MALWARE ANALYSIS

Basic Static Analysis

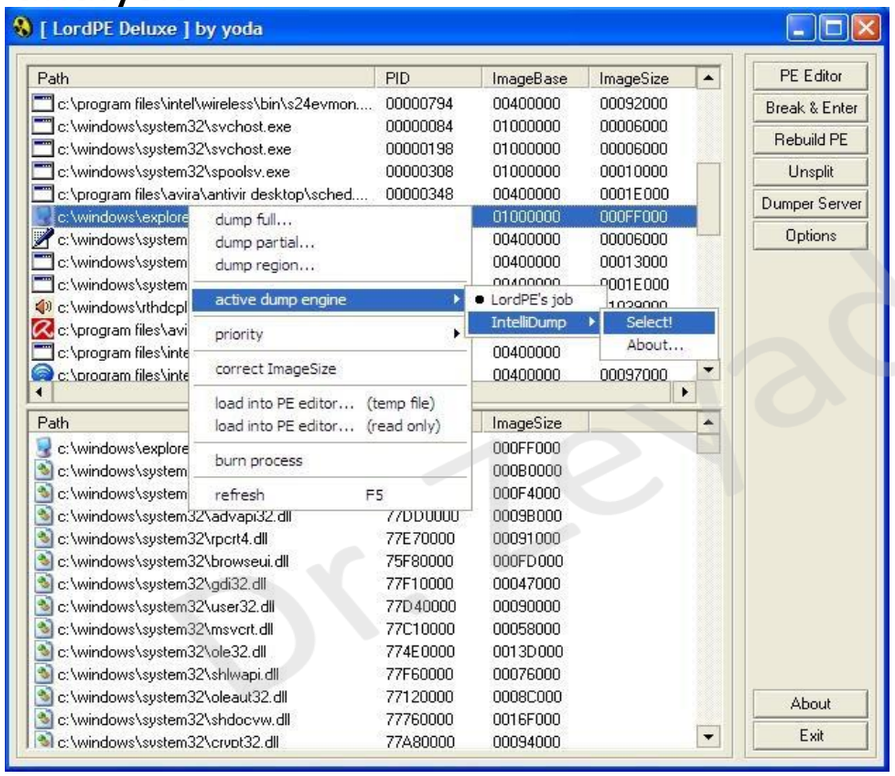
Dr. Zeyad Safaa Younus Saffawi

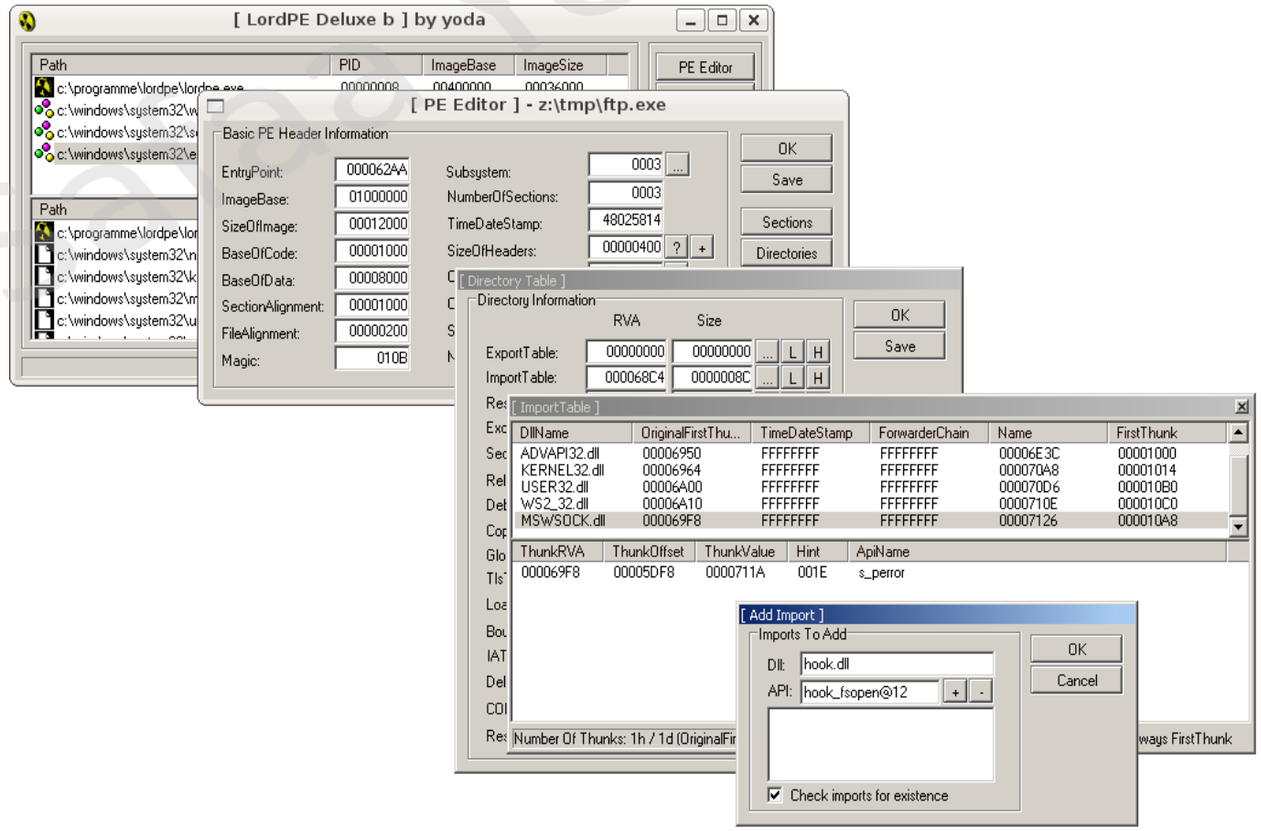
Portable Executable (PE) File Format

- The **Portable Executable (PE) format** is the standard file format for Windows executables (.exe), dynamic-link libraries (.dll), and object files (.obj).
- The PE format defines a data structure that contains all the information required by the Windows loader to manage and execute the wrapped code.
- The **PE header** provides critical metadata about the executable, including details about the code, application type, required libraries, and memory requirements.
- For malware analysts, the PE header is particularly valuable, as it offers insights into the internal structure of the program, its dependencies, and potential behavior. Examining the PE header can therefore assist in identifying anomalies, detecting malicious characteristics, and guiding further static and dynamic analysis.

LordPE Deluxe tool

- **LordPE Deluxe** is a portable executable (PE) file editor and a system programmer's tool designed to view, edit, and manipulate various components of **Windows executable files**. The application allows analysts to examine PE files in detail, modify headers, and explore the structure of executables and their associated dynamic-link libraries (DLLs). Additionally, LordPE Deluxe can be used to dump PE files from memory, optimize them, and analyze running programs, making it a valuable tool for both software development and malware analysis.





Linked Libraries and Functions

