RIT

جامعة روتشستر الأمريكية للتكنولوجيا في دبي

A Global American University in Dubai

CSEC 202 Reverse Engineering Fundamentals

Basic Static Analysis (1/2)+(2/2)

Eng. Emad Abu Khousa

Sections: 600 | 601 | 602

January 29, 2024 + February 5, 2024

Introduction to Malware Analysis

What is Malware?

NIST (National Institute of Standards and Technology) SP 800-83, titled Guide to Malware Incident Prevention and Handling for Desktops and Laptops:

"**Malware, also known as malicious code, refers to a program that is covertly inserted into another program with the intent to destroy data, run destructive or intrusive programs, or otherwise compromise the confidentiality, integrity, or availability of the victim's data, applications, or operating system**."

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-83r1.pdf

“mal·ware" / malwɛr/

code that is used to perform malicious actions

The term is both:

- a singular noun ("**malware**")
- a plural noun ("**malware**")

What is Malware Analysis?

Reverse Engineering a Specific Piece of Malware to Determine its Origin, Functionality, and Potential Impact

Malware analysis is an essential aspect of cybersecurity practices, including Forensic Analysis and Incident Response (**FAIR**) and penetration testing.

Why Malware Analysis?

- **Determine what exactly happened**: Establish the sequence of events that led to the malware infection.
- **Determine the malicious intent of the malware**: Understand the purpose behind the malware, whether it's to steal data, disrupt operations, etc.
- **Identify indicators of compromise (IoCs)**: Find evidence of the infection, like suspicious network traffic or files.
- **Determine the complexity level of an intruder**: Assess the sophistication of the attacker based on the complexity of the malware.

Why Malware Analysis?

- **Identify the exploited vulnerability**: Discover the security weaknesses that were leveraged to gain unauthorized access.
- **Identify the extent of damage caused by the intrusion**: Quantify the impact, including data loss, financial cost, or reputational damage
- **Evaluate the harm from an intrusion**: Assess the overall damage and potential future risks posed by the malware.
- **Distinguish the gatecrasher or insider responsible for the malware entry**: Identify whether the attack was conducted by an external entity or an insider with authorized access.

Why Malware Analysis?

- **Catch the perpetrator responsible for installing the malware**: Work towards identifying and apprehending the source of the malware.

- **Find signatures for host and network-based intrusion detection systems**: Develop patterns or signatures that can be used to detect and prevent future infections.
  - **Host-based signatures (or indicators)**
    - What a malware does to a system
    - Based on behaviors, not on the characteristics of the malware itself
    - Files created or modified, changes to registry
  - **Network-based signatures**
    - Monitoring network traffic
    - Can be created without malware static analysis
    - But with malware static analysis, it is more effective

General Guidelines for Malware Analysis

- During software analysis, pay attention to the essential features instead of understanding every detail.
- Try different tools and approaches to analyze the malware, as a single approach may not be useful. If you're not having luck with one tool, try another.
- Malware analysis is like a **cat-and-mouse** game. To succeed as a malware analyst, you must be able to recognize, understand, and defeat these techniques, and respond to changes in the art of malware analysis.

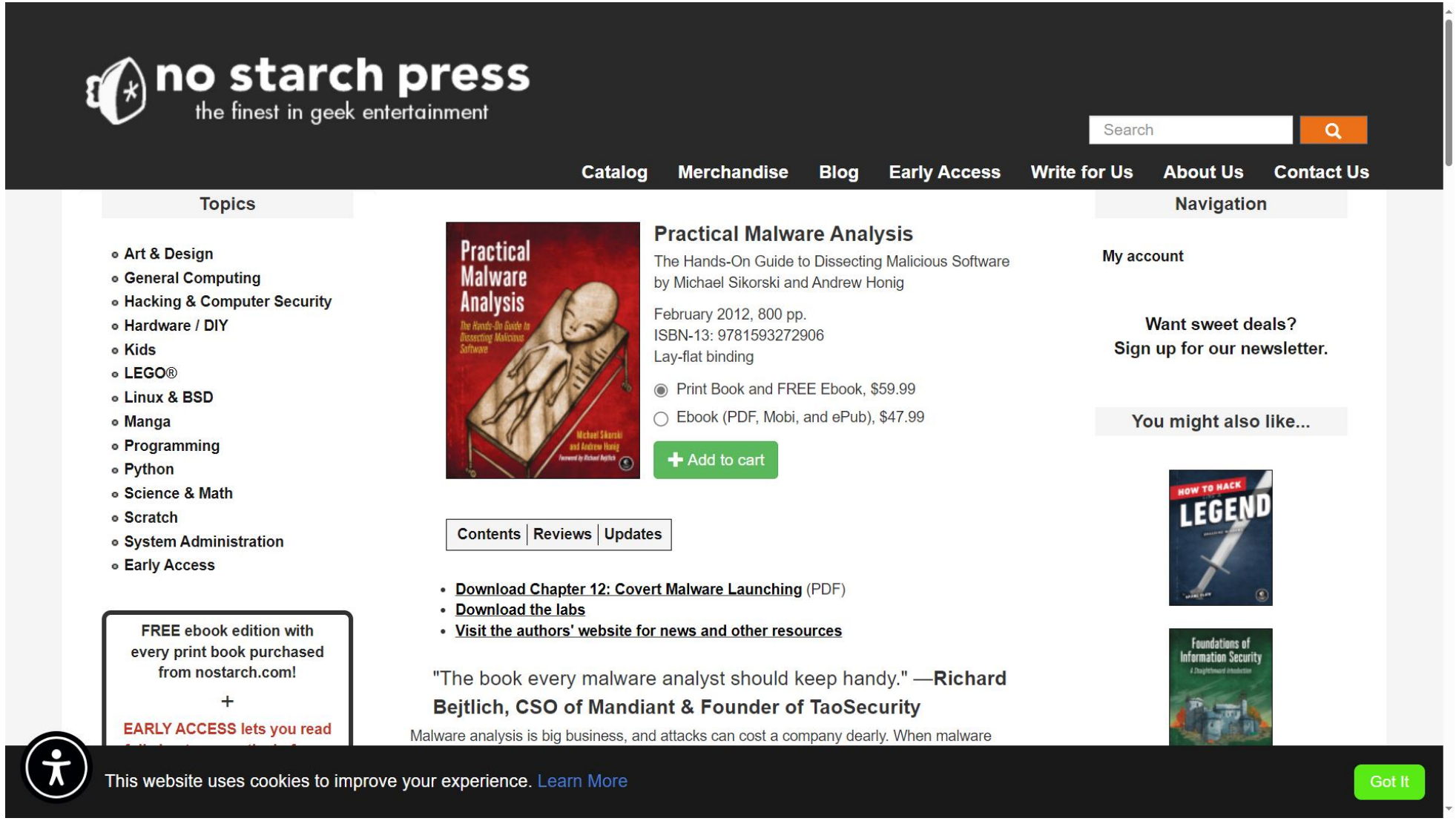
Why Malware Analysis?

Main Objective

Figure out how malware works

”It is Us vs. the Cat”

Troops of 4-


no starch press
the finest in geek entertainment
Search
Catalog
Merchandise
Blog
Early Access
Write for Us
About Us
Contact Us
Topics
Art & Design
General Computing
Hacking & Computer Security
Hardware / DIY
Kids
LEGO®
Linux & BSD
Manga
Programming
Python
Science & Math
Scratch
System Administration
Early Access
FREE ebook edition with every print book purchased from nostarch.com!
+
EARLY ACCESS lets you read
Practical Malware Analysis
Practical Malware Analysis
The Hands-On Guide to Dissecting Malicious Software
by Michael Sikorski and Andrew Honig
February 2012, 800 pp.
ISBN-13: 9781593272906
Lay-flat binding
Print Book and FREE Ebook, $59.99
Ebook (PDF, Mobi, and ePub), $47.99
Add to cart
Contents
Reviews
Updates
Download Chapter 12: Covert Malware Launching (PDF)
Download the labs
Visit the authors' website for news and other resources
"The book every malware analyst should keep handy." —Richard Bejtlich, CSO of Mandiant & Founder of TaoSecurity
Malware analysis is big business, and attacks can cost a company dearly. When malware
Navigation
My account
Want sweet deals?
Sign up for our newsletter.
You might also like...
HOW TO HACK
LEGEND
Foundations of Information Security
This website uses cookies to improve your experience. Learn More
Got It

Malware Analysis Procedure

Malware Analysis Procedure

It is extremely dangerous to analyze malware on production devices connected to production networks. Therefore, one should always analyze malware samples in a testing environment on an isolated network, often referred to as a "Testbed".

Malware analysis involves the following steps:

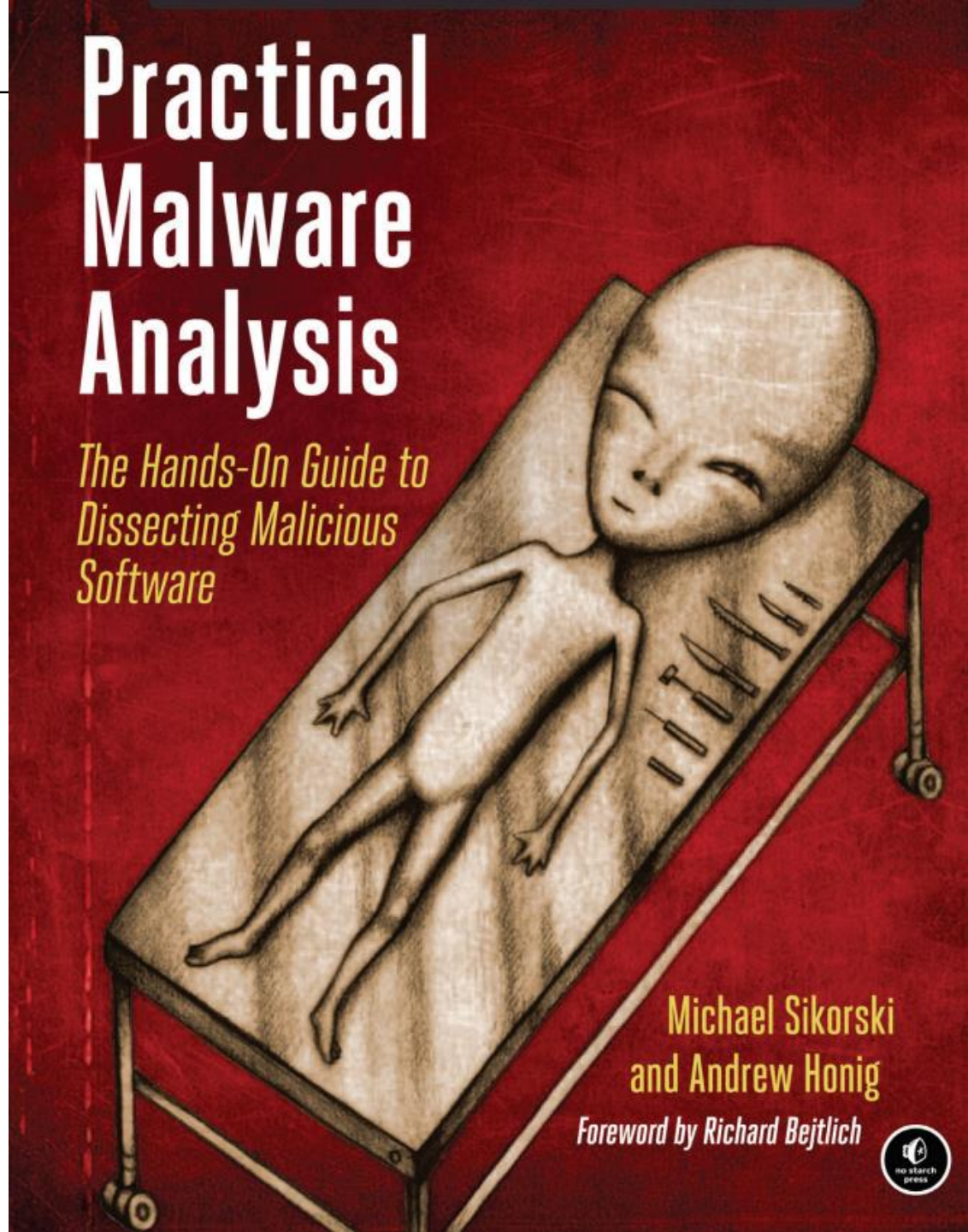
1. **Preparing Testbed:** Setting up a secure environment to examine the malware without risking the main network.

2. **Static Analysis:** Looking at the malware's code without actually running the program to get initial insights.

3. **Dynamic Analysis:** Running the malware in a controlled setting to observe its behavior and how it interacts with other systems.

Malware Analysis Procedure

1. **Preparing Testbed:** Setting up a secure environment to examine the malware without risking the main network.



Be Ready: Preparing Testbed

- **1. Allocate a physical system for the analysis lab:** This involves setting aside a dedicated computer for analyzing malware, ensuring that any potential harm does not affect other systems.

- **2. Install a Virtual Machine (VMware, Hyper-V, etc.) on the system:** A virtual machine is used to create an isolated environment where the malware can be run without affecting the host system.

- **3. Install guest OS in the Virtual machine(s):** The guest operating system is installed on the virtual machine; it's the environment where the malware will be executed and analyzed.

Be Ready: Preparing Testbed

- **4. Isolate the system from the network by ensuring that the NIC card is in "host only" mode:** This step prevents the malware from communicating over the network by setting the network interface card to only communicate with the host system.
- **5. Simulate internet services using tools such as INetSim:** Since the system is isolated from the network, simulation tools like INetSim can provide fake network services (like HTTP or DNS) to the malware, which may require these to function properly.
- **6. Disable the "shared folders" and "guest isolation":** These features are typically disabled to prevent the malware from accessing the host system or other virtual machines, further containing any potential damage.

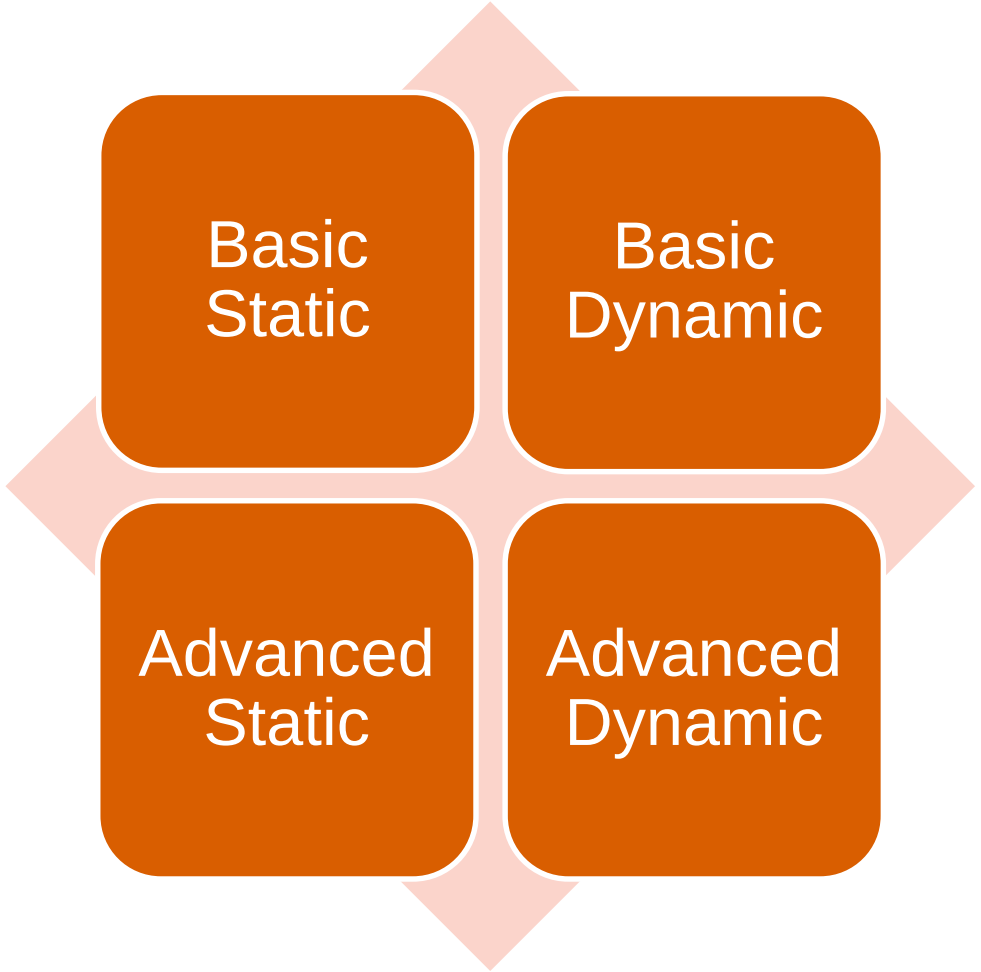
Be Ready: Preparing Testbed

- **7. Install malware analysis tools:** Various tools are installed that are used to analyze the behavior and characteristics of the malware.

- **8. Generate the hash value of each OS and tool:** This is likely a step for integrity checking; by generating hash values of the operating systems and tools, analysts can ensure that they have not been tampered with.

- **9. Copy the malware over to the guest OS:** Finally, the actual malware samples are transferred to the guest operating system for analysis.

Malware Analysis Techniques



You'll need to use a variety of tools (and tricks) in order to see the full picture.

Malware Analysis Techniques

- **Basic Static Analysis**
  - Analyze characteristics of the file itself without analyzing behaviors.
  - (*We don't run the file, we don't see the code*)

- **Advanced Static Analysis**
  - Reverse-engineering the malware's internals by loading the executable into a disassembler, and looking at the program instructions in order to discover what the program does.
  - (*We don't run the file, we see the code, we reverse engineer the code*)

Malware Analysis Techniques

- **Basic Dynamic Analysis**
  - Running the malware and observing its behavior on the system in order to remove the infection, produce effective signatures, and identify malware family and/or threat actor

- **Advanced Dynamic Analysis**
  - Using a debugger to examine the internal state of a running malicious executable.

Basic Static Analysis (1/2)

WARNING: Attention Tigers

As you engage in malware analysis, it's crucial to handle all specimens with extreme caution. You will be examining a malware known as "**brbbot.exe**" which is recognized for its dangerous capabilities.

This malware has the potential to cause significant harm to systems and networks.

