CFC190324
(Python Fundamentals)
Project Title: Log Analyzer

Student Code: S8
Student Name: Jeremiah Lee
Trainer Name: James

Table of Contents

1) Introduction

In the ever-evolving landscape of cybersecurity, understanding system activity and potential threats is paramount. This project, titled "Operation Log Analyzer," tackles this challenge head-on by developing a specialised Python tool for in-depth log analysis.

Our focus centres on the treasure trove of information contained within /var/log/auth.log, a critical file that chronicles system authentication attempts. By harnessing the power of Python, we aim to extract valuable data, identify recurring patterns, and ultimately assess potential security vulnerabilities. Through meticulous examination, "Operation Log Analyzer" strives to transform raw log data into actionable intelligence, providing a clear picture of system behaviour, security threats, and any anomalies that might signal malicious activity.

This project's mission is to empower system administrators with a comprehensive understanding of their systems' security posture, enabling them to make informed decisions and safeguard critical infrastructure.

2) Methodologies

In this chapter, we will briefly describe the commands used to obtain the relevant information.

2.1) Log Parse auth.log: Extract command usage.

This refers to the process of analysing and extracting information from the auth.log file. We will be focusing on extracting what commands were used within the auth.log file, including the users and the timestamp.

Prior to extracting the timestamp, user and command, we first had to open the auth.log file. The following commands were used:

```
file=open('auth.log','r')
```

This line of code is used to open a file for reading.
open(): This is a built-in function in Python used to open a file.
'Auth.log': This is the argument passed to the open() function, specifying the name of the file you want to open. In this case, it's auth.log.
'r': This is the second argument passed to the open() function, indicating the mode in which you want to open the file. The 'r' specifies read mode, meaning you intend to read data from the file.

Therefore, by executing this line, I am creating an object named file that is associated with the auth.log file. This file object allows you to interact with the contents of the auth.log file, such as reading its lines or iterating through them.

Following that, the following command was used:

```
for eachline in file:
```

for: This keyword introduces a for loop in Python.
eachline: This is a variable name that I had chosen to represent each line of text read from the file within the loop. In subsequent for loops, different names will be given depending on the type of objective to be found. Subsequent uses of eachline will be work in the similar manner, but would just be given a different name for clarity.

in: This keyword specifies the iterable object over which the loop will iterate. In this case, in file indicates that the loop will iterate over each line in the file object referenced above.

2.1.1) Including the timestamp

The following command was used to obtain the timestamp, executed user and command used:

```
if 'COMMAND' in eachline:
```

if: This keyword introduces an if statement, which checks a condition before executing a block of code.
'COMMAND': This is a string literal representing the word "COMMAND" (enclosed in single quotes).
in: This keyword is used for membership testing. In this context, it checks if the string "COMMAND" is present within the current line stored in the variable eachline.
eachline: This variable, defined within a for loop iterating over lines in a file, holds the content of the current line being processed.

In this case, this command searches each line in the file for the phrase “COMMAND” and focuses on those lines to be redirected into the subsequent commands lower down the script.

```
splatline=eachline.split()
```

eachline: This refers to the variable that holds a string containing a line of text read from the file (as explained above in 2.1).
.split(): This is a method call on the string object in eachline. The .split() method splits the string into a list of words based on a delimiter (by default, whitespace characters like spaces, tabs, or newlines).
splatline: This is the variable name I had chosen to store the resulting list of words.

This therefore enables the eachline string to be split based on white spaces.

The result will be a list of words. An example used on the above sentence will be as below:

['This', 'therefore', 'enables', 'the', 'eachline', 'string', 'to', 'be', 'split', 'based', 'on', 'white', 'spaces.']

This list is then assigned to the variable splatline.

In order to obtain the time stamp, the following command was used:

```
timestamp=' '.join(splatline[:3])
```

timestamp: This is a variable name I had chosen to store the extracted timestamp.
' ': This is a string literal representing a single space character.
.join( ): This is the .join() method called on a string object (the space character in this case).
splatline: This refers to a variable containing a list of words split from a line of text using the .split() method (as explained previously).
[:3]: This is a slicing operation applied to the splatline list. Slicing extracts a portion of the list. In this case, it selects elements from the beginning of the list (index 0) up to, but not including, the third element (index 2).

In the case of the list below:
['This', 'therefore', 'enables', 'the', 'eachline', 'string', 'to', 'be', 'split', 'based', 'on', 'white', 'spaces.']

It prints out “This therefore enables”.
“This” is index zero, with “therefore” being the 1st index, with “enables” being the 2nd index based on the zero index rule.

2.1.2) Including the executing user

```
user=splatline[5]
```

User: This is the variable name I had chosen to store the extracted user.
splatline: Explained in 2.1.1, it stores the list of words whom had been split.
[5]: This is indexing notation used to access specific elements within the splatline list. Indexing starts from 0, so [5] refers to the sixth element in the list.

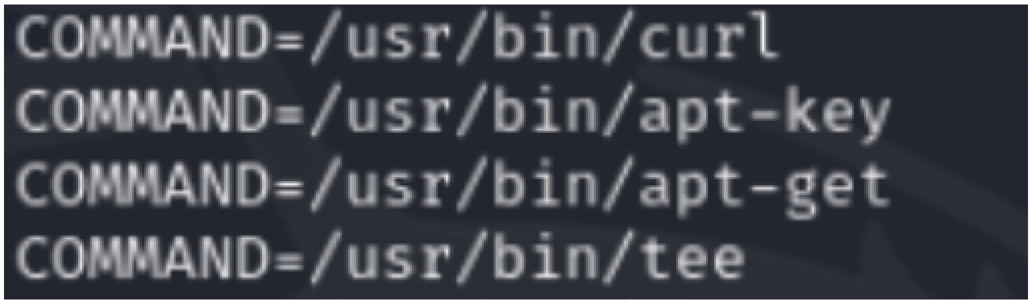
In the case of the list below:
['This', 'therefore', 'enables', 'the', 'eachline', 'string', 'to', 'be', 'split', 'based', 'on', 'white', 'spaces.']

It will print the string “string”

2.1.3) Include the command

commandequal=splatline[13]

commandequal: This is the variable name I had chosen to store the extracted command with equal

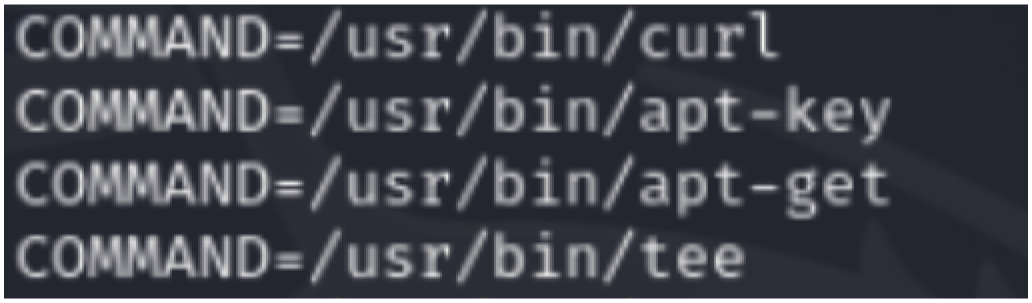


```
COMMAND=/usr/bin/curl
COMMAND=/usr/bin/apt-key
COMMAND=/usr/bin/apt-get
COMMAND=/usr/bin/tee
```

The above image gives an example of the “equal” in the variable
splatline: Explained in 2.1.1, it stores the list of words that has been split.
[13]: This is indexing notation used to access specific elements within the splatline list. Indexing starts from 0, so [13] refers to the 14th element in the list.

slcommandequal=commandequal.split('/')

Slcommandequal: The variable name I assigned for the output of the command “commandequal.split(‘/’),
commandequal: Variable explained in previous section
.split(‘/’): The .split() method splits the string into a list of words based on a delimiter. In this case the ‘/’ was used as a delimiter



```
COMMAND=/usr/bin/curl
COMMAND=/usr/bin/apt-key
COMMAND=/usr/bin/apt-get
COMMAND=/usr/bin/tee
```

In the above case, the list will be split into [‘COMMAND=’,’usr’,’bin’,’tee’]
The last element in the list is the command name.

```
command=slcommandequal[-1]
```

command: Variable name I had chosen to store the command used
Slcommandequal[-1]: obtains the last element in the list of slcommandequal.

