Project

Windows Forensics

Israel Gatterer

Objective

Creating a bash script that maps network devices for ports, services, and vulnerabilities.

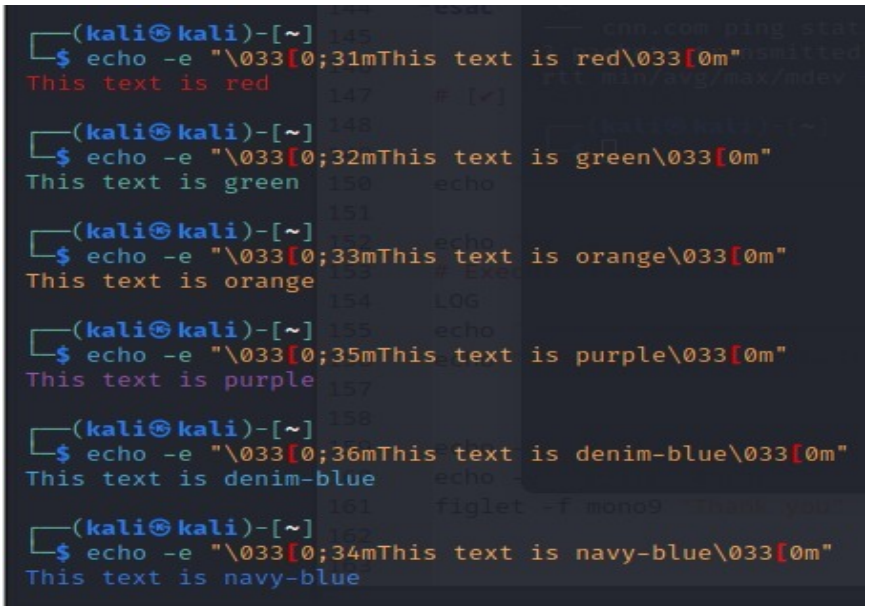
Windows Forensics Project

Installing figlet, volatility & foremost

Figlet is a tool for creating ASCII art text banners. The function presents if (**if**) it is already installed in the system. If it is, it displays a message saying so (**then**) . If it is not (**else**), it displays a message indicating that it will be installed and then proceed to install figlet using the apt-get package manager with output redirected to /dev/null to suppress any output. The "**fi**" statement is used to close the "**if-then-else**" block.

The script assumes that the user running it has sufficient privileges to use sudo to install packages. Also, if the system does not use apt-get as the package manager, this script may not work as expected.
Overall, the purpose of this script is to add some visual flair to the project by installing Figlet, which is not necessary for the actual project.

echo -e "\033[0;32m…….. \033[0m" - This is a Bash command that uses the echo command to print a message to the console with colored text. The **'-e'** option enables the interpretation of escape sequences, and **\033[0;32m** sets the text color to green (or 31 to red for example).




```
1   #!/bin/bash
2
3   # Project Windows Forensics
4
5   #Installing Figlet (Not mentioned in the script. Only for decorative usage)
6   if [ -d /usr/share/figlet ]
7   then
8       echo -e "\033[0;32mFiglet is already installed \033[0m"
9   else
10      echo -e "\033[0;31m[!] Figlet could not be found. Installing now. \033[0m"
11      sudo apt-get install figlet 1>/dev/null
12  fi
```

The next function checks if **Volatility**, a memory forensics tool, is already installed in the system. If it is not, it downloads the standalone Linux version of Volatility version 2.6 from the official website of the Volatility Foundation (credit), and then extracts the files and renames the directory for ease of use. If it is already installed, it displays a message saying so.

The script first checks **if** Volatility exists by using the command **-v** command. If it does, the script prints a message saying that it's already installed.

If Volatility is not found, the script proceeds to install it by running **sudo apt-get install -y volatility**. The **-y** flag is used to automatically answer "yes" to any prompts that may appear during installation.

The script then verifies that Volatility has been installed successfully by checking if the volatility command exists. If it does, the script prints a message saying that Volatility has been installed successfully. Otherwise, the script prints an error message saying that it failed to install Volatility.

