

MALWARE ANALYSIS REPORT

08/05/2024

Prepared By:

Olivier Team

Prepared For:

Epic Educations r.l

INDEX

1 - Introduction

2 - Generalities of

malware 2.1 - Parameter

Analysis 2.2 - Analysis of
variables

2.3 - Section analysis

2.4 - Analysis of imported libraries

3 - Analysis of routines between the memory locations 00401017 and 00401047

3.1 - Purpose of the function call to the memory location 00401021

3.2 - Parameters passed to the leased function 00401021

3.3 - The object represented in the leased parameter 00401017

3.4 - Instructions between 00401027 and 00401029

3.5 - Translation of assembly code at 00401027/00401029 into C constructs

3.6 - Lease Call Evaluation 00401047, ValueName value 3.7 -

Overall assessment of the code extract

4 - Analysis of routines between the memory locations 00401080 and 00401128

4.1 - value of parameter 'ResourceName

4.2 - Functionality implemented by the malware in the section under examination

4.3 - Possibility of identifying functions with basic static analysis

4.4 - Conclusions on basic static analysis

4.5 - Flow chart

5 - Malware Execution / Dynamic Analysis

5.1 - Analysis with Process Monitor

5.2 - Key register analysis

5.3 - Register key value analysis

5.4 - File System Analysis

5.5 - Conclusions on malware behaviour

6 - Malware Functionality

6.1 - Graphical Identification and Authentication

6.2 - Graph of operation

7 - Glossary

1- Introduction

This report answers the questions posed using the tools and techniques learnt in the theoretical lectures.

With reference to the executable file `Malware_Build_Week_U3`, loaded on a Windows 7 operating system, we are going to take a closer look.

2- Generalities of malware

With reference to the executable file **Malware_Build_Week_U3**, answer the following questions using the tools and techniques learnt in the theory lessons:

- How many parameters are passed to the `Main()` function?; **(2.1)**
- How many variables are declared within the `Main()` function?; **(2.2)**
- Which sections are present within the executable file? **(2.3)**

Briefly describe at least 2 of those identified;

- Which libraries does Malware import? **(2.4)**

For each of the imported libraries, make assumptions based only on a static analysis of the functionality that the malware could implement.

Use the functions that are called within the libraries to support your assumptions;

2.1 - Parameter Analysis

Three parameters were passed to the `Main()` function, see figure below:

arg
c
arg
v
env
p

The parameter is the argument that is passed to the function.

We recognise the parameter because it is at a positive offset (1) from the EBP reference register (2)

```
; 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_117= byte ptr -117h
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
```

2.2 - Analysis of variables

Five variables were declared within the main() function, see figure below:

hModule

Date

var_117

var_8

var_4

Variables are defined locally in the context of a function.

We can recognise the variables because they are at a negative offset from the reference register.

```
; 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_117= byte ptr -117h
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
```

2.3 Section analysis

The executable file has the following sections (3): **.text**, **.rdata**, **.data**, **.rsrc**.

The **.text** section contains the executable code of the programme.

The **.rdata** section contains the data reading information, such as text strings and constants.

The **.data** section contains the initialized and uninitialised data used by the programme.

The **.rsrc** section contains resources such as icons, menus and dialogues used by the application.

We traced the sections of the executable thanks to a tool for static malware analysis, CFF Explorer, see figure below.

Malware_Build_Week_U3.exe									
Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	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

2.4 Analysis of imported libraries

Malware imports the following libraries (4): ADVAPI32.dll, KERNEL32.dll.

ADVAPI32.dll: This library is commonly used for advanced service operations such as registry management and security.

Malware could use this library to manipulate the registry or alter security settings.

The functions imported from this library are RegSetValueA and **RegCreateKeySetA** respectively, we can deduce from this that the malware can use the registry and thus achieve persistence (5), we will go into more detail on this topic during the dynamic analysis of the executable.

KERNEL32.dll: This library provides access to low-level functions such as memory management, input/output operations and interrupts.

Malware could use this library to manipulate memory or perform I/O operations. There are 51 imported functions, and by studying them we can deduce that:

- The malware can dynamically resolve and import external resources (GetProcAddress and LoadLibraryA).
- The malware is likely to use resources (FindResourceA , FreeResource , LoadResource , LockResource and SizeofResource)
- There are many imports that give malware the ability to operate with files. (WriteFile , CreateFileA , ReadFile , etc.).
- In addition, malware could allocate space to execute malicious code, hide sensitive data or manipulate the operation of other applications via these memory management functions.(VirtualAlloc , HeapAlloc , VirtualFree , HeapFree , etc.).