Thereafter, the following command was used to inform the reader of the output.

```
print('On',timestamp,',',user,'executed',command)
```

This prints the timestamp, user and command used in a readable string sentence.

Subsequently the command

```
file.close()
```

Was used to close the file. Closing the file is important as the system allocates resources to handle interactions with the file. Closing the file also ensures data integrity as some files are not physically written to the storage device until the file is closed.

2.2) Log Parse auth.log: Monitor user authentication changes

2.2.1) Print details of newly added users, including the Timestamp

The following command was used to open the file:

```
file=open('auth.log','r') #opens the file named "auth.log" and sets up reading mode
```

The methodology of the said command was explained in chapter 2.1.

```
for elnewuser in file:
```

Similar to chapter 2.1, however this time, instead of “eachline” used as a variable name, “elnewuser” is used as a variable name (el is used as an abbreviation for each line).

```
if 'new user' in elnewuser:
```

Similar to the if command as explained in chapter 2.1, this command essentially searches for the string ‘new user’ in each elnewuser (each line new user).

```
slnewuser=elnewuser.split()
```

Similar to chapter 2.1, however this time, the variable “slnewuser” is used instead of “splatline” (sl is used as an abbreviation for splat line). The .split() method splits the string into a list of words based on a delimiter (by default, whitespace characters like spaces, tabs, or newlines).

```
newusertimestamp=' '.join(slnewuser[:3])
```

newusertimestamp: This is a variable name I had chosen to store the extracted timestamp.
' ': This is a string literal representing a single space character.
.join( ): This is the .join() method called on a string object (the space character in this case).
slnewuser: This refers to a variable likely containing a list of words split from a line of text using the .split() method (as explained previously).
[:3]: This is a slicing operation applied to the splatline list. Slicing extracts a portion of the list. In this case, it selects elements from the beginning of the list (index 0) up to, but not including, the third element (index 2).

newuserequal=slnewuser[7]

newuserequal: This is the variable name I had chosen to store the extracted user. Explained in 2.1.1, it stores the list of words whom had been split. The variable was nabled newuserequal as the output of the variable includes the '=' sign (USER=Usernameiwant). The image below gives an example.

```
USER=root
```

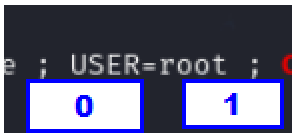
[7]: This is indexing notation used to access specific elements within the slnewuser list. Indexing starts from 0, so [8] refers to the sixth element in the list.

splatnewuserequal=newuserequal.split('=')

Splatnewuserequal: The variable name that has been chosen to store list of strings that had been split from the newuserequal.split() function.
.split(): This is a method call on the string object in newuserequal. The .split() method splits the string into a list of words based on a delimiter (by default, whitespace characters like spaces, tabs, or newlines). In this case, it identifies the '=' character and splits the string using that as a delimiter.
Using the example mentioned above, it will give ['USER', 'Usernameiwant']

newuser=splatnewuserequal[1]

newuser: The variable name chosen to store the name of the new user added
splatnewuserequal: Indexing notation used to access specific elements in splatnewuserequal list. In this case it selects the 2nd element in the list following the zero index rule. Using the previous example, it gives the output 'Usernameiwant'. A visual representation is given below.



```
e ; USER=root ;
```

```
print('A new user with the username',newuser,'was added','on',newusertimestamp)
```

Subsequently, the print command was used to print the output into a human readable sentence for the reader.

Subsequently the command

```
file.close()
```

Was used to close the file.

2.2.2) Print details of deleted users, including the Timestamp

```
file=open('auth.log','r')
```

This line of code is used to open a file for reading.
open(): This is a built-in function in Python used to open a file.
‘Auth.log’: This is the argument passed to the open() function, specifying the name of the file you want to open. In this case, it's auth.log.
‘r’: This is the second argument passed to the open() function, indicating the mode in which you want to open the file. The 'r' specifies read mode, meaning you intend to read data from the file.

Therefore, by executing this line, I am creating an object named file that is associated with the auth.log file. This file object allows you to interact with the contents of the auth.log file, such as reading its lines or iterating through them.

for eldeluser in file:

Similar to chapter 2.1, however this time, instead of “eachline” used as a variable name, “eldeluser” is used as a variable name (el is used as an abbreviation for each line).

if 'userdel' in eldeluser:

if: This keyword introduces an if statement, which checks a condition before executing a block of code.
'usedel': This is a string literal representing the word "userdel" (enclosed in single quotes).
in: This keyword is used for membership testing. In this context, it checks if the string "userdel" is present within the current line stored in the variable eachline.
eachline: This variable, defined within a for loop iterating over lines in a file, holds the content of the current line being processed.

In this case, this command searches each line in the file for the phrase “userdel” and focuses on those lines to be redirected into the subsequent commands lower down the script.

sldeluser=eldeluser.split()

eldeluser: This refers to the variable that holds a string containing a line of text read from the file (as explained above in 2.1).
.split(): This is a method call on the string object in eachline. The .split() method splits the string into a list of words based on a delimiter (by default, whitespace characters like spaces, tabs, or newlines).
sldeluser: This is the variable name I had chosen to store the resulting list of words from the split up of eldeluser.

This therefore enables the eldeluser string to be split based on white spaces.

```
delusertimestamp=' '.join(sldeluser[:3])
```

delusertimestamp: This is a variable name I had chosen to store the extracted timestamp of the moment the user was deleted.
' ': This is a string literal representing a single space character.
.join( ): This is the .join() method called on a string object (the space character in this case).
sldeluser: This refers to a variable containing a list of words split from a line of text using the .split() method (as explained previously).
[:3]: This is a slicing operation applied to the splatline list. Slicing extracts a portion of the list. In this case, it selects elements from the beginning of the list (index 0) up to, but not including, the third element (index 2).

```
deluser=sldeluser[5]
```

delser: This is the variable name I had chosen to store the extracted user name of the deleted user.
sldeluser: Explained in 2.1.1, it stores the list of words which had been split.
[5]: This is indexing notation used to access specific elements within the splatline list. Indexing starts from 0, so [5] refers to the sixth element in the list.

```
print('A user with the username',deluser,'was deleted','on',delusertimestamp)
```

The print function was then used to print the output into a human readable format for the reader.

```
file.close()
```

The file was then closed.

2.2.3) Print details of changing passwords, including the Timestamp.

```
file=open('auth.log','r')
```

This line of code is used to open a file for reading.
open(): This is a built-in function in Python used to open a file.
‘auth.log’: This is the argument passed to the open() function, specifying the name of the file you want to open. In this case, it's auth.log.
‘r’: This is the second argument passed to the open() function, indicating the mode in which you want to open the file. The 'r' specifies read mode, meaning you intend to read data from the file.

```
for elchpass in file:
```

for: This keyword introduces a for loop in Python.
elchpass: This is a variable name that I had chosen to represent each line of text read from the file within the loop for the section of changing passwords
in: This keyword specifies the iterable object over which the loop will iterate. In this case, in file indicates that the loop will iterate over each line in the file object referenced above.

```
chpasstimestamp=' '.join(slchpass[:3])
```

shpasstimestamp: This is a variable name I had chosen to store the extracted timestamp when the user had changed the password.
' ': This is a string literal representing a single space character.
.join( ): This is the .join() method called on a string object (the space character in this case).
slchpass: This refers to a variable containing a list of words split from a line of text using the .split() method (as explained previously).
[:3]: This is a slicing operation applied to the splatline list. Slicing extracts a portion of the list. In this case, it selects elements from the beginning of the list (index 0) up to, but not including, the third element (index 2).

```
chpassuser=slchpass[-1]
```

chpassuser: This is the variable name I had chosen to store the extracted user that changed passwords.
slchpass: Explained in 2.1.1, it stores the list of words whom had been split.
[-1]: This is indexing notation used to access specific elements within the splatline list. For the index of [-1], it reads as the last position in the string.

```
print(chpassuser, 'changed password on', chpasstimestamp)
```

Thereafter, the information is printed into a human-readable format for the user.

```
file.close()
```

The file was then closed.