The script uses color codes to make the output more readable. **The \033[0;32m** and **\033[0m** codes are used to print the success message in green, while the \033[0;31m and \033[0m codes are used to print the error messages in red. The **1>/dev/null** command is used to suppress any output from the apt-get update and apt-get install commands.

```
13
14  #Installing Volatility
15  # Check if Volatility exists
16  if [ -x "$(command -v volatility)" ]; then
17    echo -e "\033[0;32mVolatility is already installed \033[0m"
18  else
19    echo -e "\033[0;31m[!] Volatility could not be found. Installing now. \033[0m"
20
21    # Installing Volatility
22    echo -e "\033[0;31mInstalling Volatility...\033[0m"
23    sudo apt-get update 1>/dev/null
24    sudo apt-get install -y volatility 1>/dev/null
25
26    # Verify installation
27    if [ -x "$(command -v volatility)" ]; then
28      echo -e "\033[0;31mVolatility has been installed successfully.\033[0m"
29    else
30      echo -e "\033[0;31m[!] Failed to install Volatility.\033[0m"
31    fi
32  fi
```

foremost tool checks whether foremost is already installed on the system, by checking if the directory /usr/bin/foremost exists. If it does exist, the script prints the message "foremost is already installed" in green text using the echo command.

If foremost is not already installed, the script prints the message "Foremost could not be found. Installing now." in red text using the echo command, and then proceeds to install foremost by running the following commands: **‘sudo apt-get update 1>/dev/null’** **‘sudo apt-get install -y foremost 1>/dev/null’** .

```
┌──(root㉿kali)-[/home/kali/Desktop/Projects/Wf]
└─# bash WF-project.sh
foremost could not be found. Installing now
```

```
34  # Installing foremost
35  # Check if foremost exists
36  if [ -x "$(command -v foremost)" ]; then
37    echo -e "\033[0;32mforemost is already installed \033[0m"
38  else
39    echo -e "\033[0;31m[!] foremost could not be found. Installing now. \033[0m"
40
41    # Installing foremost
42    echo -e "\033[0;31mInstalling foremost...\033[0m"
43    sudo apt-get update 1>/dev/null
44    sudo apt-get install -y foremost 1>/dev/null
45
46    # Verify installation
47    if [ -x "$(command -v foremost)" ]; then
48      echo -e "\033[0;31mforemost has been installed successfully.\033[0m"
49    else
50      echo -e "\033[0;31m[!] Failed to install foremost.\033[0m"
51    fi
52  fi
```

root

This script verifies whether the current user is the root user before proceeding. The script uses the whoami command to retrieve the name of the current user, and compares it to the string "root" using a Bash if statement.

If the current user is not the root user, the script prints the message "[x]:: You are not root. Please exit." in red text using the echo command.

The script then exits using the exit command, which terminates the script immediately. This prevents the script from running any further commands, since the script requires administrative privileges to execute.

```
54  # 1. Automate HDDm the given, binary image for ending files and executable code
55  # 1.1 Verify user is root
56  if [ "$(whoami)" != "root" ]
57  then
58      echo -e "\033[0;31 [x]:: You are not root. Please exit. \033[0m"
59      exit
60  fi
```

The following script checks whether the file specified by the user exists using the -e option of the test command, which is equivalent to the **[-e file]** test in Bash. If the file exists, the script prints the message "[✔] File $filename exists!"

If the file does not exist, the script prints the message "The file does not exist."

```
62  # 1.2 Allow the user to specify the filename; check if the file exists
63
64      echo -e "\033[0;32m Please enter a Memory or a HDD filename: \033[0m"
65      read filename
66
67  if [ -e "$filename" ];
68      then
69          echo -e "\033[0;32m [✔]File $filename exists!\033[0m"
70      else
71          #no' 31= red output
72          echo -e "\033[0;31m The file does not exist.\033[0m"
73  fi
```

Memory or image file?

The first line of the script uses the'**read'** command to prompt the user to enter a filename, and stores the filename in the variable FILE.

