National University of Computer and Emerging Sciences
Islamabad Campus

CY2002

Digital Forensics

MemSieve
(Password Parser)
User Guide

Submitted by:

Laaibah Qayyum	22i1604
Ali Imran	22i1568
Areej Zeb	22i1561
Ahmed Mustafa	22i1591

Date: 17 November 2024

Table of Contents

Overview

The tool we have developed is known as the **MemSieve (Password Parser)**, designed to facilitate the extraction and analysis of sensitive information like passwords for forensic and cybersecurity purposes. This tool is versatile and comprises three main modules that cater to various use cases:

Password Extraction from Live RAM:

This module enables real-time analysis of volatile memory to retrieve passwords. It provides an efficient way to capture data directly from the running system, which can be critical in incident response or forensic investigations when time-sensitive information is required.

Password Retrieval from Memory Dumps:

This module focuses on analyzing previously captured memory dumps. It parses these files to extract stored passwords, offering an offline analysis capability. This feature is valuable for post-incident reviews and allows detailed investigations into memory artifacts.

Integration with PAMSpy for Enhanced Analysis:

The tool also includes an additional feature leveraging **PAMSpy**, which extends its capabilities to monitor and extract authentication details within Linux systems. PAMSpy allows Password Parser to interact with Linux's Pluggable Authentication Modules (PAM) to capture credentials effectively, adding a specialized dimension for Linux environments.

Target Audience

The Password Parser tool is designed for professionals and individuals involved in cybersecurity, digital forensics, and incident response. The key target audience includes:

Forensic Analysts:

- Professionals analyzing digital evidence in criminal investigations.
- Those requiring tools to extract passwords from volatile or non-volatile memory for forensic purposes.

Penetration Testers and Ethical Hackers:

- Individuals testing the security of systems and identifying vulnerabilities involving password storage and memory management.

Academicians and Researchers:

- Students and researchers studying memory forensics and security vulnerabilities in authentication mechanisms.

Law Enforcement Agencies:

- Teams working on cybercrime cases requiring tools to recover critical authentication data.

System Requirement:

- Operating Systems Supported : Kali Linux

Software Requirements

To set up and use the Password Parser tool effectively, the following software is required:

Development Tools:

- Build tools like `build-essential`.
- Git for cloning repositories.

Memory Analysis Tools:

- Volatility framework (requires setup).

Python Environment:

- Python 2.7 for compatibility with Volatility and certain libraries.
- Pip for Python 2.7 to manage dependencies.

Libraries and Dependencies:

- distorm3 - for disassembly support.
- Yara - for malware pattern matching.
- pycrypto - for cryptographic functions.
- pillow - for image handling.

System Libraries:

libdistorm3-dev, libcapstone-dev, libraw1394-11, and tzdata for system compatibility.

Installation commands

Run the command below in order for setting up your system.

- sudo apt install -y build-essential git libdistorm3-dev yara libraw1394-11 libcapstone-dev capstone-tool tzdata
- sudo apt install -y python2 python2.7-dev libpython2-dev
- curl https://bootstrap.pypa.io/pip/2.7/get-pip.py --output get-pip.py

- sudo python2 get-pip.py
- sudo python2 -m pip install -U setuptools wheel
- python2 -m pip install -U distorm3 yara pycrypto pillow openpyxl ujson pytz ipython capstone
- sudo python2 -m pip install yara
- sudo ln -s /usr/local/lib/python2.7/dist-packages/usr/lib/libyara.so /usr/lib/libyara.so
- git clone https://github.com/volatilityfoundation/volatility.git
- cd volatility
- sudo python2 setup.py install

For pamspy module

- openssh-server
- sudo systemctl enable --now ssh
- sudo systemctl status ssh

Setup of Files:

Download the Repository

git clone https://github.com/Laaaibah/MemSieve.git

Unzip the Files

- Extract the downloaded archive (e.g., .zip, .tar.gz) using a suitable command:
- unzip [filename]
- tar -xvf [filename]

Organize Files

- Move the extracted files into the **Volatility** folder for seamless integration

Navigate to Volatility Folder

- Open a terminal and change the directory to the **Volatility** folder:
- cd /path/to/volatility

