***Windows Forensics Practical (WFP)***

Group 17 – Malia  
2024-07-09

Table of Contents

[Change Control iii](#_Toc171719546)

[I. Objectives 1](#_Toc171719547)

[II. 4-in-1 Shell Items Parser Tool 2](#_Toc171719548)

[A. How it Works? 2](#_Toc171719549)

[B. How is it Better? 2](#_Toc171719550)

[C. Code Explanation 3](#_Toc171719551)

[III. Activity Screenshots 4](#_Toc171719552)

[A. Prerequisites 4](#_Toc171719553)

[B. Running the Tool 4](#_Toc171719554)

[1. Shortcut Parser 5](#_Toc171719555)

[2. Jump List Parser 9](#_Toc171719556)

[3. Prefetch Parser 13](#_Toc171719557)

[4. Shellbags Parser 16](#_Toc171719558)

[IV. Conclusion 21](#_Toc171719559)

[V. Appendix (Shell Items Parser and Collection Tool Code) 22](#_Toc171719560)

[VI. References 31](#_Toc171719561)

[Figure 1. tool\_paths code snippet 4](#_Toc171719509)

[Figure 2. Running the Shell Items Parser and Collection Tool (main interface) 4](#_Toc171719510)

[Figure 3. Selecting Shortcut Parser and Live Analysis 5](#_Toc171719511)

[Figure 4. Shortcut Parser Live Analysis finished 6](#_Toc171719512)

[Figure 5. Folder 'parsed\_data' is created after Shortcut Parser live analysis 6](#_Toc171719513)

[Figure 6. 'parsed\_data' folder (all csv and json files from the Shortcut Parser live analysis are located here) 7](#_Toc171719514)

[Figure 7. Selecting Shortcut Parser and Offline Analysis (a dictionary of shortcut file paths is displayed, categorized by folder) 7](#_Toc171719515)

[Figure 8. Selecting Shortcut Parser and Offline Analysis (snippet of file paths from the dictionary of shortcut file paths) 8](#_Toc171719516)

[Figure 9. Shortcut Parser Offline Analysis finished 8](#_Toc171719517)

[Figure 10. Folder 'parsed\_data' is created after Shortcut Parser offline analysis 9](#_Toc171719518)

[Figure 11. 'parsed\_data' folder (the csv and json files of the selected file from the Shortcut Parser offline analysis are located here) 9](#_Toc171719519)

[Figure 12. Selecting Jump List Parser and Live Analysis 10](#_Toc171719520)

[Figure 13. Jump List Parser Live Analysis finished 10](#_Toc171719521)

[Figure 14. Folder 'parsed\_data' is created after Jump List Parser live analysis 10](#_Toc171719522)

[Figure 15. 'parsed\_data' folder (all csv and json files from the Jump List Parser live analysis are located here) 11](#_Toc171719523)

[Figure 16. Selecting Jump List Parser and Offline Analysis (a dictionary of Jump List file paths is displayed, categorized by folder) 11](#_Toc171719524)

[Figure 17. Selecting Jump List Parser and Offline Analysis (snippet of file paths from the dictionary of Jump List file paths) 12](#_Toc171719525)

[Figure 18. Jump List Parser Offline Analysis finished 12](#_Toc171719526)

[Figure 19. Folder 'parsed\_data' is created after Jump List Parser offline analysis 12](#_Toc171719527)

[Figure 20. 'parsed\_data' folder (the csv and json files of the selected file from the Jump List Parser offline analysis are located here) 13](#_Toc171719528)

[Figure 21. Selecting Prefetch Parser and Live Analysis 13](#_Toc171719529)

[Figure 22. Prefetch Parser Live Analysis finished 14](#_Toc171719530)

[Figure 23. Folder 'parsed\_data' is created after Prefetch Parser live analysis 14](#_Toc171719531)

[Figure 24. 'parsed\_data' folder (all csv and json files from the Prefetch Parser live analysis are located here) 14](#_Toc171719532)

[Figure 25. Selecting Prefetch Parser and Offline Analysis (a dictionary of Prefetch file paths is displayed, categorized by folder) 15](#_Toc171719533)

[Figure 26. Selecting Prefetch Parser and Offline Analysis (snippet of file paths from the dictionary of Prefetch file paths) 15](#_Toc171719534)

[Figure 27. Prefetch Parser Offline Analysis finished 16](#_Toc171719535)

[Figure 28. Folder 'parsed\_data' is created after Prefetch Parser offline analysis 16](#_Toc171719536)

[Figure 29. 'parsed\_data' folder (the csv and json files of the selected file from the Prefetch Parser offline analysis are located here) 16](#_Toc171719537)

[Figure 30. Selecting Shellbags Parser and Live Analysis 17](#_Toc171719538)

[Figure 31. Shellbags Parser Live Analysis finished 17](#_Toc171719539)

[Figure 32. Folder 'parsed\_data' is created after Shellbags Parser live analysis 17](#_Toc171719540)

[Figure 33. 'parsed\_data' folder (all csv and json files from the Shellbags Parser live analysis are located here) 18](#_Toc171719541)

[Figure 34. Selecting Shellbags Parser and Offline Analysis (a dictionary of Shellbags file paths is displayed, categorized by folder) 18](#_Toc171719542)

[Figure 35. Shellbags Parser Offline Analysis finished 19](#_Toc171719543)

[Figure 36. Folder 'parsed\_data' is created after Shellbags Parser offline analysis 19](#_Toc171719544)

[Figure 37. 'parsed\_data' folder (the csv and json files of the selected file from the Shellbags Parser offline analysis are located here) 20](#_Toc171719545)

# Change Control

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Updates | Author | Date |
| 0.1 | Draft | Gideon Andrew L. Malia | 2024-07-09 |
| 0.2 | Working Document | Gideon Andrew L. Malia | 2024-07-12 |
| 0.x |  |  |  |
| 1.0 |  |  |  |

# Objectives

In this project, the group aims:

1. to develop a comprehensive Python codebase capable of parsing and collecting various types of Shell Items, including shortcuts, jump lists, prefetch files, and shellbags, by integrating prebuilt tools;
2. to create a dynamic and versatile tool that can operate live or offline to analyze and extract valuable information from Shell Items, enhancing forensic analysis, system investigation, and data recovery processes; and
3. to facilitate efficient forensic analysis by providing robust parsing capabilities for different Shell Items, including shortcuts, jump lists, prefetch files, and shellbags, and ensuring seamless collaboration with other parsers.

# 4-in-1 Shell Items Parser Tool

The 4-in-1 Shell Items Parser Tool is a Python-based tool designed for comprehensive forensic analysis by parsing various types of Shell Items found in Windows. It integrates specialized prebuilt tools for parsing shortcuts (.LNK files), jump lists, prefetch files, or shellbags. This integration allows users to efficiently extract and analyze crucial data from these items, providing a streamlined and detailed approach to digital forensics.

This tool will give a valuable contribution in digital forensics because it integrates robust and specialized parsers for various Shell Items, ensuring comprehensive and accurate data extraction. The tool's ability to operate offline and live analysis modes adds significant value, allowing forensic investigators to adapt to different scenarios and requirements. By streamlining the forensic analysis process and providing detailed insights into user activity and system usage, this tool enhances the efficiency and effectiveness of forensic investigations, ultimately aiding in the timely resolution of cases and the recovery of valuable data.

## How it Works?

The tool begins with a menu interface wherein users can select the type of Shell Item they wish to parse. The options included parsing shortcuts, jump lists, prefetch files, and shellbags. Once a selection is made, the tool searches predefined system directories for relevant files such as shortcut files on the desktop, in the start menu, taskbar, and recent items. Users are prompted to specify a file or directory for parsing, and the tool validates the input to ensure that the file or directory exists before proceeding. The tool then calls the appropriate prebuilt parser (e.g. LECmd.exe for shortcuts, JLECmd.exe for jump lists, PECmd.exe for prefetch files, or SBECmd.exe for shellbags) using a subprocess. The parsed output was saved in the CSV and JSON formats for further analysis. Additionally, the tool offers a live analysis mode in which it continuously monitors specified directories for new files and parses them in real-time, providing immediate insights into changes. Robust error handling ensures smooth operation and manages issues such as files not found errors, subprocess errors, and invalid user inputs. A visual explanation can be read at Running the Tool section.

## How is it Better?

The primary advantage of this tool lies in its integration with specialized tools: LECmd.exe, JLECmd.exe, PECmd.exe, and SBECmd.exe. These tools, developed by Eric Zimmerman, are renowned for their accuracy and detailed analytical capabilities in the forensic community. By leveraging these advanced capabilities, the shell parser tool ensures a high precision in parsing different Shell Items. Unlike other tools that focus on a single type of Shell Item, this tool provides a unified interface for parsing multiple types. This comprehensive approach saves time and effort because investigators can analyze all relevant Shell Items using a single tool without switching between different parsers.

The automated discovery, selection, and parsing of files significantly enhances efficiency. The tool's ability to convert CSV output to the JSON format facilitates easier integration with other analysis tools and workflows, further streamlining the forensic process. Its versatility in operating both offline and live analysis modes makes it adaptable to various forensics scenarios. It can analyze data from dead systems, where live analysis tools may fail, providing flexibility and robustness in different investigative contexts. An intuitive menu interface and clear user prompts make the tool accessible to users with varying levels of technical expertise. Detailed error messages and guided input ensure a smooth user experience, reducing the likelihood of errors, and enhancing usability.

Compared to specialized tools such as LECmd.exe and JLECmd.exe, which are highly specialized for parsing shortcuts and jump lists, respectively, integration into this shell parser tool benefits from their detailed analysis capabilities while offering a more comprehensive solution. PECmd.exe, designed specifically for parsing Windows prefetch files, provides insights into the application execution history, and SBECmd.exe, focused on parsing shellbags, offers metadata about folder access through Windows Explorer, which is essential for understanding user activity on the system. The integration of these specialized tools into a single Python-based shell parser enhances efficiency by automating the processes of file discovery, selection, and parsing. Compared with other parsers that may require manual handling and switching between different tools, this shell parser streamlines the entire process, saving significant time and effort.

The live analysis feature sets this tool apart by allowing real-time monitoring of specified directories. This capability is crucial for ongoing investigations where immediate insights into system changes are necessary, providing a significant advantage over traditional methods that only offer static analysis. This Python-based shell parser tool offers a comprehensive, efficient, and user-friendly solution for parsing various Shell Items in Windows. Integrating specialized tools and providing a unified interface for multiple types of Shell Items enhances the forensic analysis process. Its versatility in offline and live analysis modes, combined with robust error handling and automated workflows, makes it a superior choice for forensic investigators seeking detailed and reliable insights into system activity.

