Windows Forensics Project

Author : Adir Salinas (code S18)

Class code : 7736

Lecturer : Natali Erez

This document will talk about the Windows Forensics project:

- First, we'll discuss why we're doing this project and provide a brief explanation along with some details about how the project is set up.
- Secondly, we will provide instructions on how to use the script, explaining the various sections and detailing their individual responsibilities.
- Finally, we will break down the entire script, examining each command individually and explaining its specific purpose.

Let's begin with an overview of HDD and memory files.

So, what is an HDD and memory files?
A Hard Disk Drive (HDD) is a vital non-volatile storage device that holds critical digital data, including the operating system, software applications, and user files.

Memory files, integral to digital forensics, store temporary data such as actively processed information, playing a pivotal role. They enable the detection and analysis of malicious code or malware within the computer's volatile memory. Additionally, memory files can reveal sensitive data, including usernames and hashed or plaintext passwords, as well as other crucial information. Notably, this valuable information is transient and is wiped clean when the computer is powered off.

Analyzing these types of files provides essential information, including details on running processes, open network connections, and user interactions. Extracting and analyzing data from both HDDs and memory files are integral tasks in digital investigations, supported by a range of specialized tools designed for specific purposes.

In this process, various forensics tools play a pivotal role in extracting data effectively and accurately. These tools are necessary for digital investigators to navigate and comprehend the complex information that stored in HDDs and memory files. Further discussion on these tools will be provided later in the document.

Why Analyzing These Files is Crucial in Certain Cases?

The analysis of these files is essential for law enforcement and military investigations.
In cases where a suspect engages in illegal computer activity, this analysis provides essential evidence for court use. It facilitates the extraction of crucial data, assisting investigators in deciphering the case. And that's why every piece of information is necessary.

What is the purpose of the project and why is it necessary?

If investigators already know the forensic tools and how to extract the required data from files, why would they need a specialized tool to do it for them?

We are human beings, and we tend to make mistakes by missing vital information or issuing incorrect commands. The risk of losing crucial data is ever-present. To address this challenge, we've developed a tool designed to automate and correct these issues systematically. An additional advantage of automation is evident when investigators need to analyze multiple memory files. Manually typing each command can be time-consuming, potentially taking hours.
The goal is to enhance efficiency through automation, ensuring a thorough and error-free analysis of data in the most comprehensive manner possible.

This project addresses the need for an efficient solution in digital forensic analysis. By automating hard disk drive (HDD) and memory investigations, the program ensures the accurate identification, extraction, and display of critical data elements, such as network traffic and human-readable information.
It seamlessly integrates with Volatility software for thorough memory analysis and generates detailed reports. This automated approach is essential, as manual processes by investigators may result in data loss.
Additionally, the program includes a dedicated function for installing essential forensic tools, ensuring a streamlined and accurate operation.

Script structure and operating instructions

Before running the script, ensure you have either a memory file, a hard disk drive, or a directory containing the files you want to analyze, which will be provided to the tool for analysis.

Before running the script,
please ensure that a directory named 'DataAnalysis' does not exist in your current location.
If it does, kindly remove it to ensure smooth execution of the script.
(To remove it, you can either right-click on the 'DataAnalysis' directory and choose 'Remove' or use the following command in the terminal: **rm -rf DataAnalysis**.)

Providing instructions for extracting an HDD or a memory file from a computer

Instructions to extract memory file from a computer:

There are several methods for extracting a memory file from a computer.
The following demonstrates extraction using the 'Magnet RAM Capture' application.

Step 1: Download and Install:
In order to install the Magnet application, please follow the link to access the official website: https://www.magnetforensics.com/ click on the "**Resources**" tab within the website. This will open a submenu, and from there, click on "**Free Tools**" option. Scroll down the screen until you locate the option "**MAGNET HASH SETS MANAGER**." Click on it, and on the right side of the screen, you will find the option "**Get Free Tool**." The option will navigate you to a section where you can enter your details. After providing the required information, an email will be sent to you containing the link for downloading the application along with further instructions.

Step 2: Launch the Application:
Open the application on your computer, and a window for the Magnet Forensics User Agreement will appear. Select the option "**I Accept**," and the Magnet app will then launch.

Step 3: Capturing the memory:
Once the application has opened, under the "**Save RAM capture to…**" select **Browse** and choose the location on your computer where you want to save the memory file.
Name the file with your desired name and add the extension ". raw " then click on "**Start**" to initiate the memory capture process.

Step 4: Result of the memory file:
After the computer's memory has been captured and saved in your specified destination, you can proceed with the analysis using the tool and the generated memory file.

Instructions to extract hard disk drive (HDD) from a computer:

There are several methods for extracting a hard disk drive from a computer.
The following demonstrates extraction using the 'FTK Imager' application.

Step 1: Download and Install:

In order to download the FTK Imager application, please follow the link to access the installing website: https://accessdata-ftk-imager.software.informer.com/3.1/ .
Navigate to the **'Download'** tab at the top of the page, and then select the **'Download now'** option within the website Choose a destination to download the application. Once the application is downloaded, install it. After installation, launch the application and proceed by clicking **'Next'** and agreeing to the subsequent windows that appear until FTK Imager is successfully opened.

Step 2: Capturing the memory:
After opening the application, locate the '**File**' option at the top right of the screen. Under this option, you will find **'Create Disk Image...**' click on this option. After selecting this option, a window will appear with several choices. Choose **'Physical Drive**' if you intend to extract your physical drive. From the list of available drives, select the appropriate one and click **'Finish'**. Select **'Add'** and choose the desired image type. **'Raw'** is the most commonly used option. You will be given the option to add evidence information. choose **'Next'** after inserting the information. Then, select **'Browse'** to choose a location to save the file on your computer. Finally, click **'Finish'** and **'Start'** to initiate the process."

Step 4: Result of the memory file:
After the computer's hard disk drive has been captured and saved in your specified destination, you can proceed with the analysis using the tool and the generated HDD file.

After obtaining the files you wish to analyze, we can proceed with the script to analysis your files. Ensure that if you have a few files you wish to analyze, place them inside a directory together before providing them to the script.

What will the script perform in each section?

To run the script download the script file to your Linux and type in the terminal **“bash WF.project.sh”**

The script will analyze your provided files, organize the data into directories and generate files and reports to facilitate the investigation.

- The first section of the script displays text of the start of the script, will show the purpose of the script in a few words. Then, will install essential forensic tools if they are not already installed, ensuring that all necessary tools are properly set up on your system.

- The second section of the script will request the necessary files for analysis, verifying their existence before proceeding with the analysis. The script will ask you to choose whether you want to scan a single file or a directory of files by choosing the number **1** for a single file and the number **2** for a directory. Next, you will be required to provide the full name of the file/directory or the full path. Ensure that if you input the full name of the file/directory, their location needs to be in the same directory as the script being executed.

- The third section of the script will deeply analyze the selected files using forensic tools, organizing all the data into directories and files. It will generate reports to facilitate further analysis. We will shortly discuss what are the tools and what every tool does.

- The final section of the script will compress the directory of the analysis data and present on screen a diagram that will assist the investigator by showing the location of each forensics tool data that was extracted and where all the files are located at.

After the analysis of the file/s over all the analysis data will be stored within a compressed zip file named 'DataAnalysis.zip' To access the contents, simply right-click on the file and choose 'Extract Here’.

Explaining the whole script and each command

The last part of this document will explain everything about command, script and more in order to understand everything, and will break down each part of the script and the purpose of the different commands**. So, Let's Start…**

What is a command?

In computing, a "command" refers to a specific instruction or directive given to a computer or software to perform a particular action. Commands are typically issued through a commandline interface (CLI) or a terminal, where users enter text-based instructions to interact with the operating system or software applications to do an action. It can be a single word, a line of code, or a series of instructions that tell the computer what to do. Later we will talk about different commands and what each command instructs the computer to do.

What is a script?

In computing, a "script" is a set of instructions or number of commands that are written in a programming language to perform a specific task or automate a series of tasks. The script executed the commands or tasks by reading them line by line.
In simple words instead of typing each command every time, you writing all the command you want to execute in a script and when you run the script on a computer then all the commands you written will be executed one after the other.

What is a function?

In programming, a "function" is a set of commands grouped together to perform a specific task. Functions allow you to break down your script into smaller, more manageable pieces. Every function have a name that you set when creating the function and the function will be operate and execute only when we call the name of the function, if we are not calling the function the function will not be operated and the commands inside will not work.

What is a statement?

In programming, a "statement" is a single line of code that performs a specific action. It is the basic building block of a program and represents an executable instruction. Statements are the means by which a programmer gives instructions to a computer to perform tasks, make decisions, or control the flow of the program.

In simple words, a statement It's a line of code that performs a specific action, such as assigning a value to a variable, printing something on the screen, or making a decision based on certain conditions. Each statement is like a step in a set of instructions that the computer follows to carry out a task.

What is a variable?

In programming, a "variable" is a container or placeholder for storing data. It has a name and a value, and the value can change as the program runs. Variables are used to store and manipulate information within a program. in simple terms, think of a "variable" like a box with a name. You can put stuff (information or data) in the box, for example set a variable with a certain name and save inside the name a command output, when you call the variable name later, it will print in screen the output or data.

Lets talk about few important commands that will be executed in the script :

"echo" - Making space between commands or displaying text on screen.

"clear" - clearing the output terminal screen.

"sleep" - this command used to introduce a delay or pause in the script for a certain period of time. Most of the time there is a number after the command to wait in seconds.

" cd " - Short for "change directory," is used in the script and allows you to navigate between directories and move to a different location.

"grep" - searching for word or sentence inside an output and print them on screen.

"awk" - have a lot of used most of them in the script is to separate certain words or location.

"read" - read command reads the user's input and assigns it to the variable, using the flag (-p) after the command for printing text on screen before the input.

“sudo” - command is used to execute commands with elevated privileges. and it allows a permitted user to execute a command as the superuser.

"date" - command displays the current date and time on the system.

“sudo apt-get install " - command to install diffrent packages or applications.

"> /dev/null 2>&1" - this command used in addition to another command to execute the command quietly without displaying any output or errors.

Forensic tools that we're going to use, along with brief descriptions of each.

Bulk Extractor:

Bulk Extractor is a digital forensics tool known for efficiently scanning and extracting valuable information from large datasets. without parsing the file system or file system structures. The results are stored in feature files that can be easily inspected, parsed, or processed with automated tools, making it a valuable asset in digital investigations.

Binwalk:

Binwalk is a command-line tool specializing in analyzing and extracting files from binary data, particularly firmware images and embedded systems. Its signature analysis and pattern recognition make it essential for forensic analysis and security research.

Foremost:

Foremost is a digital forensics tool designed for file recovery and carving. It excels in extracting files and artifacts from disk images, file systems, and other storage media, using predefined headers, footers, and data structures. Foremost is valuable for recovering lost or deleted files during digital investigations.

Exiftool:

Exiftool is a versatile command-line tool used for reading, writing, and editing metadata information in a wide range of files, including images, audio, and documents. It is especially valuable in digital forensics for extracting and analyzing metadata details, such as timestamps, camera settings, and author information, providing insights into the history and origin of files.

Strings:

Strings is a command-line utility that extracts human-readable text from binary files. In digital forensics, it is commonly used to reveal plain text content embedded within files, assisting investigators in uncovering relevant information, passwords, or hidden messages. Strings is a straightforward yet powerful tool for analyzing the textual content of binary data during forensic examinations.

Volatility:

Volatility is a powerful open-source memory forensics framework. It specializes in the analysis of volatile memory (RAM) in digital investigations. By extracting and examining information stored in memory, Volatility allows forensic analysts to uncover details such as running processes, network connections, and open files. Its flexibility and support for various operating systems make it a key tool for memory forensics professionals.

Functions names, their order and short description:

Function for the start of the script include displaying a few details and outlining the purpose of the script: **START**

Function that check the needed Forensics tools and in case they are not installed, installing them : **APPSSCHK**

Functions that are within the function **APPSCHK** and responsible for installing the forensics tools: **BULK, SCALPEL, BINWALK, FOREMOST, geoiplookup, STRINGS, EXIFTOOL**

Functions that are within the function **APPCHK** and responsible to check if the forensics tools is installed: **BULKINFO, SCALPELINFO, BINWALKINFO,FOREMOSTINFO, STRINGSINFO ,GEOIPINFO, EXIFINFO**

Function to analyzed the file with different carvers while creating directories and files that storing the data: **ANALYZE**

Function that are within the function **APPSCHK** and responsible to analyze the file using the forensics tools and different carvers while creating directories and files that storing the data: **BulkAnalysis, BinwalkAnalysis, Foremost, Exiftool**

Function that use Strings command to extract information from a file, using different patterns and keywords and then organizes and inserts the results into directories and files: **STRINGS**

Functions inside that **STRINGS** function that use the strings command and extract information from a file, using different patterns and keywords: **emails, IPAddr, MACAddr, FileExt, Security, Credentials, URLS, UserWords.**

Function to check all created files for the presence of “. pcap" files: **pcapsearch**

Function for volatility analysis to extract information from a file, using volatility command and then organizes and inserts the results into directories and files: **VOL**

Function that are within the function **APPCHK** that remove the existing Volatility files, if they exist, and install Volatility to ensure proper execution of the commands: **VOLINSTL**

Function that are within the function **APPCHK** that analysis the file using volatility command while using different keywords: **VOLAnalysis, imageinfo, pslist, netscan, dlllist, filescan.**

Function that will Check Whether the Provided File Is Compatible for Analysis with Volatility: **VOLCHK**

Function that uses the volatility to scan for Registry Files information within the file while using different keywords: **RegFiles, SYSTEM, SOFTWARE, SAM, SECURITY**

Function to ask the user for a file to analyze, verifying its existence, and if the file exists, calling the function that analyzes the file: **FILE.**

Function to ask the user for a Directory to analyzed, verifying its existence, and if the file exists, calling the function that analyzes the Directory: **DIRECTORY**

Function that will ask the user Whether he want to Analyze a Single File or A Directory of Files and then calling the needed functions to continue: **CHOICE**

Function to initiates the compression process to create a zip archive of the analysis data: **ZIP**

Function for the end of the script that shows a diagram about the locations of all directories and files that was created are stored at: **END.**

Braking down the script to parts and explain each part in details

(Basic command that was explained before will not have description about them)

Inside the functions there will be other functions names that getting called when needed!