Flow of Program

- Execute the required commands in the terminal while in the Volatility folder for proper setup and usage of the tool.
- Use the `g++` compiler to compile your C++ source file into an executable: g++ -o project project.cpp

- Make the compiled executable file runnable by assigning execute permissions: chmod +x project

```
┌──(kali㉿kali)-[~/volatility]
└─$ g++ -o project project.cpp

┌──(kali㉿kali)-[~/volatility]
└─$ chmod +x project
```

- After compiling and setting the necessary permissions, execute the program by running: ./project

```
┌──(kali㉿kali)-[~/volatility]
└─$ ./project
*******************************************
*        Welcome to the Password Parser        *
*          Choose an option below:             *
*******************************************

Do you want to fetch the password from?
1. Memory Dump
2. Live RAM Memory
3. Pamspy: Credentials Dumper for Linux (EXTRA)
Enter your choice:
```

Memory Dump Password Retrieval:

Step 0: Running hashdump and lsadump

- Extract password hashes: Runs the `hashdump` plugin to retrieve password hashes from memory, saving the output in `hashdump_output.txt` and displaying it on the console.
- Extract LSA secrets: Runs the `lsadump` plugin to retrieve LSA (Local Security Authority) secrets, saving the output in `lsadump_output.txt` and displaying it on the console.

```
Memory Analysis Script - CTF Tools

Step 0: Running hashdump and lsadump ...
Running hashdump to gather password hashes ...
Volatility Foundation Volatility Framework 2.6.1
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Rick:1000:aad3b435b51404eeaad3b435b51404ee:518172d012f97d3a8fcc089615283940:::
Hashdump completed successfully. Output saved to ./hashdump_output.txt
Running lsadump to gather LSA secrets ...
Volatility Foundation Volatility Framework 2.6.1
DefaultPassword
0x00000000  28 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   (...............
0x00000010  4d 00 6f 00 72 00 74 00 79 00 49 00 73 00 52 00   M.o.r.t.y.I.s.R.
0x00000020  65 00 61 00 6c 00 6c 00 79 00 41 00 6e 00 4f 00   e.a.l.l.y.A.n.O.
0x00000030  74 00 74 00 65 00 72 00 00 00 00 00 00 00 00 00   t.t.e.r.........

DPAPI_SYSTEM
0x00000000  2c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ,...............
0x00000010  01 00 00 00 36 9b ba a9 55 e1 92 82 09 e0 63 4c   ....6...U.....cL
0x00000020  20 74 63 14 9e d8 a0 4b 45 87 5a e4 bc f2 77 a5   .tc....KE.Z...w.
0x00000030  25 3f 47 12 0b e5 4d a5 c8 35 cf dc 00 00 00 00   %?G...M..5......
Lsadump completed successfully. Output saved to ./lsadump_output.txt
Waiting for 5 seconds ...
```

```
┌──(kali㉿kali)-[~/volatility]
└─$ hash-identifier
/usr/share/hash-identifier/hash-id.py:13: SyntaxWarning: invalid escape sequence '\ '
  logo='''   ##########################################################################
   ######################################################################
   #     __  __                     __           ______    _____         #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\       #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \      #
   #     \ \  _  \  /'__`\   /',__\ \ \  _ `\      \ \ \   \ \ \ \ \ \    #
   #      \ \ \ \ \/\ \L\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \    #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/     #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.2 #
   #                                                            By Zion3R #
   #                                                     www.Blackploit.com #
   #                                                    Root@Blackploit.com #
   ######################################################################
--------------------------------------------------
 HASH: aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0

Possible Hashs:
[+] md5($pass.$salt) - Joomla
```

Step 1: Running pslist to gather processes

- Executes the pslist Volatility plugin to extract a list of running processes from the memory dump.
- Saves the output to a file for analysis.

```
Memory Analysis Script - CTF Tools

Step 1: Running pslist to gather processes ...
Volatility Foundation Volatility Framework 2.6.1
```

