Python Project - Log Analyzer
Shaun Sng (S17)
17 Apr 2024

Table of Contents

1. Introduction

Summary - This project mainly entails developing a Python script with the following scope:

1.1 Parse auth.log: Extract command usage, with timestamp, executing user and command.

1.2 Parse auth.log: Monitor user authentication changes.

- Print details of newly added users, including the Timestamp.
- Print details of deleted users, including the Timestamp.
- Print details of changing passwords, including the Timestamp.
- Print details of when users used the su command.
- Print details of users who used the sudo; include the command.
- Print ALERT! If users failed to use the sudo command; include the command.

Aim - This project aims to primarily test student's Python scripting knowledge, as well as to build up knowledge and familiarity with critical log files such as auth.log. Beyond the format of the file itself, the aim is also to sensitise students to important information and approaches that may be required when processing and analysing logs.

2. Methodologies

2.1 Read log file, prepare libraries

The /var/log/auth.log file is opened with an absolute file path, so that the script can be run from any directory in a linux system. *Readlines* method is used to store the log as a *list* in the variable *data.* The list format facilitates slicing to extract specific pieces of information required such as the timestamp and related user.

Deliberate output or print to screen is done *"checking for executed commands…"* so that the user is prompted that the script is working. We include empty lines and *sleep* for visual spacing and to slow down the script for the user to absorb the information.

The time library is imported to allow use of *sleep,* and Python shebang to facilitate conversion to an executable file.

```
1  #!/usr/bin/env python3
2
3  # importing time library to use sleep for readability.
4
5  import time
6
7  # Read var/log/auth.log. Open with readlines. List format facilitates parsing.
8  # Commented out earlier test auth2.log.
9  # raw = open("auth2.log", 'r')
10
11 raw = open("/var/log/auth.log", 'r')
12 data = raw.readlines()
13
14 # Output to screen to show user script is running.
15
16 print("\n")
17 print("1. Checking for executed commands:\n")
18 time.sleep(1)
19
```

```
┌──(kali㉿kali)-[~/python/proj]
└─$ ./project.py


1. Checking for executed commands:
```

2.2 Checking for executed commands

A for loop is used to check through each line in auth.log, using the keyword “COMMAND” to extract portions where command execution is recorded. List slicing is used to extract the executing user and timestamp, saved into variables *exec_user* and *exec_date*. The information is output to screen in a single sentence using f-string. For improved readability, key information is bold or coloured.

```
20  # Extracting record of commands executed, by searching for 'COMMAND' keyword.
21  # Processing each line twice into x and y variable.
22  # Split using 'COMMAND=' easier to extract command.
23  # Split without arguments facilitates slicing to get Timestamp and User.
24
25  for line in data:
26      if 'COMMAND' in line:
27          y = line.split()
28          x = line.split('COMMAND=')
29          full_command=x[-1][:-1]
30          exec_user=y[3]
31          exec_date=y[0]
32
33          # Print results in sentence for readability, highlighting key info.
34          print(f"User \033[1m{exec_user}\033[0m executed the command:\033[96m{full_command}\033[0m on \033[1m{exec_date}\033[0m")
35
```

```
1. Checking for executed commands:

User kali executed the command:/usr/sbin/adduser tempuser on 2024-04-10T01:51:47.989792-04:00
User kali executed the command:/usr/sbin/deluser tempuser on 2024-04-10T01:52:50.058529-04:00
User kali executed the command:/usr/sbin/adduser tempuser2 on 2024-04-10T02:31:38.272044-04:00
User kali executed the command:/usr/bin/zsh on 2024-04-10T02:34:06.152317-04:00
User kali executed the command:/usr/bin/netstat -tapn on 2024-04-10T02:35:24.271583-04:00
User optimus executed the command:/usr/bin/cat auth.log on 2024-04-10T03:19:55.473027-04:00
User optimus executed the command:/usr/bin/cat auth.log on 2024-04-10T03:22:25.840689-04:00
User kali executed the command:/usr/bin/netstat -tapn on 2024-04-10T04:06:26.888265-04:00
User kali executed the command:/usr/bin/netstat -tapn on 2024-04-10T04:07:32.571424-04:00
```