At the beginning of the script, we set up some variables to make the text look fancier. These variables help us style the text by making it bold or changing its color. Let's take a closer look at each of these commands and what they do.

```
bold=$(tput bold) # sets the variable "bold" to the e
normal=$(tput sgr0) # sets the variable "normal" to t
green=$(tput setaf 2) # sets the variable "green" to
blue=$(tput setaf 4) # sets the variable "blue" to th
red=$(tput setaf 1) # sets the variable "red" to the
purple=$(tput setaf 5) # sets the variable "purple" t
```

bold=$(tput bold)

The command sets the variable 'bold' to the escape sequence for bold text formatting using the 'tput' terminal utility, enabling the styling of text as bold in the script

normal=$(tput sgr0)

The command sets the variable ' normal ' that uses the 'tput' terminal utility to assign the escape sequence for resetting all text attributes, The “sgr0” in the command represents the "Select Graphic Rendition" escape sequence that resets all text attributes to their default values, allowing text to return to its normal appearance in the script.

green=$(tput setaf 2)

The command sets the variable 'green' that uses the 'tput' command to assign the escape sequence for green text color. The 'setaf' command is a part of the 'tput' utility, representing 'Set ANSI Foreground,' and the '2' parameter specifies the green color, enabling subsequent text in the script to appear in green.

blue=$(tput setaf 4)

The command employs the 'tput' utility to set the variable 'blue' with the escape sequence for changing text color to blue. In the 'setaf' command, representing 'Set ANSI Foreground,' the parameter '4' specifies the blue color, enabling subsequent text in the script to appear in blue.

red=$(tput setaf 1)

The command utilizes the 'tput' utility to set the variable 'red' with the escape sequence for changing text color to red. In the 'setaf' command, representing 'Set ANSI Foreground,' the parameter '1' specifies the red color, enabling subsequent text in the script to appear in red.

purple=$(tput setaf 5)

The command 'purple=$(tput setaf 5)' employs the 'tput' utility to set the variable 'purple' with the escape sequence for changing text color to purple. In the 'setaf' command, representing 'Set ANSI Foreground,' the parameter '5' specifies the purple color, enabling subsequent text in the script to appear in purple.

The following command ensures user authentication and downloads essential applications for text formatting and styling.

```
clear
echo
echo "[!] Before We Start, Please Insert Your Password:"
echo
sleep 2
sudo -kv # updates the user's cached credentials, asking for the user's password a
sudo apt-get install -y figlet > /dev/null 2>&1 # command to install the 'figlet'
sudo apt-get install -y lolcat > /dev/null 2>&1 # command to install the 'lolcat'
```

sudo -kv

The sudo -kv command updates the user's cached credentials, requesting the user's password, if necessary, without executing any specific command. Used within the script to ensure that previously entered sudo passwords won't be prompted again for subsequent sudo commands.

sudo apt-get install -y figlet

Command to install the “figlet” package. The figlet command generates normal text into ASCII art text banners using various font styles. It’s for visual purposes only.
(The flag -y is for automatically answer "yes" for questions)

sudo apt-get install -y lolcat

Command that installs the 'lolcat' package, which adds rainbow coloring to text output in the terminal, It’s for visual purposes only. (The flag -y is for automatically answer "yes" for questions)

The first function called “START” for the start of the script includes displaying a few details and outlining the purpose of the script:

```
function START () # Function for the start of the script and display short descripti
{
    clear
    echo
    echo
    figlet -t -c -f slant Welcome To My Digital Forensic    Analysis Tool ! |lolcat
    sleep 3
    echo
    echo
    echo
    echo "${bold}                        This Program Automates Precise Analysis of
    sleep 6.5
    echo
    echo
    echo "${bold}                                                  Extract Crucial
    sleep 4
    echo
    echo
    echo "${bold}                                              Combining Investigatior
    sleep 5
    echo
    echo
    echo "${bold}                                                While Ensuring Effici
    sleep 4
    echo
    echo
    echo
    echo "${bold}
    sleep 1.5
    echo "
    read -n 1 -s -r # The command reads a single character from the user without dis

}
START
```

figlet -t -c -f slant Welcome To My Digital Forensic Analysis Tool ! | lolcat

This command employs 'figlet' to convert text into ASCII art ,"-t" for expanding to the entire screen,"-c" for centering the text, and "-f" for specify which font to use in this case its "slant font". The output is further enhanced by piping it to 'lolcat' for rainbow coloring, contributing to a visually appealing welcome message in the Digital Forensic Analysis Tool script.

read -n 1 -s -r

This command instructs the shell to read a single character from the user's input without displaying it on the screen, the flag “-n 1” This option specifies to read only one character. The flag “-s” makes the input silent, meaning the entered character is not displayed on the screen. And the flag “-r” prevents backslashes from being treated as escape characters.
In simple words waiting for the user to input a character before continuing with the rest of the script.

Function called “APPSCHK” that checks the needed Forensics tools and in case they are not installed, installing them. Inside the function there is more function that responsible for each tool that we will explain each one of them.

The following functions that are within the function APPSCHK are responsible for installing the forensics tools

```
function APPSCHK () # The following functions is for downloading the nec
{
    echo
    echo
    echo
    echo
    echo
    echo "${bold}[?] Checking If The Necessary Forensics Tools Are Insta
    echo

    function BULK () # Function to install the 'Bulk_extractor' package
    {
        sudo apt-get -qq -y install bulk-extractor > /dev/null 2>&1
    }
    function SCALPEL () # Function to install the 'Scalpel' package
    {
        sudo apt-get -qq -y install scalpel > /dev/null 2>&1
    }
    function BINWALK () # Function to install the 'Binwalk' package
    {
        sudo apt-get -qq -y install binwalk > /dev/null 2>&1
    }

    function FOREMOST () # Function to install the 'Foremost' package
    {
        sudo apt-get -qq -y install foremost > /dev/null 2>&1
    }

    function geoiplookup () # Function to install the 'geoip-bin' packag
    {
        sudo apt-get -qq -y install geoip-bin > /dev/null 2>&1
    }


    function STRINGS () # Function to install the 'Strings' package
    {
        sudo apt-get -qq -y install binutils > /dev/null 2>&1
    }

    function EXIFTOOL () # Function to install the 'Exiftool' package
    {
        sudo apt-get -qq -y install libimage-exiftool-perl > /dev/null 2
    }
```

Each of the following functions installing his own forensics tools with the same command.

sudo apt-get -qq -y install (name of the tool)

The command ‘sudo apt-get -qq -y install’ is a command for installing a specific tool, The flags “-qq” make the installation process quiet and the flag “-y” assume "yes" as the answer to any prompts, facilitating unattended installations. “ (name of the tool)” replace this command with the actual name of the tool that the script is installing.

The following functions that are within the function APPSCHK responsible to check if the forensics tools are installed if there are not they calling the function that installed them.

```
function BULKINFO () # Function to check if Bulk_extractor applicati
{
    if command -v bulk_extractor > /dev/null 2>&1  #  if statment th
    then
        return   # command to return from the current function and c
    else
        echo '[!] The Forensic Tool "Bulk_extractor" Is Not Installe
        echo
        sleep 1.5
        echo "[*] Advancing With The Installation Of The Forensic To
        BULK  # Calling the function that installed the application.
        echo
        echo "${bold}[✓] Installation Completed Successfully.  ${nor
    fi
}
BULKINFO

function SCALPELINFO () # Function to check if scalpel application i
{
    if command -v scalpel > /dev/null 2>&1  # if statment that check
    then
        return  # command to return from the current function and co
    else
        echo '[!] The Forensic Tool "Scalpel" Is Not Installed On Yo
        echo
        sleep 1.5
        echo "[*] Advancing With The Installation Of The Forensic To
        SCALPEL  # Calling the function that installed the applicati
        echo
        echo "${bold}[✓] Installation Completed Successfully.  ${nor
    fi
}
SCALPELINFO
```

```
function BINWALKINFO ()  # Function to check if binwalk application
{
    if command -v binwalk > /dev/null 2>&1  # if statment that check
    then
        return   # command to return from the current function and c
    else
        echo '[!] The Forensic Tool "Binwalk" Is Not Installed On Yo
        echo
        sleep 1.5
        echo "[*] Advancing With The Installation Of The Forensic To
        BINWALK # Calling the function that installed the applicatio
        echo
        echo "${bold}[✓] Installation Completed Successfully.  ${nor
    fi
}
BINWALKINFO

function FOREMOSTINFO ()  # Function to check if foremost applicatio
{
    if command -v foremost > /dev/null 2>&1  # if statment that chec
    then
        return   # command to return from the current function and c
    else
        echo '[!] The Forensic Tool "Foremost" Is Not Installed On Y
        echo
        sleep 1.5
        echo "[*] Advancing With The Installation Of The Forensic To
        FOREMOST # Calling the function that installed the applicati
        echo
        echo "${bold}[✓] Installation Completed Successfully.  ${nor
    fi
}
FOREMOSTINFO
```

```
    function STRINGSINFO ()  # Function to check if foremost application
    {
        if command -v strings > /dev/null 2>&1 # if statment that check
        then
            return  # command to return from the current function and co
        else
            echo '[!] The Forensic Tool "Strings" Is Not Installed On Yo
            echo
            sleep 1.5
            echo "[*] Advancing With The Installation Of The Forensic To
            STRINGS  # Calling the function that installed the applicati
            echo
            echo "${bold}[✓] Installation Completed Successfully.  ${nor
        fi
    }
    STRINGSINFO

    function GEOIPINFO ()  # Function to check if geoiplookup applicatio
    {
        if command -v geoiplookup > /dev/null 2>&1  # if statment that c
        then
            return   # command to return from the current function and c
        else
            echo '[!] The Forensic Tool "Geoiplookup" Is Not Installed O
            echo
            sleep 1.5
            echo "[*] Advancing With The Installation Of The Forensic To
            geoiplookup  # Calling the function that installed the appli
            echo
            echo "${bold}[✓] Installation Completed Successfully.  ${nor
        fi
    }
    GEOIPINFO


    function EXIFINFO () # Function to check if exiftool application is
    {
        if command -v exiftool > /dev/null 2>&1  # if statment that chec
        then
            return  # command to return from the current function and co
        else
            echo '[!] The Forensic Tool "Exiftool" Is Not Installed On Y
            echo
            sleep 1.5
            echo "[*] Advancing With The Installation Of The Forensic To
            EXIFTOOL # Calling the function that installed the applicati
            echo
            echo "${bold}[✓] Installation Completed Successfully.  ${nor
        fi
    }
    EXIFINFO

    sleep 3
    echo
    echo "[✓] All The Necessary Applications Have Been Checked, Prepared
    sleep 5
    echo
    echo
}
APPSCHK
```

Each one of the following functions has the same if statement with identical conditions and commands. The explanation provided below applies to all these functions.

if command -v (package name)

The following if statement is an indicator that checks if a package is installed on your system So, the command with -v that stands for “verbose” check if a path of the package was received in the output. The (package name) placeholder changes for each function, representing the tool associated with that function. If the command receives a path in its output, indicating the package is installed, it continues to then statement and returns without taking any action, otherwise, it notifies the user that the package is not installed on their system, and calling the function the installed the package inside the else statement.

return

The return command is used to return from the current function and continue to the rest of the script without taking any action.

Function that analyzes the files using the forensics tools and different carvers while creating directories and files that storing the data

```
function ANALYZE ()  # Function to analyzed the file with diff
{
    sleep 3.5
    clear
    echo
    echo
    echo "${bold}${blue}=====================================
    location=$(pwd) # captures the current working directory p
    mkdir DataAnalysis  > /dev/null 2>&1  # creates a director
    cd DataAnalysis  # Entering the new created directory Name

    file_path="$1"  # The variable assignment sets the variabl
    base_name=$(basename "$file_path")  # The command extracts
    file_name="Data_$base_name"  # The command creates a new v

    mkdir "$file_name" && cd $file_name  # creating a director
```

location=$(pwd)

The command sets the variable "location" to the present working directory using the (pwd) command, representing the location path of the current directory in the script.

Within the rest of the script the "location" variable will be used often.
Along with the "cd" command, it means changing the location to the directory stored in that variable.
(where the script file is located).

mkdir DataAnalysis

The command creates a new directory named "DataAnalysis"

cd DataAnalysis

changes the current location to that newly created directory and enters the "DataAnalysis" directory.

file_path="$1"

The variable assignment sets the variable "file_path" to the value of the first command-line argument passed to the script or function. Means later the value that is going to be send is the user files that he wants to be analyzed.

base_name=$(basename "$file_path")

The command extracts the base name (filename without the directory path) of the file specified in the "file_path" variable and assigns it to the "base_name" variable. This is useful when you want to isolate the filename from its full path in case user input a path to the file.

File_name="Data_$base_name"

The command creates a new variable named "file_name" and assigns it a value composed of the string "Data_" followed by the base name extracted from the "base_name" variable.

mkdir "$file_name" && cd $file_name

creates a new directory with the name stored in the "file_name" variable and then changes the current working directory to that newly created directory.

The following functions that are within the function ANALYZE are responsible for extracting data and information from a file, and then organizes and inserts the results into directories for each function his own forensics tool that being used.