Step 2: Searching for target processes

- Filters the `pslist` output to find specific processes (e.g., `lsass.exe`, `chrome.exe`) by name.
- Extracts the Process IDs (PIDs) of these target processes for further analysis.

```
Memory Analysis Script - CTF Tools

Step 2: Searching for target processes ...
Found lsass.exe with PID 500.
Found chrome.exe with PID 4076.
Found svchost.exe with PID 604.
Waiting for 5 seconds ...
```

Step 3: Creating memory dumps

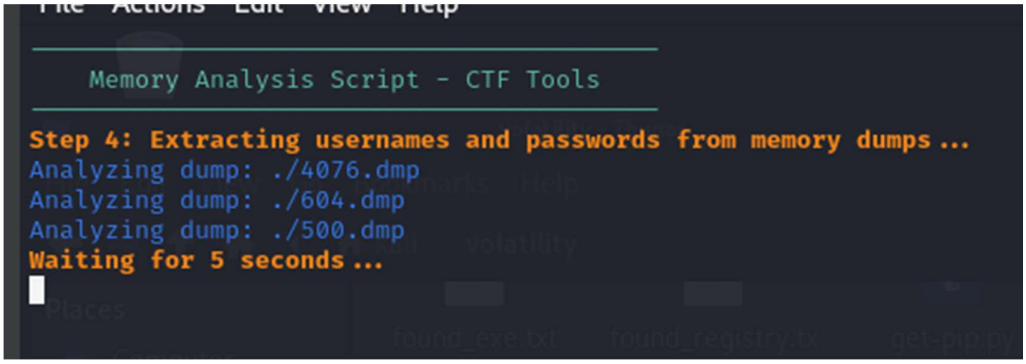
- Uses the extracted PIDs to create memory dumps of the target processes using the Volatility `memdump` plugin.
- Saves each dump file in the specified directory.

```
Memory Analysis Script - CTF Tools

Step 3: Creating memory dumps ...
Volatility Foundation Volatility Framework 2.6.1
**********************************************************************
Writing chrome.exe [  4076] to 4076.dmp
Memory dump created: ./4076.dmp
Volatility Foundation Volatility Framework 2.6.1
**********************************************************************
Writing svchost.exe [   604] to 604.dmp
Memory dump created: ./604.dmp
Volatility Foundation Volatility Framework 2.6.1
**********************************************************************
Writing lsass.exe [   500] to 500.dmp
Memory dump created: ./500.dmp
Waiting for 5 seconds ...
```

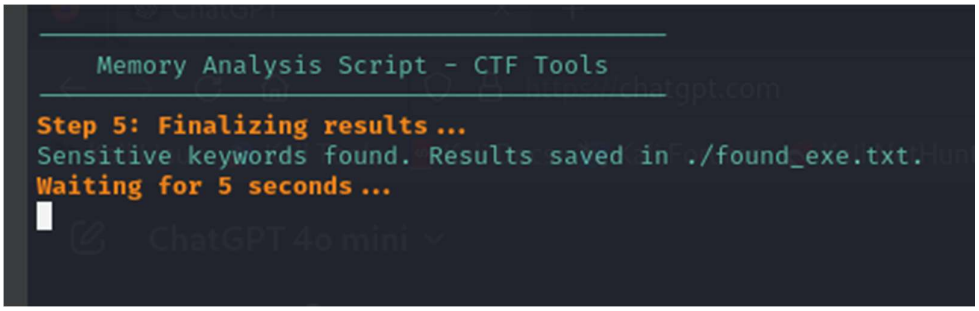
Step 4: Extracting usernames and passwords from memory dumps

- Analyzes the memory dumps of the target processes.
- Searches for strings matching sensitive keywords (e.g., `username`, `password`) and saves the results to a file.



Step 5: Finalizing results

- Checks if any sensitive keywords were found in the memory dumps.
- Displays the status and saves the results to an output file.