Additional Processing - Split method was run twice for each line and saved into x and y variables for further processing. Firstly, split was run without any argument so that the elements of each line are broken up into many elements in the list. This facilitates easy list slicing to extract the **timestamp** and **user**. Second, split is run using the delimiter “COMMAND=”. This allows easy extracting of the full command, which is the last element (index -1) in this created list. This allows us to treat all commands as one chunk and not to deal with the variation in spacing, as some commands are longer/shorter than others.

```
 ; COMMAND=/usr/sbin/adduser linuxsec
 ; COMMAND=/usr/bin/cat /etc/shadow
 ; COMMAND=/usr/bin/cat /etc/shadow -n
 ; COMMAND=/usr/bin/apt-get install net-tools
 ; COMMAND=/usr/bin/apt-get install vsftpd
 ; COMMAND=/usr/bin/apt-get install apache2
 ; COMMAND=/usr/bin/apt-get install whois
 ; COMMAND=/usr/bin/apt-get install geoip-bin
 ; COMMAND=/usr/bin/apt-get update
 ; COMMAND=/usr/bin/apt-get install cmatrix
 ; COMMAND=/usr/bin/cat /etc/shadow
R=root ; COMMAND=/usr/bin/apt install xclip
 ; COMMAND=/usr/bin/apt-get install xclip
D=/home/linuxsec ; USER=root ; COMMAND=/usr/sbin/service --status

R=root ; COMMAND=/usr/sbin/service --status-all
R=root ; COMMAND=/usr/sbin/service ssh status
 ; COMMAND=/usr/bin/ssh stop
 ; COMMAND=/usr/sbin/service ssh stop
 ; COMMAND=/usr/bin/nano sshd_config
 ; COMMAND=/usr/sbin/service ssh stop
 ; COMMAND=/usr/bin/nano sshd_config
 ; COMMAND=/usr/sbin/service ssh start
 ; COMMAND=/usr/bin/cp /var/log/auth.log .
```

2.3 Checking for added and deleted users, password changes.

A similar for loop is used to screen the data, extracting where creation of new users was captured by using the keyphrase **'new user'**. For each stage/question, the user is prompted of the ongoing checks, and the output slowed down using *sleep* to facilitate reading.

```
36  # Prompt user again, to also slow down the output.
37
38  print("\n")
39  print("2. Checking for newly added users:\n")
40  time.sleep(1)
41
42  # Straightforward check for newly added user by searching for keyphrase 'new user'.
43
44  for line in data:
45      if 'new user' in line:
46          line=line.split()
47          new_user=line[5][5:-1]
48          new_date=line[0]
49          print(f"New user \033[96m{new_user}\033[0m was added on \033[1m{new_date}\033[0m")
50
```

```
2. Checking for newly added users:

New user tempuser was added on 2024-04-10T01:51:48.122365-04:00
New user tempuser2 was added on 2024-04-10T02:31:38.321815-04:00
```

Selecting Keyword - Applying appropriate keywords is a critical initial step to extract the appropriate portions of the log. These words or phrases were determined by studying the auth.log file, and experimenting with options. For example, logs related to new user information can also be captured using "useradd". However using this keyword would retrieve additional records we are not interested in, such as *adding a user to groups*.

```
┌──(kali㉿kali)-[~/python/proj]
└─$ cat auth.log | grep useradd
Sep 30 14:03:45 server useradd[828]: new group: name=tc, GID=1000
Sep 30 14:03:45 server useradd[828]: new user: name=tc, UID=1000, GID=1000, home=/home/tc, shell=/bin/bash, from=none
Sep 30 14:03:45 server useradd[828]: add 'tc' to group 'adm'
Sep 30 14:03:45 server useradd[828]: add 'tc' to group 'cdrom'
Sep 30 14:03:45 server useradd[828]: add 'tc' to group 'sudo'
Sep 30 14:03:45 server useradd[828]: add 'tc' to group 'dip'
Sep 30 14:03:45 server useradd[828]: add 'tc' to group 'plugdev'
Sep 30 14:03:45 server useradd[828]: add 'tc' to group 'lxd'
Sep 30 14:03:45 server useradd[828]: add 'tc' to shadow group 'adm'
Sep 30 14:03:45 server useradd[828]: add 'tc' to shadow group 'cdrom'
Sep 30 14:03:45 server useradd[828]: add 'tc' to shadow group 'sudo'
Sep 30 14:03:45 server useradd[828]: add 'tc' to shadow group 'dip'
Sep 30 14:03:45 server useradd[828]: add 'tc' to shadow group 'plugdev'
Sep 30 14:03:45 server useradd[828]: add 'tc' to shadow group 'lxd'
Sep 30 14:03:47 server useradd[1311]: new user: name=lxd, UID=999, GID=100, home=/var/snap/lxd/common/lxd, shell=/bin/false, from=none
Sep 30 14:06:34 server useradd[1743]: new user: name=linuxsec, UID=1001, GID=1001, home=/home/linuxsec, shell=/bin/bash, from=/dev/pts/0
Sep 30 14:26:24 server useradd[2396]: new user: name=ftp, UID=114, GID=120, home=/srv/ftp, shell=/usr/sbin/nologin, from=none
```