The following function used Bulk_extractor as the forensics tool for analysis.

```
function BulkAnalysis ()  # Function that using the Bulk_extractor command to extract data and information from a file, and then organizes and inserts the results into directories
{
    sleep 3
    echo
    echo
    echo
    echo "                                                ${bold} Extracting And Conducting Forensic Analysis Using Bulk_Extractor...${normal}  "
    cd $location/DataAnalysis/$file_name # changes the current location to a the locations under the vairables and names.
    mkdir Bulk_extractor  # inside the location make a directory named "Bulk_extractor"
    cd $location/DataAnalysis/$file_name/Bulk_extractor # changes the current location to a the locations under the vairables and names

    # making a file named "Data_Report" when using ">" and when using ">>" inserting text and data inside the file without overwriting the data inside the file.
    echo > Data_Report
    echo "Analysis Audit" >> Data_Report
    echo >> Data_Report
    echo "Bulk_extractor Started At: $(date) " >> Data_Report
    time=$(date) # captures the current date and time using the "date" command and assigns it to the variable "time".

    cd $location  # changes the current location to the location under the variable "location".
    bulk_extractor "$1" -o $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_FullData > /dev/null 2>&1  # uses the bulk_extractor tool to analyze the file specified by the use
    cd $location/DataAnalysis/$file_name/Bulk_extractor  # changes the current location to the location under the variables.
    cp -r Bulk_FullData  Bulk_Filtered_Data && cd Bulk_Filtered_Data # copies the contents of the directory "Bulk_FullData" to a new directory called "Bulk_Filtered_Data" and then
    find . -type f -size 0 -exec rm -f {} + # the command searches for files in the current directory and its subdirectories with a size of 0 bytes and then removes them using the

    cd $location  # changes the current location to the location under the variable "location".
    # The following insering diffrent text and data inside the "Data_Report" file. when using ">>" its keep the current data that in the file and not overwrite it.
    echo "Command: bulk_extractor "$1" -o $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_FullData "  >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report
    echo "Output directory (full data): $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_FullData"  >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report
    echo "Output directory (filtered data): $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_Filtered_Data"  >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report
    echo "-------------------------------------------------------------------------------" >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report
    echo  >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report
    echo "File Name: "$1" " >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report
    cd $location  # changes the current location to the location under the variable "location".
    echo "File Size: $( du -h "$1" |awk '{print $1}') ( $(ls -l "$1" |awk '{print $5}') bytes)"  >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report  # The "du -h "$1"
    cd $location/DataAnalysis/$file_name/Bulk_extractor # changes the current location to the location under the variables.


    # The following insering diffrent text and data inside the "Data_Report" file. when using ">>" its keep the current data that in the file and not overwrite it.
    echo >> Data_Report
    echo "Beginning of the Analysis: $time " >> Data_Report
    echo "The End of the Analysis: $(date) " >> Data_Report
    echo >> Data_Report
    echo "-------------------------------------------------------------------------------" >> Data_Report
    echo >> Data_Report
    echo "The following displays all the capture files within the Bulk_Filtered_Data directory. " >> Data_Report
    cd $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_Filtered_Data  # changes the current location to the location under the variables.
    echo "Total files captured: $(find . -type f |wc -l) Files" >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report  # the command "find . -type f |wc -l" counts and d
    cd $location/DataAnalysis/$file_name/Bulk_extractor  # changes the current location to the location under the variables.
    echo >> Data_Report
    cd $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_Filtered_Data   # changes the current location to the location under the variables.
    echo " $(find . -type f ) " >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report
    echo
    echo "Bulk_extractor Concluded At: $(date) " >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report

    echo
    echo
    echo
    echo "${bold}${blue}==================================================>${red} Data Was Successfully Analyzed And Extracted With Bulk_Extractor! ${blue}<====================
}
BulkAnalysis "$1"
```

Cd $***/$***/$***

Whenever the cd command is being used in the script with various variables or names, it is changing the location of the current directory to the one specified by those variables or different names.

mkdir Bulk_extractor

Inside the current location make a directory named "Bulk_extractor"

echo > (file name)

When using the command “echo” with a single “>” and after it a file name, this command empties the content of the specified file (or creates an empty file if it doesn't exist).
It uses a single > symbol, known as output redirection, to overwrite the contents of the file.

echo >> (file name)

When using the command “echo” with a double “>” and after it a file name, this command appends the output of the echo command to the end of the specified file.
It uses double >> symbols for output redirection, ensuring that the existing content in the file is preserved, and adding a new content without overwriting or deleting the output inside.

time=$(date)

This command captures the current date and time using the "date" command and assigns it to the variable "time".

bulk_extractor "$1" -o $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_FullData

This command uses the bulk_extractor forensics tool to analyze the file specified by the user input ($1). The extracted data and information will be saved in the directory path that specify in flag "-o" under the variable's $location, "DataAnalysis", $file_name, and "Bulk_extractor/Bulk_FullData".

cp -r Bulk_FullData Bulk_Filtered_Data && cd Bulk_Filtered_Data

This command copies the contents of the directory "Bulk_FullData" to a new directory called "Bulk_Filtered_Data" and then changes the current location and enters "Bulk_Filtered_Data".

find . -type f -size 0 -exec rm -f {} +

The command locates and removes all files (not directories) with a size of 0 bytes within the current directory and its subdirectories. The "-type f" flag ensures that only regular files are considered, and the "-size 0 " flag filters for files with a size of 0 bytes. The "-exec rm -f {} +" part executes the "rm -f" command on each found file, forcibly removing them.
The " {} " placeholder represents the current file being processed, and the "+" at the end optimizes the execution by passing multiple files at once.

The following command using the >> with an echo command and inserting different text and data inside the "Data_Report" file.

```
cd $location  # changes the current location to the location under the variable "location".
# The following insering diffrent text and data inside the "Data_Report" file. when using ">>" its keep the current data that in the f
echo "Command: bulk_extractor "$1" -o $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_FullData "  >> $location/DataAnalysis/$fil
echo "Output directory (full data): $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_FullData"  >> $location/DataAnalysis/$file_n
echo "Output directory (filtered data): $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_Filtered_Data"  >> $location/DataAnalysi
echo "-------------------------------------------------------------------------------" >> $location/DataAnalysis/$file_name
echo  >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report
echo "File Name: "$1" " >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report
cd $location  # changes the current location to the location under the variable "location".
echo "File Size: $( du -h "$1" |awk '{print $1}') ( $(ls -l "$1" |awk '{print $5}') bytes)"  >> $location/DataAnalysis/$file_name/Bulk
cd $location/DataAnalysis/$file_name/Bulk_extractor # changes the current location to the location under the variables.

# The following insering diffrent text and data inside the "Data_Report" file. when using ">>" its keep the current data that in the f
echo >> Data_Report
echo "Beginning of the Analysis: $time " >> Data_Report
echo "The End of the Analysis: $(date) " >> Data_Report
echo >> Data_Report
echo "-------------------------------------------------------------------------------" >> Data_Report
echo >> Data_Report
echo "The following displays all the capture files within the Bulk_Filtered_Data directory. " >> Data_Report
cd $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_Filtered_Data  # changes the current location to the location under the varia
echo "Total files captured: $(find . -type f |wc -l) Files" >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report  # the com
cd $location/DataAnalysis/$file_name/Bulk_extractor  # changes the current location to the location under the variables.
echo >> Data_Report
cd $location/DataAnalysis/$file_name/Bulk_extractor/Bulk_Filtered_Data   # changes the current location to the location under the vari
echo " $(find . -type f ) " >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report
echo
echo "Bulk_extractor Concluded At: $(date) " >> $location/DataAnalysis/$file_name/Bulk_extractor/Data_Report
```

The following function used Binwalk as the forensics tool for analysis.

```
function BinwalkAnalysis ()  # Function that using the Binwalk comma
{
    sleep 3
    echo
    echo
    echo
    echo "                                                        $
    echo "
    cd $location/DataAnalysis/$file_name  # changes the current loca
    mkdir Binwalk  # inside the location make a directory named "Bin
    cd $location/DataAnalysis/$file_name/Binwalk # changes the curre

    # The following insering diffrent text and data inside the "Binw
    echo "  " > Binwalk_Analyzed_Data
    echo "[!] Presenting the Binwalk-Analyzed Data Along with the Sc
    echo "  " >> Binwalk_Analyzed_Data
    echo " >  The Binwalk Scan Was Initiated At: $(date) " >> Binwal
    echo "  " >> Binwalk_Analyzed_Data
    cd $location  # changes the current location to the location und
    binwalk "$1" > $location/DataAnalysis/$file_name/Binwalk/Binwalk
    cd $location/DataAnalysis/$file_name/Binwalk # changes the curre
    echo " >  The Binwalk Scan Was Concluded At: $(date) " >> Binwal
    echo "  " >> Binwalk_Analyzed_Data
    echo "[*] The Following Displays Keywords Used to Scan the Data,
    echo "  " >> Binwalk_Analyzed_Data
    echo " -  The Keyword 'Microsoft Executable, Portable' Indicates
    echo " -  The Keyword 'Zip Archive Data' Indicates for a Compres
    echo " -  The Keyword 'HTML Document Header' Indicates the Poten
    echo " -  The Keyword 'XML Document' Indicates for Configuration
    echo " -  The Keyword 'GIF Image Data' Indicates the Potential P
    echo " -  The Keyword 'JPEG Image Data' Indicates the Potential
    echo " -  The Keyword 'PNG image' Indicates the Potential Presen
    echo "  " >> Binwalk_Analyzed_Data
    echo "[*] The Following Displays the Entire Data Extracted from
    echo "  " >> Binwalk_Analyzed_Data
    cat Binwalk_FullData >> Binwalk_Analyzed_Data  # inserting all t
    rm Binwalk_FullData # removing the file "Binwalk_FullData"

    echo
    echo
    echo
    echo "${bold}${blue}============================================

}
BinwalkAnalysis "$1"
```

Cd $***/$*/$*****

Whenever the cd command is being used in the script with various variables or names, it is changing the location of the current directory to the one specified by those variables or different names.

mkdir Binwalk

Inside the current location make a directory named " Binwalk"

echo > (file name)

When using the command “echo” with a single “>” and after it a file name, this command empties the content of the specified file (or creates an empty file if it doesn't exist).
It uses a single > symbol, known as output redirection, to overwrite the contents of the file.

echo >> (file name)

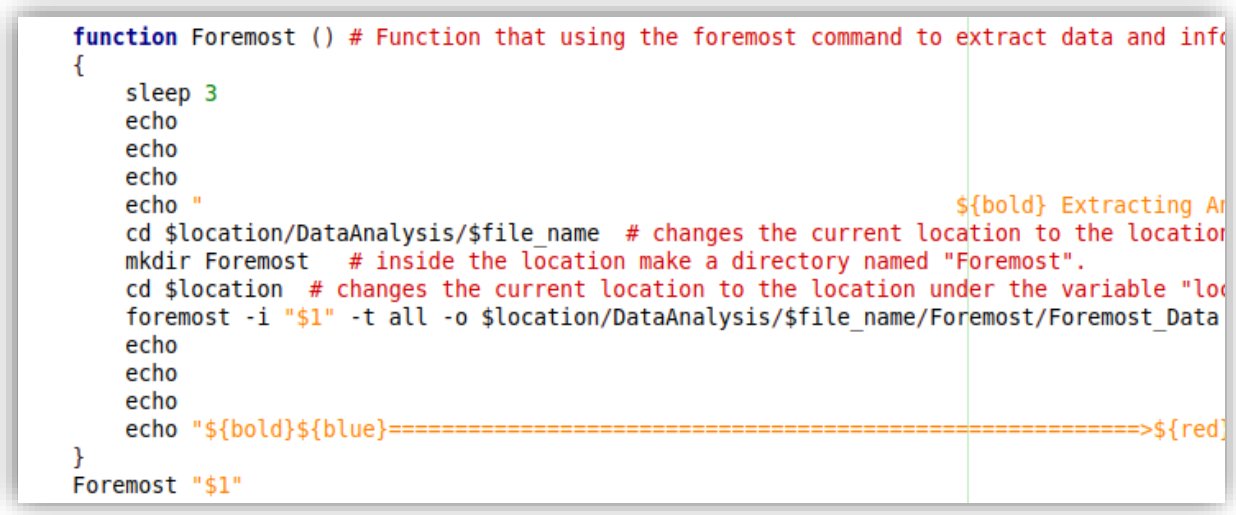
When using the command “echo” with a double “>” and after it a file name, this command appends the output of the echo command to the end of the specified file.
It uses double >> symbols for output redirection, ensuring that the existing content in the file is preserved, and adding a new content without overwriting or deleting the output inside.

The following commands creating a directory named “Binwalk_Analyzed_Data” inserting different text and data inside the "Binwalk_Analyzed_Data" file.

binwalk "$1" > $location/DataAnalysis/$file_name/Binwalk/Binwalk_FullData

The command uses the binwalk tool to analyze the file specified by the user input ($1).
The extracted data and information will be saved in the directory path that specify after the symbol ">" under the variable’s location, "DataAnalysis", file_name, and "Binwalk/Binwalk_FullData".

The following function used Foremost as the forensics tool for analysis.



```
function Foremost () # Function that using the foremost command to extract data and info
{
    sleep 3
    echo
    echo
    echo
    echo "                                                    ${bold} Extracting Ar
    cd $location/DataAnalysis/$file_name  # changes the current location to the location
    mkdir Foremost   # inside the location make a directory named "Foremost".
    cd $location  # changes the current location to the location under the variable "loc
    foremost -i "$1" -t all -o $location/DataAnalysis/$file_name/Foremost/Foremost_Data
    echo
    echo
    echo
    echo "${bold}${blue}============================================================>${red}
}
Foremost "$1"
```

mkdir Foremost

inside the location make a directory named "Foremost".

foremost -i "$1" -t all -o $location/DataAnalysis/$file_name/Foremost/Foremost_Data

This command uses the "foremost" tool to conduct file carving on the specified input file ("$1") with the “-i ” and extracts various file types specified by the -t all option. The extracted data is then stored in the directory path under the variable’s location after the “-o” flag.

The following function used Exiftool as the forensics tool for analysis.

```
function Exiftool () # Function that using the Exiftool command to extract data and
{
    sleep 3
    echo
    echo
    echo
    echo "                                                    ${bold} Extractin
    cd $location/DataAnalysis/$file_name # changes the current location to the locat
    mkdir Exiftool   # inside the location make a directory named "Exiftool".
    cd $location # changes the current location to the location under the variable "
    exiftool "$1" > $location/DataAnalysis/$file_name/Exiftool/Exiftool_data # comma
    cd $location/DataAnalysis/$file_name/Exiftool # changes the current location to
    echo "   " > Exiftool_Data
    echo "[*] This Document Showcases the Results of the Metadata Output Obtained by
    echo "   " >> Exiftool_Data
    echo "[*] The Following Displays All Exiftool Data Output: " >> Exiftool_Data
    echo "   " >> Exiftool_Data
    cat Exiftool_data >> Exiftool_Data  # inserting all the text inside the Exiftool
    rm Exiftool_data # removing the file "Exiftool_data"
    sleep 2.5
    echo
    echo
    echo
    echo "${bold}${blue}============================================================>${
}
Exiftool "$1"
```

mkdir Exiftool

Inside the location make a directory named "Exiftool".

exiftool "$1" > $location/DataAnalysis/$file_name/Exiftool/Exiftool_data

This command uses the "exiftool" tool to extract metadata from the specified file ("$1") and writes the extracted information to a file named "Exiftool_data" within the directory path under the variables location.

cat Exiftool_data >> Exiftool_Data

inserting all the text inside the Exiftool_data file and copy it inside Exiftool_Data without overwrite it.

rm Exiftool_data

This command removing the file "Exiftool_data"

The following function used Strings as the forensics tool for analysis.

```
function STRINGS () # Function that using the Strings command to extract information from a file,
{
    sleep 3
    echo
    echo
    echo
    echo "                                                    ${bold} Extracting And Conduct
    cd $location/DataAnalysis/$file_name  # changes the current location to the location under th
    mkdir Strings    # inside the location make a directory named "Strings".
    cd Strings  # entering the new created directory
    mkdir Strings_Analyzed  # inside the location make a directory named "Strings_Analyzed" .
```

mkdir Strings

Inside the location make a directory named "Strings".

cd Strings

Entering the new created directory named “Strings”

mkdir Strings_Analyzed

inside the new created directory location make a directory named "Strings_Analyzed" .