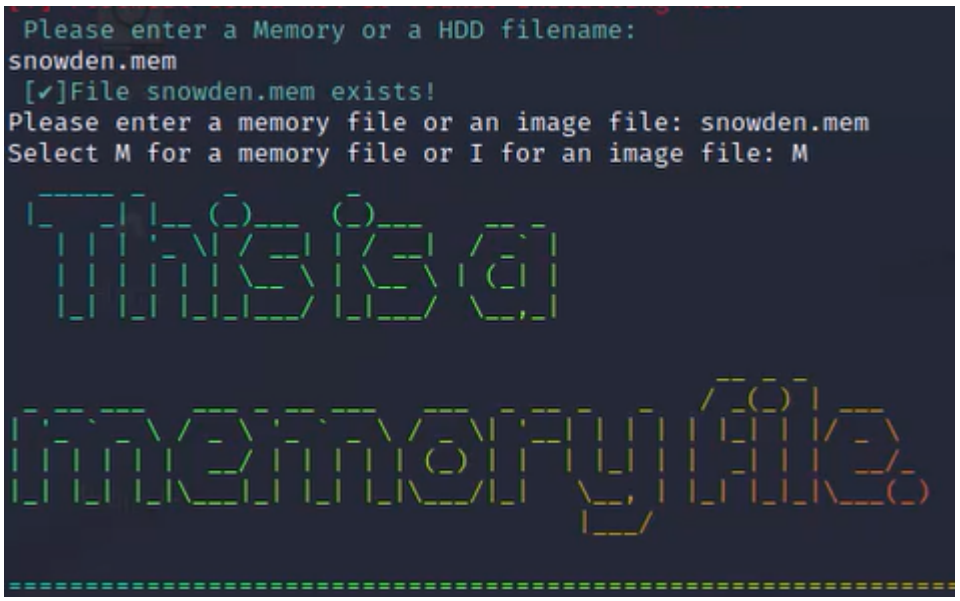
The second line of the script uses the read command to prompt the user to select "M" for a memory file or "I" for an image file, and stores the selection in the variable SEL.

The purpose of this prompt is to determine whether the forensics tools should be applied to a memory file or an image file, as the process may differ depending on the type of file.

For example, if the user selects "M" for a memory file, the script may use the volatility tool to analyze the file. If the user selects "I" for an image file, the script may use the foremost tool to extract data from the file.

The '**read'** command is used to obtain input from the user in both prompts, and the '**-p'** option is used to display a prompt message to the user. The input provided by the user is stored in the specified variable.

```
75  # Specify whether the file entered is a memory or image file in order to apply appropriate.
76  read -p "Please enter a memory file or an image file: " FILE
77
78  read -p "Select M for a memory file or I for an image file: " SEL
```

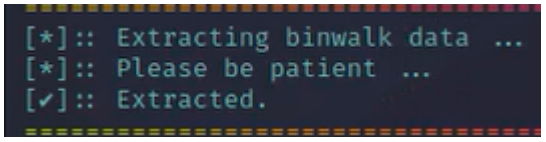


binwalk is a tool to extract embedded files and executable code from a binary image.

The function is called BNWLK().

When called BNWLK, it prints a message to the console indicating that it is extracting binwalk data and asks the user to be patient. It then runs the binwalk command with the specified $FILE input and saves the output to a file called “file_binwalk”.

After binwalk finishes, the function prints a message indicating that the extraction is complete and adds a line of equal signs using the lolcat tool for decoration.




```
80  #(1) Extracting from the given, binary image for enbedded files and executable code.
81
82  function BNWLK ()
83  {
84      echo -e "\033[0;32m[*]:: Extracting binwalk data ...\033[0m"
85      echo -e "\033[0;32m[*]:: Please be patient ...\033[0m"
86      binwalk $FILE > file_binwalk 2>/dev/null
87      echo -e "\033[0;32m[✔]:: Extracted. \033[0m"
88      echo "==========================================================================================" | lolcat
89  }
```

lolcat - The command starts with echo, which is used to output text to the console. The text being output is a line of equal signs, which is enclosed in double quotes.

The **‘|’** symbol is a pipe, which is used to pass the output of the echo command to another command. In this case, it passes the output to the lolcat command.

lolcat is a command-line tool that adds rainbow color to text. It reads input from the console and applies color to it, then outputs the colored text to the console. In this case, it applies color to the line of equal signs generated by **echo**.

bulk extract data using **‘bulk_extractor’**. The script indicates that it is a non-parsing method that scans the disk image and extracts useful information.

bulk_extractor is a tool for extracting features from a disk image. It can be used to search for specific patterns, including email addresses, URLs, credit card numbers, and other sensitive information.

The function is passed the file location ($FILE) and uses the **‘-o’** option to specify the output directory as file_bulk. The ‘1>/dev/null’ redirects standard output to /dev/null, so only errors will be printed to the console.

