

*Build Week S12/L1*

# BUILD WEEK

*Team Leader: Sergio Bodron*

*Goldy Gabriel*

*Aguiglia Andrea*

*Bordese Andrea*

*Meneo Nicola*

*Chiriatti Mattia*

# Giorno 1 - Mattia

---

*Con riferimento al file eseguibile Malware\_Build\_Week\_U3, rispondere ai seguenti quesiti utilizzando i tool e le tecniche apprese nelle lezioni teoriche:*

- *Quanti parametri sono passati alla funzione Main()?*
- *Quante variabili sono dichiarate all'interno della funzione Main()?*
- *Quali sezioni sono presenti all'interno del file eseguibile? Descrivete brevemente almeno 2 di quelle identificate*
- *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.*

# Giorno 1 - Quesiti 1 e 2

Utilizzando IDA Explorer, potremo dare uno sguardo approfondito al codice Assembly del Malware e capire cosa si prefigge di fare lo stesso malware.

In questo caso, i parametri richiamati dalla funzione Main(), e poi passati attraverso lo stack, sono principalmente 3 e sono indicati da un offset positivo rispetto al registro EBX indicato in seguito:

- argc
- argv
- envp

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

hModule= dword ptr -11Ch
Data= byte ptr -118h
var_117= byte ptr -117h
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
```

Le variabili, invece, richiamate dalla funzione Main() sono ben 5 e verranno poi allocate nello stack all'inizio della stessa funzione e, inoltre, sono indicate da un offset negativo rispetto al registro EBX indicato in seguito:

- var\_117
- var\_8
- var\_4
- hModule
- Data

# Giorno 1 - Quesito 3

Utilizzando CFF Explorer, potremo dare uno sguardo approfondito al codice del malware e capire subito con che sezioni e librerie interagisce lo stesso malware.

In questo caso, le sezioni con cui interagisce sono 4: .text, .rdata, .data, .rsrc.

- **.text** contiene il codice eseguibile del programma, ovvero tutte le istruzioni che poi verranno eseguite dalla CPU;
- **.rdata** contiene dati solo-lettura (read-only data), come stringhe di testo o costanti utilizzate dal programma. Questi dati non possono essere modificati durante l'esecuzione del programma;
- **.data** contiene dati modificabili durante l'esecuzione del programma, come variabili globali o dati allocati dinamicamente durante l'esecuzione
- **.rsrc** contiene le risorse del programma, come icone, stringhe localizzate, informazioni sulla versione e altri dati non eseguibili utilizzati dall'applicazione

| Name    | Virtual Size | Virtual Address | Raw S |
|---------|--------------|-----------------|-------|
| Byte[8] | Dword        | Dword           | Dword |
| .text   | 00005646     | 00001000        | 00006 |
| .rdata  | 000009AE     | 00007000        | 00001 |
| .data   | 00003EA8     | 00008000        | 00003 |
| .rsrc   | 00001A70     | 0000C000        | 00002 |

# Giorno 1 - Quesito 4

Utilizzando CFF Explorer, potremo dare uno sguardo approfondito al codice del malware e capire subito con che sezioni e librerie interagisce lo stesso malware.

In questo caso, le librerie importate con cui interagisce sono 2: KERNEL32.dll, ADVAPI32.dll.

- KERNEL32.dll contiene le funzioni principali per interagire con il sistema operativo, ad esempio: manipolazione dei file, la gestione della memoria;
- ADVAPI32.dll contiene le funzioni per interagire con i servizi ed i registri del sistema operativo.

All'interno delle librerie, invece, sono raccolte diverse funzioni utili al malware per applicare i suoi effetti negativi e, nel caso, anche devastanti sulla macchina vittima.

| Module Name  | Imports      | OFTs  |
|--------------|--------------|-------|
| szAnsi       | (nFunctions) | DwOR  |
| KERNEL32.dll | 51           | 00007 |
| ADVAPI32.dll | 2            | 00007 |

# Giorno 1 - Quesito 4

Analizzando le funzioni contenute in KERNEL32.dll si nota facilmente come ci siano delle funzioni orientate sia alla creazione e allo storage di nuovi file, oltre anche a funzioni orientate a prendere il controllo della linea di comando presente sulla macchina vittima.

In definitiva, si può desumere che il malware provi a ottenere un accesso completo a quella che è la macchina vittima, essendo KERNEL32.dll una libreria che copre molte funzioni di controllo, di scrittura e creazione file, oltre a prendere anche il controllo della linea di comando.

ADVAPI32.dll, invece, contiene due funzioni per interagire col registro, come impostare i valori e creare chiavi di registro. Al momento, il malware potrebbe comportarsi come un Trojan.

| 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        |

## Giorno 2 - Andrea Bordese

---

*Con riferimento al Malware in analisi, spiegare:*

- *Lo scopo della funzione chiamata alla locazione di memoria 00401021;*
- *Come vengono passati i parametri alla funzione alla locazione 00401021;*
- *Che oggetto rappresenta il parametro alla locazione 00401017;*
- *Il significato delle istruzioni comprese tra gli indirizzi 00401027 e 00401029;*
- *Con riferimento all'ultimo quesito, tradurre il codice Assembly nel corrispondente costrutto C;*
- *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.*

## Giorno 2 - Quesiti 1, 2 e 3