## Code Explanation

The Python code was structured to provide a user-friendly interface, starting with a menu-driven system that allowed users to select the type of Shell Item they wanted to parse. The ‘display\_menu’ function was created to present the options, and corresponding functions for listing files (‘list\_shortcut\_files’, ‘list\_jump\_list\_files’, ‘list\_prefetch\_files’, ‘list\_shellbags\_files’) were developed to search for specific system directories and gather relevant files. These functions utilize the ‘list\_files’ helper function to handle the file-discovery process.

A ‘parse\_file’ function was developed to handle the parsing process. This function takes the tool path and file path as arguments, executes the appropriate command using the subprocess module, and manages the output by saving it in the CSV and JSON formats. The ‘convert\_csv\_to\_json’ function was included to facilitate this conversion, ensuring data could be easily integrated into other analytical tools or workflows.

A crucial aspect of the tool is its ability to perform live analysis. The ‘live\_analysis’ function was implemented to continuously monitor the specified directories for new files. This function uses a loop to periodically check for new files and parse them in real-time, thereby providing immediate insights into system changes. Error handling was a significant focus throughout the code, with checks for file existence, subprocess errors, and invalid user inputs to ensure smooth operation and a positive user experience.

The final step was to ensure the tool's flexibility and adaptability by allowing it to operate in offline and live analysis modes. The ‘get\_live\_choice’ function prompts users to choose their preferred mode, and the code is structured to handle each scenario appropriately. This comprehensive approach resulted in a robust, efficient, and user-friendly shell parser tool capable of providing a detailed forensic analysis of various Shell Items in Windows systems.

# Activity Screenshots

## Prerequisites

Before running the tool, make sure to download [.NET Framework 6](https://dotnet.microsoft.com/en-us/download/dotnet/6.0) for the pre-built tools to work seamlessly. Also, download the following pre-built tools as these tools are primarily used to create the Shell Items Parser and Collection tool (all tools can also be downloaded collectively by downloading [EZ Tools](https://ericzimmerman.github.io/#!index.md)):

* [LECmd.exe](https://www.sans.org/tools/lecmd/)
* [JLECmd.exe](https://www.sans.org/tools/jlecmd/)
* [PECmd.exe](https://www.sans.org/tools/pecmd/)
* [SBECmd.exe](https://www.sans.org/tools/sbecmd/)

After downloading, make sure to change ‘tool\_paths’ in the source code to the corresponding paths of the parsers. Follow the format to avoid errors.

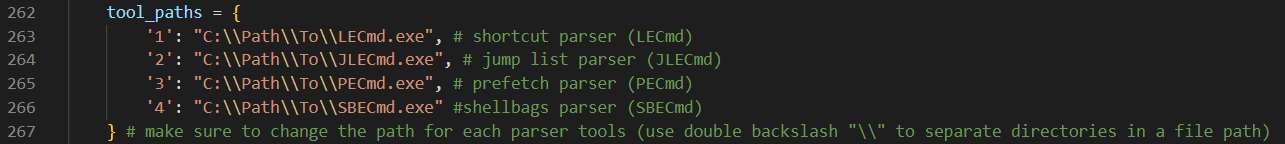


Figure 1. tool\_paths code snippet

## Running the Tool

To run the Shell Items Tool, make sure to run the tool in a command line with Administrator privileges to avoid encountering any errors in the future.

A screenshot of a computer

Description automatically generated

Figure 2. Running the Shell Items Parser and Collection Tool (main interface)

Choosing live analysis will parse all the directories listed on each parser, while choosing offline analysis will allow the user to input a single path for parsing. For convenience, the program lists all file paths in each corresponding directory when offline analysis is chosen, enabling the user to simply copy and paste the desired path. After the analysis, the csv and json files of the parsed file will be stored on a folder ‘parsed\_data’. If the folder is not present, the program will create this folder where the program is executed.

### Shortcut Parser

For the shortcut parser, the program will search for shortcuts in the following directories:

* Desktop Path: C:\Users\<YourUsername>\Desktop
* Start Menu Paths:
  + C:\ProgramData\Microsoft\Windows\Start Menu\Programs
  + C:\Users\<Your Username>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs
* Taskbar Path: C:\Users\<Your Username>\AppData\Roaming\Microsoft\Internet Explorer\Quick Launch\User Pinned\TaskBar
* Recent Items Path: C:\Users\<Your Username>\AppData\Roaming\Microsoft\Windows\Recent

A screenshot of a computer

Description automatically generated

Figure 3. Selecting Shortcut Parser and Live Analysis

A screenshot of a computer

Description automatically generated

Figure 4. Shortcut Parser Live Analysis finished

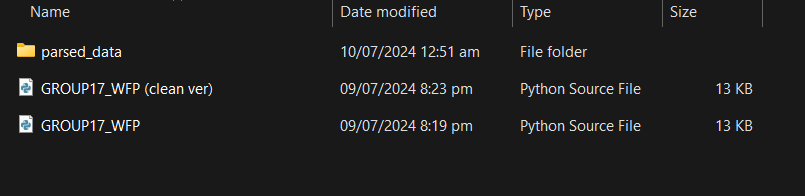


Figure 5. Folder 'parsed\_data' is created after Shortcut Parser live analysis

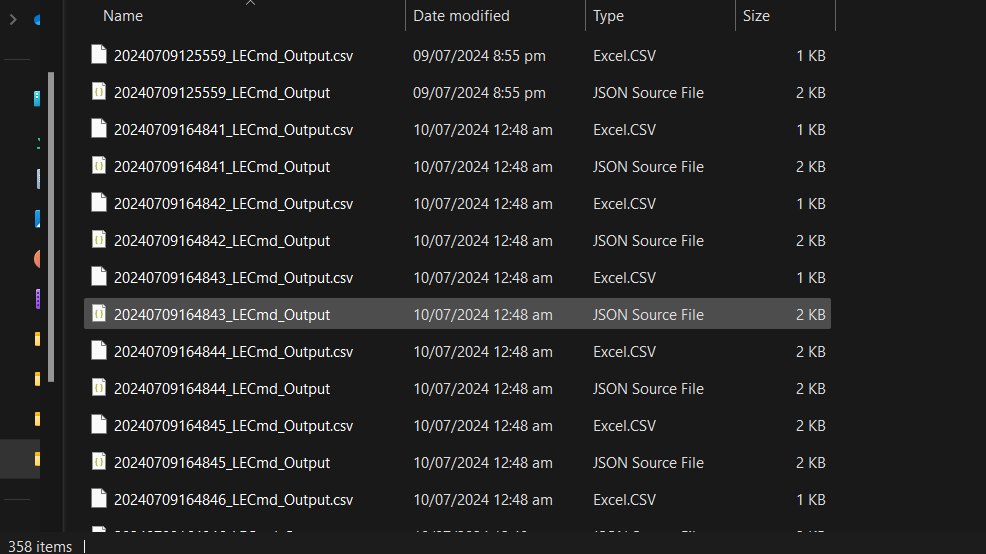


Figure 6. 'parsed\_data' folder (all csv and json files from the Shortcut Parser live analysis are located here)

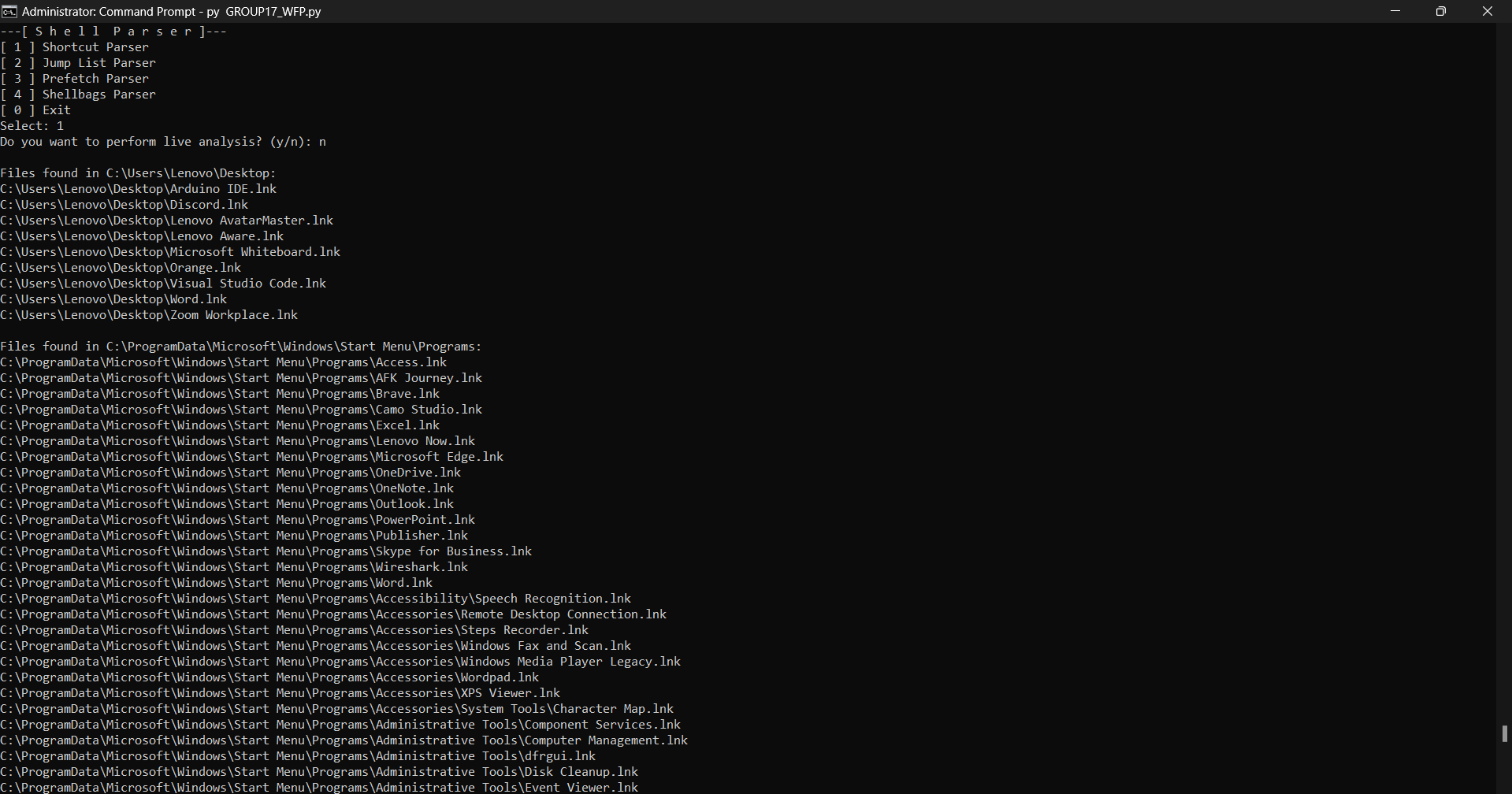


Figure 7. Selecting Shortcut Parser and Offline Analysis (a dictionary of shortcut file paths is displayed, categorized by folder)