After the data is extracted, the function prints a message to the console and inserts a line of equal signs using the lolcat program for decoration.

```
#(2)Scans the disk image and extracts useful information without parsing the file system or file system structures.

function BULK ()
{
    echo -e "\033[0;32m [*]:: Extracting bulk_extractor data ...\033[0m"
    echo -e "\033[0;32m[*]:: Please be patient ...\033[0m"
    bulk_extractor $FILE -o file_bulk 1>/dev/null
    echo -e "\033[0;32m [✔]:: Extracted. \033[0m"
    echo "================================================================================================" | lolcat
}
```

```
 [*]:: Extracting bulk_extractor data ...
[*]:: Please be patient ...
 [✔]:: Extracted.
```

foremost tool used to extract lost files based on their headers, footers and internal data structures. The function first displays a message to indicate that it is starting to extract the data using foremost. Then, it runs the foremost command with the specified input file ($FILE) and the ‘**-t’** all option to specify that all file types should be extracted. The output is directed to a directory named ‘file_forem’ using the **‘-o’** option.
Finally, a message is displayed to indicate that the extraction is complete, and a line of equal signs is printed for formatting using the lolcat tool.

```
#(3) Extracting lost files based on their headers, footers and internal data structures.

function FOREM ()
{
    echo -e "\033[0;32m[*]:: Extracting foremost data ...\033[0m"
    echo -e "\033[0;32m[*]:: Please be patient ...\033[0m"
    foremost $FILE -t all -o file_forem 1>/dev/null
    echo -e "\033[0;32m[✔]:: Extracted. \033[0m"
    echo "================================================================================================" | lolcat
}
```

```
[*]:: Extracting foremost data ...
[*]:: Please be patient ...
Processing: snowden.mem
|***|
[✔]:: Extracted.
```

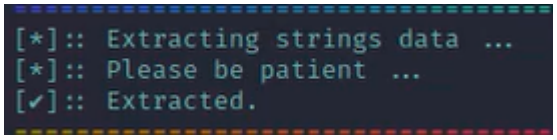
strings

The STR() function extracts strings from the given memory or image file using the strings utility in Linux.

The function starts by printing a message indicating that the strings data is being extracted and advises the user to be patient. It then runs the strings command with the given file and outputs the result to a file called “file_strings”. Finally, it prints a success message and a separator line using the lolcat utility for decorative purposes.

```
#(4) Extracting strings from your mem file.

function STR ()
{
    echo -e "\033[0;32m[*]:: Extracting strings data ...\033[0m"
    echo -e "\033[0;32m[*]:: Please be patient ...\033[0m"
    strings $FILE > file_strings 1>/dev/null
    echo -e "\033[0;32m[✔]:: Extracted. \033[0m"
    echo "=============================================================================================" | lolcat
}
```



The following script defines a function VOL() which extracts RAM information from a memory dump file using the **volatility** tool.

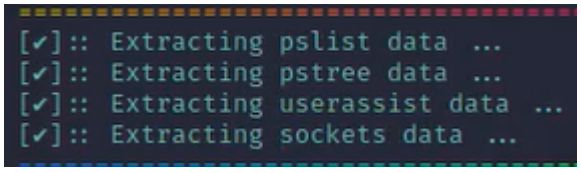
The function first extracts the imageinfo from the dump file using volatility and stores the output in a file named file_profile. It then retrieves the suggested profile from file_profile and stores it in a variable named FILE_PROFILE.

The function then creates a list of VOLDATA which contains pslist, pstree, userassist, and sockets. It then iterates over each item in VOLDATA and extracts the relevant data using volatility and stores the output in a file with the name of the item and .txt extension.

```
#(5) Extracting RAM information.

function VOL ()
{
    echo -e "\033[0;32m[✔]:: Extracting imageinfo ... \033[0m" 1>/dev/null
    ./vol -f $FILE imageinfo > file_profile 2>/dev/null
    FILE_PROFILE=$(cat file_profile | grep -i suggested | awk '{print $4}' | awk -F ',' '{print $1}')
    VOLDATA="pslist pstree userassist sockets"

    for i in $VOLDATA
    do
        echo -e "\033[0;32m[✔]:: Extracting $i data ...\033[0m"
        ./vol -f $FILE --profile=$FILE_PROFILE $i > $i.txt 2>/dev/null
    done
        echo "=============================================================================================" | lolcat
}
```



The WF script includes several functions for extracting data from the file, including using the binwalk tool to extract embedded files and executable code, using foremost to extract lost files based on their headers, footers, and internal data structures, using strings to extract strings from the file, and using the Volatility Framework to extract RAM information.