In the figure below we see, via CFF Explorer, the libraries imported by the malware

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

3- Analysis of routines between the memory locations 00401017 and 00401047

With reference to the malware under analysis, explain:

- The purpose of the function call to the memory location 00401021; **(3.1)**
- How parameters are passed to the function at 00401021; **(3.2)**
- Which object represents the parameter at location 00401017; **(3.3)**
- The meaning of the instructions between the addresses 00401027 and 00401029. (if needed, also evaluate one or two other assembly lines); **(3.4)**
- With reference to the last question, translate the Assembly code into the corresponding C construct; **(3.5)**
- Now evaluate the call to the location 00401047, what is the value of the parameter "ValueName?"; **(3.6)**
- Of the two functionalities just seen, explain which functionality Malware is implementing in this section; **(3.7)**



3.1- Purpose of the function call to the memory location 00401021

The function called is **RegCreateKeyExA**: This function creates a new key in the registry . It allows you to specify the name of the key, access rights, an optional security object that determines access to the key, and other optional parameters.

Specifically, the function creates a new registry at

path indicated at 00401017, we are going to study how malware modifies registers with dynamic analysis.

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

3.2 - Parameters passed to the leased function 00401021

The location **00401021** corresponds to the call instruction ds:**RegCreateKeyExA**.

When this instruction is executed, parameters are passed to the RegCreateKeyExA function in the stack and registers, following the calling convention of the platform, which in this case is LIFO.

In detail, the parameters are passed in this way:

Parameters are usually passed into the dedicated parameter registers (such as ECX, EDX, etc.) and onto the stack (6), in the order in which they are requested by the called function.

For the **RegCreateKeyExA** function, the order of the parameters is as follows:

hKey (handle of the parent registry key)

lpSubKey (pointer to the string specifying the name of the subkey)

Reserved (generally used as zero)

lpClass (pointer to the string containing the class name of the value to be created or NULL)

dwOptions (options f o r opening/creating the key)

samDesired (desired key access)

lpSecurityAttributes (pointer to security structures, usually NULL) **phkResult**

(pointer to a variable where the handle of the opened or created key is returned)

lpdwDisposition (indicates whether a registry key was created new or opened if it already existed).

```
.push 0 ; lpdwDisposition
lea eax, [ebp+hObject]
push eax ; phkResult
push 0 ; lpSecurityAttributes
push 0F003Fh ; samDesired
push 0 ; dwOptions
push 0 ; lpClass
push 0 ; Reserved
push offset SubKey ; "SOFTWARE\\Microsoft\\Windows NT\\CurrentVe"...
push 80000002h ; hKey
call ds:RegCreateKeyExA
```



3.3- The object represented in the leased parameter 00401017

At 00401017, the parameter passed is the offset of the subkey name in the Windows Registry. This parameter is a pointer to the subkey name that the RegCreateKeyEx function will use to create or open the specified registry key.

The subkey represents the path within the registry where key will be created or opened. In this particular case, the name of the subkey appears to be 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon'.

```
; char SubKey[]
SubKey      db 'SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon',0
; DATA XREF: sub_401000+17To
```

3.4- Instructions between 00401027 and 00401029

Instructions between the addresses **00401027** and **00401029** perform the if construct function, using the test instruction as operand and the jz instruction as condition.

```
test     eax, eax
jz       short loc_401032
```

3.5- Translation of assembly code at 00401027/00401029 into C constructs

In the image below is the instruction translated into C language

```
if (eax == 0) {
    // Blocco di codice da eseguire se eax è zero
    goto loc_401032;
}
```

3.6- Evaluation of the lease call 00401047, ValueName value

The value of the ValueName parameter is "GinaDLL".

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

```
; char ValueName[]
ValueName    db 'GinaDLL',0
; DATA XREF: sub_401000+3ETo
```



3.7- Overall assessment of the code extract

From the functions just analysed, we can deduce that the malware is designed to modify the registry in order to gain persistent access and to perform malicious actions within the Windows operating system.

In addition, the fact that the malware is attempting to set a registry value called 'GinaDLL' indicates an attempt to replace the DLL component of the Windows authentication process, which could allow the malware to collect login credentials or gain unauthorised access to the system.

4 - Analysis of routines between the memory locations 00401080 and 00401128

-What is the value of the 'ResourceName' parameter passed to the FindResourceA() function; **(4.1)**

- The succession of function calls that Malware makes in this section of code we saw during the theoretical lessons. What functionality is the Malware implementing?; **(4.2)**

- Is it possible to identify this functionality using basic static analysis ? (from day 1 in practice); **(4.3)**

- If so, please list the supporting evidence. Both of the main malware functions seen so far are called within the Main() function; **(4.4)**

- Draw a flowchart comprising the three functions; **(4.5)**

4.1- Value of the ResourceName parameter

The Value of the parameter passed to the FindResourceA function is contained in the ECX register at the time of the call. "TGAD" is the parameter lpName passed

```
* .text:004010BE      mov     ecx, lpName
* .text:004010C4      push    ecx          ; lpName
```

```
; LPCSTR lpName
lpName      dd offset aTgad          ; DATA XREF: sub_401080+3E1r
; "TGAD"
```



4.2 - Functionality implemented by the malware in the section under examination

The **FindResourceA** function belongs to the Windows system library KERNEL32.dll and is used to locate a resource within a specified module, such as an executable file or a DLL. The resource is found using the type and name of the resource specified as parameters.