- **Imports** are functions used by a program that reside in a different program or library. They enable programmers to reuse existing code rather than rewriting it from scratch. These imported functions are connected to the main executable through a process known as **linking**. Libraries can be linked in three primary ways: **statically**, **at runtime**, or **dynamically**.
- For malware analysts, understanding how libraries are linked is crucial, as it provides insight into the dependencies and behavior of a program. By examining imported functions and their linking method, analysts can infer which external routines the malware relies on and predict how it interacts with the operating system or other software components, facilitating more effective analysis and detection.

Static Linking

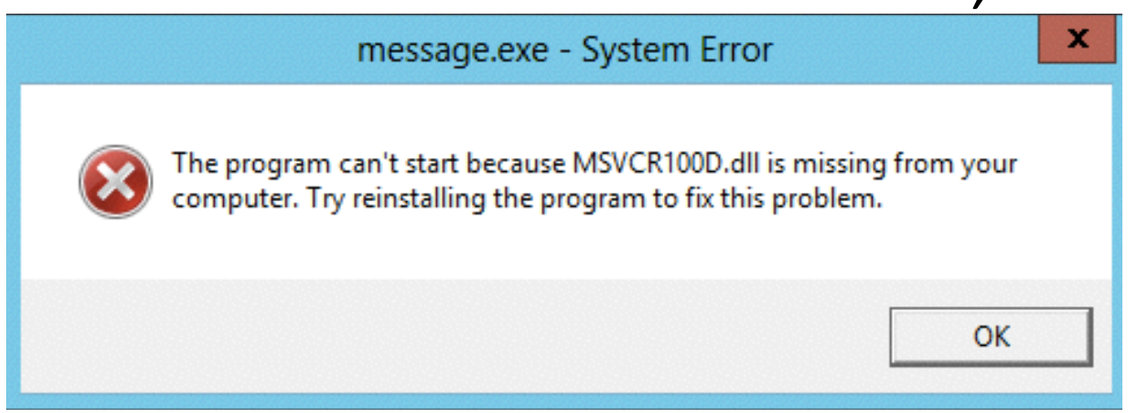
- **Static linking** is rarely used in Windows executables but is more commonly employed in Unix and Linux systems. In this linking method, all required code from the library is copied directly into the executable file.
- As a result, the executable becomes self-contained but significantly larger in size.
- Understanding static linking is important in malware analysis, as it affects file size, portability, and the visibility of external dependencies within the executable.

Runtime Linking

- **Dynamic linking at runtime** is uncommon in legitimate software but frequently observed in malware, particularly in packed or obfuscated programs.
- Unlike static linking, dynamically linked functions are connected to the necessary libraries only when they are required during execution, rather than at program startup. This technique is typically implemented using Windows API functions such as **LoadLibrary** and **GetProcAddress**.
- For malware analysts, understanding runtime dynamic linking is essential, as it helps reveal hidden dependencies and behaviors that may not be apparent through static analysis alone.