The following functions that are within the function STRINGS are using the strings command and extract information from a file, using different patterns and keywords.

The next function using the Strings command to extract information from a file, about Email address’s structure using specific pattern and then organizes and inserts the results into file."

```
function emails () # Function that using the Strings command to extract information from a file, about Email addresses structure using sp
{
    cd $location  # changes the current location to the location under the variable "location".
    strings "$1" |grep -E -o '\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b' > $location/DataAnalysis/$file_name/Strings/Strings_Ar
    cd $location/DataAnalysis/$file_name/Strings/Strings_Analyzed  # changes the current location to the location under the variables.

    # The following insering diffrent text and data inside the "Strings_Emails" file. when using ">>" its keep the current data that in t
    echo "  " >> Strings_Emails
    echo "[!] The Total Number Of Emails Stracutre Is: $(cat Strings_emails|wc -l)" >> Strings_Emails
    echo "[*] The Following Displays the Count of Email Addresses from Well-Known Domains:" >> Strings_Emails
    echo "  " >> Strings_Emails
    echo " >  The Total Count of Email Addresses with the '.com' Domain Is: $(cat Strings_emails|grep -i .com|wc -l) " >> Strings_Emails
    echo " >  The Total Count of Email Addresses with the '.net' Domain Is: $(cat Strings_emails|grep -i .net|wc -l) " >> Strings_Emails
    echo " >  The Total Count of Email Addresses with the '.dll' Domain Is: $(cat Strings_emails|grep -i .dll|wc -l) " >> Strings_Emails
    echo " >  The Total Count of Email Addresses with the '.org' Domain Is: $(cat Strings_emails|grep -i .org|wc -l) " >> Strings_Emails
    echo " >  The Total Count of Email Addresses with the '.gov' Domain Is: $(cat Strings_emails|grep -i .gov|wc -l) " >> Strings_Emails
    echo "  " >> Strings_Emails
    echo "[*] The Following Displays All the Discovered Email Addresses.: " >> Strings_Emails
    echo "  " >> Strings_Emails
    cat Strings_emails >> Strings_Emails  # inserting all the text inside the Strings_emails file and copy it inside Strings_Emails witho
    rm Strings_emails  # removing the file "Exiftool_data"
}
emails "$1"
```

The following inserting different text and data inside the "Strings_Emails" file.

strings "$1" |grep -E -o '\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b' > $location/DataAnalysis/$file_name/Strings/Strings_Analyzed/Strings_emails

The provided command extracts email addresses with the according pattern from the content of the file specified by the argument "$1". and save it into the location under the variables.

cat Strings_emails >> Strings_Emails

inserting all the text inside the Strings_emails file and copy it inside Strings_Emails without overwrite it.

rm Strings_emails

removing the file named "Exiftool_data"

Function that uses the Strings command to extract information from a file, about IP address's structure using specific pattern and then organizes and inserts the results into file."

```
function IPAddr ()  # Function that using the Strings command to extract information from a file, about IP addresses structure using specific pattern and
{
    cd $location # changes the current location to the location under the variable "location".
    strings "$1" |grep -E -o '\b([0-9]{1,3}\.){3}[0-9]{1,3}\b' |sort |uniq  > $location/DataAnalysis/$file_name/Strings/Strings_Analyzed/Strings_IP.Addr
    cd $location/DataAnalysis/$file_name/Strings/Strings_Analyzed  # changes the current location to the location under the variables.

    # The following insering diffrent text and data inside the "Strings_IP_Addr" file. when using ">>" its keep the current data that in the file and not
    echo "  " >> Strings_IP_Addr
    echo "[!] The Total Count Of IP Addresses Is: $(cat Strings_IP.Addr|wc -l)" >> Strings_IP_Addr
    echo "  " >> Strings_IP_Addr
    echo "[*] The Following Displays All the Discovered IP Addresses: " >> Strings_IP_Addr
    echo "  " >> Strings_IP_Addr
    cat Strings_IP.Addr >> Strings_IP_Addr  # inserting all the text inside the Strings_IP.Addr file and copy it inside Strings_IP_Addr without overwrite
    rm Strings_IP.Addr  # removing the file "Strings_IP.Addr"
}
IPAddr "$1"
```

The following is inserting different text and data inside the "Strings_IP_Addr" file.

strings "$1" |grep -E -o '\b([0-9]{1,3}\.){3}[0-9]{1,3}\b' |sort |uniq > $location /DataAnalysis/$file_name/Strings/Strings_Analyzed/Strings_IP.Addr

The provided command extracts Ip addresses with the according pattern from the content of the file specified by the argument "$1". and save it into the location under the variables.

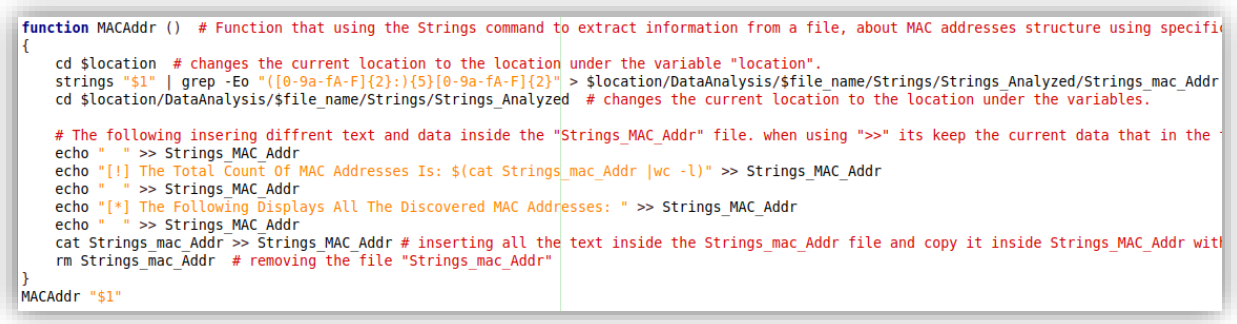
cat Strings_IP.Addr >> Strings_IP_Addr

inserting all the text inside the Strings_IP.Addr file and copy it inside Strings_IP_Addr without overwrite it.

rm Strings_IP.Addr

removing the file "Strings_IP.Addr"

Function that uses the Strings command to extract information from a file, about MAC address's structure using specific pattern and then organizes and inserts the results into file.



```
function MACAddr ()  # Function that using the Strings command to extract information from a file, about MAC addresses structure using specifi
{
    cd $location  # changes the current location to the location under the variable "location".
    strings "$1" | grep -Eo "([0-9a-fA-F]{2}:){5}[0-9a-fA-F]{2}" > $location/DataAnalysis/$file_name/Strings/Strings_Analyzed/Strings_mac_Addr
    cd $location/DataAnalysis/$file_name/Strings/Strings_Analyzed  # changes the current location to the location under the variables.

    # The following insering diffrent text and data inside the "Strings_MAC_Addr" file. when using ">>" its keep the current data that in the
    echo "  " >> Strings_MAC_Addr
    echo "[!] The Total Count Of MAC Addresses Is: $(cat Strings_mac_Addr |wc -l)" >> Strings_MAC_Addr
    echo "  " >> Strings_MAC_Addr
    echo "[*] The Following Displays All The Discovered MAC Addresses: " >> Strings_MAC_Addr
    echo "  " >> Strings_MAC_Addr
    cat Strings_mac_Addr >> Strings_MAC_Addr # inserting all the text inside the Strings_mac_Addr file and copy it inside Strings_MAC_Addr wit
    rm Strings_mac_Addr  # removing the file "Strings_mac_Addr"
}
MACAddr "$1"
```

The following inserting different text and data inside the "Strings_MAC_Addr" file. when using ">>" it keep the current data that in the file and not overwrite it.

strings "$1" | grep -Eo "([0-9a-fA-F]{2}:){5}[0-9a-fA-F]{2}" > $location/ DataAnalysis /$file_name/Strings/Strings_Analyzed/Strings_mac_Addr

The provided command extracts MAC addresses with the according pattern from the content of the file specified by the argument "$1". and save it into the location under the variables.

cat Strings_mac_Addr >> Strings_MAC_Addr

inserting all the text inside the Strings_mac_Addr file and copy it inside Strings_MAC_Addr without overwrite it.

rm Strings_mac_Add

removing the file "Strings_mac_Addr"

Function that uses the Strings command to extract information from a file, about File extensions information using specific keywords and then organizes and inserts the results into file."

```
function FileExt ()  # Function that using the Strings command to extract information from a file, about File extensions information using specific ke
{
    cd $location  # changes the current location to the location under the variable "location".
    strings "$1" | grep -iE '\.(exe|dll|zip|html|xml|gif|jpeg|jpg|png)$' > $location/DataAnalysis/$file_name/Strings/Strings_Analyzed/Strings_File_EXT
    cd $location/DataAnalysis/$file_name/Strings/Strings_Analyzed  # changes the current location to the location under the variables.

    # The following insering diffrent text and data inside the "Strings_File_Ext" file. when using ">>" its keep the current data that in the file and
    echo "  " >> Strings_File_Ext
    echo "[!] The Total Count Of Potential Files Within The Data That Was Found is: $(cat Strings_File_EXT |wc -l)" >> Strings_File_Ext
    echo "  " >> Strings_File_Ext
    echo " -  The Keyword 'exe' Indicates for Executables Files With The '.exe' Extension During The Analysis Process: $(cat Strings_File_EXT |grep -i
    echo " -  The Keyword 'dll' Indicates for Dynamic Link Library Type of Files With The '.dll' Extension During The Analysis Process: $(cat Strings_
    echo " -  The Keyword 'zip' Indicates for Compressed Files with the '.zip' Extension During the Analysis Process.: $(cat Strings_File_EXT |grep -i
    echo " -  The Keyword 'html' Indicates for Hypertext Markup Language Files With The '.html' Extension During The Analysis Process: $(cat Strings_F
    echo " -  The Keyword 'xml' Indicates for Configuration Files, Web Pages and Documents With The '.xml' Extension During The Analysis Process: $(ca
    echo " -  The Keyword 'gif' Indicates for Files Containing Graphics in GIF Image Format With The '.gif' Extension During The Analysis Process: $(ca
    echo " -  The Keyword 'jpeg' Indicates for Files Containing Graphics in JPEG Image Format With The '.jpeg' Extension During The Analysis Process:
    echo " -  The Keyword 'jpg' Indicates for Files Containing Graphics in JPG Image Format With The '.jpg' Extension During The Analysis Process: $(ca
    echo " -  The Keyword 'png' Indicates for Files Containing Graphics in PNG Image Format With The '.png' Extension During The Analysis Process: $(ca
    echo "  " >> Strings_File_Ext
    echo "[*] The Following Displays All The Files Extension Data: " >> Strings_File_Ext
    echo "  " >> Strings_File_Ext
    cat Strings_File_EXT >> Strings_File_Ext  # inserting all the text inside the Strings_File_EXT file and copy it inside Strings_File_Ext without ov
    rm Strings_File_EXT  # removing the file "Strings_File_EXT"
}
FileExt "$1"
```

The following inserting different text and data inside the "Strings_File_Ext" file.
when using ">>" its keep the current data that in the file and not overwrite it.

strings "$1" | grep -iE '\.(exe|dll|zip|html|xml|gif|jpeg|jpg|png)$' > $location/DataAnalysis/$file_name/Strings/Strings_Analyzed/Strings_File_EXT

The provided command extracts File extensions information with the according keywords from the content of the file specified by the argument "$1". and save it into the location under the variables.

cat Strings_File_EXT >> Strings_File_Ext

inserting all the text inside the Strings_File_EXT file and copy it inside Strings_File_Ext without overwrite it.

rm Strings_File_EXT

removing the file "Strings_File_EXT"

Function that uses the Strings command to extract information from a file, about Security information using specific keywords and then organizes and inserts the results into file.

```
function Security ()  # Function that using the Strings command to extract information from a file, about Security informat
{
    cd $location  # changes the current location to the location under the variable "location".
    strings "$1" | grep -Ei 'security|firewall|antivirus|malware' > $location/DataAnalysis/$file_name/Strings/Strings_Analy
    cd $location/DataAnalysis/$file_name/Strings/Strings_Analyzed   # changes the current location to the location under th

    # The following insering diffrent text and data inside the "Strings_Security" file. when using ">>" its keep the curren
    echo "  " >> Strings_Security
    echo "[*] This File Displays Security-Related Keywords That Have Been Scanned for Investigation Purposes." >> Strings_S
    echo "  " >> Strings_Security
    echo "[!] The Total Count Of Security-Related Keywords Is: $(cat Strings_SecuritY |wc -l)" >> Strings_Security
    echo "  " >> Strings_Security
    echo " -  The Keyword 'Security' Will Indicates for the Presence of the Keyword Security Within the Data: $(cat Strings
    echo " -  The Keyword 'firewall' Will Indicates for the Presence of the Keyword firewall Within the Data: $(cat Strings
    echo " -  The Keyword 'antivirus' Will Indicates for the Presence of the Keyword antivirus Within the Data: $(cat Strin
    echo " -  The Keyword 'malware' Will Indicates for the Presence of the Keyword malware Within the Data: $(cat Strings_S
    echo "  " >> Strings_Security
    echo "[*] The Following Displays All The Security-Related Keywords Data: " >> Strings_Security
    echo "  " >> Strings_Security
    cat Strings_SecuritY >> Strings_Security  # inserting all the text inside the Strings_SecuritY file and copy it inside
    rm Strings_SecuritY  # removing the file "Strings_SecuritY"
}
Security "$1"
```

The following is inserting different text and data inside the "Strings_Security" file. when using ">>" it keeps the current data that in the file and not overwrite it.

strings "$1" | grep -Ei 'security|firewall|antivirus|malware' > $location / DataAnalysis /$file_name/Strings/Strings_Analyzed/Strings_SecuritY

The provided command extracts Security information with the according keywords from the content of the file specified by the argument "$1".save it into the location under the variables.

cat Strings_SecuritY >> Strings_Security

inserting all the text inside the Strings_SecuritY file and copy it inside Strings_Security without overwrite it.

rm Strings_SecuritY

removing the file "Strings_SecuritY"