The **LoadResource** function of KERNEL32.dll is used to load a resource found by FindResourceA in memory.

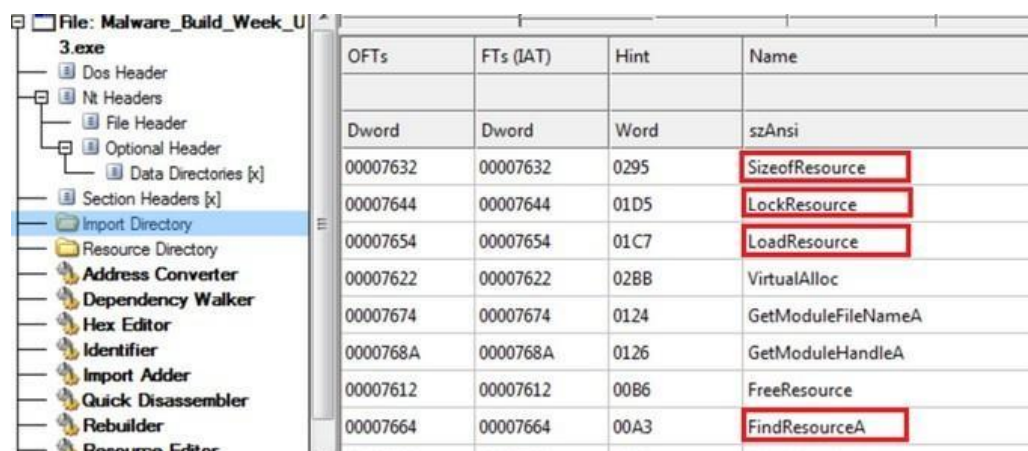
The **LockResource** function of KERNEL32.dll is used to obtain a pointer to the resource loaded in memory via LoadResource.

The **SizeofResource** function of KERNEL32.dll is used to obtain the size of a resource identified by FindResourceA.

These functions then make it possible to locate within the resource section the malware to be extracted and to load or save on disk for future execution. These types of functions are necessary features for one type of malware, the Dropper, which is a malicious programme that contains malware. When run, the dropper extracts the malware and saves it to disk. Usually, the malware is contained in the .rss section of the executable (the resource section).

4.3 - Possibility of identifying functions with basic static analysis

It is possible to identify the functions found through **IDA** as seen in section 4.2 through a tool used in basic static analysis, **CFF Explorer**.



OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00007632	00007632	0295	SizeofResource
00007644	00007644	01D5	LockResource
00007654	00007654	01C7	LoadResource
00007622	00007622	028B	VirtualAlloc
00007674	00007674	0124	GetModuleFileNameA
0000768A	0000768A	0126	GetModuleHandleA
00007612	00007612	00B6	FreeResource
00007664	00007664	00A3	FindResourceA

```
* .text:004010C9      call     ds:FindResourceA
* .text:004010E7      call     ds:LoadResource
* .text:004010FF      call     ds:LockResource
* .text:0040111B      call     ds:SizeofResource
```