Basic Static Analysis

1. Identifying File Type Using Manual Method
2. Local Antivirus Scanning: A Useful First Step
3. File Fingerprinting
4. Online Malware Scanning
5. Strings: Perform strings search
6. Identifying Packing/ Obfuscation Methods
7. Finding the Portable Executable (PE) information
8. Identifying File Dependencies

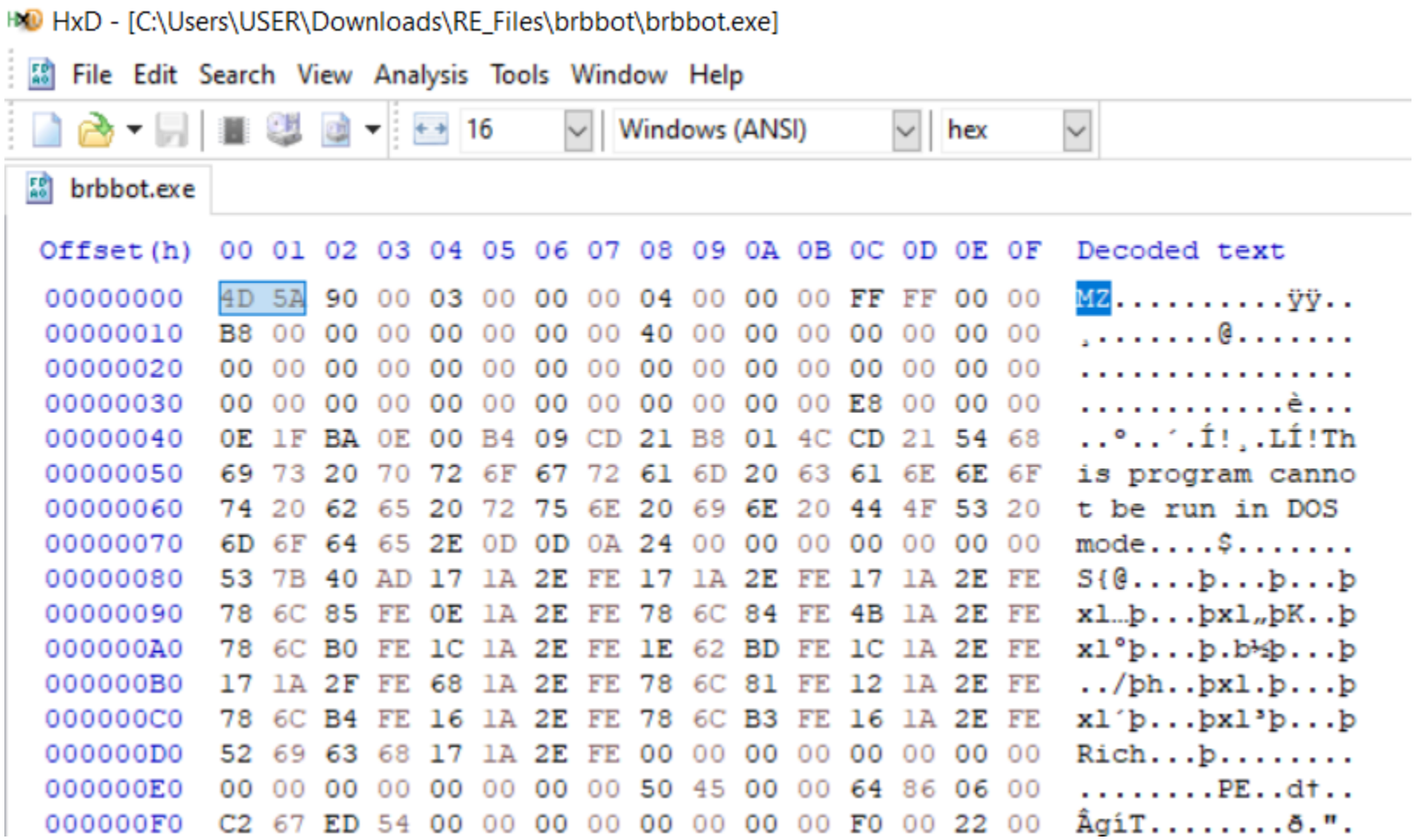
Identifying File Type Using Manual Method

Identifying the file type using a manual method involves examining the raw data of the file using a hex editor, which is a program that displays and allows editing of the binary data of a file in hexadecimal format. Here's how it works:

- **On Windows: Using a Hex Editor:** You open the file in a hex editor. This tool shows each byte of the file as a two-digit hexadecimal number. Hex editors often come with features that aid in file analysis. *HxD hex editor* **https://mh-nexus.de/en/hxd/**

- **Hex Dump on Linux:** On Linux systems, you can generate a hex dump of a file using commands like **xxd**.

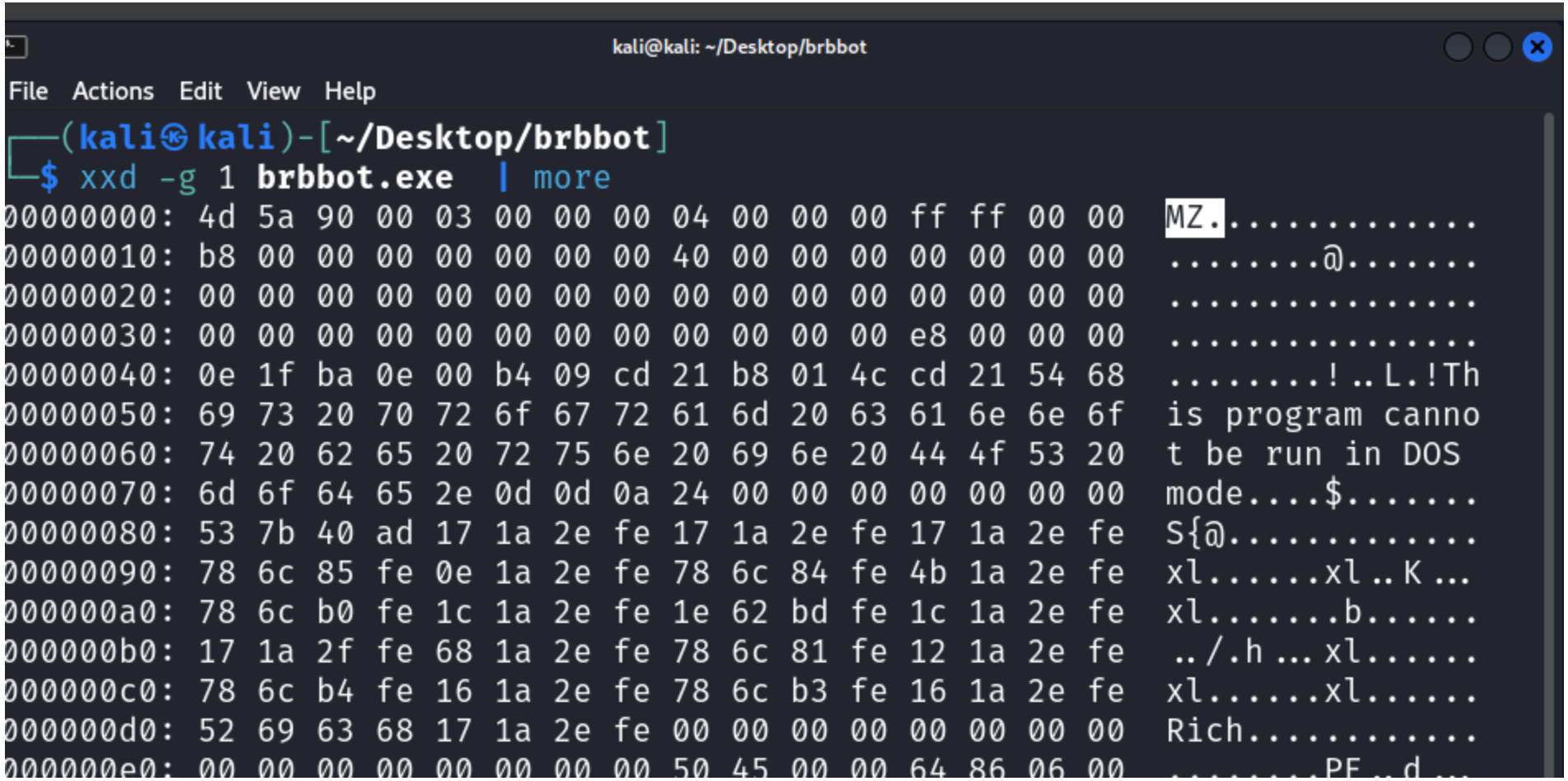
Identifying File Type Using Manual Method



https://mh-nexus.de/en/hxd/

The output **4d 5a** at the start of the hex dump indicates that the file is likely a Windows executable because "4D 5A" is the hexadecimal representation of "MZ", which is the signature for Windows executables (also known as PE files).

Identifying File Type Using Manual Method



$ xxd -g 1 brbbot.exe | more

Local Antivirus Scanning: A Useful First Step

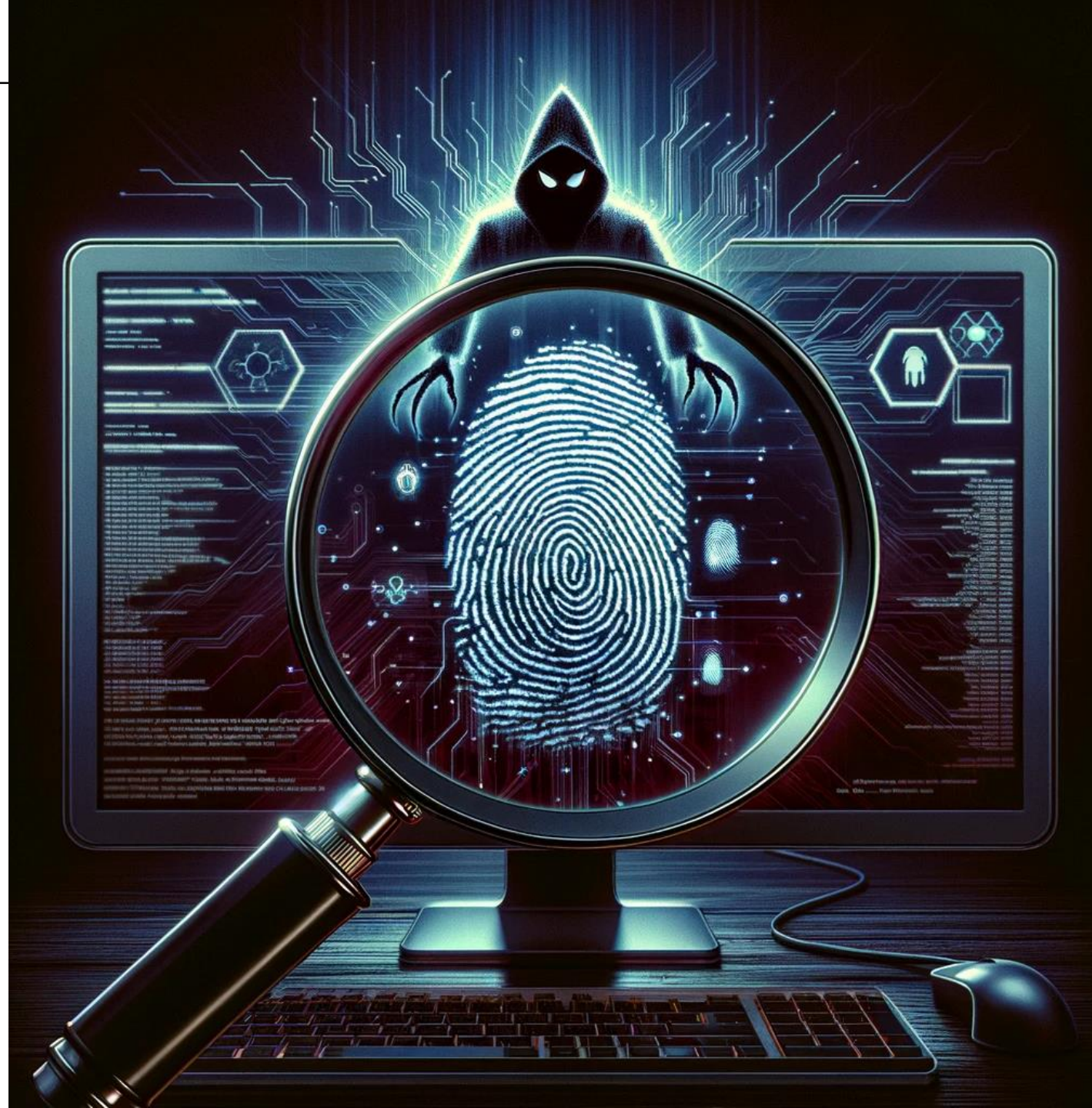
When first analyzing prospective malware, a good first step is to run it through multiple antivirus programs, which may already have identified it.

- **But antivirus tools are certainly not perfect:**
  - Depend on databases of known malware signatures.
  - Utilize heuristics for pattern and behavior analysis.
- **Challenges of Antivirus Detection:**
  - Malware authors can modify signatures to avoid detection.
  - Uncommon malware may not be present in databases.
  - Heuristics can miss new, uniquely coded malware.

File Fingerprinting

is described as the process of computing **hash** values for a given binary code.

A hash value is a numerical value produced by a **hash function**, which is designed to turn the file's data into a typically shorter, fixed-length sequence that represents the original string of data uniquely.



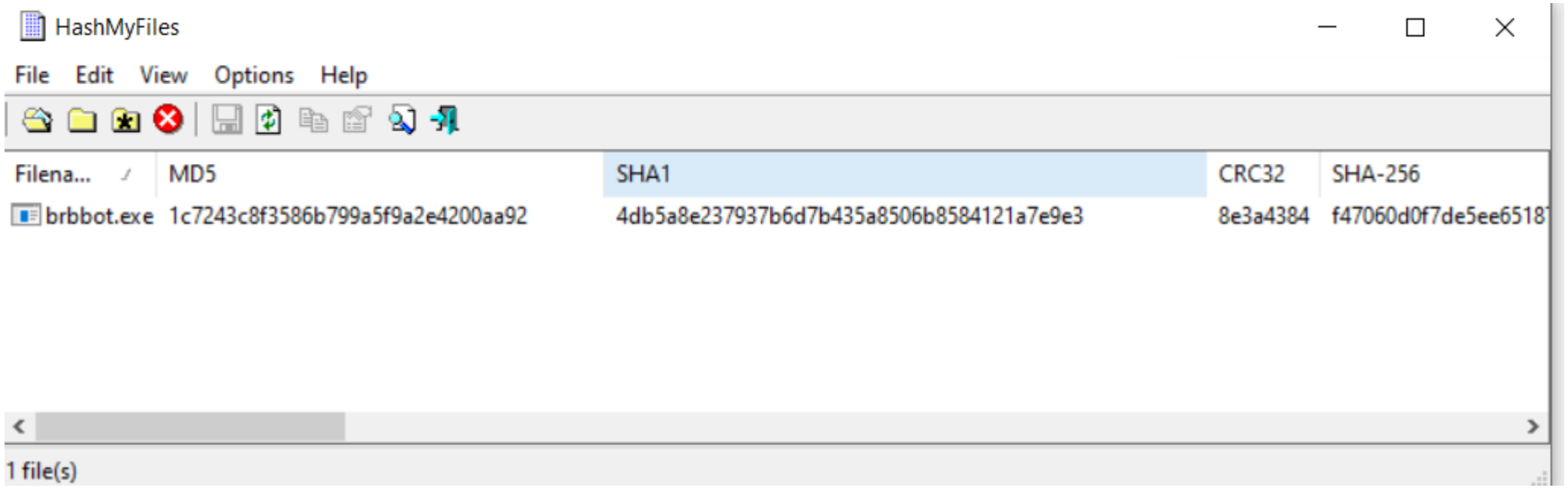
File Fingerprinting

- A hash value is a numerical value produced by a **hash function**, which is designed to turn the file's data into a typically shorter, fixed-length sequence that represents the original string of data uniquely.

- These hash values are used to **uniquely identify** malware or to periodically verify if any changes have been made to the binary code during analysis. This ensures that the file being analyzed has not been altered or tampered with since the hash value was computed.

File Fingerprinting

HashMyFiles can be used to calculate various hash values of a malware file. HashMyFiles is an application that can compute multiple types of hash values, such as MD5, SHA1, CRC32, SHA-256, SHA-512, and SHA-384.
https://github.com/foreni-packages/hashmyfiles



File Fingerprinting

```
┌──(kali㉿kali)-[~/Desktop/brbbot]
└─$ md5sum brbbot.exe
1c7243c8f3586b799a5f9a2e4200aa92  brbbot.exe

┌──(kali㉿kali)-[~/Desktop/brbbot]
└─$ sha1sum brbbot.exe
4db5a8e237937b6d7b435a8506b8584121a7e9e3  brbbot.exe

┌──(kali㉿kali)-[~/Desktop/brbbot]
└─$
```

md5sum filename | sha1sum filename | sha256sum filename

File Fingerprinting

Results	
Original text	*(binary only)*
Original bytes	4d5a900003000000040000000ffff0000b80000000000000040... (length=75776)
Adler32	4ce4220b
CRC32	8e3a4384
Haval	2d2b4199a633117678bad3b9a20acfe7
MD2	a87234a7512bcb5d94daf73e1a7afdd1
MD4	e057a33746c0f27888d48a07fbdd7d06
MD5	1c7243c8f3586b799a5f9a2e4200aa92
RipeMD128	e0cabb01b2cbfe1c2200e8acbf889904
RipeMD160	e47ac11acfb51492eb57d14e547ff05bae731fc0
SHA-1	4db5a8e237937b6d7b435a8506b8584121a7e9e3
SHA-256	f47060d0f7de5ee651878eb18dd2d24b5003bdb03ef4f49879f448f05034a21e
SHA-384	0e4902966a1761dae937b874d05fd20237c9840d54df269f5c5edf4798aed4e03aadf42f386e39acaa058ad63390d432
SHA-512	56cdf52cfcc102d2c8cc90e5a298eeaefd44002061108b0d6b330bb93c3590b3f8b2c3c4e1fa208fafeefa7dafa092bd0f57d3cf905382f88b9f66a5d84357fd
Tiger	70ac61b163b588b8e351df6c5bae7298559bbd9781691653

https://www.fileformat.info/tool/hash.htm

File Fingerprinting

Other file fingerprinting tools available for such analysis, including:

- **Mimikatz** (a tool that can be used for various purposes, including extracting plaintext passwords from memory)
- **HashCalc** (a utility that supports many hash calculations)
- **hashdeep** (a suite of tools to compute digests, checksums, and message digests)
- **tools4noobs** - Online hash calculator (an online service for hash calculations)

File Fingerprinting

Once you have a unique hash for a piece of malware, you can:

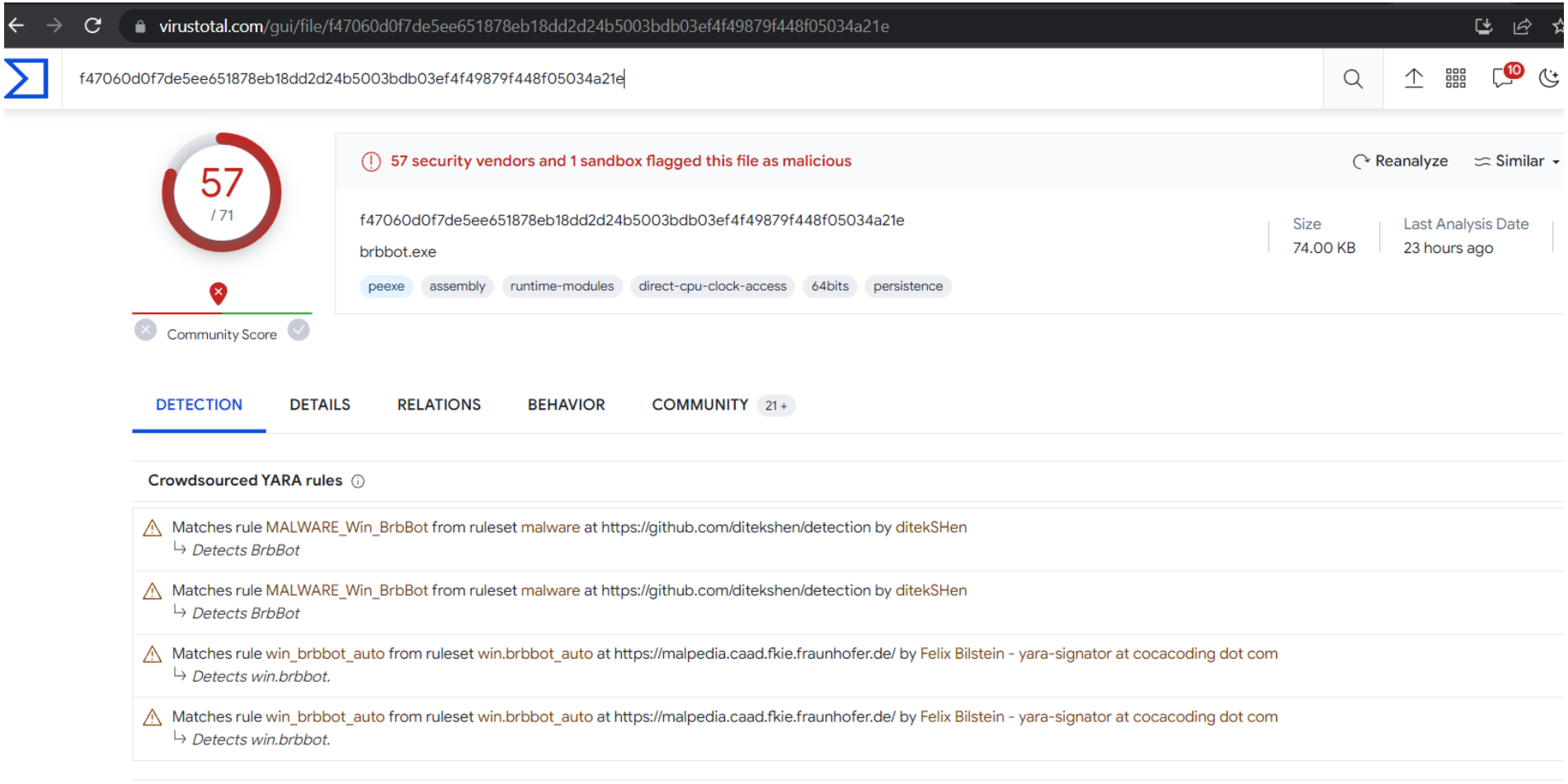
- Use the hash as a label
- Share that hash with other analysts to help them to identify malware
- Search for that hash online to see if the file has already been identified

Online Malware Scanning

Upload the suspicious file (or its hash) to **online services** like **VirusTotal**, which will scan the file with a variety of different antivirus engines.

VirusTotal is a popular choice for this because it aggregates multiple antivirus engines and can provide a comprehensive scan report, showing which engines detected the file as malicious and what type of malware it is identified as.

Online Malware Scanning



https://www.virustotal.com/

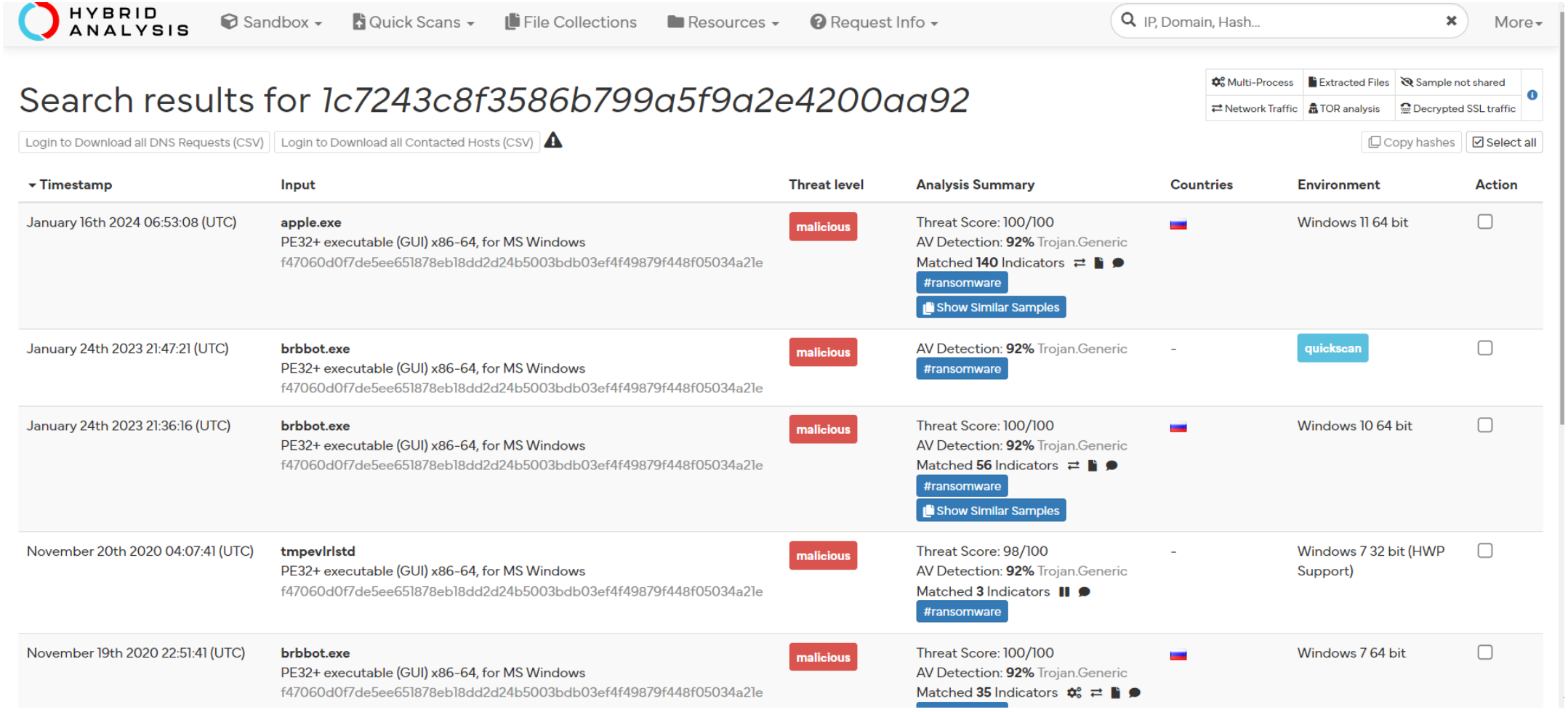
Online Malware Scanning

List of Tools:

- Hybrid Analysis
- Cuckoo Sandbox
- Jotti's Malware Scan
- Valkyrie Comodo
- FortiGuard Online Scanner

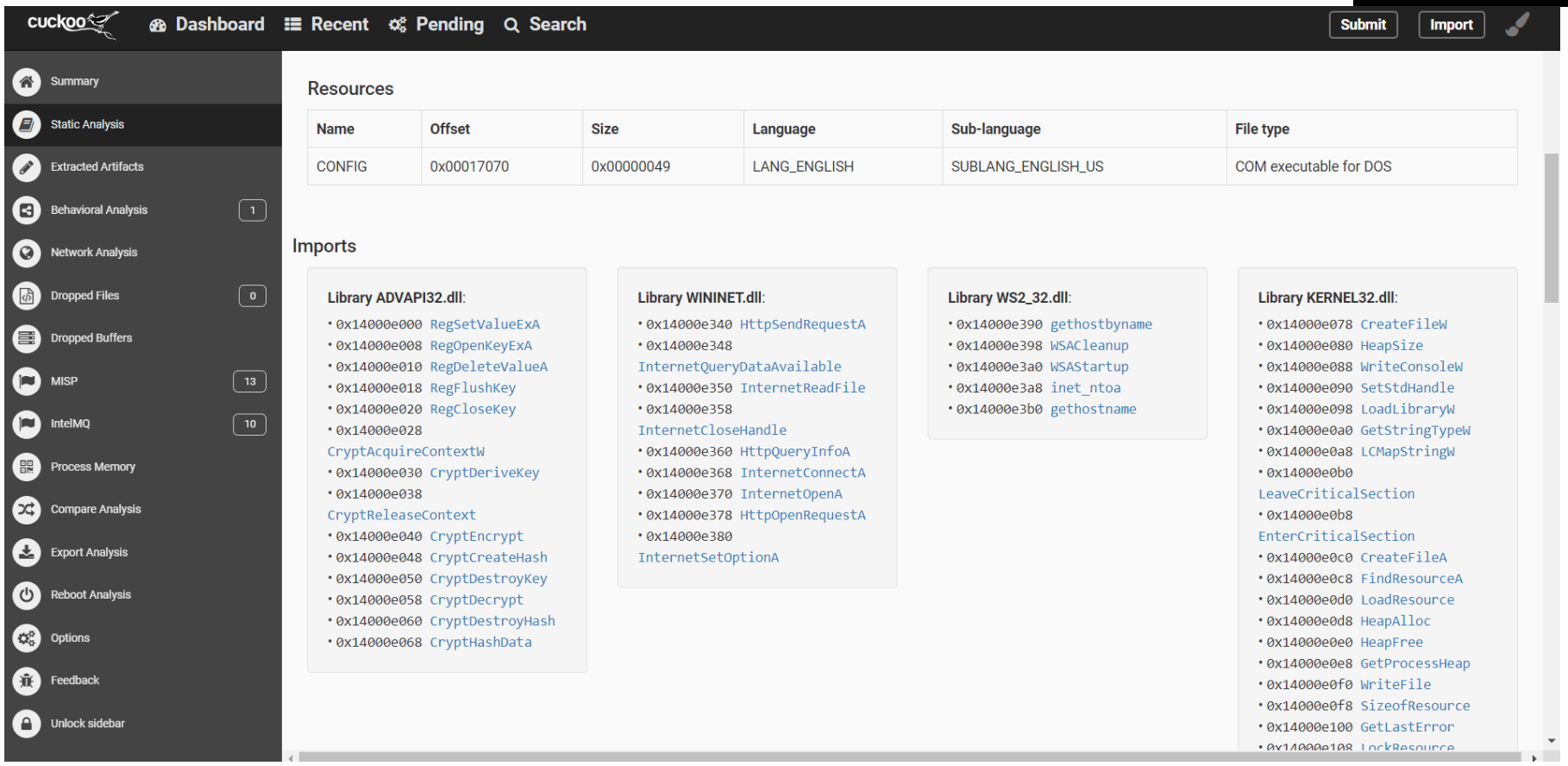
These tools provide a range of services from running the suspicious files in a contained sandbox environment to scan it

Online Malware Scanning



https://www.hybrid-analysis.com/

Online Malware Scanning



https://cuckoo.cert.ee/

Basic Static Analysis (2/2)

Basic Static Analysis

1. Identifying File Type Using Manual Method
2. Local Antivirus Scanning: A Useful First Step
3. File Fingerprinting
4. Online Malware Scanning
5. Strings: Perform strings search
6. Identifying Packing/ Obfuscation Methods
7. Finding the Portable Executable (PE) information
8. Identifying File Dependencies

Strings: Perform strings search

Purpose of String Analysis: Uncover program behavior and potential malicious intent by examining embedded strings within executable files.

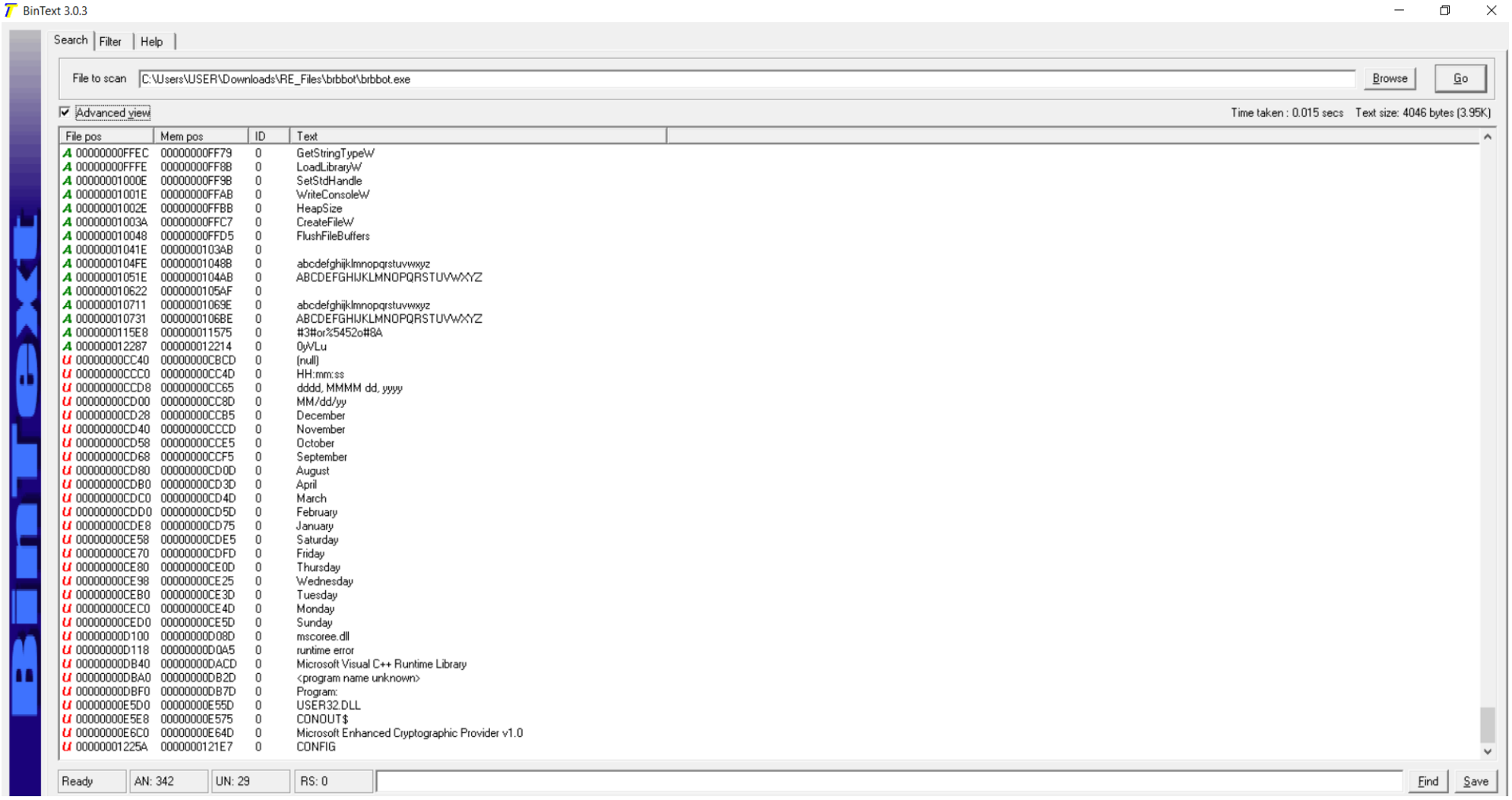
Key Actions in String Analysis:

- Search and analyze strings to reveal program functionality.
- Identify harmful actions, such as external communications through URL strings.
- Remain alert for hidden or encrypted strings that could indicate suspicious activity.

Recommended Tools:

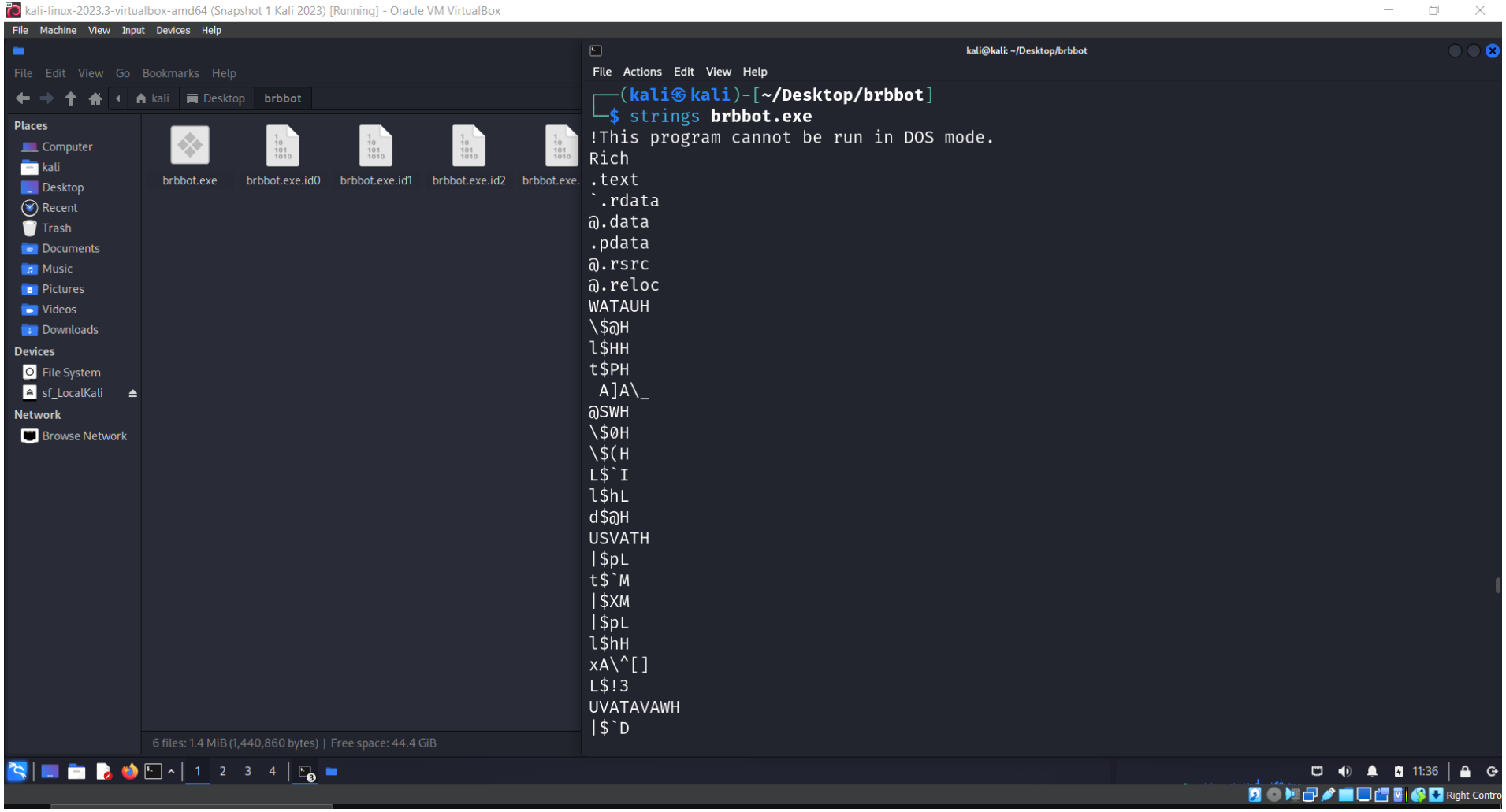
- **BinText:** Extract ASCII and Unicode strings for analysis.
- **Other Tools:** FLOSS, Strings (Microsoft), Free EXE DLL Resource Extract, FileSeek, Hex Workshop.