The **'new user'** keyword was determined to be selective enough, while providing all the info we require in a single line.

```
2. Checking for newly added users:

2024-04-10T01:51:48.122365-04:00 kali useradd[548894]: new user: name=tempuser, UID=1003, GID=1003, home=/home/tempuser, shell=/bin/
bash, from=/dev/pts/4

New user tempuser was added on 2024-04-10T01:51:48.122365-04:00
2024-04-10T02:31:38.321815-04:00 kali useradd[568426]: new user: name=tempuser2, UID=1003, GID=1003, home=/home/tempuser2, shell=/bi
n/bash, from=/dev/pts/4

New user tempuser2 was added on 2024-04-10T02:31:38.321815-04:00
```

The same approach was used to extract log information for deleted users, using the keyphrase **'delete user'**.

```
54  print("\n")
55  print("3. Checking for deleted users:\n")
56  time.sleep(1)
57
58  # Extracting log record of deleted users, using keyphrase 'delete user'.
59
60  for line in data:
61      if 'delete user' in line:
62          line=line.split()
63          # The log format for deleted user has quotation marks. Quick slicing to remove quotes.
64          deleted_user=line[-1][1:-1]
65          del_date=line[0]
66          print(f"User \033[96m{deleted_user}\033[0m was deleted on \033[1m{del_date}\033[0m")
67
68  # Usual prompt for each stage.
69
70  print("\n")
```

```
3. Checking for deleted users:

User tempuser was deleted on 2024-04-10T01:52:50.135700-04:00
```

The same approach was used to extract log information for password changes, using the keyphrase **'password changed'**.

```
74  # Extracting log record of password change using keyphrase 'password changed'.
75
76  for line in data:
77      if 'password changed' in line:
78          line=line.split()
79          pass_user=line[-1]
80          pass_date=line[0]
81          print(f"Password was changed for user \033[96m{pass_user}\033[0m on \033[1m{pass_date}\033[0m")
82
83  # Usual prompt for each stage.
```

```
4. Checking for password change:

Password was changed for user tempuser on 2024-04-10T01:51:55.044944-04:00
Password was changed for user tempuser2 on 2024-04-10T02:31:43.049631-04:00
Password was changed for user tempuser2 on 2024-04-10T02:34:22.654438-04:00
```

2.4 Checking for SU commands

The general approach remains similar when checking for SU commands. However, unlike earlier segments there is no obvious single keyword or phrase that can be used. Using ‘su’ alone results in many hits.

```
2024-04-10T03:21:45.897845-04:00 kali su: pam_unix(su:auth): authentication failure; logname=kali uid=1001 euid=0 tty=/dev/pts/4 ruser=optim
us rhost=  user=shaun
2024-04-10T03:21:48.199748-04:00 kali su[593048]: FAILED SU (to shaun) optimus on pts/4
2024-04-10T03:21:57.643588-04:00 kali su[593121]: (to shaun) optimus on pts/4
2024-04-10T03:21:57.644960-04:00 kali su[593121]: pam_unix(su:session): session opened for user shaun(uid=1002) by kali(uid=1001)
2024-04-10T03:22:06.323360-04:00 kali su[593213]: (to optimus) shaun on pts/4
2024-04-10T03:22:06.327367-04:00 kali su[593213]: pam_unix(su:session): session opened for user optimus(uid=1001) by kali(uid=1002)
2024-04-10T03:22:25.840689-04:00 kali sudo:  optimus : TTY=pts/4 ; PWD=/var/log ; USER=root ; COMMAND=/usr/bin/cat auth.log
2024-04-10T03:22:25.841251-04:00 kali sudo: pam_unix(sudo:session): session opened for user root(uid=0) by kali(uid=1001)
2024-04-10T03:22:25.842502-04:00 kali sudo: pam_unix(sudo:session): session closed for user root
2024-04-10T03:23:50.321639-04:00 kali su[594062]: (to kali) optimus on pts/4
2024-04-10T03:23:50.323038-04:00 kali su[594062]: pam_unix(su:session): session opened for user kali(uid=1000) by kali(uid=1001)
2024-04-10T04:06:18.119300-04:00 kali sudo: pam_unix(sudo:auth): authentication failure; logname=kali uid=1000 euid=0 tty=/dev/pts/4 ruser=k
```

