

Build Week 3 - Malware Analysis

23/02/2024

Francesco Alfonsi

Leonardo Londero

Matteo Iacullo

Nicolò Schittone

Riccardo Agostino Monti

INDICE

INDICE.....	1
Obiettivi Giorno 1:.....	3
1.1 Quanti parametri sono passati alla funzione main()?.....	3
Parametro.....	3
Funzione.....	4
1.2 Quante variabili sono dichiarate all'interno della funzione Main()?.....	6
Variabili.....	6
1.3 Quali sezioni sono presenti all'interno del file eseguibile?.....	7
Sezioni di un file .exe.....	7
1.4 Quali librerie importa il malware?.....	10
Cos'è una Libreria API Win32?.....	10
Cos'è una DLL?.....	10
• RegSetValueExA : Questa funzione imposta i dati e il tipo di un valore specificato in una chiave del Registro di sistema.....	12
• RegCreateKeyExA : Questa funzione crea una specifica chiave del Registro di sistema. Se la chiave esiste già, la funzione la apre.....	12
Conclusioni.....	13
Dropper.....	13
 Obiettivi Giorno 2:.....	 14
2.1 Spiegare lo scopo della funzione chiamata alla locazione di memoria 00401021:.....	14
RegCreateKeyExA.....	15
2.2 Come vengono passati i parametri alla funzione alla locazione 00401021:.....	16
2.3 Che oggetto rappresenta il parametro alla locazione 00401017:.....	16
lpSubKey.....	17
2.4 Spiegare il blocco delle istruzioni comprese tra gli indirizzi 00401027 e 00401029:.....	17
Operazione logica AND.....	18
Salto condizionale jz.....	18
OllyDbg.....	18
 Obiettivi Giorno 3:.....	 22

3.1 Qual è il valore del parametro <<ResourceName>> passato alla funzione FindResourceA()?	22
FindResourceA().....	22
3.2 Che funzionalità sta implementando il Malware?	24
Potenziali rischi per la sicurezza.....	25
3.3 - 3.4 È possibile identificare questa funzionalità utilizzando l'analisi statica basica? Elencare le evidenze a supporto	26
3.5 Diagramma di Flusso	26
Obiettivi Giorno 4:	27
4.0 Preparazione dell'ambiente di lavoro	28
Creazione di un'istantanea.....	28
ProcMon Setup.....	29
4.1 Cosa notate all'interno della cartella dove è situato l'eseguibile del Malware?	30
4.2 Quale chiave di registro viene creata?	31
4.3 Quale valore viene associato alla chiave di registro creata?	32
4.4 Quale chiamata di sistema ha modificato il contenuto della cartella dove è presente l'eseguibile del Malware?	32
Obiettivi Giorno 5:	33
5.1 Cosa può succedere se il file .dll lecito viene sostituito con un file .dll malevolo, che intercetta i dati inseriti?	34
DLL Malevoli.....	34
GINA.dll.....	34
5.2 Realizzare un grafico che rappresenti il malware ad alto livello	35

Obiettivi Giorno 1:

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

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

1.1 Quanti parametri sono passati alla funzione main()?

Prima di passare all'analisi dei parametri della funzione main chiariamo cos'è un parametro:

Parametro

In Assembly, un parametro di una funzione è un valore che viene passato alla funzione quando viene chiamata. I parametri vengono solitamente passati attraverso lo stack.

Ad esempio, considera il seguente codice Assembly :

main:

```
push 0x08
```

```
call my_function
```

my_function:

```
pop %eax
```

```
...
```

```
ret
```

In questo esempio, 0x08 è un parametro per la funzione `my_function`. Viene passato alla funzione utilizzando l'istruzione `push` per metterlo nello stack prima della chiamata alla funzione con

`<< call >>.`

Possiamo distinguere un parametro da una variabile perché quest'ultimo quando dichiarato all'interno della funzione ha valore di offset positivo. Prima di procedere puntualizziamo cos'è una funzione:

Funzione

In Assembly, una funzione è un blocco di codice che esegue un compito specifico. Può ricevere dei valori in ingresso, chiamati parametri (descritti nel paragrafo precedente) e può restituire un valore.

Le funzioni in Assembly sono molto simili alle funzioni in altri linguaggi di programmazione. Tuttavia, a causa della natura di basso livello dell'Assembly, la gestione delle funzioni può essere più complessa.

Ad esempio, per chiamare una funzione in Assembly, si utilizza l'istruzione:

`<< call "nome funzione">>`

Per passare i parametri alla funzione solitamente si utilizzano i registri o lo stack. Dopo che la funzione ha terminato l'esecuzione, restituisce il controllo al punto del programma da cui è stata chiamata. Inoltre, c'è da notare che l'istruzione `<< call >>` mette l'indirizzo di ritorno nello stack, quindi eseguendo `<< pop %eax >>` si ottiene l'indirizzo di ritorno, non il parametro. Per accedere al parametro si fa riferimento alla posizione appropriata nello stack utilizzando l'istruzione `<< mov eax, [esp+4] >>.`

A questo punto, avendo chiaro cos'è una funzione e un suo parametro, possiamo proseguire con la risoluzione del quesito. Come spiegato in precedenza per distinguere un parametro della funzione `main` basta con l'ausilio di IDA, un comodo tool di Win XP che ci permette di risalire dall'eseguibile al codice Assembly, verificare se è presente una dichiarazione del tipo:

`<< "nome_parametro" = dword ptr [x] >>`

dove `x` è un valore intero positivo che indica l'offset del parametro. Nel caso in cui `x` dovesse essere un valore negativo possiamo affermare che l'istruzione in questione sia la dichiarazione di una variabile.

A questo punto procediamo ad aprire IDA facendo doppio click sull'icona ed importare il nostro file.

Per importare il file clicchiamo sulla cartella in alto a sinistra della GUI di IDA, si aprirà un esplora risorse. Spostiamoci nella directory corretta e importiamo il file chiamato: Malware_Build_Week_U3.

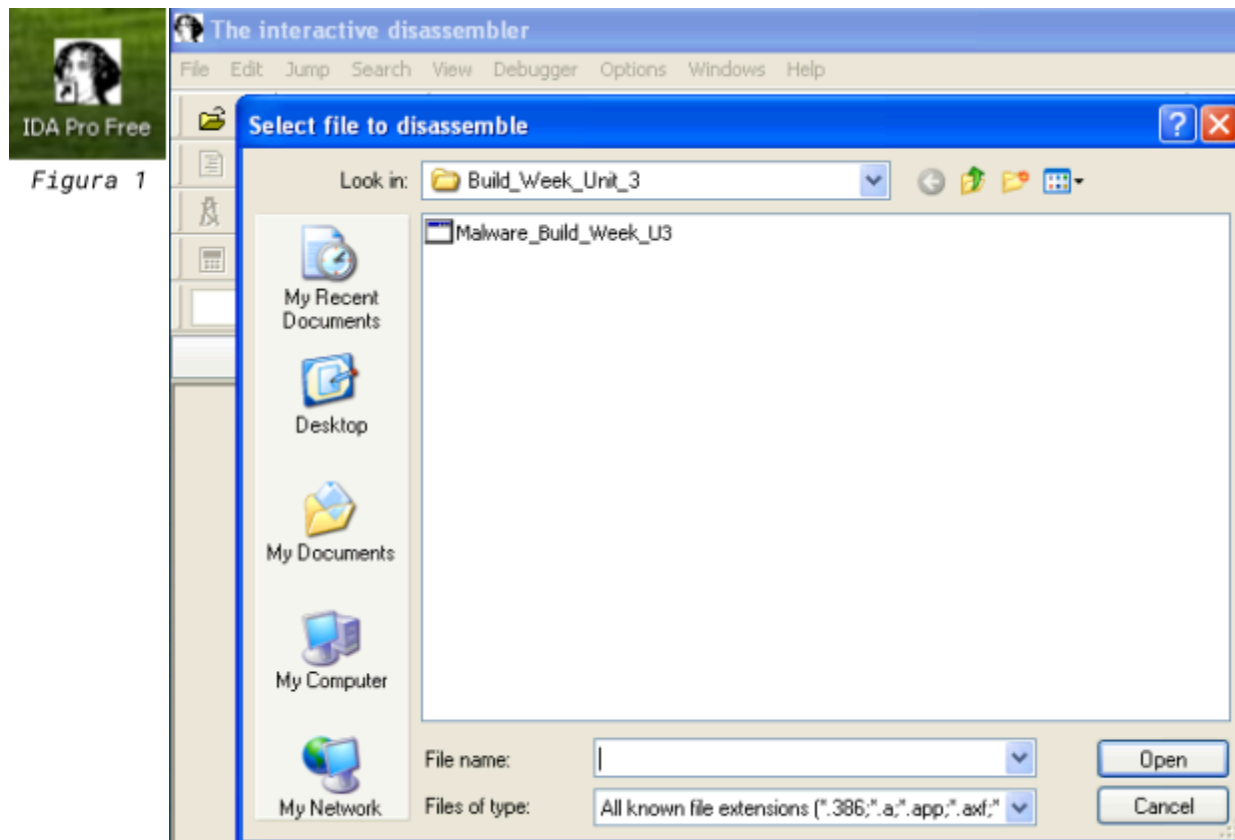


Figura 2

Una volta importato il Malware utilizzando il disassembly panel ci rechiamo nella sezione main del codice individuata dalla dichiarazione della funzione come segue:

```
<< int __cdecl main ( "parametro1", "parametro2", ...) _main proc near >>
```

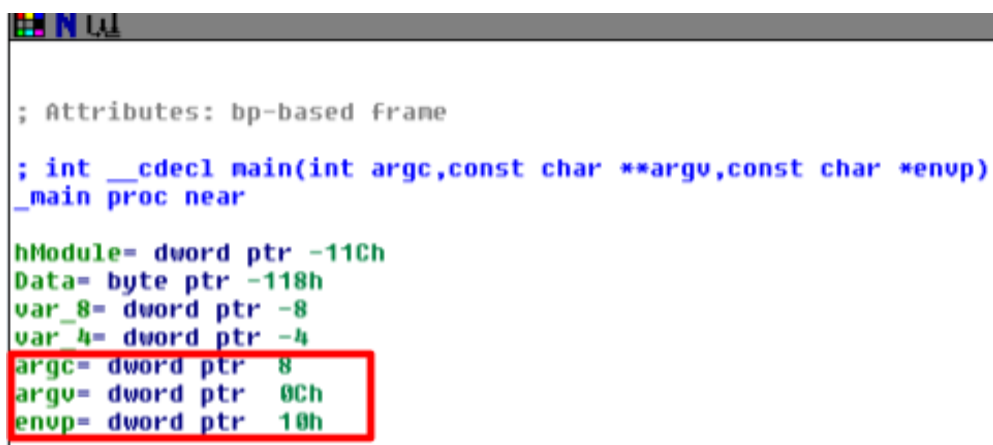


Figura 3

Con riferimento alla Figura 3, possiamo notare che i parametri della funzione main del codice sono 3, segnalati da un riquadro rosso.