Function that uses the Strings command to extract information from a file, about Credentials information using specific keywords and then organizes and inserts the results into file."

```
function Credentials ()  # Function that using the Strings command to extract information from a file, about Credentials inf
{
    cd $location  # changes the current location to the location under the variable "location".
    strings "$1" | grep -Ei 'password|login|user|auth|credential' > $location/DataAnalysis/$file_name/Strings/Strings_Analyz
    cd $location/DataAnalysis/$file_name/Strings/Strings_Analyzed  # changes the current location to the location under the

    # The following insering diffrent text and data inside the "Strings_Security" file. when using ">>" its keep the current
    echo "  " >> Strings_Credentials
    echo "[*] This File Displays Authentication And Credentials Related Keywords That Have Been Scanned for Investigation Pu
    echo "  " >> Strings_Credentials
    echo "[!] The Total Count Of Authentication And Credentials Related Keywords Is: $(cat Strings_CredentialS |wc -l)" >> S
    echo "  " >> Strings_Credentials
    echo " -  The Keyword 'password' Will Indicates for the Presence of the Keyword password Within the Data: $(cat Strings_
    echo " -  The Keyword 'login' Will Indicates for the Presence of the Keyword login Within the Data: $(cat Strings_Creden
    echo " -  The Keyword 'user' Will Indicates for the Presence of the Keyword antivirus Within the Data: $(cat Strings_Cre
    echo " -  The Keyword 'auth' Will Indicates for the Presence of the Keyword auth Within the Data: $(cat Strings_Credenti
    echo " -  The Keyword 'credential' Will Indicates for the Presence of the Keyword credential Within the Data: $(cat Stri
    echo "  " >> Strings_Credentials
    echo "[*] The Following Displays All The Authentication And Credentials Related Keywords Data: " >> Strings_Credentials
    echo "  " >> Strings_Credentials
    cat Strings_CredentialS >> Strings_Credentials  # inserting all the text inside the Strings_CredentialS file and copy it
    rm Strings_CredentialS   # removing the file "Strings_CredentialS"
}
Credentials "$1"
```

The following is inserting different text and data inside the "Strings_Security" file. when using ">>" it keeps the current data that in the file and not overwrite it.

strings "$1" | grep -Ei 'password|login|user|auth|credential' > $location/DataAnalysis /$file_name/Strings/Strings_Analyzed/Strings_CredentialS

The provided command extracts Credentials information with the according keywords from the content of the file specified by the argument "$1". and save it into the location under the variables.

cat Strings_CredentialS >> Strings_Credentials

inserting all the text inside the Strings_CredentialS file and copy it inside Strings_Credentials without overwrite it.

rm Strings_CredentialS

removing the file "Strings_CredentialS"

Function that uses the Strings command to extract information from a file, about URL address's structure using specific pattern and then organizes and inserts the results into file.

```
function URLS ()  # Function that using the Strings command to extract information from a file, about URL addr
{
    cd $location  # changes the current location to the location under the variable "location".
    strings "$1"| grep -E 'http[s]?://[a-zA-Z0-9_\.-]+' > $location/DataAnalysis/$file_name/Strings/Strings_An
    cd $location/DataAnalysis/$file_name/Strings/Strings_Analyzed  # changes the current location to the locat

    # The following insering diffrent text and data inside the "Strings_Security" file. when using ">>" its ke
    echo "  " >> Strings_URLs_Web
    echo "[!] The Total Count Of URLs And Web-Related Information: Is: $(cat Strings_URLS_Web |wc -l) Potentia
    echo "  " >> Strings_URLs_Web
    echo "[*] The Following Displays All The URLs And Web-Related Data: " >> Strings_URLs_Web
    echo "  " >> Strings_URLs_Web
    cat Strings_URLS_Web >> Strings_URLs_Web  # inserting all the text inside the Strings_URLS_Web file and co
    rm Strings_URLS_Web   # removing the file "Strings_URLS_Web"
}
URLS "$1"
```

The following is inserting different text and data inside the "Strings_Security" file. when using ">>" it keeps the current data that in the file and not overwrite it.

strings "$1"| grep -E 'http[s]?://[a-zA-Z0-9_\.-]+' > $location/DataAnalysis /$file_name /Strings/Strings_Analyzed/Strings_URLS_Web

The provided command extracts URL addresses with the according pattern from the content of the file specified by the argument "$1". and save it into the location under the variables.

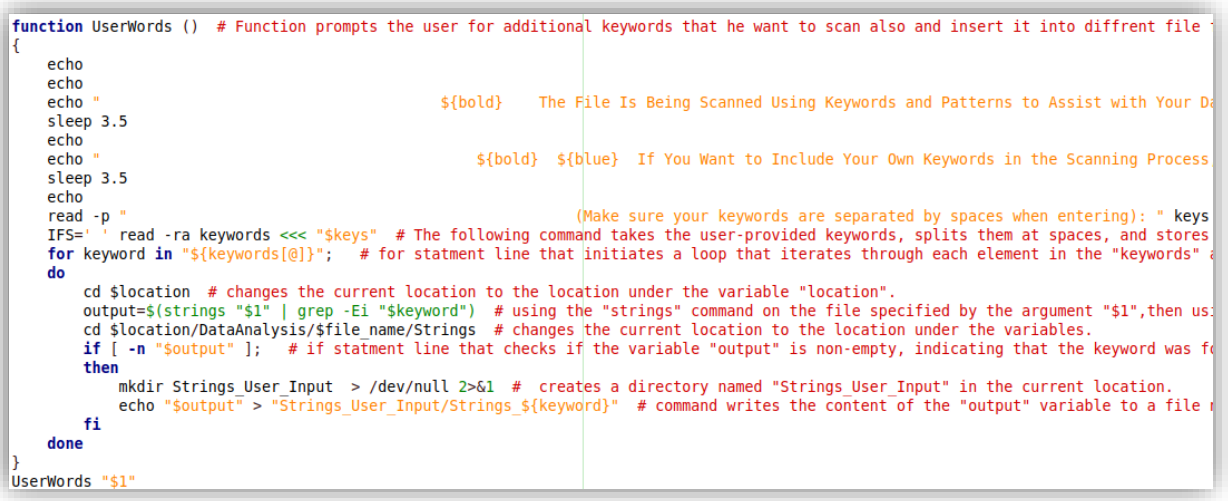
cat Strings_URLS_Web >> Strings_URLs_Web

inserting all the text inside the Strings_URLS_Web file and copy it inside Strings_URLs_Web without overwrite it.

rm Strings_URLS_Web

removing the file "Strings_URLS_Web"

Function prompts the user for additional keywords that he wants to scan also and insert it into different file for each keyword.



```
function UserWords ()  # Function prompts the user for additional keywords that he want to scan also and insert it into diffrent file
{
    echo
    echo
    echo "                                        ${bold}    The File Is Being Scanned Using Keywords and Patterns to Assist with Your Da
    sleep 3.5
    echo
    echo "                                            ${bold}  ${blue}  If You Want to Include Your Own Keywords in the Scanning Process
    sleep 3.5
    echo
    read -p "                                                 (Make sure your keywords are separated by spaces when entering): " keys
    IFS=' ' read -ra keywords <<< "$keys"  # The following command takes the user-provided keywords, splits them at spaces, and stores
    for keyword in "${keywords[@]}";   # for statment line that initiates a loop that iterates through each element in the "keywords" a
    do
        cd $location  # changes the current location to the location under the variable "location".
        output=$(strings "$1" | grep -Ei "$keyword")  # using the "strings" command on the file specified by the argument "$1",then usi
        cd $location/DataAnalysis/$file_name/Strings  # changes the current location to the location under the variables.
        if [ -n "$output" ];   # if statment line that checks if the variable "output" is non-empty, indicating that the keyword was fo
        then
            mkdir Strings_User_Input  > /dev/null 2>&1  #  creates a directory named "Strings_User_Input" in the current location.
            echo "$output" > "Strings_User_Input/Strings_${keyword}"  # command writes the content of the "output" variable to a file n
        fi
    done
}
UserWords "$1"
```

read -p " (Make sure your keywords are separated by spaces when entering): " keys

command prompts the user to enter keywords, displaying a custom message, and stores the input in the variable "keys."

IFS=' ' read -ra keywords <<< "$keys"

“ **IFS=' '** “ Sets the Internal Field Separator (IFS) to a space character. This affects how the shell splits words, and in this case, it's used to split the input string into an array of keywords.

“read -ra keywords” The read command reads input from the user or from a variable ("$keys" in this case). The options **-ra** instruct read to read the input into an array (keywords), splitting the input using the specified IFS (space in this case).

“<<< "$keys" The here-string syntax (<<<) is used to supply the content of the variable keys as input to the read command.

for keyword in "${keywords[@]}";

Inside the function there is a for statement line that initiates a loop that iterates through each element in the "keywords" array, allowing the script to perform actions for each individual keyword.

“for keyword” Initiates a for loop, where the variable keyword will iterate over the elements in the following array.

**“in "${keywords[@]}"** Specifies the array over which the loop will iterate. In this case, it's the array named keywords. The "${keywords[@]}" expands to the elements of the array.

For each word in the variable keywords (that the user has chosen), the script iterates through the **do** statement to perform a set of actions.

output=$(strings "$1" | grep -Ei "$keyword")

This command extracts readable text from a binary file specified by '$1' using the strings command. It then employs the grep command to search for occurrences of each user-input keyword. The results of this search are stored in the variable output for further use in the script.

if [-n "$output"];

if statment line that checks if the variable "output" is non-empty, indicating that the keyword was found in the file during the search. If there is a result moving to then statment.

inside the then statement the following commands will be executed…

mkdir Strings_User_Input

creates a directory named "Strings_User_Input" in the current location.

echo "$output" > "Strings_User_Input/Strings_${keyword}"

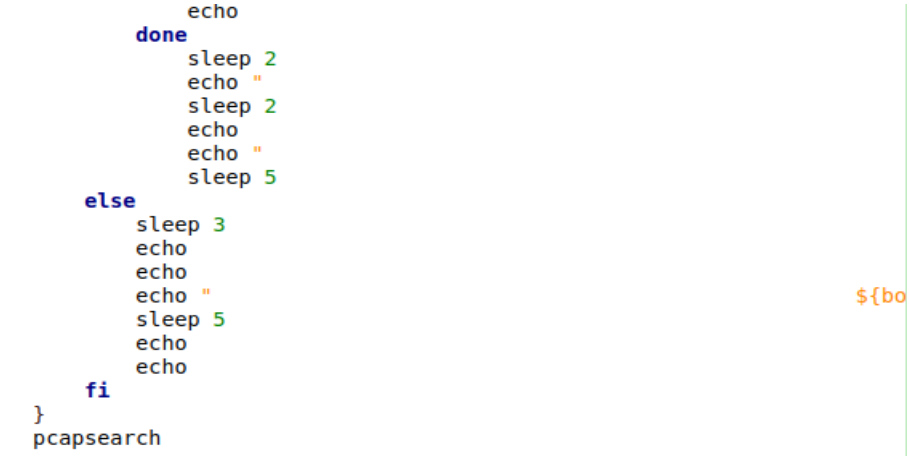
The command writes the content of the "output" variable to a file named "Strings_${keyword}" inside the "Strings_User_Input" directory. Each keyword corresponds to a separate file in the specified directory.
The file will saved as "Strings_(and the keyword the user choose)".

The end of Strings function…

Function to check all created files for the presence of ".pcap" files, if found notifying the user and saving the identified files into a separate directory.

```
function pcapsearch ()  # Function to check all created files for th
{
    sleep 3
    echo
    echo
    echo
    echo "                                                ${bold} Sc
    cd $location/DataAnalysis/$file_name
    pcapFiles=$(find . -type f -name "*.pcap" -size +0)  #  command

    if [ -n "$pcapFiles" ]  #  line checks if the variable "pcapFile
    then
        rm -rf Network_pcap_file  # command removes the directory na
        mkdir -p Network_pcap_file  # command creates a directory na
        cp -t Network_pcap_file $pcapFiles  > /dev/null 2>&1   # com
        cd Network_pcap_file  # entering the directory that was crea
        echo
        sleep 4.5
        echo
        echo "
        echo
        for i in $(ls)  # for loop that will do actions in all the f
        do
            sleep 3
            echo
            echo "
            sleep 1
            echo
            echo "
            sleep 1
            echo
            echo "
            sleep 1
            echo
            echo "
            sleep 1
            echo "-------------------------------------------------
```



pcapFiles=$(find . -type f -name "*.pcap" -size +0)

command searches for files with the ".pcap" extension, larger than 0 bytes, starting from the current directory (.) and stores the list of matching files in the "pcapFiles" variable.

if [-n "$pcapFiles"]

if statement that checks if the variable "pcapFiles" is non-empty, indicating that there are pcap files found during the search. then moving to "then" statment and if not, files was found moving to "else" statement.

Inside the then statement:

rm -rf Network_pcap_file

The command removes the directory named "Network_pcap_file"

mkdir -p Network_pcap_file

The command creates a directory named "Network_pcap_file" in the current location.
The -p option ensures that the command creates parent directories if they don't exist

cp -t Network_pcap_file $pcapFiles

The command copies the pcap files specified in the variable "pcapFiles" to the "Network_pcap_file" directory.

for i in $(ls)

A for loop is used with the ls command to display all the captured '.pcap' files, and display the user a text information about each captured pcap file.

For the size of the file the script used the command.
“ ls -lh $i |grep -i ".pcap" |awk '{print $5}'” - command lists details of the file specified by the variable $i, showing its size and other attributes. The output is then filtered using grep -i ".pcap" to extract lines containing the '.pcap' file extension in a case-insensitive manner. Finally, awk '{print $5}' is used to extract and display the size of the file from the output."

Inside the else statement there is a text to announce the user that no pcap files were found!

function for volatility analysis to extract information from a file, using volatility command and then organize and inserts the results into directories and files.

```
function VOL () # function for volatility analysis to extract information from a file, using volatility command and
{
    cd $location/DataAnalysis/$file_name   # changes the current location to the location under the variables.
    rm -rf Volatility_Analysis   # command removes the directory named "Volatility_Analysis"
    mkdir Volatility_Analysis  # command creates a directory named "Volatility_Analysis" in the current location.
    cd Volatility_Analysis  # entering the directory that was created named "Network_pcap_file".
    mkdir Commands_Data  # command creates a directory named "Commands_Data" in the current location.
```

rm -rf Volatility_Analysis

The command removes the directory named "Volatility_Analysis"

mkdir Volatility_Analysis

The command creates a directory named "Volatility_Analysis" in the current location.

cd Volatility_Analysis

entering the directory that was created named "Network_pcap_file".

mkdir Commands_Data

The command creates a directory named "Commands_Data" in the current location.

The following functions that are within the function VOL are using the Volatility command and extract information from a file, using different commands.