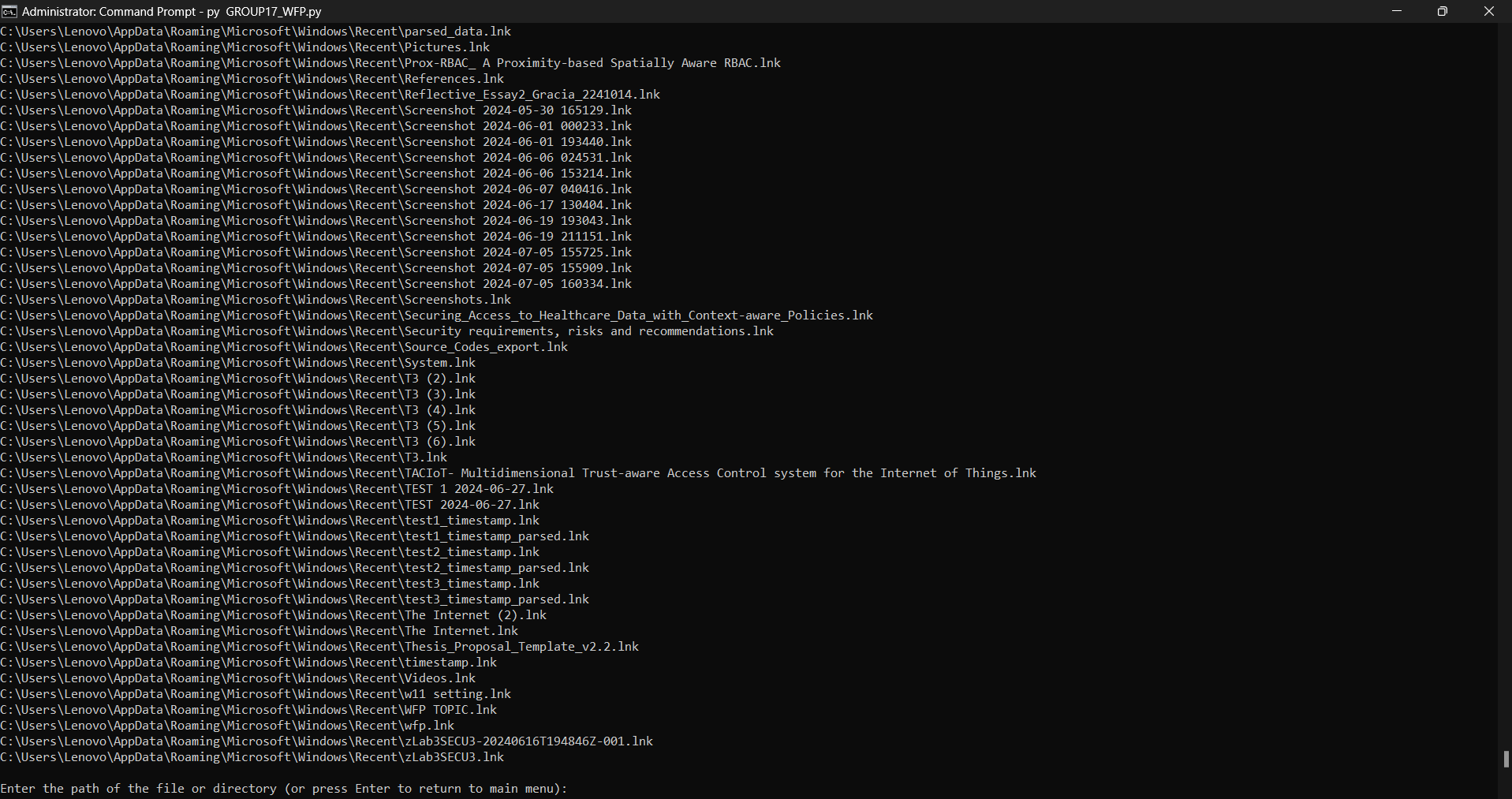


Figure 8. Selecting Shortcut Parser and Offline Analysis (snippet of file paths from the dictionary of shortcut file paths)

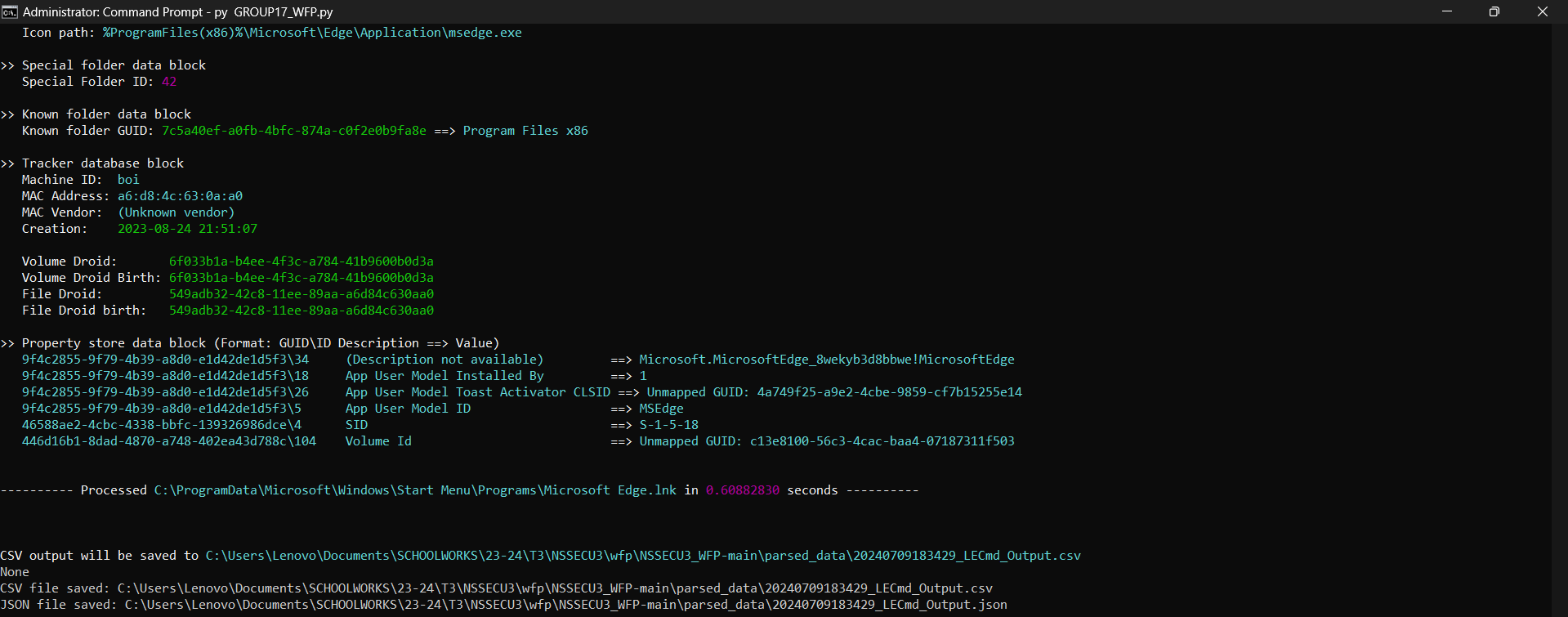


Figure 9. Shortcut Parser Offline Analysis finished

A screenshot of a computer

Description automatically generated

Figure 10. Folder 'parsed\_data' is created after Shortcut Parser offline analysis

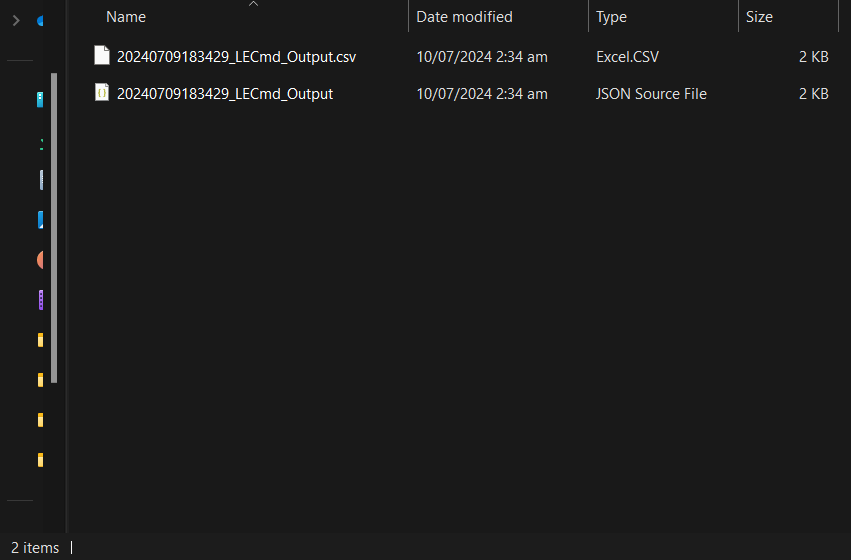


Figure 11. 'parsed\_data' folder (the csv and json files of the selected file from the Shortcut Parser offline analysis are located here)

### Jump List Parser

For the jump list parser, the program will search for shortcuts in the following directories:

* C:\Users\<Your Username>\AppData\Roaming\Microsoft\Windows\Recent\AutomaticDestinations
* C:\Users\<Your Username>\AppData\Roaming\Microsoft\Windows\Recent\CustomDestinations