The records of interest are those rows with the target user/account in parenthesis “**(to optimus)**”. To extract only these rows, we search for lines matching both keywords - **‘su’** and **‘to’** and filter out failed attempts.

```
89  # List of keywords to narrow the search, extract only one line per SU command. Drop other "duplicate" lines related to the same
90  su_success=['su','to']
91
92  print("Successful SU command execution are:\n")
93
94  for line in data:
95      # checking if all keywords appear in line.
96      if all(x in line for x in su_success):
97          # filtering out the failed attempts
98          if 'FAILED' not in line:
99              line=line.split()
100             # print(line)
101             start = line[-3]
102             target = line[-4][:-1]
103             su_time=line[0]
104             print(f"From user \033[96m{start}\033[0m to user \033[1m{target}\033[0m on {su_time}")
105
```

```
5. Checking for use of su command:

Successful SU command execution are:

From user kali to user optimus on 2024-04-10T02:33:47.086330-04:00
From user optimus to user kali on 2024-04-10T02:34:00.548819-04:00
From user root to user kali on 2024-04-10T02:34:45.240693-04:00
From user kali to user optimus on 2024-04-10T03:18:50.947573-04:00
From user optimus to user shaun on 2024-04-10T03:21:57.643588-04:00
From user shaun to user optimus on 2024-04-10T03:22:06.323360-04:00
From user optimus to user kali on 2024-04-10T03:23:50.321639-04:00
```

Extracting failed su attempts is simpler by using the keyphrase ‘FAILED SU.’

```
108  time.sleep(2)
109  print("\n")
110  print("Unsuccessful SU attempts were:\n")
111
112  # Extracting unsuccessful SU using keyphrase 'FAILED SU'.
113
114  for line in data:
115      if 'FAILED SU' in line:
116          line=line.split()
117          # print(line)
118          start = line[-3]
119          target = line[-4][:-1]
120          su_time=line[0]
121          print(f"From user \033[96m{start}\033[0m to user \033[1m{target}\033[0m on {su_time}")
122
123
124  # Prompt for last batch of SUDO command.
```

```
Unsuccessful SU attempts were:

From user kali to user root on 2024-04-10T02:33:11.714103-04:00
From user kali to user root on 2024-04-10T02:33:22.297732-04:00
From user kali to user optimus on 2024-04-10T02:33:40.862938-04:00
From user optimus to user shaun on 2024-04-10T03:21:48.199748-04:00
```

2.5 Check SUDO commands

The same approach is used for checking sudo commands. We specify two keywords each for both successful and failed execution of sudo - saved in two different lists sudo_good and sudo_bad.

Similar to section 2.2, each line is processed/split twice in order to extract the command.

```
# Similar to SU section. List of keywords to check for both successful (good) and failed (bad) SUDO command execution.
sudo_good =['sudo','COMMAND']
sudo_bad =['sudo','incorrect password']

print("Following users and commands were successfully executed with SUDO privilege:\n")

# Extracting successful SUDO execution.
for line in data:
    if all(x in line for x in sudo_good):
        y = line.split()
        x = line.split('COMMAND=')
        sudo_command=x[-1][:-1]
        exec_user=y[3]
        sudo_time=y[0]
        print(f"User \033[96m{exec_user}\033[0m executed sudo \033[1m{sudo_command}\033[0m on {su_time}")

print("\n")
```

```
5. Checking for use of SUDO command:

Following users and commands were successfully executed with SUDO privilege:

User kali executed sudo /usr/sbin/adduser tempuser on 2024-04-10T03:21:48.199748-04:00
User kali executed sudo /usr/sbin/deluser tempuser on 2024-04-10T03:21:48.199748-04:00
User kali executed sudo /usr/sbin/adduser tempuser2 on 2024-04-10T03:21:48.199748-04:00
User kali executed sudo /usr/bin/zsh on 2024-04-10T03:21:48.199748-04:00
User kali executed sudo /usr/bin/netstat -tapn on 2024-04-10T03:21:48.199748-04:00
User optimus executed sudo /usr/bin/cat auth.log on 2024-04-10T03:21:48.199748-04:00
User optimus executed sudo /usr/bin/cat auth.log on 2024-04-10T03:21:48.199748-04:00
User kali executed sudo /usr/bin/netstat -tapn on 2024-04-10T03:21:48.199748-04:00
User kali executed sudo /usr/bin/netstat -tapn on 2024-04-10T03:21:48.199748-04:00
```