2.2.4) Print details of when users used the su command.

```
file=open('auth.log','r')
```

This line of code is used to open a file for reading.
open(): This is a built-in function in Python used to open a file.
‘Auth.log’: This is the argument passed to the open() function, specifying the name of the file you want to open. In this case, it's auth.log.
‘r’: This is the second argument passed to the open() function, indicating the mode in which you want to open the file. The 'r' specifies read mode, meaning you intend to read data from the file.

```
for elsu in file:
```

for: This keyword introduces a for loop in Python.
elsu: This is a variable name that I had chosen to represent each line of text read from the file within the loop.

```
if '(su:session): session opened' in
```

if: This keyword introduces an if statement, which checks a condition before executing a block of code.
'(su:session): session opened': This is a string literal representing the sentence.
in: This keyword is used for membership testing. In this context, it checks if the string is present within the current line stored in the variable eachline.
eachline: This variable, defined within a for loop iterating over lines in a file, holds the content of the current line being processed. This way, the ‘if’ command will be used to filter for lines that contain the quoted string.

slsu=elsu.split()

elsu: This refers to the variable that holds a string containing a line of text read from the file (as explained above in 2.1).
.split(): This is a method call on the string object in eachline. The .split() method splits the string into a list of words based on a delimiter (by default, whitespace characters like spaces, tabs, or newlines).
slsu: This is the variable name I had chosen to store the resulting list of words.

sutimestamp=' '.join(slsu[:3])

sutimestamp: This is a variable name I had chosen to store the extracted timestamp when the user had used the su command.
' ': This is a string literal representing a single space character.
.join( ): This is the .join() method called on a string object (the space character in this case).
slsu: This refers to a variable containing a list of words split from a line of text using the .split() method (as explained previously).
[:3]: This is a slicing operation applied to the splatline list. Slicing extracts a portion of the list. In this case, it selects elements from the beginning of the list (index 0) up to, but not including, the third element (index 2).

sudetails=' '.join(slsu[6:])

sudetails: Variable name that I had chosen to store the extracted su command details.
' ': This is a string literal representing a single space character.

.join( ): This is the .join() method called on a string object (the space character in this case).
slsu: This refers to a variable containing a list of words split from a line of text using the .split() method (as explained previously).
[6:] This represents the starting index position for the slice. Python indexing starts from 0. So, in this case, 6 refers to the seventh element in the column. The colon indicates that all elements from the starting index (inclusive) up to the end of the sequence are selected. Hence, the command extracts the subsequence containing all elements from the 7th element to the end of the slsu list.

```
print('On',sutimestamp,'A', sudetails)
```

Thereafter, the information is printed into a human-readable format for the user.

```
file.close()
```

The file is then closed.

2.2.5) Print details of users who used the sudo; include the command.

```
file=open('auth.log','r')
```

This line of code is used to open a file for reading.
open(): This is a built-in function in Python used to open a file.
'Auth.log': This is the argument passed to the open() function, specifying the name of the file you want to open. In this case, it's auth.log.
'r': This is the second argument passed to the open() function, indicating the mode in which you want to open the file. The 'r' specifies rcead mode, meaning you intend to read data from the file.

```
for elsudopass in file:
```

for: This keyword introduces a for loop in Python.
elsudopass: This is a variable name that I had chosen to represent each line of text read from the file within the loop. (elsudopass is used to represent each line in file whereby the user had successful used a sudo command)

if 'sudo' and 'COMMAND' in elsudopass:

if: This keyword introduces an if statement, which checks a condition before executing a block of code.
‘sudo’: This is a string that represents super user permissions.
and: This adds a second condition for the if statement
'COMMAND': This is a string literal representing the word "COMMAND" (enclosed in single quotes).
in: This keyword is used for membership testing. In this context, it checks if the string “sudo” and "COMMAND" is present within the current line stored in the variable elsudopass.
elsudopass:: This variable is explained above.
Therefore this command filters for the lines whereby the string ‘sudo’ and ‘COMMAND’ is used.

slsudopass=elsudopass.split()

elsudopass: This refers to the variable that holds a string containing a line of text read from the file (as explained above in 2.1).
.split(): This is a method call on the string object in eachline. The .split() method splits the string into a list of words based on a delimiter (by default, whitespace characters like spaces, tabs, or newlines).
slsudopass: This is the variable name I had chosen to store the resulting list of words.

sudouserpass=slsudopass[5]

sudouserpass: This is the variable name I had chosen to store the extracted user that had successfully used the sudo command
slsudopass: Explained in 2.1.1, it stores the list of words whom had been split.
[5]: This is indexing notation used to access specific elements within the splatline list. Indexing starts from 0, so [5] refers to the sixth element in the list of slsudopass.

```
sudocommandequal=slsudopass[13]
```

sudocommandequal: This is the variable name I had chosen to store the extracted sudo command that had been successfully used by the user which includes the equal sign.
slsudopass: Explained in 2.1.1, it stores the list of words which had been split.
[13]: This is indexing notation used to access specific elements within the splatline list. Indexing starts from 0, so [13] refers to the 14th element in the list of slsudopass.

```
slsudocommandequal=sudocommandequal
.split('/')
```

slsudocommandequal: Variable name chosen to store the list of strings from the output of the command “sudocommandequal.split(‘/’)
sudocommandequal.split(‘/’): Splits the sudocommandequal variable in to list of substrings using the ‘/’ as a delimiter.

```
sudocommand=slsudocommandequal[-1]
```

sudocommand:Variable name I chosen to shore the sudo command variable used
Slsudocommandequal[-1]: obtains the last object in the slsudocommandequal list.

```
print(sudouserpass, 'used', sudocommand,
'with sudo')
```

Thereafter the information was printed in a human readable format for the user to read.

```
file.close()
```

The file was then closed.

2.2.6) Print ALERT! If users failed to use the sudo command; include the command.

file=open('auth.log','r')
This line of code is used to open a file for reading.
open(): This is a built-in function in Python used to open a file.
'Auth.log': This is the argument passed to the open() function, specifying the name of the file you want to open. In this case, it's auth.log.
'r': This is the second argument passed to the open() function, indicating the mode in which you want to open the file. The 'r' specifies read mode, meaning you intend to read data from the file.

Therefore, by executing this line, I am creating an object named file that is associated with the auth.log file. This file object allows you to interact with the contents of the auth.log file, such as reading its lines or iterating through them.

for elsudofail in file:

for: This keyword introduces a for loop in Python.
elsudofail: This is a variable name that I had chosen to represent each line of text read from the file within the loop whereby a user has failed to use a sudo command.

if 'sudo' and 'command not allowed' in elsudofail:

if: This keyword introduces an if statement, which checks a condition before executing a block of code.
'sudo': This is a string that represents super user permissions.
and: This adds a second condition for the if statement
'command not allowed': This is a string literal representing the sentence 'command not allowed'
in: This keyword is used for membership testing. In this context, it checks if the string "sudo" and "command is not alloed" is present within the current line stored in the variable elsudofail.
elsudofail:: This variable is explained above.