Figure 12. Selecting Jump List Parser and Live Analysis

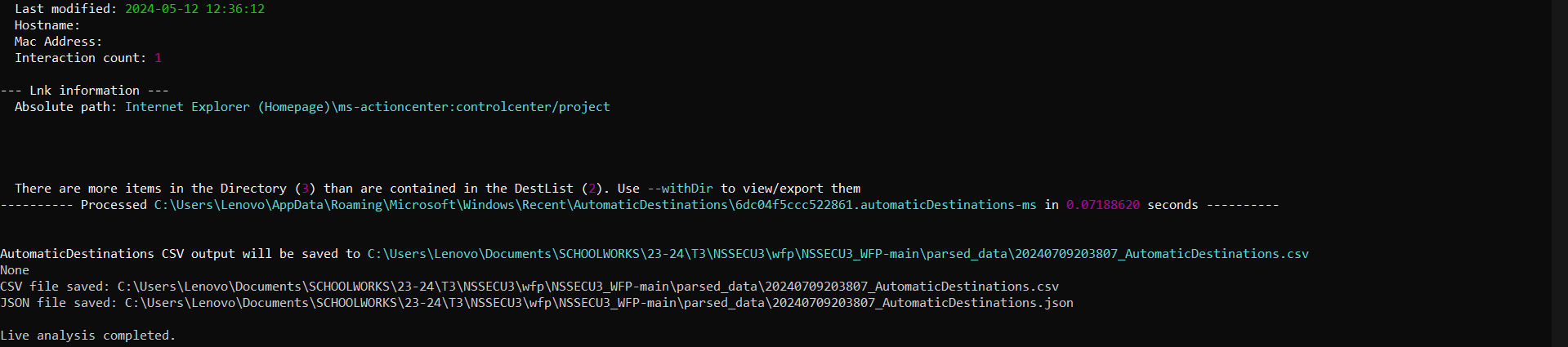


Figure 13. Jump List Parser Live Analysis finished

A screenshot of a computer

Description automatically generated

Figure 14. Folder 'parsed\_data' is created after Jump List Parser live analysis

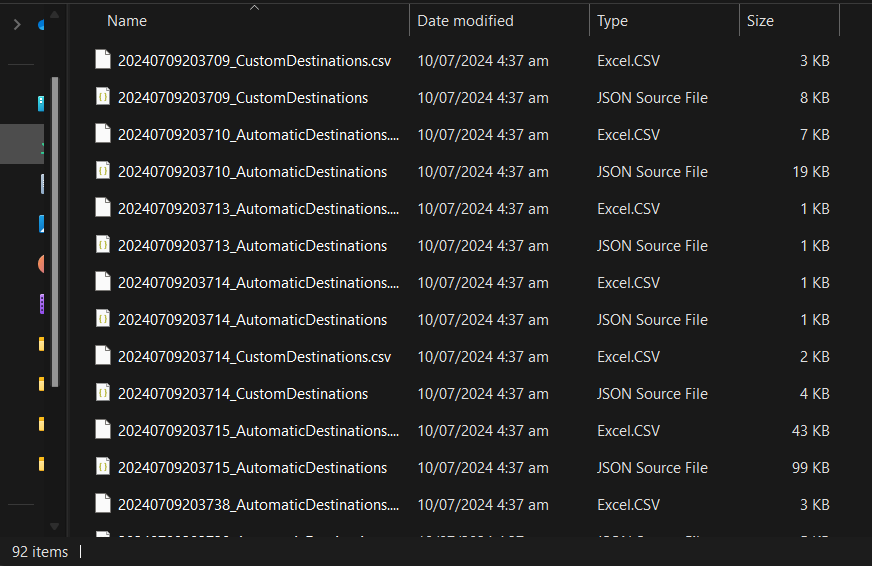


Figure 15. 'parsed\_data' folder (all csv and json files from the Jump List Parser live analysis are located here)

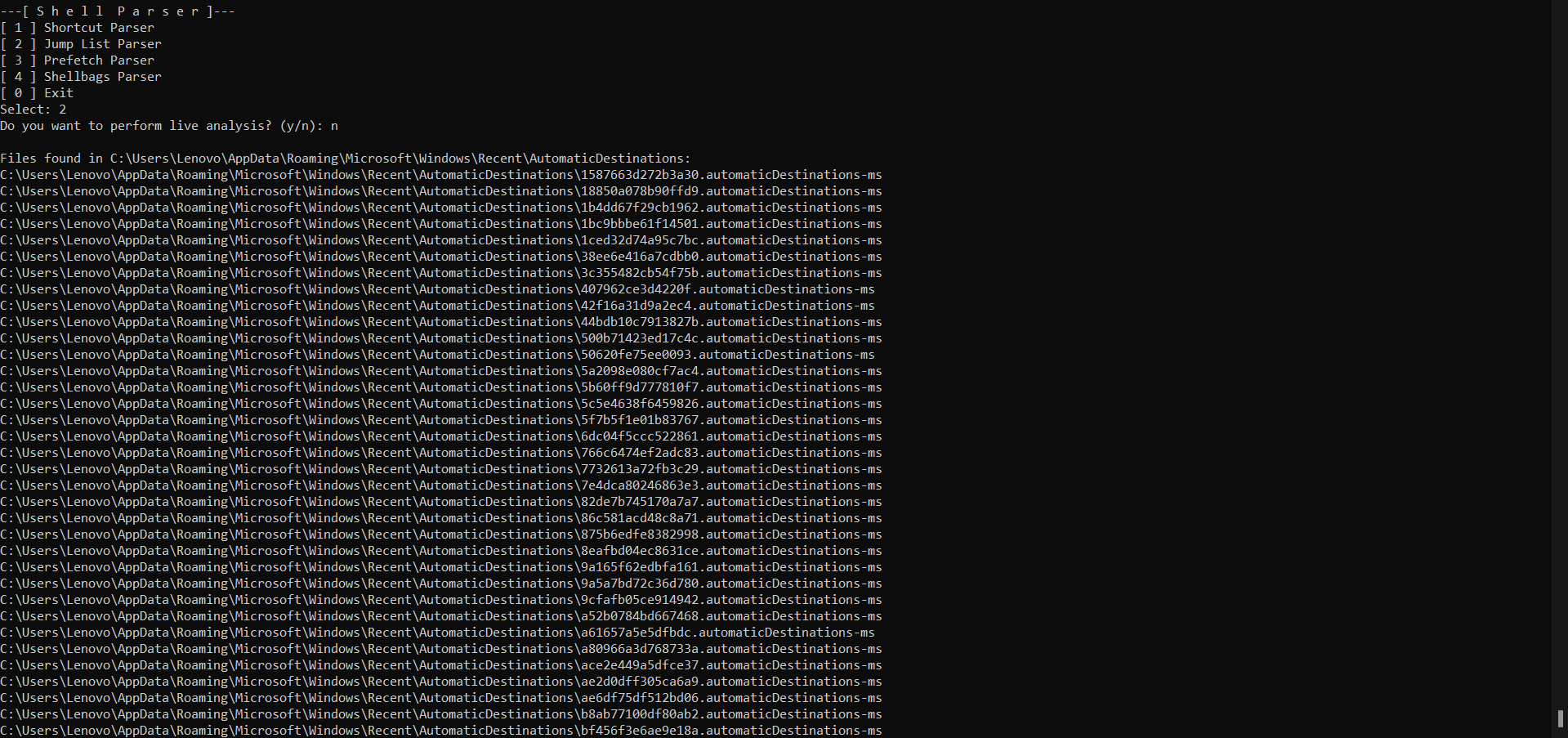


Figure 16. Selecting Jump List Parser and Offline Analysis (a dictionary of Jump List file paths is displayed, categorized by folder)

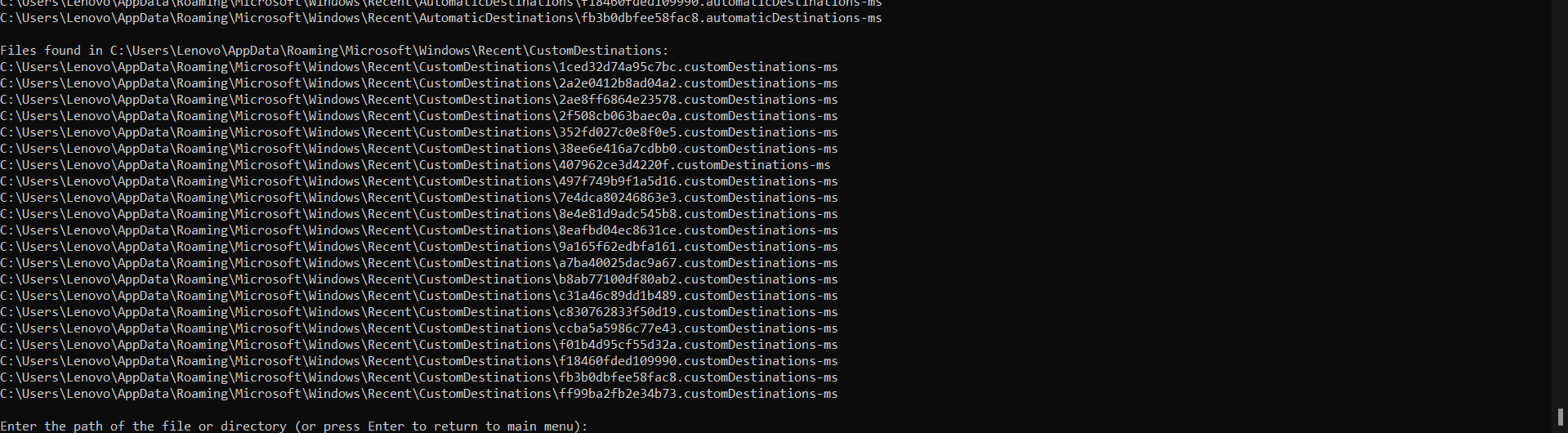


Figure 17. Selecting Jump List Parser and Offline Analysis (snippet of file paths from the dictionary of Jump List file paths)