Dynamic Linking

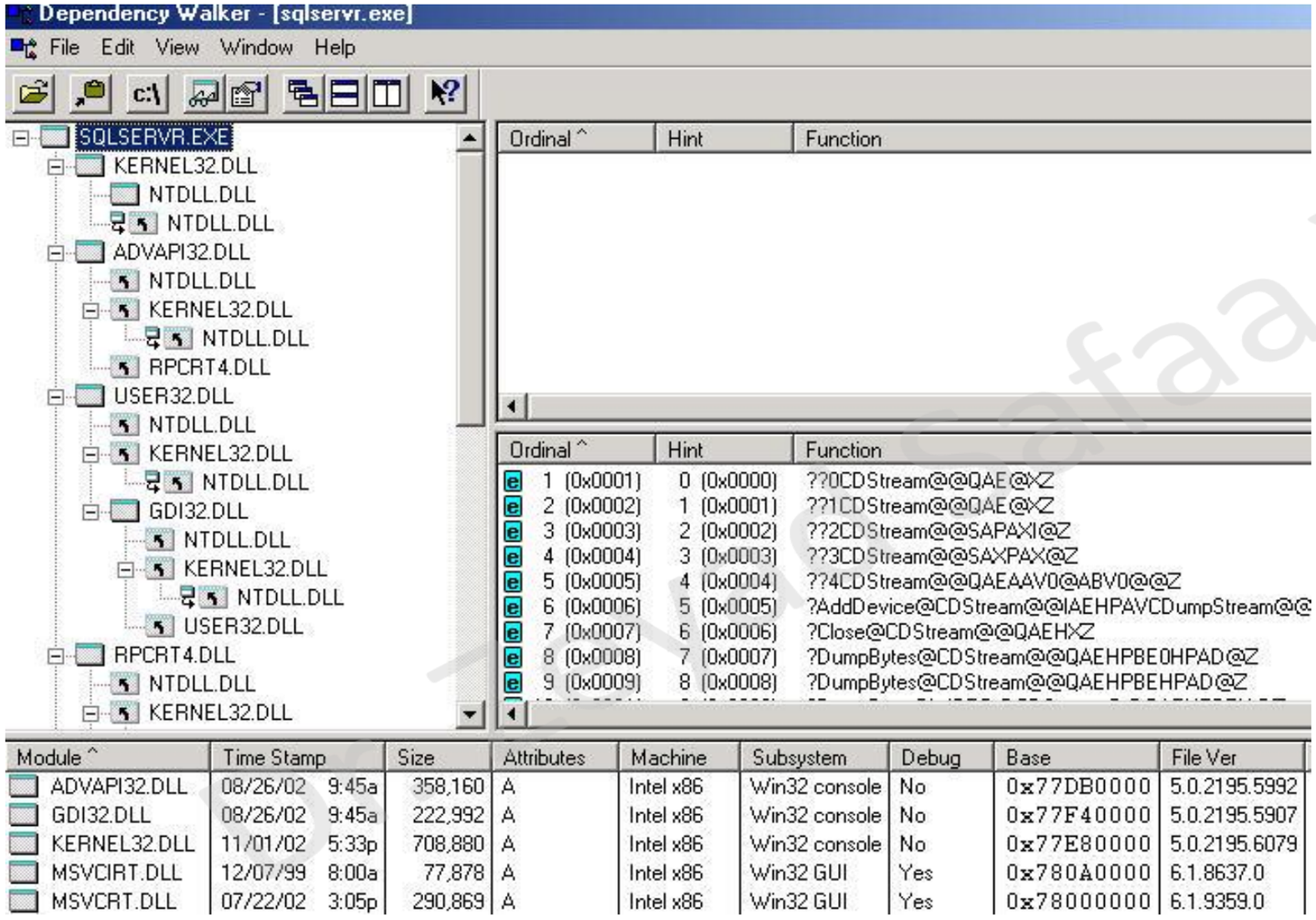
- **Dynamic linking** is the most common method used in Windows executables. In this approach, the host operating system locates and loads the required libraries when the program is executed.
- The **PE header** explicitly lists all libraries and functions that the executable depends on, providing valuable information for analysis.
- For malware analysts, the names of imported libraries and functions can offer important clues about a program's intended behavior. For example, the presence of a function such as **URLDownloadToFile** strongly suggests that the program is designed to download external content, which may indicate malicious activity.



