BUILD WEEK 3

Malware analysis and reverse engineering in practice

Analisi completa di un malware reale

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

GIORNO 1

1) Quanti parametri sono passati alla funzione Main()?

```
argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h
```

argc
Puntatore a un valore dword (parola binaria a 32 bit) che rappresenta il numero di argomenti passati al programma da linea di comando.

argu
Puntatore a un array di puntatori a char (stringhe ASCII) che rappresentano gli argomenti passati al programma da linea di comando.
envp
Puntatore a un array di puntatori a char (stringhe ASCII) che rappresentano le variabili d'ambiente del sistema operativo.

Questi parametri vengono utilizzati dal programma per accedere alle informazioni passate dall'utente al programma al momento dell'esecuzione, come gli argomenti da linea di comando e le variabili d'ambiente.

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

```
hModule= dword ptr -11Ch
Data= byte ptr -118h
var_8= dword ptr -8
var_4= dword ptr -4
```

hModule
Variabile di tipo dword pointer che ha un offset di -11Ch rispetto all'indirizzo di base dello stack.

Data
Variabile di tipo byte pointer che ha un offset di -118h rispetto all'indirizzo di base dello stack.
var_8 e var_4
Variabili di tipo dword pointer che hanno rispettivamente un offset di -8 e -4 rispetto all'indirizzo di base dello stack.

3) Quali sezioni sono presenti all'interno del file eseguibile? Descrivete brevemente almeno 2 di quelle identificate.

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

.text
Contiene le istruzioni (le righe di codice) che la CPU eseguirà una volta che il software sarà avviato.

.rdata
Include generalmente le informazioni circa le librerie e le funzioni importate ed esportate dall'eseguibile, informazione che come abbiamo visto possiamo ricavare con CFF Explorer.

.data
Contiene tipicamente i dati / le variabili globali del programma eseguibile, che devono essere disponibili da qualsiasi parte del programma.

.rsrc
Include le risorse utilizzate dall'eseguibile come ad esempio icone, immagini, menu e stringhe che non sono parte dell'eseguibile stesso.

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.

szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	51	00007534	00000000	00000000	0000769E	0000700C
ADVAPI32.dll	2	00007528	00000000	00000000	000076D0	00007000

Funzioni importate all'interno della libreria **KERNEL32**

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

Funzioni importate all'interno della libreria **ADVAPI32**

Dword	Dword	Word	szAnsi
000076AC	000076AC	0186	RegSetValueExA
000076BE	000076BE	015F	RegCreateKeyExA

KERNEL32.DLL
Libreria che contiene le funzioni principali per interagire col sistema operativo, come per esempio la manipolazione di file e la gestione della memoria.

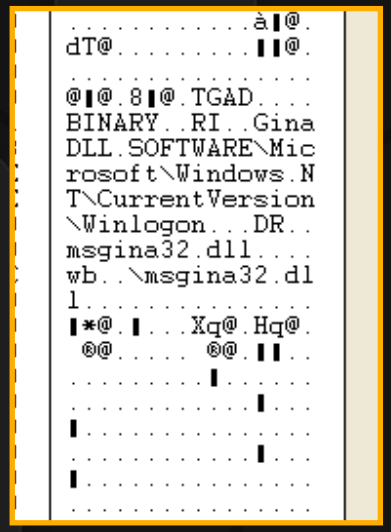
Dalle funzioni richiamate all'interno della libreria possiamo ipotizzare che il malware, altera i file, gestisce i processi e prende informazioni riguardanti Windows (credenziali).

ADVAPI32.DLL
Libreria che contiene le funzioni per interagire con i registri e i servizi del sistema operativo Microsoft.

In questo caso, le funzioni della libreria hanno il compito di modificare i registri, creando o aprendo una nuova chiave e modificandone il valore.