Strings: Perform strings search



https://www.majorgeeks.com/files/details/bintext.html

Strings: Perform strings search



$ strings brbbot.exe

Strings: notes

- **Standard Program Signature:** The "!This program cannot be run in DOS mode." is a standard message found at the beginning of Windows executables (PE files), indicating it's not a DOS program.
- **Section Names:** Strings like .text, .rdata, .data, .pdata, .rsrc, and .reloc refer to different sections of the executable file, indicating the layout of the binary.
- **Garbled or Encoded Data:** Strings with nonsensical sequences (like WATAUH, A]A_, UVATAVAWH) could be encrypted or encoded data, possibly used by the program to conceal its operations or to store data in a non-human-readable format.
- **Error and Status Messages:** Strings such as "runtime error" and "Microsoft Visual C++ Runtime Library" are typical of error messages that might be displayed to a user.
- **Functionality Indicators:** References to API functions like GetProcessWindowStation, MessageBoxW, RegSetValueExA, and many others suggest what the program might be capable of doing, such as interacting with the Windows registry, displaying messages, or managing encryption.

Strings: notes

- **Potential Malware Indicators:** Strings like **sleep**, **encode**, and specific patterns of API calls could **suggest potentially malicious activity**, like waiting for a certain time before executing or **encoding data to exfiltrate**.

- **Embedded Resource Information**: References to file paths (APPDATA), temporary files (**brbconfig.tmp**), and internet-related strings (Mozilla/4.0, HTTP/1.1) can indicate network communication capabilities or paths the malware uses to store data.

- **Timestamps and Language Data:** Strings that represent dates, times (HH:mm:ss), and months (December, November, etc.) are typically used for formatting date and time data within the program.

- **Malware or Botnet Identification:** The string **brbbot** might be the name of a malware or botnet that the executable is associated with.

Strings: notes

- **Executable's Runtime Environment:** References to DLLs like ntdll.dll, ADVAPI32.dll, WININET.dll, WS2_32.dll, KERNEL32.dll, USER32.dll, and mscoree.dll show the dependencies of the executable and give clues about its interaction with the Windows operating system.

- **Debug and Error Handling:** Strings related to error handling and debugging (UnhandledExceptionFilter, **IsDebuggerPresent**) suggest the program has mechanisms to manage errors or to detect if it's being analyzed, which is common in sophisticated malware.

Strings: Suspicious activities! Or NOT?

From the strings provided, here are several activities that could be considered suspicious or indicative of potential malware:

- **1. Encoded or Obfuscated Strings:** The presence of garbled text such as "A]A_", "UVWATAUAVAWH", and other non-readable strings could suggest attempts to hide the true functionality of the program, which is a common tactic used by malware to avoid detection.

- **2. Registry Manipulation Functions:** API calls like "RegSetValueExA", "RegOpenKeyExA", "RegDeleteValueA", "RegFlushKey", and "RegCloseKey" suggest the program has the capability to modify the Windows registry, which malware often does to **establish persistence** or to change system configurations.

- **3. Cryptographic Functions:** Strings related to cryptographic operations such as "CryptAcquireContextW", "CryptDeriveKey", "CryptEncrypt", "CryptCreateHash", "CryptDecrypt", and references to cryptographic providers could indicate the program is designed to encrypt data, which could be used for legitimate purposes but also for **ransomware activities.**

- **4. Network Communication Indicators**: Use of the WinINet API ("InternetQueryDataAvailable", "InternetReadFile", "InternetCloseHandle", "HttpQueryInfoA", "InternetConnectA", "InternetSetOptionA", "HttpOpenRequestA", "HttpSendRequestA", "InternetOpenA") points to the capability for network communication, which could be used for data exfiltration or command and control communication (**C2**).

Strings: Suspicious activities! Or NOT?

- **5. Botnet or Malware Name:** The string "brbbot" may be the name of a botnet or a specific piece of malware.
- **6. Sleep Function:** The "sleep" function could be used by malware to avoid dynamic analysis by making the malware sleep during analysis, thus not showing any malicious activity.

- **7. System Information Gathering:** The presence of "ZwQuerySystemInformation" from "ntdll.dll" could indicate that the program collects system information, which could be used to tailor malicious payloads or to fingerprint the system for targeted attacks.

- **8. Potential Keylogging Indicators:** API calls such as "GetLastActivePopup", "GetActiveWindow", and "GetProcessWindowStation" can sometimes be used in the context of keylogging or screen capturing functionality.
- **9. File and Directory Access:** The use of "CreateFileA", "FindResourceA", "WriteFile", and other file operation APIs could suggest that the program is accessing files, possibly for malicious purposes such as modifying system files or creating files to execute payloads.

Strings: Suspicious activities! Or NOT?

- **10. Error Handling to Detect Analysis:** Functions like "IsDebuggerPresent" and "UnhandledExceptionFilter" are often checked by malware to determine if it is being analyzed or debugged, which can trigger the malware to alter its behavior to hinder analysis.

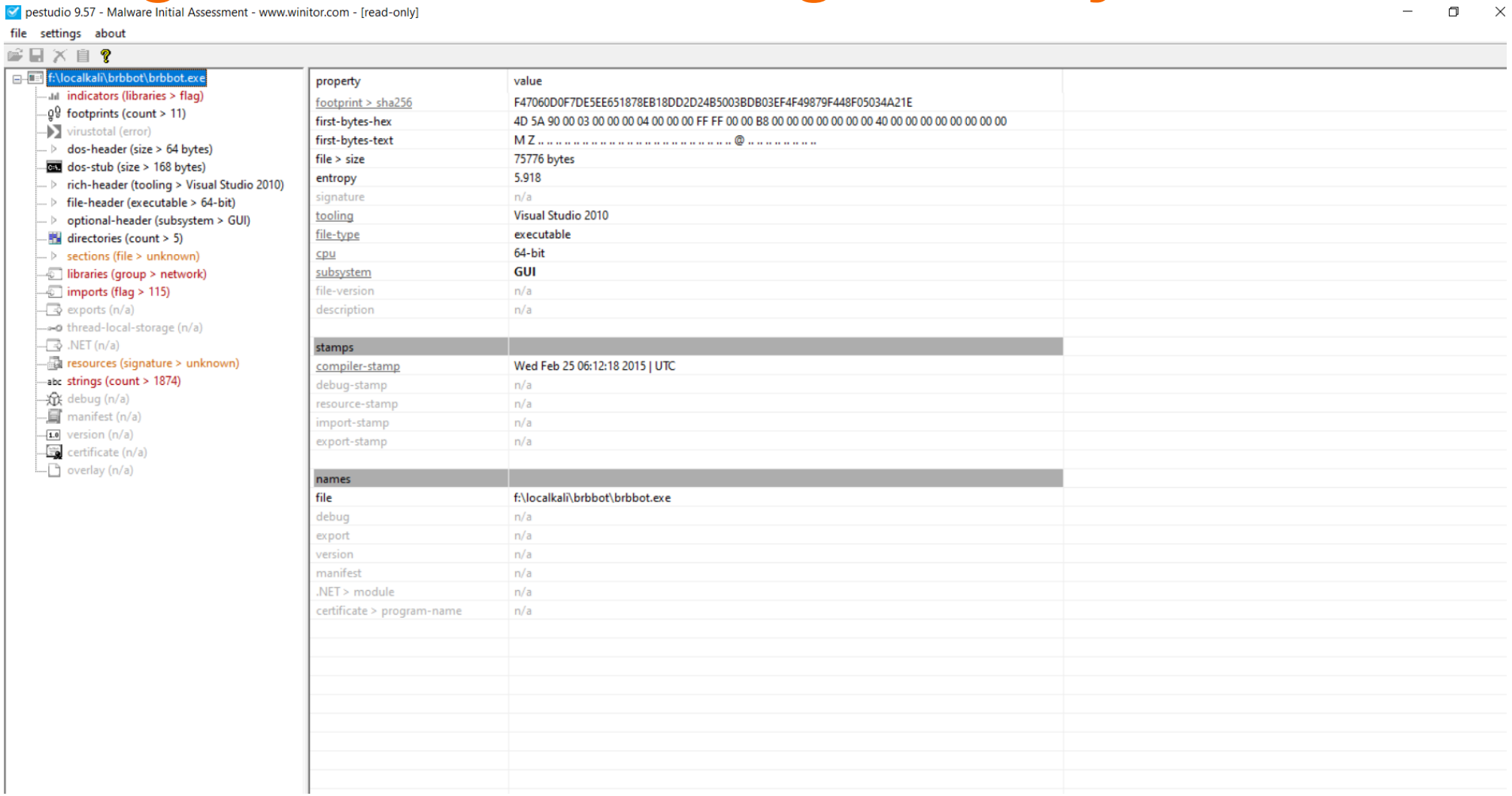
- **11. Use of GetCommandLineW and GetStartupInfoW:** These can be used by malware to intercept command-line arguments or to modify the startup behavior of the program.

Yes or No?

Strings: Suspicious activities! Or NOT?

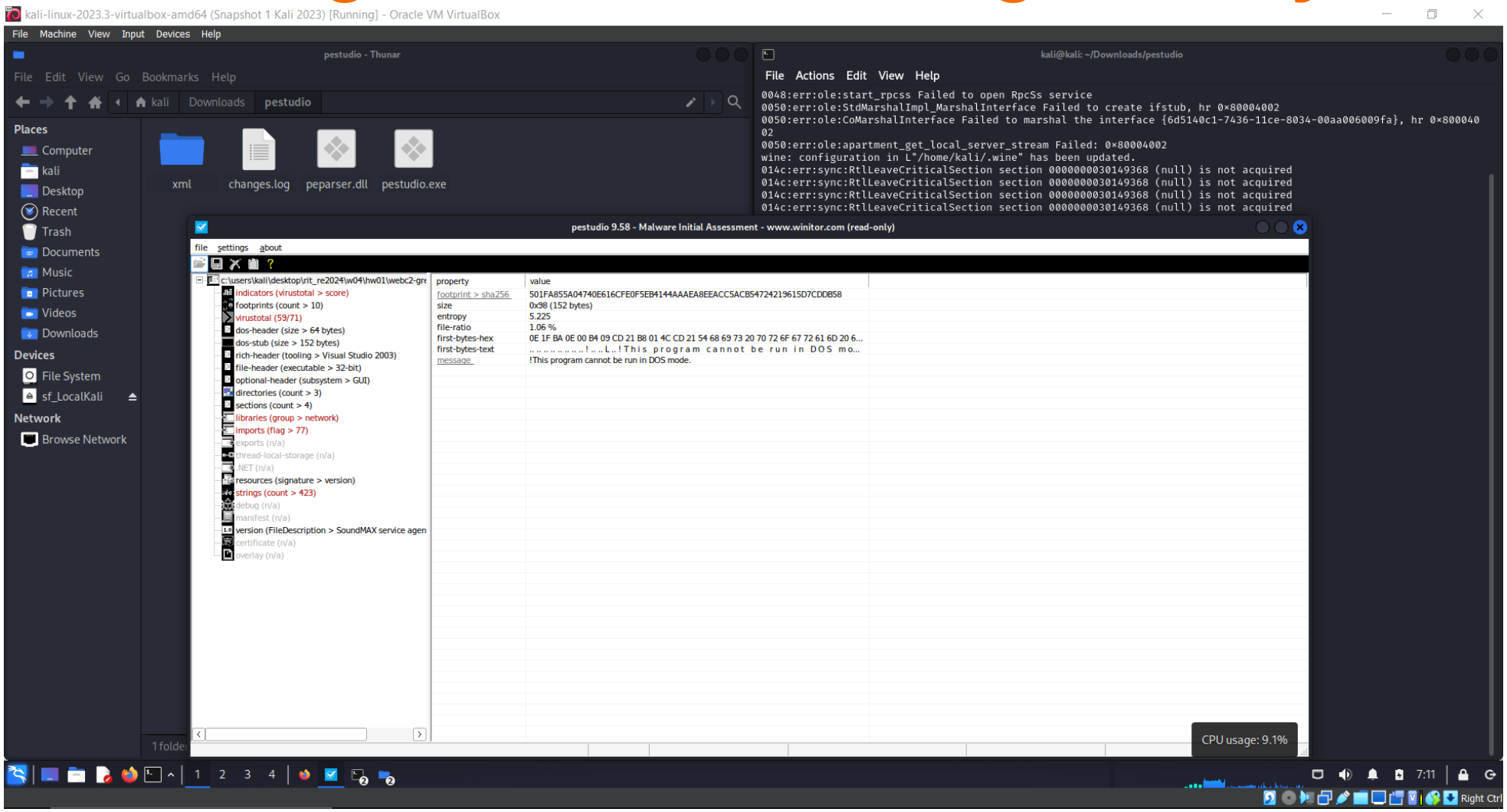
The presence of these strings alone does not confirm malicious intent, as many legitimate programs also use these functions. However, in the context of an executable being analyzed for malware, these strings would warrant a closer examination through dynamic analysis and reverse engineering to understand the context in which these functions are used and determine the true nature of the program.

Strings: PEStudio – Flags it for you!



https://www.winitor.com/download2

Strings: PEStudio – Flags it for you!



PEStudio
On Linux
Use Wine

https://www.winitor.com/download2

What if the Cat is Packed?

Identifying Packing/ Obfuscation Methods

Even though string extraction is an excellent technique to harvest valuable information, often malware authors obfuscate or armor their malware binary

- Malware authors employ **obfuscation** to conceal the malware's **functionality** from security professionals and reverse engineers.
- The primary goal is to **protect the malware's inner workings**, making detection and analysis by anti-virus programs and analysts more challenging.
- Obfuscation can significantly reduce the number of extractable strings, leaving behind strings that are often incomprehensible or meaningless.
- **Tools Used for Obfuscation:**
  - **Packers:** Used to compress and encrypt the malware binary, which can change the file's appearance and structure.
  - **Cryptors:** Employed to encrypt the contents of the malware, hindering static analysis efforts.