Therefore this command filters for the lines whereby the string ‘sudo’ and ‘command not allowed’ is present.

slsudofail=elsudofail.split()

elsudofail: This refers to the variable that holds a string containing a line of text read from the file (as explained above in 2.1).
.split(): This is a method call on the string object in eachline. The .split() method splits the string into a list of words based on a delimiter (by default, whitespace characters like spaces, tabs, or newlines).
slsudofail: This is the variable name I had chosen to store the resulting list of words.

sudouserfail=slsudofail[5]

sudouserfail: This is the variable name I had chosen to store the user that had attempted to use the sudo command but failed.
slsudofail: Explained in 2.1.1, it stores the list of words whom had been split.
[5]: This is indexing notation used to access specific elements within the splatline list. Indexing starts from 0, so [5] refers to the sixth element in the list.

sudofailcommandequal=slsudofail[17]

sudofailcommandequal: This is the variable name I had chosen to store the extracted command that was attempted to be used with the sudo command with the equal sign.
slsudofail: Explained in 2.1.1, it stores the list of words that has been split.
[17]: This is indexing notation used to access specific elements within the splatline list. Indexing starts from 0, so [17] refers to the 18th element in the list.

slsudofailcommandequal=sudofailcommandequal.split('/')

Slsudofailcommandequal: As explained above
sudofailcommandequal.split(‘/’): uses the ‘/’ as a delimiter to split the sudofailcommandequal string into substrings.

```
sudofailcommand=slsudofailcommandequal[-1]
```

Sudofailcommand: variable name I had chosen to store the sudo command that was attempted to be used
Slsudofailcommandequal[-1]: Obtains the last substring in the slsudofailcommandequal string and stores it in the variable sudofailcommand

```
print('ALERT!', sudouserfail, 'attempted to use the', sudofailcommand, 'with sudo privileges')
```

This prints the Alert, user that attempted to use the sudo command along with the respective command

```
file.close()
```

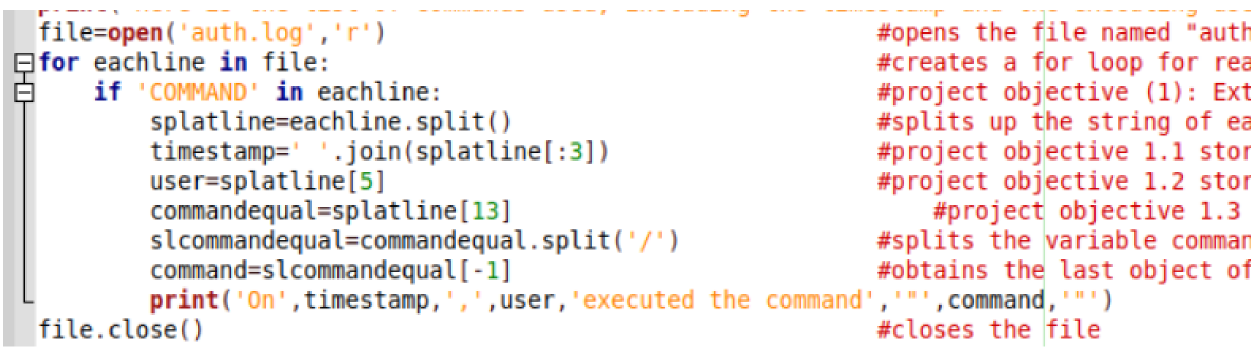
The file was then closed.

3) Discussion

In this chapter, we will evaluate the findings that were described in earlier sections. The methodology, results and process will be evaluated to support the recommendations that will be made in the following chapter.

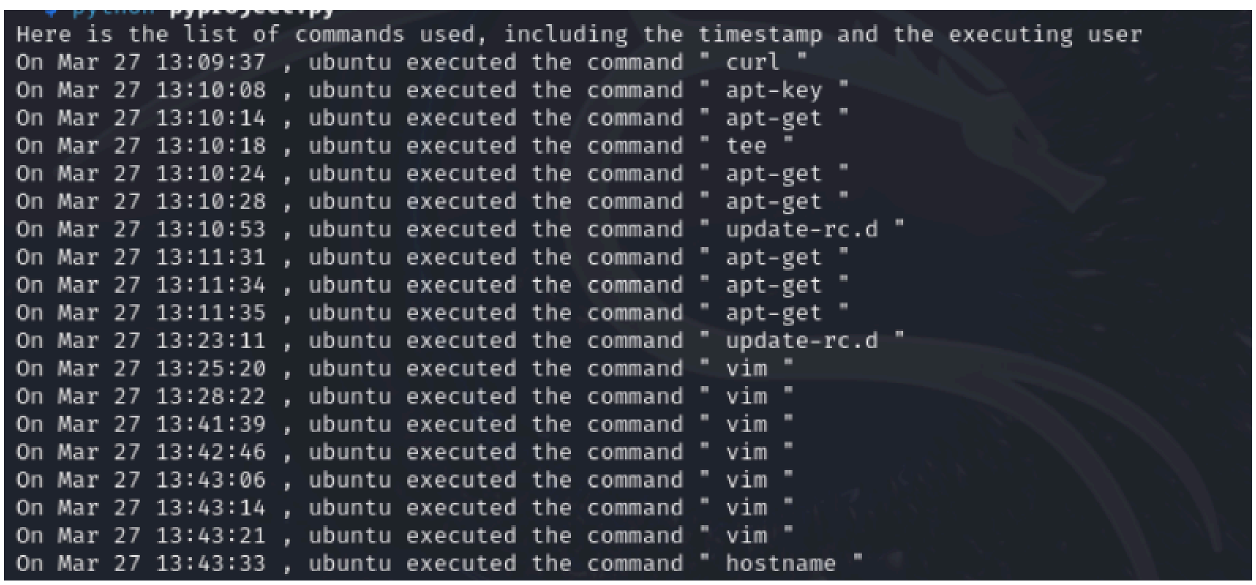
3.1)Log Parse auth.log: Extract command usage

Script:



```
file=open('auth.log','r')                                       #opens the file named "auth
for eachline in file:                                           #creates a for loop for rea
    if 'COMMAND' in eachline:                                   #project objective (1): Ext
        splatline=eachline.split()                              #splits up the string of ea
        timestamp=' '.join(splatline[:3])                       #project objective 1.1 stor
        user=splatline[5]                                       #project objective 1.2 stor
        commandequal=splatline[13]                                  #project objective 1.3
        slcommandequal=commandequal.split('/')                  #splits the variable comman
        command=slcommandequal[-1]                              #obtains the last object of
        print('On',timestamp,',',user,'executed the command','"',command,'"')
file.close()                                                    #closes the file
```

Output:



```
Here is the list of commands used, including the timestamp and the executing user
On Mar 27 13:09:37 , ubuntu executed the command " curl "
On Mar 27 13:10:08 , ubuntu executed the command " apt-key "
On Mar 27 13:10:14 , ubuntu executed the command " apt-get "
On Mar 27 13:10:18 , ubuntu executed the command " tee "
On Mar 27 13:10:24 , ubuntu executed the command " apt-get "
On Mar 27 13:10:28 , ubuntu executed the command " apt-get "
On Mar 27 13:10:53 , ubuntu executed the command " update-rc.d "
On Mar 27 13:11:31 , ubuntu executed the command " apt-get "
On Mar 27 13:11:34 , ubuntu executed the command " apt-get "
On Mar 27 13:11:35 , ubuntu executed the command " apt-get "
On Mar 27 13:23:11 , ubuntu executed the command " update-rc.d "
On Mar 27 13:25:20 , ubuntu executed the command " vim "
On Mar 27 13:28:22 , ubuntu executed the command " vim "
On Mar 27 13:41:39 , ubuntu executed the command " vim "
On Mar 27 13:42:46 , ubuntu executed the command " vim "
On Mar 27 13:43:06 , ubuntu executed the command " vim "
On Mar 27 13:43:14 , ubuntu executed the command " vim "
On Mar 27 13:43:21 , ubuntu executed the command " vim "
On Mar 27 13:43:33 , ubuntu executed the command " hostname "
```

When analysing an auth.log, it is important to be aware of the commands that users use as it allows us to detect security threats.

Monitoring commands help cybersecurity specialists detect malicious activities. By knowing what commands users are running, administrators can spot anomalies that may indicate unauthorised access, attempts to exploit vulnerabilities, or suspicious behaviour.