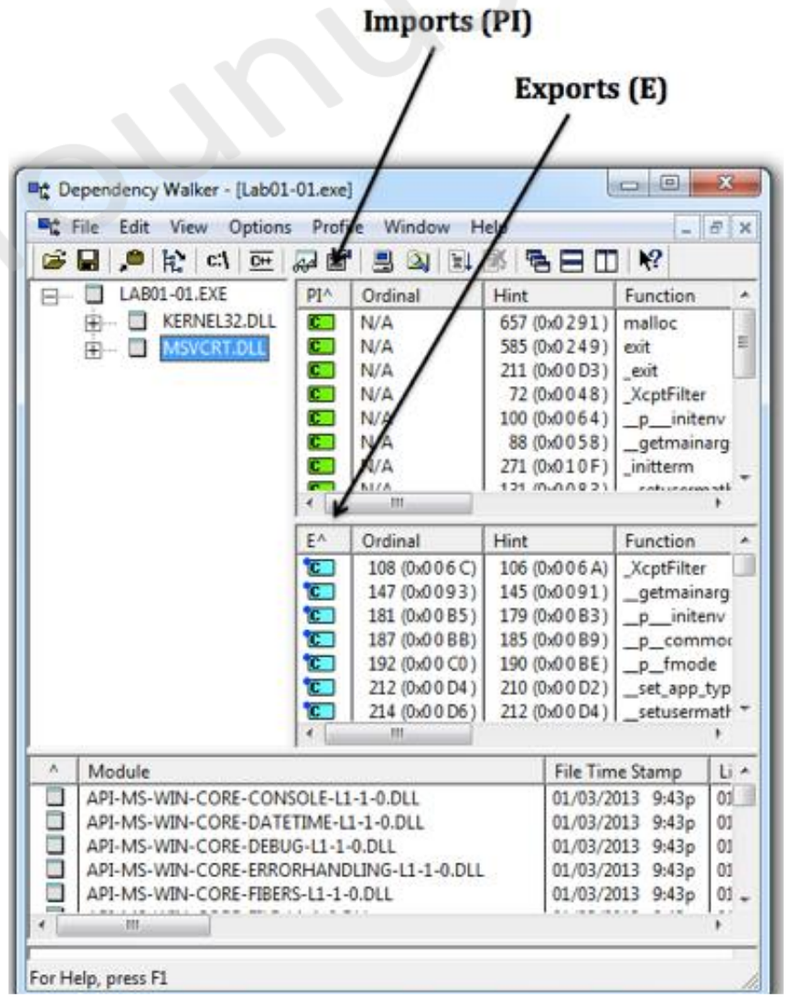
Dependency Walker tool

- **Dependency Walker** is a free tool for Microsoft Windows that allows analysts to examine the imported and exported functions of a portable executable (PE) file. The tool is particularly useful for identifying dynamically linked functions and understanding a program's dependencies.
- In general, legitimate software typically relies on a substantial number of dynamic-link libraries (DLLs) to provide functionality. In contrast, malware often imports very few DLLs, especially if it is packed or obfuscated, as the malicious code is designed to minimize visible dependencies and evade detection.
- By analyzing the DLL usage with Dependency Walker, security professionals can gain insight into the program's structure and potentially identify suspicious or malicious behavior.

Dependency Walker tool



Imports & Exports in Dependency Walker



Services.ex_ (malware)

Services.ex_ is a malware sample whose imported DLLs provide valuable insight into its functionality. By analyzing the libraries and functions that the program imports, analysts can infer its potential behavior, such as network communication, file manipulation, or interaction with system resources. Understanding these imports is therefore a critical step in uncovering the malicious actions of **services.ex_** and guiding further analysis.

- For example:
- `WS2_32.DLL` → network communication
- `KERNEL32.DLL` → system-level operations (processes, files, memory)

Common DLLs

DLL	Description
Kernel32.dll	This is a very common DLL that contains core functionality, such as access and manipulation of memory, files, and hardware.
Advapi32.dll	This DLL provides access to advanced core Windows components such as the Service Manager and Registry.
User32.dll	This DLL contains all the user-interface components, such as buttons, scroll bars, and components for controlling and responding to user actions.
Gdi32.dll	This DLL contains functions for displaying and manipulating graphics.
Ntdll.dll	This DLL is the interface to the Windows kernel. Executables generally do not import this file directly, although it is always imported indirectly by *Kernel32.dll*. If an executable imports this file, it means that the author intended to use functionality not normally available to Windows programs.

Common DLLs

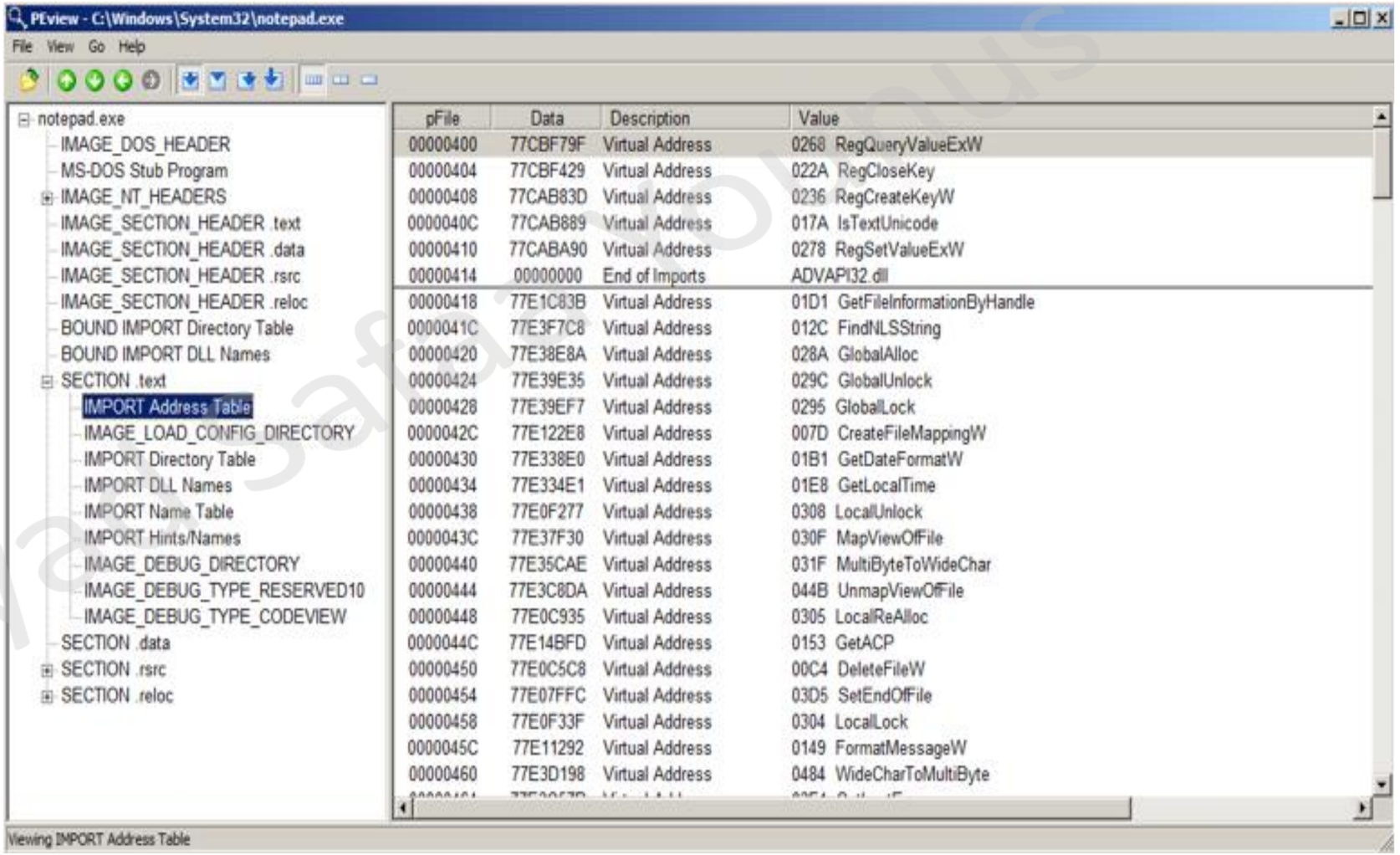
Ntdll.dll	This DLL is the interface to the Windows kernel. Executables generally do not import this file directly, although it is always imported indirectly by *Kernel32.dll*. If an executable imports this file, it means that the author intended to use functionality not normally available to Windows programs. Some tasks, such as hiding functionality or manipulating processes, will use this interface.
WSock32.dll and *Ws2_32.dll*	These are networking DLLs. A program that accesses either of these most likely connects to a network or performs network-related tasks.
Wininet.dll	This DLL contains higher-level networking functions that implement protocols such as FTP, HTTP, and NTP.