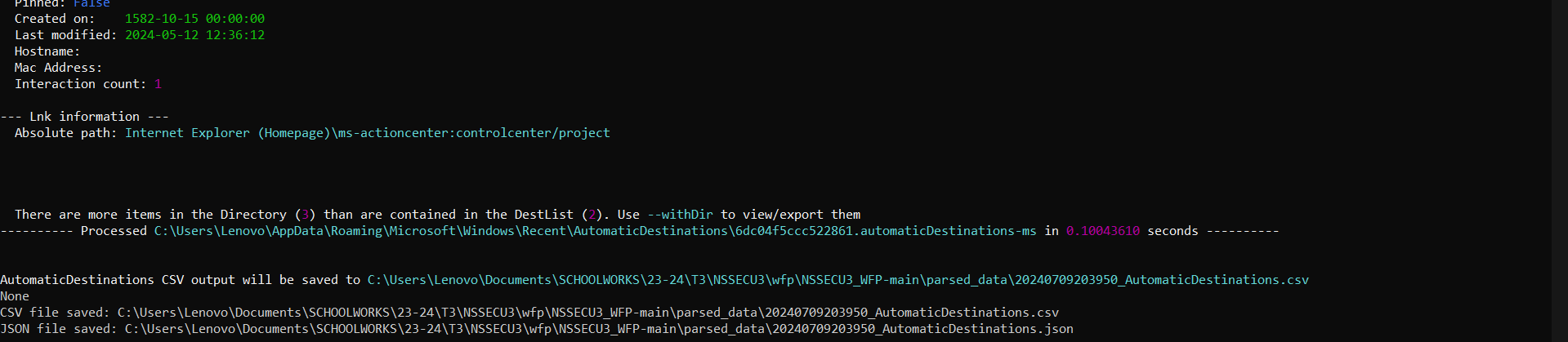


Figure 18. Jump List Parser Offline Analysis finished

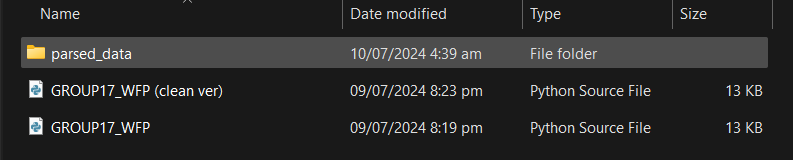


Figure 19. Folder 'parsed\_data' is created after Jump List Parser offline analysis

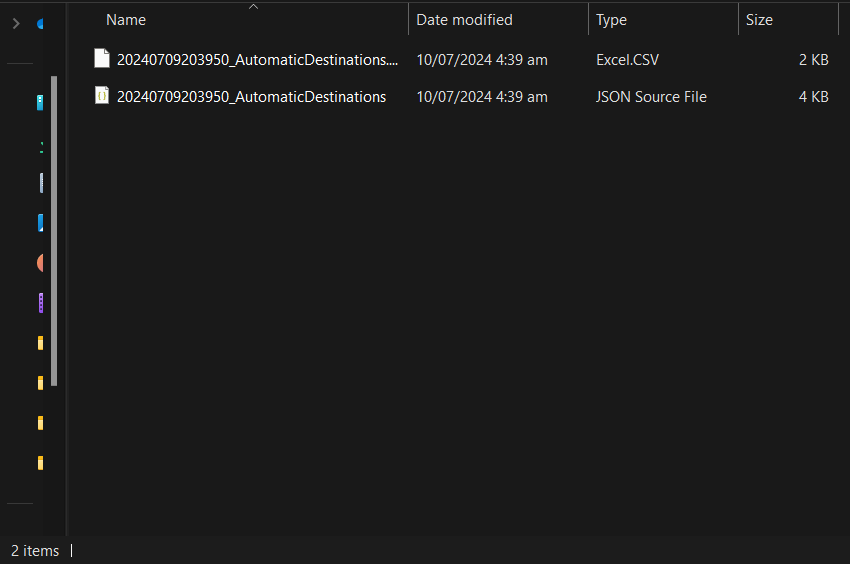


Figure 20. 'parsed\_data' folder (the csv and json files of the selected file from the Jump List Parser offline analysis are located here)

### Prefetch Parser

For the prefetch parser, the program will search for shortcuts in the following directories:

* C:\Windows\Prefetch



Figure 21. Selecting Prefetch Parser and Live Analysis

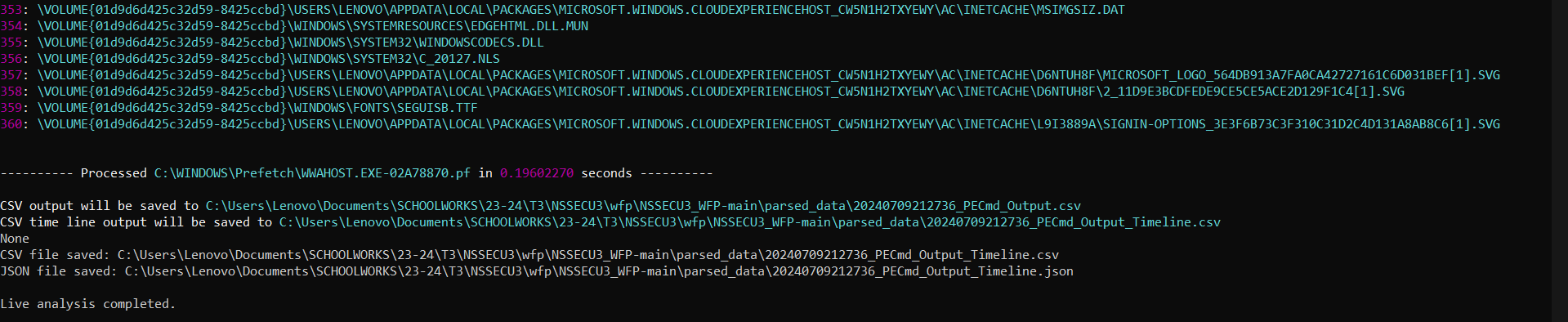


Figure 22. Prefetch Parser Live Analysis finished

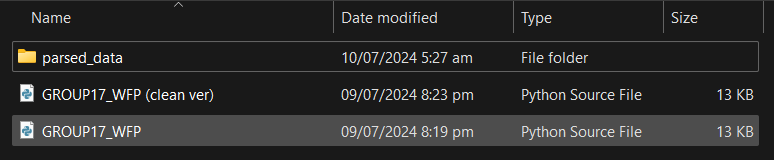


Figure 23. Folder 'parsed\_data' is created after Prefetch Parser live analysis

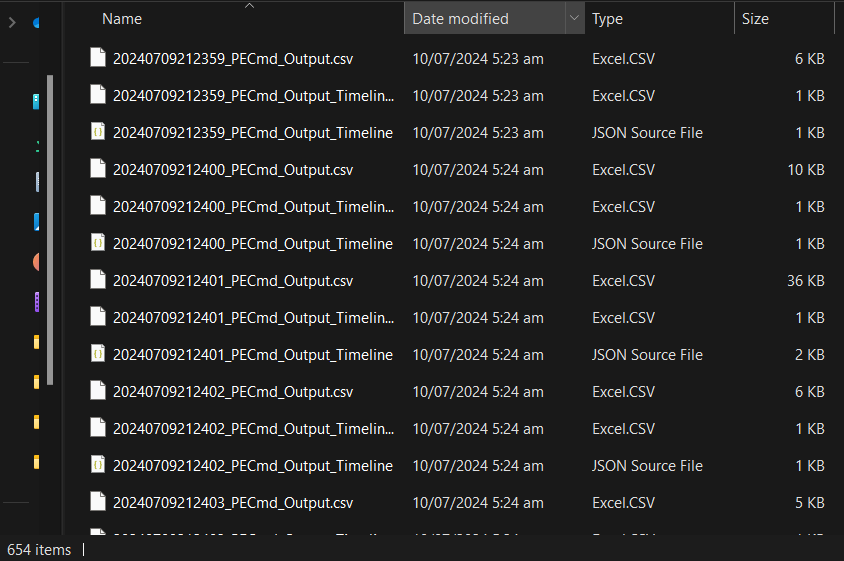


Figure 24. 'parsed\_data' folder (all csv and json files from the Prefetch Parser live analysis are located here)



Figure 25. Selecting Prefetch Parser and Offline Analysis (a dictionary of Prefetch file paths is displayed, categorized by folder)

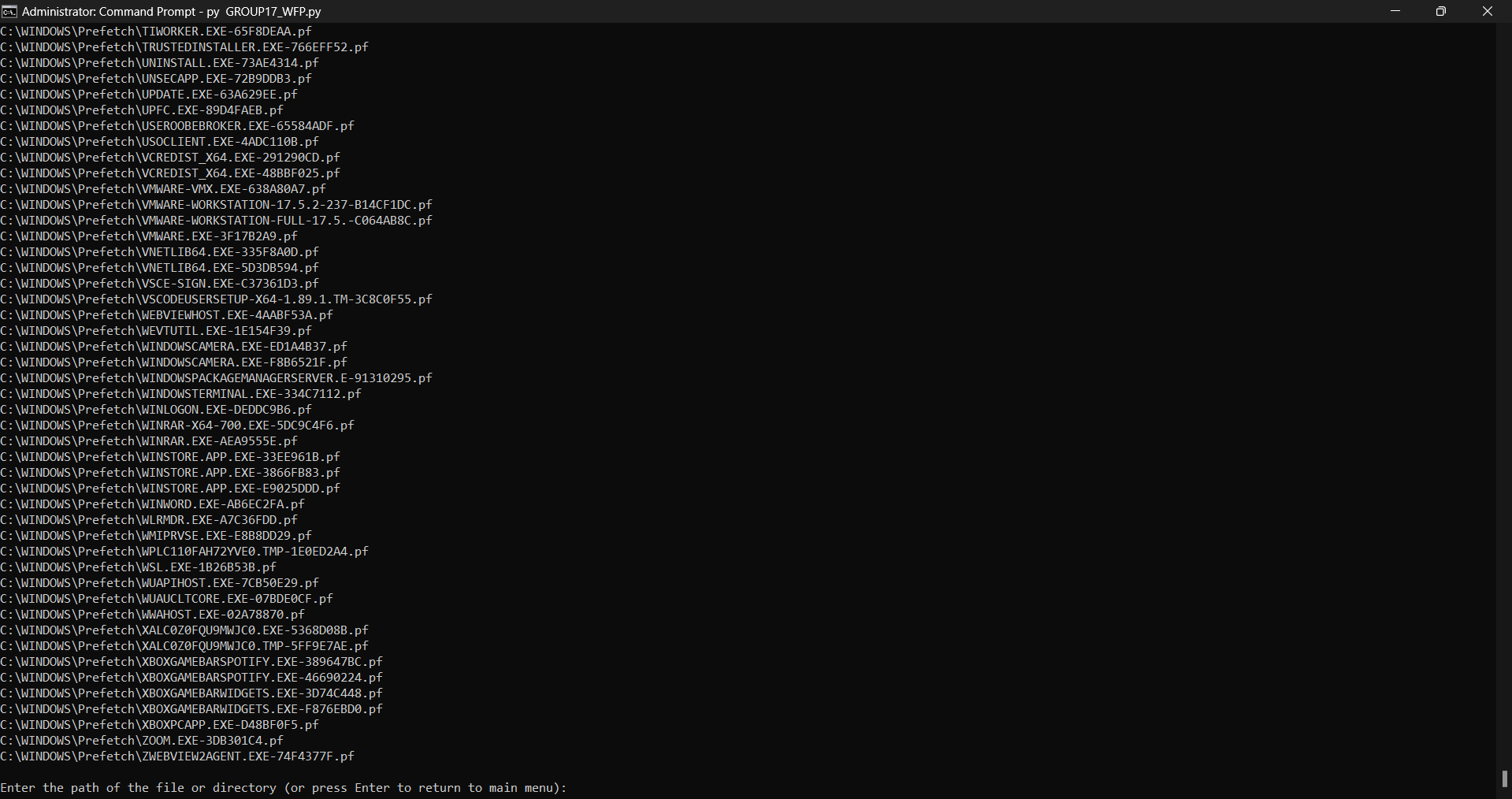


Figure 26. Selecting Prefetch Parser and Offline Analysis (snippet of file paths from the dictionary of Prefetch file paths)