1.2 Quante variabili sono dichiarate all'interno della funzione Main()?

Prima di controllare le variabili dichiarate all'interno della funzione Main, mettiamo in chiaro cos'è una variabile.

Variabili

In Assembly, una variabile è essenzialmente un'etichetta per uno spazio di memoria. Questo spazio di memoria può essere utilizzato per conservare dati che il programma deve manipolare.

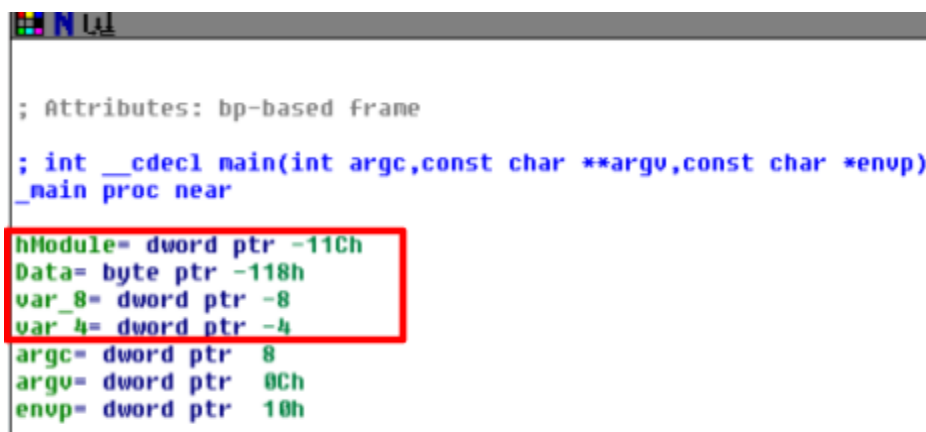
Le variabili in Assembly vengono dichiarate nella sezione .data del programma. Durante la dichiarazione della variabile bisogna specificare lo spazio occupato da tale variabile. Un esempio di dichiarazione di una variabile è:

<< "nome_variabile" = dword ptr [x] >>

Dove x è un valore intero negativo che indica l'offset della variabile. Nel caso in cui x dovesse essere positivo possiamo affermare che l'istruzione in questione sia la dichiarazione di un parametro.

È importante specificare che in Assembly è bene eseguire le operazioni esclusivamente attraverso i registri. Quindi, per utilizzare una variabile è opportuno prima spostare il valore della stessa in un registro.

Tornando a noi, per analizzare le variabili utilizzate dalla funzione main possiamo adoperare lo stesso metodo usato per l'analisi dei parametri, apriamo quindi IDA, spostiamoci nella finestra disassembly panel e torniamo ad analizzare il main.



```
; Attributes: bp-based frame
; int __cdecl main(int argc,const char **argv,const char *envp)
_main proc near
hModule= dword ptr -11Ch
Data= byte ptr -118h
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
```

Figura 4

Con riferimento alla Figura 4, possiamo notare che le variabili della funzione main del codice sono 4, segnalate da un riquadro rosso.

1.3 Quali sezioni sono presenti all'interno del file eseguibile?

Prima di controllare quali sono le sezioni presenti all'interno del file, precisiamo cosa si intende per sezione.

Sezioni di un file .exe

Le sezioni di un file .exe sono parti strutturate del file, che contengono diverse informazioni necessarie per la corretta esecuzione del programma.

In un file PE (Portable Executable), le sezioni sono elementi fondamentali che memorizzano specifici tipi di dati e codice. Svolgono un ruolo cruciale nell'organizzazione e gestione delle funzionalità del programma.

Sebbene esistano numerose sezioni, comprenderne alcune chiave è fondamentale per l'analisi malware di base. Di seguito le principali sezioni su cui porre particolare attenzione:

- **.text:** Questa sezione contiene il codice effettivo che il programma esegue. È qui che vengono archiviate le istruzioni che indicano al computer cosa fare. Analizzare questa sezione nell'analisi malware aiuta a identificare chiamate di funzione sospette, riferimenti a dati e potenziali attività dannose.
- **.rdata:** Questa sezione contiene dati di sola lettura come stringhe, costanti e risorse utilizzate dal programma. Il malware potrebbe incorporare stringhe offuscate o sfruttare vulnerabilità all'interno di questa sezione. Esaminarla può rivelare messaggi nascosti, dati di configurazione o indizi sulla sua origine.
- **.data:** Questa sezione memorizza variabili di dati inizializzate utilizzate dal programma durante l'esecuzione. Il malware potrebbe utilizzare questa sezione per archiviare dati temporanei o impostazioni di configurazione relative alle sue attività dannose. Analizzarla può aiutare a scoprire variabili nascoste, canali di comunicazione o codice iniettato.
- **.rsrc:** Questa sezione contiene le risorse del programma, come icone, immagini e menù. Sebbene apparentemente innocua, gli aggressori potrebbero abusare di questa sezione per nascondere codice dannoso o sfruttare vulnerabilità nelle librerie di analisi delle risorse. Esaminarla può rivelare payload nascosti o funzionalità mascherate.

- **.idata:** Questa sezione contiene informazioni sulle funzioni importate da altre librerie (DLL). Analizzare questa sezione può identificare importazioni sospette o inaspettate che potrebbero suggerire le capacità e le dipendenze del malware.
- **.edata:** Contiene variabili di dati non inizializzate, a volte utilizzate per memorizzare dati di configurazione o valori temporanei nel malware.
- **.reloc:** Contiene informazioni di rilocazione per regolare gli indirizzi dopo il caricamento del programma in memoria. Il malware potrebbe utilizzare questa sezione per l'iniezione di codice o tecniche anti-analisi.

Per effettuare l'analisi delle sezioni presenti nel malware, abbiamo utilizzato CFF Explorer, un tool che svolge diverse funzioni tra cui appunto la ricerca delle sezioni presenti all'interno di un eseguibile.

Avviamo CFF Explorer semplicemente cliccando due volte sulla sua icona

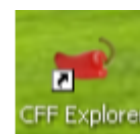


Figura 5

All'avvio dell'applicazione, cliccando sull'icona a forma di cartella posta in alto sulla sinistra (o, analogamente, scegliendo file -> open dal menù in alto), si aprirà una finestra di dialogo dedicata alla ricerca del file oggetto dell'analisi. Spostiamoci nella directory contenente il programma richiesto dall'esercizio: Malware_Build_Week_U3 e clicchiamo su open perché venga importato.

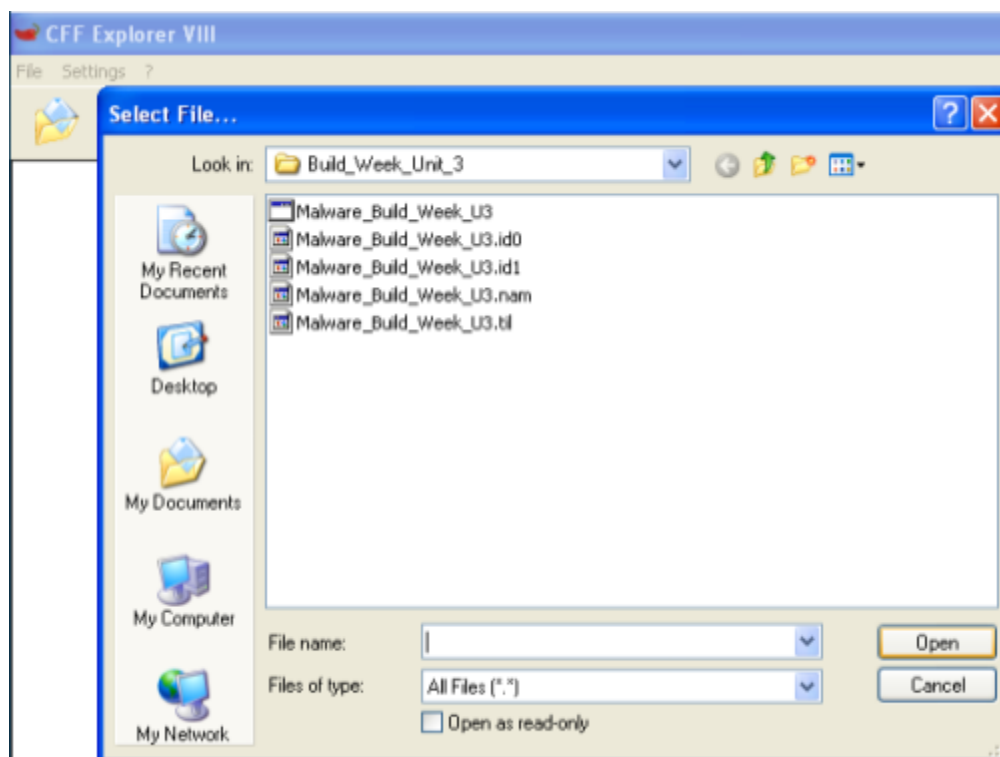


Figura 6

Una volta che l'eseguibile è stato acquisito, CFF Explorer mostrerà, nella parte sinistra dello schermo, un menù di operazioni eseguibili. Nel nostro caso, per la visualizzazione delle sezioni presenti, scegliamo la voce Section Headers.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00005646	00001000	00006000	00001000	00000000	00000000	0000	0000	60000020
.rdata	000009AE	00007000	00001000	00007000	00000000	00000000	0000	0000	40000040
.data	00003EA8	00008000	00003000	00008000	00000000	00000000	0000	0000	C0000040
.rsrc	00001A70	0000C000	00002000	0000B000	00000000	00000000	0000	0000	40000040

Figura 7

Le sezioni importate sono presenti in Figura 7; ogni sezione è già stata precedentemente approfondita, di particolare rilievo, notiamo .rsrc, sezione che di solito si trova quando si analizza un tipo di malware chiamato dropper.