Step 6: Dumping registry

- Executes the Volatility `dumpregistry` plugin to extract registry files from the memory dump.
- Stores the registry files in the designated directory.

```
Memory Analysis Script - CTF Tools
Step 6: Dumping registry...
Volatility Foundation Volatility Framework 2.6.1
**************************************************
Writing out registry: registry.0xfffff8a000053320.HARDWARE.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a00000f010.no_name.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a000109410.SECURITY.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a00033d410.BCD.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a00175b010.NTUSERDAT.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a000024010.SYSTEM.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a00377d2d0.Syscachehve.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a001495010.DEFAULT.reg

**************************************************
**************************************************
Writing out registry: registry.0xfffff8a0005d5010.SOFTWARE.reg
```

Step 7: Extracting strings from registry files

- Reads the dumped registry files and extracts human-readable strings.
- Saves the extracted strings to an output file for further keyword searches

```
Memory Analysis Script - CTF Tools

Step 7: Extracting strings from registry files ...
Extracted strings from /tmp/registry_dump/registry.0xfffff8a00000f010.no_name.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a000024010.SYSTEM.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a000053320.HARDWARE.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a000109410.SECURITY.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a00033d410.BCD.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a0005d5010.SOFTWARE.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a001495010.DEFAULT.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a0016d4010.SAM.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a00175b010.NTUSERDAT.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a00176e410.NTUSERDAT.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a002090010.ntuserdat.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a0020ad410.UsrClassdat.reg.
Extracted strings from /tmp/registry_dump/registry.0xfffff8a00377d2d0.Syscachehve.reg.
Waiting for 5 seconds ...
```

Step 8: Searching for sensitive keywords in registry strings

- Searches for sensitive keywords (e.g., `username`, `password`) in the strings extracted from the registry files.
- Saves any matches to a separate output file for review.

```
Memory Analysis Script - CTF Tools

Step 8: Searching for sensitive keywords in registry strings ...
Waiting for 5 seconds ...
```

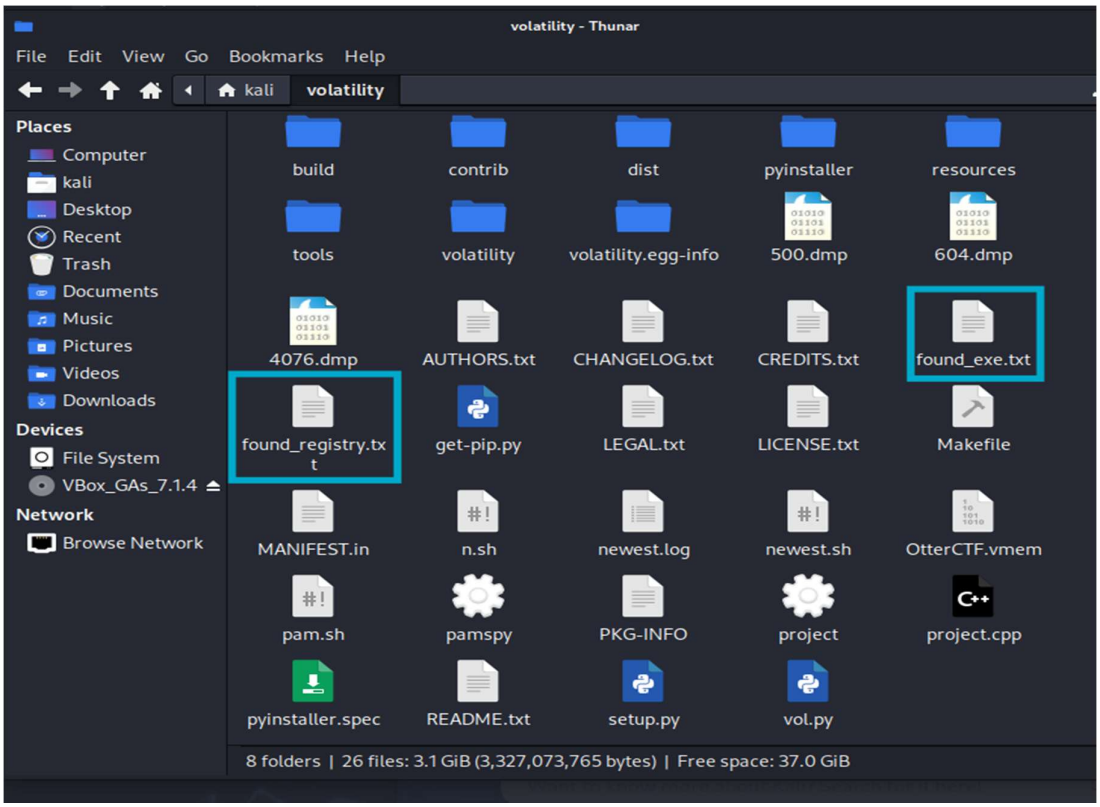
Step 9: View a registry file?

