata?

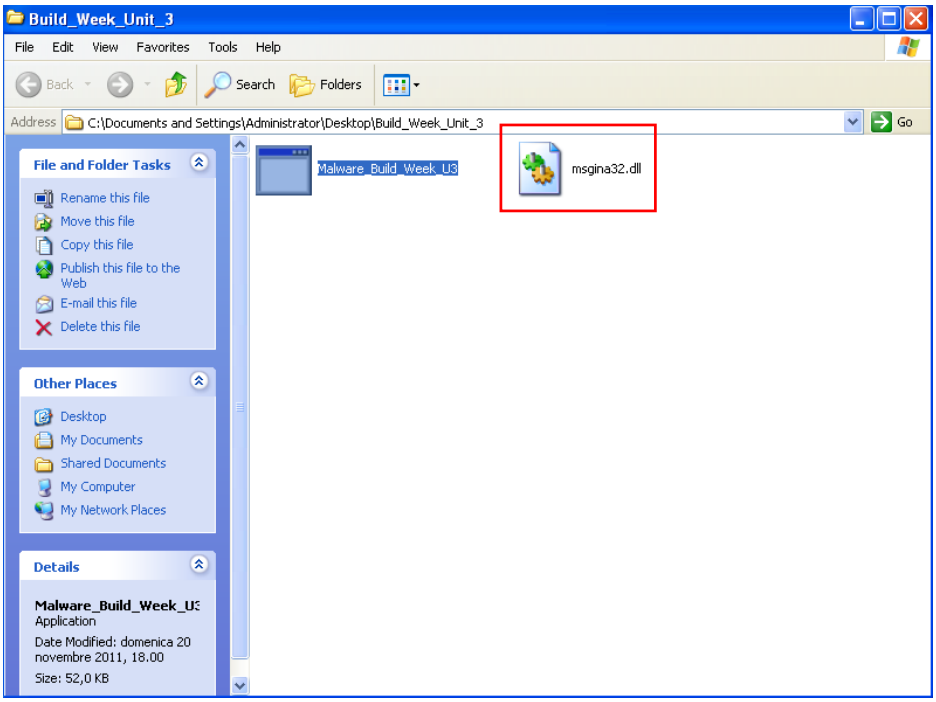
Passate ora alla visualizzazione dell'attività sul file system.

4. Quale chiamata di sistema ha modificato il contenuto della cartella dove è presente l'eseguibile del Malware?

Unite tutte le informazioni raccolte fin qui sia dall'analisi statica che dall'analisi dinamica per delineare il funzionamento del Malware.

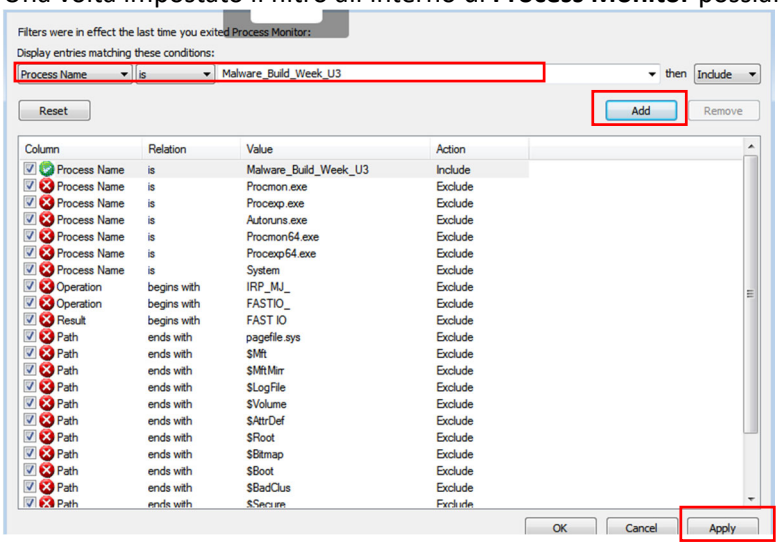
1. **Cosa notate all'interno della cartella dove è situato l'eseguibile del malware?**
   Spiegate cosa è avvenuto unendo le evidenze che avete raccolto finora.
   Nella cartella che contiene il malware, dopo averlo eseguito, vediamo un file creato dal malware.
   Dalle analisi eseguite nei giorni precedenti abbiamo dedotto che l'eseguibile fosse un **Dropper,** creando un file all'interno della macchina vittima.