```
* .text:00401017      push    offset SubKey    ; "SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon"
* .text:0040101C      push    80000002h   ; hKey
* .text:00401021      call    ds:RegCreateKeyExA
```

Utilizzando IDA Pro, potremo dare uno sguardo approfondito al codice del malware e, attraverso la conversione in linguaggio Assembly, potremo capire il comportamento del malware.

La funzione chiamata all'indirizzo di memoria 0x00401021 è RegCreateKeyExA. Questa funzione è parte delle API del Registro di sistema di Windows e viene utilizzata per creare o aprire una chiave nel Registro di sistema.

I parametri vengono passati a questa funzione tramite un'istruzione di push alla locazione 00401017. L'istruzione "push offset SubKey" inserisce l'indirizzo di memoria della stringa "SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon" nello stack.

Il parametro passato alla funzione che è rappresentato dalla stringa SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon rappresenta un percorso nel Registro di sistema di Windows. All'interno di questa chiave di registro, è possibile trovare molte sottocategorie che corrispondono a diverse parti del sistema operativo, come impostazioni di rete, impostazioni del desktop, impostazioni di sicurezza e così via.

## Giorno 2 - Quesito 4

L'istruzione "test eax, eax" effettua un AND tra i due operandi specificati. In questo caso lo sta facendo tra eax e sé stesso.

Successivamente i flag del processore vengono impostati in base al risultato dell'operazione. Se il risultato dell'operazione è 0 il flag viene settato a 0(zero), altrimenti viene resettato.

L'istruzione "jz short loc\_401032" effettua un salto condizionale. In questo caso se il valore del flag impostato è 0, verrà effettuato un salto alla locazione "loc\_401032".

Se il valore impostato dall'operazione "test eax, eax" non è 0 il programma continua l'esecuzione dopo l'istruzione "jz".

|                  |      |                  |
|------------------|------|------------------|
| • .text:00401027 | test | eax, eax         |
| • .text:00401029 | jz   | short loc_401032 |
| • .text:0040102B | mov  | eax, 1           |
| • .text:00401030 | jmp  | short loc_40107B |

|                  |      |                              |
|------------------|------|------------------------------|
| • .text:0040103E | push | offset ValueName ; "GinaDLL" |
| • .text:00401043 | mov  | eax, [ebp+hObject]           |
| • .text:00401046 | push | eax ; hKey                   |
| • .text:00401047 | call | ds:ReqSetValueExA            |

## Giorno 2 - Quesito 5

Le istruzioni comprese tra gli indirizzi di memoria 00401027 e 00401029 possono essere tradotte in un codice in linguaggio C.

Questo codice verifica che la variabile eax sia uguale a 0.

Se questa condizione è vera allora il codice salta a loc\_401032, altrimenti continua con l'esecuzione del programma.

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