- Prompts the user to choose whether they want to view a specific registry file from the dump.
- If the user chooses "yes," the script asks which file to view and how to display it (either in hex format or human-readable strings). If the user chooses "no," the script exits.

```
Memory Analysis Script - CTF Tools

All tasks completed. Results saved in ./found_registry.txt.
Step 9: View a registry file?
Do you want to view a registry file? (y/n)
y
Choose a registry file to view from the list below:
registry.0xfffff8a000053320.HARDWARE.reg
registry.0xfffff8a00000f010.no_name.reg
registry.0xfffff8a000109410.SECURITY.reg
registry.0xfffff8a00175b010.NTUSERDAT.reg
registry.0xfffff8a000024010.SYSTEM.reg
registry.0xfffff8a0005d5010.SOFTWARE.reg
registry.0xfffff8a0020ad410.UsrClassdat.reg
registry.0xfffff8a0016d4010.SAM.reg
registry.0xfffff8a00176e410.NTUSERDAT.reg
registry.0xfffff8a002090010.ntuserdat.reg
Enter the registry file name: registry.0xfffff8a002090010.ntuserdat.reg
How would you like to view the file?
1. Hex format
2. Human-readable format
Enter 1 for Hex or 2 for Human-readable:
```

The username and passwords fetched from the registry are extracted to found_registry.txt and the username and passwords fetched from exe files are extracted to found_exe.txt



Live RAM Password Extraction:

- Our tool supports extracting sensitive information from live memory, including:

  User Passwords:

  Retrieves passwords of users stored in memory.

  Saved Passwords:

  Extracts saved passwords from applications like Firefox and Thunderbird (mail client).
- This tool efficiently analyzes live RAM to uncover critical data for forensic purposes.

```
The Password Parser Project

Shadow passwords

Username: ahmed
Password Hash: $y$j9T$ihha48qF506RoNyrr5sm50$AFfIIwR.6k5ORO98s0r8RdA3E8jjL5v4mEaIVucFLY5:20042:0:99999:7:::

Username: systemd-network
Password Hash: !*:19953::::::

Username: areej
Password Hash: $y$j9T$PBK.GaTHmPu71w24v3lll1$yDuERaGzavOCTE51AqVnQG4gfz3d3.U4nHMlUkkm5A0:20042:0:99999:7:::

Username: kali
Password Hash: $y$j9T$zY1oKFxJlTgP2WcJhzbNl1$xhkUmB8R9fzETc/1kgL/nOPcWFTvhn17clxXCgyFjpC:19953:0:99999:7:::

Username: polkitd
Password Hash: !*:19953::::::

Username: stunnel4
Password Hash: !*:19953::::::

Username: ali
Password Hash: $y$j9T$CRDZyZe1FRxyWZM5BXhYJ1$sldKbEBnJiYVoqF0jFQoESfmotJJGSepkQGY.gAmRh7:20042:0:99999:7:::

Username: laiba
Password Hash: $y$j9T$zQaZQuOgpQjulZZfLzQeo/$As/Z14gep3rvdUBxSh9fhbnyLrzJwcthWBjDpEfWKz3:20042:0:99999:7:::

Username: systemd-timesync
Password Hash: !*:19953::::::
```

Decrypt the hash using john the ripper or hashcat.

john --format=crypt --wordlist=/home/kali/rock.txt hash.txt

```
┌──(kali㉿kali)-[~/volatility]
└─$ john --format=crypt --wordlist=/home/kali/rock.txt hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
Cost 1 (algorithm [1:descrypt 2:md5crypt 3:sunmd5 4:bcrypt 5:sha256crypt 6:sha512crypt]) is 0 for all loaded hashes
Cost 2 (algorithm specific iterations) is 1 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 19 candidates left, minimum 96 needed for performance.
laiba            (?)
1g 0:00:00:00 DONE (2024-11-17 02:48) 4.761g/s 90.47p/s 90.47c/s 90.47C/s djsdks..ali123456
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