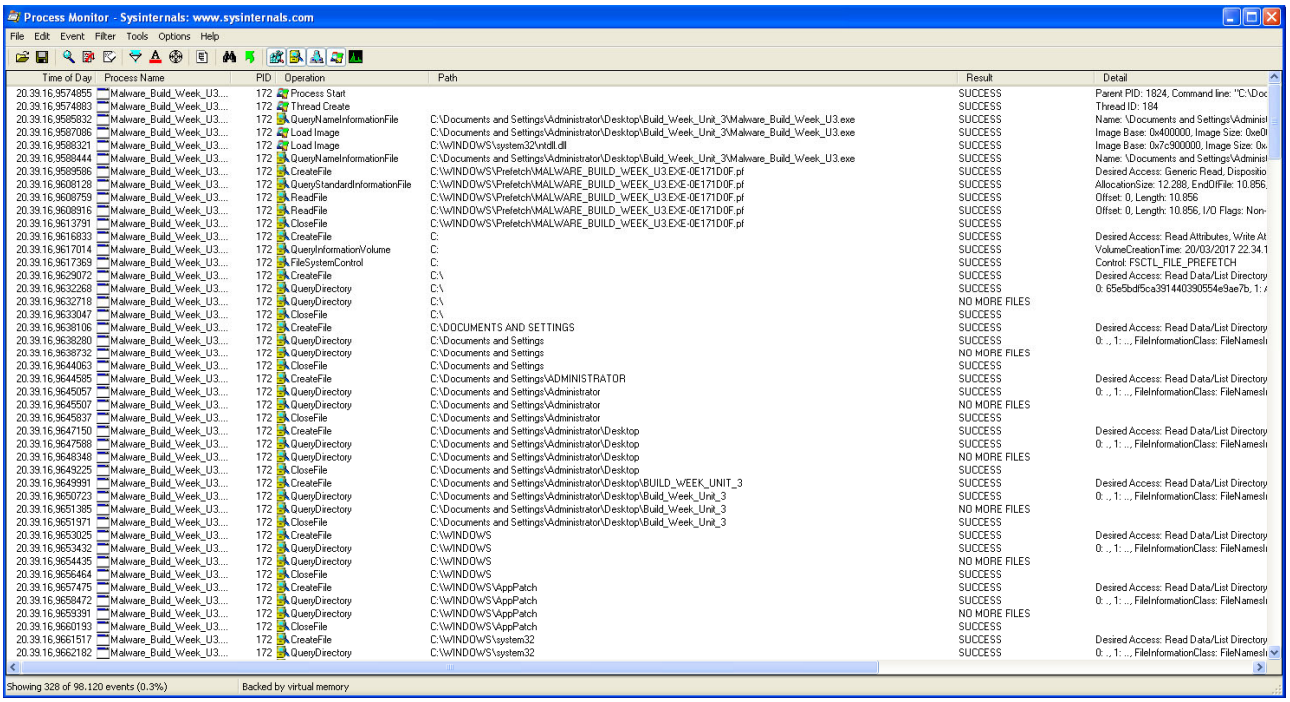




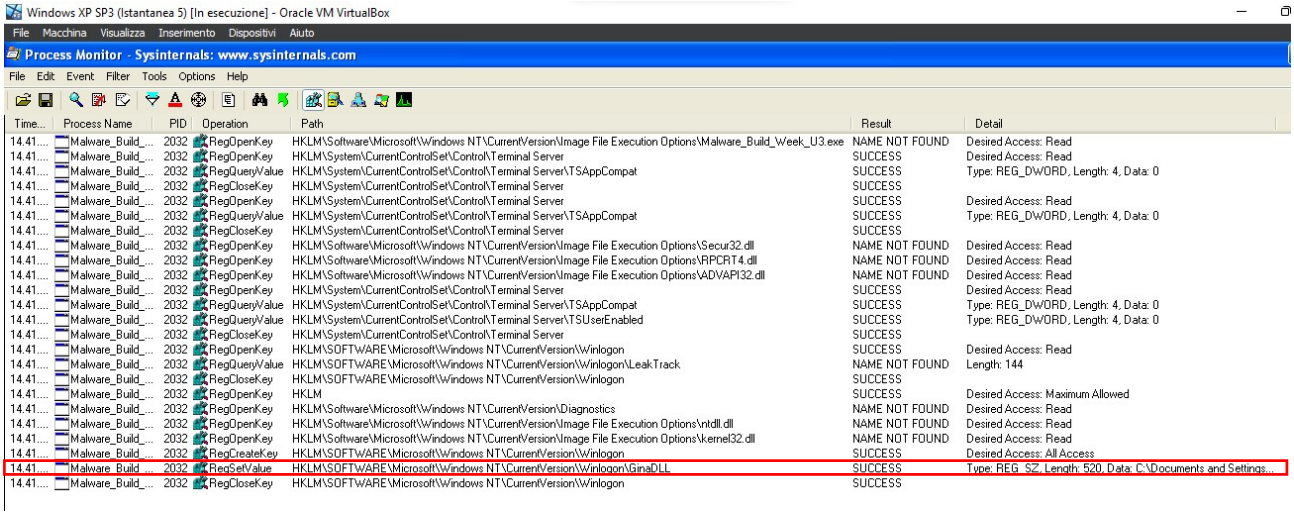
2. Quale chiave di registro viene creata?