Function to remove the existing Volatility files, if they exist, and install Volatility to ensure proper execution of the commands.

```
function VOLINSTL ()  # Function to remove the existing Volatility files, if they exist, and install Volatility to
{
    clear
    echo
    echo "                                                ${bold}          Preparing and Installing the Volatil
    sleep 2
    echo
    echo
    echo "                                                Checking Whether the Provided File Is Compatible for
    cd $location
    rm -rf volatility_2.6_lin64_standalone*  > /dev/null 2>&1  # command removes the directory named "volatility_2
    rm vol.py  > /dev/null 2>&1  # command removes the file named "vol.py".
    wget http://downloads.volatilityfoundation.org/releases/2.6/volatility_2.6_lin64_standalone.zip > /dev/null 2>
    unzip volatility_2.6_lin64_standalone.zip > /dev/null 2>&1 && cd volatility_2.6_lin64_standalone  # unzip and
    mv volatility_2.6_lin64_standalone vol.py  # The "nv"  command renames the file "volatility_2.6_lin64_standalo
    cp vol.py $location  # the following command copies the "vol.py" executable file to the location under the "lo
    cd $location  # changes the current location to the location under the variable "location".
    rm -rf volatility_2.6_lin64_standalone*  # command removes the directory named "volatility_2.6_lin64_standalon
}
VOLINSTL "$1"
```

rm -rf volatility_2.6_lin64_standalone*

The command removes any directories named "volatility_2.6_lin64_standalone*"

rm vol.py

The command removes the file named "vol.py".

wget http://downloads.volatilityfoundation.org/releases/2.6/volatility_2.6_lin64_standalone.zip

using the "wget" command that downloads from a URL address a package into linux.
the following command downloads the Volatility framework version 2.6 for Linux in a zip file from the specified URL.

unzip volatility_2.6_lin64_standalone.zip && cd volatility_2.6_lin64_standalone

The command unzip and extracts the contents of the "volatility_2.6_lin64_standalone.zip" file. and then enter the extracted directory.

mv volatility_2.6_lin64_standalone vol.py

The "nv" command renames the file "volatility_2.6_lin64_standalone" to "vol.py".

cp vol.py $location

The following command copies the "vol.py" executable file to the location under the "location" variable.

rm -rf volatility_2.6_lin64_standalone*

The command removes the directory named "volatility_2.6_lin64_standalone*"

Function that analysis the file using volatility command with different function while using different keywords and commands.

```
function VOLAnalysis ()  # Function that analysis
{
    sleep 2
    echo
    echo "${bold}${blue}==========================
    echo
```

Function to extract information with volatility about the image profile of the file

```
function imageinfo () # Function to extract information with volatility about the image profile of the file
{
    cd $location   # changes the current location to the location under the variable "location".
    echo "$(./vol.py -f "$1" imageinfo 2>&1)" > $location/DataAnalysis/$file_name/Volatility_Analysis/Commands_Data/Vol_Image
    image=$(./vol.py -f "$1" imageinfo 2>&1 |grep -i "Suggested Profile"|awk -F: '{print $2}'|awk '{print $1}'|sed 's/,//g')
    sleep 1
    echo
    echo "                                                    ${bold} The Name Of The Memory Profile Image
    sleep 2
    echo
    echo
    echo "                                                  ${bold}${green}              $image       ${no
}
imageinfo "$1"
```

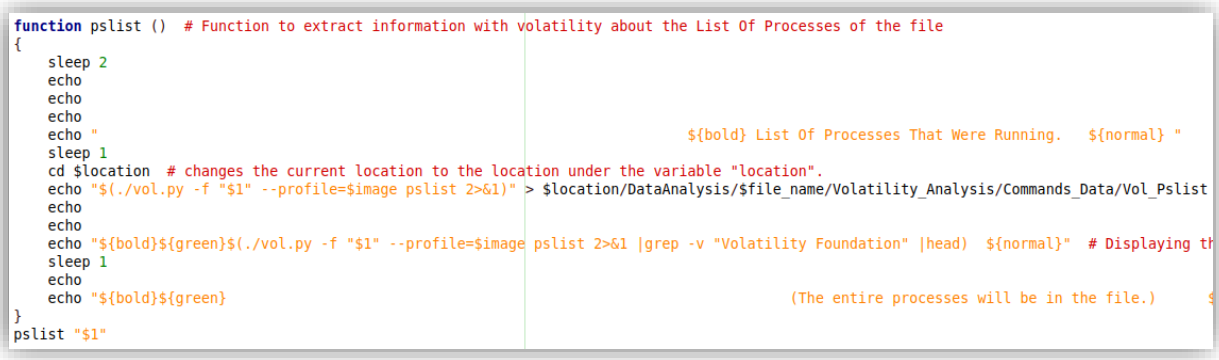
echo "$(./vol.py -f "$1" imageinfo 2>&1)" > $location/DataAnalysis/$file_name/Volatility_Analysis/Commands_Data/Vol_Imageinfo

The command executes the Volatility command imageinfo on the specified memory dump file, captures the output, and writes it to a file named "Vol_Imageinfo" within the specified directory path.

image=$(./vol.py -f "$1" imageinfo 2>&1 |grep -i "Suggested Profile"|awk -F: '{print $2}'|awk '{print $1}'|sed 's/,//g')

The command extracts the suggested profile from the output of the Volatility imageinfo command applied to the specified memory dump file and stores it in the variable "image".

Function to extract information with volatility about the List Of Processes of the file



```
function pslist ()  # Function to extract information with volatility about the List Of Processes of the file
{
    sleep 2
    echo
    echo
    echo
    echo "                                                                ${bold} List Of Processes That Were Running.   ${normal} "
    sleep 1
    cd $location  # changes the current location to the location under the variable "location".
    echo "$(./vol.py -f "$1" --profile=$image pslist 2>&1)" > $location/DataAnalysis/$file_name/Volatility_Analysis/Commands_Data/Vol_Pslist
    echo
    echo
    echo "${bold}${green}$(./vol.py -f "$1" --profile=$image pslist 2>&1 |grep -v "Volatility Foundation" |head)  ${normal}"  # Displaying th
    sleep 1
    echo
    echo "${bold}${green}                                                                (The entire processes will be in the file.)      $
}
pslist "$1"
```

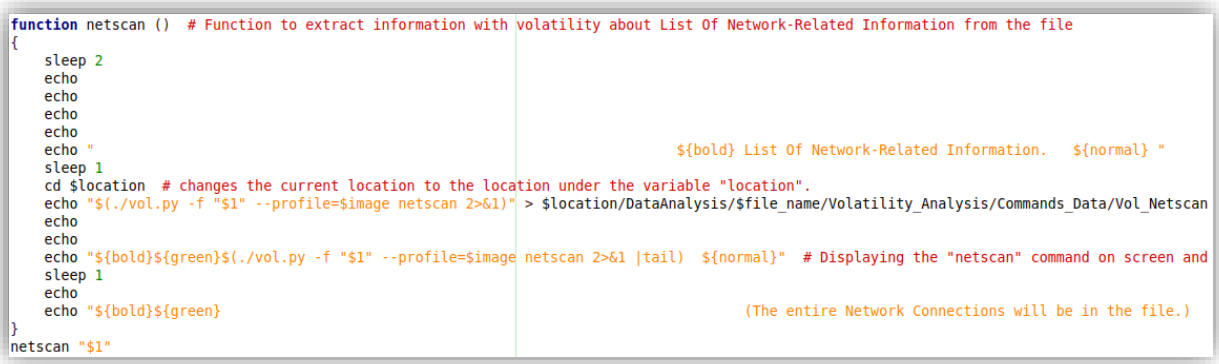
echo "$(./vol.py -f "$1" --profile=$image pslist 2>&1)" > $location/DataAnalysis/$file_name/Volatility_Analysis/Commands_Data/Vol_Pslist

The command executes the Volatility pslist command using the suggested profile obtained previously, to display list Of Processes That Were Running. captures the output and writes it to a file named "Vol_Pslist" within the specified directory path.

echo "${bold}${green}$(./vol.py -f "$1" --profile=$image pslist 2>&1 |grep -v "Volatility Foundation" |head) ${normal}"

Displaying the "pslist" command on screen and prints the first few lines of the result on the screen

Function to extract information with volatility about List Of Network-Related Information from the file



```
function netscan ()  # Function to extract information with volatility about List Of Network-Related Information from the file
{
    sleep 2
    echo
    echo
    echo
    echo
    echo "                                                              ${bold} List Of Network-Related Information.   ${normal} "
    sleep 1
    cd $location  # changes the current location to the location under the variable "location".
    echo "$(./vol.py -f "$1" --profile=$image netscan 2>&1)" > $location/DataAnalysis/$file_name/Volatility_Analysis/Commands_Data/Vol_Netscan
    echo
    echo
    echo "${bold}${green}$(./vol.py -f "$1" --profile=$image netscan 2>&1 |tail)  ${normal}"  # Displaying the "netscan" command on screen and
    sleep 1
    echo
    echo "${bold}${green}                                                                (The entire Network Connections will be in the file.)
}
netscan "$1"
```

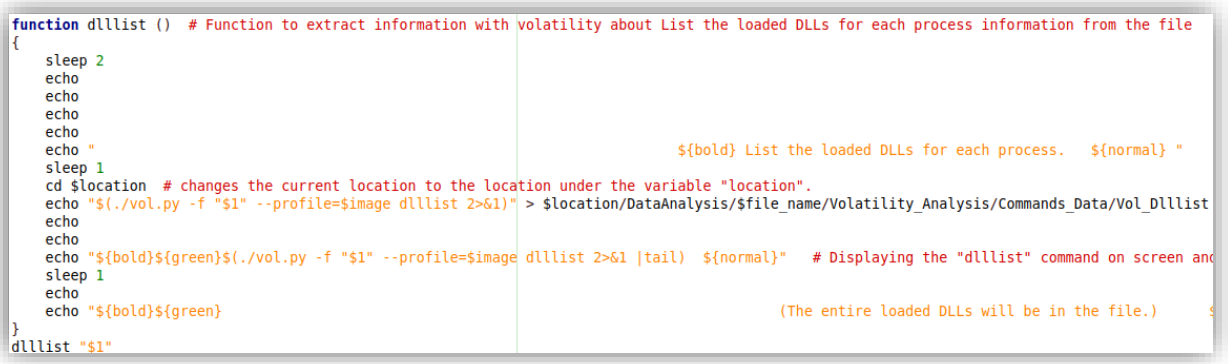
echo "$(./vol.py -f "$1" --profile=$image netscan 2>&1)" > $location/DataAnalysis/$file_name/Volatility_Analysis/Commands_Data/Vol_Netscan

The command executes the Volatility "netscan" command using the suggested profile, to display List Of Network-Related Information. captures the output, and writes it to a file named "Vol_Netscan" within the specified directory path.

echo "${bold}${green}$(./vol.py -f "$1" --profile=$image netscan 2>&1 |tail) ${normal}"

Displaying the "netscan" command on screen and prints the last few lines of the result on the screen

Function to extract information with volatility about List the loaded DLLs for each process information from the file



```
function dlllist ()  # Function to extract information with volatility about List the loaded DLLs for each process information from the file
{
    sleep 2
    echo
    echo
    echo
    echo
    echo "                                                                    ${bold} List the loaded DLLs for each process.   ${normal} "
    sleep 1
    cd $location  # changes the current location to the location under the variable "location".
    echo "$(./vol.py -f "$1" --profile=$image dlllist 2>&1)" > $location/DataAnalysis/$file_name/Volatility_Analysis/Commands_Data/Vol_Dlllist
    echo
    echo
    echo "${bold}${green}$(./vol.py -f "$1" --profile=$image dlllist 2>&1 |tail)  ${normal}"   # Displaying the "dlllist" command on screen and
    sleep 1
    echo
    echo "${bold}${green}                                                                    (The entire loaded DLLs will be in the file.)      $
}
dlllist "$1"
```

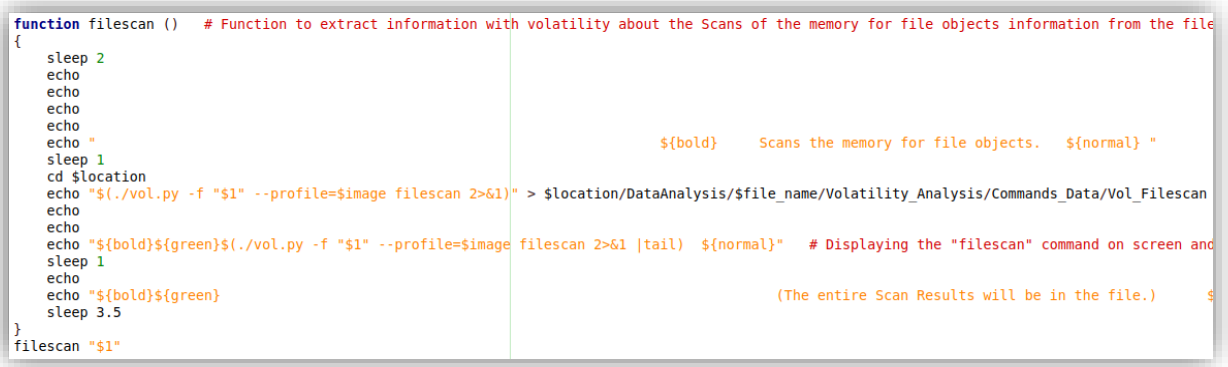
echo "$(./vol.py -f "$1" --profile=$image dlllist 2>&1)" > $location/DataAnalysis/$file_name/Volatility_Analysis/Commands_Data/Vol_Dlllist

The command executes the Volatility "dlllist" command using the suggested profile, to display List the loaded DLLs for each process. captures the output and writes it to a file named "Vol_Dlllist" within the specified directory path.

echo "${bold}${green}$(./vol.py -f "$1" --profile=$image dlllist 2>&1 |tail) ${normal}"

Displaying the "dlllist" command on screen and prints the last few lines of the result on the screen.