Exports

- **DLL and EXE files** can export functions that are used by other programs. While exported functions are commonly found in DLLs, they are relatively rare in executable (EXE) files. Both imported and exported functions are documented in the Portable Executable (PE) header.
- When an EXE contains exported functions, these exports may provide significant clues about the program's intended behavior and functionality.
- For example, an exported function named **ServiceMain** typically indicates that the program or malware is designed to run as a Windows service.
- However, malware authors often attempt to evade analysis by **renaming**, **obfuscating**, or **hiding exported functions**, thereby misleading analysts and complicating the reverse engineering process.

Notepad.exe

- Notepad .exe is a safe and clean Windows program
- useful to learn how PE files work.
- It shows how legitimate applications use DLLs and imported functions, and it can also be used as a practical example to study a real PE file for instance, by opening it in tools like **PEview** to observe how it calls **DLLs** such as **kernel32.dll** or **user32.dll**.

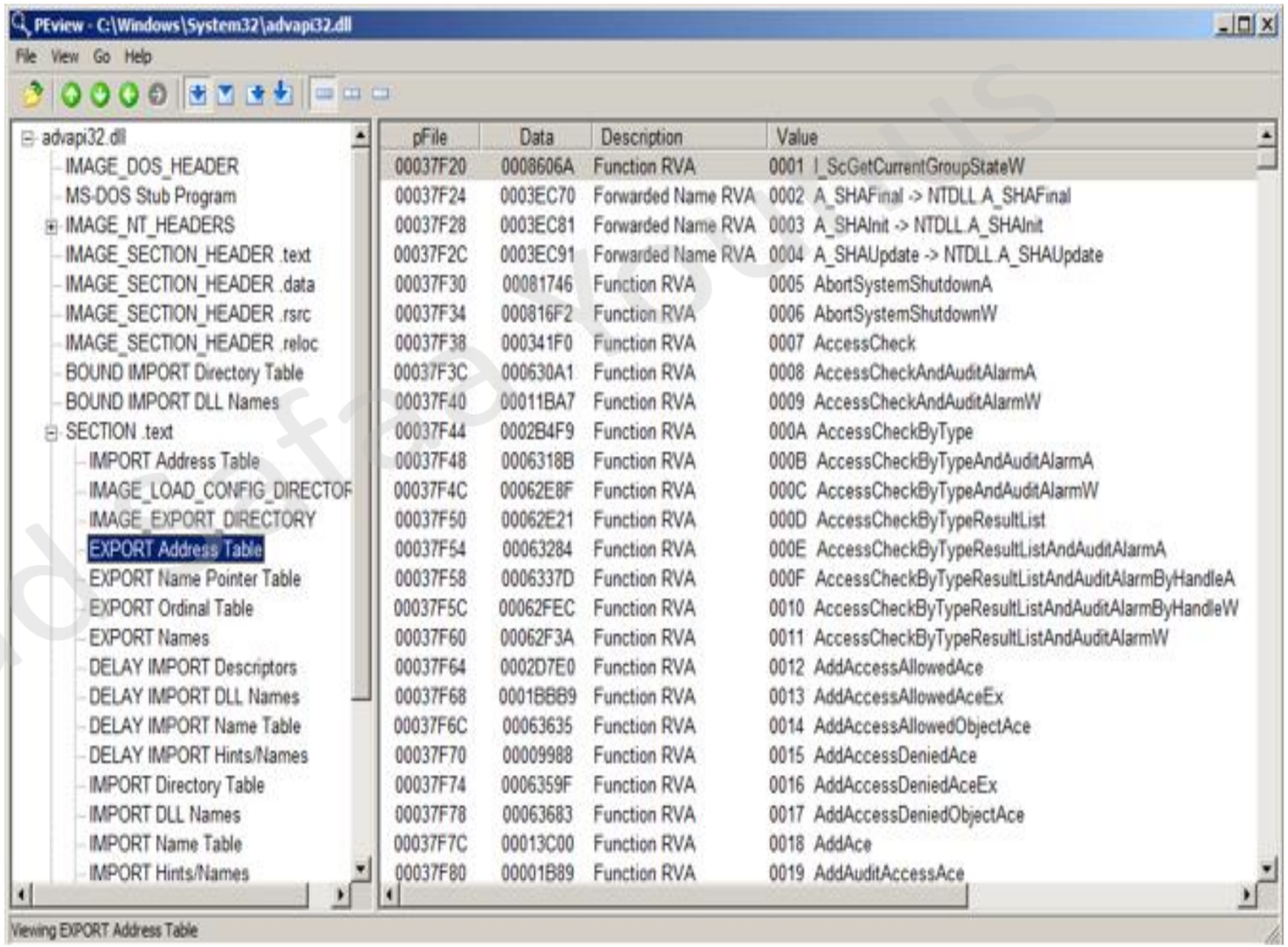


Advapi32.dll

Advapi32.dll is a Windows system library that provides functions for:

1. Security and user permissions
2. Service control
3. Registry access
4. Authentication

Malware often uses it to **elevate privileges** or **change system settings**.



