

Build Week III

Team Amedeo

Kame House Analyst

Amedeo Natalizi

Dario Santigliano

Federico Bertini

Rosario Zappalà

Guglielmo Carratello

Jun Lu

Iacopo Bombieri

Indice

Introduzione	pag 1
Concetti Fondamentali	pag 2
Tools	pag 4
Task 1	pag 7
Contenuto del capitolo in breve	pag 7
Funzione main().....	pag 8
Sezioni di cui è composto il malware	pag 9
Librerie importate dal malware	pag 9
Conclusioni	pag 12
Task 2	pag 13
Contenuto del capitolo in breve	pag 13
Spiegazione del codice	pag 14
Conclusioni	pag 16
Task 3	pag 17
Contenuto del capitolo in breve	pag 17
Analisi del codice	pag 18
Nome parametro della funzione Resource Name	pag 19
Funzionalità del malware nella sezione in esame	pag 20
Analisi statica del malware e chiarimenti.....	pag 21
Conclusioni	pag 23
Task 4	pag 24
Contenuto del capitolo in breve	pag 25
Contenuto della cartella del malware	pag 26
Analisi dei risultati di Process Monitor	pag 27
Studio del funzionamento del codice	pag 28
Test sul sistema	pag 31
Conclusioni	pag 32
Task 5	pag 33
Contenuto del capitolo in breve	pag 33
Conseguenze sostituzione file .dll	pag 34
Remediation actions	pag 35
Grafico ad alto livello	pag 36
Conclusioni	pag 36

Introduzione

Durante la lettura di questo report, sarà condotta un'analisi su un malware al fine di identificare i comportamenti assunti da quest'ultimo. A tale scopo, verranno impiegati strumenti noti nel campo del Malware Analysis come:

- IDA Pro
- CFF Explorer VIII
- Process Monitor

L'utilizzo di questi software verrà brevemente anticipata in "Sezione 1" prima di iniziare le nostre analisi.

Le analisi si concentreranno sull'identificazione e sull'interpretazione dei vari componenti del malware, inclusi segmenti di codice, funzioni crittografiche e comandi di controllo, al fine di identificare i comportamenti assunti e dedurre di che tipo di Malware stiamo parlando.

Anche per quanto riguarda i Malware andremo a dare una breve spiegazione in Sezione 1.

TEAM AMED KAME AH!



Sezione 1 - Concetti Fondamentali:

-Malware

Un malware è un software che esegue codice malevolo su una macchina vittima.

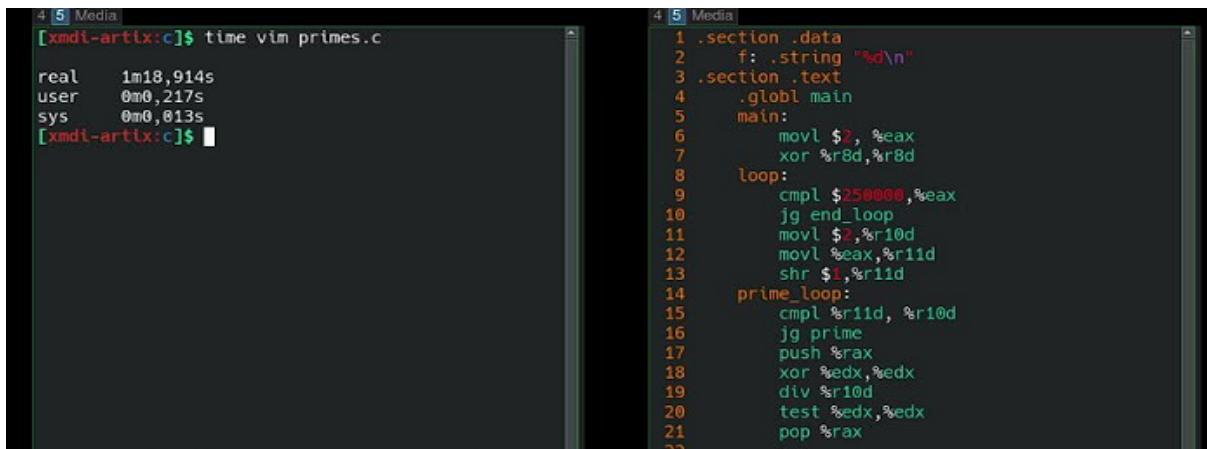
Esistono diversi tipi di malware con nomi diversi, associati al loro comportamento.

Facciamo degli esempi per rendere meglio l'idea:

- **Backdoor(Porta sul retro):** Sono dei software malevoli che creano una porta di accesso sul pc della vittima.Questo tipo di attacco dà modo al Black Hat Hacker di accedere ad ogni informazione presente sulla macchina in uso dalla vittima. Inoltre il criminale informatico potrebbe eseguire codice malevolo sulla macchina a suo piacimento, ed avrebbe il modo di installare ulteriori Malware.
- **Trojan Horse (Cavallo di Troia):** Prende il suo nome dal noto episodio dell'utilizzo da parte dei Greci del "Cavallo di troia" per assediare Troia. Allo stesso modo il malware si rende legittimo agli occhi dell'utente che lo installa, dando l'apparenza di essere innocuo. Il Software potrebbe funzionare normalmente (ad esempio un gioco) ma durante l'installazione (in questo caso del gioco) è stato installato anche il Trojan Horse, che ora darà modo al criminale informatico di recuperare ogni tipo di dato all'interno della macchina vittima, (anche i dati bancari) inoltre avrà la possibilità di poter eseguire ogni tipo di operazione a suo piacimento, compreso il controllo a distanza del dispositivo vittima.
- **Keylogger:** un Keylogger è un malware che rileva gli input forniti dall'utente (solitamente i dati recuperati dall'input della tastiera sono più importanti di quelli recuperati dall'input del mouse). Una volta installato questo malware sul pc della vittima il criminale informatico potrà accedere alla lista di ogni carattere che è stato digitato sulla macchina della vittima. In questo modo è possibile rubare le credenziali di accesso ad un portale bancario, o altre informazioni sensibili del genere.
- **Dropper:** Un Dropper è un Malware che una volta installato (in maniera ingannevole, nascosto all'interno di un'altro software) andrà ad assicurarsi della connessione ad internet per poi scaricare ulteriori malware. Questo malware ha la caratteristica di riuscire a mischiarsi in mezzo al codice di altri software, questo rende difficile la sua scomparsa effettiva all'interno della macchina vittima.

Assembly

L'assembly è considerato un linguaggio di basso livello, collocandosi tra il linguaggio macchina e quello umano. A differenza di linguaggi più vicini alla comprensione umana, come il C++, l'assembly richiede una comprensione più dettagliata delle istruzioni e dei comandi eseguiti dai programmi.



The screenshot shows two terminal windows side-by-side. The left window displays the command `time vim primes.c` and its execution time: real 1m18,914s, user 0m0,217s, sys 0m0,013s. The right window shows the assembly code for the `primes.c` program, starting with the `.section .data` and `.section .text` directives, followed by the `main:` section and a `loop:` section containing various assembly instructions.

```

4 [5] Media
[xmddi-artlx:c]$ time vim primes.c
real    1m18,914s
user    0m0,217s
sys     0m0,013s
[xmddi-artlx:c]$ 

4 [5] Media
1 .section .data
2   f: .string "%d\n"
3 .section .text
4   .globl main
5   main:
6       movl $2, %eax
7       xor %r8d,%r8d
8   loop:
9       cmpl $250000,%eax
10      jg end_loop
11      movl $2,%r10d
12      movl %eax,%r11d
13      shr $1,%r11d
14 prime_loop:
15      cmpl %r11d, %r10d
16      jg prime
17      push %rax
18      xor %edx,%edx
19      div %r10d
20      test %edx,%edx
21      pop %rax
22

```

Sezione 1 Figura 1

Questo tipo di linguaggio consente di interpretare le istruzioni eseguite dal malware appena è stato scomposto (decompilato).

Tuttavia, non è possibile risalire direttamente al codice sorgente originale a causa della trasformazione del codice sorgente in codice assembly prima della compilazione in un file eseguibile.

Nonostante questa limitazione, è comunque possibile comprendere le azioni che un file eseguibile cerca di compiere attraverso l'uso di strumenti specializzati per l'analisi dei malware. Questi strumenti consentono di esaminare il comportamento e le funzionalità del malware, anche se non è possibile accedere direttamente al codice sorgente originale.