```
──────────────────── Firefox passwords ────────────────────

[+] Password found !!!
URL: https://www.facebook.com
Login: laaibahq
Password: 63738

[+] Password found !!!
URL: https://tryhackme.com
Login: LaaibahQayyum
Password: 3QSrMF)bn&p.eu7

[+] Password found !!!
URL: https://github.com
Login: laaibah
Password: 83930

[+] Password found !!!
URL: https://flexstudent.nu.edu.pk
Login: 22I-1604
Password: 15929449


[+] 13 passwords have been found.
For more information launch it again with the -v option

elapsed time = 44.288269996643066
```

Pamspy Integration:

- **Pamspy** is a powerful tool designed to monitor and capture **PAM (Pluggable Authentication Module)** authentication events. PAM is used in Linux systems to handle authentication tasks like login, password changes, and other identity verifications.
- Pamspy hooks into the PAM library (`libpam.so`) to intercept and log sensitive information such as:

  Login Credentials:

  Captures usernames and passwords entered during system logins.

  Authentication Events:
  Tracks activities like password changes and other PAM-related actions.

  Stealth Monitoring:
  Operates in the background to log authentication attempts without user detection, making it potentially useful for penetration testing (or misuse in malicious scenarios).

When you press 3, you are directed to a page given below

```
PAM Event Capture and Analysis
Step 3: Running pamspy to capture PAM-related events ...
Capturing PAM events from /lib/x86_64-linux-gnu/libpam.so.0 ...
PID    | PROCESS | USERNAME | PASSWORD
```

- In the first terminal, ensure Pamspy is active and monitoring PAM events.
- You should see logs stating that events are being captured.
- Launch a second terminal without closing the first.
- Use the following command to initiate an SSH session:
  `ssh user@localhost`
- Replace `user` with your username.

```
┌──(kali㉿kali)-[~]
└─$ ssh ali@localhost
ali@localhost's password:
Linux kali 6.8.11-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.8.11-1kali2 (2024-05-30) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Nov 16 16:52:03 2024 from ::1
┌──(ali㉿kali)-[~]
└─$ ssh ahmed@localhost
ahmed@localhost's password:
Permission denied, please try again.
ahmed@localhost's password:
Linux kali 6.8.11-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.8.11-1kali2 (2024-05-30) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Nov 16 07:23:42 2024 from ::1
```

```
PAM Event Capture and Analysis
Step 3: Running pamspy to capture PAM-related events ...
Capturing PAM events from /lib/x86_64-linux-gnu/libpam.so.0 ...
PID    | PROCESS      | USERNAME | PASSWORD
32851  | sshd-session | laiba    | laiba
33158  | sshd-session | ali      | ali123456
33302  | sshd-session | ahmed    | ahmed
33302  | sshd-session | ahmed    | mario
33534  | sshd-session | areej    | areej
^CPAM event capture completed.

┌──(kali㉿kali)-[~/volatility]
└─$
```

Summary

The **Password Parser** tool extracts and analyzes sensitive information, specifically passwords, for cybersecurity and forensic purposes. It includes three main features:

- **Password Extraction from Live RAM**: Captures passwords from running systems.
- **Password Retrieval from Memory Dumps**: Extracts passwords from saved memory dumps.
- **Integration with Pamspy**: Monitors and logs Linux authentication events.

It's designed for forensic analysts, penetration testers, researchers, and law enforcement to recover passwords and authentication data from volatile memory and Linux systems. The tool requires Kali Linux and several software dependencies, including Python, Volatility, and Pamspy. It helps analyze memory, capture password hashes, and monitor authentication events in real-time.

References

- Volatility Cheat Sheet:

https://book.hacktricks.xyz/generic-methodologies-and-resources/basic-forensic-methodology/memory-dump-analysis/volatility-cheatsheet

- Volatility Command Reference - GitHub Wiki:

https://github.com/volatilityfoundation/volatility/wiki/command-reference

- PAMSpy - GitHub:

https://github.com/citronneur/pamspy

- Guide to live RAM Analysis

https://github.com/AlessandroZ/LaZagne/tree/master/Linux