1.4 Quali librerie importa il malware?

Come sempre, prima di procedere con l'analisi, spieghiamo cosa si intende per libreria.

Cos'è una Libreria API Win32?

Le funzioni Win32 API sono un insieme di funzioni fornite da Microsoft che permettono alle applicazioni di interagire con il sistema operativo Windows. Queste funzioni sono contenute in librerie dinamiche (DLL), come **kernel32.dll**, **user32.dll** e **gdi32.dll**, e offrono una vasta gamma di funzionalità, tra cui l'accesso diretto alle funzionalità di sistema e all'hardware.

Per esempio, si può utilizzare una funzione Win32 API per creare una finestra, leggere un file o inviare un messaggio di rete. Queste funzioni possono essere chiamate da qualsiasi linguaggio di programmazione che supporta l'interfaccia con il codice C.

Tuttavia, è importante ricordare che le API Win32 sono di basso livello. Molti sviluppatori preferiscono utilizzare linguaggi di programmazione di alto livello, come C# o Java, che offrono astrazioni di più alto livello rispetto alle API di basso livello come Win32.

Cos'è una DLL?

Un file DLL, o Libreria di collegamento dinamico, è un tipo di file che contiene codice e dati che possono essere utilizzati da più programmi nello stesso momento. Questi file vengono caricati nel programma che li utilizza mentre il programma è in esecuzione.

Le DLL permettono di condividere il codice tra diversi programmi, evitando così la duplicazione del codice e risparmiando memoria. Ad esempio, se due programmi utilizzano la stessa funzione per stampare un documento, quella funzione può essere inclusa in una DLL e entrambi i programmi possono accedere a quella funzione attraverso la DLL.

Un altro vantaggio delle DLL è che permettono di aggiornare facilmente le funzioni o i dati. Se una funzione in una DLL viene migliorata o corretta, tutti i programmi che utilizzano quella DLL beneficeranno dell'aggiornamento.

Tuttavia, l'uso delle DLL può presentare delle sfide in termini di compatibilità e versionamento, poiché un programma può dipendere da una versione specifica di una DLL.

Come già visto nel passaggio precedente, anche per la consultazione delle librerie possiamo utilizzare CFF Explorer. La voce che andremo a selezionare dal menù, questa volta, sarà però Import Directory.

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

Figura 8

Come possiamo vedere in figura 8, le librerie importate sono KERNEL32.dll e ADVAPI32.dll.

KERNEL32.dll è un elemento vitale del sistema operativo Windows. È una libreria di collegamento dinamico (DLL), che è una raccolta di funzioni utilizzate da vari programmi sul tuo computer.

Nel dettaglio, le operazioni che gestisce KERNEL32.dll sono:

- **Gestione della memoria:** KERNEL32.dll ha un ruolo fondamentale nella gestione della memoria in Windows. Quando Windows si avvia, KERNEL32.dll viene caricato in una zona protetta della memoria per evitare interferenze con altri programmi.
- **Input/Output (I/O):** KERNEL32.dll gestisce le operazioni di input e output, che sono fondamentali per la comunicazione tra il computer e l'hardware esterno, come la tastiera, il mouse o la stampante.
- **Gestione dei file:** KERNEL32.dll è anche responsabile della gestione dei file. Questo include la creazione, la lettura e la scrittura di file sul disco rigido.
- **Comunicazione con l'hardware:** KERNEL32.dll facilita la comunicazione tra i programmi e l'hardware del computer. Questo è essenziale per le operazioni come l'invio di comandi alla CPU o la lettura dei dati dalla RAM.

Se KERNEL32.dll manca o è danneggiato, potrebbero riscontrarsi vari problemi, poiché molti programmi dipendono da esso per funzionare correttamente. Ad esempio, potresti vedere messaggi di errore che indicano che KERNEL32.dll non può essere trovato o inizializzato.

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

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00007632	00007632	0295	SizeofResource
00007644	00007644	01D5	LockResource
00007654	00007654	01C7	LoadResource

Figura 9

Dalla Figura 9, possiamo notare che dalla libreria Kernel32.dll vengono importate ben 51 funzioni, tuttavia quelle più interessanti sono le 3 evidenziate in rosso, vediamole nel dettaglio:

- **SizeofResource:** Questa funzione recupera la dimensione, in byte, della risorsa specificata.
- **LockResource:** Questa funzione recupera un puntatore alla risorsa specificata in memoria. Nonostante il nome, LockResource non blocca effettivamente la memoria; viene usato solo per ottenere un puntatore alla memoria contenente i dati della risorsa.
- **LoadResource:** Questa funzione recupera un handle che può essere usato per ottenere un puntatore al primo byte della risorsa specificata in memoria

ADVAPI32.dll è una libreria di collegamento dinamico (DLL) fondamentale nei sistemi Windows, che fornisce funzioni essenziali per sicurezza, autenticazione e controllo degli accessi. Agisce come ponte tra le applicazioni e i meccanismi di sicurezza sottostanti del sistema operativo. Tra le funzionalità, di particolare importanza è il controllo dell'accesso al registro, che permette la gestione delle autorizzazioni di accesso per le chiavi e i valori del registro, garantendo la sicurezza dei dati.

Come possiamo notare dalla figura 10 le funzioni utilizzate, importate da ADVAPI32.DLL sono:

- **RegSetValueExA :** Questa funzione imposta i dati e il tipo di un valore specificato in una chiave del Registro di sistema.

- **RegCreateKeyExA** : Questa funzione crea una specifica chiave del Registro di sistema. Se la chiave esiste già, la funzione la apre.

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

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

Figura 10

Conclusioni

Guardando nel dettaglio le funzioni importate dalle singole librerie, possiamo ipotizzare che il malware modifichi il registro di sistema di Windows, creando delle nuove chiavi e assegnando loro un valore, in questo modo ipotizziamo che il malware riesca ad ottenere la persistenza nel nostro sistema operativo.

Inoltre, tra le funzioni importate dalla libreria Kernel32.dll troviamo le più comuni per la realizzazione di un dropper, vediamo nel dettaglio cosa fa.

Dropper

Un “dropper” è un tipo di programma creato per installare malware, virus, o aprire una backdoor su un sistema. Il codice del malware può essere contenuto all'interno del dropper stesso (detto dropper a fase singola) per evitare il rilevamento da parte degli antivirus, oppure il dropper, una volta attivato, può scaricare il malware nel sistema target (detto dropper a fase doppia). Inoltre, esistono due tipi principali di dropper; Alcuni non richiedono l'interazione dell'utente e sfruttano un exploit, un codice che si serve di una vulnerabilità del sistema; Altri richiedono l'interazione dell'utente convincendo la vittima che si tratta di un programma benevolo. Inoltre, un dropper che installa un malware solo nella memoria è chiamato talvolta “injector”. In sostanza, un dropper è un piccolo programma di supporto che facilita la consegna e l'installazione di malware.

Obiettivi Giorno 2:

Con riferimento al Malware in analisi, spiegare:

1. Lo scopo della funzione chiamata alla locazione di memoria 00401021
2. Come vengono passati i parametri alla funzione alla locazione 00401021;
3. Che oggetto rappresenta il parametro alla locazione 00401017
4. Il significato delle istruzioni comprese tra gli indirizzi 00401027 e 00401029.
5. Con riferimento all'ultimo quesito, tradurre il codice Assembly nel corrispondente costruito C.
6. Valutate ora la chiamata alla locazione 00401047, qual è il valore del parametro «ValueName»?
7. Nel complesso delle due funzionalità appena viste, spiegate quale funzionalità sta implementando il Malware in questa sezione.

2.1 Spiegare lo scopo della funzione chiamata alla locazione di memoria 00401021:

```
.text:00401000 hObject      = dword ptr -4
.text:00401000 lpData       = dword ptr  8
.text:00401000 cbData       = dword ptr  0Ch
.text:00401000
.text:00401000      push     ebp
.text:00401001      mov      ebp, esp
.text:00401003      push     ecx
.text:00401004      push     0             ; lpdwDisposition
.text:00401006      lea      eax, [ebp+hObject]
.text:00401009      push     eax            ; phkResult
.text:0040100A      push     0             ; lpSecurityAttributes
.text:0040100C      push     0F003Fh        ; samDesired
.text:00401011      push     0             ; dwOptions
.text:00401013      push     0             ; lpClass
.text:00401015      push     0             ; Reserved
.text:00401017      push     offset SubKey   ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"...
.text:0040101C      push     80000002h       ; hKey
.text:00401021      call     ds:RegCreateKeyExA
```

Figura 11

Per rispondere ai quesiti del giorno 2 dobbiamo innanzitutto aprire con IDA Pro il file Malware_Build_Week_U3, presente nella cartella Build_Week_Unit_3 situata nel Desktop della macchina Windows XP dedicata al Malware Analysis.

Una volta fatto ciò andiamo ad individuare l'indirizzo di memoria "00401021" come da riferimento in figura 11 e notiamo che a tale indirizzo è richiamata la funzione "RegCreateKeyExA".

RegCreateKeyExA

La funzione "RegCreateKeyExA" è una funzione che fa parte della libreria winreg.h; ha lo scopo di creare la chiave del Registro di sistema specificata. Se la chiave esiste già, la funzione la apre. Si noti che i nomi delle chiavi non fanno distinzione tra maiuscole e minuscole.

In figura 12 possiamo trovare la sintassi della funzione "RegCreateKeyExA".

Tutti quelli contrassegnati dal riquadro [in] sono parametri che la funzione accetta in input, mentre quelli col riquadro [out] sono parametri che la funzione restituisce in output.