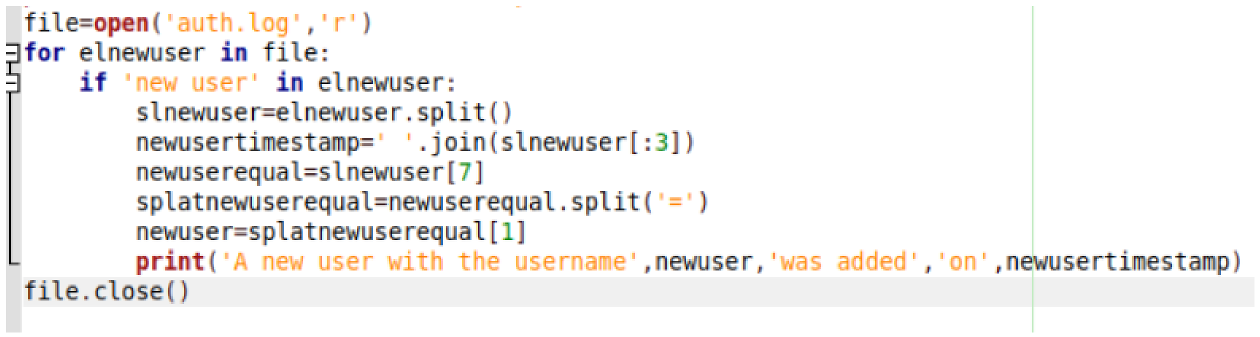
In the event of a security incident, knowing the commands executed can provide insights into the attacker's actions, the extent of compromise, and potential damage. This information is vital for containing and mitigating the incident effectively. For example, knowing what actions that the attacker has done and knowing what files the attacker has accessed will be helpful with data integrity.

After an incident, forensic analysis of command logs can help reconstruct events, identify the root cause, and determine the impact on systems and data. It assists in understanding the attacker's methods and motives. In addition, identifying the area of vulnerability can help aid the cybersecurity team deter against future attacks.

3.2)Monitor user authentication changes.

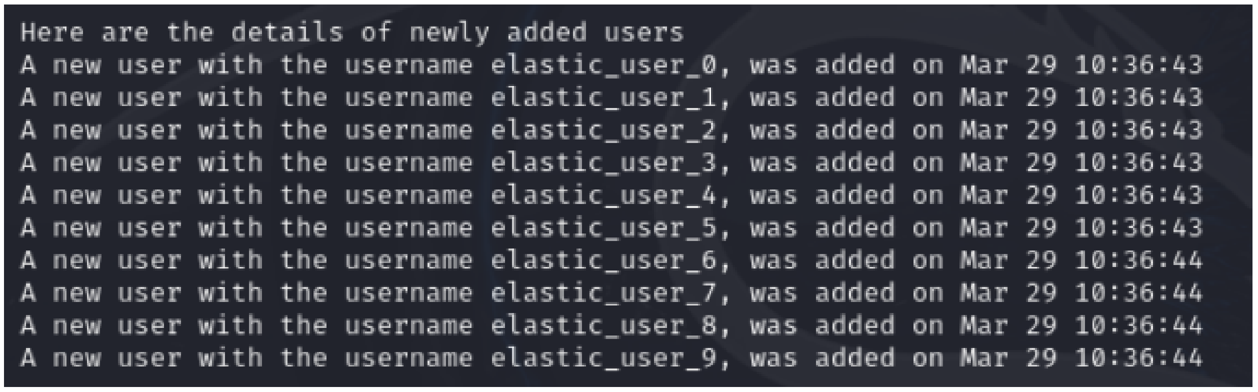
3.2.1) Print details of newly added users, including the Timestamp.

Script



```
file=open('auth.log','r')
for elnewuser in file:
    if 'new user' in elnewuser:
        slnewuser=elnewuser.split()
        newusertimestamp=' '.join(slnewuser[:3])
        newuserequal=slnewuser[7]
        splatnewuserequal=newuserequal.split('=')
        newuser=splatnewuserequal[1]
        print('A new user with the username',newuser,'was added','on',newusertimestamp)
file.close()
```

Output



```
Here are the details of newly added users
A new user with the username elastic_user_0, was added on Mar 29 10:36:43
A new user with the username elastic_user_1, was added on Mar 29 10:36:43
A new user with the username elastic_user_2, was added on Mar 29 10:36:43
A new user with the username elastic_user_3, was added on Mar 29 10:36:43
A new user with the username elastic_user_4, was added on Mar 29 10:36:43
A new user with the username elastic_user_5, was added on Mar 29 10:36:43
A new user with the username elastic_user_6, was added on Mar 29 10:36:44
A new user with the username elastic_user_7, was added on Mar 29 10:36:44
A new user with the username elastic_user_8, was added on Mar 29 10:36:44
A new user with the username elastic_user_9, was added on Mar 29 10:36:44
```

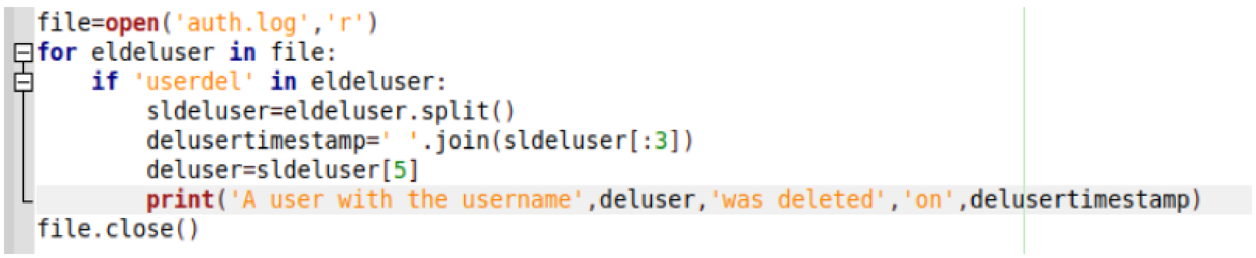
By monitoring and documenting when new users are added to the system, administrators can maintain accountability. This ensures that all user additions are authorised and align with organisational policies. Unauthorised user additions can indicate potential security breaches or insider threats.

Tracking new user creations can also help detect malicious activities. The timestamp is also important in most cases as unexpected user additions outside of business hours may indicate unauthorised access attempts of compromised accounts.

In the event of forensic investigations, timestamps of user creation can also serve as critical forensic evidence, providing investigators with a timeline of events, allowing them to reconstruct the attack scenario and allowing administrators to understand how attackers obtained access to the system.

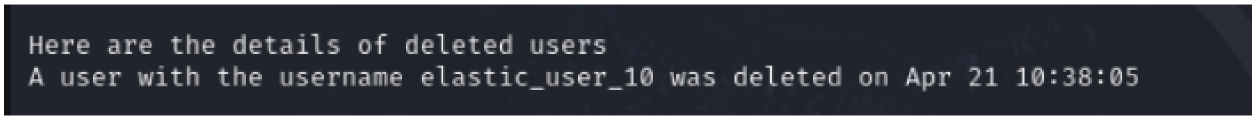
3.2.2) Print details of deleted users, including the Timestamp.