Identifying Packing/ Obfuscation Methods

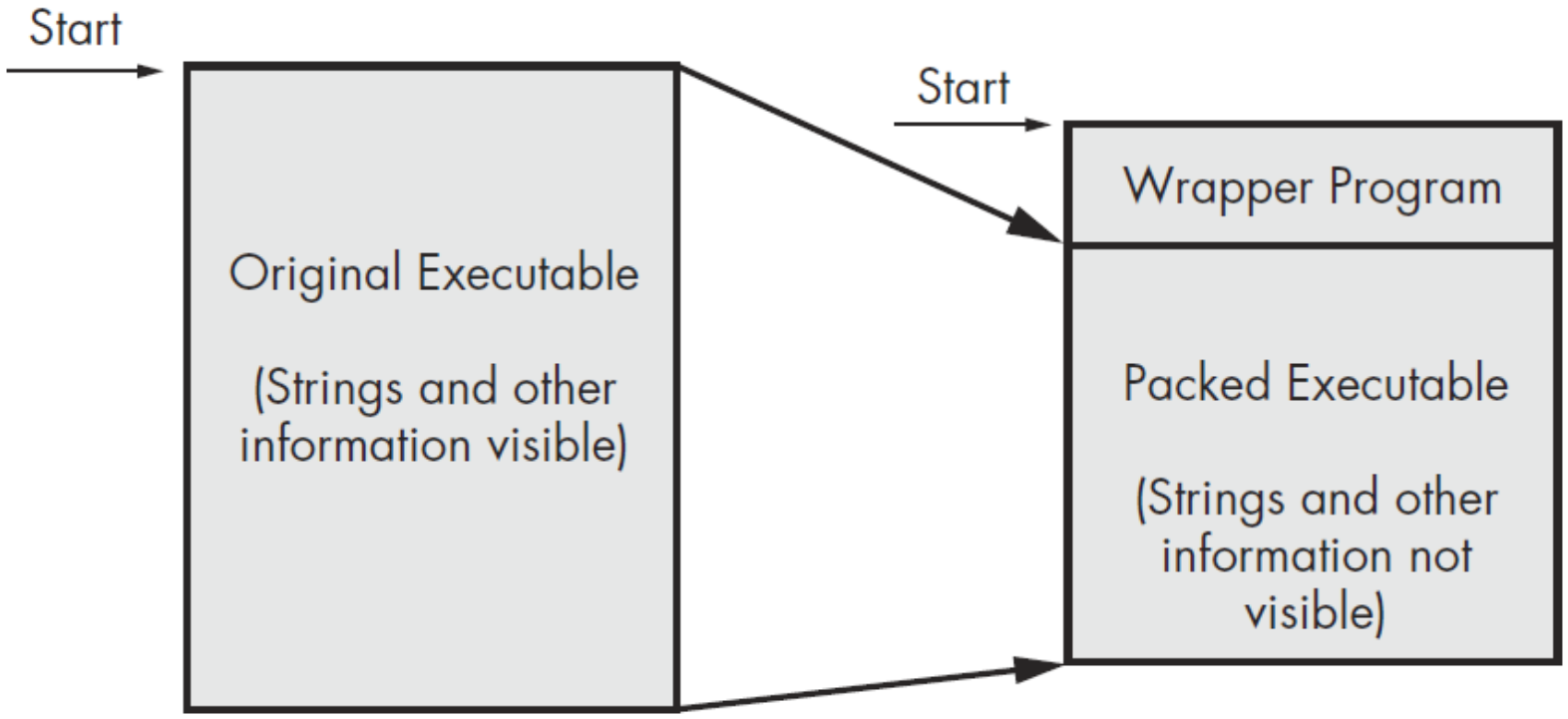
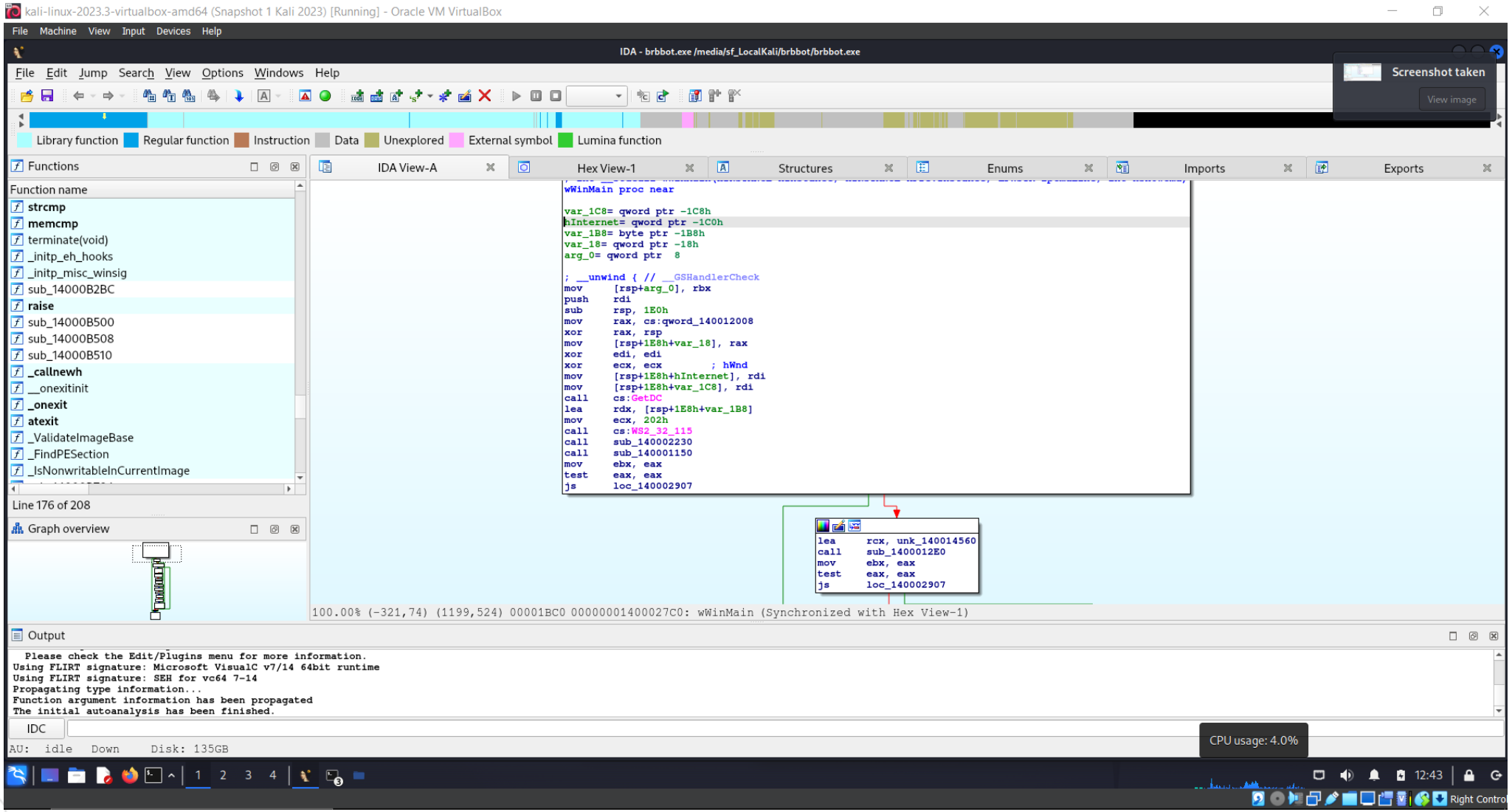
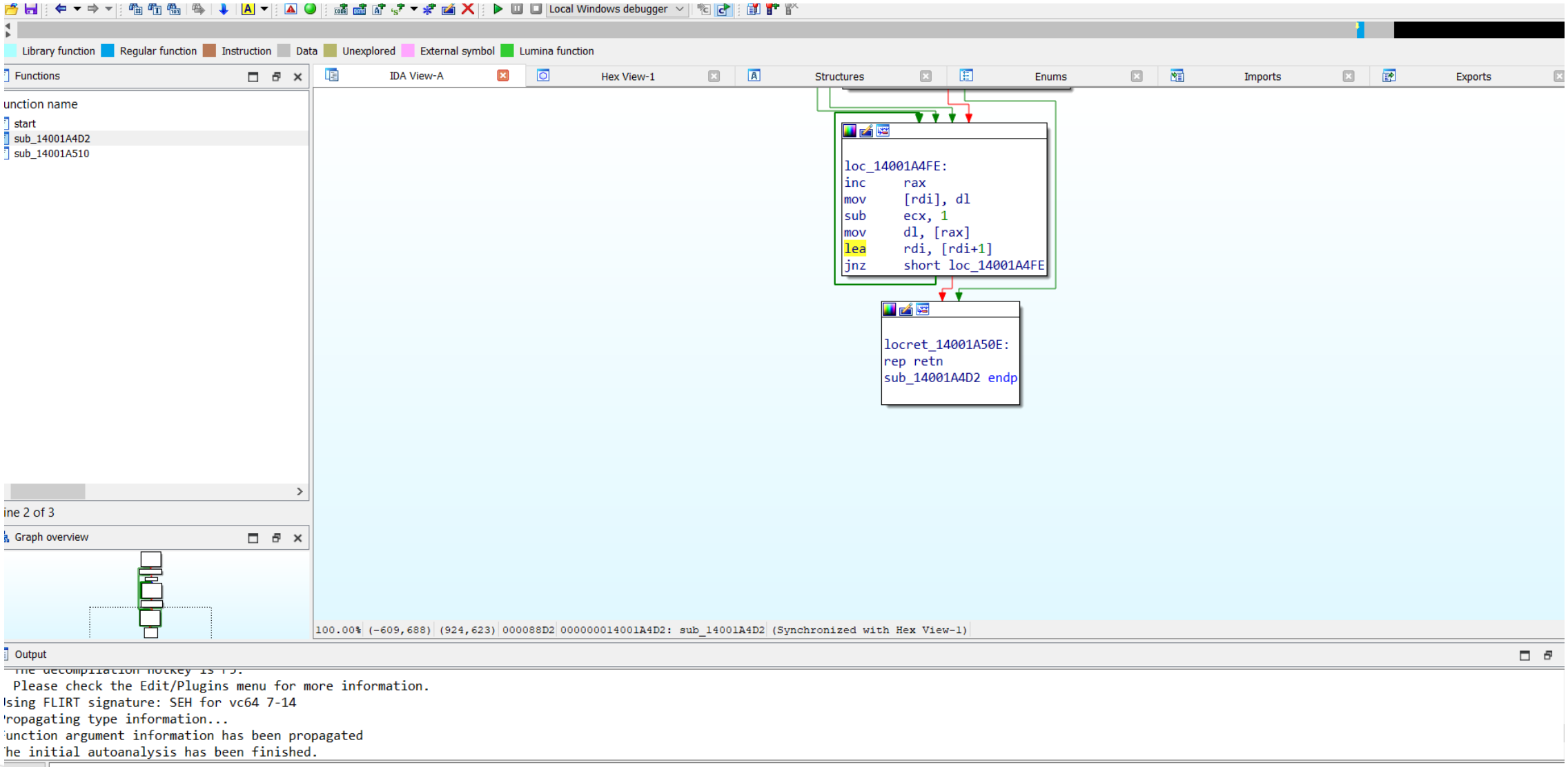


Figure 1-4: The file on the left is the original executable, with all strings, imports, and other information visible. On the right is a packed executable. All of the packed file's strings, imports, and other information are compressed and invisible to most static analysis tools.

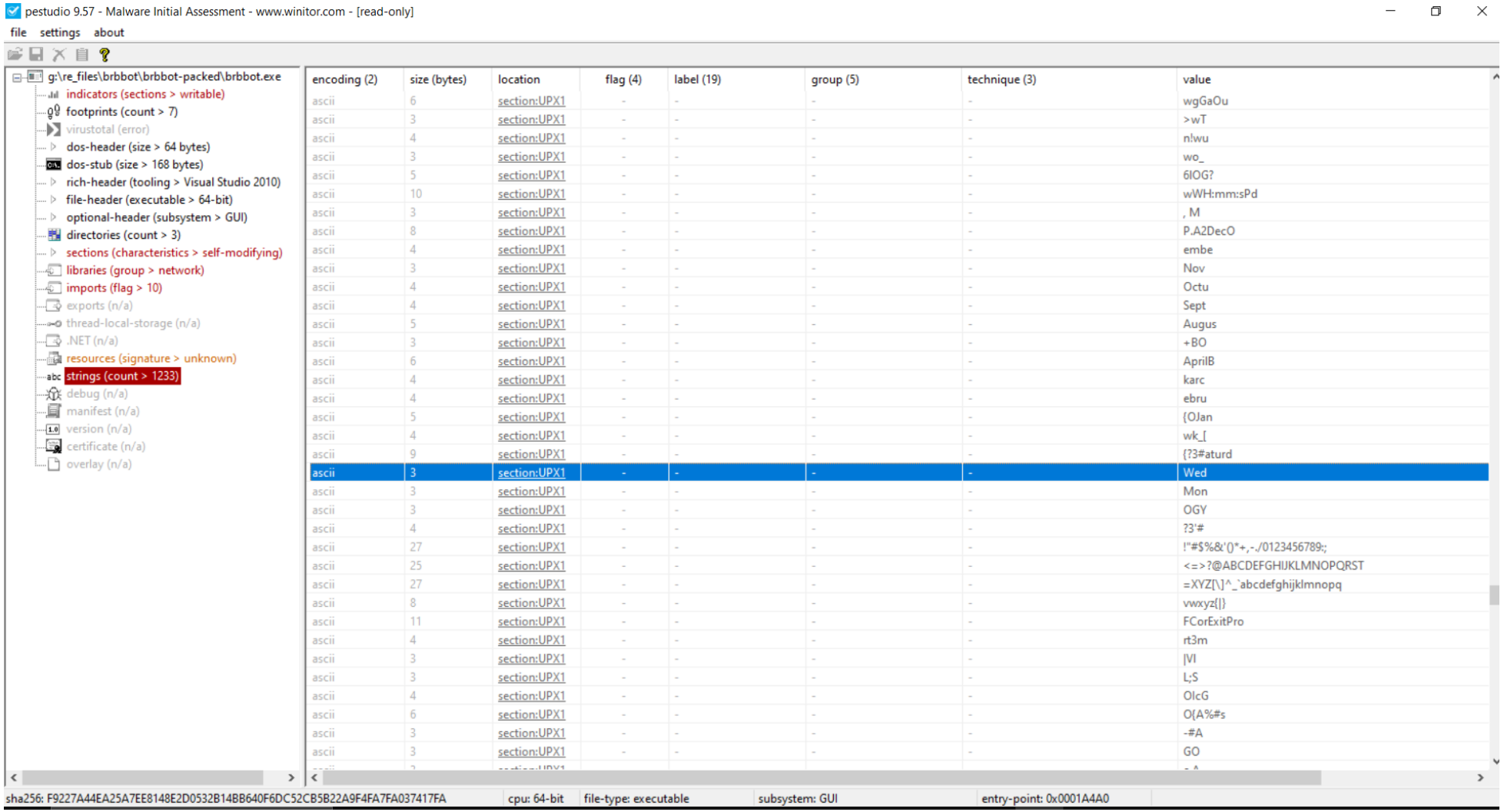
Packing/ Obfuscation (We are not looking)



Packing/ Obfuscation (We are not looking)



Packing/ Obfuscation (We are not looking)



Packing/ Obfuscation

Packing and Obfuscation: Attackers often use packers to compress, encrypt, or otherwise modify a malware executable file to avoid detection by security software.

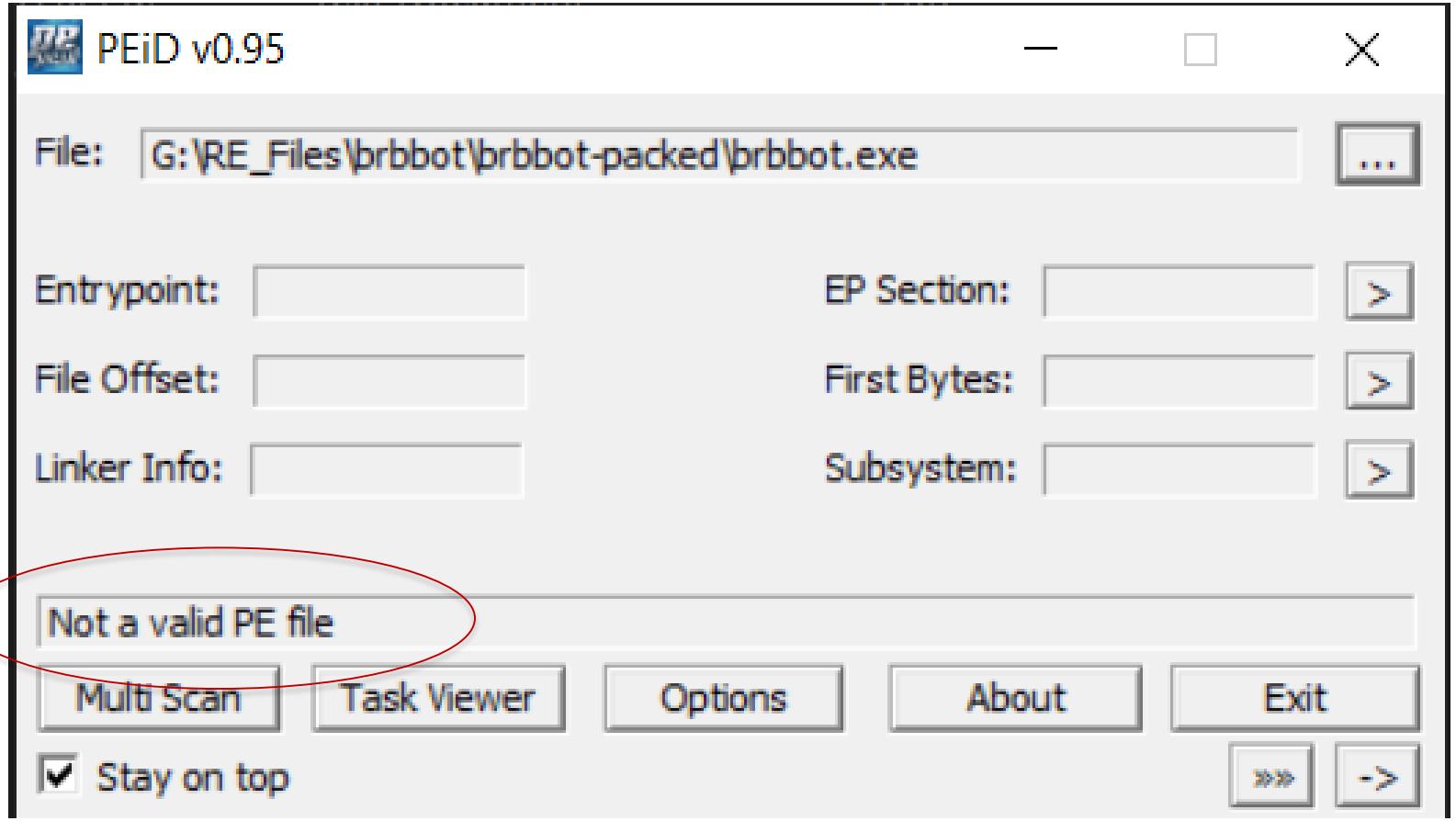
Challenges for Reverse Engineers: These obfuscation techniques complicate the task for reverse engineers. When malware is packed or obfuscated, it becomes more difficult to perform static analysis to understand the malware's actual program logic and other metadata.

Packing/ Obfuscation

PEiD Tool: We can use a tool called PEiD (Portable Executable identifier) to detect most common packers, cryptors, and compilers for PE (Portable Executable) files. ***PE files are executable binaries for Windows operating systems.**

PEiD Capabilities: The PEiD tool can provide details about Windows executable files. It can identify signatures associated with over 600 different packers and compilers. Signatures are distinctive patterns or sequences in the binary code that can be used to identify which tool was used to pack or compile the executable.

Packing/ Obfuscation

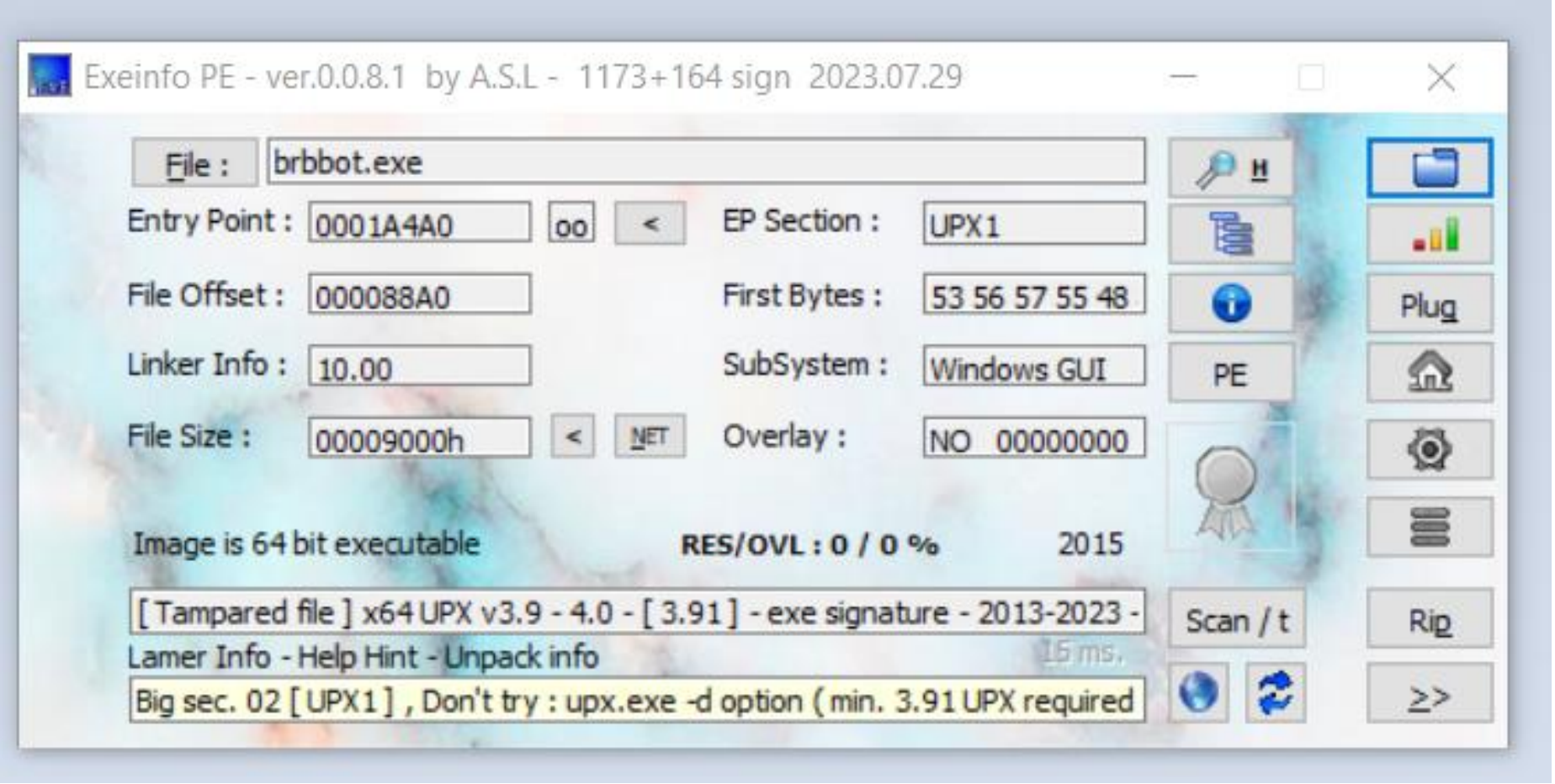


Packing/ Obfuscation

Alternative tool:

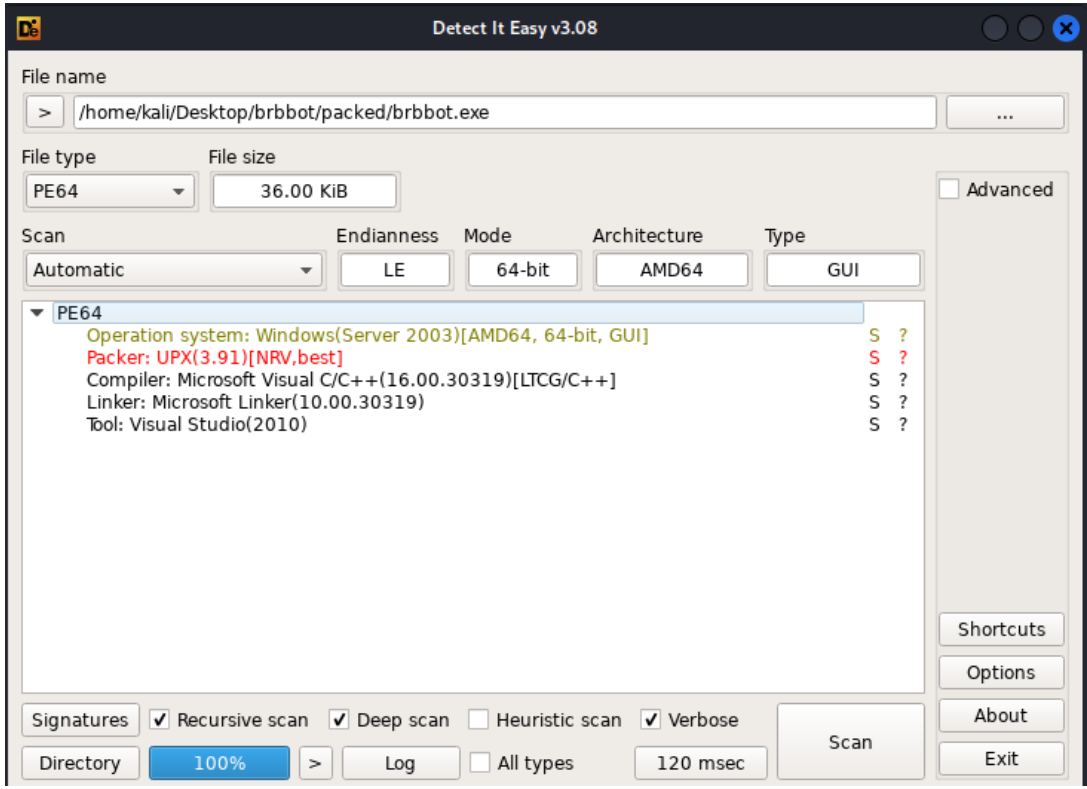
- **Exeinfo PE,** ProtectionID, RDG Packer Detector, CFF Explorer **- for windows**
- **Detect It Easy (DIE) – for Linux**

Packing/ Obfuscation



http://www.exeinfo.xn.pl/

Packing/ Obfuscation



https://github.com/horsicq/Detect-It-Easy

wget https://github.com/horsicq/DIE-engine/releases/download/3.08/Detect_It_Easy-3.08-x86_64.AppImage

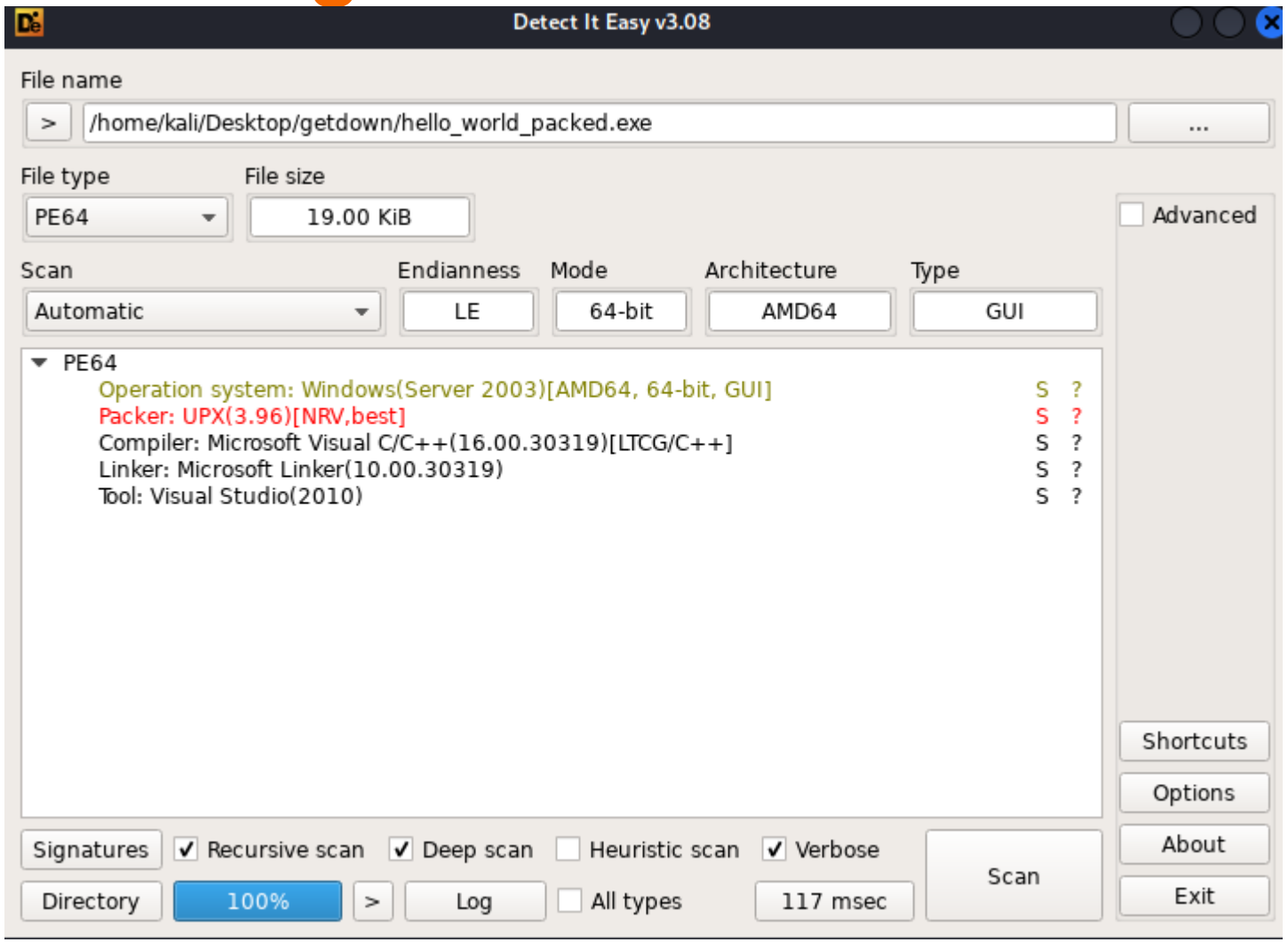
chmod +x Detect_It_Easy-3.08-x86_64.AppImage

Packing/ Obfuscation

List of tools that are used for packaging/obfuscation, which includes:

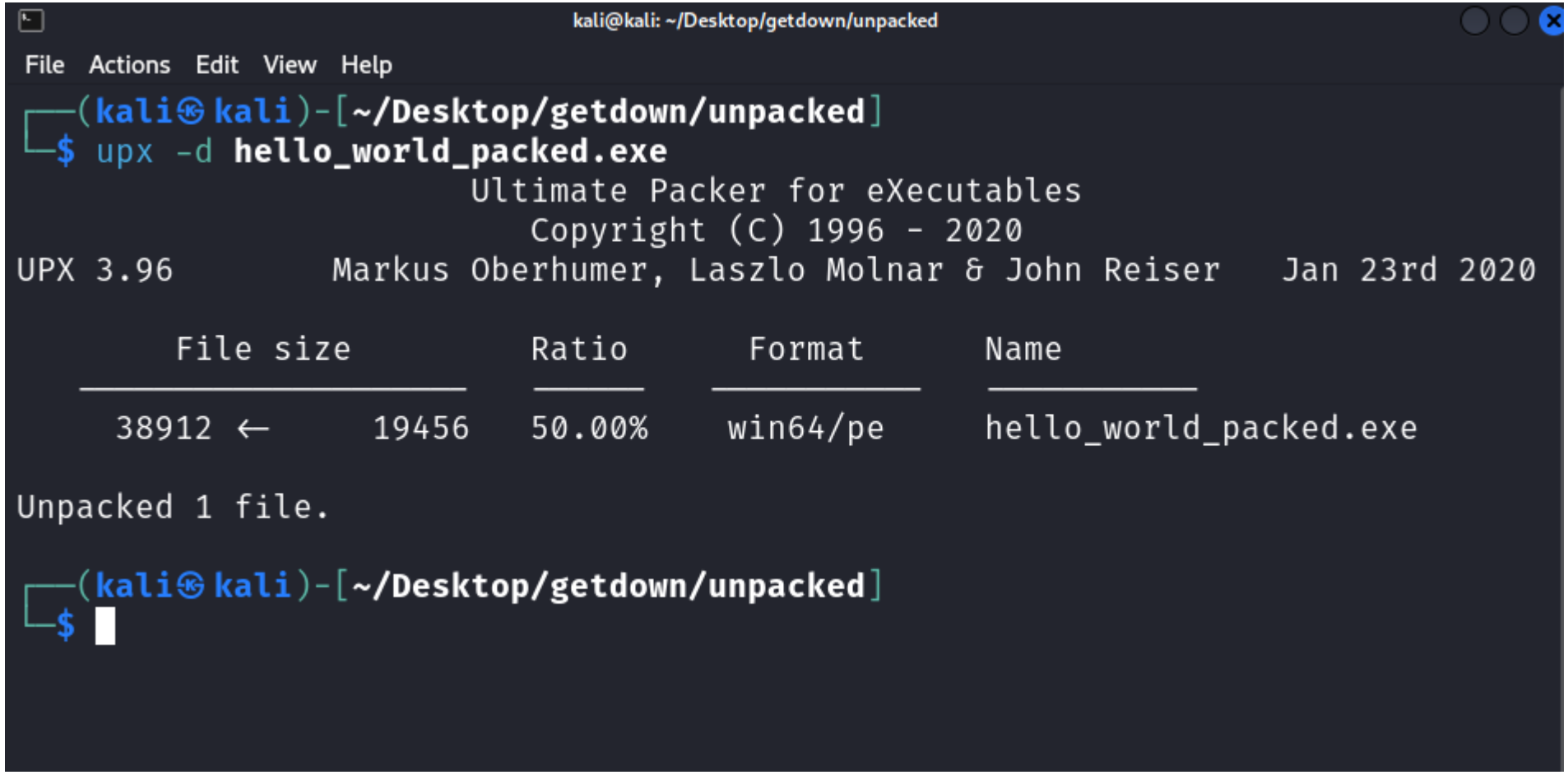
- **Macro_Pack:** A tool that can be found on GitHub, used for obfuscation.
- **UPX:** The Ultimate Packer for eXecutables, a free, portable, extendable, high-performance executable packer.
- **ASPack:** A commercial software for Windows, which is designed to compress executable files.
- **VMProtect:** A tool that protects code by executing it on a virtual machine with non-standard architecture that makes it difficult to analyze and crack.
- **ps2-packer:** Another tool available on GitHub that is likely used for packing software.

Packing/ Obfuscation – Hello World packed



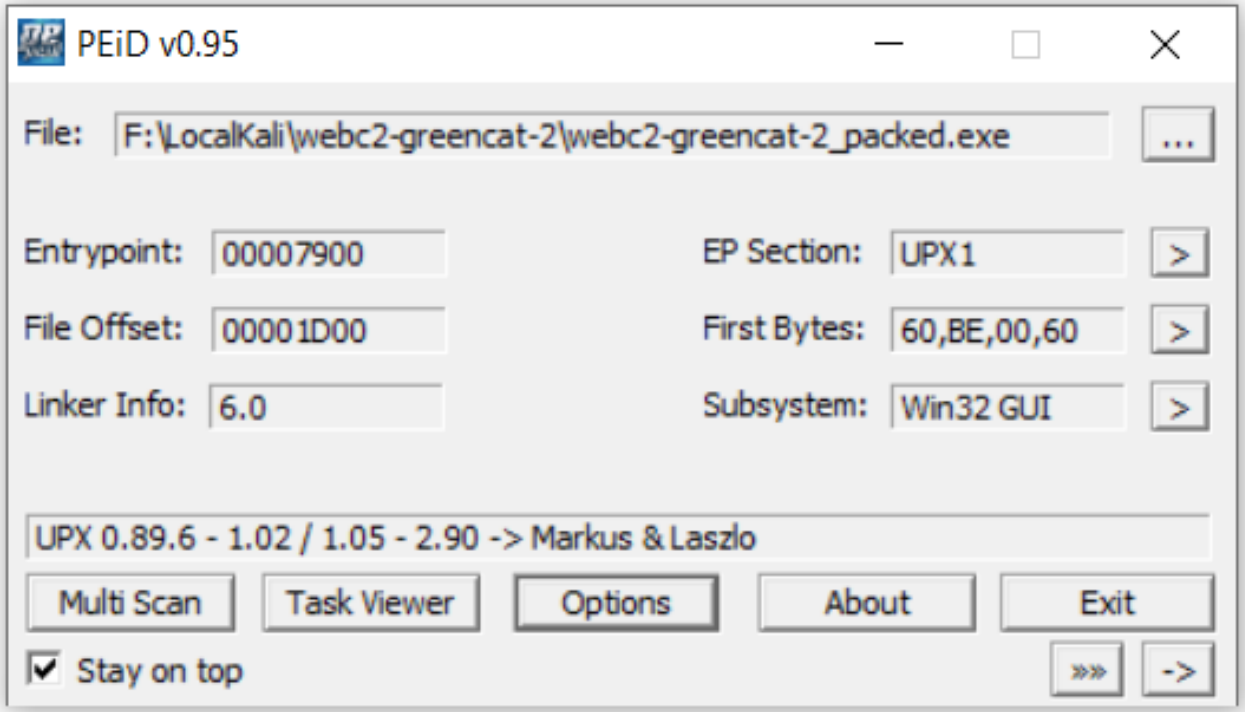
$upx -o hello_world_packed.exe getdown.exe

Packing/ Obfuscation – Hello World Un-packed



$ upx -d hello_world_packed.exe

Packing/ Obfuscation



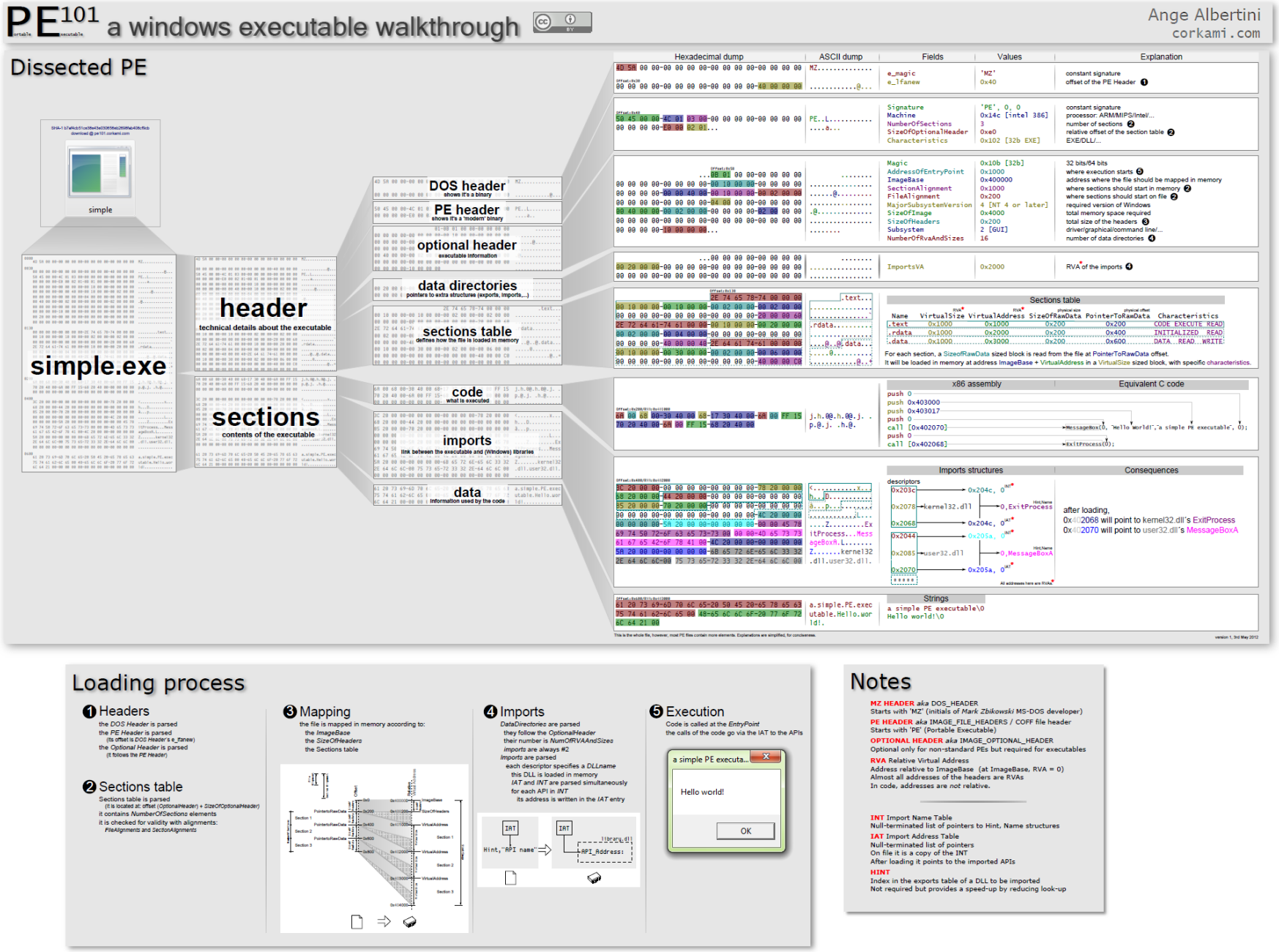
HW1: Is the "GreenCat"Packed?

