

# Project: Log Analyzer

Prepared by:

S18

# Table of content

Introduction	3
Python Scripting	3
Discussion	7
Conclusion	11
References	12

# Introduction

Monitoring the 'auth.log' file is crucial in cybersecurity operations as it provides key insights into authentication activities that help maintain system security. This log records important details such as successful and failed login attempts, user sessions, and authentication events. By examining 'auth.log', organisations can spot unauthorised access attempts, identify potential security breaches, and conduct detailed investigations to understand what happened during incidents. Regular monitoring helps meet regulatory requirements and allows organisations to improve their security by quickly reacting to suspicious activities and taking necessary steps to protect sensitive data and systems from unauthorised access.

## Python Scripting

The 'auth.log' file records system authorization events, including user logins and authentication mechanisms. It is usually found in the /var/log folder on a Linux machine. Specifying the file path of 'auth.log' ensures consistent referencing in the script. A sample 'auth.log' file is used for demonstration, facilitating testing and development without relying on real-time log data. Analysts can modify 'file\_path' to fit their system configuration.

```
# Define file path for auth.log. auth.log in a fixed path for easier reference
file_path = '/home/kali/Desktop/python/project/auth.log'
```

Figure 1: 'File\_path' can be modified based on the user's system configuration.

### File access and initialization

To open the file at file\_path in read mode ('r'), use Python's 'open()' function inside a 'with' statement. The 'with' statement is important because it automatically closes the file when done, even if there is an error. This makes the code easier to read and prevents file problems.

Before reading, you may need to reset the file's position to the start using 'file.seek(0)'. This ensures the entire file is read from the beginning.

To read the file contents, use 'file.readlines()'. This reads the entire file and stores each line in a list called 'authlog\_content'. This makes it easy to search for patterns, extract information, and analyse the log data.

```

# Read content of auth.log file
def openfile():
    with open(file_path, 'r') as file:
        # Making sure that it's always starting from the top regardless
        file.seek(0)

        # Read as list for text manipulation
        authlog_content = file.readlines()

```

Figure 2: Script for reading of 'auth.log' file

## 'Re' module

Python includes a built-in package known as 're', designed for working with Regular Expressions (RegEx), which are useful tools for searching and manipulating text strings. A Regular Expression is essentially a sequence of characters defining a specific search pattern. This module allows users to define rules for matching sets of strings, such as identifying email addresses or timestamps within a larger text body. The primary functions utilised in this script include 're.compile()', 're.match()', and 're.search()'

## Details of newly added users

The 're.compile()' function prepares a regular expression pattern into a reusable regex object, optimising the efficiency of pattern matching tasks, such as extracting timestamps.

In the context of the 'auth.log' file, the fixed string 'new user: name=' is what the Regex engine searches for within the input text. Here, 'w' matches any alphanumeric character (letters and numbers: a-zA-Z0-9), while the '+' symbol indicates that one or more of these characters are expected. Therefore, 'w+' in this scenario matches any sequence of alphanumeric characters, representing the name of the newly created user in the log entry.

```

4406 Jul 1 14:33:19 server groupadd[3323]: new group: name=newuser, GID=1003
4407 Jul 1 14:33:19 server useradd[3329]: new user: name=newuser, UID=1003, GID=1003,
home=/home/newuser, shell=/bin/bash, from=/dev/pts/1
4408 Jul 1 14:33:26 server passwd[3340]: pam_unix(passwd:chauthtok): password changed
0 Jun 29 09:15:21 server useradd[1396]: new group: name=Pris, GID=1003
1 Jun 29 09:15:21 server useradd[1396]: new user: name=Pris, UID=1003, GID=1003,
home=/home/Pris, shell=/bin/sh, from=/dev/pts/1
2 Jun 29 09:15:21 server sudo: pam_unix(sudo:session): session closed for user r

```

Figure 3: Fixed string 'new user: name=' in 'auth.log' that shows new users were added.

```
# 2.1 Print details of newly added users, including Timestamp

newuser = re.compile(r'new user: name=(\w+)')

for line in authlog_content:
    timestamp = extract_timestamp(line)

    if newuser.search(line):
        new_users.append(line.strip())

print('Newly Added Users')
for entry in new_users:
    print(entry)
```

Figure 4: Script to print out the lines for newly added users.