Il riquadro [optional] sta ad indicare parametri opzionali che non sono strettamente necessari al corretto svolgimento delle operazioni della funzione pertanto si può optare per non includerli.

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

Figura 12

2.2 Come vengono passati i parametri alla funzione alla locazione 00401021:

```

.text:00401000 hObject      = dword ptr -4
.text:00401000 lpData      = dword ptr  8
.text:00401000 cbData      = dword ptr 0Ch
.text:00401000
.text:00401000          push    ebp
.text:00401001          mov     ebp, esp
.text:00401003          push    ecx
.text:00401004          push    0             ; lpdwDisposition
.text:00401006          lea     eax, [ebp+hObject]
.text:00401009          push    eax             ; phkResult
.text:0040100A          push    0             ; lpSecurityAttributes
.text:0040100C          push    0F003Fh         ; sanDesired
.text:00401011          push    0             ; dwOptions
.text:00401013          push    0             ; lpClass
.text:00401015          push    0             ; Reserved
.text:00401017          push    offset SubKey    ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"..
.text:0040101C          push    80000002h       ; hKey
.text:00401021          call   ds:RegCreateKeyEx

```

Figura 13

Come possiamo osservare nella figura 13 è stata evidenziata con un riquadro rosso la porzione di codice che si occupa di passare i parametri alla funzione "RegCreateKeyEx".

Tra l'indirizzo di memoria 00401001 e 00401003 troviamo le istruzioni che creano lo stack di memoria della funzione, mentre nelle righe successive fino all'indirizzo 0040101C troviamo una serie di istruzioni "push" che spostano in cima allo stack appena creato i parametri della funzione. A destra delle istruzioni push è possibile trovare ulteriori dettagli sui singoli parametri che ci informano dell'identità del singolo parametro.

Dopo che tutti i parametri sono stati correttamente caricati nello stack troviamo la chiamata alla funzione "RegCreateKeyExA" che effettuerà delle operazioni con i parametri assegnati.

2.3 Che oggetto rappresenta il parametro alla locazione 00401017:

Il parametro alla locazione 00401017 rappresenta il puntatore ad una sottochiave di una chiave identificata dal parametro "hKey".

Nel caso della funzione "RegCreateKeyExA" all'indirizzo di memoria 00401021, il parametro lpSubkey punta alla sottochiave:

<< "SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" >>

che è una sottochiave della chiave di registro "HKEY_LOCAL_MACHINE" individuata dal parametro "hKey".

Il parametro "hKey" è un handle per una chiave di registro aperta. Il processo chiamante deve avere accesso "KEY_CREATE_SUB_KEY" alla chiave.

Questo handle viene restituito dalla funzione "RegCreateKeyEx" o "RegOpenKeyEx" oppure può essere uno dei seguenti tasti predefiniti:

HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS

lpSubKey

Il parametro "lpSubKey" individua il nome di una sottochiave aperta o creata da questa funzione. La sottochiave specificata deve essere una sottochiave della chiave identificata dal parametro "hKey"; può essere fino a 32 livelli di profondità nell'albero del registro.

Se "lpSubKey" è un puntatore a una stringa vuota, "phkResult" riceve un nuovo handle per la chiave specificata da "hKey".

Questo parametro non può essere NULL.

2.4 Spiegare il blocco delle istruzioni comprese tra gli indirizzi 00401027 e 00401029:

```
.text:00401027      test     eax, eax
.text:00401029      jz       short loc_401032
.text:0040102B      mov     eax, 1
.text:00401030      jmp     short loc_40107B
.text:00401032      ; -----
.text:00401032      loc_401032:
.text:00401032      ; CODE XREF: sub_401000+291j
.text:00401032      mov     ecx, [ebp+cbData]
.text:00401035      push    ecx                ; cbData
.text:00401036      mov     edx, [ebp+lpData]
.text:00401039      push    edx                ; lpData
.text:0040103A      push    1                  ; dwType
.text:0040103C      push    0                  ; Reserved
.text:0040103E      push    offset ValueName ; "GinaDLL"
.text:00401043      mov     eax, [ebp+hObject]
.text:00401046      push    eax                ; hKey
.text:00401047      call    ds:RegSetValueExA
.text:0040104D      test    eax, eax
.text:0040104F      jz       short loc_401062
.text:00401051      mov     ecx, [ebp+hObject]
.text:00401054      push    ecx                ; hObject
.text:00401055      call    ds:CloseHandle
.text:0040105B      mov     eax, 1
.text:00401060      jmp     short loc_40107B
.text:00401062      ; -----
```

Figura 14

All'indirizzo di memoria 00401029 troviamo un'istruzione test, vediamo la definizione di questa istruzione.

L'istruzione TEST calcola l'AND logico bit per bit del primo operando (operando sorgente 1) e del secondo operando (operando sorgente 2) e imposta i flag di stato SF, ZF e PF in base al risultato. Il risultato viene quindi scartato.

Operazione logica AND

Vediamo più in dettaglio il calcolo dell'AND logico per capire a fondo il funzionamento dell'istruzione test.

L'operazione AND bitwise è un'operazione binaria che prende in input due rappresentazioni binarie di uguale lunghezza ed esegue AND logico su ogni coppia di bit corrispondente.

Ricordiamo che AND logico restituisce 1 solamente quando entrambi i valori comparati sono uguali ad 1, in tutti gli altri casi restituisce 0.

Salto condizionale jz

All'indirizzo di memoria 00401029 troviamo l'istruzione jz short loc_401032.

Jz è un'istruzione facente parte della famiglia dei salti condizionali. Un salto condizionale è un'istruzione che esegue un "salto" ad un indirizzo di memoria indicato solamente dopo aver verificato una condizione, nel caso di jz la condizione da verificare è lo stato di ZF, ovvero del Zero Flag, l'istruzione infatti esegue il salto solamente se ZF=1.

OllyDbg

Con il programma OllyDbg possiamo verificare il contenuto dei registri con precisione al dettaglio della singola istruzione quindi ci aiuteremo col suddetto programma per capire fino in fondo il funzionamento di queste due istruzioni.

OllyDbg è un debugger per sistemi Microsoft Windows, disponibile solo in versione a 32 bit. Traccia i registri, riconosce le funzioni e i parametri passati alle principali librerie standard, le chiamate alle API del sistema operativo, eventuali salti condizionali, tabelle, costanti, variabili e stringhe.

Per caricare il malware su OllyDbg procediamo ad avviare il programma che possiamo trovare all'interno della cartella "odbg110" nel Desktop della macchina Windows XP per il malware analysis. Una volta fatto ciò clicchiamo sull'icona a forma di cartella in alto a

sinistra nel programma e selezioniamo il file `Malware_Build_Week_U3` situato nella cartella `Build_Week_Unit_3` situata a sua volta nel Desktop.

Una volta caricato il file procediamo con l'impostazione di un breakpoint all'indirizzo di memoria `00401027` ed in seguito avviamo il programma. Un breakpoint è essenzialmente un punto che noi indichiamo al programma per dirgli di arrestare l'esecuzione del file quando lo incontra. A questo punto possiamo verificare nella finestra dei registri lo stato del registro `EAX` prima dell'esecuzione dell'istruzione di test.

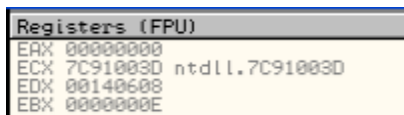


Figura 15

Come possiamo vedere `EAX` equivale a `00000000`, quindi quando l'istruzione `test` verrà eseguita effettuerà l'AND logico tra due istanze di `EAX`, in altre parole comparerà `00000000` con `00000000`. Il risultato della seguente operazione sarà dunque `0` e dunque `ZF` verrà aggiornato al valore `1`.

Siccome `jz` si esegue solamente se `ZF=1` e abbiamo appena appurato che questa condizione è verificata nel nostro caso, possiamo concludere che il salto viene effettuato.

2.5 Con riferimento all'ultimo quesito, tradurre il codice Assembly nel corrispondente costruito C.

Abbiamo già spiegato qual è il comportamento delle due istruzioni agli indirizzi di memoria `00401027` e `00401029`. Vediamo di seguito la traduzione in linguaggio C delle istruzioni appena viste.

```
if (EAX == 0) {
    goto loc_401032;
}
```

Dove:

- `if (<<valore>> == 0) {`
 `goto loc_401032;`
}

Corrisponde all'istruzione `<< jz short loc_401032 >>`, che effettua il salto quando lo `ZF` ha valore `1`.

- `EAX` è il registro che si controlla utilizzando l'istruzione `<< test EAX, EAX >>` che imposta lo `ZF` a `1` solo quando il registro è uguale a `0`.

2.6 Valutate ora la chiamata alla locazione 00401047, qual' è il valore del parametro «ValueName»?

```

.text:00401027      test     eax, eax
.text:00401029      jz      short loc_401032
.text:0040102B      mov     eax, 1
.text:00401030      jnp     short loc_40107B
.text:00401032      ; -----
.text:00401032      loc_401032:      ; CODE XREF: sub_401000+291j
.text:00401032      mov     ecx, [ebp+cbData]
.text:00401035      push    ecx      ; cbData
.text:00401036      mov     edx, [ebp+lpData]
.text:00401039      push    edx      ; lpData
.text:0040103A      push    1        ; dwType
.text:0040103C      push    0        ; Reserved
.text:0040103E      push    offset ValueName ; "GinaDLL"
.text:00401043      mov     eax, [ebp+hObject]
.text:00401046      push    eax      ; hKey
.text:00401047      call    ds:RegSetValueExA
.text:00401048      test     eax, eax
.text:0040104F      jz      short loc_401062
.text:00401051      mov     ecx, [ebp+hObject]
.text:00401054      push    ecx      ; hObject
.text:00401055      call    ds:CloseHandle
.text:00401058      mov     eax, 1
.text:00401060      jnp     short loc_40107B
.text:00401062      ; -----

```

Figura 16

All'indirizzo di memoria 00401047 possiamo vedere la chiamata alla funzione "RegSetValueExA" facente parte della libreria "winreg.h".