Una volta impostato il filtro all'interno di **Process Monitor** possiamo vedere la seguente schermata.



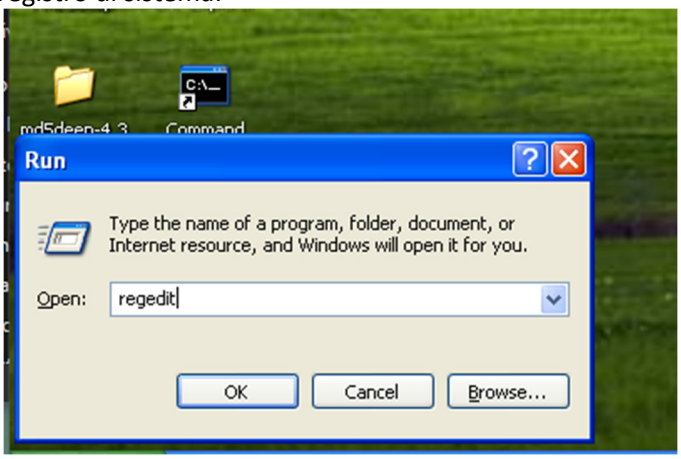


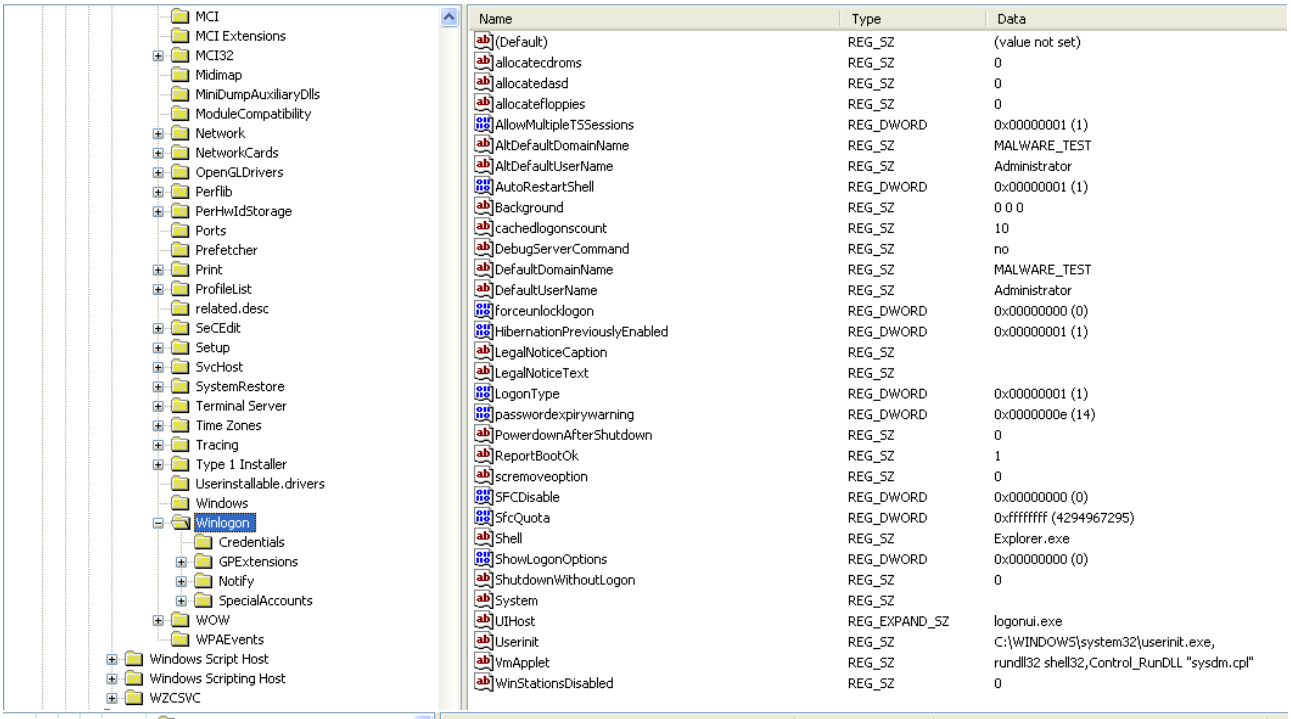
Applicando un ulteriore filtro possiamo vedere quale chiave di registro viene creata.