Example of Malware

- A **keylogger** is a type of malicious software (malware) designed to secretly record the keys pressed on a keyboard by the user. Its main goal is to steal sensitive information such as usernames, passwords, credit card numbers, and private messages without the user's knowledge.
- Keyloggers often run in the background and use Windows system libraries to monitor keyboard and mouse activities, then store or send the collected data to the attacker.
- **Keylogger Analysis**
  - A keylogger typically imports **User32.dll** and makes use of several Windows API functions related to input handling.
  - The function **SetWindowsHookEx** is commonly used in spyware and keyloggers, as it allows the program to install a hook procedure that intercepts keyboard and mouse events. This is one of the most popular techniques for capturing keystrokes.
  - The malware may export functions such as **LowLevelKeyboardProc** and **LowLevelMouseProc**, which are used as callback procedures to capture keyboard and mouse input and forward the collected data to another component or remote destination.
  - Additionally, the function **RegisterHotKey** can be used to define a specific key combination (for example, **Ctrl + Shift + P**) that triggers a particular action, such as harvesting, saving, or transmitting the logged keystroke data.

Example of Malware

- A Packed Program : ***A Dead End***
  - A very short import list and no readable strings usually indicate the file is packed/obfuscated.
  - Very few functions, Normal programs (even “Hello World”) import more functions.
  - Because the sample is packed, basic static analysis won’t reveal much.

Kernel32.dll	User32.dll
GetModuleHandleA	MessageBoxA
LoadLibraryA	
GetProcAddress	
ExitProcess	
VirtualAlloc	
VirtualFree	