The script searches for 'new user: name=' in each line of 'auth.log' using 're.search()'. If a match is found, the line is appended to the 'new\_users' list. 'strip()' removes extra whitespaces. The results are printed in a readable format.

### Details of deleted users

For deleted users, the script matches the string 'delete user' and appends the results to 'deleted\_users'.

```
# 2.2 Details of deleted users, including Timestamp

deluser = re.compile(r'delete user')

for line in authlog_content:
    # ~ timestamp = extract_timestamp(line)

    if deluser.search(line):
        deleted_users.append(line.strip())

print('Deleted Users')
for entry in deleted_users:
    print(entry)
```

Figure 5: Script to match 'delete user', then print the results in a readable format.

```
root(uid=0) by tc(uid=1000)
4461 Jul  1 14:40:29 server userdel[3441]: delete user 'newuser'
4462 Jul  1 14:40:29 server userdel[3441]: removed group 'newuser'
```

Figure 6: Script to identify 'delete user' and display the results in an easily readable format.

### Details of users with password changes

The script searches for 'password changed' to find users who changed passwords.

```
27 Jun 29 09:24:21 server passwd[1445]: pam_unix(passwd:chauthtok): password changed
for joker

4408 Jul  1 14:33:26 server passwd[3340]: pam_unix(passwd:chauthtok): password changed
for newuser
```

Figure 6: The fixed string 'password changed' in 'auth.log' displays users who have updated their password.

## Details of usage of 'su' commands

The 'su' command allows a user to switch accounts. '\bsu\b' specifies the word boundary, ensuring the script searches for 'su' commands only.

```
sucommand = re.compile(r'\bsu\b')
```

Figure 7: Script to display only the 'su' commands.

## Details of users who used sudo

'Sudo' allows users to execute commands with elevated privileges, providing the necessary permissions to perform tasks that require higher-level access to the system.

'.' is a regular expression pattern that matches any sequence of characters, no matter their type or length. This flexibility allows it to capture any text that appears between 'sudo' and 'COMMAND='. 'COMMAND=' is a fixed string present in the 'auth.log' file, and it indicates the exact command that was executed with elevated privileges.

The parentheses () in the regular expression create a capturing group, which means they group part of the pattern and save the matched text for later use. This is beneficial when you need to extract specific parts of the matched text for further processing or analysis.

```
sudocommand = re.compile(r'sudo.*COMMAND=(.*)')
```

Figure 8: Script to extract 'sudo' command and the commands used with 'sudo'

## Details if users failed to use 'sudo' command

This script searches for 'authentication failure', 'incorrect password', and 'not', printing entries if results are found.

```
sudoalert = re.compile(r'(sudo:.*(authentication failure|Failed password|NOT))')
```

Figure 9: Script for instances where users failed to use 'sudo' command

# Discussion

Regular review of 'auth.log' essential for monitoring unusual activity. Analysts can identify unauthorised access attempts, suspicious logins, and reconstruct attack chains. Extracting timestamps provides insights into the timeline of events, which is crucial for cyber attack investigations.

By parsing 'auth.log', analysts can detect unauthorised access attempts such as failed login attempts, brute-force attacks, or suspicious login activities. During a security breach, analysts can reconstruct an attack chain, answering questions like what happened, when it happened, where it occurred, who is responsible, whether their actions were successful, and what the results were.

Extracting timestamps is vital as it offers valuable insights into the timeline of events and activities on a system. In the case of a cyber attack, analysts can quickly establish a timeline of events, identify the sequence of actions taken by attackers, and reconstruct the attack chain.

In this example, the document simulates attacks on a server and how parsing a log file can help quickly extract data.