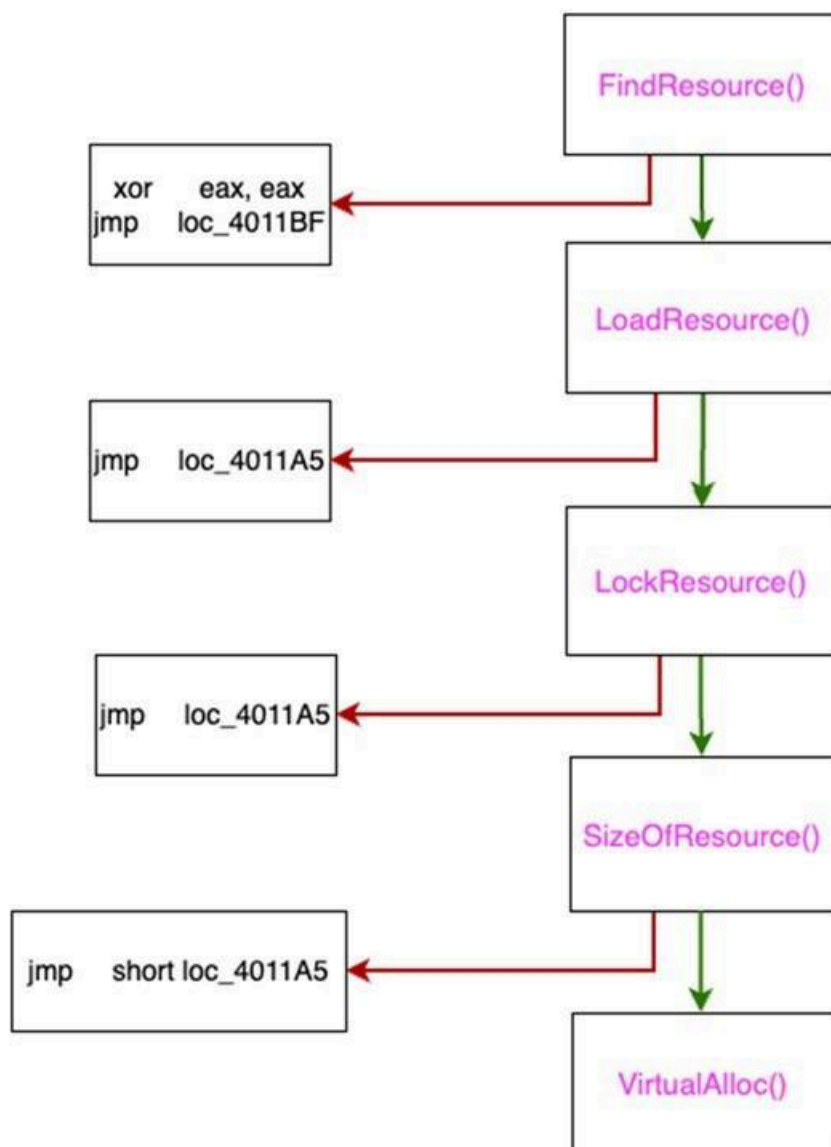


4.4- Conclusions on basic static analysis

With basic static analysis we can identify which functions are imported by the malware, but we have no way of understanding how these functions are used. We can therefore say that the static analysis provides a partial idea of how the functions within the malware work

4.5- Flow chart

The figure below gives an idea of how the functions just described operate.



5- Malware Execution / Dynamic Analysis

Prepare the environment and tools for running the Malware (hint: start mainly Process Monitor and make sure to remove any filters by clicking on the 'reset' button when prompted at start-up). Run the Malware by double-clicking on the executable icon

-What do you notice inside the folder where the malware executable is located? Explain what happened, combining the evidence you have gathered so far to answer the question; **(5.1)**

Filter by including only the activity on the Windows registry .

-What registry key is created?

-What value is associated with the created registry key?; **(5.3)**

Now switch to the display of the activity on the file system.

-What system call changed the contents of the folder where the malware executable is located?

Combine all the information gathered so far from both static and dynamic analysis to delineate how the malware works; **(5.5)**

5.1 - Analysis with Process Monitor

After the malware is executed, it creates a new file within its own directory through the succession of functions analysed above and contained in subroutine_401080 (**4.2**).

```
loc_4010B8:
mov     eax, lpType
push    eax                ; lpType
mov     ecx, lpName
push    ecx                ; lpName
mov     edx, [ebp+hModule]
push    edx                ; hModule
call    ds:FindResourceA
mov     [ebp+hResInfo], eax
cmp     [ebp+hResInfo], 0
jnz     short loc_4010DF
```

```
loc_4010DF:
mov     eax, [ebp+hResInfo]
push    eax                ; hResInfo
mov     ecx, [ebp+hModule]
push    ecx                ; hModule
call    ds:LoadResource
mov     [ebp+hResData], eax
cmp     [ebp+hResData], 0
jnz     short loc_4010FB
```

```
loc_4010FB:
mov     edx, [ebp+hResData]
push    edx                ; hResData
call    ds:LockResource
mov     [ebp+Str], eax
cmp     [ebp+Str], 0
jnz     short loc_401113
```

```
loc_401113:
mov     eax, [ebp+hResInfo]
push    eax                ; hResInfo
mov     ecx, [ebp+hModule]
push    ecx                ; hModule
call    ds:SizeofResource
mov     [ebp+Count], eax
cmp     [ebp+Count], 0
ja      short loc_40112C
```