Script



```
file=open('auth.log','r')
for eldeluser in file:
    if 'userdel' in eldeluser:
        sldeluser=eldeluser.split()
        delusertimestamp=' '.join(sldeluser[:3])
        deluser=sldeluser[5]
        print('A user with the username',deluser,'was deleted','on',delusertimestamp)
file.close()
```

Output



```
Here are the details of deleted users
A user with the username elastic_user_10 was deleted on Apr 21 10:38:05
```

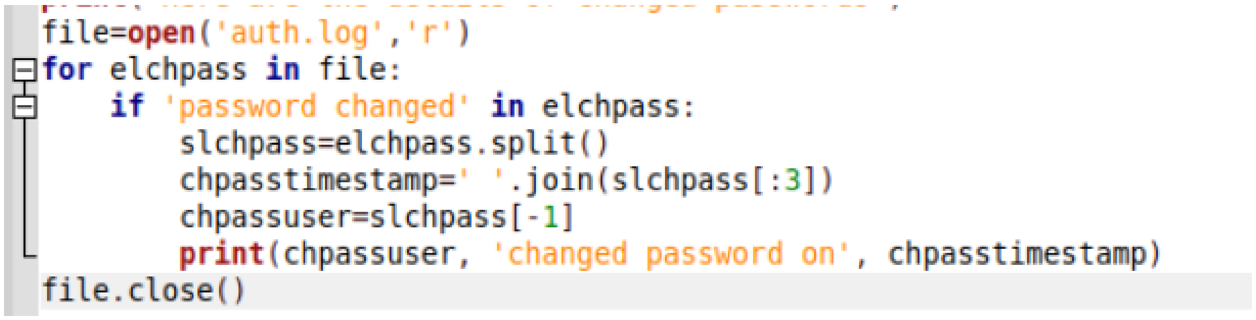
Similar to user addition, user deletion is also critical in security incident detection. Sudden or unexpected deletion of user accounts may indicate malicious activities where attackers may attempt to cover tracks. External attackers deleting accounts may also indicate intentions where they attempt to compromise accounts.

Monitoring user deletions helps detect insider threats where authorised users misuse their privileges. Employees with malicious intent may delete user accounts to disrupt operations, steal data, or cover their tracks. Knowing which users were deleted and when can aid in identifying and mitigating insider threats. According to Ekran (2024), the average cost of insider threat incidents rose from $8.3 million in 2018 to $16.2 million in 2023.

User deletion is also important in non-malicious operations. Ensuring that users are deleted when they leave the organisation reduces the risk of unauthorised access or misuse. In Singapore, a previous employee had accessed the company's computer test system and deleted virtual servers, causing damages of up to $1million (Lim, 2024).

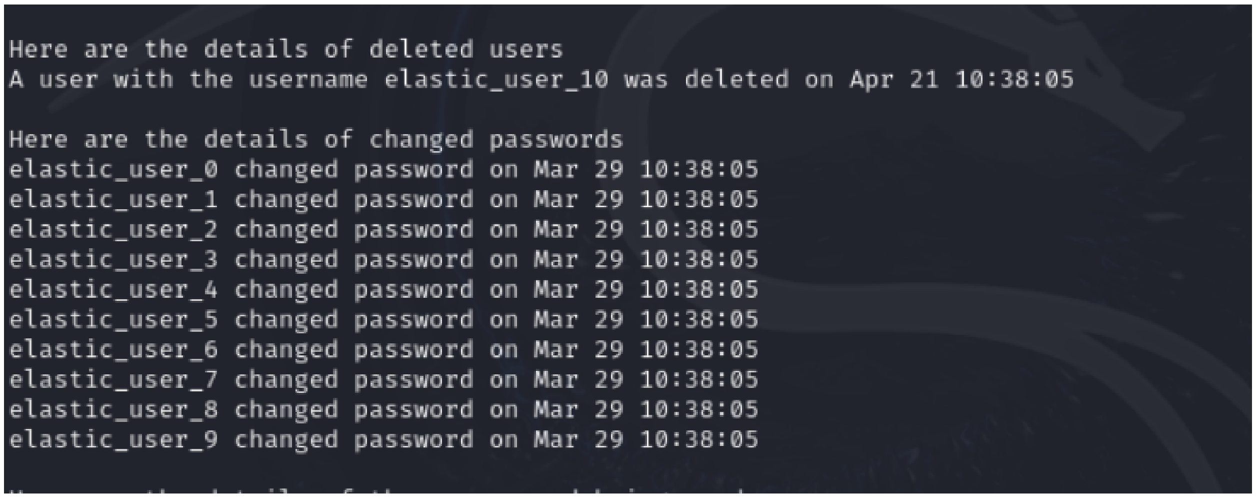
3.2.3)Print details of changing passwords, including the Timestamp.

Script



```
file=open('auth.log','r')
for elchpass in file:
    if 'password changed' in elchpass:
        slchpass=elchpass.split()
        chpasstimestamp=' '.join(slchpass[:3])
        chpassuser=slchpass[-1]
        print(chpassuser, 'changed password on', chpasstimestamp)
file.close()
```

Output:



```
Here are the details of deleted users
A user with the username elastic_user_10 was deleted on Apr 21 10:38:05

Here are the details of changed passwords
elastic_user_0 changed password on Mar 29 10:38:05
elastic_user_1 changed password on Mar 29 10:38:05
elastic_user_2 changed password on Mar 29 10:38:05
elastic_user_3 changed password on Mar 29 10:38:05
elastic_user_4 changed password on Mar 29 10:38:05
elastic_user_5 changed password on Mar 29 10:38:05
elastic_user_6 changed password on Mar 29 10:38:05
elastic_user_7 changed password on Mar 29 10:38:05
elastic_user_8 changed password on Mar 29 10:38:05
elastic_user_9 changed password on Mar 29 10:38:05
```

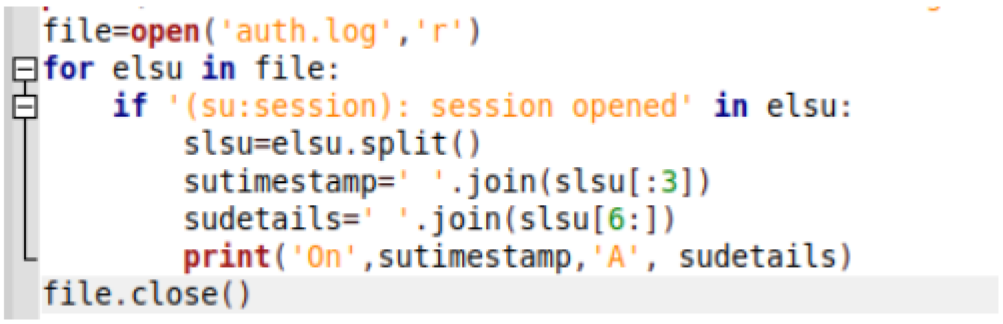
A sudden password change, especially by an administrator for a user's account, could indicate a potential compromise. This might warrant further investigation to ensure the account wasn't hijacked and the password change was unauthorised.

In a non-malicious context, it is important that administrators ensure that passwords are changed regularly (McAfee, 2024).

It is also important for administrators to ensure that passwords are changed immediately after a data breach (McAfee, 2024).

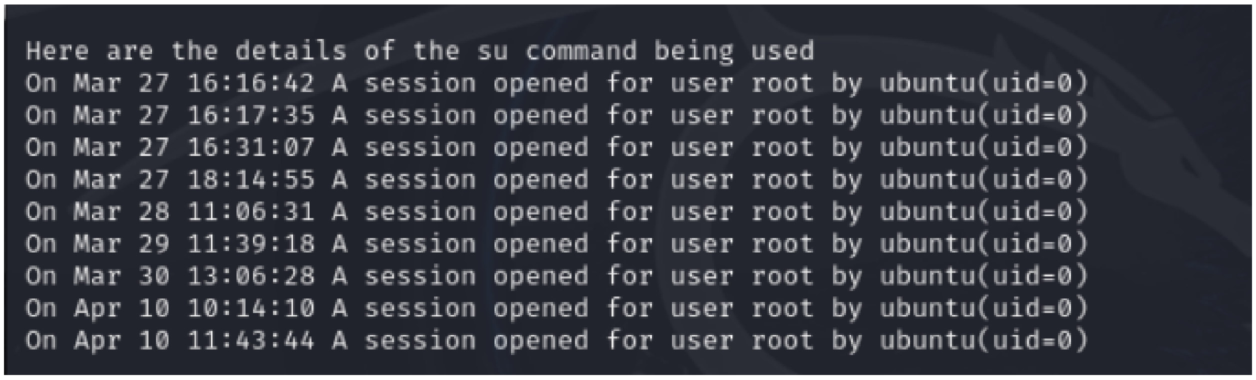
3.2.4)Print details of when users used the su command.