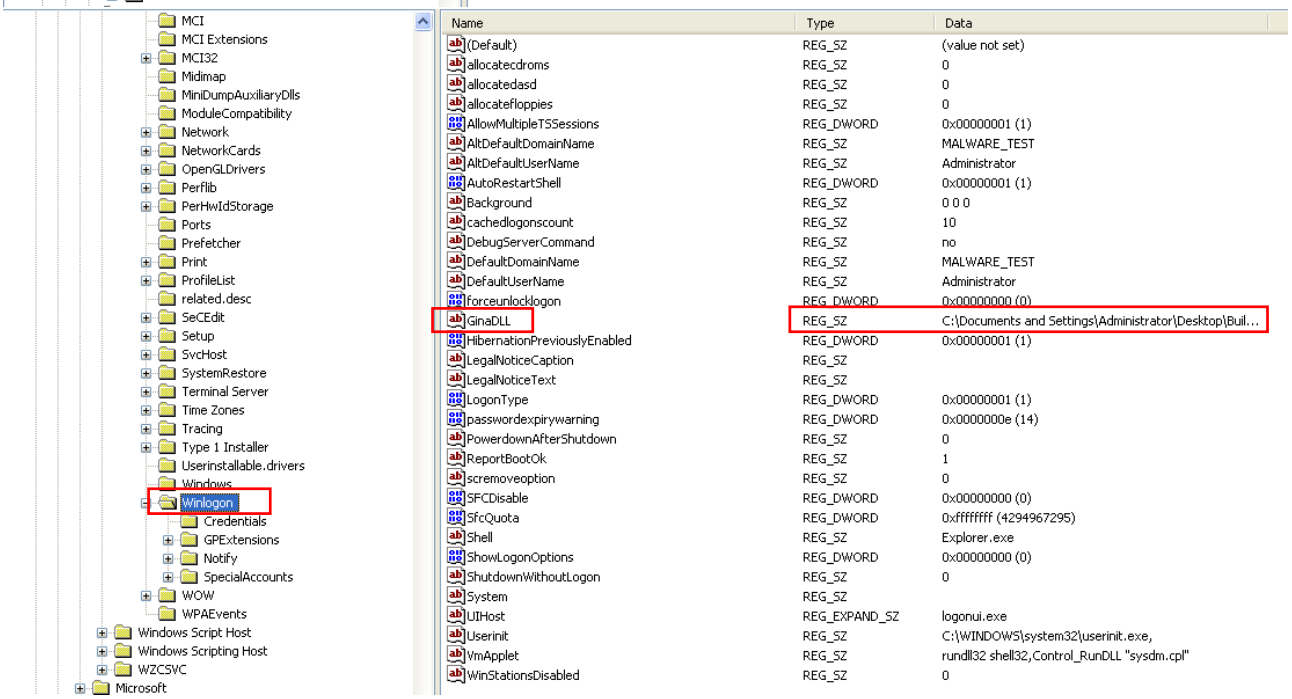


3. Quale valore viene associato alla chiave di registro creata?

Per poter leggere il valore associato alla chiave di registro creata, andiamo a visualizzare l'editor del registro di sistema.







Qui possiamo notare la presenza di una libreria aggiunta dal malware, dove il valore sarebbe il path dove è presente il file.

4. **Quale chiamata di sistema ha modificato il contenuto della cartella dove è presente l'eseguibile del malware?**