Failed sudo commands are of particular interest. A helpful **“incorrect password”** phrase helps to zero in on these records.

```
print("\n")
print("Failed execution of SUDO commands were:\n")
time.sleep(2)

# Extracting failed SUDO execution.
for line in data:
    if all(x in line for x in sudo_bad):
        # print(line)
        y = line.split()
        x = line.split('COMMAND=')
        sudo_command=x[-1][:-1]
        exec_user=y[3]
        sudo_time=y[0]
        print(f"\033[31mALERT!\033[0m User \033[96m{exec_user}\033[0m attempted to execute sudo \033[1m{sudo_command}\033[0m on {su_time} with incorrect password.")
```

```
Failed execution of SUDO commands were:

ALERT! User kali attempted to execute sudo /usr/bin/netstat -tapn on 2024-04-10T03:21:48.199748-04:00 with incorrect password.
ALERT! User kali attempted to execute sudo /usr/bin/netstat -tapn on 2024-04-10T03:21:48.199748-04:00 with incorrect password.
```

3. Discussion

This section elaborates on challenges encountered in scripting and discusses some of the limitations as a result of design choices.

3.1 Timestamp

Auth.log files from previous exercises and labs have a shorter timestamp - only the month, day and time separated with spaces. When split into a list these became different elements.

However, the “live” var/log/auth.log in Kali Linux, timestamp has a slightly different format which is *more detailed* and also a single unbroken string without spaces. For the purpose of this project, we opt to process the log based on our live Kali system. However, If the formatting/configuration is different on other systems, the output returned by the scripts would be different/truncated or not readable.

Auth.log files from previous projects/scenarios - simpler	Auth.log from user’s Kali shows more detailed timestamp
Feb 17 06:59:57 server sudo: Feb 17 07:00:10 server sudo: h start Feb 17 07:00:10 server sudo: 000) Feb 17 07:00:10 server sshd[3 Feb 17 07:00:10 server sshd[3 Feb 17 07:00:10 server sudo: Feb 17 07:00:42 server sshd[3 y user Feb 17 07:00:42 server sshd[3 Feb 17 07:00:42 server sshd[3 Feb 17 07:00:42 server system Feb 17 07:00:42 server system Feb 17 07:17:01 server CRON[3 Feb 17 07:17:01 server CRON[3 Feb 17 07:24:36 server sshd[3 Feb 17 07:29:20 server sudo: /auth.log . Feb 17 07:29:20 server sudo: 000)	2024-04-10T03:55:01.497350-04:00 kali CRON[6 2024-04-10T04:05:01.504322-04:00 kali CRON[6 id=0) by (uid=0) 2024-04-10T04:05:01.506115-04:00 kali CRON[6 2024-04-10T04:06:18.119300-04:00 kali sudo: 1000 euid=0 tty=/dev/pts/4 ruser=kali rhost= 2024-04-10T04:06:26.888265-04:00 kali sudo: r/log ; USER=root ; COMMAND=/usr/bin/netstat 2024-04-10T04:07:29.032638-04:00 kali sudo: 1000 euid=0 tty=/dev/pts/4 ruser=kali rhost= 2024-04-10T04:07:32.570445-04:00 kali sudo: 2024-04-10T04:07:32.571284-04:00 kali sudo: i] 2024-04-10T04:07:32.571424-04:00 kali sudo: /log ; USER=root ; COMMAND=/usr/bin/netstat

While initially the intention was to use the **full timestamp** without further adjustment, it was decided to process the information for improved readability, similar to the sample/example shared by the trainer. A function was created to parse this

timestamp and output in the readable DD-MM-YYYY hh:mm:sss. While the script was updated, we have retained the earlier screenshots in this report.

```
14  def trim_date(stamp):
15      date = stamp.split('T')[0]
16      day = date[8:10]
17      month = date [5:7]
18      year = date[0:4]
19      time= stamp.split('T')[1][0:8]
20
21      cleaned = day + "-" + month + "-" + year + " " + time
22
```