Tools

- **IDA PRO**

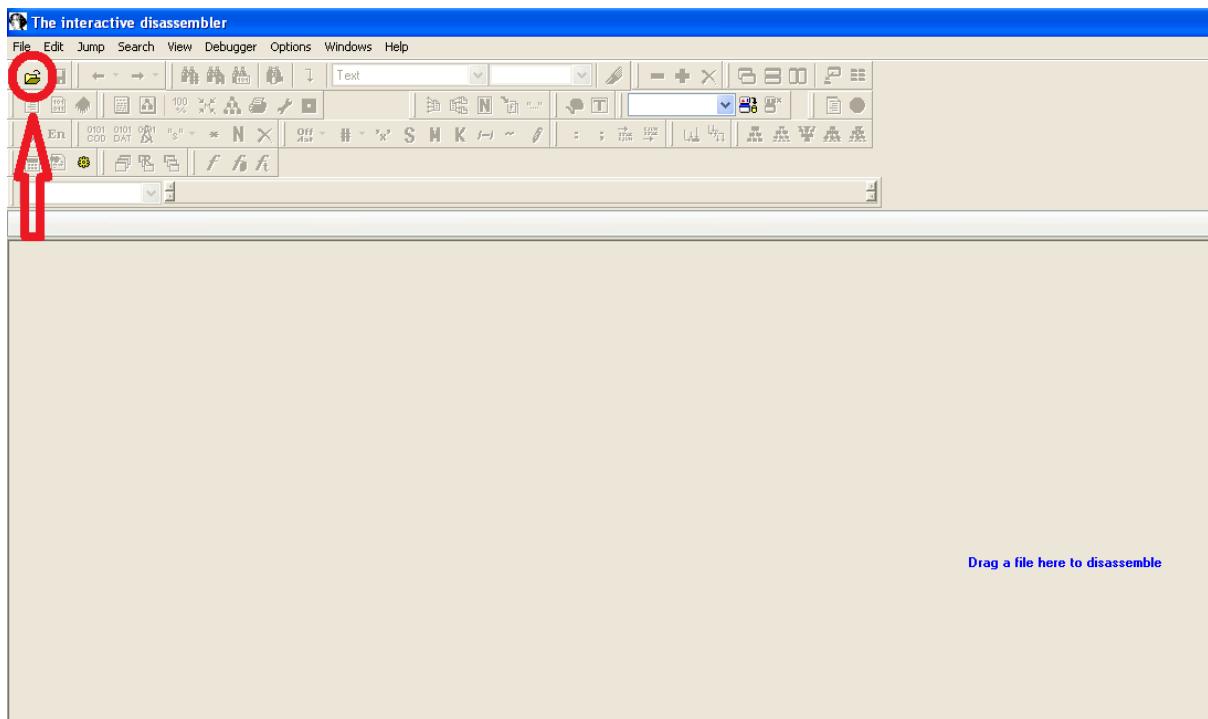
IDA PRO è un disassembler, cioè un software utilizzato per analizzare il codice di un programma eseguibile traducendolo in linguaggio assembly.

L'icona del software è presente in Sezione 1 Figura 1.



Sezione 1 Figura 2

Una volta avviato il software sarà visibile una schermata vuota come in Sezione 1 Figura 2, ciò che bisogna fare è: andare in alto, a sinistra, col puntatore del nostro mouse per selezionare il File eseguibile oggetto dell'analisi.



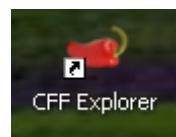
Sezione 1 Figura 3

//Nota: Selezionando l'icona, presente in Sezione 1 Figura 2, il software chiederà di inserire il percorso dell'eseguibile oggetto delle analisi, per rilevare il malware ed il suo funzionamento, che andremo ad effettuare.

Una volta caricato il file sarà possibile leggere il codice in assembly dell' eseguibile e visualizzare le sezioni di cui è composto.

CFF Explorer

CFF Explorer è un software utilizzato dagli analisti di sicurezza informatica per esaminare, analizzare e comprendere la struttura interna dei file eseguibili. Utilizzando il programma CFF Explorer si riesce ad ottenere una visione più ampia delle attività del malware.



Sezione 1 Figura 4



Sezione 1 Figura 5

In Sezione 1 Figura 4 è possibile osservare la schermata iniziale del Software CFF Explorer. Per caricare un file eseguibile da analizzare bisogna utilizzare il tasto in alto a sinistra, come indicato in Figura.

Il Software dividerà l'eseguibile in sezioni che contengono informazioni comprensibili ad un operatore nel campo del Malware Analysis.

Process Monitor



Sezione 1 Figura 6

Process Monitor è un software di monitoraggio del sistema per Windows sviluppata da Microsoft. Fornisce una dettagliata visualizzazione in tempo reale delle attività del sistema, inclusi processi in esecuzione, registri di sistema, attività di rete, operazioni del file system ed altro ancora. Ognuno di questi filtri si può applicare tramite la barra in alto, un esempio è mostrato in Sezione Sezione 1 Figura x

Time of Day	Process Name	PID	Operation	Path
12:38:05,3614...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3616...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3617...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3619...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3620...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3622...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3623...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3624...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3625...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3626...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3628...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3629...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3630...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3632...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3633...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3634...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3636...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3638...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3639...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3640...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3642...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3644...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3645...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3649...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system
12:38:05,3649...	Procmon.exe	1232	ReadFile	C:\Windows\system32\config\system

Sezione 1 Figura 7
Sulla destra possiamo trovare:

Result	Detail
SUCCESS	Offset: 184,320, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 192,512, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 299,008, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 368,640, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 4,501,504, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,245,184, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,052,672, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,687,552, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,683,456, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,699,840, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,024,000, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,028,096, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,064,960, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,036,288, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,003,520, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 4,476,928, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 3,792,896, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,060,864, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 933,888, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 1,069,056, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 3,534,848, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O
SUCCESS	Offset: 4,329,472, Length: 4,096, I/O Flags: Non-cached, Paging I/O, Synchronous Paging I/O

Sezione 1 Figura 8

Task 1

- 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.

Contenuto del capitolo in breve:

In accordo con le specifiche della task, il malware è stato sottoposto inizialmente a un'analisi statica utilizzando lo strumento IDA. Durante questa fase, si è focalizzata l'attenzione sulla funzione main per individuare variabili e parametri rilevanti nel codice assembly.

Successivamente, si è proceduto con l'utilizzo dello strumento CFF Explorer per esaminare le sezioni e le librerie importate dal malware. Tra le librerie importate, la presenza di kernel32.dll ha suggerito la possibilità di una funzionalità di dropper, mentre l'utilizzo di advapi32.dll ha fatto ipotizzare la capacità del malware di effettuare modifiche al registro per ottenere persistenza nel sistema.

Inoltre, durante l'analisi del codice ASCII delle sezioni, è stata individuata una corrispondenza con GinaDLL, un componente di Windows che gestisce l'interfaccia di login. Questa scoperta suggerisce che il malware potrebbe interagire con tale componente per il raggiungimento dei suoi obiettivi.

Funzione Main():

 NUL

```
; 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
```

Sezione 2 Figura 1

I parametri identificati sono 3:

argc = dword ptr 8 => rappresenta il numero di argomenti passati al programma

argv = dword ptr 0Ch => è un puntatore all'array di stringhe che sono passate al programma

envp = dword ptr 10h => abbreviazione di "environment pointer" (puntatore all'ambiente), è un parametro passato in assembly durante l'avvio. Contiene un puntatore a un array di stringhe che rappresentano le variabili di ambiente disponibili al programma.

Le variabili di ambiente sono variabili di configurazione del sistema che possono essere utilizzate dai programmi per accedere a informazioni specifiche o per modificare il comportamento del programma stesso.

Le variabili sono 4:

hModule = dword ptr -11Ch => Questa variabile potrebbe essere utilizzata per archiviare un handle o un puntatore a un modulo caricato. Il valore decimale è 12.

Data = byte ptr -118h => Questa variabile è probabilmente utilizzata per archiviare dati temporanei o buffer di dati. Il valore decimale è 280.

var_8 = dword ptr -8 => Questa variabile è di tipo dword (32-bit) e potrebbe essere utilizzata per archiviare dati temporanei o variabili locali.

var_4 = dword ptr -4 => Questa variabile è di tipo dword (32-bit) e potrebbe essere utilizzata per archiviare dati temporanei o variabili locali.

Le sezioni (section headers) di cui è composto il malware sono:

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Word	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

Sezione 2 Figura 2

- **.text:** contiene il codice che viene eseguito dal malware, è la sezione dove sono memorizzate le istruzioni macchina che vengono interpretate ed eseguite dal processore (CPU) durante l'esecuzione.
- **.data:** contiene variabili globali e statiche che vengono utilizzate dal malware durante l'esecuzione. Fornisce un luogo per memorizzare dati persistenti o temporanei necessari per il funzionamento del malware
- **.rdata:** contiene dati inizializzati staticamente destinati ad essere accessibili in sola lettura, contribuendo così alla stabilità del malware.
- **.rsrc:** contiene una vasta gamma di risorse che vengono utilizzate dal malware durante l'esecuzione come immagine ed icone, stringhe di testo e menu.