## Details of addition and deletion of users

```
Newly Added Users
Jun 29 09:15:21 server useradd[1396]: new user: name=Pris, UID=1003, GID=1003, home=/home/Pris, shell=/bin/sh, from=/dev/pts/1
Jun 29 09:39:05 server useradd[2059]: new user: name=pris, UID=1003, GID=1003, home=/home/pris, shell=/bin/bash, from=/dev/pts/1
Jul 1 14:33:19 server useradd[3329]: new user: name=newuser, UID=1003, GID=1003, home=/home/newuser, shell=/bin/bash, from=/dev/pts/1
Jul 2 11:46:46 server useradd[1474]: new user: name=newuser1, UID=1003, GID=1003, home=/home/newuser1, shell=/bin/bash, from=/dev/pts/1
Jul 2 12:53:08 server useradd[2076]: new user: name=joker, UID=1002, GID=1002, home=/home/joker, shell=/bin/bash, from=/dev/pts/1
Jul 2 12:56:47 server useradd[2151]: new user: name=clown, UID=1003, GID=1003, home=/home/clown, shell=/bin/bash, from=/dev/pts/1

Deleted Users
Jun 29 09:20:48 server userdel[1422]: delete user 'Pris'
Jun 29 09:40:03 server userdel[2087]: delete user 'pris'
Jul 1 14:40:29 server userdel[3441]: delete user 'newuser'
Jul 2 12:50:08 server userdel[1913]: delete user 'joker'
Jul 2 12:51:39 server userdel[1979]: delete user 'newuser1'
Jul 2 12:59:43 server userdel[2207]: delete user 'clown'
Jul 2 13:00:26 server userdel[2231]: delete user 'joker'
```

Figure 10: Terminal display of the data of newly added users and deleted users extracted from 'auth.log'

In this simulation, the user 'joker' was added as a new user on July 2 at 12:53:08 and deleted shortly after at 13:00:26, with 'clown' also being created and deleted within that timeframe. This rapid sequence of user creation and deletion suggests a potential security breach, where an attacker may have created unauthorised accounts to gain system access.

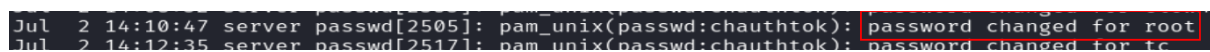
By monitoring new and deleted user accounts, analysts can swiftly detect unauthorised or unexpected activities initiated by attackers attempting to exploit

system vulnerabilities. Unauthorised users pose a threat to data integrity by potentially manipulating or corrupting data. Monitoring deletions ensures that legitimate users retain access to essential information, preserving operational continuity.

Maintaining a record of these activities establishes an audit trail that can be reviewed to ensure the integrity of the system and data, confirming that no unauthorised modifications have occurred. This practice helps verify that the data remains accurate and secure from unauthorised alterations. In the event of a security breach, such as a denial-of-service attack, attackers may create numerous user accounts to overwhelm system resources, leading to disruptions for legitimate users. Prompt detection and response to such activities can prevent availability issues and ensure continuous system functionality.

## Password changes

Parsing password changes is essential for security teams to maintain a detailed record, documenting who altered passwords, when these changes occurred, and which accounts were impacted. This meticulous record is invaluable during forensic investigations following a security breach, enabling teams to trace and comprehend the sequence of events leading to the incident.

A terminal window showing two lines of log output. The first line is 'Jul 2 14:10:47 server passwd[2505]: pam\_unix(passwd:chauthtok): password changed for root' and the second line is 'Jul 2 14:12:35 server passwd[2517]: pam\_unix(passwd:chauthtok): password changed for tc'. The text 'password changed for root' in the first line is highlighted with a red rectangular box.

```
Jul 2 14:10:47 server passwd[2505]: pam_unix(passwd:chauthtok): password changed for root
Jul 2 14:12:35 server passwd[2517]: pam_unix(passwd:chauthtok): password changed for tc
```

Figure 11: Terminal display of password change for root

In instances where an unauthorised individual gains access and changes a user's password (as depicted in the above figure for the root user), the consequences can be severe. The legitimate user will immediately lose access to their account, resulting in productivity losses and potential disruptions to critical operations. If the compromised account holds sudo privileges, the attacker can execute commands with full system access, posing a substantial threat to system integrity. They could potentially manipulate sensitive data, establish persistent access through backdoors, and introduce malware or ransomware. Signs of such a compromise include unexpected password change alerts, numerous failed login attempts, unusual account behaviour, and the appearance of unrecognised new accounts.