Operation	Path	Result	Detail
QueryNameInformationFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	Name: \Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe
QueryNameInformationFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	Name: \Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe
CreateFile	C:\WINDOWS\Prefetch\MALWARE_BUILD_WEEK_U3.EXE-0E171D0F.pf	SUCCESS	Desired Access: Generic Read, Disposition: Open, Options: Synchronous IO Non-Alert, Attributes: n/a, ShareMode: None, AllocationSize: n/a, OpenResult: Opened
QueryStandardInformationFile	C:\WINDOWS\Prefetch\MALWARE_BUILD_WEEK_U3.EXE-0E171D0F.pf	SUCCESS	AllocationSize: 12,288, EndOfFile: 10,824, NumberOfLinks: 1, DeletePending: False, Directory: False
ReadFile	C:\WINDOWS\Prefetch\MALWARE_BUILD_WEEK_U3.EXE-0E171D0F.pf	SUCCESS	Offset: 0, Length: 10,824
ReadFile	C:\WINDOWS\Prefetch\MALWARE_BUILD_WEEK_U3.EXE-0E171D0F.pf	SUCCESS	Offset: 0, Length: 10,824, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
CloseFile	C:\WINDOWS\Prefetch\MALWARE_BUILD_WEEK_U3.EXE-0E171D0F.pf	SUCCESS	
CreateFile	C:\Documents and Settings\Administrator\Desktop\BUILD_WEEK_UNIT_3	SUCCESS	Desired Access: Read Data/List Directory, Synchronize, Disposition: Open, Options: Directory, Synchronous IO Non-Alert, Open For Backup, Attributes: n/a, ShareMode: Read,
QueryDirectory	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS	0: ., 1: .., FileInformationClass: FileNamesInformation, 3: Malware_Build_Week_U3.idb
QueryDirectory	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	NO MORE FILES	
CloseFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS	
CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\MALWARE_BUILD_WEEK_U3.EXE	SUCCESS	Desired Access: Read Data/List Directory, Read Attributes, Disposition: Open, Options: Non-Directory File, Attributes: N, ShareMode: Read, Write, Delete, AllocationSize: n/a,
CreateFileMapping	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	SyncType: SyncTypeCreateSection, PageProtection: PAGE_READWRITE
QueryStandardInformationFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	AllocationSize: 53,248, EndOfFile: 53,248, NumberOfLinks: 1, DeletePending: False, Directory: False
CreateFileMapping	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	SyncType: SyncTypeOther
CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\MSGINA32.DLL	NAME NOT FOUND	Desired Access: Read Data/List Directory, Read Attributes, Disposition: Open, Options: Non-Directory File, Attributes: N, ShareMode: Read, Write, Delete, AllocationSize: n/a
ReadFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	Offset: 4,096, Length: 4,096, I/O Flags: Non-cached, Paging I/O
CloseFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	
CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\MALWARE_BUILD_WEEK_U3.EXE	SUCCESS	Desired Access: Execute/Traverse, Disposition: Open, Options: Non-Directory File, Attributes: N, ShareMode: Read, Write, Delete, AllocationSize: n/a, OpenResult: Opened
CreateFileMapping	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	SyncType: SyncTypeCreateSection, PageProtection: PAGE_EXECUTE
CreateFileMapping	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	SyncType: SyncTypeOther
ReadFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	Offset: 4,096, Length: 24,576, I/O Flags: Non-cached, Paging I/O
ReadFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	Offset: 28,672, Length: 4,096, I/O Flags: Non-cached, Paging I/O
ReadFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	Offset: 32,768, Length: 12,288, I/O Flags: Non-cached, Paging I/O
CloseFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	
CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS	Desired Access: Execute/Traverse, Synchronize, Disposition: Open, Options: Directory, Synchronous IO Non-Alert, Attributes: n/a, ShareMode: Read, Write, AllocationSize: n/
FileSystemControl	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS	Control: FSCTL_IS_VOLUME_MOUNTED
QueryOpen	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe.Local	NAME NOT FOUND	
CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS	Desired Access: Generic Write, Read Attributes, Disposition: Overwritelf, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: N, ShareMode: Read, Write, Allocat
CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS	Desired Access: Synchronize, Disposition: Open, Options: Directory, Synchronous IO Non-Alert, Open For Backup, Attributes: N, ShareMode: Read, Write, AllocationSize: n/a,
CloseFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS	
WriteFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS	Offset: 0, Length: 4,096
WriteFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS	Offset: 4,096, Length: 2,560
CloseFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS	
CloseFile	C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS	

CreateFile è la chiamata di sistema che ci fa vedere dove è stato creato il file richiesto (procmon) "**msgina32.dll**".

Giorno 5:

GINA (Graphic identication & authentication) è un componente lecito di Windows che permette l'autenticazione degli utenti tramite interfaccia grafica - ovvero permette agli utenti di inserire **username** e **password** nel classico riquadro Windows, come quello in figura a destra che usate anche voi per accedere alla macchina virtuale.