Andando nella sezione Import Directory è possibile andare ad analizzare le librerie importate dal malware:

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.dll	51	00007534	00000000	00000000	0000769E	0000700C
ADVAPI32.dll	2	00007528	00000000	00000000	00007600	00007000

Sezione 2 Figura 3

- **Kernel32.dll:** contiene una gamma di funzioni che sono essenziali per il funzionamento del sistema operativo come ad esempio: gestione di processi e thread, gestione della memoria, gestione dispositivi I/O, gestione delle stringhe e gestione del tempo e delle date.
- **Advapi32.dll:** fornisce funzionalità di basso livello relative alla sicurezza, gestione dei servizi e accesso ai registri. In particolare include funzioni per la lettura e scrittura dei dati nei registri di sistema inclusi registri utente e di sistema.

Libreria **Kernel32.dll:**

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

Sezione 2 Figura 4

Potrebbe trattarsi di un dropper per via delle funzioni contenute nella libreria kernel32.dll: FindResourceA, LoadResource, LockResource, SizeOfResource.

FindResourceA: Questa funzione viene utilizzata per individuare una risorsa specifica all'interno di un file eseguibile o di una libreria. Il malware potrebbe utilizzare questa funzione per cercare risorse dannose all'interno di file eseguibili o librerie presenti nel sistema.

LoadResource: Dopo aver individuato una risorsa con FindResourceA, il malware potrebbe caricarla in memoria utilizzando LoadResource. Questo è utile se il malware contiene risorse aggiuntive (ad esempio, payload dannosi) all'interno del suo eseguibile o all'interno di altre librerie.

LockResource: Dopo aver caricato la risorsa in memoria, il malware potrebbe utilizzare LockResource per ottenere un puntatore al contenuto della risorsa stessa. Questo è importante se il malware ha bisogno di accedere direttamente ai dati della risorsa.

SizeOfResource: Questa funzione restituisce la dimensione, in byte, di una risorsa specifica. Il malware potrebbe utilizzare questa informazione per allocare la quantità corretta di memoria per caricare la risorsa utilizzando LoadResource.

Insieme, queste funzioni potrebbero essere utilizzate da un dropper per caricare risorse dannose (come payload, file di configurazione o altri dati dannosi) in memoria e quindi eseguirle per compromettere ulteriormente il sistema bersaglio. Ad esempio, il dropper potrebbe utilizzare queste funzioni per caricare un payload dannoso in memoria e quindi eseguirlo per ottenere accesso non autorizzato al sistema o per svolgere altre attività dannose.

Nella libreria kernel.dll non è stata trovata nessuna funzione di CreateProcess(), tuttavia sono presenti CreateFilea() e WriteFile(), si può quindi ipotizzare che il malware viene salvato sul disco per utilizzi futuri.

Libreria **Advapi32.dll**:

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

Sezione 2 Figura 5

Le funzioni RegSetValueExA e RegCreateKeyExA presenti in Advapi32.dll possono essere utilizzate per ottenere la persistenza del malware. Ecco come potrebbero essere utilizzate:

RegSetValueExA: Questa funzione viene utilizzata per scrivere dati in un valore specifico all'interno del registro di sistema. Il malware potrebbe utilizzare questa funzione per scrivere informazioni relative alla sua esecuzione nel registro di avvio automatico o in altre chiavi di registro rilevanti per garantire che venga eseguito automaticamente all'avvio del sistema.

RegCreateKeyExA: Questa funzione viene utilizzata per creare una nuova chiave nel registro di sistema. Il malware potrebbe utilizzare questa funzione per creare una nuova chiave nel registro di avvio automatico o in altre posizioni di avvio del sistema, dove può successivamente scrivere le informazioni necessarie per garantire la persistenza.

Insieme, queste due funzioni possono essere utilizzate per modificare il registro di sistema in modo che il malware venga eseguito automaticamente all'avvio del sistema, garantendo così la persistenza nel sistema bersaglio.

Andando ad analizzare le varie sezioni, nel codice Ascii di .data c'è una corrispondenza con **GinaDLL**. Si tratta di un componente del sistema operativo Windows che gestisce l'interfaccia di login grafica e le operazioni di autenticazione.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
00000000	00	00	00	00	00	00	00	00	00	00	E0	1E	40	00		à @.	
00000010	64	54	40	00	00	00	00	00	00	00	85	1F	40	00		dT@ @.	
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000030	40	80	40	00	38	80	40	00	54	47	41	44	00	00	00	@ @.8 @.TGAD ..	
00000040	42	49	4E	41	52	59	00	00	52	49	0A	00	47	69	6E	61	
00000050	44	4C	4C	00	53	4F	46	54	57	41	52	45	5C	4D	69	63	
00000060	72	6F	73	6F	66	74	5C	57	69	6E	64	6F	77	73	20	4E	
00000070	54	5C	43	75	72	72	65	6E	74	56	65	72	73	69	6F	6E	
00000080	5C	57	69	6E	6C	6F	67	6F	6E	00	00	00	44	52	0A	00	
00000090	6D	73	67	69	6E	61	33	32	2E	64	6C	6C	00	00	00	00	
000000A0	77	62	00	00	5C	6D	73	67	69	6E	61	33	32	2E	64	6C	
000000B0	6C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000C0	0B	2A	40	00	01	00	00	00	58	71	40	00	48	71	40	00	
000000D0	A0	AE	40	00	00	00	00	00	A0	AE	40	00	01	01	00	00	
000000E0	00	00	00	00	00	00	00	00	10	00	00	00	00	00	00	00	
000000F0	00	00	00	00	00	00	00	00	00	00	00	02	00	00	00	00	
00000100	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000110	00	00	00	00	00	00	00	00	00	00	00	02	00	00	00	00	
00000120	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Sezione 2 Figura 6

Conclusioni:

L'analisi condotta ha portato a importanti conclusioni riguardo all'effetto e al comportamento del malware sul sistema operativo. La scoperta della creazione di una nuova chiave di registro e dell'associazione di un valore ad essa indica chiaramente l'intenzione del malware di influenzare il processo di accesso di Windows. Allo stesso modo, l'osservazione della modifica del file system evidenzia la capacità del malware di manipolare i file di sistema. Tuttavia, ulteriori indagini sono necessarie per comprendere appieno la natura e le potenziali minacce rappresentate da questo malware e per sviluppare strategie efficaci per la sua rimozione e la protezione del sistema.

Task 2

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.

Contenuto del capitolo in breve:

In questa seconda giornata, per analizzare le specifiche locazioni di memoria e comprendere il comportamento del malware, è stato utilizzato il software d'analisi **IDA Pro**. Attraverso questo strumento è stato possibile esaminare il codice assembly nelle locazioni di memoria specificate al fine di comprendere la logica del programma.

Inizialmente, si è individuata la locazione di memoria specifica menzionata, come ad esempio 00401021. Utilizzando **IDA Pro**, si è acceduto direttamente a questa locazione di memoria e si è proceduto ad analizzare il codice assembly circostante per comprendere lo scopo della funzione presente in quella posizione. Successivamente, è stato studiato il modo in cui i parametri vengono passati alla funzione in questione. Ciò è stato fatto esaminando il codice assembly e identificando quali registri vengono utilizzati per trasferire i parametri e come vengono preparati prima della chiamata della funzione.

In alcuni casi è stato necessario tradurre il codice assembly in linguaggio C, per avere una migliore comprensione della logica del programma e delle operazioni eseguite.

In sintesi, per affrontare queste task è stato essenziale utilizzare strumenti di analisi come **IDA Pro** per esaminare il codice assembly del malware, analizzare come vengono passati i parametri alle funzioni, interpretare le istruzioni e, se necessario, tradurre il codice assembly per una migliore comprensione della logica del programma.

Spiegazione del codice:

1.

Da come si può osservare dalla figura, lo scopo della chiamata alla locazione di memoria 00401021 è **RegCreateKeyExA**. Questa funzione serve per creare una nuova chiave di registro.

```
.text:00401017 push    v , neset vcu
      push    offset SubKey ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"...
      push    80000002h ; hKey
      call    ds:RegCreateKeyExA
      test   eax, eax
      jz     short loc_401032
      mov    eax, 1
      jmp    short loc_40107B
```

Sezione 2 Figura 7

2.

```
.text:00401000 ; int __cdecl sub_401000(BYTE *lpData,DWORD cbData)
.text:00401000 sub_401000 proc near ; CODE XREF: _main+B84p
.text:00401000
.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
.text:00401027     test   eax, eax
.text:00401029     jz     short loc_401032
.text:0040102B     mov    eax, 1
.text:00401030     jmp    short loc_40107B
```

Sezione 2 Figura 8

i parametri vengono passati attraverso una serie di comandi push, ecco alcuni esempi:
push 0; lpSecurityAttributes => passa un puntatore NULL al parametro lpSecurityAttributes.

Dice alla funzioni di utilizzare le impostazioni di sicurezza predefinite.

push offset Subkey; SOFTWARE\\Microsoft\\Windows NT\\CurrentVe”..

=> passa l'argomento della sotto-chiave di registro indicata alla chiamata di funzione. In altre parole sta indicando dove andare a creare la chiave di registro.