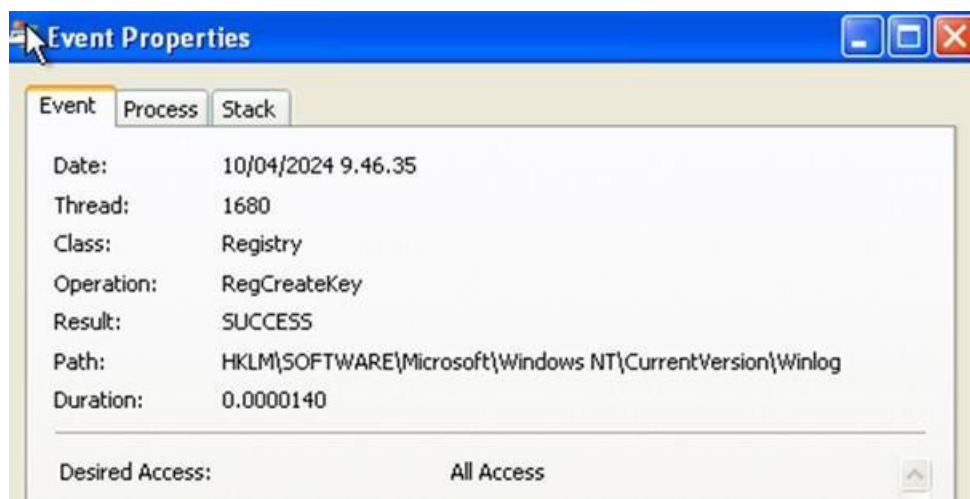
The image below shows the directory of the executable file where the new file (msgina32.dll) was created, the 4 files .id0, .id1, .nam, .til were created by IDA after the malware was uploaded to the tool.

Malware_Build_Week_U3	17/01/2024 17:48	Applicazione	52 KB
Malware_Build_Week_U3.id0	09/04/2024 14:17	File ID0	384 KB
Malware_Build_Week_U3.id1	09/04/2024 14:17	File ID1	184 KB
Malware_Build_Week_U3.nam	09/04/2024 14:17	File NAM	16 KB
Malware_Build_Week_U3.til	09/04/2024 09:20	File TIL	1 KB
msgina32.dll	09/04/2024 14:24	Estensione dell'ap...	7 KB

5.2 - Key register analysis

The image below shows the new registry key created by the malicious file (Winlogon (7)). We can see the new registry key created on the Process monitor, filtering only the activities on the Windows registry.

Time of Day	Process Name	PID	Operation	Path
12:24:56.3182070	Malware_Build_...	2904	RegCreateKey	HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon



In the image below we see the new key created in Regedit.



5.3- Register key value analysis

The value associated with the new registry key and the path

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\GinaDLL.

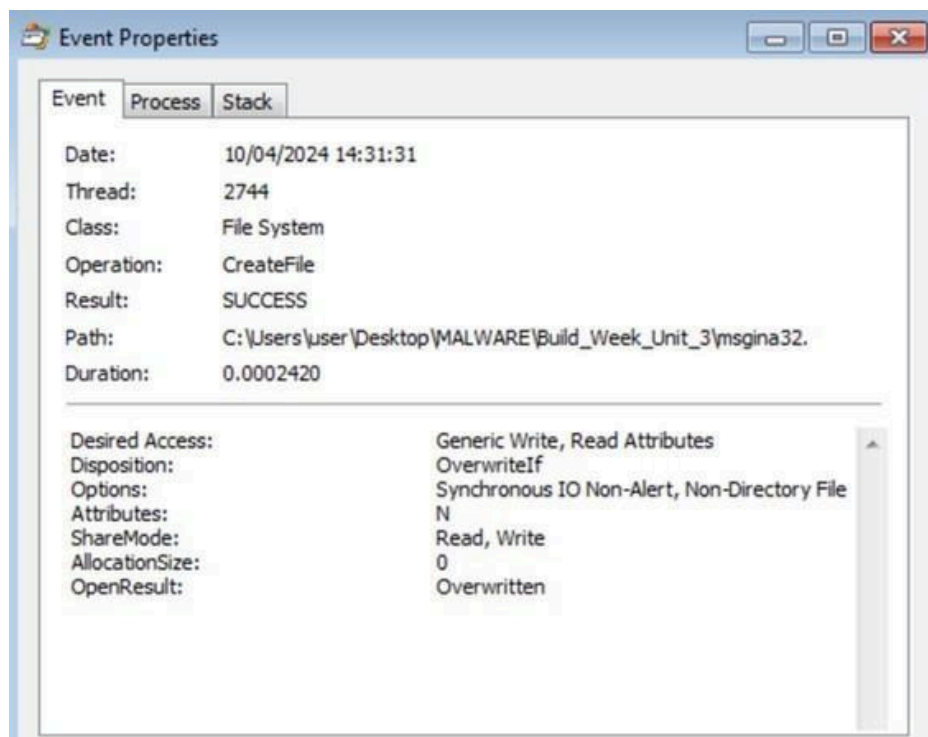
The string tells us the path to the library that is executed upon user authentication (winlogon.exe).

```
12:24:56.3182343 Malware_Build_... 2904 RegSetValue HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL
12:24:56.3182628 Malware_Build_... 2904 RegCloseKey HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Winlogon
```