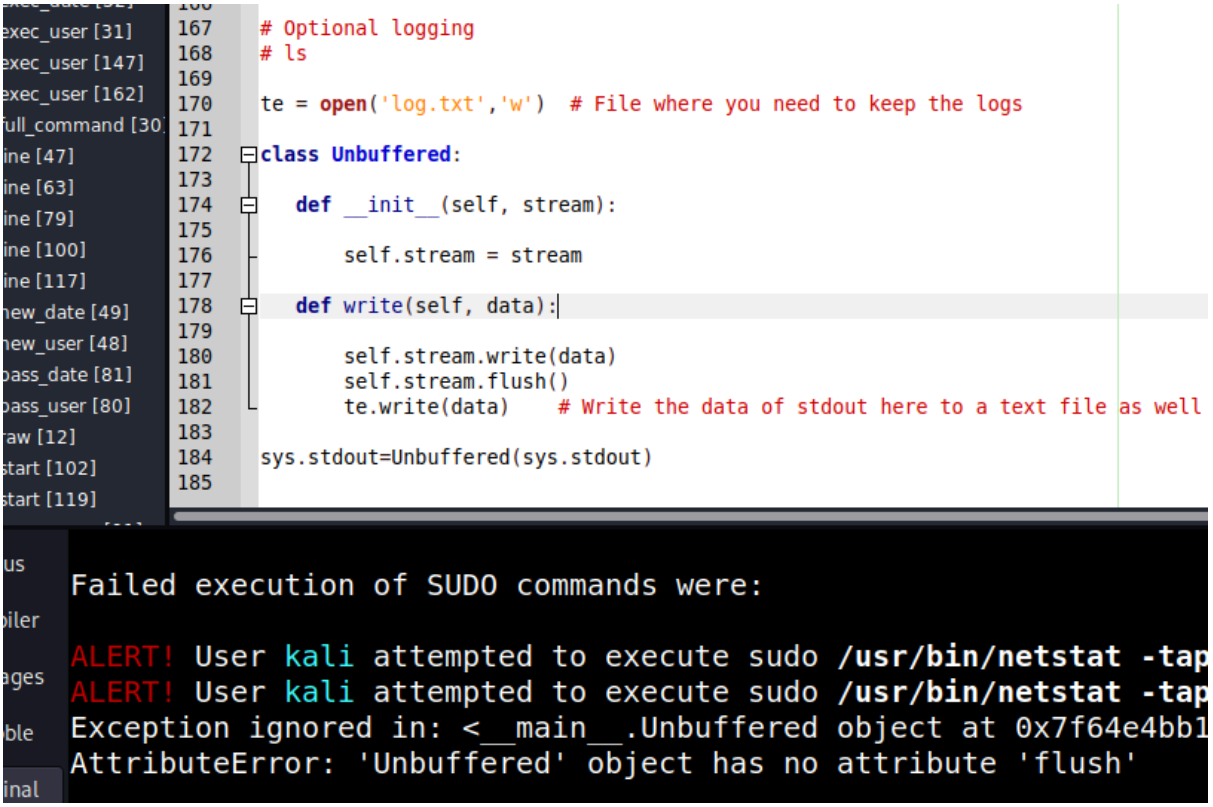
```
1. Checking for executed commands:

User kali executed the command:/usr/sbin/adduser tempuser on 10-04-2024 01:51:47
User kali executed the command:/usr/sbin/deluser tempuser on 10-04-2024 01:52:50
User kali executed the command:/usr/sbin/adduser tempuser2 on 10-04-2024 02:31:38
User kali executed the command:/usr/bin/zsh on 10-04-2024 02:34:06
```

3.2 Design

One of the more notable inefficiencies of the current script is that for almost all sections the core framework and process is the same - running a similar For Loop referencing the same data. One potential improvement of the script is to perhaps work this into a function and call the function using different arguments.

3.3 Logging

Currently the script outputs results to screen. This is somewhat limited, especially if there is a significant amount of information. Ideally the results should also be saved/output to file. Brief research was done to attempt to build this into the script but was not successful.



One workaround option is for the user to execute the script and output to file manually (>> python filename.py > output.log).

4. Conclusion & Recommendations

The project is a basic but necessary exercise in Python scripting, and familiarisation with auth.log. There remains room for improvement to increase the reliability and efficiency with the script, build-in logging and explore other approaches and tools for log analysis.

5. References

Various websites that provided guidance on this project are listed below.

1. https://stackoverflow.com/questions/3389574/check-if-multiple-strings-exist-in-another-string
2. https://stackoverflow.com/questions/14906764/how-to-redirect-stdout-to-both-file-and-console-with-scripting
3. https://unix.stackexchange.com/questions/164826/date-command-iso-8601-option
4. https://stackoverflow.com/questions/42710386/how-to-execute-a-python-script-and-write-output-to-a-file