Questa funzione ha come scopo quello di impostare i dati e il tipo di un valore specificato in una chiave di registro.

C++

```

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

```

Figura 17



La figura 17 mostra la sintassi della funzione in questione e mostra anche i vari parametri che la funzione accetta in input.

In figura 16 possiamo vedere che all'indirizzo 0040103E viene passato il parametro "lpValueName" con un valore preciso, ovvero "GinaDLL".

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

Le funzioni "RegSetValueExA" e "RegCreateKeyExA" sono entrambe parte dell'API del Registro di sistema di Windows e vengono comunemente utilizzate per manipolare le voci e le chiavi del registro. Tuttavia, possono anche essere utilizzate in modo dannoso da malware per persistere nel sistema e modificare le impostazioni del registro per scopi dannosi come il furto di informazioni, l'esecuzione di codice dannoso all'avvio del sistema o la creazione di backdoor nel sistema.

Nel nostro caso, con le due chiamate di funzione viste, il Malware crea una chiave di registro, le assegna come valore Gina.DLL e successivamente la sovrascrive a quella originale.

GINA

GINA (Graphical identification and authentication) è un componente di alcuni sistemi operativi Windows che fornisce servizi di autenticazione sicura e di accesso interattivo. Si tratta di una libreria sostituibile collegata dinamicamente che viene caricata all'inizio del processo di avvio nel contesto di Winlogon all'avvio della macchina; è inoltre responsabile dell'avvio dei processi iniziali per un utente (come la shell di Windows) al primo accesso.

Ipoteticamente, quindi, il Malware sta cercando di ottenere persistenza in modo da essere eseguito all'avvio del sistema operativo, così da ricavare informazioni di login attraverso il controllo della shell grafica.

Obiettivi Giorno 3:

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

1. Qual è il valore del parametro «ResourceName» passato alla funzione FindResourceA()?
2. Il susseguirsi delle chiamate di funzione che effettua il Malware in questa sezione di codice l'abbiamo visto durante le lezioni teoriche. Che funzionalità sta implementando il Malware?
3. È possibile identificare questa funzionalità utilizzando l'analisi statica basica?
4. In caso di risposta affermativa, elencare le evidenze a supporto.
5. 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.

3.1 Qual è il valore del parametro <<ResourceName>> passato alla funzione FindResourceA()?

Prima di rispondere alla domanda è opportuno prima conoscere il funzionamento di FindResourceA().

FindResourceA()

La funzione FindResourceA() è una funzione dell'API Win32 che localizza una risorsa con un tipo e un nome specifici all'interno di un modulo specificato. Questa funzione serve per determinare la posizione dei dati delle risorse binarie, come le icone, i bitmap, le stringhe, etc.

La sintassi della funzione è la seguente:

```
HRSRC FindResourceA(  
    [in, optional] HMODULE hModule,  
    [in] LPCSTR lpName,  
    [in] LPCSTR lpType  
);
```

Dove:

- hModule (opzionale) : È un handle per il modulo che contiene la risorsa. Se questo parametro è NULL, la funzione cerca nel modulo usato per creare il processo corrente.
- lpName : È il nome della risorsa. Questo parametro può essere un puntatore a una stringa o un identificatore intero della risorsa.
- lpType: È il tipo di risorsa. Questo parametro può essere un puntatore a una stringa o un identificatore intero del tipo di risorsa.

Il valore restituito dalla funzione è un handle per il blocco di informazioni della risorsa specificata. Per ottenere un handle per la risorsa, si deve passare questo handle alla funzione LoadResource. Se la funzione fallisce, il valore è NULL.

Ora, una volta a conoscenza di questo possiamo utilizzare OllyDBG per localizzare la routine tra le locazioni di memoria 00401080 e 00401128. Per farlo possiamo premere sull'icona nera in alto, come indicato in figura 18 oppure semplicemente usare la combinazione di tasti CTRL+G.

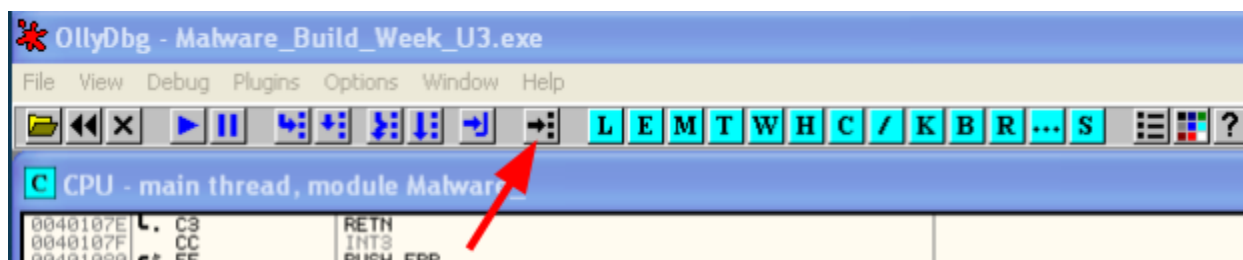


Figura 18

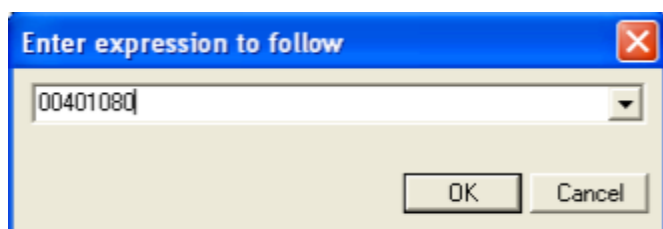


Figura 19

In questo modo si aprirà una finestra di dialogo dove è possibile inserire l'indirizzo di memoria che si vuole raggiungere, come indicato in figura 19, nel nostro caso andiamo ad inserire l'indirizzo 00401080 per iniziare la nostra analisi.

A questo punto, localizzato l'indirizzo di memoria di partenza possiamo cercare la funzione FindResourceA() e andare a ritroso fino a trovare il parametro ResourceName.

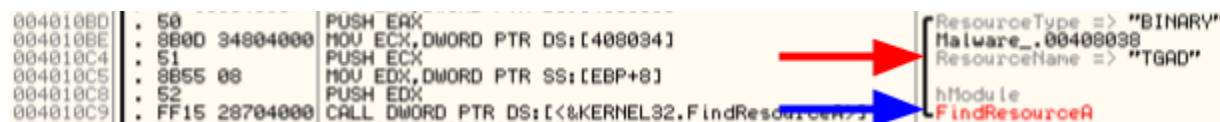


Figura 20

Come mostrato in Figura 20, la funzione FindResourceA è presente all'indirizzo di memoria 004010C9, indicato da una freccia blu; i parametri di FindResourceA sono indicati prima della funzione stessa e notiamo nella parte sinistra di OllyDBG che sono "tradotti" in modo da renderli leggibili. Troviamo quindi il valore di ResourceName, modificato all'indirizzo di memoria 004010BE e reso uguale a **TGAD**. Possiamo inoltre aggiungere che ResourceName è il nome della risorsa che la funzione proverà a trovare, è quindi il parametro **lpName** della funzione FindResourceA.

3.2 Che funzionalità sta implementando il Malware?

<pre> 00401080 \$ 55 PUSH EBP 00401081 . 8BEC MOV EBP,ESP 00401083 . 83EC 18 SUB ESP,18 00401086 . 56 PUSH ESI 00401087 . 57 PUSH EDI 00401088 . C745 EC 000000 MOV DWORD PTR SS:[EBP-14],0 0040108F . C745 E8 000000 MOV DWORD PTR SS:[EBP-18],0 00401096 . C745 F8 000000 MOV DWORD PTR SS:[EBP-8],0 0040109D . C745 F0 000000 MOV DWORD PTR SS:[EBP-10],0 004010A4 . C745 F4 000000 MOV DWORD PTR SS:[EBP-C],0 004010AB . 837D 08 00 CMP DWORD PTR SS:[EBP+8],0 004010AF . 75 07 JNZ SHORT Malware_.004010B8 004010B1 . 33C0 XOR EAX,EAX 004010B3 . E9 07010000 JMP Malware_.004011BF 004010B8 > A1 30804000 MOV EAX,DWORD PTR DS:[408030] 004010BD . 50 PUSH EAX 004010BE . 8B0D 34804000 MOV ECX,DWORD PTR DS:[408034] 004010C4 . 51 PUSH ECX 004010C5 . 8B55 08 MOV EDX,DWORD PTR SS:[EBP+8] 004010C8 . 52 PUSH EDX 004010C9 . FF15 28704000 CALL DWORD PTR DS:[<&KERNEL32.FindResourceA>] 004010CF . 8945 EC MOV DWORD PTR SS:[EBP-14],EAX 004010D2 . 837D EC 00 CMP DWORD PTR SS:[EBP-14],0 004010D6 . 75 07 JNZ SHORT Malware_.004010DF 004010D8 . 33C0 XOR EAX,EAX 004010DA . E9 E0000000 JMP Malware_.004011BF 004010DF > 8B45 EC MOV EAX,DWORD PTR SS:[EBP-14] 004010E2 . 50 PUSH EAX 004010E3 . 8B4D 08 MOV ECX,DWORD PTR SS:[EBP+8] 004010E6 . 51 PUSH ECX 004010E7 . FF15 14704000 CALL DWORD PTR DS:[<&KERNEL32.LoadResource>] 004010ED . 8945 E8 MOV DWORD PTR SS:[EBP-18],EAX 004010F0 . 837D E8 00 CMP DWORD PTR SS:[EBP-18],0 004010F4 . 75 05 JNZ SHORT Malware_.004010FB 004010F6 . E9 AA000000 JMP Malware_.004011A5 004010FB > 8B55 E8 MOV EDX,DWORD PTR SS:[EBP-18] 004010FE . 52 PUSH EDX 004010FF . FF15 10704000 CALL DWORD PTR DS:[<&KERNEL32.LockResource>] 00401105 . 8945 F8 MOV DWORD PTR SS:[EBP-8],EAX 00401108 . 837D F8 00 CMP DWORD PTR SS:[EBP-8],0 0040110C . 75 05 JNZ SHORT Malware_.00401113 0040110E . E9 92000000 JMP Malware_.004011A5 00401113 > 8B45 EC MOV EAX,DWORD PTR SS:[EBP-14] 00401116 . 50 PUSH EAX 00401117 . 8B4D 08 MOV ECX,DWORD PTR SS:[EBP+8] 0040111A . 51 PUSH ECX 0040111B . FF15 0C704000 CALL DWORD PTR DS:[<&KERNEL32.SizeofResource>] 00401121 . 8945 F0 MOV DWORD PTR SS:[EBP-10],EAX 00401124 . 837D F0 00 CMP DWORD PTR SS:[EBP-10],0 00401128 . 77 02 JA SHORT Malware_.0040112C </pre>	<pre> ResourceType => "BINARY" Malware_.00408038 ResourceName => "TGAD" hModule FindResourceA hResource hModule LoadResource nHandles SetHandleCount hResource hModule SizeofResource </pre>
---	--

Figura 21

Nell'estratto del codice assembly recuperato attraverso OllyDbg, che possiamo vedere in Figura 21, notiamo il susseguirsi delle funzioni colorate di rosso.