CONCLUSIONI

Possiamo ipotizzare che il malware interagisca coi registri per poter, magari, ottenere una persistenza oppure modificare dei parametri per effettuare delle operazioni malevole che andrà a sfruttare.
Inoltre sappiamo che interagisce con altri processi tramite le varie funzioni importate dalla libreria KERNEL32.



Possiamo vedere che all'interno della sezione .data del tool CFF, il malware interagisce con **Winlogon** e che la DLL è **msgina32.dll**.

Questo ci fa ipotizzare che il malware vada a modificare i criteri di autenticazione ed accesso che sono previsti per l'interazione con l'utente. Ma non possiamo dedurre se questo sia per appropriarsi delle credenziali dell'utente o per escluderlo al di fuori.

GIORNO 2

1) Lo scopo della funzione chiamata alla locazione di memoria 00401021.

```
.text:00401021                call    ds:RegCreateKeyExA
```

La funzione **RegCreateKeyExA** ha il compito di creare una specifica chiave di registro, se già esistente, la funzione la apre.

2) Come vengono passati i parametri alla funzione alla locazione 00401021.

```
.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
```

I parametri vengono passati tramite istruzione **push** (memorizza le variabili nel segmento di memoria dello stack).

3) Che oggetto rappresenta il parametro alla locazione 00401017.

```
db 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon',0
```

Il parametro rappresenta un processo del sistema Microsoft per la gestione dei logon (**Winlogon**).
In questo caso, il malware, lo modifica per poter recuperare informazioni sull'utente.

4) Il significato delle istruzioni comprese tra gli indirizzi 00401027 e 00401029.

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

L'istruzione test effettua un **AND** tra eax ed eax, quindi imposta la **zero flag** a 1 se eax è uguale a 0.

5) Con riferimento all'ultimo quesito, tradurre il codice Assembly nel corrispondente costrutto C.

```
if (eax == 0) {

   // codice in caso di zero

} else {

   // codice in caso di non zero

}
```

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

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

GinaDLL è il valore del parametro ValueName.

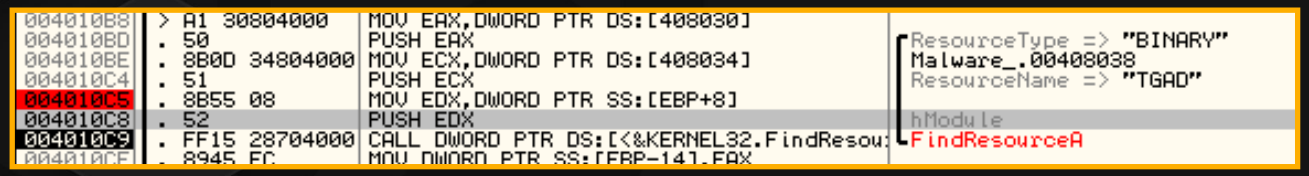
7) Nel complesso delle due funzionalità appena viste, spiegate quale funzionalità sta implementando il Malware in questa sezione.

Prendendo in considerazione le due funzionalità e ipotizzando che il salto venga effettivamente effettuato, si prende in considerazione la chiave di registro che viene creata o aperta e successivamente viene modificata col valore **GinaDLL**.

GIORNO 3

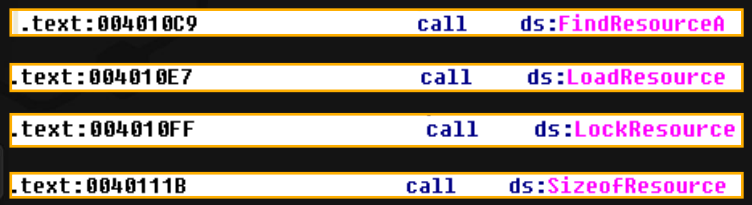
Riprendete l'analisi del codice, analizzando le routine tra le locazioni di memoria 00401080 e 00401128.

1) Qual è il valore del parametro «ResourceName» passato alla funzione FindResourceA);



Attraverso OLLYDB possiamo andare a ricercare il valore di ResourceName, il quale è TGAD.