Alla locazione 00401017 il parametro rappresenta il percorso della chiave di registro che la funzione RegCreateKeyExA sta cercando di creare. Il percorso è **SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon**; questa chiave di registro è una componente molto sensibile in Windows in quanto si occupa, tra le funzioni più rilevanti, dell'autenticazione dell'utente e creazione della sessione utente.

```

*.text:00401021          call    ds:RegCreateKeyExA
*.text:00401027          test    eax, eax
*.text:00401029          jz     short loc_401032
*.text:0040102B          mov     eax, 1
*.text:00401030          jmp     short loc_40107B

```

Sezione 2 Figura 9

Alla locazione di memoria 00401027 viene fatta una operazione logica di confronto AND tra lo stesso registro (eax). Queste operazioni vengono fatte per prendere decisioni sul flusso del programma. Un'operazione logica AND tra lo stesso registro ha sempre risultato 0.

Attraverso questa comparazione si identifica il valore dello ZF (Zero Flag); se il risultato della comparazione è 0 lo ZF verrà impostato ad 1;

jz è un salto condizionale che avviene quando il valore dello ZF è 1.

Date queste nozioni teoriche, si può dedurre che il salto condizionale alla locazione 00401029 verrà compiuto, in quanto il risultato della comparazione **test eax, eax** è 0.

3.

Traducendo in linguaggio C, il codice Assembly in analisi potrebbe essere scritto in questo modo:

```

if ( eax==0){
    salta alla locazione 00401032
}
else{
    continua con il codice}

```

Il costrutto **if-else** in linguaggio C è utilizzato per controllare il flusso di esecuzione del programma in base ad una condizione booleana (vero o falso). Se la condizione è vera (se `eax == 0`) eseguirà una certa parte di codice (salta alla locazione 00401032); se la condizione è falsa allora eseguirà un'altra parte del codice (continua con il codice).

```

*.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

```

Sezione 2 Figura 10

4.

Il `ValueName` alla locazione 0040103E è `GinaDLL`. GINA (Graphical Identification and Authentication) è un componente del sistema operativo Windows che serve alla gestione dell'interfaccia grafica di autenticazione all'avvio del sistema operativo.

Probabilmente il malware, dopo aver ottenuto la persistenza, assegna alla chiave di registro il valore `GinaDLL` per modificare il comportamento del sistema operativo durante la fase di autenticazione di Windows.

Conclusioni:

Nella task 2 viene analizzato come il malware ottiene la **persistenza** e come si maschera da file legittimo.

Persistenza: è la capacità dei malware di rimanere nascosti e attivi all'interno del sistema anche dopo il reboot.

Innanzitutto, crea una nuova chiave nel registro di sistema. Questa chiave viene posizionata in una specifica area del registro, nota come "/Winlogon". È essenziale notare che il registro di sistema è una parte critica del sistema operativo Windows in cui vengono memorizzate varie configurazioni e impostazioni. La sezione "/Winlogon" è di particolare importanza, poiché contiene le informazioni necessarie per l'autenticazione durante il processo di avvio del sistema operativo.

Successivamente, il malware si nasconde dietro l'apparenza di un file legittimo. Questo avviene tramite l'impostazione di un valore specifico all'interno della chiave di registro appena creata. Questo valore, noto come 'Gina.dll', è fondamentale per il funzionamento del sistema di autenticazione di Windows. Il malware sfrutta questa caratteristica, assumendo l'identità di un file legittimo per eludere la rilevazione.

In sintesi, attraverso l'uso di tecniche sofisticate di manipolazione del registro di sistema e mascheramento della propria identità dietro file legittimi, il malware riesce a garantire la sua persistenza nel sistema, consentendo così di eseguire le sue attività dannose in modo discreto e persistente.

Task 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?
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.

Contenuto del capitolo in breve:

Durante la terza giornata di attività, il team di analisi si è dedicato a un'approfondita esplorazione delle sezioni di codice specifiche del malware, circoscritte all'interno dell'intervallo di indirizzi di memoria compreso tra 00401080 e 00401128.

Questo segmento critico del codice ha richiesto un'analisi dettagliata al fine di identificare il parametro "ResourceName", comprendere le funzionalità implementate dal malware in questa porzione e sviluppare un diagramma di flusso per delineare le interazioni tra la sezione in questione e la funzione principale del programma.

Per raggiungere tali obiettivi, il team ha impiegato una combinazione di strumenti avanzati, tra cui IDA Pro e OllyDBG.

Analisi del codice:

Andando ad analizzare con il tool **IDA Pro** il malware nella porzione di codice compresa tra l'indirizzo di memoria 00401080 e 00401128 sono state identificate le seguenti sezioni di codice. Queste appartengono al modulo denominato (HMODULE).

Di seguito sono riportati gli screen delle sezioni in esame:

```
.text:00401080 ; int __cdecl sub_401080(HMODULE hModule)
.text:00401080 sub_401080    proc near               ; CODE XREF: _main+3F↓p
.text:00401080
.text:00401080     hResData      = dword ptr -18h
.text:00401080     hResInfo      = dword ptr -14h
.text:00401080     Count         = dword ptr -10h
.text:00401080     var_C          = dword ptr -8Ch
.text:00401080     Str           = dword ptr -8
.text:00401080     File           = dword ptr -4
.text:00401080     hModule        = dword ptr 8
.text:00401080
.text:00401080     push    ebp
.text:00401081     mov     ebp, esp
.text:00401083     sub     esp, 18h
.text:00401086     push    esi
.text:00401087     push    edi
```

Sezione 2 Figura 11

```
.text:004010B8 ; -----
.text:004010B8
.text:004010B8 loc_4010B8:                           ; CODE XREF: sub_401080+2F↑j
.text:004010B8     mov     eax, lpType
.text:004010B8     push    eax                   ; lpType
.text:004010BD     mov     ecx, lpName
.text:004010BE     push    ecx                   ; lpName
.text:004010C4     mov     edx, [ebp+hModule]
.text:004010C5     push    edx                   ; hModule
.text:004010C8     push    edx
.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 ; -----
```

Sezione 2 Figura 12

```
.text:004010DF ; -----
.text:004010DF
.text:004010DF loc_4010DF:                           ; CODE XREF: sub_401080+56↑j
.text:004010DF     mov     eax, [ebp+hResInfo]
.text:004010E2     push    eax                   ; hResInfo
.text:004010E3     mov     ecx, [ebp+hModule]
.text:004010E6     push    ecx                   ; hModule
.text:004010E7     call    ds:LoadResource
.text:004010ED     mov     [ebp+hResData], eax
.text:004010F0     cmp     [ebp+hResData], 0
.text:004010F4     jnz     short loc_4010FB
.text:004010F6     jmp     loc_4011A5
.text:004010FB ; -----
```

Sezione 2 Figura 13

```

.text:004010FB ; -----
.text:004010FB
.text:004010FB loc_4010FB:                                ; CODE XREF: sub_401080+74↑j
→ .text:004010FB      mov     edx, [ebp+hResData]
→ .text:004010FE      push    edx                           ; hResData
→ .text:004010FF      call    ds:LockResource
→ .text:00401105      mov     [ebp+Str], eax
→ .text:00401108      cmp     [ebp+Str], 0
→ .text:0040110C      jnz    short loc_401113
→ .text:0040110E      jmp    loc_4011A5
.text:00401113 ;

```

Sezione 2 Figura 14

```

.text:00401113 ;
.text:00401113
.text:00401113 loc_401113:                                ; CODE XREF: sub_401080+8C↑j
→ .text:00401113      mov     eax, [ebp+hResInfo]
→ .text:00401116      push    eax                           ; hResInfo
→ .text:00401117      mov     ecx, [ebp+hModule]
→ .text:0040111A      push    ecx                           ; hModule
→ .text:0040111B      call    ds:SizeofResource
→ .text:00401121      mov     [ebp+Count], eax
→ .text:00401124      cmp     [ebp+Count], 0
→ .text:00401128      ja     short loc_40112C
→ .text:0040112A      jmp    short loc_4011A5
.text:0040112C ;

```

Sezione 2 Figura 15

1.Nome del parametro della funzione ResourceName :

Utilizzando il software **OllyDBG**, è stato possibile individuare il valore del parametro “resource name”, che era impostato su “**TGAD**”.

004010BD	/ M	00000000	MOV EAX,DWORD PTR DS:[400000]	
004010BE	.	50	PUSH EAX	ResourceType => "BINARY"
004010C4	.	8B0D 34804000	MOV ECX,DWORD PTR DS:[408034]	Malware_.00408038
004010C5	.	51	PUSH ECX	ResourceName => "TGAD"
004010C8	.	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]	hModule
004010C9	.	52	PUSH EDX	FindResourceA
		FF15 28704000	CALL DWORD PTR DS:[<&KERNEL32.FindResou	

Sezione 2 Figura 16

Dopo aver caricato il malware in OllyDBG, è stata **analizzata la memoria** durante l'esecuzione attraverso il debugging passo passo identificando il parametro preso in esame utilizzato nel **codice assembly**.