Queste funzioni fanno parte dell'API di Windows e hanno utilizzi legittimi nelle applicazioni:

- FindResource: Già spiegata in precedenza [qua](#).
- LoadResource: Questa funzione recupera un handle che può essere usato per ottenere un puntatore al primo byte della risorsa specificata in memoria.
- LockResource: Questa funzione recupera un puntatore alla risorsa specificata in memoria. Nonostante il nome, LockResource non blocca effettivamente la memoria; viene usato solo per ottenere un puntatore alla memoria contenente i dati della risorsa.
- SizeofResource: Questa funzione recupera la dimensione, in byte, della risorsa specificata.

Potenziati rischi per la sicurezza

Le funzioni appena viste possono essere utilizzate impropriamente da autori di malware per scopi dannosi, per esempio:

- Scaricare ed eseguire codice dannoso: Il malware potrebbe utilizzare FindResource e LoadResource per accedere e caricare codice dannoso aggiuntivo incorporato nello stesso eseguibile o in un file di risorse separato.
- Nascondere contenuti dannosi: Le risorse possono essere utilizzate per nascondere codice o dati dannosi dagli strumenti di analisi statica, rendendo la loro individuazione più difficile.
- Sfruttare vulnerabilità: In rari casi, gli aggressori potrebbero sfruttare vulnerabilità nelle librerie di analisi delle risorse per ottenere l'esecuzione del codice o l'escalation dei privilegi.

Il loro susseguirsi ci induce quindi a pensare che, al momento dell'esecuzione, il programma vada a cercare e ad estrarre dalla sua sezione risorse (sezione .rsrc, che abbiamo già avuto modo di discutere durante il primo giorno) un componente, probabilmente un altro Malware. Il componente malevolo verrà estratto dal parametro TAGD della funzione FindResourceA.

3.3 - 3.4 È possibile identificare questa funzionalità utilizzando l'analisi statica basica? Elencare le evidenze a supporto.

Come già visto nelle conclusioni del report del primo giorno, è già stato possibile con la sola analisi statica basica identificare le funzionalità di dropper e persistenza del malware.

3.5 Diagramma di Flusso

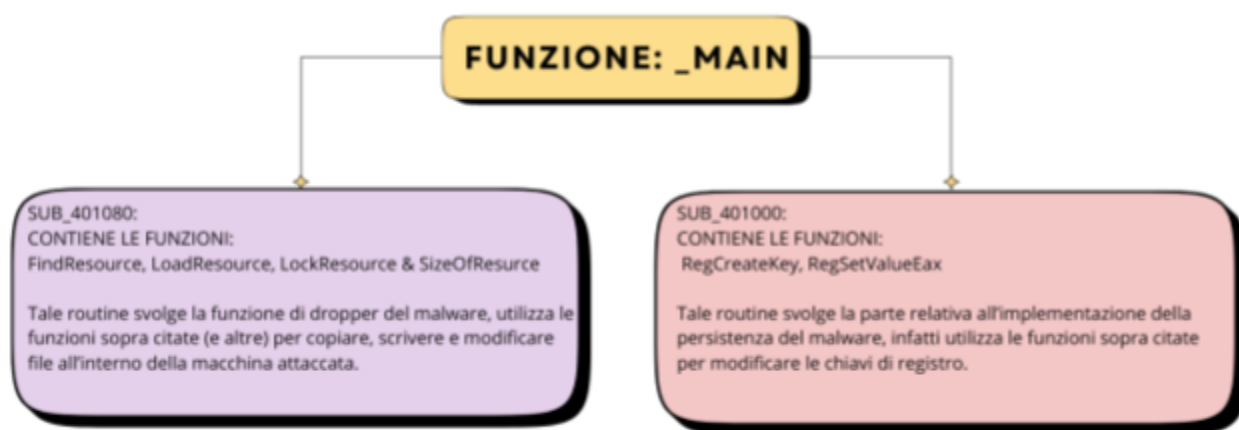


Grafico 1

Obiettivi Giorno 4:

Preparare 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.

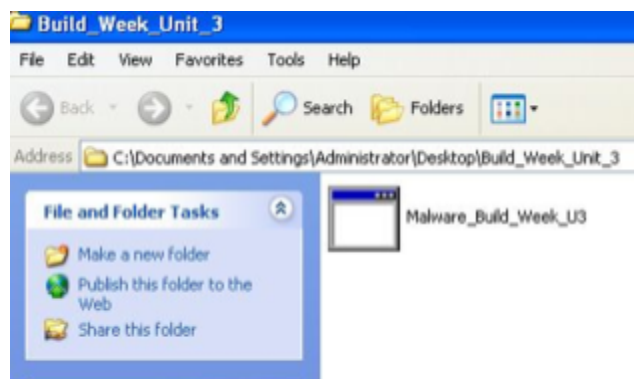


Figura 22

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

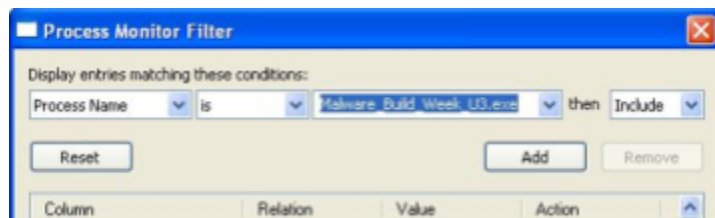


Figura 23

Analizzate ora i risultati di Process Monitor (consiglio: utilizzate il filtro come in figura sotto per estrarre solo le modifiche apportate al sistema da parte del Malware). Fate click su «ADD» poi su «Apply» come abbiamo visto nella lezione teorica e come mostrato in figura 23.

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.

4.0 Preparazione dell'ambiente di lavoro.

Per svolgere l'esercizio di oggi è necessario fare analisi dinamica avanzata.

L'analisi dinamica avanzata presuppone che sia fatto eseguire il file malware sulla macchina Windows XP dedicata a malware analysis, è dunque opportuno prendere alcune precauzioni per poter contenere i danni collaterali che possono essere causati da alcuni malware particolarmente distruttivi a prescindere dal malware specifico che andremo ad eseguire oggi siccome non ne conosciamo ancora completamente il comportamento.

Creazione di un'istantanea

Prima di eseguire il malware andiamo a creare un'istantanea della nostra macchina.

Questa operazione ci permette di "salvare" lo stato della macchina in un punto preciso e di poterla riportare a quel punto qual'ora si dovesse verificare un evento catastrofico, come ad esempio la corruzione dei file di sistema. Nel nostro caso creare un'istantanea ci permette di ritornare ad un punto nel tempo in cui non abbiamo mai avviato il malware che andremo ad analizzare oggi.

Per creare un'istantanea di una macchina apriamo il gestore di Oracle Virtual Box e clicchiamo sull'icona di menù accanto alla nostra macchina Windows XP, in seguito clicchiamo sull'opzione "Dettagli" che ci verrà proposta dal menù a tendina che si aprirà.

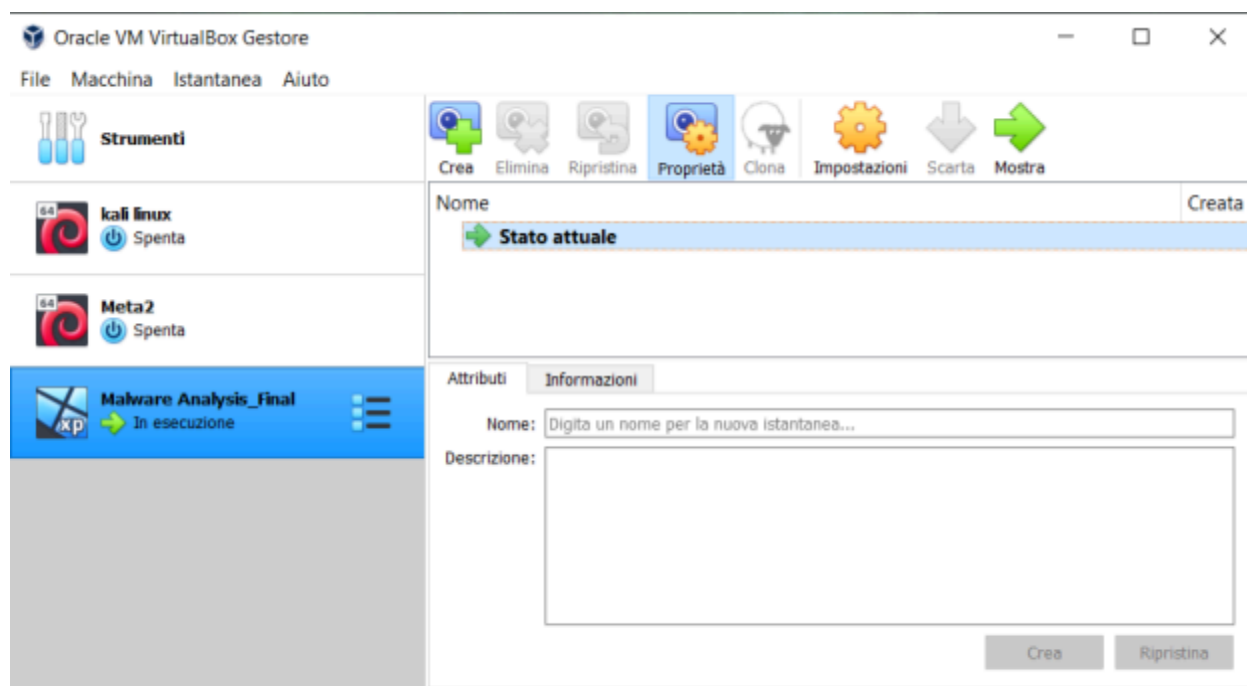


Figura 24

Fatto ciò premiamo una volta sul pulsante “Crea” in alto per creare un’istantanea della macchina.