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?



Questo susseguirsi di chiamate di funzioni, fanno intendere che queste API's permettono di localizzare all'interno della sezione ' *risorse* ' il malware da estrarre e successivamente da caricare in memoria per l'esecuzione o da salvare sul disco per l'esecuzione futura.
Attraverso queste caratteristiche possiamo intendere che il malware è un **DROPPER**.

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

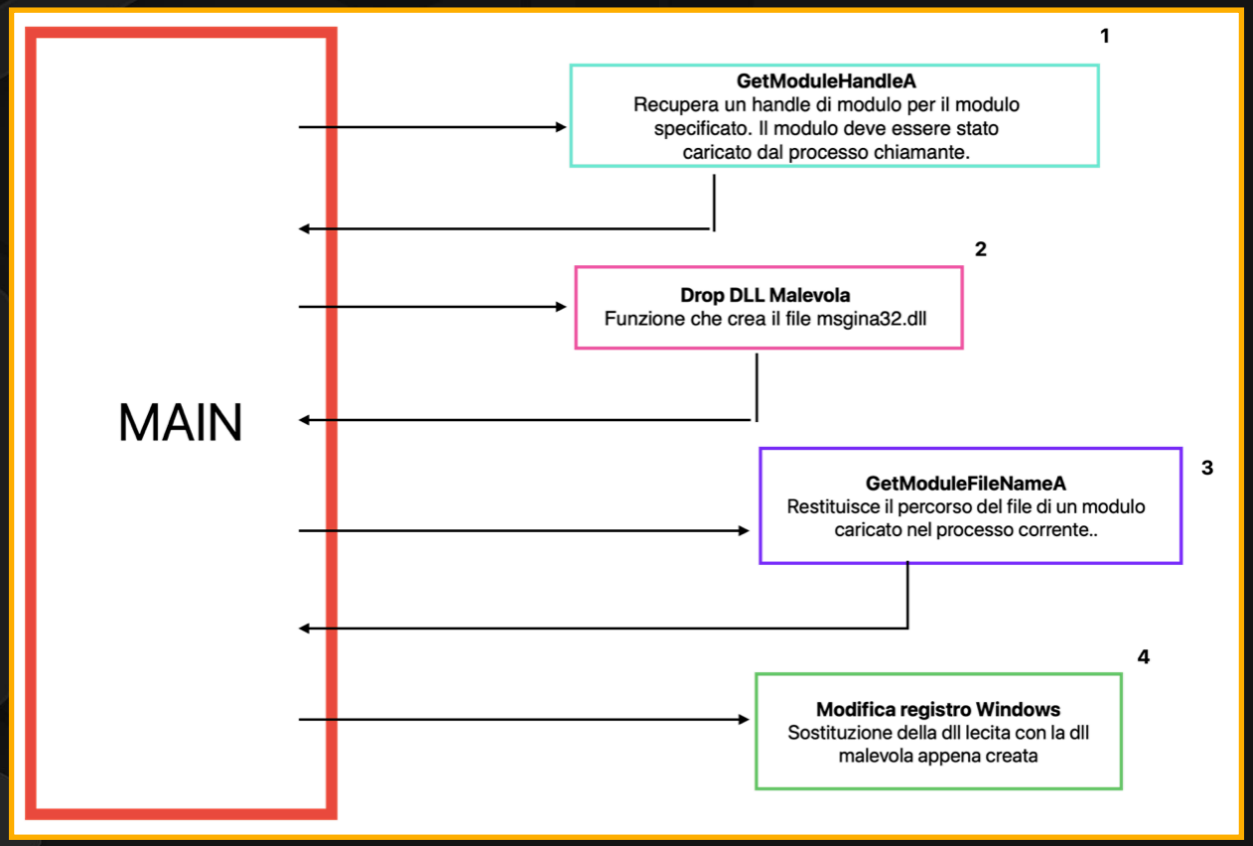
Attraverso l'utilizzo del tool **CFFExplorer** possiamo andare a ricercare le funzionalità, per riuscire a capire in modo approssimativo il compito del Malware.