OllyDBG è un **debugger** per il sistema operativo Windows utilizzato nell'ambito del **reverse engineering** e dell'**analisi dei software**.

Consente di **esaminare, monitorare e modificare l'esecuzione di programmi**, nonché di comprendere il loro funzionamento..

Attraverso **OllyDBG** è possibile **visualizzare codice Assembly** per **monitorare la memoria e i registri di sistema**, tracciando il **flusso di esecuzione** del programma.

2. Funzionalità del malware nella sezione in esame:

Alla richiesta riguardante le funzionalità che implementa il malware in questa sezione, è stata eseguita un'analisi statica avanzata utilizzando il tool **IDA Pro** ed è stata identificata la chiamata a 4 funzioni principali, queste sono:

- **FindResourceA** : Questa funzione viene utilizzata per individuare una risorsa all'interno del modulo (generalmente un file .exe o .dll) identificato da un handle di modulo(HMODULE), come quello della sezione in esame.
- **LoadResource** : Questa funzione viene utilizzata per caricare una risorsa identificata da un handle tramite la funzione “FindResourceA”.
- **LockResource** : Questa funzione viene utilizzata per ottenere un puntatore al contenuto della risorsa caricata in memoria. Fa riferimento alla risorsa caricata dalla funzione “LoadResource”.
- **SizeofResource** : Questa funzione viene utilizzata per ottenere la dimensione, in byte, di una risorsa all'interno di un modulo eseguibile (come un file .exe o .dll). Può essere utile per sapere quanto spazio in memoria è necessario caricare per una specifica risorsa, in questo caso la risorsa è quella chiamata dalla funzione “FindResourceA”.

Queste funzioni sono comunemente utilizzate insieme per accedere e manipolare le risorse presenti all'interno di un file eseguibile su sistema operativo Windows, come icone, stringhe e altri dati che possono essere utilizzati dall'applicazione durante l'esecuzione.

Si può quindi ipotizzare che questo modulo serva a caricare in memoria una risorsa chiamata “TGAD” attraverso una serie di funzione capaci di manipolare la risorsa eseguibile del malware.

Grazie all'utilizzo del tool **Olly DBG**, oltre ad identificare il parametro ResourceName = “TGAD”, all'interno della sezione in esame è stato identificato un file eseguibile denominato “msgina32.dll”.

MSGINA32.DLL, noto anche come "Microsoft Graphical Identification and Authentication", è un file DLL (Dynamic Link Library) utilizzato nel sistema operativo Microsoft Windows.

In generale, è un componente fondamentale del sistema operativo Windows, poiché gestisce l'accesso e l'autenticazione degli utenti, che sono funzioni critiche per la sicurezza e l'integrità del sistema operativo.

3. Analisi statica del malware e chiarimenti:

Dall'**analisi statica basica** del malware, è stata inizialmente identificata la presenza significativa di alcune **funzioni** all'interno della libreria **KERNEL32.dll**. In particolare le funzioni **LoadResource**, **LockResource** e **SizeOfResource** hanno catturato l'attenzione.

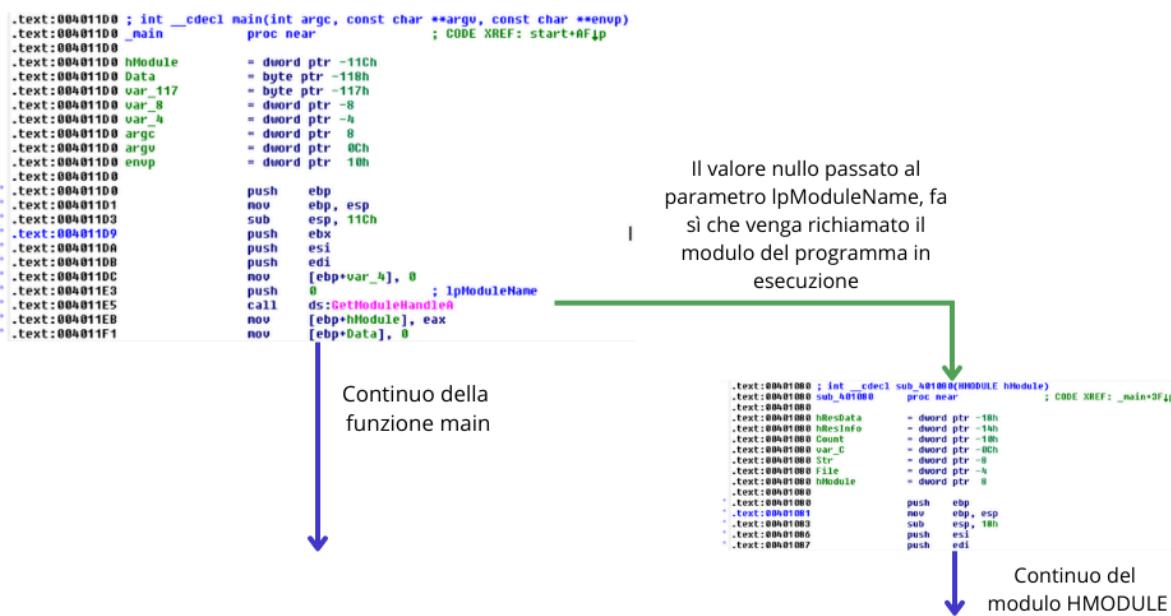
Queste funzioni sono comunemente utilizzate per **accedere alle risorse** di un eseguibile Windows, suggerendo un possibile utilizzo delle risorse di sistema da parte del malware. Inoltre, è stata osservata la presenza della sezione **.rsrc** all'interno del file eseguibile, la quale contiene risorse quali immagini, icone e stringhe.

Questo ha portato ad ipotizzare che il dropper del malware possa essere progettato per estrarre e utilizzare queste risorse durante l'esecuzione.

In conclusione le **ipotesi iniziali** che erano state fatte hanno avuto **riscontro** anche durante questa analisi.

Diagramma di flusso e chiarimenti sulle chiamate di funzione:

è stato implementato un diagramma di flusso che mostra come dalla funzione main la funzione GetModuleHandleA richiama il modulo (HMODULE) :



Sezione 2 Figura 17

La funzione `GetModuleHandleA` è un'altra funzione dell'API di Windows che viene utilizzata per ottenere l'handle di un modulo (file esequibile o DLL) caricato in memoria.

Questa funzione restituisce un HMODULE che può essere utilizzato per accedere alle risorse e ai dati all'interno del modulo. Questa funzione utilizza il parametro lpModuleName per essere chiamata.

lpModuleName è generalmente una stringa che specifica il nome del modulo. Se questo parametro è NULL come nel nostro caso in esame, GetModuleHandleA restituisce un handle per il modulo chiamante (il modulo del programma in esecuzione). Il modulo chiamato quindi, sarà il modulo HMODULE analizzato.

Questa funzione è utile quando si desidera accedere a risorse, dati o funzioni all'interno di un modulo già caricato in memoria, ma non si dispone dell'handle del modulo.

La risorsa del modulo HMODULE chiamato dalla funzione "GetModuleHandleA" corrisponde ad un eseguibile chiamato msgina32.dll.

Sempre all'interno della funzione main, subito dopo la chiamata alla funzione GetModuleHandleA si trova la funzione "GetModuleFileNameA", questa è una funzione dell'API di Windows utilizzata per ottenere il percorso completo del file di un modulo eseguibile (come un file .exe o .dll) associato a un determinato handle di modulo.

Si ipotizza quindi che restituirà il percorso del file eseguibile "msgina32.dll" caricato dal modulo (HMODULE) che a sua volta viene chiamato dalla funzione "GetModuleHandleA".

Conclusioni:

In conclusione, la giornata 3 di lavoro è stata focalizzata sull'analisi del malware, in particolare sulle sezioni di codice comprese tra gli indirizzi di memoria 00401080 e 00401128. Attraverso l'utilizzo di strumenti come IDA Pro e OllyDBG, sono stati raggiunti diversi obiettivi.

Inizialmente, mediante OllyDBG, è stato individuato il valore del parametro "ResourceName", che è stato trovato impostato su "TGAD". Successivamente, è stata condotta un'analisi statica avanzata con IDA Pro per identificare le principali funzioni implementate dal malware nella sezione in esame. Tra queste funzioni figurano FindResourceA, LoadResource, LockResource e SizeofResource, che sono comunemente utilizzate per accedere e manipolare le risorse all'interno di un file eseguibile su sistema operativo Windows. È stato inoltre confermato che, tramite un'analisi statica del malware e l'analisi delle funzioni importate dalla libreria KERNEL32.dll, è stato possibile suggerire un possibile utilizzo delle risorse di sistema da parte del malware.