Abbiamo così creato un’ istantanea della macchina e possiamo procedere con la preparazione del resto degli strumenti che ci serviranno per l’analisi di oggi.

ProcMon Setup

Avviamo la macchina e successivamente avviamo il programma “ProcMon” situato all’interno della cartella “ProcessMonitor” situata nel Desktop.

Ci assicuriamo anche di cliccare sul tasto “Reset” presente nella schermata che ci viene mostrata dopo aver aperto il programma così da eliminare eventuali precedenti impostazioni di filtraggio dei processi.

In questo momento Procmon starà già catturando informazioni relative ai processi attivi.

A questo punto facciamo eseguire il file “Malware_Build_Week_U3” presente all’interno della cartella “Build_Week_Unit_3” nel nostro Desktop e cominciamo ad analizzare il malware.

4.1 Cosa notate all'interno della cartella dove è situato l'eseguibile del Malware?

Quando avviamo il malware, notiamo subito che, all'interno della cartella in cui è contenuto il suo eseguibile, si viene a creare il file msgina32.dll. In accordo con quelle erano le nostre deduzioni circa la natura del malware: un dropper, possiamo quindi affermare la loro correttezza. La sua prima azione è stata infatti quella di identificare ed estrarre la risorsa msgina32.dll dalla sezione risorse (.rsrc) denominata TGAD (visto nel giorno 2).

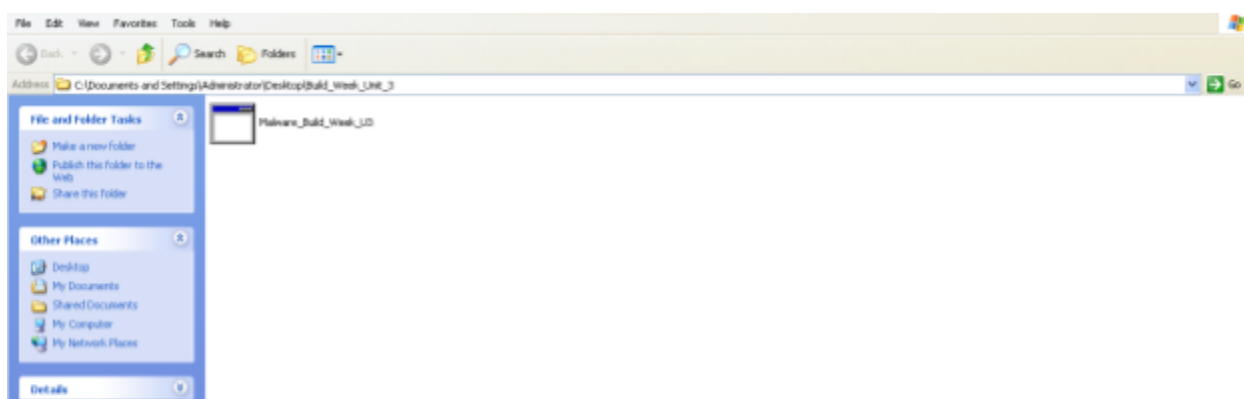


Figura 25

Di seguito, vediamo la cartella del malware prima della sua esecuzione e subito dopo, rispettivamente in figura 25 e 26.

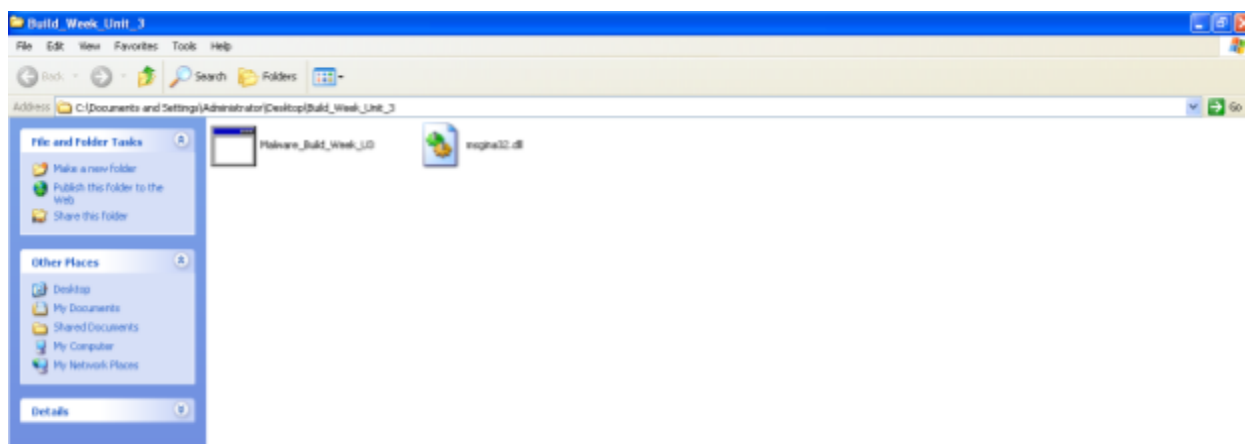


Figura 26

4.2 Quale chiave di registro viene creata?

Per controllare gli eventi riguardanti le chiavi di registro, avviamo innanzitutto il malware cliccando due volte sul suo file eseguibile presente nella cartella Build_Week_Unit_3 sul Desktop del nostro ambiente controllato. Entriamo successivamente all'interno di Procmon e filtriamo i processi relativi soltanto al malware. Per farlo clicchiamo sull'icona a forma di imbuto dal menù in alto (o, analogamente, scegliamo la voce Filter dallo stesso menù, o ancora utilizziamo la combinazione di tasti CTRL+ L); si aprirà una finestra di dialogo dalla quale sarà possibile scegliere di filtrare i processi per nome. Adottiamo la combinazione *Process Name is Nome del processo* e clicchiamo su Add - come in Figura 27. A questo punto il programma mostrerà soltanto gli eventi relativi al malware.

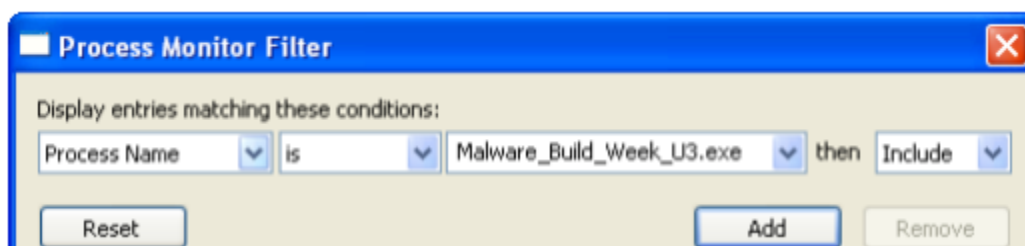


Figura 27

Per esaminare soltanto le variazioni riguardanti le chiavi di registro, selezioniamo l'apposita icona sempre contenuta nel menù in alto (riquadro blu in figura 28).