## Su commands

```
Jul 2 14:09:40 server su: pam_unix(su:auth): authentication failure; logname=clown uid=1002 euid=0 tty=/dev/pts/0
ruser=clown rhost= user=tc
Jul 2 14:09:42 server su: FAILED SU (to tc) clown on pts/0
Jul 2 14:09:52 server su: pam_unix(su:auth): authentication failure; logname=clown uid=1002 euid=0 tty=/dev/pts/0
ruser=clown rhost= user=tc
Jul 2 14:09:54 server su: FAILED SU (to tc) clown on pts/0
Jul 2 14:10:01 server su: pam_unix(su:auth): authentication failure; logname=clown uid=1002 euid=0 tty=/dev/pts/0
ruser=clown rhost= user=tc
Jul 2 14:10:02 server su: FAILED SU (to tc) clown on pts/0
Jul 2 14:10:09 server su: (to tc) clown on pts/0
Jul 2 14:10:09 server su: pam_unix(su:session): session opened for user tc(uid=1000) by clown(uid=1002)
Jul 2 14:12:52 server su: pam_unix(su:auth): authentication failure; logname=clown uid=1000 euid=0 tty=/dev/pts/0
ruser=tc rhost= user=clown
Jul 2 14:12:54 server su: FAILED SU (to clown) tc on pts/0
Jul 2 14:12:59 server su: (to clown) tc on pts/0
Jul 2 14:12:59 server su: pam_unix(su:session): session opened for user clown(uid=1002) by clown(uid=1000)
Jul 2 14:13:44 server su: (to tc) clown on pts/0
Jul 2 14:13:44 server su: pam_unix(su:session): session opened for user tc(uid=1000) by clown(uid=1002)
Jul 2 14:14:22 server su: (to clown) tc on pts/0
```

Figure 12: Terminal display of attempts of 'su' into user tc.

In the depicted scenario, user 'clown' attempted multiple times to switch user ('su') into 'tc', encountering authentication failures until July 2 at 14:10:09, where the log shows that user 'clown' successfully switched to 'tc' as indicated by the message 'session opened for user tc(uid=1000) by clown(uid=1002)'. This indicates that 'clown' initiated a session for 'tc', who possesses sudo privileges.

The repeated authentication failures strongly suggest a brute force attack, where 'clown' persistently tried to guess 'tc's password until succeeding. After gaining access as 'tc', the attacker attempted to switch back to 'clown', possibly to cover their tracks or exploit the system further using 'tc's elevated privileges. This successful switch poses a significant security risk because 'tc' can execute commands with heightened permissions, potentially compromising the entire system.

To mitigate these risks, immediate actions should include isolating the affected system, changing passwords, and revoking unauthorised access. A thorough investigation should follow, involving log analysis and vulnerability assessment. Enhancing security measures like implementing multi-factor authentication, conducting regular audits, and educating users on security best practices are crucial. Deploying intrusion detection systems and reviewing incident response processes will further strengthen the organisation's defences against future attacks.

## udo commands

By analysing sudo commands, analysts can reconstruct the sequence of events during a breach and discern what an attacker may seek with administrative privileges.

```
Jul 2 14:13:07 server sudo:    clown : user NOT in sudoers ; TTY=pts/0 ; PWD=/home/tc ; USER=root ; COMMAND=/usr/
bin/ls
Jul 2 14:13:15 server sudo:    clown : user NOT in sudoers ; TTY=pts/0 ; PWD=/home/tc ; USER=root ; COMMAND=/usr/
bin/passwd
Jul 2 14:13:38 server sudo:    clown : user NOT in sudoers ; TTY=pts/0 ; PWD=/var/log ; USER=root ; COMMAND=/usr/
bin/cat auth.log
Jul 2 14:14:14 server sudo:    tc : TTY=pts/0 ; PWD=/home/tc ; USER=root ; COMMAND=/usr/sbin/usermod -ag sudo
clown
Jul 2 14:14:31 server sudo:    clown : TTY=pts/0 ; PWD=/home/tc ; USER=root ; COMMAND=/usr/bin/ls
Jul 2 14:17:40 server sudo:    tc : TTY=pts/0 ; PWD=/home/clown ; USER=root ; COMMAND=/usr/sbin/deluser clown
Jul 2 14:17:47 server sudo:    tc : TTY=pts/0 ; PWD=/home/clown ; USER=root ; COMMAND=/usr/bin/rm -rf clown
Jul 2 14:18:01 server sudo:    tc : TTY=pts/0 ; PWD=/home ; USER=root ; COMMAND=/usr/bin/rm -rf clown
```

Figure 13: Terminal display of user using sudo commands

In the scenario described, 'clown' attempted to gain elevated privileges before being granted sudo access. This suggests an unauthorised attempt to perform actions reserved for privileged users without proper authorization. Such actions pose risks of a security breach, where unauthorised attempts could be part of a larger effort to compromise system security, access sensitive information, or execute unauthorised commands that disrupt or harm systems. There is also concern about insider threats, where employees misuse their access rights to undermine organisational security.