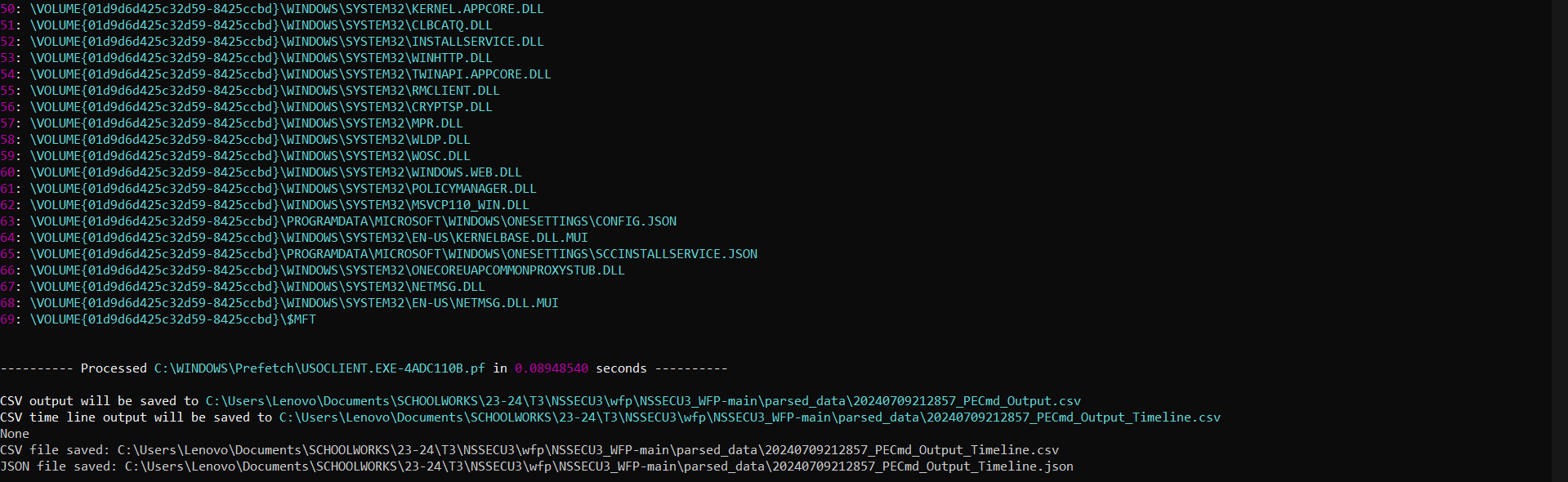


Figure 27. Prefetch Parser Offline Analysis finished

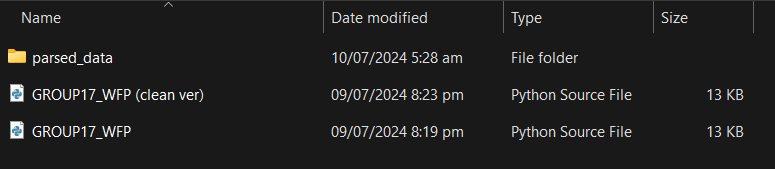


Figure 28. Folder 'parsed\_data' is created after Prefetch Parser offline analysis

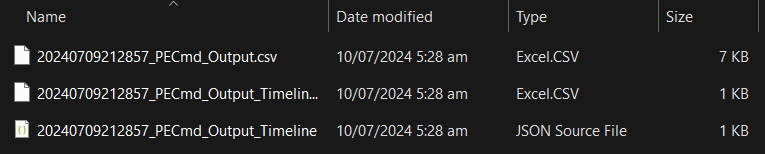


Figure 29. 'parsed\_data' folder (the csv and json files of the selected file from the Prefetch Parser offline analysis are located here)

### Shellbags Parser

For the shellbags parser, the program will search for shortcuts in the following directories:

* C:\Users\Default
* C:\Users\Default User
* C:\Users\<Your Username>

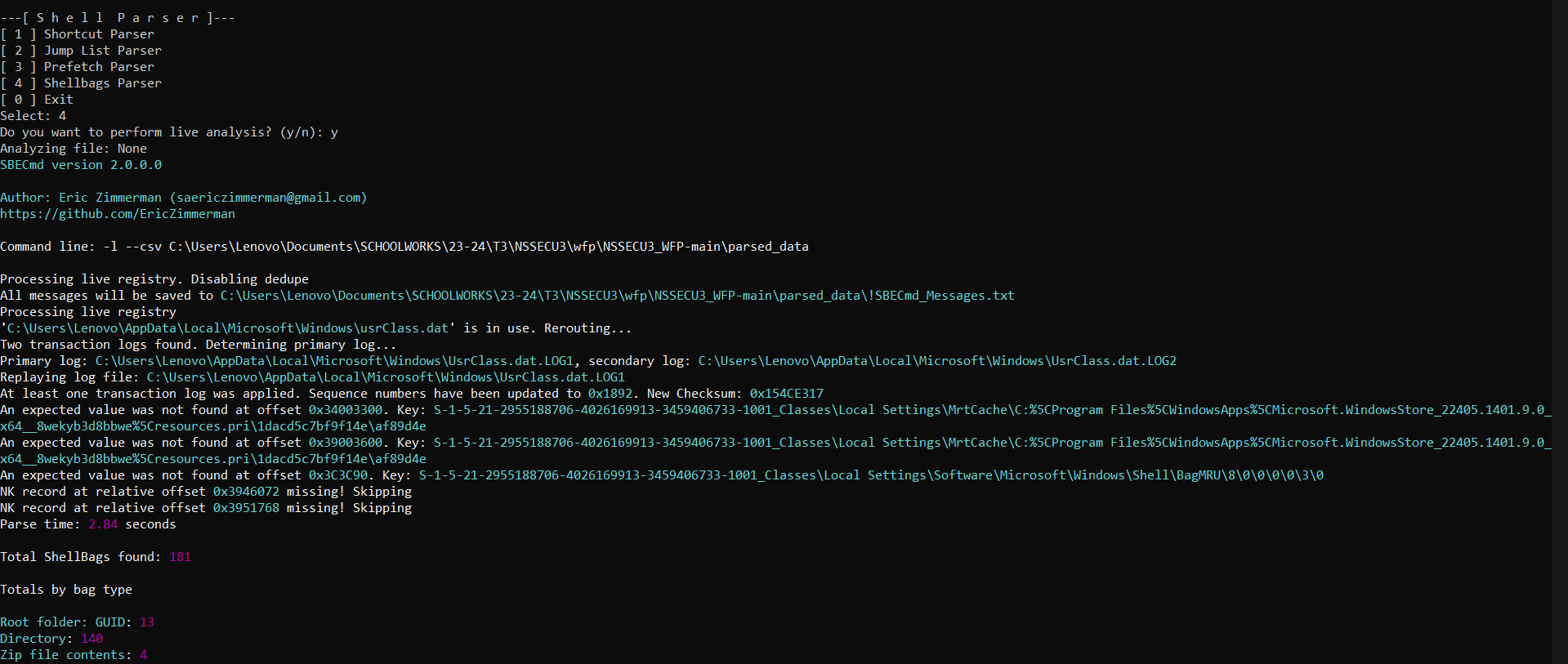


Figure 30. Selecting Shellbags Parser and Live Analysis

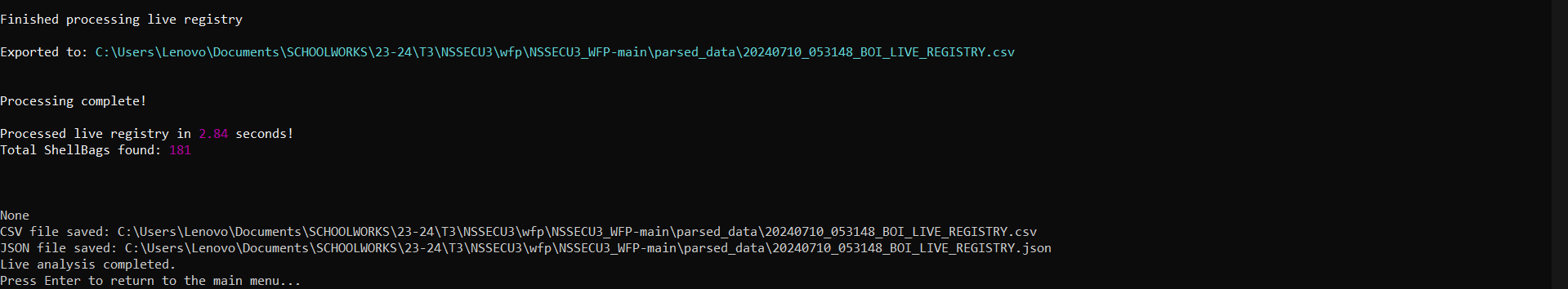


Figure 31. Shellbags Parser Live Analysis finished

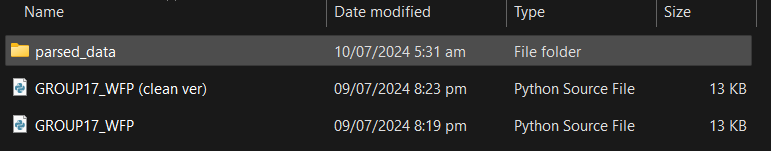


Figure 32. Folder 'parsed\_data' is created after Shellbags Parser live analysis

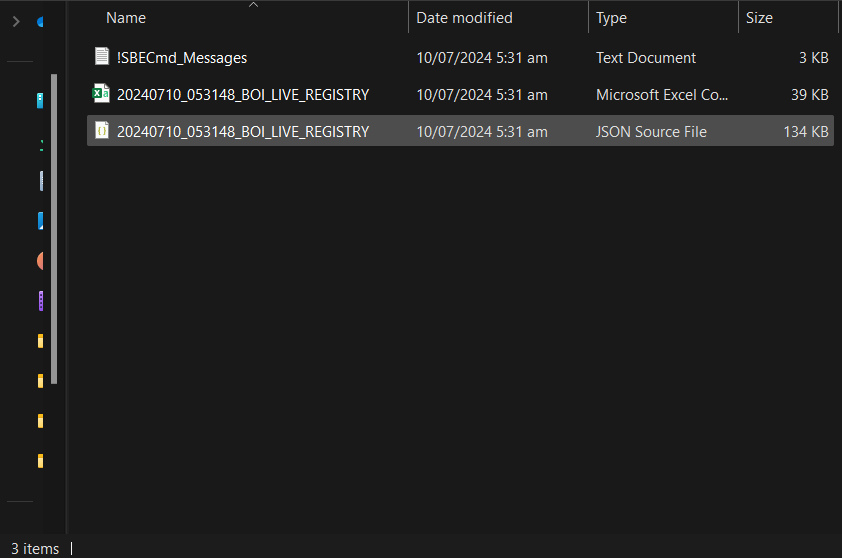


Figure 33. 'parsed\_data' folder (all csv and json files from the Shellbags Parser live analysis are located here)



Figure 34. Selecting Shellbags Parser and Offline Analysis (a dictionary of Shellbags file paths is displayed, categorized by folder)

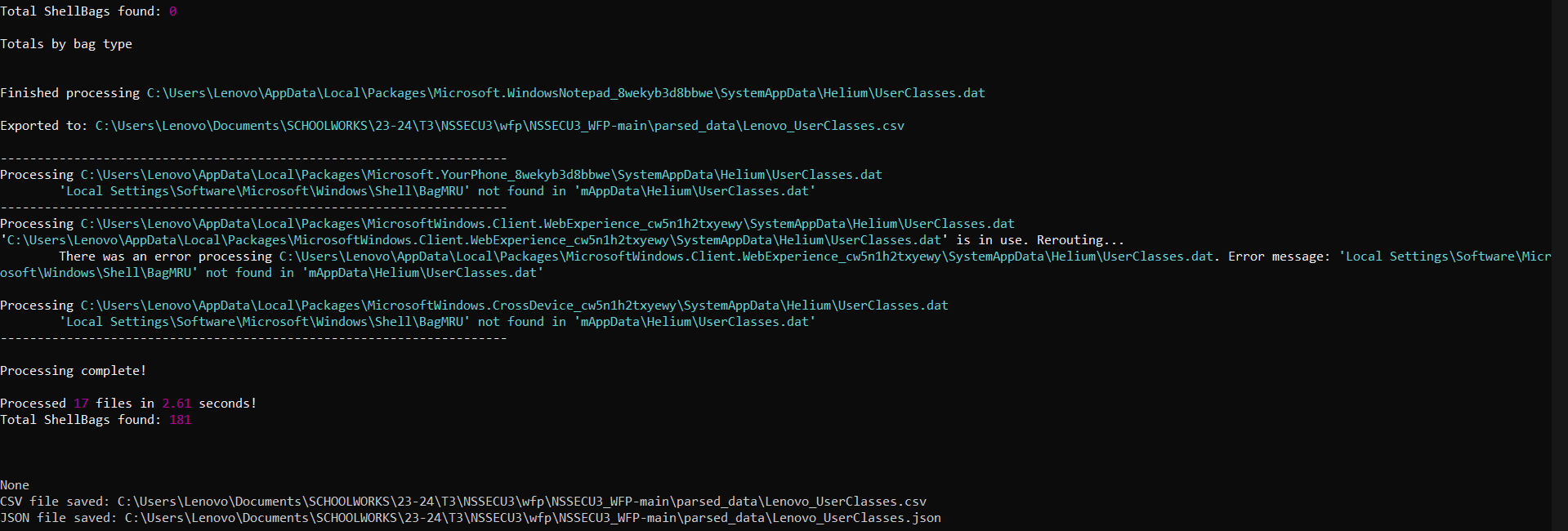


Figure 35. Shellbags Parser Offline Analysis finished

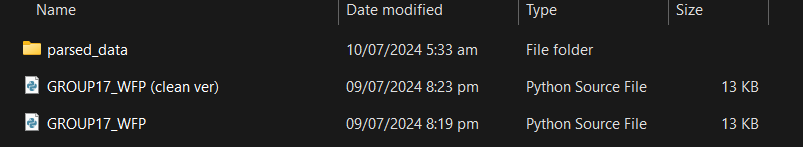


Figure 36. Folder 'parsed\_data' is created after Shellbags Parser offline analysis

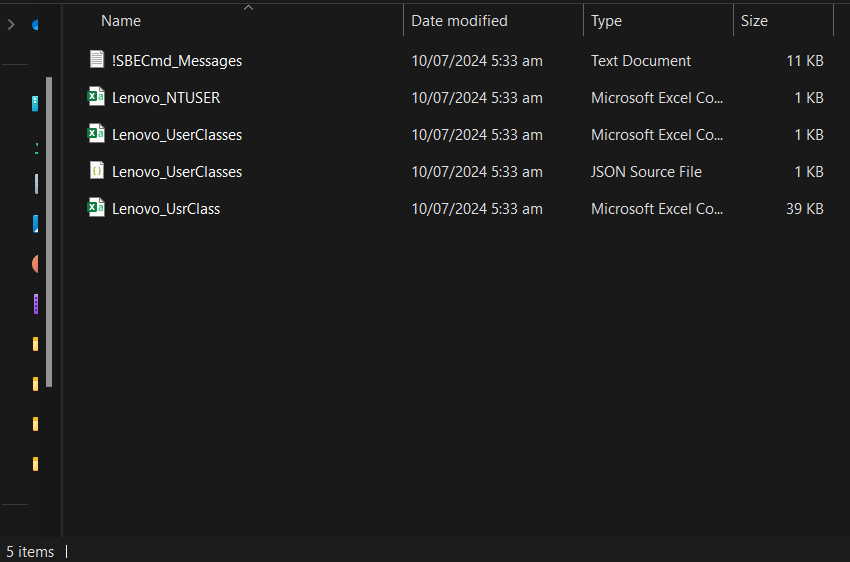


Figure 37. 'parsed\_data' folder (the csv and json files of the selected file from the Shellbags Parser offline analysis are located here)

# Conclusion

The development of this Python-based shell parser tool marks a significant advancement in digital forensic analysis, providing a comprehensive and efficient solution for parsing various Shell Items in Windows. By integrating specialized prebuilt tools for shortcuts, jump lists, prefetch files, and shellbags, this tool offers a unified interface that streamlines the forensic process. This integration ensures high-precision and detailed analysis, leveraging the capabilities of well-regarded tools in the forensic community.

The tool's design prioritizes user experience and efficiency, featuring an intuitive menu interface, automated file discovery, and selection processes. Its ability to convert parsed output from CSV to JSON format facilitates seamless integration with other analytical tools and workflows. Additionally, the inclusion of a live analysis mode, which monitors directories in real time, provides immediate insights into system changes, enhancing the adaptability of the tool to various forensic scenarios.

Compared with traditional single-function parsers, this tool stands out for its versatility and comprehensive approach. This reduces the time and effort required for forensic investigators to switch between different tools and manually handle various file types. Robust error handling mechanisms further ensure smooth operation and user guidance, minimizing the likelihood of errors.

In summary, this shell parser tool exemplifies a well-rounded forensic solution that addresses the complexities and demands of modern digital investigation. Its comprehensive capabilities, user-friendly design, and efficiency improvements make it a superior choice for forensic analysts seeking detailed and reliable insights into systemic activities. This tool not only enhances the accuracy and depth of forensic analysis but also significantly improves the overall efficiency of the investigative process.

# Appendix (Shell Items Parser and Collection Tool Code)

import os

import subprocess

import csv

import json

from datetime import datetime

import time

import sys

def display\_menu():

    """

    Display the main menu of the shell parser program.

    """

    print("\n---[ S h e l l  P a r s e r ]---")

    print("[ 1 ] Shortcut Parser")

    print("[ 2 ] Jump List Parser")

    print("[ 3 ] Prefetch Parser")

    print("[ 4 ] Shellbags Parser")

    print("[ 0 ] Exit")

    choice = input("Select: ")

    return choice

def list\_files(folder\_paths, extension):

    """

    List files with a specific extension from given folders.

    Args:

    - folder\_paths (list): List of folder paths to search.

    - extension (str): File extension to filter by.

    Returns:

    - files (dict): Dictionary of file paths categorized by folder.

    """

    files = {}

    for path in folder\_paths:

        if os.path.exists(path):

            files[path] = []

            for root, \_, filenames in os.walk(path):

                for filename in filenames:

                    if filename.endswith(extension):

                        file\_path = os.path.join(root, filename)

                        files[path].append(file\_path)

        else:

            print(f"Path does not exist: {path}")

    return files

def list\_shortcut\_files():

    """

    List shortcut files (.lnk) from various system locations.

    Returns:

    - files (dict): Dictionary of shortcut file paths categorized by folder.

    """

    desktop\_path = os.path.join(os.environ['USERPROFILE'], 'Desktop')

    start\_menu\_paths = [

        os.path.join(os.environ['PROGRAMDATA'], 'Microsoft', 'Windows', 'Start Menu', 'Programs'),

        os.path.join(os.environ['APPDATA'], 'Microsoft', 'Windows', 'Start Menu', 'Programs')

    ]

    taskbar\_path = os.path.join(os.environ['APPDATA'], 'Microsoft', 'Internet Explorer', 'Quick Launch', 'User Pinned', 'TaskBar')

    recent\_items\_path = os.path.join(os.environ['APPDATA'], 'Microsoft', 'Windows', 'Recent')

    return list\_files([desktop\_path] + start\_menu\_paths + [taskbar\_path, recent\_items\_path], '.lnk')

def list\_jump\_list\_files():

    """

    List jump list files (no extension) from recent items locations.