Infine, è stato elaborato un diagramma di flusso che illustra come la funzione main richiami il modulo HMODULE tramite la funzione GetModuleHandleA, il quale corrisponde all'eseguibile "msgina32.dll".

Task 4

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

- Cosa notate all'interno della cartella dove è situato l'eseguibile del Malware? Spiegate cosa è avvenuto, unendo le evidenze che avete raccolto finora per rispondere alla domanda Analizzate ora i risultati di Process Monitor (consiglio: utilizzate il filtro come in figura sotto per estrarre solo le modifiche apportate al sistema da parte del Malware). Fate click su «ADD» poi su «Apply» come abbiamo visto nella lezione teorica.

Filtrate includendo solamente l'attività sul registro di Windows.

- 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.

Report sull'Analisi del Malware - Giorno 4

Durante il quarto giorno di analisi del malware, abbiamo osservato diverse attività significative all'interno del sistema, fornendo una panoramica più chiara delle azioni svolte dal malware e dei suoi effetti sul sistema operativo.

Contenuto del capitolo in breve:

Durante l'analisi del malware, è stato osservato che una volta eseguito, è stato automaticamente installato un file all'interno della cartella. Questo comportamento ha sollevato sospetti riguardo alla presenza di una funzionalità di dropper all'interno del malware.

Successivamente, passando all'analisi con Process Monitor, è stato possibile rilevare che il malware ha creato una chiave di registro utilizzando le funzioni della libreria advapi32.dll, e ha generato il percorso del file msgina32.dll.

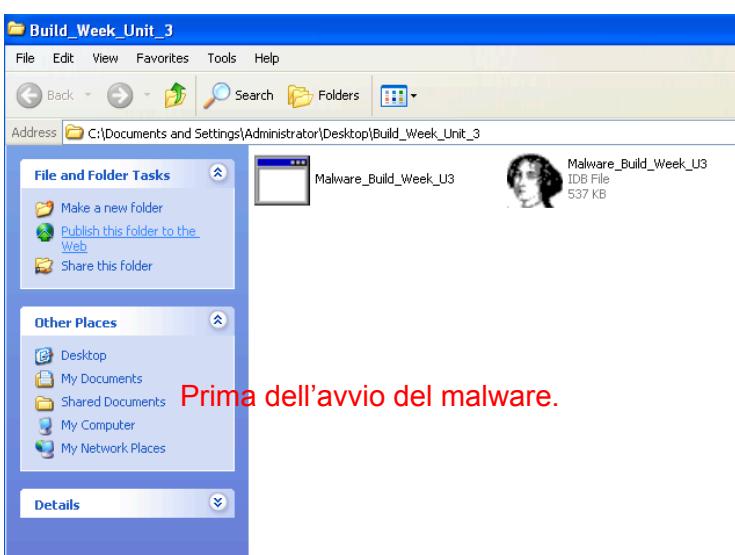
Inoltre, sfruttando IDA, è stata condotta un'analisi dettagliata di parti cruciali del codice assembly. Questo approccio ha fornito informazioni fondamentali sulla logica di funzionamento del programma, contribuendo così alla comprensione del comportamento del malware.

Attraverso l'analisi dettagliata del malware, è emerso un comportamento sospetto che indica la capacità del malware di catturare e salvare le credenziali di autenticazione al momento del log out dal sistema. Queste credenziali vengono poi archiviate in un file all'interno di una directory specifica del file system.

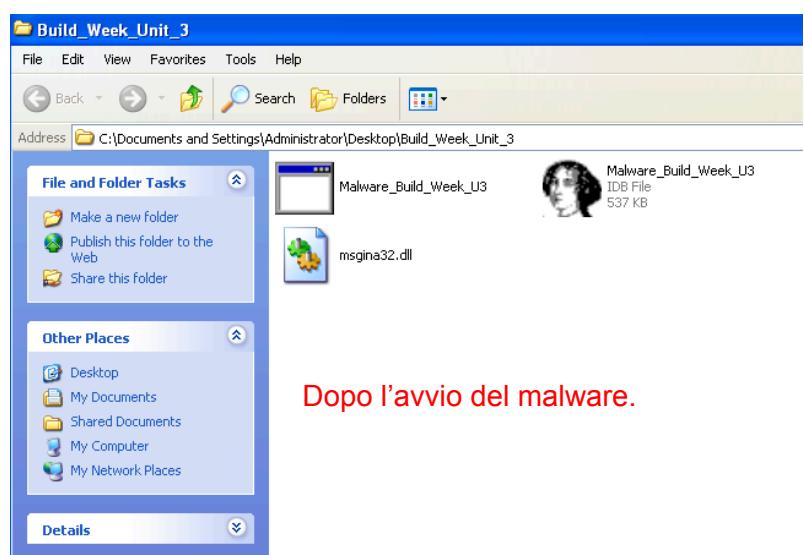
Per confermare questa ipotesi, è stato eseguito un test pratico: riavviando la macchina infettata e successivamente verificando il contenuto del file identificato durante l'analisi. Questo test ha confermato che il malware effettivamente raccoglie e salva le credenziali di autenticazione, rappresentando una minaccia significativa per la sicurezza del sistema e delle informazioni sensibili.

1. Contenuto della Cartella del Malware:

Prima dell'esecuzione del file malware, la cartella in cui è situato non conteneva alcun file aggiuntivo. Tuttavia, dopo l'avvio del malware, è stato creato un nuovo file denominato "msgina32.dll". Questo suggerisce un'infezione attiva della cartella da parte del malware, con il potenziale scopo di introdurre funzionalità dannose nel sistema.



Sezione 2 Figura 18



Sezione 2 Figura 19

2. Analisi dei Risultati di Process Monitor:

Dall'analisi dei risultati di Process Monitor, è emerso che il malware ha effettuato diverse operazioni sul registro di Windows e sul file system del sistema operativo.

Registro di Windows:

Il malware ha creato con successo una nuova chiave di registro sotto il percorso "**HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon**" utilizzando l'operazione "**RegCreateKey**". Questa chiave di registro è stata denominata "**GinaDLL**". Successivamente, è stato associato un valore di stringa a questa chiave di registro utilizzando l'operazione "**RegSetValue**". Il valore associato, denominato "**GinaDLL**", punta al percorso del file "**msgina32.dll**" precedentemente creato. Quest'azione indica chiaramente un tentativo del malware di modificare il comportamento del sistema, probabilmente introducendo codice dannoso nel processo di accesso di Windows.

8:25:22.48605...	Malware_Build_Week_U3...	900	RegCreateKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
8:25:22.48607...	Malware_Build_Week_U3...	900	RegSetValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL
8:25:22.48818...	Malware_Build_Week_U3...	900	RegCloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

Sezione 2 Figura 20

File System:

L'attività sul file system del sistema operativo è stata osservata attraverso una specifica chiamata di sistema che ha modificato il contenuto della cartella in cui è presente l'eseguibile del malware. Questa modifica coincide con la creazione del file "**msgina32.dll**" e suggerisce un'azione diretta del malware nel manipolare i file di sistema.

8:59:44.14669...	Malware_Build_Week_U3...	368	RegOpenKey	HKLM
8:59:44.14671...	Malware_Build_Week_U3...	368	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Diagnostics
8:59:44.14673...	Malware_Build_Week_U3...	368	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ntdll.dll
8:59:44.14674...	Malware_Build_Week_U3...	368	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\kernel32.dll
8:59:44.14702...	Malware_Build_Week_U3...	368	CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Unit_3\msgina32.dll
8:59:44.14709...	Malware_Build_Week_U3...	368	CreateFile	C:\Documents and Settings\Administrator\Desktop\Build_Unit_3
8:59:44.14714...	Malware_Build_Week_U3...	368	CloseFile	C:\Documents and Settings\Administrator\Desktop\Build_Unit_3
8:59:44.14715...	Malware_Build_Week_U3...	368	IRP_MJ_CLOSE	C:\Documents and Settings\Administrator\Desktop\Build_Unit_3
8:59:44.14725...	Malware_Build_Week_U3...	368	ReadFile	C:
8:59:44.14768...	Malware_Build_Week_U3...	368	WriteFile	C:\Documents and Settings\Administrator\Desktop\Build_Unit_3\msgina32.dll

Sezione 2 Figura 21

Studio del funzionamento del codice:

In che modo il malware ruba le credenziali?

Implementando WlxLoggedOutSAS, che cattura il nome utente, la password precedente e quella nuova, oltre al dominio.

In effetti, tra le molte funzioni esportate da msgina32.dll, solo una funzione (WlxLoggedOutSAS) presenta comportamenti sospetti. Le altre funzioni trasmettono semplicemente gli input all'indirizzo della funzione originale.