The script also includes a function called LOG that collect data from the extracted files and write it to a log file called yourmem_log.

The following code is iterating over a list of file extensions stored in LOGDATA and for each file extension, it is executing a set of commands using the find command:

‘find . -type f -name "*.$i" | awk -F '/' '{print $NF}' ‘**- This finds all files in the current directory and its subdirectories that have the file extension **‘$i’, and prints out only the filename (without the path) using awk.
1>/dev/null - This redirects the standard output to /dev/null, effectively discarding it.
‘wc -l’ - This counts the number of lines of the output of the previous find command, which is the number of files found that have the extension **‘$i’**.
So for each file extension, this code finds all the files with that extension in the current directory and its subdirectories, and prints out the filename for each file found. Then it prints out the total number of files found with that extension.

```
141  # Collecting data from the extracted files.
142
143  function LOG ()
144  {
145      LOGDATA='txt exe gif wav dll pcap'
146
147      for i in $LOGDATA
148      do
149          echo -e "\033[0;32m[✔]:: Extracting $i files. \033[0m"
150          find . -type f -name "*.$i" | awk -F '/' '{print $NF}' 1>/dev/null
151          echo -e "\033[0;32m[✔]:: Calculated $i extracting files. \033[0m"
152          find . -type f -name "*.$i" | awk -F '/' '{print $NF}' | wc -l 1>/dev/null
153      done
154  } > yourmem_log
```

(Lines 159-189)
The following screenshot displays another script for analyzing a memory or image file.
The script has several functions to extract information from the given file, including using tools like binwalk, foremost, strings, and Volatility to extract embedded files, lost files, strings, and RAM information.
The extracted data is then collected and analyzed, and the results are written to a log file.
The script also has a case statement that allows the user to specify whether the file is a memory or image file and executes the appropriate functions. Overall, the script appears to be a useful tool for forensic analysis of memory and image files.

sleep 0.2 - The sleep command used to pause the execution of a script for 0.2 seconds. This is useful to create a delay between the execution of different commands or when someone want to give the user time to read a message before continuing.

'case' is a conditional construct in unix-like shell scripting languages that allows to test a variable against a series of patterns, and execute code based on the first pattern that maches.
'case' is often used in combinations with 'esac'' which is simpy the word "case" spelled backwards, and it used to signal the end of a 'case' block.

```
157  # Running the function according to the type file.
158
159  case $SEL in
160
161  M)
162      figlet "$MEM This is a memory file." | lolcat
163      echo
164      echo "=============================================================================================" | lolcat
165  # Execute the functions
166      BULK
167      BNWLK
168      FOREM
169      STR
170      VOL
171      sleep 0.2
172  ;;
173  I)
174      figlet "[✔]:: Confirm $MEM image file" | lolcat
175
176      echo "=============================================================================================" lolcat
177  # Execute the functions
178      BULK
179      BNWLK
180      FOREM
181      STR
182      VOL
183      sleep 0.2
184  ;;
185  esac
186
187  # [✔] Sucsess [✔]
188
189  echo "=============================================================================================" | lolcat
```

The last screenshot - The end of the bash script that extracts data from a memory file or image file. The script defines several functions, including:
BULK: extract data from the memory file using bulk_extractor.
BNWLK: extract data from the memory file using binwalk.
FOREM: extract lost files from the memory file using foremost.
STR: extract strings from the memory file.
VOL: extract various types of information from the memory file using the Volatility Framework.

LOG: collect data from the extracted files.
The script then prompts the user to select a memory file or image file, and runs the appropriate functions based on the user's choice

Finally, it runs the LOG function to collect data from the extracted files, and saves the log data to a file.

The last line of the script displays a message to the user indicating that the log data has been saved to a file.

```
191  figlet -f mono9 "Extracting data file" | lolcat
192
193  # Execute function LOG
194  LOG
195
196  echo "=============================================================================================" | lolcat
197  echo -e "\033[0;33mYour log data has been saved to 'file' log :) \033[0m"
198
```