Assignment-1

Course: Cybersecurity Fundamentals

University of Adelaide

Name: Karan Sethi

Student ID: a1745936

**Note - Final answers are highlighted in blue font**

Answer 1: The grep command that was used to find the password was -

|  |
| --- |
| # grep -A 1 "^And.\*it$" text |

* The -A 1 grep option was used to display the 1 line following the located lines.
* ^ and $ in the regex were used to signify start and end of the expression respectively.
* .\* used to match any number of characters between ‘And’ and ‘it’.
* The operation was performed on the ‘text’ file in the Q01 folder.

The above grep command returned the following lines as the result -

|  |
| --- |
| And give't Iago: what he will do with it emotive-tombac-partible-maritime |

In this above result, the first line matches the regex and the second line is the passphrase answer required: “emotive-tombac-partible-maritime”

Answer 2: The command that was used to find the passphrase that occurs only once was -

|  |
| --- |
| # cat here | sort | uniq -u |

* The cat command displays the contents of here file but that output is piped with ‘|’ into sort
* The output of sort is piped into uniq command with -u option which makes it a filter to display only the lines which appear once in the standard input
* The output of uniq command comes to standard output and is shown in the terminal

The above command returned the following passphrase only which occurred only once in the here file and it is our answer -

|  |
| --- |
| stylist-recurve-cyanate |

Answer 3:

The script that was written to compare sha256 sum with the given hash and find the one with match was -

|  |
| --- |
| #!/bin/bash # This is a script to check sha256 sum against a given value for all the files in the folder and return the file whose sum matches against the value.  # we loop over each file and get its sha256sum and compare it with our given value # we then get the contents of the file and write it to the screen  checksum="77c5ea3694cd024108e5b9dcb0e8de8e5ddd4efac0588fb29a9027cca546c4a5" **for** file **in** file00\*; **do**  echo "$checksum $file" | sha256sum --status -c  **if** [ $? -eq 0 ]; **then**  echo  echo "Found a successful match of sha265sum hash!"  echo "The provided hash $checksum matched with the hash of the file: ${file}"  echo  echo "The contents of the file are: "  cat $file  exit  **fi** **done** |

* As given in the comments in the start of the script, we simply loop over all the files matching the regex file00\* which were given and verify if the sha256sum hash of the file matches with the given hash stored in checksum variable in the start
* When we find a match of the hash we simply tell which file it was and what are its contents before exiting the loop.

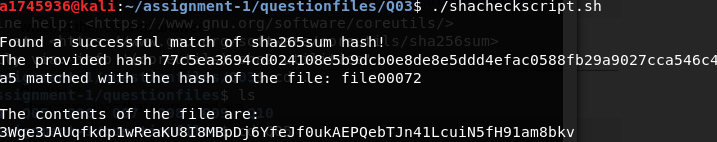
The given hash

77c5ea3694cd024108e5b9dcb0e8de8e5ddd4efac0588fb29a9027cca546c4a5

matched with the hash of the file - file00072 and its contents were -

|  |
| --- |
| 3Wge3JAUqfkdp1wReaKU8I8MBpDj6YfeJf0ukAEPQebTJn41LcuiN5fH91am8bkv |

which is the required answer



Answer 4:

The commands that were used are -

|  |
| --- |
| # cat words.txt | tr 'aeio' '4310' > l33tversion.txt # sha256sum l33tversion.txt |

* cat is used on the words.txt file to print contents to standard output but that is piped into the ‘tr’ transform command which replaces aeio with 4310 respectively and the output of that is piped into l33tversion.txt. Then the sha256 hash of l33tversion.txt is calculated and printed using the second command above.

The answer hash returned was -

|  |
| --- |
| 96a50457d46c2c8596ccdf8fe79d4953e4cff9766ae089fd78f7742f6437499e |

Answer 5: The script that was written and used to decrypt with gpg using brute force passwords in a dict file is -

|  |
| --- |
| #!/bin/bash # This script is to brute force an encrypted file using a list of passwords stored in a file called 'dict.txt' # After gpg command for decryption executes we can check result using return code variable $?  # STEPS - # get the contents of l33t converted password list and loop over each line containing a password # print result secret along with brute force password which worked to decrypt the secret.txt.gpg file # send errors to /dev/null # compare each result with 0, if it is 0 print the password that worked and exit the loop   cat ./dict.txt | **while** read pass; **do**  res=$(gpg --batch --passphrase "$pass" --decrypt secret.txt.gpg 2> /dev/null)  **if** [ 0 -eq $? ]; **then**  echo  echo "YAY found secret! : $res"  echo "with the password : $pass"  echo  exit  **fi** **done** |

As given in the comments before the script, we loop over the contents of dict.txt which contains the l33t converted password list and try each password using gpg for decryption of the encrypted file secret.txt.gpg