La funzione WlxLoggedOutSAS fa parte del meccanismo di autenticazione GINA di Windows e viene attivata quando un utente effettua il logout dal sistema

In figura la funzione:

```
; int __stdcall WlxLoggedOutSAS(PUOID pWlxContext,DWORD dwSasType,
public WlxLoggedOutSAS
WlxLoggedOutSAS proc near

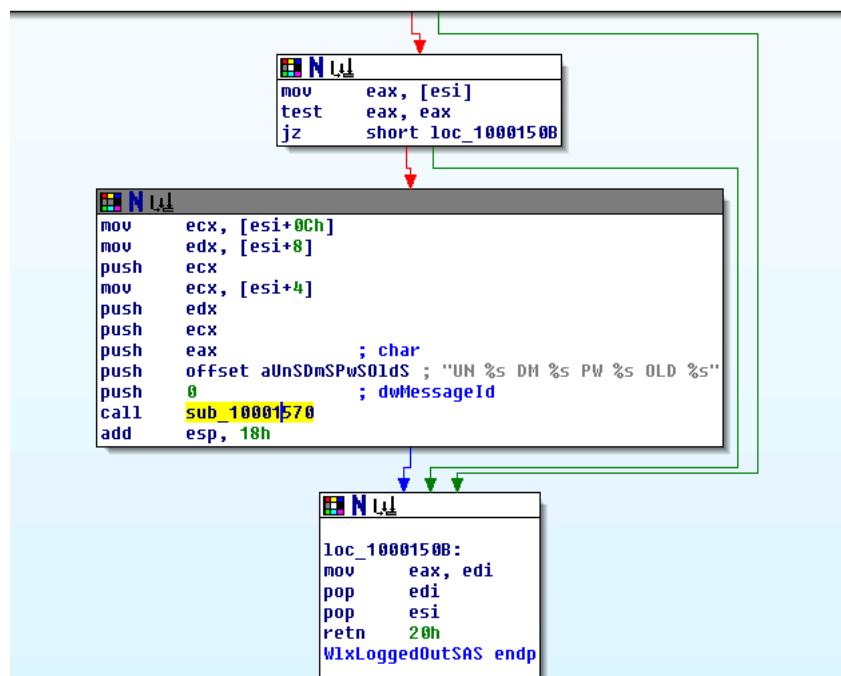
pWlxContext= dword ptr  0Ch
dwSasType= dword ptr  10h
pAuthenticationId= dword ptr  14h
pLogonSid= dword ptr  18h
pdwOptions= dword ptr  1Ch
phToken= dword ptr  20h
pNprNotifyInfo= dword ptr  24h
pProfile= dword ptr  28h

push    esi
push    edi
push    offset aWlxloggedoutsa ; "WlxLoggedOutSAS"
call    sub_10001000
push    64h
mov     edi, eax
call    ??2@YAPAXI@Z      ; operator new(uint)
mov     eax, [esp+4+pProfile]
mov     esi, [esp+4+pNprNotifyInfo]
mov     ecx, [esp+4+phToken]
mov     edx, [esp+4+pdwOptions]
add    esp, 4
push    eax
mov     eax, [esp+4+pLogonSid]
push    esi
push    ecx
mov     ecx, [esp+0Ch+pAuthenticationId]
push    edx
mov     edx, [esp+10h+dwSasType]
push    eax
mov     eax, [esp+14h+pWlxContext]
push    ecx
push    edx
push    eax
```

Sezione 2 Figura 22

La funzione accetta in input diversi parametri, tra cui pNprNotifyInfo che punta a WLX_MPR_NOTIFY_INFO, una struttura che contiene informazioni su dominio, nome utente e password per un utente.

Una copia degli input viene passata a un'altra funzione alla locazione 10001570, il che suggerisce un potenziale rischio di compromissione dei dati sensibili dell'utente attraverso la trasmissione non sicura di tali informazioni.



Sezione 2 Figura 23

Analisi della funzione chiamata alla locazione 10001570:

```

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

hMem= dword ptr -854h
var_850= word ptr -850h
var_828= word ptr -828h
var_800= word ptr -800h
dwMessageId= dword ptr 4
arg_4= dword ptr 8
arg_8= byte ptr 0Ch

mov    ecx, [esp+arg_4]
sub    esp, 854h
lea    eax, [esp+854h+arg_8]
lea    edx, [esp+854h+var_800]
push   esi
push   eax          ; va_list
push   ecx          ; wchar_t *
push   800h          ; size_t
push   edx          ; wchar_t *
call   _vsnwprintf
push   offset word_10003320 ; wchar_t *
push   offset aMsutil32_sys ; "msutil32.sys"
call   _wfopen
mov    esi, eax
add    esp, 18h
test   esi, esi
jz    loc_1000164F

```

Sezione 2 Figura 24

```

lea    eax, [esp+858h+var_800]
push   edi
lea    ecx, [esp+85Ch+var_850]
push   eax
push   ecx          ; wchar_t *
call   _wstrtime
add    esp, 4
lea    edx, [esp+860h+var_828]
push   eax
push   edx          ; wchar_t *
call   _wstrdate
add    esp, 4
push   eax
push   offset aSSS  ; "%s %s - %s "
push   esi          ; FILE *
call   _fprintf
mov    edi, [esp+870h+dwMessageId]
add    esp, 14h
test   edi, edi
jz    short loc_10001637

```



```

push   0           ; Arguments
lea    eax, [esp+860h+hMem]
push   0           ; nSize
push   eax          ; lpBuffer
push   400h          ; dwLanguageId
push   edi          ; dwMessageId
push   0           ; lpSource
push   1100h          ; dwFlags
call   ds:FormatMessageW
mov    ecx, [esp+85Ch+hMem]
push   ecx
push   edi
push   offset aErrorcodeError ; "ErrorCode:%d ErrorMessage:%s. \n"
push   esi          ; FILE *
call   _fprintf

```



```

loc_10001637:    ; "\n"
push   offset asc_1000329C
push   esi          ; FILE *
call   _fprintf
add    esp, 8
push   esi          ; FILE *
call   _fclose
add    esp, 4
pop    edi

```

Sezione 2 Figura 24,25

Analizzando l'indirizzo sub_10001570, emerge che il processo acquisisce la data (_wstrdate), l'ora (_wstrtime) e un altro valore, tutti destinati al file msutil32.sys, seguiti da un carattere di nuova linea (\n). Dalle caratteristiche osservate di questa funzione, il valore finale scritto sul file è determinato dal contenuto passato dalla funzione chiamante: UN %s DM %s PW %s OLD %s.

Sulla base di questa analisi, è ragionevole concludere che il malware genererà un file denominato msutil32.sys, che conterrà informazioni quali la data, l'ora, il Dominio, il Nome utente e la Password di un utente specifico ogni volta che viene invocata la funzione esportata WlxLoggedOutSAS di msgina32.dll. Vista l'esecuzione da parte di winlogon.exe, ci si aspetta che il file risieda nella directory C:\Windows\System32.

Test sul sistema:

Al successivo riavvio del sistema, viene presentata una schermata che richiede le credenziali d'accesso.



Sezione 2 Figura 26

Dopo un'analisi approfondita del file msutil32.sys nella directory C:\Windows\System32, è stata confermata la presenza delle credenziali salvate all'interno di tale file.

```
09/06/22 10:54:55 - UN Administrator DM MALWARE_TEST PW malware OLD (null)
09/06/22 14:15:57 - UN Administrator DM MALWARE_TEST PW malware OLD (null)
09/16/22 17:55:51 - UN Administrator DM MALWARE_TEST PW malware OLD (null)
09/17/22 14:18:15 - UN Administrator DM MALWARE_TEST PW malware OLD (null)
09/17/22 18:32:55 - UN Administrator DM MALWARE_TEST PW malware OLD (null)
02/20/24 11:22:49 - UN Administrator DM MALWARE_TEST PW malware OLD (null)
```



Sezione 2 Figura 27

Conclusioni:

L'analisi condotta sul malware ha rivelato un comportamento sospetto che denota la sua capacità di catturare e memorizzare le credenziali di autenticazione al momento del log out dal sistema. Questa scoperta, supportata da un test pratico che ha confermato il recupero delle credenziali, evidenzia una minaccia significativa per la sicurezza del sistema e delle

informazioni sensibili. La presenza di una chiave di registro e la manipolazione del file system indicano chiaramente le intenzioni del malware di influenzare il processo di accesso di Windows e di compromettere i file di sistema. Tuttavia, per una comprensione completa delle potenziali minacce e per sviluppare strategie di rimozione e protezione efficaci, sono necessarie ulteriori indagini.

Task 5

GINA (Graphic authentication & authentication) è un componente legittimo 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 legittimo 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.

Contenuto del capitolo in breve:

In risposta alle richieste della traccia, è stata compilata una lista delle potenziali conseguenze nel caso in cui un file .dll legittimo venisse sostituito con un file .dll malevolo.

Di conseguenza, sono state delineate una serie di azioni di rimedio per prevenire o risolvere una situazione di emergenza simile. Queste azioni includono l'implementazione di controlli

più rigorosi sull'integrità dei file, l'adozione di procedure di verifica e validazione più robuste durante l'installazione di nuovi componenti software e l'istituzione di politiche di gestione delle modifiche più stringenti.