Finding the Portable Executable (PE) information

- The **PE** (**P**ortable **E**xecutable) format is a **file format for executables**, object code, DLLs, FON Font files, and others used in 32-bit and 64-bit versions of Windows operating systems.

When the binary is executed, the operating system loader reads the information from the PE header and then loads the binary content from the file into the memory.

Finding the Portable Executable (PE) information



Finding the Portable Executable (PE) information

The PE header contains information such as :

- where the executable needs to be loaded into memory,
- the address where the execution starts,
- the list of libraries/functions on which the application relies on, and
- the resources used by the binary.

Finding the Portable Executable (PE) information

A PE file consists of several sections, each of which contains a specific type of data. The exact number of sections can vary, as it is defined by the programmer or compiler.

Structure of a PE File

- **Headers:** Contain information about the file, such as its size, entry point, and resource directory.
- **Sections:** Contain the actual code, data, and resources of the file. There are many different sections, such as the .text section for code, the .data section for initialized data, and the .bss section for uninitialized data.

Finding the Portable Executable (PE) information

Some standard sections typically found in PE files:

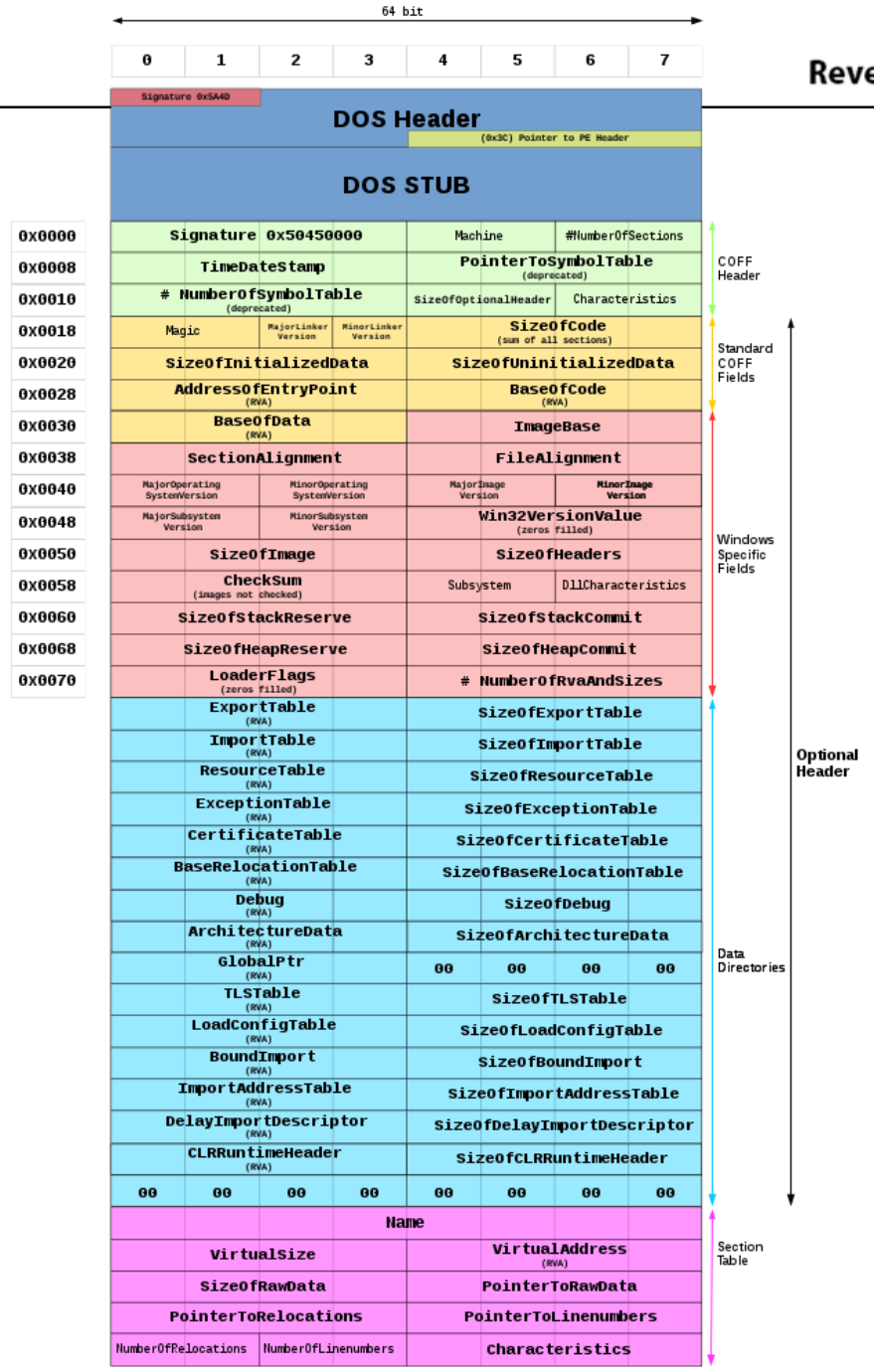
- **.text:** This section contains the executable code.
- **.data:** This section holds initialized data, such as global and static variables.
- **.rdata:** This section includes read-only data, such as string literals and constant data.
- **.bss:** Holds uninitialized data. It doesn't take up space in the file but is allocated in memory **when the program starts.**

Finding the Portable Executable (PE) information

Some standard sections typically found in PE files:

- **.idata:** Contains **import data**, such as imported function names and pointers to them.
- **.edata:** Contains **export data**, including names of functions that can be accessed by other programs.
- **.rsrc:** This section stores **resources** used by the executable, such as icons, menus, and dialog boxes.
- **.reloc:** Holds **relocation information**, necessary for programs that can be loaded at different memory addresses.

Finding the Portable Executable (PE) information



Finding the Portable Executable (PE) information

Common Tools for PE Analysis

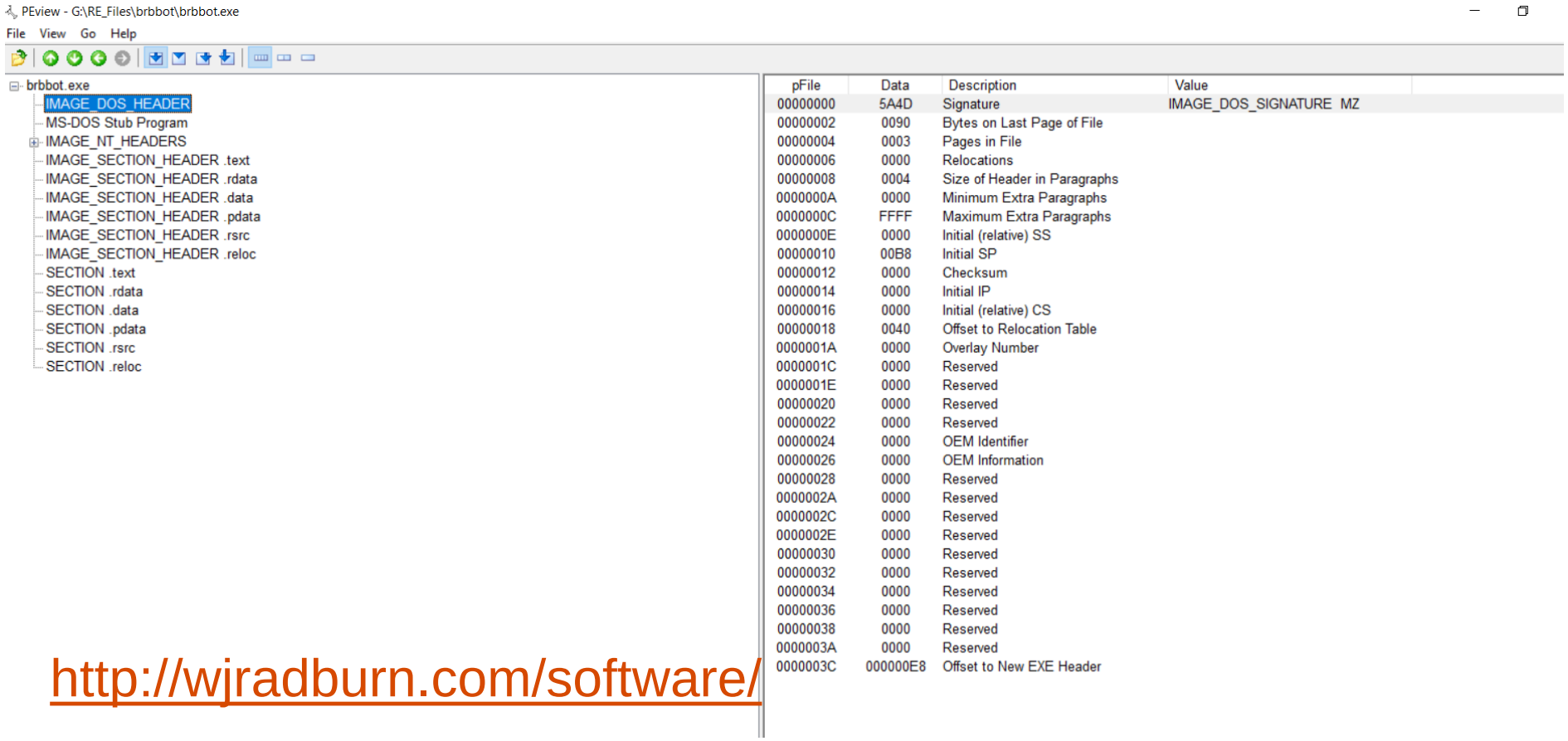
PE Explorer: A comprehensive tool for opening, viewing, and editing **32-bit PE files** such as EXEs, DLLs, and ActiveX Controls.

Additional PE Analysis Tools:

- Portable Executable Scanner (**pescan**): Useful for scanning and analyzing PE files. Available at tzworks.net.
- **Resource Hacker**: A versatile tool for resource management in Windows executables. Available at angusj.com.
- **PEView:** Allows inspection of the internal structure of PE files. Available at aldeid.com.

Finding the Portable Executable (PE) information

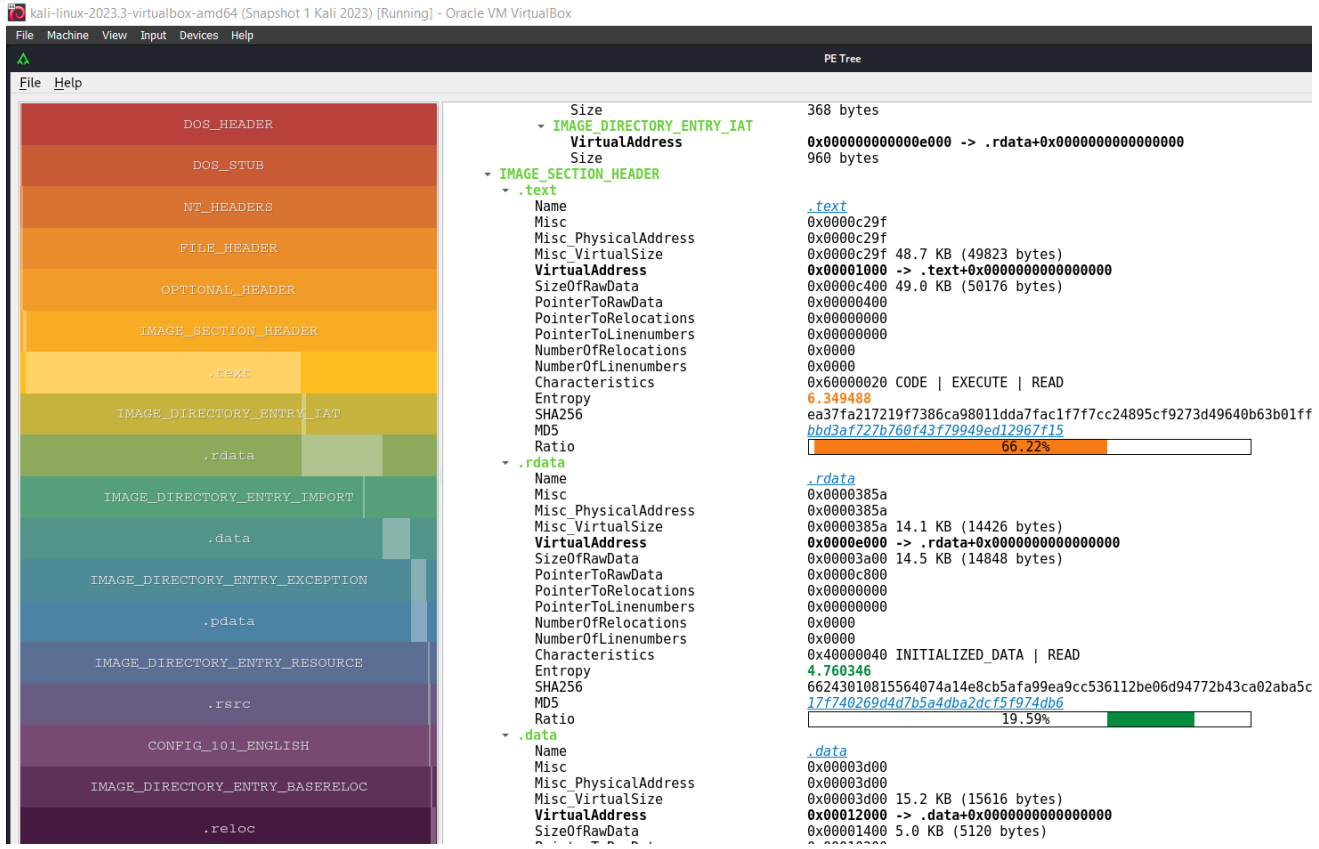
PEView



http://wjradburn.com/software/

Finding the Portable Executable (PE) information

PE Tree

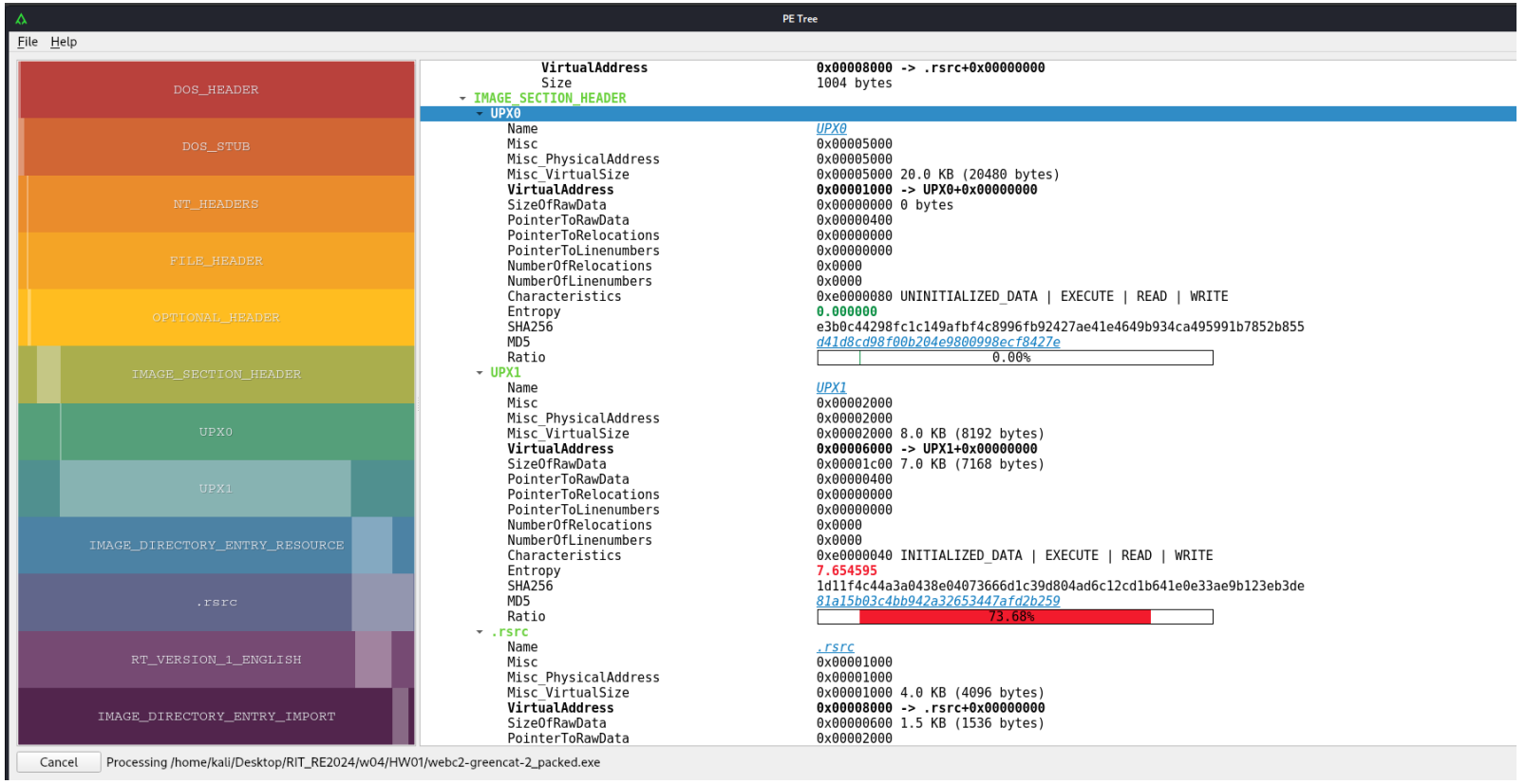



```
$ git clone https://github.com/blackberry/pe_tree.git
$ cd pe_tree
$ python3 -m venv env
$ source ./env/bin/activate
$ pip install -e .
```