Function to extract information with volatility about the Scans of the memory for file objects information from the file



```
function filescan ()   # Function to extract information with volatility about the Scans of the memory for file objects information from the file
{
    sleep 2
    echo
    echo
    echo
    echo
    echo "                                                                ${bold}     Scans the memory for file objects.   ${normal} "
    sleep 1
    cd $location
    echo "$(./vol.py -f "$1" --profile=$image filescan 2>&1)" > $location/DataAnalysis/$file_name/Volatility_Analysis/Commands_Data/Vol_Filescan
    echo
    echo
    echo "${bold}${green}$(./vol.py -f "$1" --profile=$image filescan 2>&1 |tail)  ${normal}"   # Displaying the "filescan" command on screen and
    sleep 1
    echo
    echo "${bold}${green}                                                                    (The entire Scan Results will be in the file.)      $
    sleep 3.5
}
filescan "$1"
```

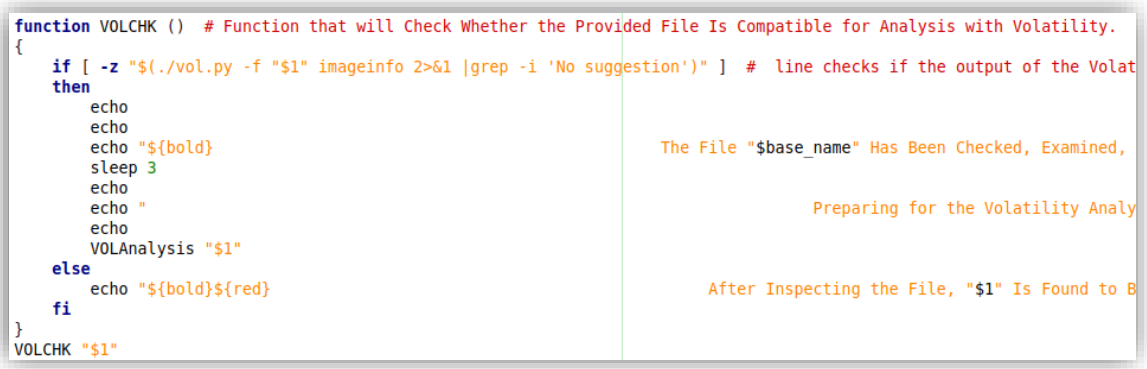
echo "$(./vol.py -f "$1" --profile=$image filescan 2>&1)" > $location/DataAnalysis/$file_name/Volatility_Analysis/Commands_Data/Vol_Filescan

The command executes the Volatility filescan command using the suggested profile, Scans the memory for file objects. captures the output and writes it to a file named "Vol_Filescan" within the specified directory path.

echo "${bold}${green}$(./vol.py -f "$1" --profile=$image filescan 2>&1 |tail) ${normal}

Displaying the "filescan" command on screen and prints the last few lines of the result on the screen.

Function that will Check Whether the Provided File Is Compatible for Analysis with Volatility.



```
function VOLCHK ()  # Function that will Check Whether the Provided File Is Compatible for Analysis with Volatility.
{
    if [ -z "$(./vol.py -f "$1" imageinfo 2>&1 |grep -i 'No suggestion')" ]  #  line checks if the output of the Volat
    then
        echo
        echo
        echo "${bold}                                          The File "$base_name" Has Been Checked, Examined,
        sleep 3
        echo
        echo "                                                          Preparing for the Volatility Analy
        echo
        VOLAnalysis "$1"
    else
        echo "${bold}${red}                                       After Inspecting the File, "$1" Is Found to B
    fi
}
VOLCHK "$1"
```

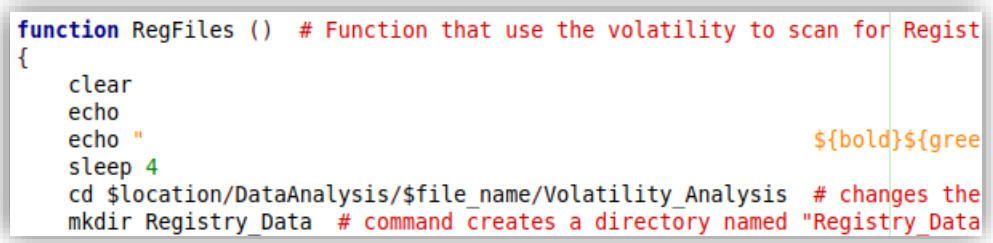
if [-z "$(./vol.py -f "$1" imageinfo 2>&1 |grep -i 'No suggestion')"]

if statement that checks if the output of the Volatility imageinfo command on the specified memory dump file does not contain the phrase "No suggestion," indicating that a suggested profile is available. then moving to "then" statment if it's not contain moving to "else" statment.

In the statement after displaying text on screen if the file is compatible to use or not.

End of VOL function…

Function that uses the volatility to scan for Registry Files information within the file while using different keywords.



```
function RegFiles ()  # Function that use the volatility to scan for Regist
{
    clear
    echo
    echo "                                                    ${bold}${gree
    sleep 4
    cd $location/DataAnalysis/$file_name/Volatility_Analysis  # changes the
    mkdir Registry_Data  # command creates a directory named "Registry_Data
```

cd $location/DataAnalysis/$file_name/Volatility_Analysis

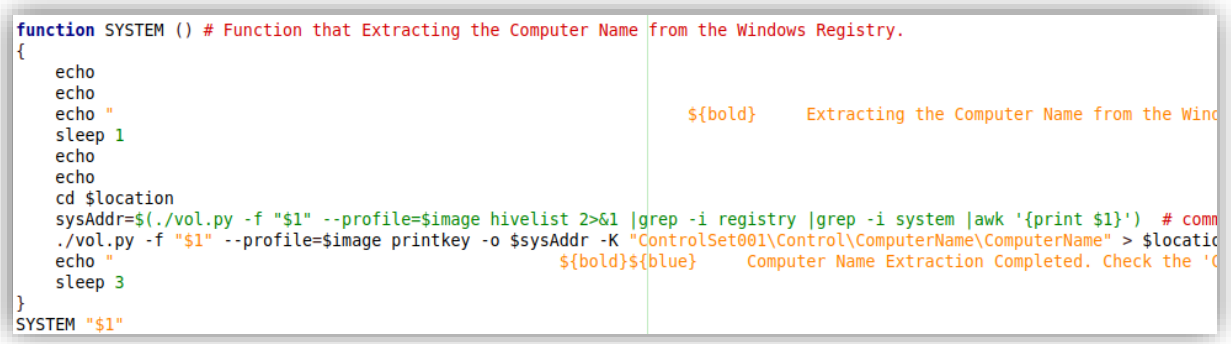
The command changes the current location to the location under the variables.

mkdir Registry_Data

The command creates a directory named "Registry_Data" in the current location.

The following functions that are within the function RegFiles are using the Volatility command and extract information from a file about the registry data, using different commands.

Function that Extracting the Computer Name from the Windows Registry.



```
function SYSTEM () # Function that Extracting the Computer Name from the Windows Registry.
{
    echo
    echo
    echo "                                                    ${bold}     Extracting the Computer Name from the Winc
    sleep 1
    echo
    echo
    cd $location
    sysAddr=$(./vol.py -f "$1" --profile=$image hivelist 2>&1 |grep -i registry |grep -i system |awk '{print $1}')  # comm
    ./vol.py -f "$1" --profile=$image printkey -o $sysAddr -K "ControlSet001\Control\ComputerName\ComputerName" > $locatic
    echo "                                                ${bold}${blue}     Computer Name Extraction Completed. Check the '
    sleep 3
}
SYSTEM "$1"
```

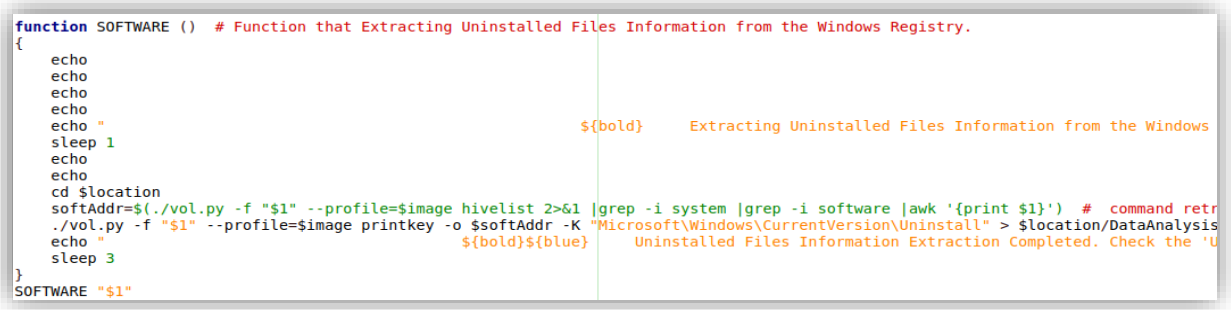
sysAddr=$(./vol.py -f "$1" --profile=$image hivelist 2>&1 |grep -i registry |grep -i system |awk '{print $1}')

The command retrieves the physical address of the Windows registry hive for the "System" file using Volatility and stores it in the variable "sysAddr."

./vol.py -f "$1" --profile=$image printkey -o $sysAddr -K "ControlSet001\Control \ComputerName\ComputerName" > $location/DataAnalysis/$file_name /Volatility_Analysis/Registry_Data/ComputerName

The command uses Volatility to print the registry key "ControlSet001\Control\ComputerName \ComputerName" from the specified memory dump file, and the output is saved to a file named "ComputerName" within the specified directory path for further analysis.

Function that Extracting Uninstalled Files Information from the Windows Registry.



```
function SOFTWARE ()  # Function that Extracting Uninstalled Files Information from the Windows Registry.
{
    echo
    echo
    echo
    echo
    echo "                                                ${bold}     Extracting Uninstalled Files Information from the Windows
    sleep 1
    echo
    echo
    cd $location
    softAddr=$(./vol.py -f "$1" --profile=$image hivelist 2>&1 |grep -i system |grep -i software |awk '{print $1}')  #  command retr
    ./vol.py -f "$1" --profile=$image printkey -o $softAddr -K "Microsoft\Windows\CurrentVersion\Uninstall" > $location/DataAnalysis
    echo "                                          ${bold}${blue}     Uninstalled Files Information Extraction Completed. Check the 'U
    sleep 3
}
SOFTWARE "$1"
```

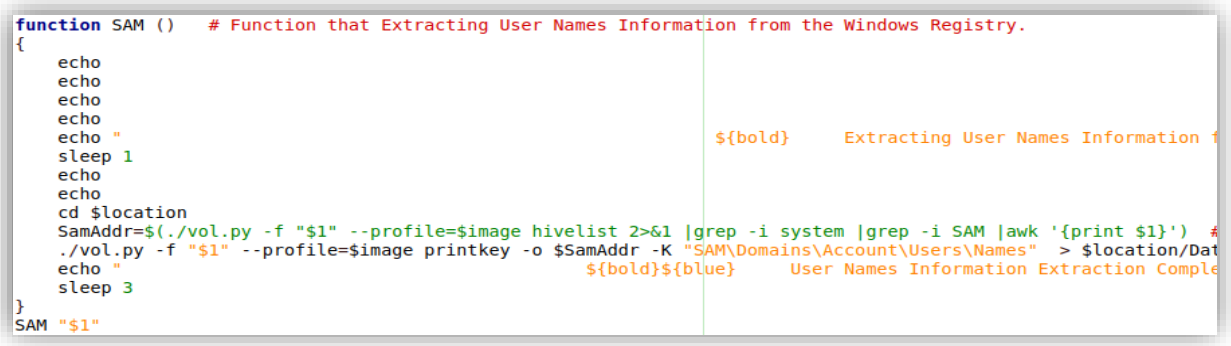
softAddr=$(./vol.py -f "$1" --profile=$image hivelist 2>&1 |grep -i system |grep -i software |awk '{print $1}')

The command retrieves the physical address of the Windows registry hive for the "Software" file using Volatility and stores it in the variable "softAddr."

./vol.py -f "$1" --profile=$image printkey -o $softAddr -K "Microsoft\Windows \CurrentVersion\Uninstall" > $location/DataAnalysis/$file_name/Volatility_Analysis /Registry_Data/Uninstall

The command uses Volatility to print the registry key "Microsoft\Windows\CurrentVersion\Uninstall" from the specified memory dump file, and the output is saved to a file named "Uninstall" within the specified directory path for further analysis.

Function that Extracting User Names Information from the Windows Registry.



```
function SAM ()   # Function that Extracting User Names Information from the Windows Registry.
{
    echo
    echo
    echo
    echo
    echo "                                                    ${bold}     Extracting User Names Information f
    sleep 1
    echo
    echo
    cd $location
    SamAddr=$(./vol.py -f "$1" --profile=$image hivelist 2>&1 |grep -i system |grep -i SAM |awk '{print $1}')  #
    ./vol.py -f "$1" --profile=$image printkey -o $SamAddr -K "SAM\Domains\Account\Users\Names"  > $location/Dat
    echo "                                                ${bold}${blue}     User Names Information Extraction Comple
    sleep 3
}
SAM "$1"
```

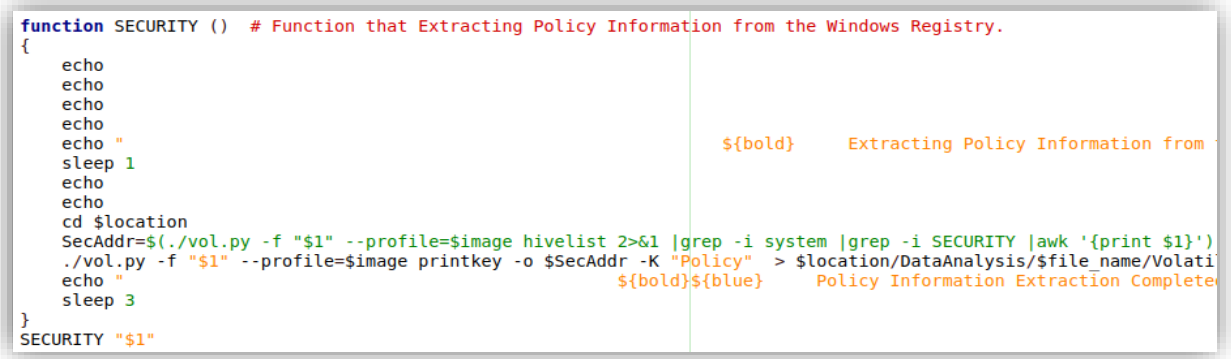
SamAddr=$(./vol.py -f "$1" --profile=$image hivelist 2>&1 |grep -i system |grep -i SAM |awk '{print $1}')

The command retrieves the physical address of the Windows registry hive for the "SAM" file using Volatility and stores it in the variable "SamAddr."

./vol.py -f "$1" --profile=$image printkey -o $SamAddr -K "SAM\Domains\Account\Users\Names" > $location/DataAnalysis/$file_name/Volatility_Analysis/Registry_Data/UserNames

The command utilizes Volatility to print the registry key "SAM\Domains\Account\Users\Names" from the specified memory dump file. The output is then saved to a file named "UserNames" within the specified directory path for further analysis.

