Team Precuzzo - EPICODE

S10 - L1

TEAM

Iosif Castrucci
Donato Tralli
Gianpaolo Miliccia Mendoza

IMPORT DELLE LIBRERIE

Using CFF Explorer, we see from the import directory section that the malware U3_W2_L1 imports 4 fundamental libraries for its operation:

1. **Kernel32.dll Description**: This library contains the core functions of the Windows operating system, used for managing memory, processes and threads, I/O operations (input/output), and more. Main Functions:

- **CreateFile**: To create or open files or devices.
- **ReadFile**: To read data from a file or device.
- **WriteFile**: To write data to a file or device. Importance: Essential for performing basic operating system operations that are fundamental for the functioning of any Windows application.

2. **Advapi32.dll Description**: This library provides advanced Windows API functions that allow interaction with the system registry and management of Windows services. Main Functions:

- **RegOpenKeyEx**: To open a registry key.
- **RegSetValueEx**: To set a registry value.
- **StartService**: To start a Windows service. Importance: Crucial for applications that need to modify the system registry or manage services, often used by malware to persist in the system or alter critical configurations.

IMPORT DELLE LIBRERIE

MSVCRT.dll Description: This library, part of Microsoft Visual Studio, contains functions from the standard C runtime library, used for string manipulation, memory allocation, file management, and more.

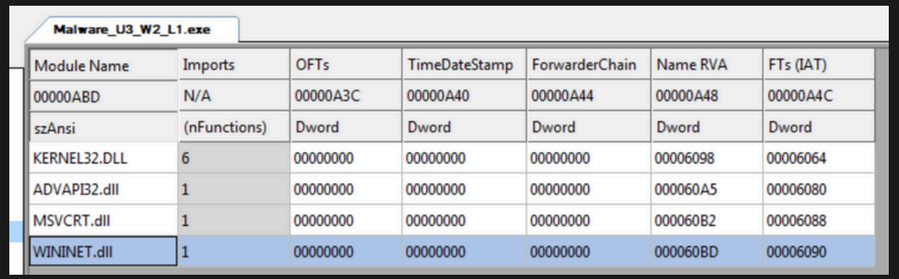
Main Functions:

malloc: To allocate memory dynamically.
printf: To print formatted text to the console.
fopen: To open a file.
Importance: Essential for many applications written in C and C++, including malware, to perform basic operations of memory and string management.

Wininet.dll Description: This library provides functions to implement network services and supports network protocols such as FTP, HTTP, and NTP.

Main Functions:

InternetOpen: To initialize an Internet session.
InternetConnect: To connect to a server.
HttpSendRequest: To send an HTTP request to the server.
Importance: Crucial for applications that need to communicate over the Internet, used by malware to download additional components, exfiltrate data, or communicate with a command and control (C2) server.



Malware_U3_W2_L1.exe

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
00000ABD	N/A	00000A3C	00000A40	00000A44	00000A48	00000A4C
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.DLL	6	00000000	00000000	00000000	00006098	00006064
ADVAPI32.dll	1	00000000	00000000	00000000	000060A5	00006080
MSVCRT.dll	1	00000000	00000000	00000000	000060B2	00006088
WININET.dll	1	00000000	00000000	00000000	000060BD	00006090

Conclusion

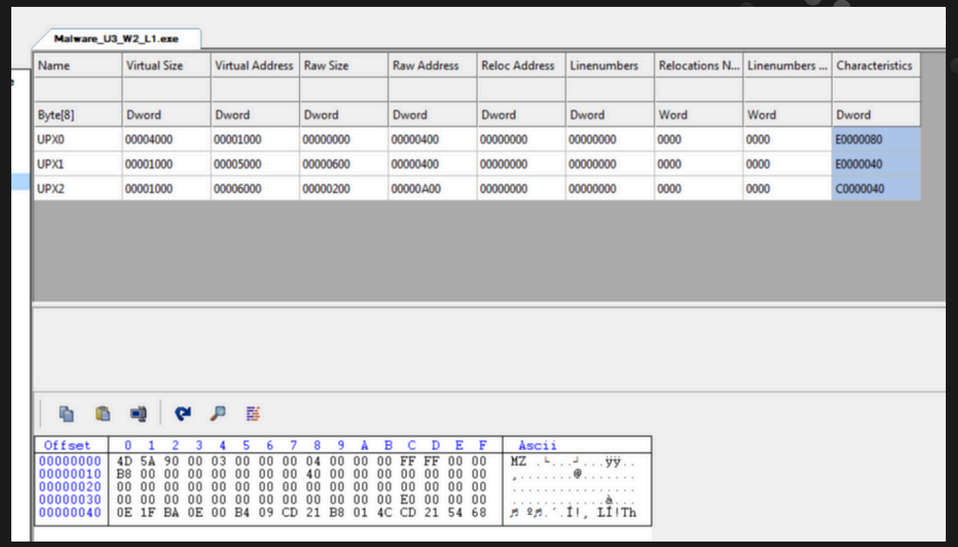
The analysis of the libraries imported by malware provides a clear indication of its capabilities and potential impact on the system. Using tools like CFF Explorer, we can quickly identify the dependencies of malware and obtain crucial information to counter threats.