1. **Cosa può succedere se il file .dll lecito viene sostituito con un file .dll malevolo, che intercetta i dati inseriti?**

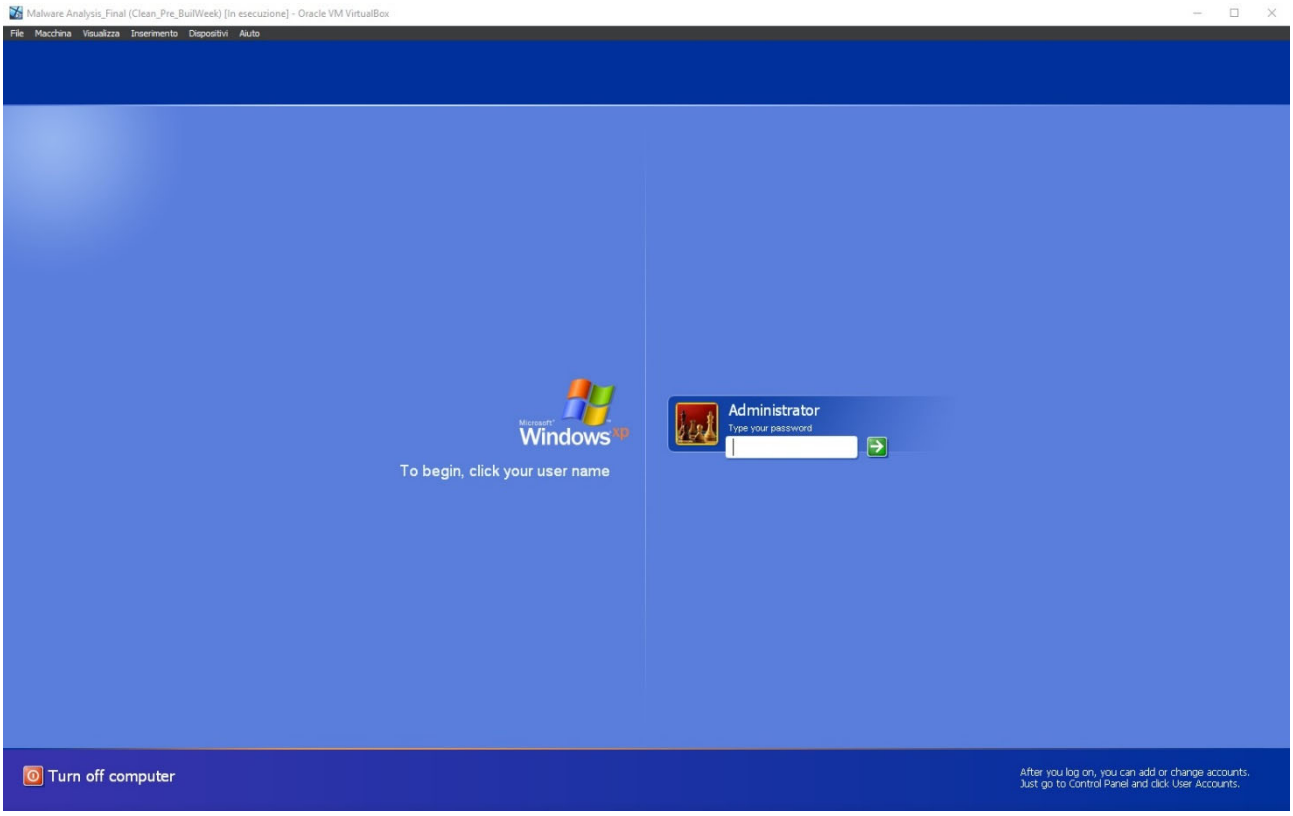
   Il malware studiato nei giorni precedenti ha la capacità di intromettersi tra le credenziali inserite dall'utente e il "controllo" che effettua il sistema operativo.

   La funzionalità specifica di "gina.dll" dipende dalla versione di Windows utilizzata. Nelle versioni precedenti di Windows, come Windows NT e Windows 2000, "gina.dll" veniva utilizzato come componente del sistema di identificazione e autenticazione grafica (GINA). GINA era responsabile della gestione del processo di login e forniva un'interfaccia grafica (GUI) per l'accesso al sistema.

   È possibile che il malware tenti di utilizzare "gina.dll" per eseguire azioni dannose, come il furto delle credenziali di accesso, la modifica del processo di accesso o l'installazione di un ulteriore malware.

   È importante notare che la sola presenza di "gina.dll" non indica che un file sia dannoso, poiché è comunemente utilizzato anche da software legittimi. Per determinare se un file è un malware o meno, è necessaria un'ulteriore analisi del comportamento e del codice del file.