4) In caso di risposta affermativa, elencare le evidenze a supporto.

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

In questo caso le evidenze che riusciamo a riscontrare, sono le diverse funzioni che vengono implementate all'interno della libreria KERNEL32.

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.



1. La funzione Main richiama la funzione **GetModuleHandleA** passando attraverso push il parametro lpModuleName, che restituisce hModule.

```
push    0                       ; lpModuleName
call    ds:GetModuleHandleA
```

2. In questo caso abbiamo rinominato la funzione chiamata dal Main con il nome **dropDLLMalevola** (funzione che crea il file msgina32.dll)
Il Main passa il parametro hModule e successivamente la funzione restituisce lpFileName, che corrisponde al path completo del file appena creato.

```
push    eax             ; hModule
call    dropDLLMalevola
```

3. Il Main passa 3 parametri (come vediamo in figura) alla funzione **GetModuleFileName**, che restituisce nSize (lunghezza del path).

```
push    10Eh            ; nSize
lea     ecx, [ebp+Data]
push    ecx             ; lpFilename
push    0               ; hModule
call    ds:GetModuleFileNameA
```

4. Anche in questo caso abbiamo rinominato la funzione, per renderla più comprensibile (**modificaRegistroWindows**).
Il Main pusha i due parametri alla funzione, che si occuperà di modificare il registro inserendo la path della dll appena creata come valore della chiave.

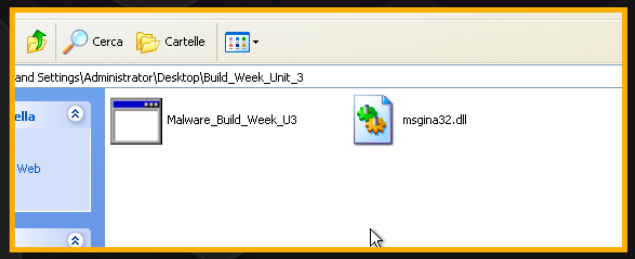
```
push    104h            ; cbData
lea     eax, [ebp+Data]
push    eax             ; lpData
call    modificaRegistoWindows
```

GIORNO 4

Preparazione dell'ambiente di lavoro ed i tool per l'esecuzione del Malware.

---

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

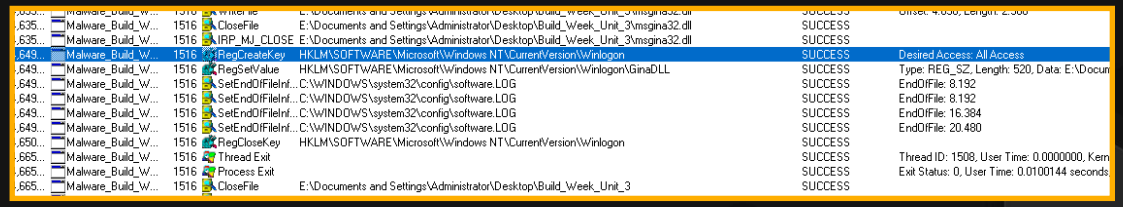


Come possiamo vedere, all'interno della stessa cartella del Malware, viene creato un file dll, il quale è stato preventivamente notato nella giornata precedente, riguardante la funzione dropDLLMalevola.

Analizzate ora i risultati di **Process Monitor**.
Filtrate includendo solamente l'attività sul registro di Windows.