Inoltre, per migliorare la comprensione del funzionamento del malware, è stato creato uno schema grafico che spiega il suo comportamento utilizzando un linguaggio di alto livello. Questo schema fornisce una panoramica chiara delle azioni intraprese dal malware e dei potenziali rischi associati.

Conseguenze sostituzione file .dll:

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

Se un file .dll lecito viene sostituito con un file .dll malevolo che intercetta i dati inseriti, possono verificarsi diverse conseguenze negative per il sistema e per gli utenti:

Furto di credenziali: Il file malevolo può intercettare le credenziali inserite dagli utenti, come username e password, consentendo agli attaccanti di accedere indebitamente a sistemi, account o dati sensibili.

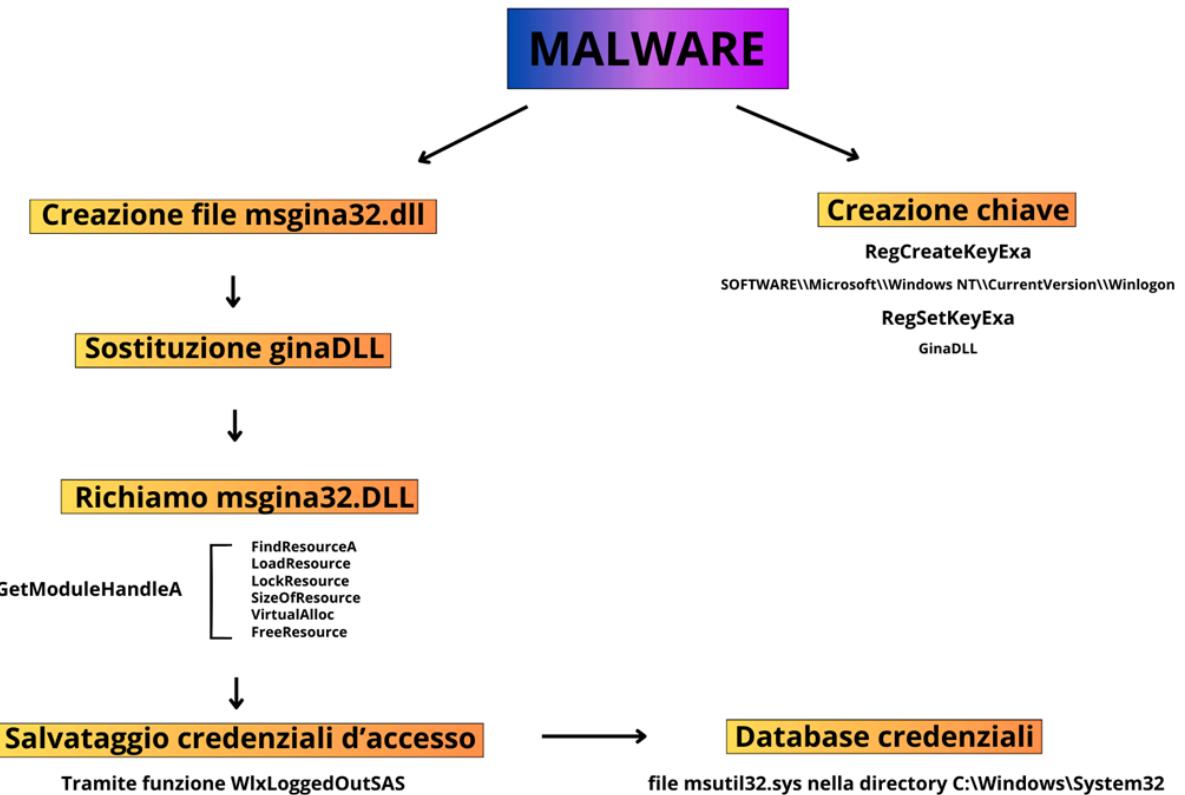
Compromissione della sicurezza: L'accesso non autorizzato ottenuto tramite le credenziali rubate può compromettere la sicurezza del sistema, consentendo agli attaccanti di eseguire azioni dannose come il furto di informazioni, la modifica dei dati o il danneggiamento del sistema.

Diffusione di malware: Il file .dll malevolo potrebbe anche essere progettato per distribuire ulteriori malware o consentire agli attaccanti di ottenere un controllo remoto sul sistema compromesso, consentendo loro di eseguire ulteriori azioni dannose.

Perdita di reputazione: Un'eventuale violazione della sicurezza dovuta alla sostituzione di un file .dll lecito con uno malevolo può danneggiare la reputazione dell'azienda o dell'organizzazione coinvolta, compromettendo la fiducia dei clienti e degli utenti.

Il file malevolo **msgina32.dll** è progettato per mimetizzarsi come il file originale agli occhi dell'utente. Tuttavia, una volta che vengono inserite le credenziali di accesso, il file malevolo entra in azione, intercettando le informazioni sensibili dell'utente. La sua funzione principale è sostituire il file legittimo, acquisire le credenziali dell'utente e le trasmette all'attaccante attraverso il file **msutil32.sys**. Questo processo avviene in modo discreto e senza che l'utente ne sia consapevole, portando a una compromissione della sicurezza del sistema.

Grafico ad alto livello:



Sezione 2 Figura 28

Remediation actions:

Isolamento del sistema compromesso: Isolare immediatamente il sistema infetto dalla rete per prevenire ulteriori danni e limitare la diffusione del malware ad altri dispositivi nella rete.

Identificazione e rimozione del malware: Utilizzare un software antivirus o anti-malware aggiornato per individuare e rimuovere il malware dal sistema compromesso. È importante eseguire una scansione completa del sistema per individuare eventuali file dannosi residui e assicurarsi che il sistema sia completamente pulito.

Ripristino della libreria GINA originale: Ripristinare la libreria GINA originale (gina.dll) utilizzando una copia pulita da una fonte affidabile. È fondamentale garantire che la versione ripristinata sia priva di modifiche e non compromessa.

Analisi delle credenziali compromesse: Identificare le credenziali di accesso compromesse salvate dal malware e prendere le misure necessarie per proteggere tali credenziali. Questo potrebbe includere la reimpostazione delle password per gli account utente interessati e il monitoraggio attivo per rilevare eventuali attività sospette o tentativi di accesso non autorizzati.

Aggiornamento delle politiche di sicurezza: Rivedere e aggiornare le politiche di sicurezza per rafforzare le difese del sistema e prevenire futuri attacchi. Questo potrebbe includere l'implementazione di misure aggiuntive di sicurezza, come l'adozione di autenticazione a più fattori e l'implementazione di controlli più stretti sull'accesso ai file di sistema.

Formazione e sensibilizzazione degli utenti: Fornire formazione agli utenti sull'importanza della sicurezza informatica e sulle pratiche consigliate per proteggere i propri dispositivi e le proprie credenziali da attacchi malware. Questo può aiutare a ridurre il rischio di futuri incidenti di sicurezza dovuti a comportamenti non sicuri degli utenti.

Monitoraggio continuo: Implementare un sistema di monitoraggio continuo per rilevare tempestivamente eventuali attività sospette o anomalie nel sistema e rispondere prontamente a potenziali minacce alla sicurezza.

Aggiornamenti e patching: Assicurarsi che il sistema operativo e le applicazioni siano aggiornati con le ultime patch di sicurezza per mitigare vulnerabilità note e ridurre il rischio di futuri attacchi.

Il malware msgina.dll è stato un problema significativo fino a Windows Vista.

Con l'avvento delle versioni successive di Windows, Microsoft ha apportato importanti miglioramenti alla sicurezza del sistema operativo, compresa l'API di autenticazione. Uno degli sviluppi chiave è stato l'introduzione di componenti come Credential Providers, che offrono un modo più robusto e sicuro per gestire l'autenticazione degli utenti rispetto al vecchio GINA.dll.

Conclusioni:

L'analisi delle potenziali conseguenze derivanti dalla sostituzione di un file .dll lecito con uno malevolo ha evidenziato gravi rischi per la sicurezza del sistema e per l'immagine aziendale. La possibilità di compromettere l'integrità del sistema e di danneggiare la reputazione dell'azienda è emersa come una minaccia significativa.

In risposta a tali rischi, sono state delineate azioni di rimedio volte a prevenire o mitigare situazioni di emergenza simili. Queste azioni includono l'implementazione di controlli più rigorosi sull'integrità dei file, l'adozione di procedure di verifica e validazione più robuste durante l'installazione di nuovi componenti software e l'istituzione di politiche di gestione delle modifiche più stringenti.

Inoltre, al fine di migliorare la comprensione del funzionamento del malware, è stato creato uno schema grafico che fornisce una panoramica chiara delle sue azioni e dei potenziali rischi associati. Questo schema rappresenta uno strumento utile per comprendere il comportamento del malware e per sviluppare strategie di mitigazione e protezione più efficaci.