Script



```
file=open('auth.log','r')
for elsu in file:
    if '(su:session): session opened' in elsu:
        slsu=elsu.split()
        sutimestamp=' '.join(slsu[:3])
        sudetails=' '.join(slsu[6:])
        print('On',sutimestamp,'A', sudetails)
file.close()
```

Output



```
Here are the details of the su command being used
On Mar 27 16:16:42 A session opened for user root by ubuntu(uid=0)
On Mar 27 16:17:35 A session opened for user root by ubuntu(uid=0)
On Mar 27 16:31:07 A session opened for user root by ubuntu(uid=0)
On Mar 27 18:14:55 A session opened for user root by ubuntu(uid=0)
On Mar 28 11:06:31 A session opened for user root by ubuntu(uid=0)
On Mar 29 11:39:18 A session opened for user root by ubuntu(uid=0)
On Mar 30 13:06:28 A session opened for user root by ubuntu(uid=0)
On Apr 10 10:14:10 A session opened for user root by ubuntu(uid=0)
On Apr 10 11:43:44 A session opened for user root by ubuntu(uid=0)
```

The su command allows the user the switch to a different account within the same terminal session. It grants the privileges of the target user (usually the root user).

Monitoring auth.log entries for failed su and sudo attempts by unknown users can help detect potential hacking attempts. Knowledge of who used the command and when it was used allows the administrator to identify risky or suspicious activity.

Monitoring su and sudo usage is critical in detecting privilege escalation attempts. Privilege escalation refers to the exploitation of software, operating systems, or network vulnerabilities to gain unauthorised access to resources that are usually restricted to higher-level users or administrators.

According to Cynet (2024), there are two types of privilege escalation.

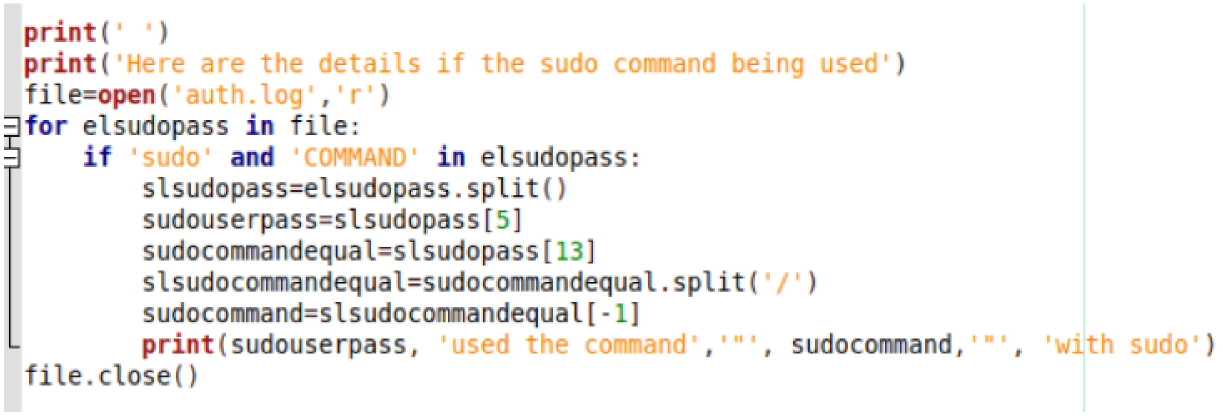
1) Horizontal privilege escalation - where an attacker takes over another account and misuses the privileges granted to the other user.
2) Vertical privilege escalation - where an attacker gains or attempts to gain access to more permissions in an account they have compromised,

In the context of the su command, a sudden increase in su attempts from unusual times or locations could indicate unauthorised access attempts or compromised accounts. Analysing successful su attempts to root or other privileged accounts can help identify potential horizontal privilege escalation attacks, even if they don't involve directly exploiting su.

Monitoring privilege escalation attempts allow attackers to open up new areas of attack on the targeted system. For instance, it can allow the user to gain access to other connected systems, deploy malicious payloads onto the system and could potentially gain root access to a target system or an entire network.

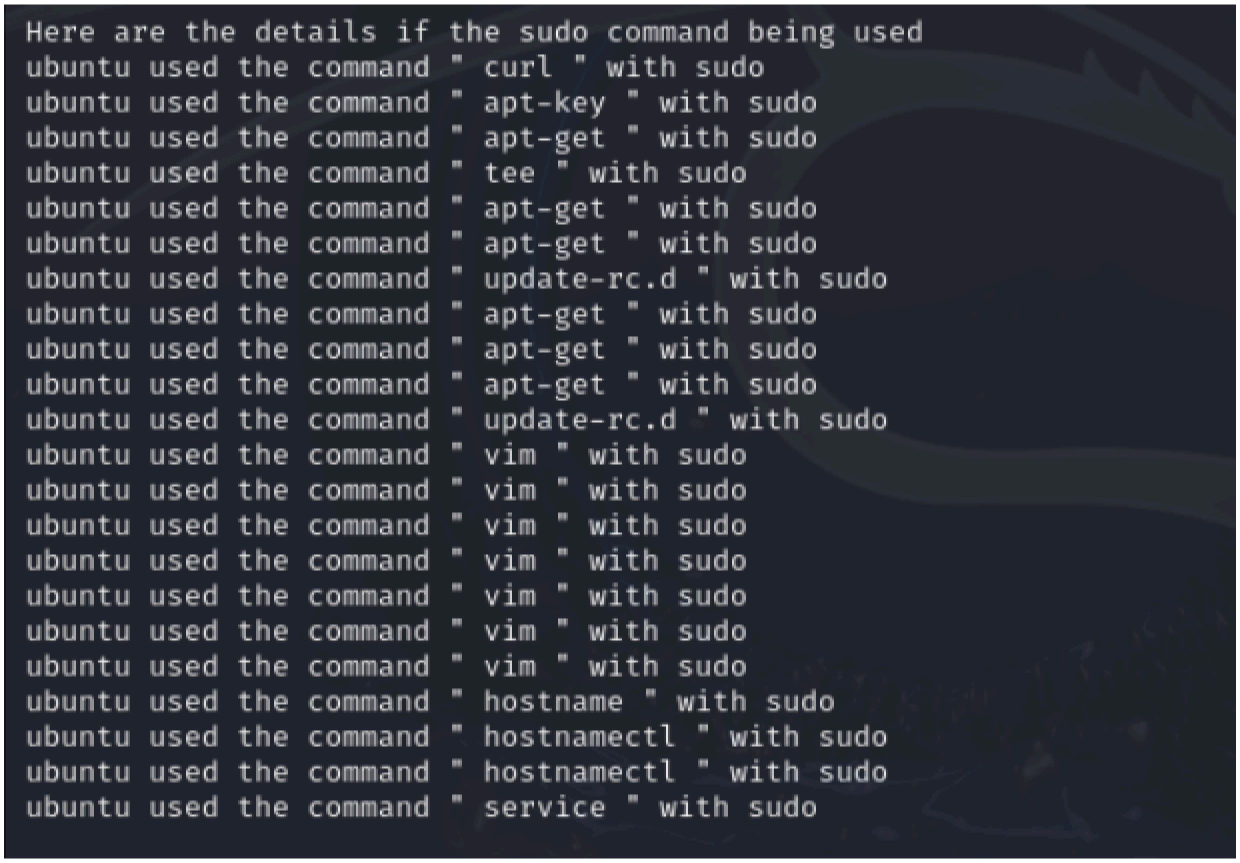
3.2.5)Print details of users who used the sudo; include the command

Script:



```
print(' ')
print('Here are the details if the sudo command being used')
file=open('auth.log','r')
for elsudopass in file:
    if 'sudo' and 'COMMAND' in elsudopass:
        slsudopass=elsudopass.split()
        sudouserpass=slsudopass[5]
        sudocommandequal=slsudopass[13]
        slsudocommandequal=sudocommandequal.split('/')
        sudocommand=slsudocommandequal[-1]
        print(sudouserpass, 'used the command','"', sudocommand,'"', 'with sudo')
file.close()
```