```
int main() {
    int eax = 0;

    if (eax == 0) {
        // Salta a loc_401032
    } else {
        // Continua con l'esecuzione del programma
    }

    return 0;
}
```

## Giorno 2 - Quesito 6, 7

|                  |                                   |
|------------------|-----------------------------------|
| • .text:0040103E | push offset ValueName ; "GinaDLL" |
| • .text:00401043 | mov eax, [ebp+hObject]            |
| • .text:00401046 | push eax ; hKey                   |
| • .text:00401047 | call ds:RegSetValueExA            |

-L'istruzione alla locazione 0x00401047 rappresenta una chiamata alla funzione "RegSetValueExA". In questa funzione la stringa "GinaDLL" viene passata come parametro "ValueName".

Questo parametro viene passato alla funzione "RegSetValueExA" dall'istruzione "push offset ValueName" nella locazione 0040103E.

Analizzando questi dati sembrerebbe che il malware cerchi di modificare o impostare un valore di registro nel Registro di sistema di Windows.

La chiave di registro viene creata dalla funzione "RegCreateKeyExA" e successivamente viene impostato un valore di registro dalla funzione "RegSetValueExA".

Viene utilizzata la stringa "GinaDLL" come valore di registro e questo può significare che il malware cerchi di modificare la DLL di autenticazione in modo da ottenere l'accesso al sistema.

## Giorno 3 - Aguglia Andrea

---

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

- *Qual è il valore del parametro «ResourceName» passato alla funzione FindResourceA();*
- *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?*
- *È possibile identificare questa funzionalità utilizzando l'analisi statica basica ? (dal giorno 1 in pratica)*
- *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(). Disegnare un diagramma di flusso (inserite all'interno dei box solo le informazioni circa le funzionalità principali) che comprenda le 3 funzioni.*