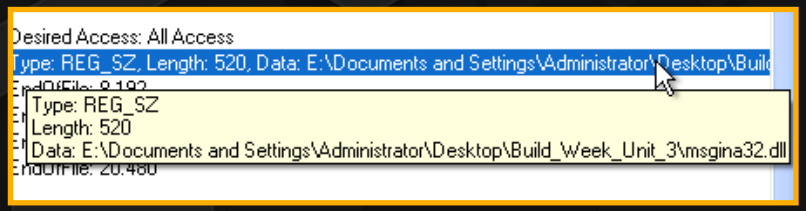
---

2) Quale chiave di registro viene creata?



Viene creata la **chiave di registro di Winlogon**.

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

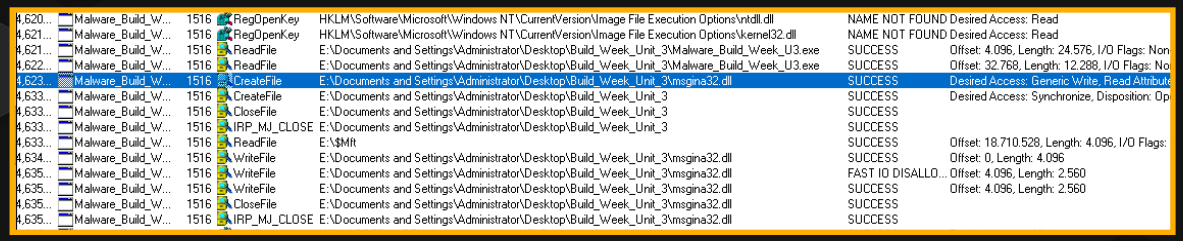


Come abbiamo già visto viene inserito come valore, la **path** completa della dll malevola.

Passate ora alla visualizzazione dell'attività sul **file system**.