5.4- File System Analysis

The image below shows the File System section, on Process Monitor, where the **msgina32.dll** file is created

CreateFile	C:\Users\user\Desktop\MALWARE\Bu...	SUCCESS	Desired Access: G...
WriteFile	C:\Users\user\Desktop\MALWARE\Bu...	SUCCESS	Offset: 0, Length: 4...
WriteFile	C:\Users\user\Desktop\MALWARE\Bu...	SUCCESS	Offset: 4.096, Leng...
CloseFile	C:\Users\user\Desktop\MALWARE\Bu...	SUCCESS	



5.5- Conclusions on malware behaviour

After a thorough study of the malware, we came to the conclusion that it is a dropper that subsequently downloads a credential stealer (msgina32.dll) onto the target machine.

The 'dropper' in question secretly installs the credential stealer on the compromised computer system. Its main purpose is to 'drop' (hence the term) or release the malicious component (**msgina32.dll**) into the system without the user's knowledge. Here is a step-by-step explanation of how it works:



Phase 1: Infiltration

The dropper makes its way into the target system through techniques such as:

- Deceptive downloads
- Malicious e-mail attachments
- Shared files
- Exploits of software vulnerabilities
- System infection via USB devices

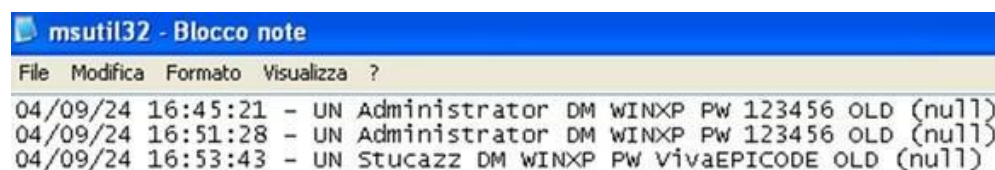
Phase 2: Evasion and Persistence

Once executed, the dropper evades detection by antivirus (precisely because the executable file is not recognised as malicious) and establishes persistence on the system by modifying registry keys to run automatically at operating system start-up. **Step 3: Execution of the Credential Stealer**

The dropper imports the malicious DLL library designed to work with the access interface of Windows, known as **GINA (Graphical Identification and Authentication)**. GINA was used in versions of Windows up to Windows Vista to manage login screens. A malicious DLL modifies GINA's behaviour to capture user credentials when they are entered during the login process.

Step 4: Creation of the system file and data exfiltration

The file 'msgina32.dll' creates the file 'msutil32.sys' where it will store the stolen credentials. The path to access that file is as follows: **%SystemRoot%\System32\msutil32.sys**.



```
File  Modifica  Formato  Visualizza  ?
04/09/24 16:45:21 - UN Administrator DM WINXP PW 123456 OLD (null)
04/09/24 16:51:28 - UN Administrator DM WINXP PW 123456 OLD (null)
04/09/24 16:53:43 - UN Stucazz DM WINXP PW vivaEPICODE OLD (null)
```

Subsequently, this file can be transmitted to the cybercriminal via the Internet, thus completing the theft of information.

It is important to note that modern versions of Windows have replaced GINA with Credential Provider, but the concept of credential interception by malicious components remains relevant. Defences against such threats include the use of up-to-date antivirus software, firewalls, implementation of robust security policies, and user training on identifying phishing tactics and the risks of downloading software from untrusted sources.

6 - Malware Functionality

GINA (Graphical identification and authentication) is a lawful component of Windows that enables user authentication via a graphical interface - that is, it allows users to enter their username and password in the classic Windows box, like the one in the picture on the right that you also use to log into the virtual machine.

-What can happen if the lawful .dll file is replaced with a malicious .dll file that intercepts the input data?

Based on the answer above, outline the profile of the malware and its functionality.