# Giorno 3 - Quesiti 1,2

Avviamo sia IDA che OllyDBG per rispondere alle domande poste.

Individuiamo la funzione FindResource() in OllyDBG all'interno della routine che inizia all'indirizzo 00401080. Il parametro <>ResourceName<> passato alla funzione è <>TGAD<>.

La sequenza delle chiamate alle funzioni:

- FindResource();
- LoadResource();
- SizeOfResource();

Suggerisce un comportamento tipico di un dropper.

In altre parole, durante l'esecuzione, il malware estraе da una sua sezione un componente malevolo, come ad esempio un altro malware (nel caso specifico, potrebbe essere GINA.dll).

The screenshot shows the assembly code for a malware sample in OllyDbg. The code starts at address 00401080 and includes several API calls:

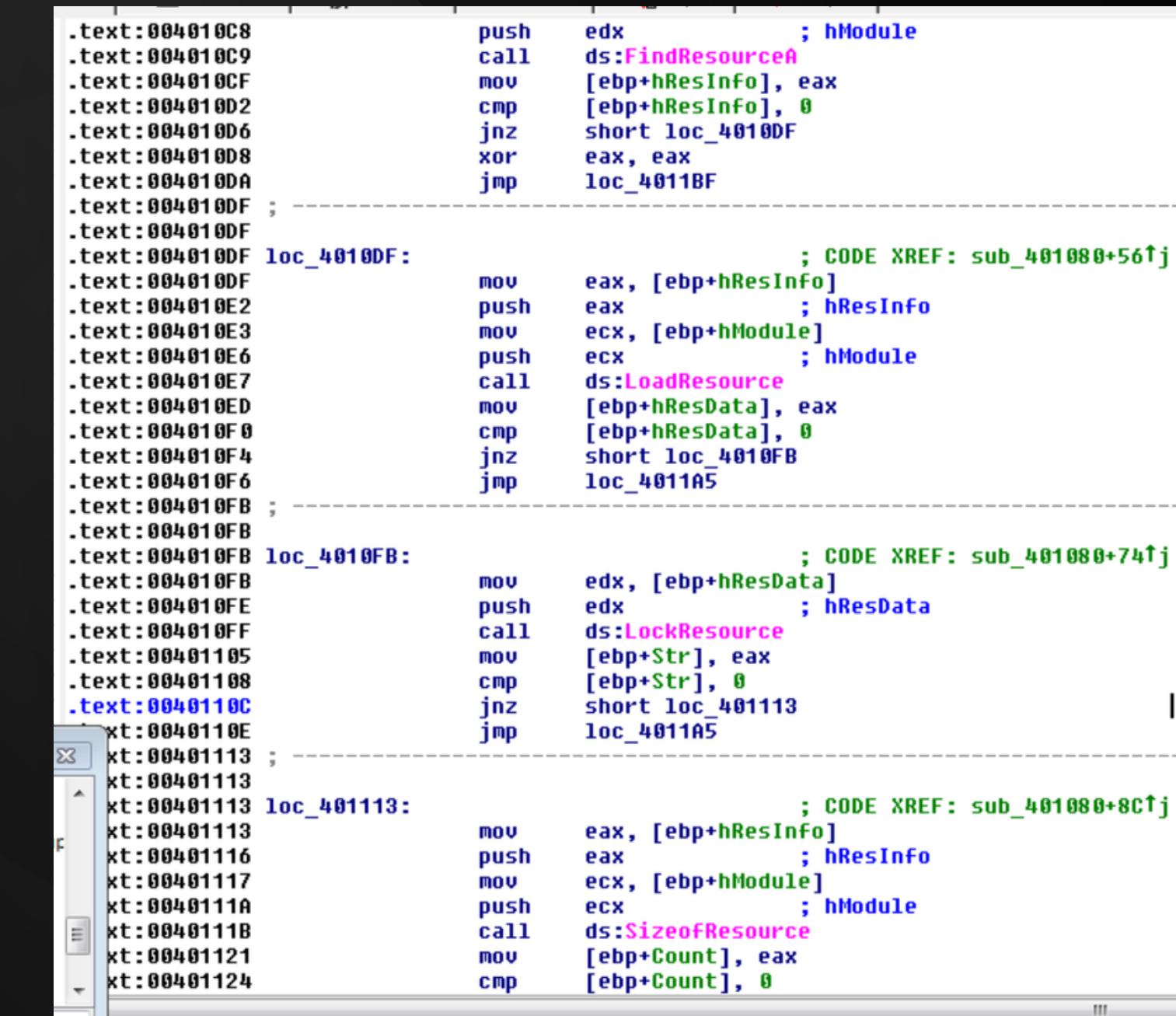
- FindResource()**: Located at 00401080, it pushes EBP, ESI, and EDI onto the stack, then calls KERNEL32.FindResource (FF15 28704000). A red bracket on the right indicates the API call with the label "FindResource".
- LoadResource()**: Located at 004010D0, it pushes EAX, ECX, and ECX onto the stack, then calls KERNEL32.LoadResource (FF15 14704000). A red bracket on the right indicates the API call with the label "LoadResource".
- LockResource()**: Located at 00401100, it pushes EAX, ECX, and ECX onto the stack, then calls KERNEL32.LockResource (FF15 10704000). A red bracket on the right indicates the API call with the label "LockResource".
- SizeofResource()**: Located at 00401120, it pushes EAX, ECX, and ECX onto the stack, then calls KERNEL32.SizeofResource (FF15 0C704000). A red bracket on the right indicates the API call with the label "SizeofResource".

Annotations on the right side of the assembly code provide context for the API parameters:

- ResourceType => "BINARY"**: Next to the first parameter of the FindResource call.
- Malware\_.00408038**: Next to the second parameter of the FindResource call.
- ResourceName => "TGAD"**: Next to the third parameter of the FindResource call.
- hModule**: Next to the first parameter of the LoadResource call.
- FindResourceA**: Next to the second parameter of the FindResource call.
- hResource**: Next to the first parameter of the LockResource call.
- hModu Le**: Next to the second parameter of the LoadResource call.
- LoadResource**: Next to the third parameter of the LoadResource call.
- hResource**: Next to the first parameter of the LockResource call.
- hModu Le**: Next to the second parameter of the LockResource call.
- LockResource**: Next to the third parameter of the LockResource call.
- hResource**: Next to the first parameter of the SizeofResource call.
- hModu Le**: Next to the second parameter of the SizeofResource call.
- SizeofResource**: Next to the third parameter of the SizeofResource call.

# Giorno 3 - Quesito 3,4

Grazie all'impiego di IDA, siamo in grado di individuare questa caratteristica attraverso un'analisi statica di base sin dalla prima giornata. Le evidenze a sostegno di questa identificazione includono la presenza di determinati pattern di codice, l'analisi delle funzioni coinvolte e la traccia di istruzioni specifiche che evidenziano la presenza della funzionalità in questione. L'interfaccia di IDA, con diversi colori per i diversi elementi dello stesso codice, facilita il processo di individuazione di queste prove, consentendo una comprensione dettagliata del comportamento del codice sottostante.

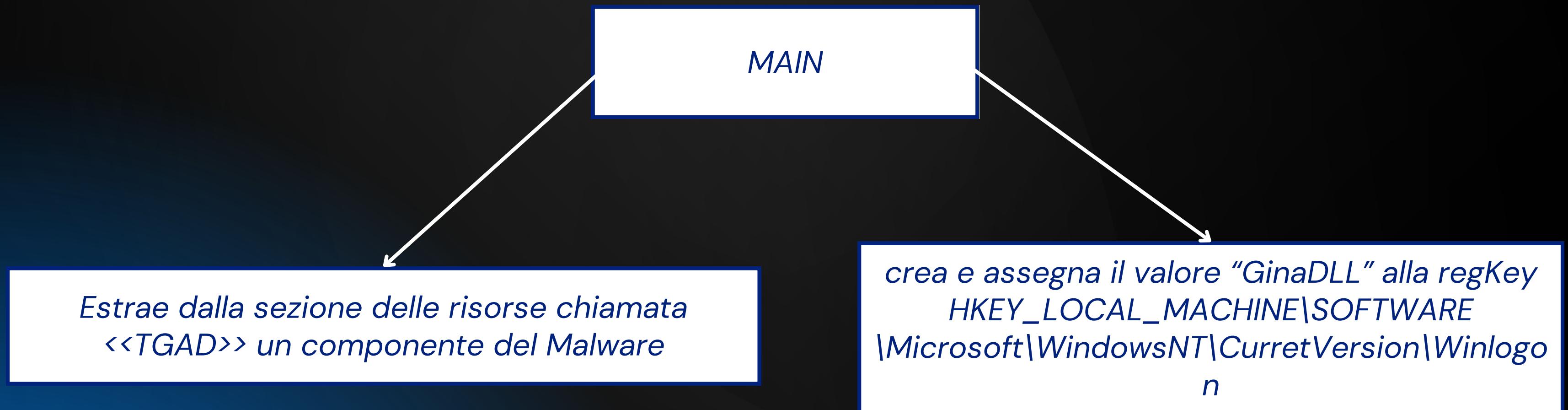


The screenshot shows the assembly view of IDA Pro with the following assembly code:

```
.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 ; ----- 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 ; ----- 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+Str], eax
.text:00401108        cmp     [ebp+Str], 0
.text:0040110C        jnz    short loc_401113
.text:0040110E        jmp    loc_4011A5
.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
.text:00401121        mov     [ebp+Count], eax
.text:00401124        cmp     [ebp+Count], 0
```

## Giorno 3 - Quesito 5

Sotto il profilo grafico, è possibile creare un diagramma di flusso che rispecchi in maniera analoga quanto mostrato in precedenza.



## Giorno 4 - Meneo Nicola

---

*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.*

- *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. Fate click su «ADD» poi su «Apply» come abbiamo visto nella lezione teorica.*

## Giorno 4 - Meneo Nicola

---

*Filtrate includendo solamente l'attività sul registro di Windows .*

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

*Passate ora alla visualizzazione dell'attività sul File System.*

- *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.*

# Giorno 4 - Quesit ?

Dopo aver preparato e avviato Process Monitor, possiamo eseguire il Malware.

Subito dopo l'avvio, possiamo notare che nella cartella dedicata all'eseguibile si va a creare il file "msgina32.dll".

In base a questo evento possiamo dichiarare che il Malware si comporti da dropper. Questo file verrà creato dalla sezione risorse del file malevolo chiamata "TGAD", sezione trovata grazie all'uso di OllyDBG, dove abbiamo visto come il parametro "ResourceName" vi sia stato assegnato, grazie alla routine che inizia all'indirizzo 00401080.

| Nome  | Ultima modifica  | Tipo                         | Dimensione |
|---|------------------|------------------------------|------------|
|  Malware_Build_Week_U3 | 17/01/2024 17:48 | Applicazione                 | 52 KB      |
|  msgina32.dll          | 05/03/2024 09:25 | Estensione dell'applicazione | 7 KB       |

# Giorno 4 - Quesit ?

Dopo aver configurato il filtro per visualizzare solo gli eventi prodotti dal Malware, passiamo all'analisi sull'attività delle chiavi di registro, dando inizio all'analisi dinamica del codice malevolo (con cosa?).

Come dimostrato dalla figura qui sotto, l'analisi conferma che il malware ha creato una nuova chiave di registro a cui viene assegnato il valore "GinaDLL". Questo conferma il comportamento del malware visto tramite IDA, dove GinaDLL assume valore di registro al fine di modificare le chiavi di registro.

|                   |      |               |  |               |                        |
|-------------------|------|---------------|--|---------------|------------------------|
| Malware_Build_... | 2612 | RegCreateKey  | HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon         | SUCCESS       | Desired Access: All... |
| Malware_Build_... | 2612 | RegSetInfoKey | HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon         | SUCCESS       | KeySetInformation...   |
| Malware_Build_... | 2612 | RegQueryKey   | HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon         | SUCCESS       | Query: HandleTag...    |
| Malware_Build_... | 2612 | RegSetValue   | HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL | ACCESS DENIED | Type: REG_SZ, Le...    |
| Malware_Build_... | 2612 | RegCloseKey   | HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon         | SUCCESS       |                        |

# Giorno 4 - Quesit ?

Infine passiamo all'analisi delle attività sul file system.

Sempre con riferimento all'immagine, notiamo la chiamata alla funzione "CreateFile" che, nel nostro caso, risulta necessaria alla creazione del file "msgina32.dll" nella cartella del Malware.

In tal modo, dopo l'esecuzione del malware verrà "droppato" il secondo file malevolo all'interno della cartella dedicata al malware.

|                   |      |            |  |         |                         |
|-------------------|------|------------|--|---------|-------------------------|
| Malware_Build_... | 1896 | CreateFile | C:\Users\user\Desktop\MALWARE\Build_Week_Unit_3\msgina32.dll | SUCCESS | Desired Access: G...    |
| Malware_Build_... | 1896 | WriteFile  | C:\Users\user\Desktop\MALWARE\Build_Week_Unit_3\msgina32.dll | SUCCESS | Offset: 0, Length: 4... |
| Malware_Build_... | 1896 | WriteFile  | C:\Users\user\Desktop\MALWARE\Build_Week_Unit_3\msgina32.dll | SUCCESS | Offset: 4.096, Leng...  |
| Malware_Build_... | 1896 | CloseFile  | C:\Users\user\Desktop\MALWARE\Build_Week_Unit_3\msgina32.dll | SUCCESS |                         |

## Giorno 5 - Gabriel Goldy

---

*GINA (Graphical identification and 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.*

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

*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.*

## Giorno 5 - Quesito 1

Nel momento in cui il file .dll lecito viene sostituito con uno malevolo, la sicurezza del sistema viene gravemente compromessa minacciando l'integrità e la confidenzialità dei dati, in quanto il malware può intercettare le credenziali immesse dall'utente ed eventualmente modificarle.

In particolare, le funzioni del malware studiato forniscono funzionalità di persistenza (grazie alle funzioni di creazione/modifica delle chiavi di registro) e di manipolazione delle risorse (find/load/lock/sizeof Resource), riuscendo così a nascondere la sua presenza nel sistema e a essere eseguito all'avvio.

Unendo le analisi fatte sul malware possiamo, quindi, ipotizzare che si tratti di un Trojan in quanto si "maschera" da GINA.dll, un'applicazione legittima di Windows, e abbia funzionalità di persistenza nel sistema grazie alle funzioni di scrittura sul registro.



# Giorno 5 - Grafico