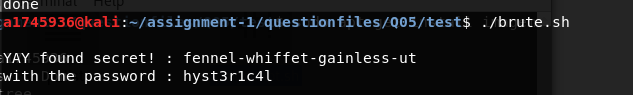
* We send errors to /dev/null as we don’t want to see them
* When we don’t have an error means the $? return value from gpg command worked as password for the file and we then print the secret and the password that worked to the screen using echo commands
* We exit the loop after finding the password

The secret answer after decryption of the file is -

|  |
| --- |
| fennel-whiffet-gainless-ut |

The brute force password from dict.txt (l33t converted password list) that worked is -

|  |
| --- |
| hyst3r1c4l |



Answer 6:

The script that was used is -

|  |
| --- |
| #!/bin/bash # Script to ping all hosts in a text file ip\_list in the same directory  # open ip\_list file and send output lines to while loop cat ./ip\_list | **while** read output **do**  #ping each host once and send output to /dev/null  ping -c 1 "$output" > /dev/null  #print based on ping response of 0 or 1  **if** [ $? -eq 0 ]; **then**  echo "$output is up!"  **else**  echo "$output is down!"  **fi** **done** |

* After piping the contents of ip\_list file into a while loop which runs till the end of the file, we ping each host in each line once and say if it is up or not based on the ping host response.

The result of the script was -

|  |
| --- |
| 10.0.0.1 is down! 10.0.0.2 is down! 10.0.0.3 is down! 10.0.0.4 is down! 10.0.0.5 is down! 10.0.0.6 is down! 10.0.0.7 is down! 10.0.0.8 is down! 10.0.0.9 is down! 10.0.0.10 is up! 10.0.0.11 is up! 10.0.0.12 is up! 10.0.0.13 is up! 10.0.0.14 is up! 10.0.0.15 is up! 10.0.0.16 is down! 10.0.0.17 is up! 10.0.0.18 is down! 10.0.0.19 is up! 10.0.0.20 is up! 10.0.0.21 is down! 10.0.0.22 is down! 10.0.0.23 is down! 10.0.0.24 is up! |

Answer 7:

The command that was used is -

|  |
| --- |
| # find -size 30c | xargs cat |

* We find the file in all the folders having size of 30 bytes and use that file as argument to the cat command which prints the contents of the file to the screen.

It returned the following secret answer result -

|  |
| --- |
| The file name was - “file5” located in Q07/folder4/folder3/folder1/file5  And it contained :  carroty-symmetry-insofar-bree |

Answer 8:

The commands that were used are -

|  |
| --- |
| # file secret |

This gives the following description about the file -

|  |
| --- |
| secret: bzip2 compressed data, block size = 900k |

We use bzip2 to decompress the file -

|  |
| --- |
| # bzip2 -d secret |

This gives the following output -

|  |
| --- |
| bzip2: Can't guess original name for secret -- using secret.out |

And creates a new file secret.out

We check the type of file of secret.out -

|  |
| --- |
| # file secret.out |

This gives the following description about the file -

|  |
| --- |
| secret.out: gzip compressed data, was "secret" |

It gives the error ‘gzip: secret.out: unknown suffix -- ignored’ if we try the gzip command as it can’t recognize the file format. So we change the file format to gz using the following command and use gzip -

|  |
| --- |
| # mv secret.out secret.gz  # gzip -d secret.gz |

We check the file format -

|  |
| --- |
| # file secret |

Change the file format again otherwise gzip gives errors -

|  |
| --- |
| # mv secret secret.gz |

Use gzip again -

|  |
| --- |
| # gzip -d secret.gz |

We check the file format -

|  |
| --- |
| # file secret |

This tells that the file is ASCII text so now we can read it using cat command -

|  |
| --- |
| secret: ASCII text  # cat secret |

Secret text contained in the file and the final answer -

|  |
| --- |
| diagram-loculus-sora-broode |

Answer 9:

Command run to calculate hash of the poem in sonnet file was -

|  |
| --- |
| # cat sonnet | sha256sum |

* cat prints out the file to standard output but that was piped using ‘|’ into the sha256sum command which takes input and digests it to give the hash and prints it out to standard output

Command run to calculate hash of the reverse of the poem in the sonnet file was -

|  |
| --- |
| # tac sonnet | sha256sum |

tac prints out the file in reverse to standard output but that was piped using ‘|’ into the sha256sum command which takes input and digests it to give the hash and prints it out to standard output.

The hashes returned respectively were -

* With cat

|  |
| --- |
| 0647b576fbd4da8eddc52e597b2339b9410a2ae8bff8bbbeae7279151248c9a9 |

* With tac

|  |
| --- |
| 0426e39f391f94a402c1e7130cc691cd8da4d17eeaa1265781bb7faa24015a1b |

Answer 10:

Commands that were run are -

|  |
| --- |
| # fortune | cowsay -f sheep.cow |

The result after executing the command was -

|  |
| --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / You possess a mind not merely twisted, \ \ but actually sprained. /  ----------------------------------------  \  \  \_\_   UooU\.'@@@@@@`.  \\_\_/(@@@@@@@@@@)  (@@@@@@@@)  `YY~~~~YY'  || || |

--------------------------- End of Document---------------------------