---

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

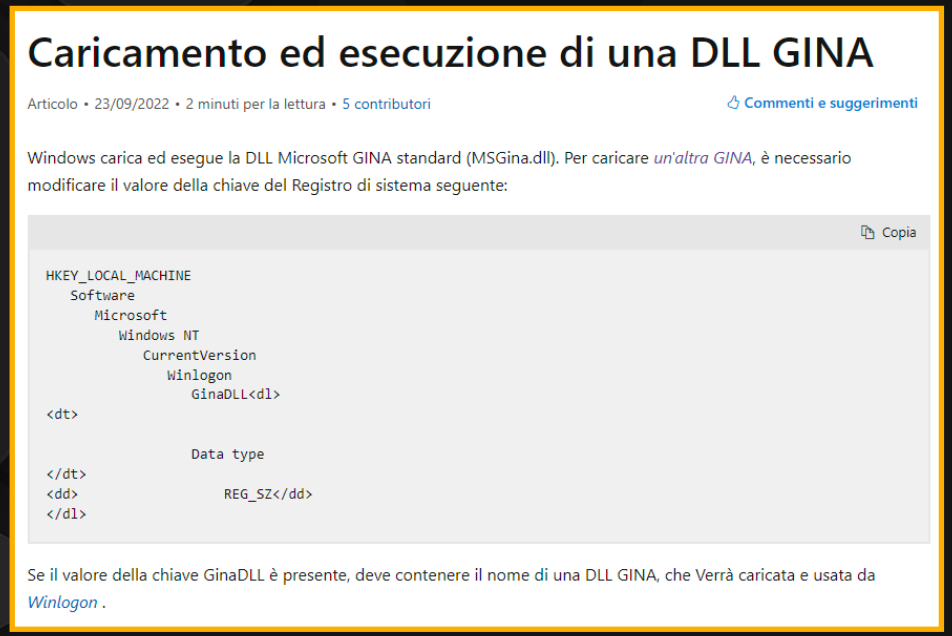


4,620...	Malware_Build_W...	1516	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ntdll.dll	NAME NOT FOUND	Desired Access: Read
4,621...	Malware_Build_W...	1516	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\kernel32.dll	NAME NOT FOUND	Desired Access: Read
4,621...	Malware_Build_W...	1516	ReadFile	E:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	Offset: 4.096, Length: 24.576, I/O Flags: Non
4,622...	Malware_Build_W...	1516	ReadFile	E:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\Malware_Build_Week_U3.exe	SUCCESS	Offset: 32.768, Length: 12.288, I/O Flags: No
4,623...	Malware_Build_W...	1516	CreateFile	E:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS	Desired Access: Generic Write, Read Attribute
4,633...	Malware_Build_W...	1516	CreateFile	E:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS	Desired Access: Synchronize, Disposition: Op
4,633...	Malware_Build_W...	1516	CloseFile	E:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS	
4,633...	Malware_Build_W...	1516	IRP_MJ_CLOSE	E:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3	SUCCESS	
4,633...	Malware_Build_W...	1516	ReadFile	E:\$Mft	SUCCESS	Offset: 18.710.528, Length: 4.096, I/O Flags:
4,634...	Malware_Build_W...	1516	WriteFile	E:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS	Offset: 0, Length: 4.096
4,635...	Malware_Build_W...	1516	WriteFile	E:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	FAST IO DISALLO...	Offset: 4.096, Length: 2.560
4,635...	Malware_Build_W...	1516	WriteFile	E:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS	Offset: 4.096, Length: 2.560
4,635...	Malware_Build_W...	1516	CloseFile	E:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS	
4,635...	Malware_Build_W...	1516	IRP_MJ_CLOSE	E:\Documents and Settings\Administrator\Desktop\Build_Week_Unit_3\msgina32.dll	SUCCESS	

La chiamata di sistema utilizzata per la modifica del contenuto è **CreateFile**.

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

Come abbiamo visto fino ad ora, il Malware crea una dll, chiamata **msgina32.dll**, all'interno della sua stessa cartella.
Successivamente va a **modificare il registro Windows** che ha il compito della gestione dell'identificazione grafica e l'autenticazione degli utenti, facendo in modo che venga utilizzata la dll appena creata invece della preesistente.



Caricamento ed esecuzione di una DLL GINA

Articolo • 23/09/2022 • 2 minuti per la lettura • 5 contributori

Commenti e suggerimenti

Windows carica ed esegue la DLL Microsoft GINA standard (MSGina.dll). Per caricare *un'altra GINA*, è necessario modificare il valore della chiave del Registro di sistema seguente:

Copia

```
HKEY_LOCAL_MACHINE
   Software
      Microsoft
         Windows NT
            CurrentVersion
               Winlogon
                  GinaDLL<dl>
<dt>

                  Data type
</dt>
<dd>                  REG_SZ</dd>
</dl>
```

Se il valore della chiave GinaDLL è presente, deve contenere il nome di una DLL GINA, che Verrà caricata e usata da *Winlogon* .

Da una piccola **ricerca online** siamo riusciti a risalire ad una documentazione che ci permette di comprendere in modo migliore il caricamento della dll GINA.

Questo dando come valore alla chiave di registro **Winlogon** la path della dll malevola.
Possiamo dedurre che il Malware posso andare a compromettere la confidenzialità o integrità delle credenziali di accesso dell'utente.

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?

Se il file viene sostituito con una dll creata espressamente dall'attaccante, è possibile intercettare le credenziali inserite dall'utente della macchina.
Il Malwere va a minacciare la confidenzialità e integrità delle credenziali valide per l'accesso.

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.

Per comprendere meglio il suo funzionamento, siamo andati ad analizzare la dll creata, dallo stesso Malware.

```
; int __stdcall WlxLoggedOutSAS(PVOID pWlxContext,DWORD dwSasType,PLUID pAuthenticationId,PSID pLogonSid,PDWORD pdwOpti
                public WlxLoggedOutSAS
WlxLoggedOutSAS proc near
```

Questa funzione viene lanciata al **logout** ed al suo interno troviamo diversi richiami interessanti che possiamo analizzare.