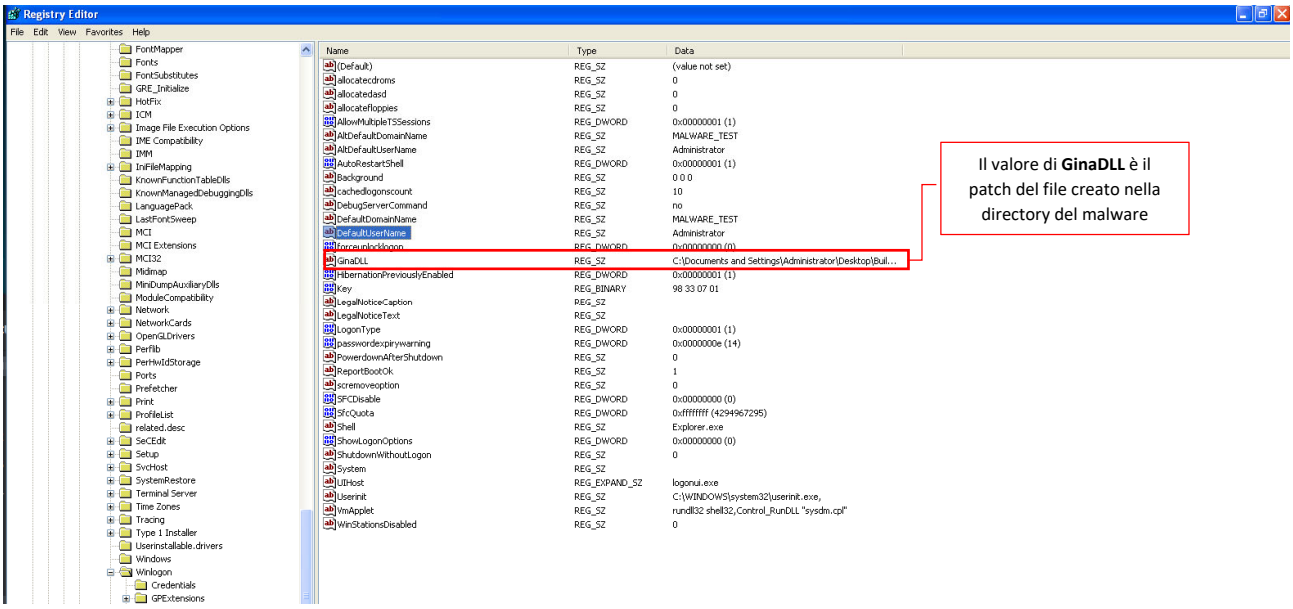
   Una prova che ci permette di capire che il nostro file **.dll** sia stato compromesso è proprio la schermata di login



Questa è la schermata di login prima che avviassimo il malware

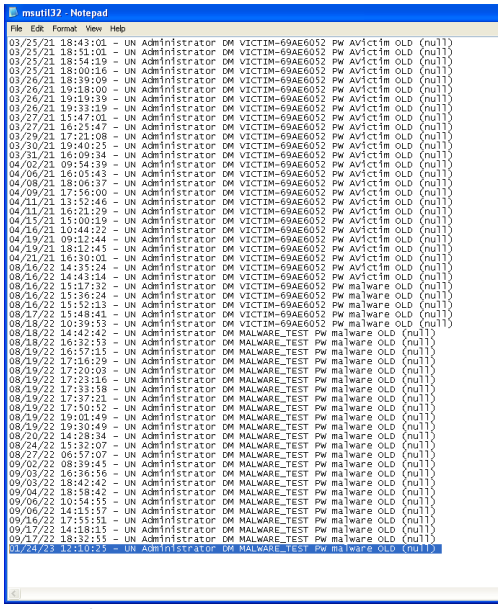
Questa invece è la schermata di login che appare nel momento che riaccediamo a windows dopo che il malware è stato avviato.

Il malware installa **msgina32.dll** come DLL GINA aggiungendola alla posizione del registro di sistema **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL**, che causa il caricamento della DLL dopo il riavvio del sistema.