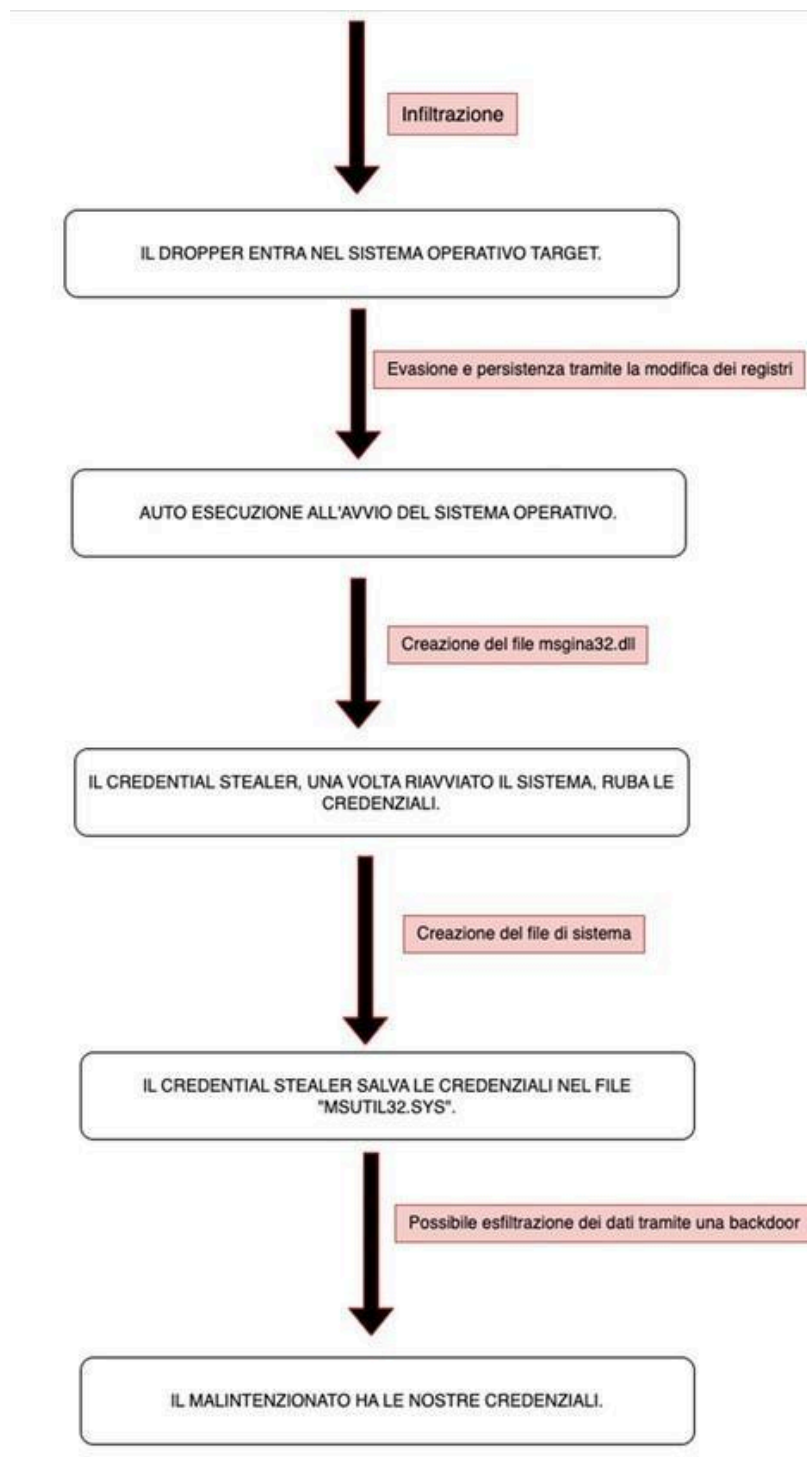
Connect all the dots to create a graph representing its purpose at a high level; **(6.2)**



6.1- Graphical Identification and Authentication

When imported, the malware replaces the lawful .dll file with a malicious .dll file (msgina32.DLL) that intercepts the credentials entered by the user and then inserts them into the file at the path **%SystemRoot%\System32\msutil32.sys**.

6.2- Graph of operation



7- Glossary

OFFSET (1) - Address difference between two points, usually used to access variables, data or jump instructions. It is essentially a distance calculated with respect to a reference point.

EBP (2) - The EBP register, also known as the Base Pointer, is a register used in x86 processor architectures. Its main role is to point to the base of the current stack frame in a programme. When a function is called, the address to return to and the function parameters are often stored in the stack. EBP is used to maintain a constant reference point, so that all local parameters and variables can be accessed using an offset to EBP, regardless of how much other data has been placed on the stack.

SECTIONS (3)- An executable file in PE (Portable Executable) format is divided into several sections with specific purposes. These sections organise the file so that the operating system can load it into memory and execute it efficiently.

LIBRARIES (4) - Windows libraries are collections of pre-written code that developers can use in their applications to perform specific functions without having to write all the code from scratch. There are two main formats:

- **DLL** (Dynamic Link Libraries): These are shared libraries that contain functions, data and resources that Windows applications can use. DLLs help promote code modularity, code reuse, memory efficiency and disk space reduction. When an application runs, the functions of a DLL are not loaded into memory until they are actually called (dynamic linking).
- **Static Libraries** (.lib): Unlike DLLs, static libraries are embedded directly in the application executable at compile time. This means that each application has its own copy of the library code, leading to an increase in the size of the executable but reducing external dependencies and loading times at start-up.