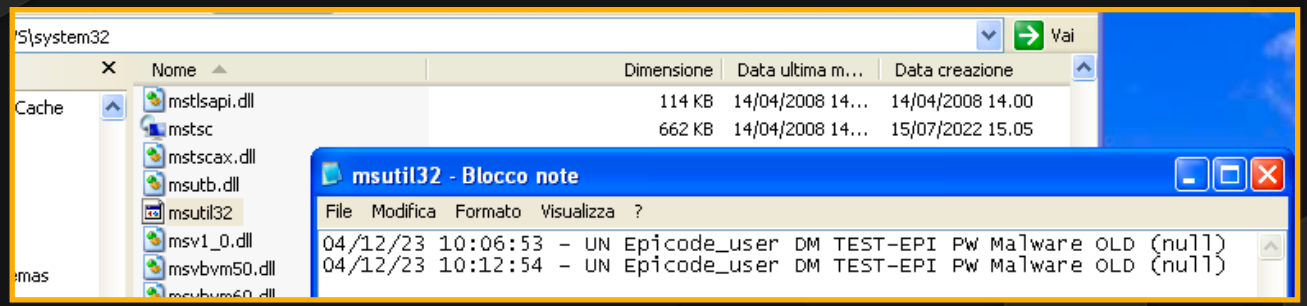
```
push    eax                     ; char
push    offset aUnSDmSPwSOldS ; "UN %s DM %s PW %s OLD %s"
push    0                       ; dwMessageId
call    SalvaCred
```

Alla funzione (che abbiamo rinominato con SalvaCred) viene passata una **stringa sospetta**.

```
.text:10001570 ; int __cdecl SalvaCred(DWORD dwMessageId,wchar_t *,char)
.text:10001570 SalvaCred       proc near               ; CODE XREF: WlxLoggedOutSAS+63↑p
.text:10001570
.text:10001570 hMem            = dword ptr -854h
.text:10001570 var_850         = word ptr -850h
.text:10001570 var_828         = word ptr -828h
.text:10001570 var_800         = word ptr -800h
.text:10001570 dwMessageId     = dword ptr  4
.text:10001570 arg_4           = dword ptr  8
.text:10001570 arg_8           = byte ptr  0Ch
.text:10001570
.text:10001570                 mov     ecx, [esp+arg_4]
.text:10001574                 sub     esp, 854h
.text:1000157A                 lea     eax, [esp+854h+arg_8]
.text:10001581                 lea     edx, [esp+854h+var_800]
.text:10001585                 push    esi
.text:10001586                 push    eax             ; va_list
.text:10001587                 push    ecx             ; wchar_t *
.text:10001588                 push    800h            ; size_t
.text:1000158D                 push    edx             ; wchar_t *
.text:1000158E                 call    _vsnwprintf
.text:10001593                 push    offset word_10003320 ; wchar_t *
.text:10001598                 push    offset aMsutil32_sys ; "msutil32.sys"
.text:1000159D                 call    _wfopen
```

Una volta dentro la funzione **SalvaCred**, possiamo andare a notare che viene creato un file sospetto chiamato **msutil32.sys**, che presumibilmente contiene, al suo interno, la stringa sospetta.

Una volta che andiamo a ricercare il file all'interno di **system32**, abbiamo un riscontro positivo delle nostre ipotesi.
Il file contiene le **informazioni di login** dell'utente e un timestamp di quando esse sono state catturate.



CONCLUSIONI

In sintesi il funzionamento del malware, consiste nella creazione di una dll malevola che va a sostituirsi alla lecita tramite la modifica del registro.
Successivamente al logout dell'utente, la dll va a salvare le credenziali dell'utente e le inserisce in un file appositamente creato ed occultato.

FINE

MEMBRI:

MARIANO HANGANU
ALESSIO BATTAGLIA
GIOVANNI ETERNATO
SAMUELE MASSACESI
RICCARDO GRECO
GABRIELE DI SALVO
ALESSIO KENYON
VINCENZO PIO RUSCILLO