https://github.com/blackberry/pe_tree

Finding the Portable Executable (PE) information

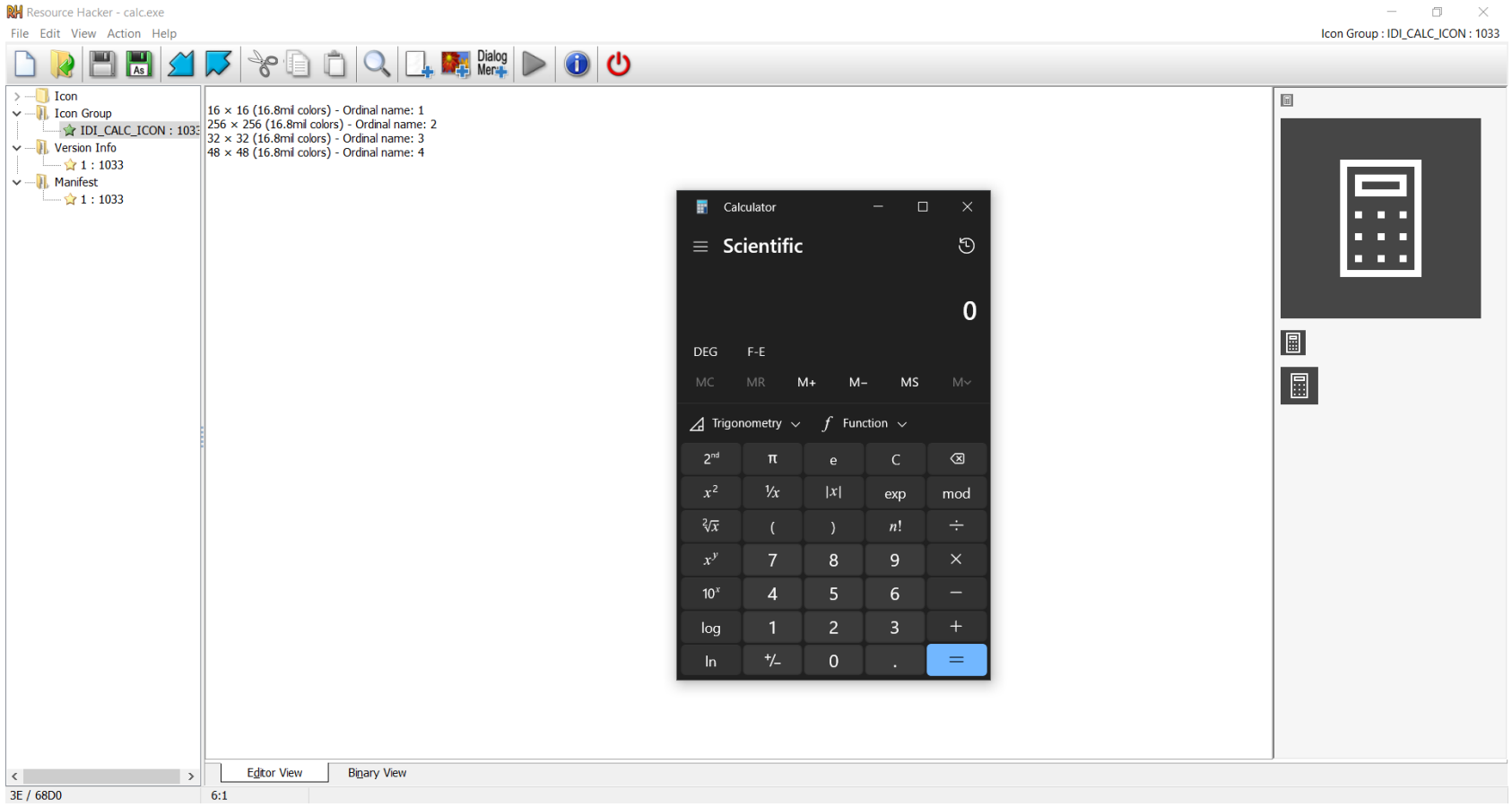
PETree




```
┌──(env)─(kali㉿kali)-[~/Desktop/RIT_RE2024/w04/HW01]
└─$ pe-tree webc2-greencat-2_packed.exe
```

Finding the Portable Executable (PE) information

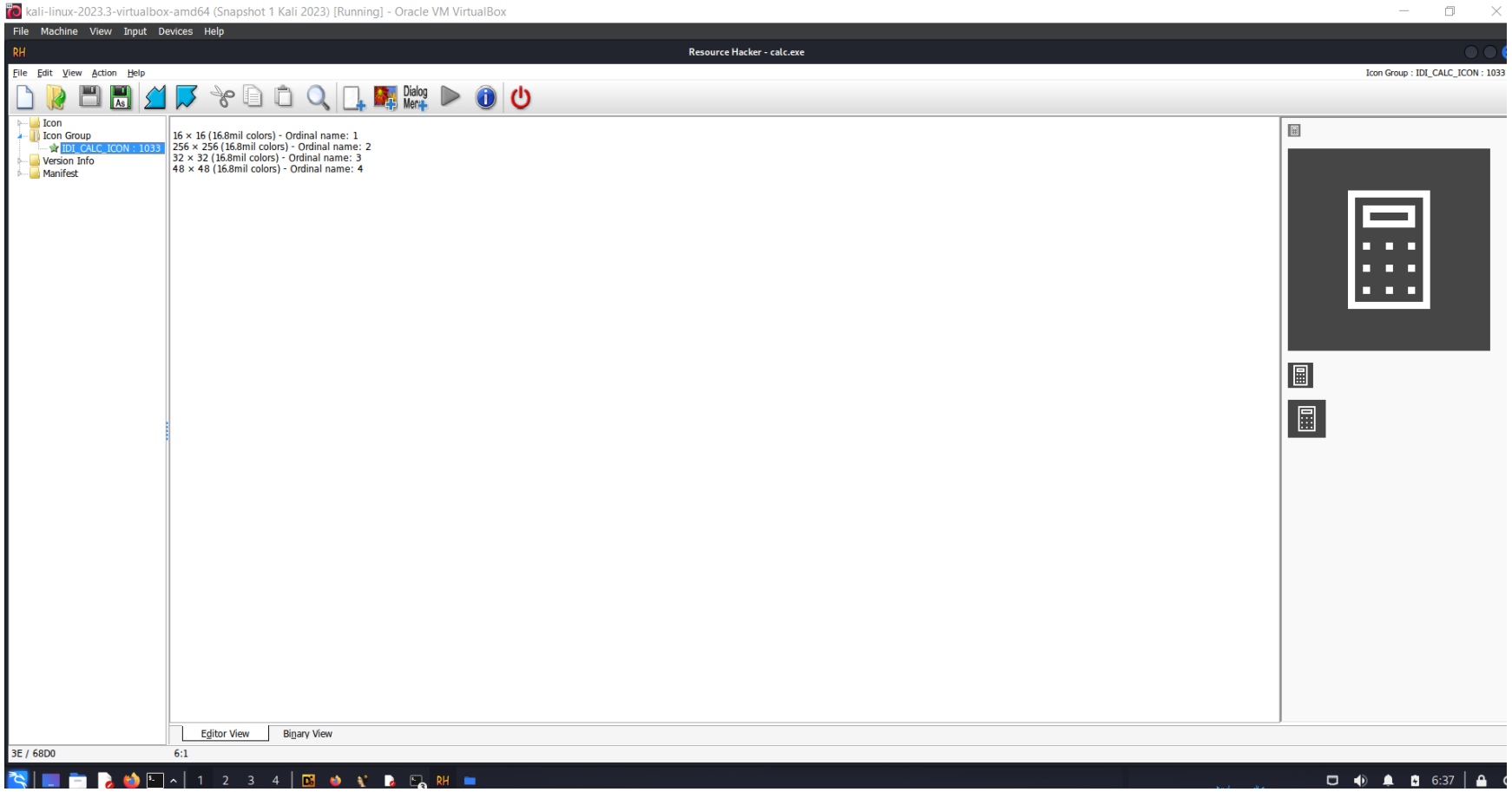
Resource Hacker



https://www.majorgeeks.com/files/details/resource_hacker.html

Finding the Portable Executable (PE) information

Resource Hacker On Linux



https://snapcraft.io/install/resourcehacker/debian

Identifying File Dependencies

- Programs can **borrow code (link)** from libraries instead of rewriting it.
- This connection (linking) can happen **before**, **during the start**, or **after program execution**.
- Knowing how this connection works is crucial for analyzing malware, especially understanding details inside the program file.

Identifying File Dependencies

Static Linking

- Incorporates all necessary library code directly into the executable at **compile time.**
- Results in **a larger executable** file size
- It's hard to distinguish between the statically linked library code and the original code of the executable because the PE file header doesn't indicate the presence of statically linked code.

Identifying File Dependencies

Runtime Linking

- Connects to library functions **as needed** during the program's execution.

- Less predictable and often utilized by malware for obfuscation purposes.

Identifying File Dependencies

Dynamic Linking

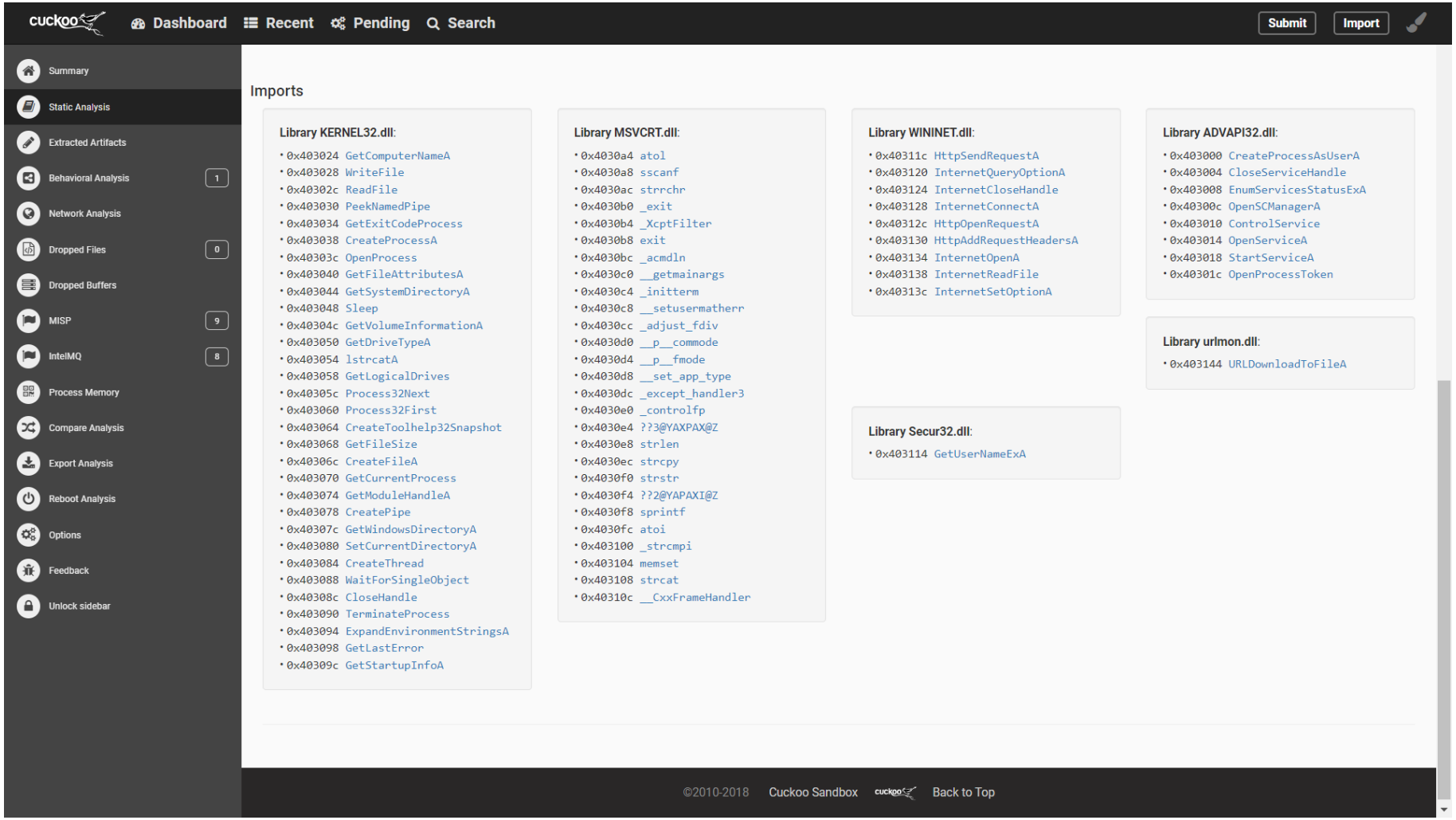
- Resolves program dependencies at program **load time**, rather than at compile time.
- More space-efficient and facilitates easier analysis of a program's potential actions.

Identifying File Dependencies

Common DLL

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. 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.

Identifying File Dependencies



https://cuckoo.cert.ee/analysis/3886261/static/

Identifying File Dependencies

Imports

Library KERNEL32.dll:

- 0x403024 GetComputerNameA
- 0x403028 WriteFile
- 0x40302c ReadFile
- 0x403030 PeekNamedPipe
- 0x403034 GetExitCodeProcess
- 0x403038 CreateProcessA
- 0x40303c OpenProcess
- 0x403040 GetFileAttributesA
- 0x403044 GetSystemDirectoryA
- 0x403048 Sleep
- 0x40304c GetVolumeInformationA
- 0x403050 GetDriveTypeA
- 0x403054 lstrcatA

Library WININET.dll:

- 0x40311c HttpSendRequestA
- 0x403120 InternetQueryOptionA
- 0x403124 InternetCloseHandle
- 0x403128 InternetConnectA
- 0x40312c HttpOpenRequestA
- 0x403130 HttpAddRequestHeadersA
- 0x403134 InternetOpenA
- 0x403138 InternetReadFile
- 0x40313c InternetSetOptionA

Identifying File Dependencies

A comprehensive analysis of function imports from various libraries reveals the multifaceted nature of a potential malware threat. For Example, this malware may be capable of:

Process Management: Utilizing functions such as CreateProcessA, OpenProcess, and TerminateProcess from KERNEL32.dll to control system processes, which can disrupt or manipulate applications and services.

File Operations: Employing ReadFile and WriteFile to access and modify data, potentially leading to theft or corruption of information.

System Interaction: Leveraging GetSystemDirectoryA and CreateToolhelp32Snapshot for system directory access and process enumeration, which could be indicative of injection or evasion techniques.

Identifying File Dependencies

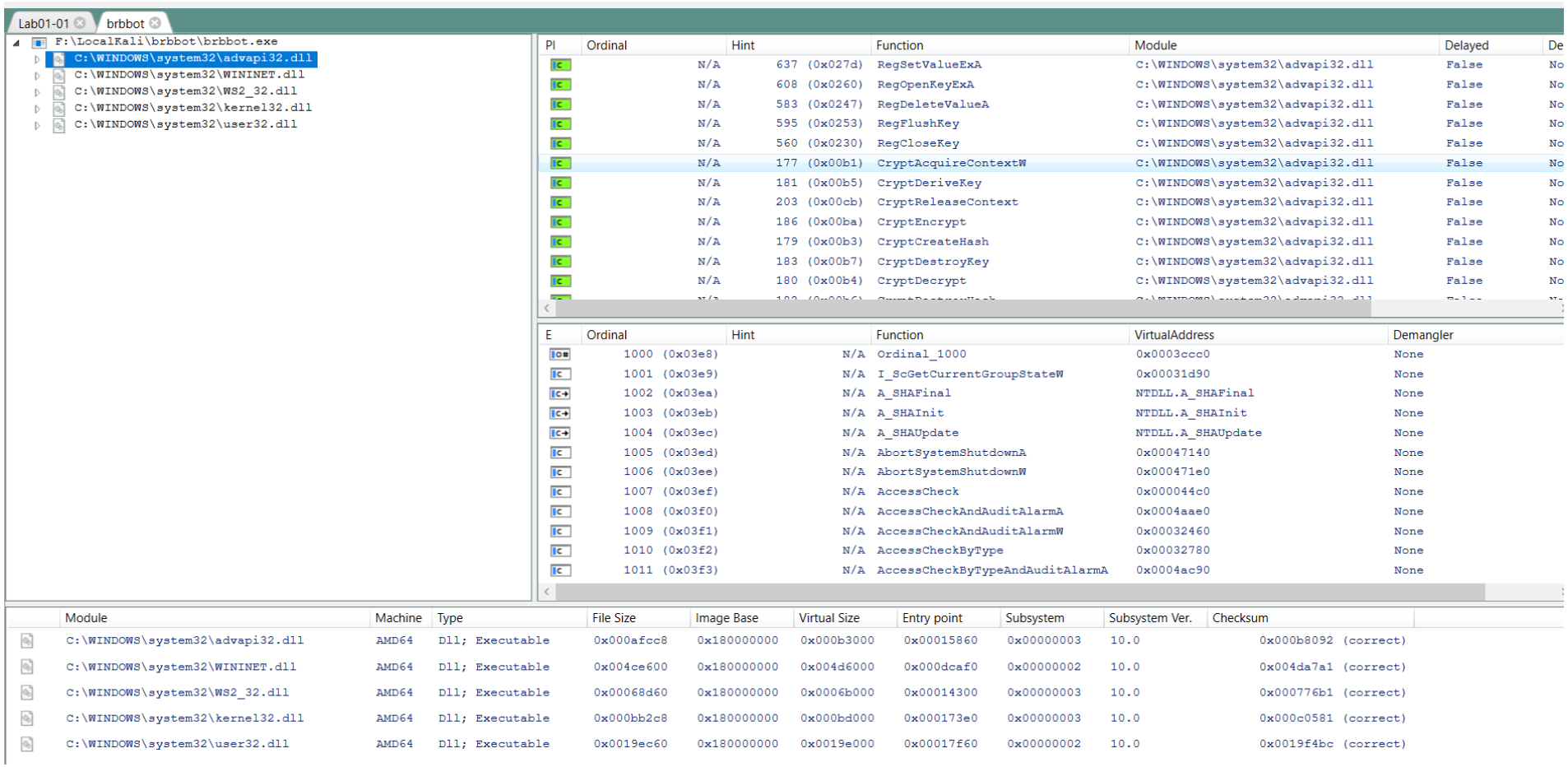
Network Communication: Using WININET.dll functions like InternetOpenA and HttpSendRequestA to establish network connections for command and control activities or unauthorized data transmission.

Service Manipulation: Accessing ADVAPI32.dll's OpenServiceA and StartServiceA to interact with system services, likely for establishing persistence or initiating unauthorized operations.

Privilege Escalation: Manipulating access tokens through OpenProcessToken, hinting at attempts to gain elevated privileges.

These indicators point to a malware designed for stealth, persistence, and a broad spectrum of malicious activities, emphasizing the need for robust cybersecurity measures.

Identifying File Dependencies



https://github.com/lucasg/Dependencies/

Identifying File Dependencies

Other Dependency Checking Tools

- Dependency Walker (*http://www.dependencywalker.com/*)
- Dependency-check (https://jeremylong.github.io)
- Snyk (https://snyk.io)
- PE Explorer Dependency Scanner (http://www.pe-explorer.com)
- RetireJS (https://retirejs.github.io)

Hands – On:

Troop #	Malware	Done?
1	Lab01-01.exe	
2	Lab01-02.exe	
3	Lab01-03.exe	
4	Lab01-04.exe	
5	Lab03-01.exe	
6	Lab03-03.exe	
7	Lab03-03.exe Lab03-04.exe	

https://nostarch.com/malware

Course Overview

- **Title: "CSEC 202 - Reverse Engineering Fundamentals"**

Instructor	Office	Phone	Email	Semester-Year
Emad Abu Khousa	B217		eakcad@rit.edu	Spring-2024
Office Hours:	M: 12:00-01:00 TR: 11:00-12:00			

- **600: TR 12:05-01:20, Room D-006**
- **601: MW 01:05-02:25, Room C-109**
- **602: TR 01:30-02:50, Room D-207**

Thank You and Q&A