Function that Extracting Policy Information from the Windows Registry.



```
function SECURITY ()  # Function that Extracting Policy Information from the Windows Registry.
{
    echo
    echo
    echo
    echo
    echo "                                                    ${bold}     Extracting Policy Information from
    sleep 1
    echo
    echo
    cd $location
    SecAddr=$(./vol.py -f "$1" --profile=$image hivelist 2>&1 |grep -i system |grep -i SECURITY |awk '{print $1}')
    ./vol.py -f "$1" --profile=$image printkey -o $SecAddr -K "Policy"  > $location/DataAnalysis/$file_name/Volati
    echo "                                                ${bold}${blue}     Policy Information Extraction Complete
    sleep 3
}
SECURITY "$1"
```

SecAddr=$(./vol.py -f "$1" --profile=$image hivelist 2>&1 |grep -i system |grep -i SECURITY |awk '{print $1}')

The command retrieves the physical address of the Windows registry hive for the "SECURITY" file using Volatility and stores it in the variable "SecAddr."

./vol.py -f "$1" --profile=$image printkey -o $SecAddr -K "Policy" > $location/DataAnalysis/$file_name/Volatility_Analysis/Registry_Data/Policy

command employs Volatility to print the registry key "Policy" from the specified memory dump file. The output is then saved to a file named "Policy" within the specified directory path for further analysis.

Function to ask the user for a file to analyze, verifying its existence, and if the file exists, calling the function that analyzes the file.

```
function FILE ()  # Function to ask the user for a file to analyzed , verifying its existence, and if the
{
    echo
    echo
    echo "${bold}[*] Preparing The Forensic Tools To Analyze The Selected File...${normal}"

    while true  # while loop that will ask the user for the file ,verifying its existence, and if the fil
    do
        sleep 1.5
        echo
        echo
        read -p " >  Please Enter The Name/Full Path Of The File You Want To Analyze: " file_name_path  #
        if [ -f "$file_name_path" ]   #  line checks if the file specified by the variable "file_name_pat
        then
            sleep 2.5
            echo
            echo
            echo "${bold}[✓] File Is Exists, Progressing... ${normal}  "
            ANALYZE "$file_name_path"  # This command calling the function "ANALYZE" with the specified f
            break  # The break statement is used within the loop to exit the loop prematurely, jumping to
        else
            echo
            echo
            sleep 2
            echo "${bold}[!] File Not Found, Please Try Again... ${normal} "
            sleep 2
        fi
    done
}
```

while true

while loop that will ask the user for the file, verifying its existence, and if the file exists, calling the function that analyzes the file. if not loop until a valid file is input.

read -p " > Please Enter The Name/Full Path Of The File You Want To Analyze: " file_name_path

The command prompts the user to enter the name or full path of the file they want to analyze and stores the input in the variable "file_name_path."

if [-f "$file_name_path"]

The if statement checks if the file specified by the variable "file_name_path" exists. If the file exists, move to "then" statment otherwise move to else statment.

Command inside the then statement :

ANALYZE "$file_name_path"

This command calling the function "ANALYZE" with the specified file path ("$file_name_path") as an argument, initiating the analysis process for the specified file.

Break

The break statement is used within the loop to exit the loop prematurely, jumping to the code that follows the loop.

Function to ask the user for a Directory to analyze, verifying its existence, and if the Directory exists, calling the function that analyzes the files within Directory.

```
function DIRECTORY ()  # Function to ask the user for a Directory to analyzed , verifying its existe
{
    echo
    echo
    echo "${bold}[*] Preparing The Forensic Tools To Analyze The Selected Directory...${normal}"

    while true  # while loop that will ask the user for the Directory ,verifying its existence, and
    do
        sleep 1.5
        echo
        read -p " > Enter The Name/Full Path Of The Directory You Want To Analyze: " dir_name_path
        if [ -d "$dir_name_path" ];   # line checks if the Directory specified by the variable "dir_
        then
            sleep 2.5
            echo
            echo
            echo "${bold}[✓] Directory Is Exists, Progressing... ${normal}  "
            for file in "$dir_name_path"/*;   # line initiates a loop that iterates over each file i
            do
                ANALYZE "$file"   # This command calling the function "ANALYZE" with the specified f
            done
            break   # The break statement is used within the loop to exit the loop prematurely, jump
        else
            echo
            echo
            sleep 2
            echo "${bold}[!] Directory Not Found, Please Try Again... ${normal} "
            sleep 2
            echo
        fi
    done
}
```

while true

while loop that will ask the user for the Directory, verifying its existence, and if the Directory exists, calling the function that analyzes the file. if not loop until a valid Directory is input.

read -p " > Enter The Name/Full Path Of The Directory You Want To Analyze: " dir_name_path

The command prompts the user to enter the name or full path of the directory they want to analyze and stores the input in the variable "dir_name_path."

if [-d "$dir_name_path"];

if statement that checks if the Directory specified by the variable "dir_name_path" exists. If the Directory exists, move to "then" statement otherwise move to else statement.

for file in "$dir_name_path"/*;

The for loop initiates a loop that iterates over each file in the specified directory (represented by the variable "dir_name_path"), assigning each file to the variable "file" for further processing within the loop.

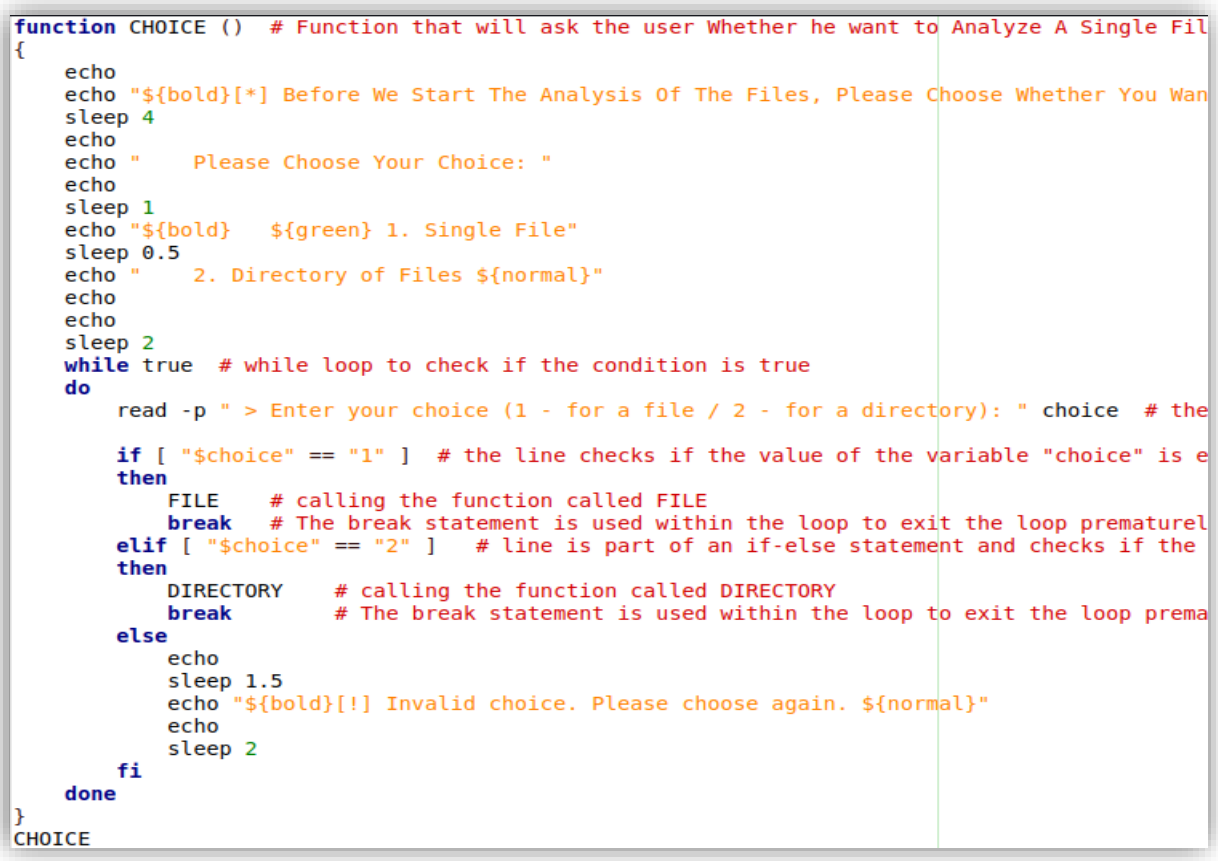
ANALYZE "$file"

This command calls the function "ANALYZE" with the specified file path ("$file") as an argument, initiating the analysis process for each file in the specified directory.

Break

The break statement is used within the loop to exit the loop prematurely, jumping to the code that follows the loop.

Function that will ask the user Whether he wants to Analyze a Single File or a Directory of Files and then calling the needed functions to continue.



```
function CHOICE ()  # Function that will ask the user Whether he want to Analyze A Single Fil
{
    echo
    echo "${bold}[*] Before We Start The Analysis Of The Files, Please Choose Whether You Wan
    sleep 4
    echo
    echo "    Please Choose Your Choice: "
    echo
    sleep 1
    echo "${bold}   ${green} 1. Single File"
    sleep 0.5
    echo "    2. Directory of Files ${normal}"
    echo
    echo
    sleep 2
    while true  # while loop to check if the condition is true
    do
        read -p " > Enter your choice (1 - for a file / 2 - for a directory): " choice  # the

        if [ "$choice" == "1" ]  # the line checks if the value of the variable "choice" is e
        then
            FILE    # calling the function called FILE
            break   # The break statement is used within the loop to exit the loop prematurel
        elif [ "$choice" == "2" ]   # line is part of an if-else statement and checks if the
        then
            DIRECTORY    # calling the function called DIRECTORY
            break        # The break statement is used within the loop to exit the loop prema
        else
            echo
            sleep 1.5
            echo "${bold}[!] Invalid choice. Please choose again. ${normal}"
            echo
            sleep 2
        fi
    done
}
CHOICE
```

while true

while loop to check if the condition is true

read -p " > Enter your choice (1 - for a file / 2 - for a directory): " choice

The command prompts the user to enter their choice (either 1 for a file or 2 for a directory) and stores the input in the variable "choice" for further use in the script.

if ["$choice" == "1"]

The line checks if the value of the variable "choice" is equal to the string "1".

Commands inside the then statement…

FILE

calling the function called FILE

break

The break statement is used within the loop to exit the loop prematurely, jumping to the code that follows the loop.

elif ["$choice" == "2"]

The elif statement is part of an if-else statement and checks if the value of the variable "choice" is equal to the string "2" move to then statement. If the previous conditions in the if statement were not true.

Commands inside the then statement…

DIRECTORY

calling the function called DIRECTORY

break

The break statement is used within the loop to exit the loop prematurely, jumping to the code that follows the loop.

Inside the else statement if none of the conditions was true alerting the user invalid choice!

Function to initiate the compression process to create a zip archive of the analysis data.

```
function ZIP () #  Function to initiates the compression process to c
{
        echo
        echo
        echo "                                                    ${bold}$
        sleep 4
        echo
        echo "                                                    ${bold}$
        sleep 1
        echo "                                                    ${bold}$
        cd $location  # changes the current location to the location
        zip -r DataAnalysis.zip DataAnalysis > /dev/null 2>&1  # comp
        cd $location  # changes the current location to the location
        rm -rf DataAnalysis > /dev/null 2>&1  # The command rm -rf Da
        echo
        echo
        echo "                                                    ${bold}$
        echo
}
ZIP
```

zip -r DataAnalysis.zip DataAnalysis

compress the contents of the "DataAnalysis" directory into a zip file named "DataAnalysis.zip, The "-r" flag it instructs the zip command to recursively include all files and subdirectories within the specified directory.

rm -rf DataAnalysis

The command rm -rf DataAnalysis is used to remove the directory named "DataAnalysis" and its contents forcefully and recursively.

Function for the end of the script that shows a diagram about the locations of all directories and files that were created are stored at.

```
function END ()  # Function for the end of the script that shows a diagram about the locations of all directories and files t
{
    clear
    echo
    echo "                                                    ${red}${bold} The Following Diagram Displays Where All the Da
    sleep 5
    echo
    echo
    echo "${bold}                          |=========================>${blue}${bold} Exiftool ${normal}${bold}=============
    sleep 0.1
    echo "${bold}                          |  "
    sleep 0.1
    echo "${bold}                          |               ${bold}   "
    sleep 0.1
    echo "${bold}                          |                                                  |========> ${green}Filtered_dat
    sleep 0.1
    echo "${bold}                          |                                                  |                          "
    sleep 0.1
    echo "${bold}                          |                                                  |                          "
    sleep 0.1
    echo "${bold}                          |=========================>${blue}${bold} Bulk_extractor ${normal}${bold} ======
    sleep 0.1
    echo "${bold}                          |                                                  |                          "
    sleep 0.1
    echo "${bold}                          |                                                  |                          "
    sleep 0.1
    echo "${bold}                          |                                                  |========> ${green}Full_Data
    sleep 0.1
    echo "${bold}                          |  "
    sleep 0.1
    echo "${bold}                          |  "
    sleep 0.1
    echo "${bold}                          |=========================>${blue}${bold} Foremost ${normal}${bold}=============
    sleep 0.1
    echo "${blue}${bold} DataAnalysis ${normal}${bold}===========>${blue}${bold} Data_(Name of the file)${normal}${bold}==|
    sleep 0.1
    echo "${bold}                                  |=========================>${blue}${bold} Strings ${normal}${bold} ======
    sleep 0.1
    echo "${bold}                          |                                                |                          "
    sleep 0.1
    echo "${bold}                          |                                                |========> ${green}Strings_User_
    sleep 0.1
    echo "${bold}                          |  "
    sleep 0.1
    echo "${bold}                          |  "
    sleep 0.1
    echo "${bold}                          |=========================>${blue}${bold} Network_pcap_file ${normal}${bold}=====
    sleep 0.1
    echo "${bold}                          |  "
    sleep 0.1
    echo "${bold}                          |  "
    sleep 0.1
    echo "${bold}                          |=========================>${blue}${bold} Binwalk ${normal}${bold}==============
    sleep 0.1
    echo "${bold}                          |                           ${bold}   "
    sleep 0.1
    echo "${bold}                          |  "
    sleep 0.1
    echo "${bold}                          |                                                |========> ${green}Commands_Data
    sleep 0.1
    echo "${bold}                          |                                                |                          "
    sleep 0.1
    echo "${bold}                          |=========================>${blue}${bold} Volatility_Analysis${normal}${bold} ===
    sleep 0.1
    echo "${bold}                                                                           |                          "
    sleep 0.1
    echo "${bold}                                                                           |========> ${green}Registry_Data
    echo
    sleep 5
    echo "                                                     ${bold}                  Enjoy your investigation!
    sleep 3
}
END "$1"
```

You've Reach the End of the Document

Hope I was able to explain the project more clearly

Have a Successful Forensic Exploration!