PERSISTENCE (5) - Persistence , in the context of computing and computer security, refers to the ability of an element, such as a file, process or software, to remain active or persist in the system even after the user has attempted to delete it or after a system restart. This concept is particularly relevant when it comes to malware, as many malware seek to ensure their persistence in the system in order to maintain access and continue to damage or steal data without being easily detected or removed by users or antivirus software. In essence, persistence is the ability of an element to 'stay alive' in the system despite efforts to remove it.



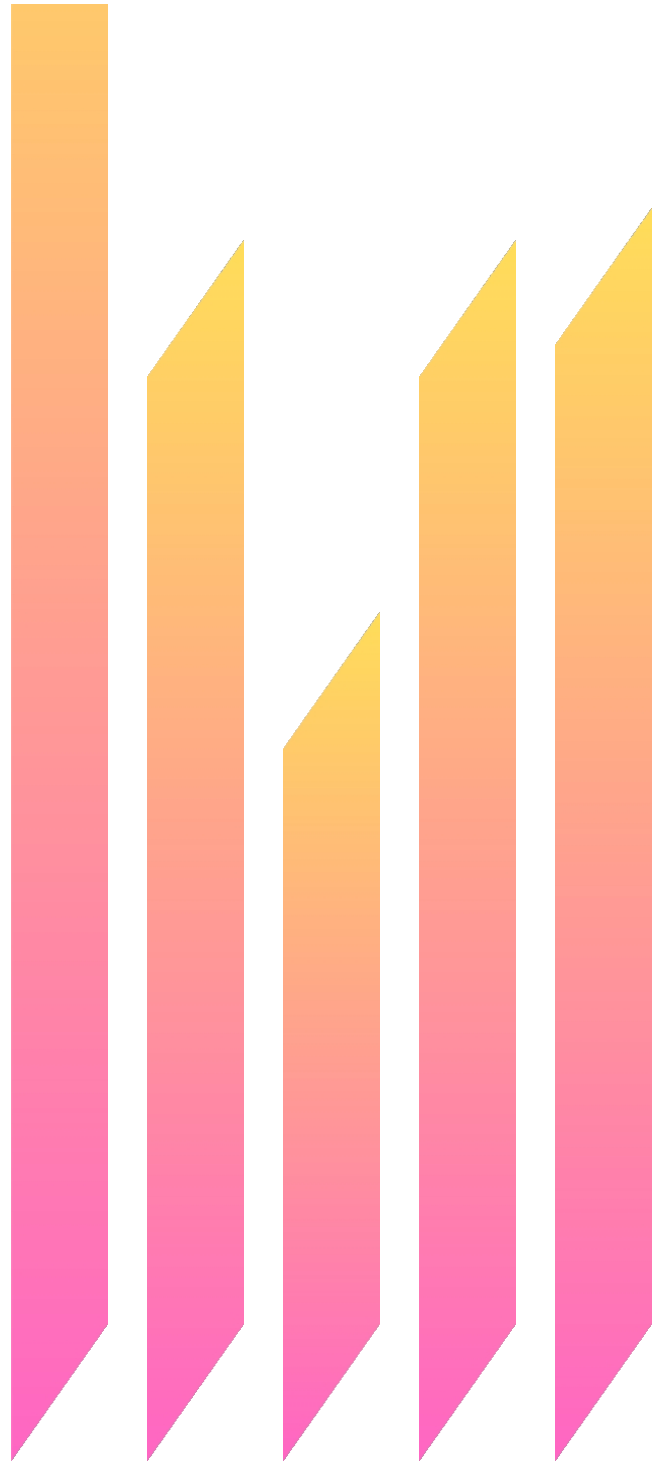
STACK (6) - The stack is a fundamental data structure used in computer science for organising and managing information. In short, it is a collection of elements ordered according to a **LIFO** (Last In, First Out) access principle, which means that the last element entered is the first to be extracted.

In the context of computer programmes, the stack is often used to manage function calls and local variables during code execution. When a function is called, its local variables are allocated to the stack and are removed when the function returns.

This stack management of function calls makes it possible to keep track of programme execution and efficiently manage local variables.

WINLOGON (7) - Winlogon is a critical component of the Windows operating system, responsible for managing user logon and logout. Its full name is Windows Logon Process, and it is a process that operates in background to manage various functions related to security and user access.





THANK YOU FOR YOUR ATTENTION

08/05/2024

Prepared By:

Oliviero Camarota

Pignatello Giuseppe

Vitale Francesco

Scopece Francesco Pio