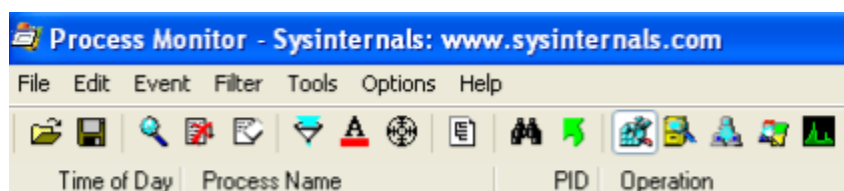


Figura 28

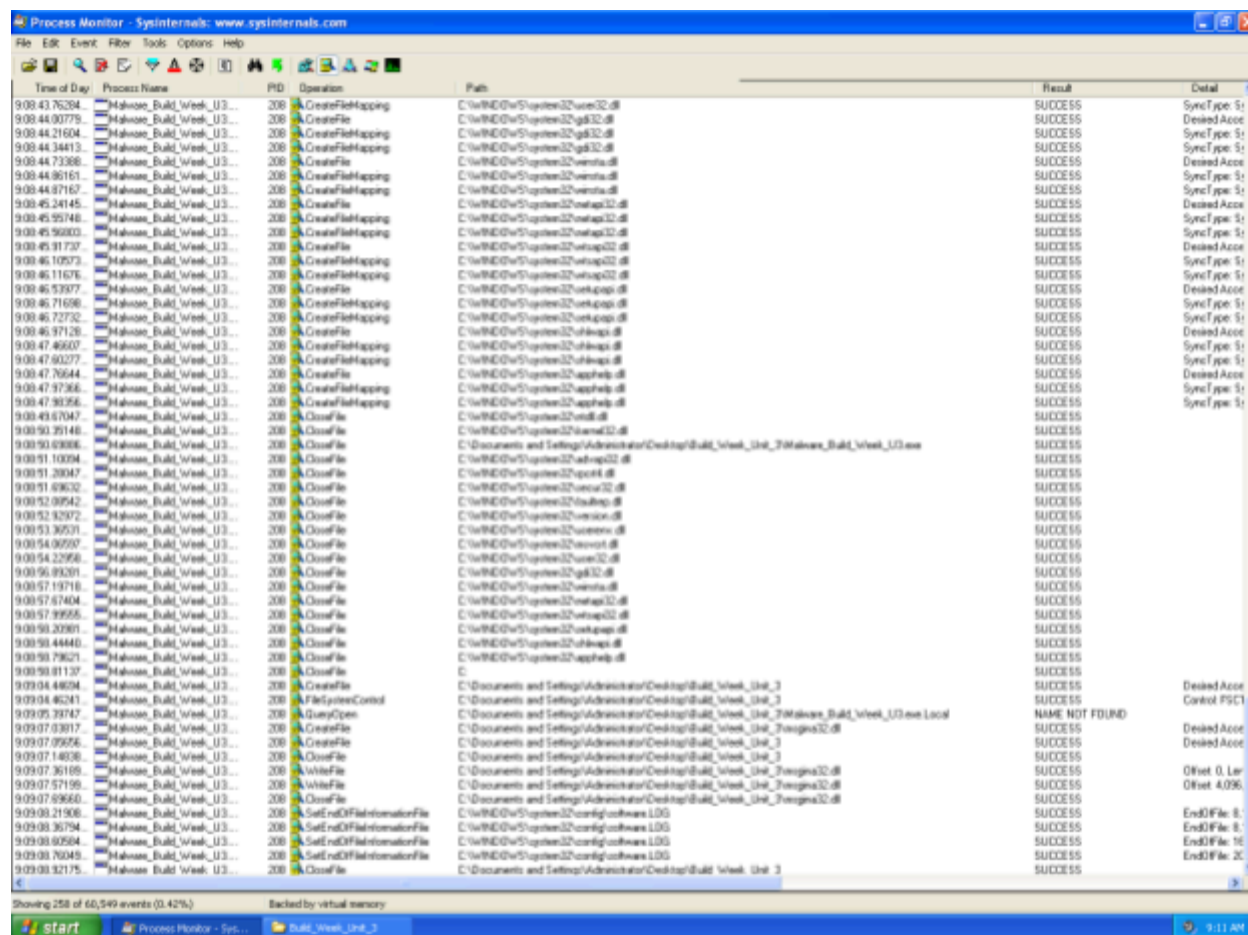
[illegible]

Malware_Build_Week_U3...	208	RegCreateKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
Malware_Build_Week_U3...	208	RegSetValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL
Malware_Build_Week_U3...	208	RegCloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

SUCCESS Desired Access: All Access
SUCCESS Type: REG_SZ, Length: 520, Data: C:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.d ←

C:\DocumentsandSettings\Administrator\Desktop\Build Week Unit 3\msgina32.dll.

Questa è la funzione che crea il file msgina32.dll all'interno della cartella del Malware originale a valle della sua estrazione.



Time of Day	Process Name	PID	Operation	Path	Result	Detail
9:00:43.76294	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:44.00779	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:44.21604	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:44.34413	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:44.73386	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:44.95161	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:44.97167	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:45.24145	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:45.55748	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:45.59803	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:45.91737	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:46.10573	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:46.11676	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:46.53977	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:46.71696	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:46.72732	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:46.97128	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:47.46607	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:47.50277	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:47.76644	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:47.97366	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:47.98366	Malware_Bulk_Wireless_U3...	208	CreateFileMapping	C:\WINDOWS\system32\msgina32.dll	SUCCESS	SynType: S1
9:00:48.67047	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:50.30146	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:50.60886	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:51.10094	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:51.20047	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:51.69632	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:52.00542	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:52.52972	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:53.30531	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:54.06257	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:54.22966	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:56.99301	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:57.19718	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:57.67404	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:57.99956	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:58.20801	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:58.44440	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:58.79621	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:58.81137	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:58.44804	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:58.46241	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:58.93747	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:59.30177	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:59.39556	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:59.71436	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:59.76109	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:59.77199	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:59.78860	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:59.81906	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:59.85794	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:59.89594	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:59.90419	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access
9:00:59.92175	Malware_Bulk_Wireless_U3...	208	CreateFile	C:\WINDOWS\system32\msgina32.dll	SUCCESS	Desired Access

Figura 32

In conclusione, unendo i risultati ottenuti con l'analisi statica e quella dinamica, possiamo delineare il seguente comportamento del malware:

Identifica nella sezione risorse chiamata TGAD il file msgina32.dll. Successivamente, lo copia e importa nella cartella dove è situato l'eseguibile del malware. Affinché questo avvenga, quest'ultimo utilizza le già discusse funzioni per l'interazione con il file system: *CreateFile* e *WriteFile*.

Una volta completato questo lavoro, crea la chiave di registro *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon* e le assegna il valore *Gina.dll*.

Obiettivi Giorno 5:

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

1. Cosa può succedere se il file .dll lecito viene sostituito con un file .dll malevolo, che intercetta i dati inseriti?
2. Sulla base della risposta sopra, delineate il profilo del Malware e delle sue funzionalità. Unite tutti i punti per creare un grafico che ne rappresenti lo scopo ad alto livello.

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

DLL Malevoli

Se un file DLL legittimo viene sostituito con un file DLL malevolo che intercetta i dati inseriti, possono verificarsi diverse conseguenze dannose ad esempio:

Furto di dati sensibili: i file DLL dannosi possono quindi intercettare i dati immessi dall'utente, come nome utente, password, informazioni sulla carta di credito o altri dati personali sensibili. Queste informazioni dunque possono essere inviate a un server controllato da un utente malintenzionato per il furto di identità o altri scopi fraudolenti.

Attacchi di phishing: i dati intercettati possono essere utilizzati per condurre attacchi di phishing mirati, inviando agli utenti e-mail o messaggi fraudolenti per indurli a ottenere ulteriori informazioni personali o finanziarie.

Compromettere la sicurezza del sistema: l'inserimento di un file DLL dannoso potrebbe consentire a un utente malintenzionato di compromettere il sistema, consentendogli di eseguire codice dannoso, installare malware aggiuntivo o ottenere privilegi elevati sul sistema.

GINA.dll Malevolo

Il file GINA.dll è una parte importante del sistema operativo Windows poiché gestisce l'autenticazione dell'utente. Se il file legittimo GINA.dll viene sostituito con una versione dannosa progettata per intercettare i dati immessi durante il processo di accesso, possono verificarsi una serie di conseguenze negative, tra cui:

Compromissione della sicurezza: un file dannoso può acquisire e registrare le credenziali dell'utente, inclusi nome utente e password, durante il processo di accesso al sistema operativo. Queste informazioni possono quindi essere utilizzate per ottenere l'accesso non autorizzato al sistema o ad altri sistemi correlati.

Accesso non autorizzato: una volta compromesse le credenziali di un utente, un utente malintenzionato può ottenere un accesso non autorizzato al sistema operativo e ai dati sensibili in esso archiviati. Ciò potrebbe portare al furto di informazioni sensibili o alla compromissione della sicurezza dell'intera rete.

Installazione di altri malware: una volta che il file dannoso GINA.dll è attivo nel sistema, i malintenzionati possono utilizzarlo come punto di ingresso per installare altro malware sul computer. Questi malware possono essere progettati per eseguire varie azioni dannose, come il furto di dati, il danneggiamento di file o il controllo remoto del sistema.

Compromettere l'integrità del sistema: la sostituzione del file GINA.dll con una versione dannosa può compromettere l'integrità del sistema operativo poiché potrebbe influire sul corretto funzionamento dell'autenticazione dell'utente. Ciò può causare problemi di stabilità del sistema o addirittura un arresto anomalo completo del sistema.

5.2 Realizzare un grafico che rappresenti il malware ad alto livello.

Dalle informazioni in nostro possesso siamo riusciti a stilare il seguente grafico:

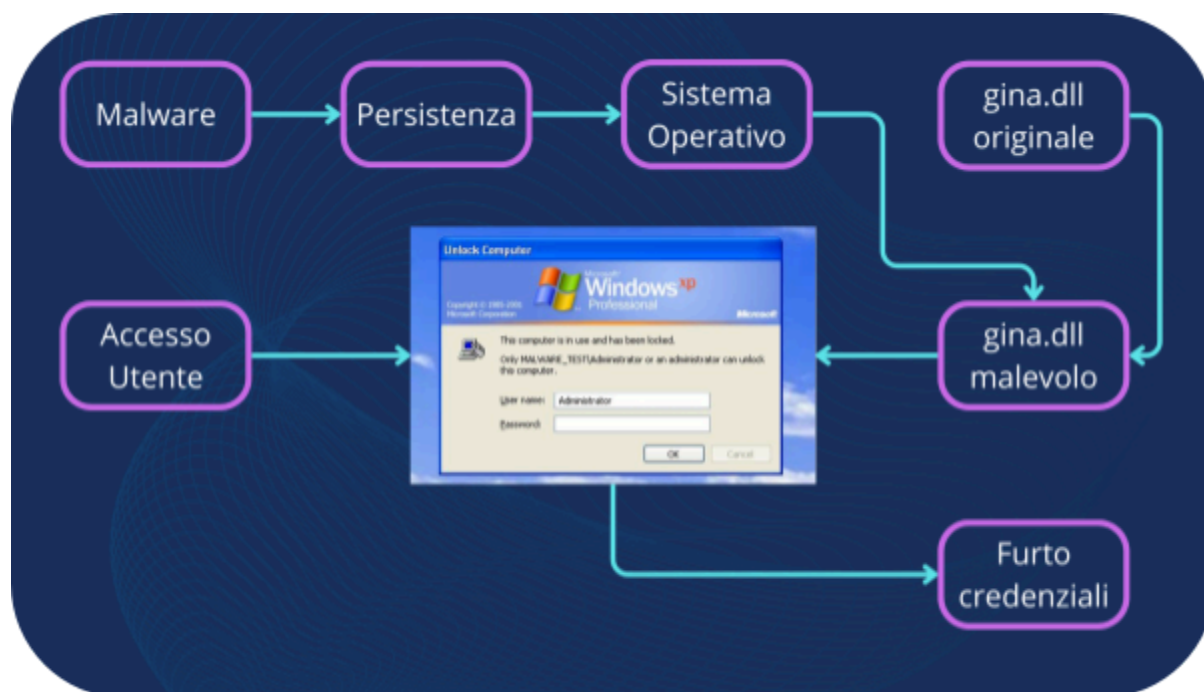


Grafico 2