    Returns:

    - files (dict): Dictionary of jump list file paths categorized by folder.

    """

    jump\_list\_paths = [

        os.path.join(os.environ['APPDATA'], 'Microsoft', 'Windows', 'Recent', 'AutomaticDestinations'),

        os.path.join(os.environ['APPDATA'], 'Microsoft', 'Windows', 'Recent', 'CustomDestinations')

    ]

    return list\_files(jump\_list\_paths, '')

def list\_prefetch\_files():

    """

    List prefetch files (.pf) from the Windows prefetch folder.

    Returns:

    - files (dict): Dictionary of prefetch file paths categorized by folder.

    """

    prefetch\_path = os.path.join(os.environ['SYSTEMROOT'], 'Prefetch')

    return list\_files([prefetch\_path], '.pf')

def list\_shellbags\_files():

    """

    List NTUSER.DAT and UsrClass.dat files from user directories for Shellbags parsing.

    Returns:

    - ntuser\_files (dict): Dictionary of NTUSER.DAT file paths categorized by folder.

    - usrclass\_files (dict): Dictionary of UsrClass.dat file paths categorized by folder.

    """

    users\_dir = os.path.join(os.environ['SYSTEMDRIVE'] + '\\', 'Users')

    ntuser\_files = {}

    usrclass\_files = {}

    for user\_folder in os.listdir(users\_dir):

        user\_path = os.path.join(users\_dir, user\_folder)

        if os.path.isdir(user\_path):

            ntuser\_path = os.path.join(user\_path, 'NTUSER.DAT')

            if os.path.exists(ntuser\_path):

                ntuser\_files[user\_folder] = ntuser\_files.get(user\_folder, []) + [ntuser\_path]

            usrclass\_path = os.path.join(user\_path, 'AppData', 'Local', 'Microsoft', 'Windows', 'UsrClass.dat')

            if os.path.exists(usrclass\_path):

                usrclass\_files[user\_folder] = usrclass\_files.get(user\_folder, []) + [usrclass\_path]

    return ntuser\_files, usrclass\_files

def display\_files(files):

    """

    Display list of files found, categorized by their directories.

    Args:

    - files (dict): Dictionary of file paths categorized by folder.

    """

    for folder, file\_list in files.items():

        print(f"\nFiles found in {folder}:")

        for file in file\_list:

            print(file)

def parse\_file(tool\_path, file\_path, is\_shellbags=False, is\_live=False):

    """

    Parse a file using the specified tool and save results to CSV and JSON.

    Args:

    - tool\_path (str): Path to the parsing tool executable.

    - file\_path (str): Path to the file to be parsed.

    - is\_shellbags (bool): Flag indicating if parsing Shellbags.

    - is\_live (bool): Flag indicating if live analysis mode.

    """

    output\_dir = os.path.join(os.path.dirname(\_\_file\_\_), 'parsed\_data')

    os.makedirs(output\_dir, exist\_ok=True)

    while True:

        try:

            if is\_shellbags:

                if is\_live:

                    command = [tool\_path, '-l', '--csv', output\_dir]

                else:

                    command = [tool\_path, '-d', os.path.dirname(file\_path), '--csv', output\_dir]

            else:

                command = [tool\_path, '-f', file\_path, '--csv', output\_dir]

            print(f"Analyzing file: {file\_path}")

            result = subprocess.run(command, check=True, text=True)

            print(result.stdout)

            csv\_file\_path = find\_latest\_csv(output\_dir)

            if csv\_file\_path:

                print(f"CSV file saved: {csv\_file\_path}")

                convert\_csv\_to\_json(csv\_file\_path)

            else:

                print("No CSV file generated.")

            break  # Exit the loop if successful

        except FileNotFoundError:

            if not is\_live:

                print(f"File not found: {file\_path}")

                file\_path = input("Please enter a valid file path or press Enter to cancel: ").strip()

                if not file\_path:

                    break  # Cancel if user presses Enter without input

            else:

                raise  # Reraise the error during live analysis

        except subprocess.CalledProcessError as e:

            print(f"An error occurred while parsing the file: {e}")

            break  # Exit the loop on other subprocess errors

def find\_latest\_csv(directory):

    """

    Find the latest CSV file in a directory.

    Args:

    - directory (str): Directory to search for CSV files.

    Returns:

    - latest\_file (str): Path to the latest CSV file found.

    """

    csv\_files = [os.path.join(directory, f) for f in os.listdir(directory) if f.endswith('.csv')]

    if not csv\_files:

        return None

    latest\_file = max(csv\_files, key=os.path.getctime)

    return latest\_file

def convert\_csv\_to\_json(csv\_file\_path):

    """

    Convert a CSV file to JSON format and save.

    Args:

    - csv\_file\_path (str): Path to the CSV file.

    """

    json\_file\_path = csv\_file\_path.replace('.csv', '.json')

    try:

        with open(csv\_file\_path, 'r', encoding='utf-8') as csv\_file:

            reader = csv.DictReader(csv\_file)

            rows = list(reader)

        with open(json\_file\_path, 'w', encoding='utf-8') as json\_file:

            json.dump(rows, json\_file, indent=4)

        print(f"JSON file saved: {json\_file\_path}")

    except Exception as e:

        print(f"An error occurred while converting CSV to JSON: {e}")

def live\_analysis(tool\_path, directories, extension, duration=60):

    """

    Perform live analysis of files using the specified tool for a fixed duration.

    Args:

    - tool\_path (str): Path to the parsing tool executable.

    - directories (list): List of directories to monitor.

    - extension (str): File extension filter.

    - duration (int): Duration to run the live analysis (in seconds).

    """

    seen\_files = set()

    end\_time = time.time() + duration

    try:

        while time.time() < end\_time:

            current\_files = set()

            for folder, file\_list in list\_files(directories, extension).items():

                current\_files.update(file\_list)

            new\_files = current\_files - seen\_files

            for file\_path in new\_files:

                try:

                    parse\_file(tool\_path, file\_path, is\_shellbags=False, is\_live=True)

                except FileNotFoundError:

                    print(f"File not found during live analysis: {file\_path}")

                except Exception as e:

                    print(f"An error occurred during live analysis: {e}")

            seen\_files = current\_files

            time.sleep(10)  # Adjust the sleep time as needed

    except KeyboardInterrupt:

        pass  # Handle Ctrl+C to exit

    print("\nLive analysis completed.")

def get\_live\_choice():

    """

    Prompt user to choose whether to perform live analysis.

    Returns:

    - is\_live (bool): True if user chooses live analysis, False otherwise.

    """

    while True:

        choice = input("Do you want to perform live analysis? (y/n): ").strip().lower()

        if choice in {'y', 'n'}:

            return choice == 'y'

        print("Invalid choice. Please enter 'y' or 'n'.")

def main():

    """

    Main function to run the shell parser program.

    """

    tool\_paths = {

        '1': "C:\\Path\\To\\LECmd.exe", # shortcut parser (LECmd)

        '2': "C:\\Path\\To\\JLECmd.exe", # jump list parser (JLECmd)

        '3': "C:\\Path\\To\\PECmd.exe", # prefetch parser (PECmd)

        '4': "C:\\Path\\To\\SBECmd.exe" #shellbags parser (SBECmd)

    } # make sure to change the path for each parser tools (use double backslash "\\" to separate directories in a file path)

    list\_functions = {

        '1': list\_shortcut\_files,

        '2': list\_jump\_list\_files,

        '3': list\_prefetch\_files,

        '4': list\_shellbags\_files

    }

    directories = {

        '1': [

            os.path.join(os.environ['USERPROFILE'], 'Desktop'),

            os.path.join(os.environ['PROGRAMDATA'], 'Microsoft', 'Windows', 'Start Menu', 'Programs'),

            os.path.join(os.environ['APPDATA'], 'Microsoft', 'Windows', 'Start Menu', 'Programs'),

            os.path.join(os.environ['APPDATA'], 'Microsoft', 'Internet Explorer', 'Quick Launch', 'User Pinned', 'TaskBar'),

            os.path.join(os.environ['APPDATA'], 'Microsoft', 'Windows', 'Recent')

        ],

        '2': [

            os.path.join(os.environ['APPDATA'], 'Microsoft', 'Windows', 'Recent', 'AutomaticDestinations'),

            os.path.join(os.environ['APPDATA'], 'Microsoft', 'Windows', 'Recent', 'CustomDestinations')

        ],

        '3': [

            os.path.join(os.environ['SYSTEMROOT'], 'Prefetch')

        ]

    }

    extensions = {

        '1': '.lnk',

        '2': '',

        '3': '.pf'

    }

    while True:

        choice = display\_menu()

        if choice in tool\_paths:

            is\_live = get\_live\_choice()

            if is\_live:

                if choice == '4':

                    try:

                        parse\_file(tool\_paths[choice], None, is\_shellbags=True, is\_live=True)

                        print("Live analysis completed.")

                        input("Press Enter to return to the main menu...")

                    except Exception as e:

                        print(f"An error occurred during live analysis: {e}")

                        input("Press Enter to return to the main menu...")

                else:

                    try:

                        live\_analysis(tool\_paths[choice], directories[choice], extensions[choice])

                    except Exception as e:

                        print(f"An error occurred during live analysis: {e}")

                        input("Press Enter to return to the main menu...")

            else:

                if choice == '4':

                    ntuser\_files, usrclass\_files = list\_shellbags\_files()

                    print("\nNTUSER.DAT files found:")

                    display\_files(ntuser\_files)

                    print("\nUsrClass.dat files found:")

                    display\_files(usrclass\_files)

                else:

                    files = list\_functions[choice]()

                    display\_files(files)

                while True:

                    file\_path = input("\nEnter the path of the file or directory (or press Enter to return to main menu): ").strip()

                    if not file\_path:

                        break  # Exit if user presses Enter without input

                    if not os.path.exists(file\_path):

                        print("File or directory does not exist.")

                    else:

                        parse\_file(tool\_paths[choice], file\_path, is\_shellbags=(choice == '4'), is\_live=False)

                        break

        elif choice == '0':

            break

        else:

            print("Invalid selection. Please choose a valid option.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# References

**There are no sources in the current document.**