The PE File Headers and Sections

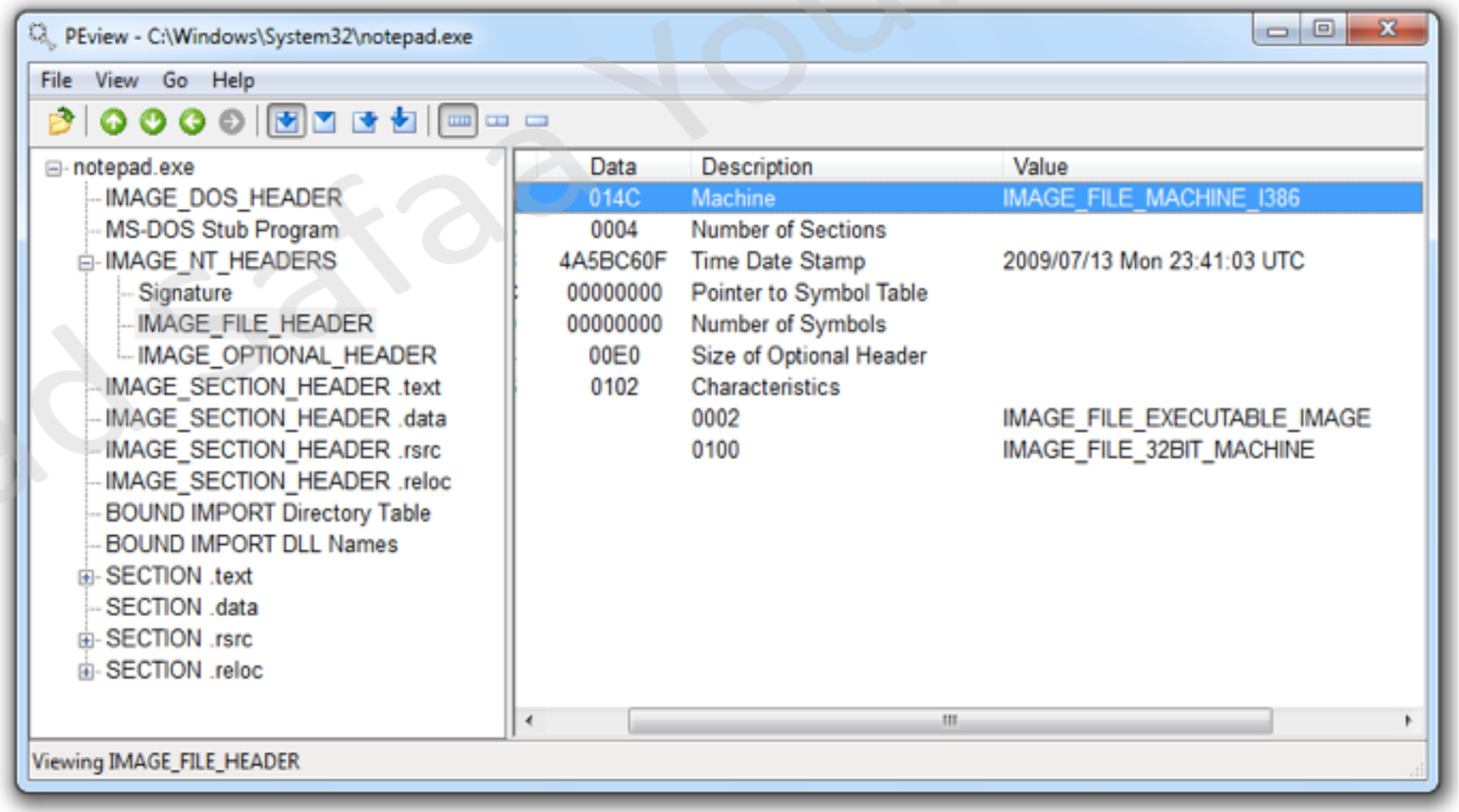
Common PE Sections

- **.text:** Contains the executable code of the program. This is where the main instructions that the CPU executes are stored.
- **.rdata:** Contains read-only data such as **imports and exports information**. Holds data that is globally accessible but cannot be modified during execution.
- **.data:** Stores global and static variables that are accessed and modified throughout the program while it is running.
- **.rsrc:** Stores resources needed by the executable, such as **strings, icons, images, dialogs, and menus**.

Examining PE Files with PEview

- The **PE header** stores key metadata about the executable within its header. **PEview** tool can be used to inspect these headers.

- **Key part of Header:**
- **IMAGE_DOS_HEADER**: Historical, not useful.
- **IMAGE_NT_HEADERS**: Signature (always the same).
- **IMAGE_FILE_HEADER**: Includes compile time (can be forged).
- **IMAGE_OPTIONAL_HEADER** : Defines subsystem (Console or GUI).



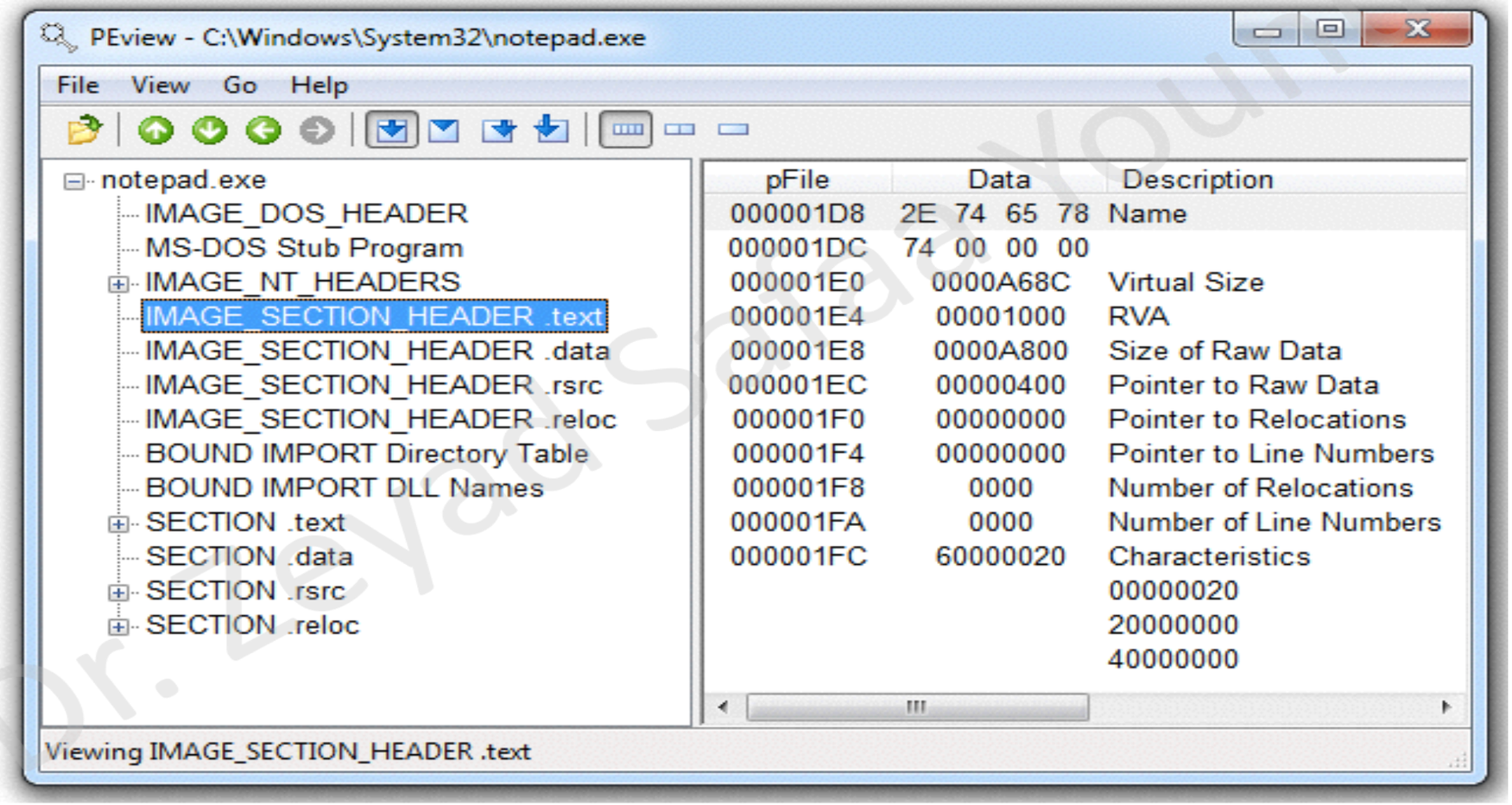
Time Date Stamp

- Shows when this executable was compiled
- Older programs are more likely to be known to antivirus software
- But sometimes the date is wrong
  - All Delphi programs show June 19, 1992
  - Date can also be faked

IMAGE_SECTION_HEADER

- Virtual Size – size in memory (RAM)
- Size of Raw Data – size on disk
  - For **.text** section, normally equal, or nearly equal size.
  - Big difference: may indicate packed code.
  - Unusual section names may be suspiciou
- Packed executables show Virtual Size much larger than Size of Raw Data for **.text** section
- Notes:
  - *Normal program*: similar sizes, standard section names.
  - *Packed malware*: unusual names, Raw Data = 0, large Virtual Size.