SELEZIONE DEI MALWARE

From CFF Explorer, examining the "section header" section, we see that the executable consists of 3 sections. Unfortunately, the malware has hidden the real names of the sections, making it difficult to determine their specific functions. However, we can hypothesize the types of sections that might be present in a typical executable and the importance of their analysis.

Typical Sections of an Executable

1. .text: This section usually contains the executable code of the application. It is essential because it holds the instructions that the processor will execute.
2. .data: This section contains the initialized data used by the program. Here are global and static variables that the program uses and modifies during execution.
3. .rdata: This section is dedicated to read-only data, such as constant text strings and other data that do not change during program execution.
4. .bss: This section contains uninitialized data. It is used to declare global or static variables that do not have an initial value.
5. .rsrc: This section contains resources such as icons, cursors, strings, and other data used by the application.



OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
N/A	000060C8	0000	LoadLibraryA
N/A	000060D6	0000	GetProcAddress
N/A	000060E6	0000	VirtualProtect
N/A	000060F6	0000	VirtualAlloc
N/A	00006104	0000	VirtualFree
N/A	00006112	0000	ExitProcess

IMPORTANCE OF SECTION ANALYSIS

Analyzing the sections of an executable can provide critical information about the malware's functionality. Even if the section names are hidden, we can use reverse engineering techniques to determine their content and purpose:

Disassemble the Code: Use tools like IDA Pro or Ghidra to analyze the code in the .text section and understand what the malware does.
Monitor Data Access: Analyze how the malware accesses and uses data in the .data and .bss sections to uncover key variables and sensitive data the malware manipulates.
Analyze Resources: Examine the .rsrc section to reveal graphical elements, error messages, and other resources the malware might use for its interface or to deceive the user.
Identify Common Patterns: Compare the malware sections with those of other known malware to identify common techniques and suspicious behaviors.

Conclusion

Although the malware has hidden the section names to complicate analysis, it is possible to use various tools and reverse engineering techniques to better understand its functionalities. This type of analysis is essential for developing effective countermeasures and protecting systems from cyber attacks.

The malware in question represents an advanced threat, designed to evade basic static analysis. This evasion technique is reflected in the presence of the imported functions "LoadLibrary" and "GetProcAddress," suggesting the use of dynamic library loading during execution (runtime). This strategy allows the malware to hide information about imported libraries, making it more difficult to identify its real functionality through static analysis.

Details of the Techniques Used

LoadLibrary: This API allows the malware to load a dynamic library (DLL) during execution, instead of declaring it statically. This means that static analysis of the executable file cannot determine in advance which libraries will be used by the malware, complicating the identification of its dependencies and behaviors.

GetProcAddress: Used in combination with LoadLibrary, this API allows obtaining the address of an exported function from a specific DLL. This mechanism allows the malware to invoke specific functions dynamically, without these being visible during static analysis.

Evasion of Static Analysis

Dynamic loading and runtime resolution of functions significantly complicate static analysis. Information about the modules used and the functions called is resolved only during execution, meaning that a traditional static analyzer cannot see these relationships in the original binary file.

Static analysis tools, such as disassemblers and decompilers, reveal only what is explicitly present in the file. Therefore, if functions and libraries are not loaded until runtime, they do not appear in the analysis output.

Importance of Dynamic Analysis

Faced with such advanced techniques, dynamic analysis becomes crucial. This type of analysis involves running the malware in a controlled environment (sandbox) to observe its behavior in real-time.

Behavior Monitoring: Dynamic analysis can reveal how the malware interacts with the system, which libraries it loads, and which functions it invokes, overcoming the limitations of static analysis.

Collection of Indicators of Compromise (IoC): During execution, various indicators of compromise can be identified, such as changes to the system registry, creation of suspicious files, and abnormal network communications.

Dynamic Analysis Tools: Using tools like Cuckoo Sandbox, Detux, or API tracing tools can provide a detailed analysis of the malware's behavior.

Conclusion

The complexity and evasion techniques of this malware require a combined approach of static and dynamic analysis to fully understand its capabilities and develop effective countermeasures. Although the malware uses advanced techniques to hide its activities, robust dynamic analysis can reveal its hidden operations, enabling security analysts to create adequate defenses.