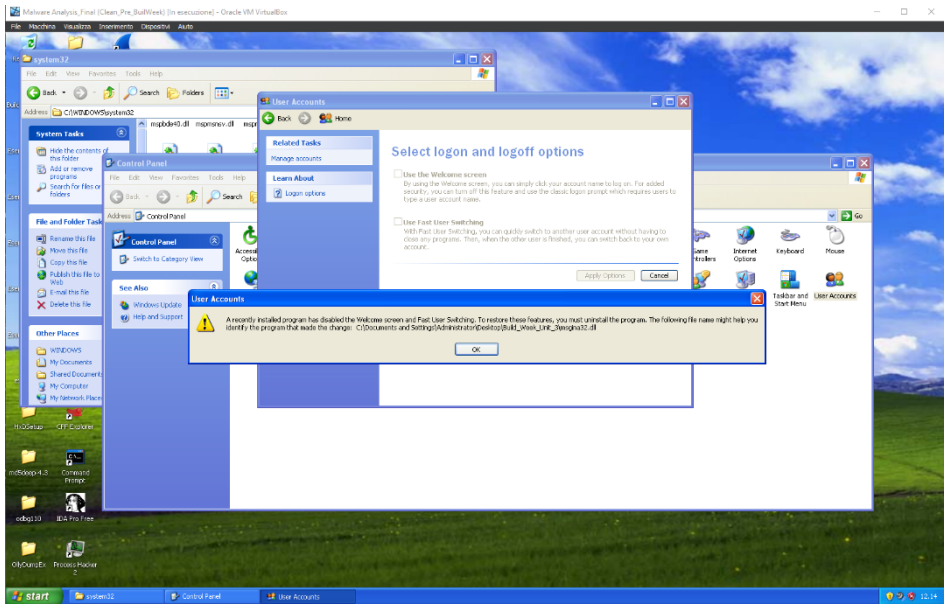


Il malware ruba le credenziali dell'utente eseguendo l'intercettazione GINA. Il file msgina32.dll è in grado di intercettare tutte le credenziali utente inviate al sistema per l'autenticazione.

Il malware registra le credenziali rubate in **%SystemRoot%\System32\msutil32.sys**. Il nome utente, il dominio e la password vengono registrati nel file. Una volta che il malware è stato installato, è necessario un riavvio del sistema per l'intercettazione.



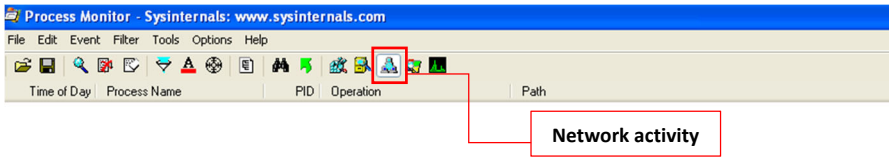
Le credenziali intercettate purtroppo non possono più essere cambiate. Il malware non permette la rimozione dei file di registro (**msgina32.dll**) creati e nemmeno la modifica da parte dell'utente. La modifica anche se in fase di avvio con modalità provvisoria è inutile. Una volta che il malware è stato avviato e ha ottenuto persistenza l'utente prima di poter fare qualsiasi cosa deve costantemente inserire la password di login all'avvio della macchina.

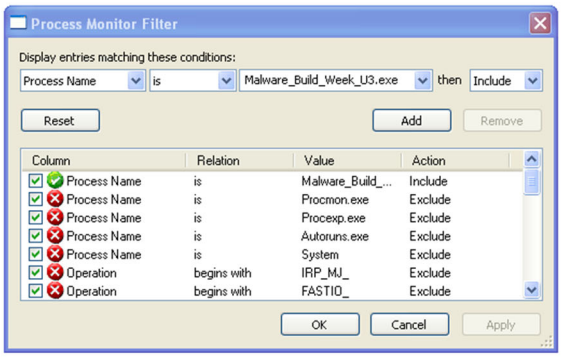


Qui possiamo notare come un messaggio di errore ci informa che un programma installato ha disabilitato i nostri privilegi, in questo caso ROOT, bloccandoci. Nell'allocazione di memoria **00401017** troviamo che il Malware utilizza una chiave di registro per ottenere la persistenza, seguendo il path che viene inserito come parametro, "**Software\Microsoft\Windows NT\Current Versione\ Winlogon".**

È possibile che il malware tenti di utilizzare "gina.dll" per eseguire azioni dannose, come il furto delle credenziali di accesso, la modifica del processo di accesso o l'installazione di ulteriore malware che però non viene scaricato da fonti esterne ma è già presente all'interno dello stesso.

Possiamo notarlo dal fatto che non ci sia attività di rete (controllando sulla finestra **network activity** di **Process Monitor** non notiamo alcuna attività) ma presenta comunque delle funzioni al suo interno, **FindResource**, **LoadResource**, che ci dicono che l'eseguibile dannoso va alla ricerca di risorse da qualche parte (deduciamo al suo interno) e le carichi sul target.





Dopo un'analisi attenta dei processi che il malware utilizza e delle librerie che modifica possiamo dire che si tratta di un **Dropper**, che carica sulla macchina target uno **Spyware**. Si tratta di un termine generico che indica qualunque software dannoso che infetta PC e dispositivi mobili al fine di raccogliere informazioni sugli utenti e i dati.

2. **Sulla base della risposta sopra, delineate il profilo del Malware e delle sue funzionalità. Unite tutti i punti per creare un grafico che ne rappresenti lo scopo ad alto livello.**