.text file Not Packed



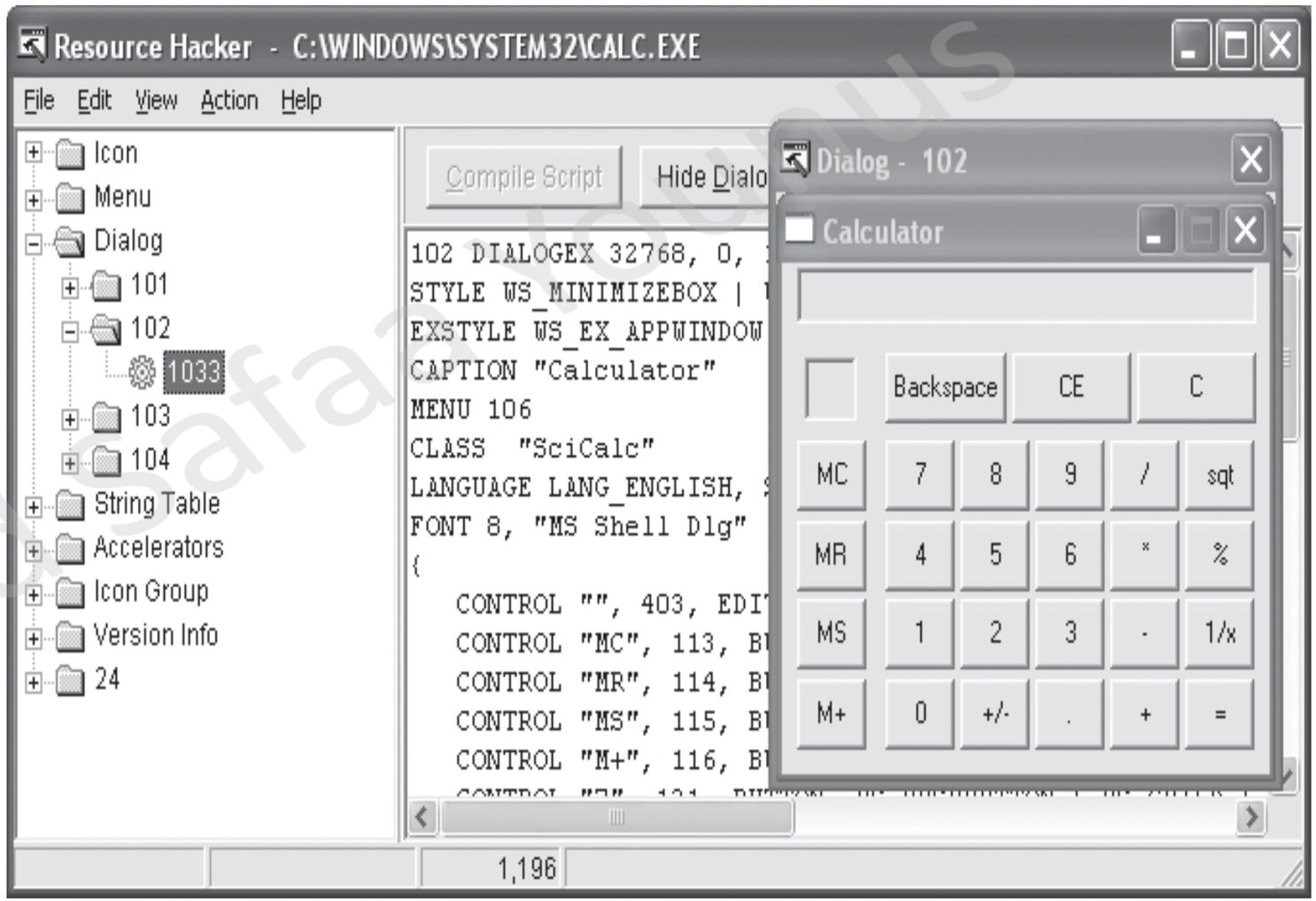
Information for packedprogram.exe

Name	Virtual size	Size of raw data
.text	A000	0000
.data	3000	0000
.rdata	4000	0000
.rsrc	19000	3400
Dijfpds	20000	0000
.sdfuok	34000	3313F
Kijijl	1000	0200

Resource Hacker

- Resource Hacker is a free tool used to view the **.rsrc (resource) section** of PE files.

- This section includes icons, menus, dialog boxes, string tables, and version information.



References

- Sikorski, M., & Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press.
- Bowne, S. (n.d.). *CNIT 126 Ch 0: Malware Analysis Primer & 1: Basic Static Techniques* [PowerPoint slides]. SlideShare. https://www.slideshare.net/slideshow/cnit-126-ch-0-malware-analysis-primer-1-basic-static-techniques/71040577
- Bowne, S. (n.d.). *CNIT 126: Ch 0-1 Basic Static Techniques (Part 1)* [Video]. YouTube. https://www.youtube.com/watch?v=4ZY4LC2XpFM