Output:



```
Here are the details if the sudo command being used
ubuntu used the command " curl " with sudo
ubuntu used the command " apt-key " with sudo
ubuntu used the command " apt-get " with sudo
ubuntu used the command " tee " with sudo
ubuntu used the command " apt-get " with sudo
ubuntu used the command " apt-get " with sudo
ubuntu used the command " update-rc.d " with sudo
ubuntu used the command " apt-get " with sudo
ubuntu used the command " apt-get " with sudo
ubuntu used the command " apt-get " with sudo
ubuntu used the command " update-rc.d " with sudo
ubuntu used the command " vim " with sudo
ubuntu used the command " vim " with sudo
ubuntu used the command " vim " with sudo
ubuntu used the command " vim " with sudo
ubuntu used the command " vim " with sudo
ubuntu used the command " vim " with sudo
ubuntu used the command " vim " with sudo
ubuntu used the command " hostname " with sudo
ubuntu used the command " hostnamectl " with sudo
ubuntu used the command " hostnamectl " with sudo
ubuntu used the command " service " with sudo
```

As mentioned in the previous section the unauthorised usage of the sudo command can be seen as vertical privilege escalation

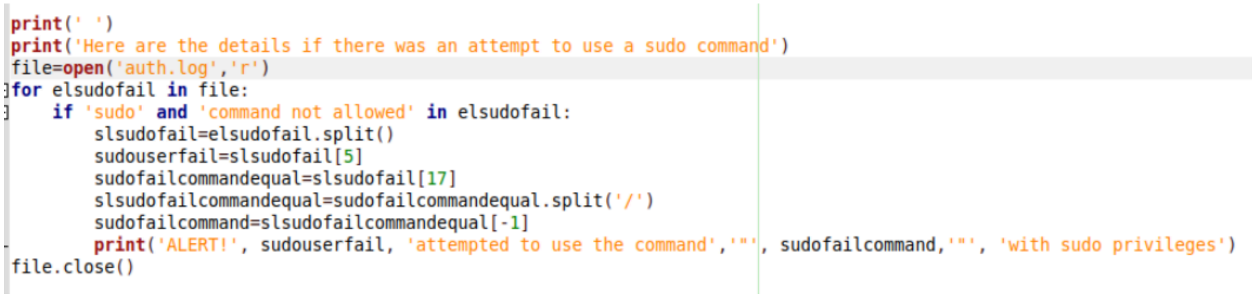
Monitoring such privilege escalation attempts allow attackers to open up new areas of attack on the targeted system. For instance, it can allow the user to gain access to other connected systems, deploy malicious payloads onto the system and could potentially gain root access to a target system or an entire network (Cynet, 2024).

In case of a security breach, analysing sudo commands used by the attacker can provide valuable forensic evidence. It can help understand what actions were taken, what resources were accessed, and potentially identify the attacker's goals.

It is also important to monitor the use of sudo commands as it could give hints to unusual activity. Deviations from typical user behaviour patterns can be spotted by analysing sudo commands. This could indicate compromised accounts or users exceeding their authorised actions.

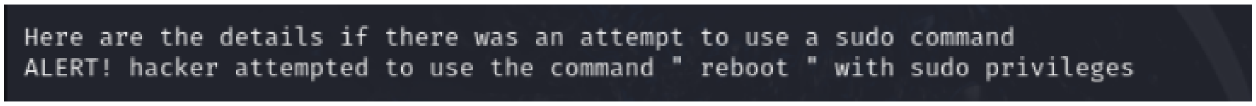
3.2.6)Print ALERT! If users failed to use the sudo command; include the command.

Script



```
print(' ')
print('Here are the details if there was an attempt to use a sudo command')
file=open('auth.log','r')
for elsudofail in file:
    if 'sudo' and 'command not allowed' in elsudofail:
        slsudofail=elsudofail.split()
        sudouserfail=slsudofail[5]
        sudofailcommandequal=slsudofail[17]
        slsudofailcommandequal=sudofailcommandequal.split('/')
        sudofailcommand=slsudofailcommandequal[-1]
        print('ALERT!', sudouserfail, 'attempted to use the command','"', sudofailcommand,'"', 'with sudo privileges')
file.close()
```

Output



```
Here are the details if there was an attempt to use a sudo command
ALERT! hacker attempted to use the command " reboot " with sudo privileges
```

While it is important to know the successful usage of sudo commands, it is also vital that administrators are aware of the failed attempts of using sudo commands.

Continuous failed sudo command attempts from the same user might indicate brute force attacks or systematic attempts to guess the passwords to exploit vulnerabilities. Early detection of failed sudo attempts enables proactive measures to prevent privilege abuse and minimise the risk of unauthorised access or data breaches. Prompt response can mitigate potential damage and strengthen overall cybersecurity defences.

4) Conclusion

In conclusion, this project has successfully developed a tool that parses the /var/log/auth.log file to extract valuable information about system activity and user behaviour. By focusing on both command usage (including timestamps, users, and executed commands) and user authentication changes (new users, deleted users, password changes, su usage, sudo usage with commands, and failed sudo attempts), the tool provides comprehensive insights into potential security threats, system functionality, and user accountability. This information can be used to identify suspicious activity, monitor user access, and maintain a strong security posture for the system.

While we explored the potential threats that may be identified from the data extraction from the auth.log, not all threats are the same. Threat analysis plays an important role in assessing the likelihood of a threat occuring and the potential damage that it could cause. This allows the organisation to prioritise security investments and focus on the most critical risks.

5) Recommendations

Following the extraction of data from auth.log, interpretation of the data is also vital in the context of cybersecurity, more particularly in the area of threat analysis. The following steps should be taken following the data extraction.

1) Pattern identification

Pattern identification is the process of recognizing recurring or distinctive arrangements within data. This includes the understanding of a baseline level of activity in the auth.log. For instance, it might not be unusual for a user to execute commands in a surveillance organisation, but it would be for organisations that only operate during business hours. This usually involves analysing historical data.

2) Identifying deviations

Deviations from the norm will then be identified once the baselines have been established. This could mean multiple attempts to execute sudo commands or a sudden spike in users being created.

3) Prioritisation or threat analysis

After observing the deviations from baselines, the organisation needs to prioritise the level of suspicious activity based on their severity, potential impact and relevance to its individual security policies. The Common Vulnerability Scoring System (CVSS) is a standardised method for scoring the severity of security vulnerabilities (FIRST, n.d.). The CVSS is based on six base metrics:

1) Access vector - measures how remote an attacker can be to attack a target.
2) Access Complexity - measures the complexity of attack required to exploit the vulnerability once an attacker has gained access to the target system.
3) Authentication - measures the number of times an attacker must authenticate to the target system
4) Confidentiality - measures the impact on confidentiality of a successful exploit of the vulnerability on the target system.
5) Integrity - measures the impact on integrity of a successful exploit of the vulnerability on the target system.
6) Availability - measures the impact on availability of a successful exploit of the vulnerability on the target system.

Though CVSS is not specific to auth.log analysis, it can play a supporting role in helping the team prioritise potential threats within the log.

In addition to analysis of the auth.log, other logs like System logs and Application logs (e.g. Apache access logs) could provide a more comprehensive view of threats.

References

Ekran. (2024). 7 real-life data breaches caused by insider threats | ekran system. Retrieved from https://www.ekransystem.com/en/blog/real-life-examples-insider-threat-caused-breaches

FIRST. (n.d.). Frequently asked questions. Retrieved from https://www.first.org/cvss/v1/faq

Lim, R. (2024). Fired employee accessed company’s computer “test system” and deleted servers, causing it to lose S$918,000. Retrieved from https://www.channelnewsasia.com/singapore/former-employee-hack-ncs-delete-virtual-servers-quality-testing-4402141

McAfee. (2024). How often should you change your passwords? Retrieved from https://www.mcafee.com/learn/how-often-should-you-change-your-passwords/#:~:text=Cybersecurity%20experts%20recommend%20changing%20your,has%20access%20to%20your%20account.