Additionally, successful sudo commands by 'tc' granting 'clown' administrative privileges by adding them to the sudoers group expose the system to potential misuse if 'clown's account is compromised, highlighting the importance of monitoring and controlling administrative access to mitigate such risks effectively.

## Failed sudo commands

Parsing failed sudo commands is essential for effectively detecting and responding to security incidents. Instances of failed sudo attempts can indicate various security threats, including brute-force attacks, attempts to guess credentials, or unauthorised access efforts. By parsing and analysing logs containing failed sudo commands, security teams can identify patterns of suspicious behaviour, take corrective actions to strengthen security measures, and mitigate risks associated with unauthorised access or compromised credentials.

```
ALERT! Jul  2 12:52:50 server sudo:      tc : 3 incorrect password attempts ; TTY=pts/0 ; PWD=/home/tc ; USER=root ; COMMAND=/usr/sbin/adduser joker
ALERT! Jul  2 12:52:58 server sudo: pam_unix(sudo:auth): authentication failure; logname=tc uid=1000 euid=0 tty=/dev/pts/0 ruser=tc rhost= user=tc
ALERT! Jul  2 12:53:35 server sudo:      joker : user NOT in sudoers ; TTY=pts/0 ; PWD=/home/tc ; USER=root ; COMMAND=/usr/bin/ls
ALERT! Jul  2 14:13:07 server sudo:      clown : user NOT in sudoers ; TTY=pts/0 ; PWD=/home/tc ; USER=root ; COMMAND=/usr/bin/ls
```

Figure 14: Terminal display of failed sudo commands

In the displayed terminal output, multiple failed sudo attempts and authentication issues are evident. The logs reveal repeated incorrect password entries for user 'tc', suggesting potential brute-force or unauthorised access attempts that could compromise the account if successful. There are also authentication failures indicating possible misconfigurations or compromised credentials for 'tc', possibly exploited by attackers attempting to escalate privileges. Additionally, users 'joker' and 'clown' encountered "user NOT in sudoers" errors, indicating unauthorised attempts to elevate privileges. These incidents indicate malicious actors attempting to gain administrative access from compromised low-privilege accounts, posing risks of system compromise, data breaches, or further exploitation upon success. Continuous monitoring, robust password policies, and immediate investigation of such incidents are critical to mitigating these risks and safeguarding the integrity of the system.

# Conclusion

In summary, parsing the 'auth.log' file is crucial for maintaining comprehensive cybersecurity measures. This log file serves as a vital record of authentication activities, enabling security teams to detect and respond to unauthorised access attempts swiftly. By analysing 'auth.log', organisations can identify suspicious login patterns, detect potential insider threats, and investigate security incidents effectively. This proactive monitoring not only aids in enforcing access controls and ensuring compliance with regulatory standards but also strengthens overall security posture by enabling prompt incident response and mitigation of risks associated with unauthorised access or compromised credentials. Continuous parsing and analysis of 'auth.log' support organisations in safeguarding sensitive data, maintaining system integrity, and mitigating the impact of cybersecurity threats.

## References:

Cobb M. (2023). Security log management and logging best practices. [online]. TechTarget. Available at:

<https://www.techtarget.com/searchsecurity/tip/Security-log-management-and-logging-best-practices#:~:text=Because%20logs%20contain%20details%20of,attack%20has%20affected%20IT%20resources>

Friedl, J. (2009). Mastering regular expressions. [ebook]. Python. Available at: <https://docs.python.org/3/library/re.html> (Accessed 28 June 2024)

Hule V. (2021). Python compile regex pattern using re.compile(). [online]. PYNative. Available at: <https://pynative.com/python-regex-compile/> (Accessed: 29 June 2024)

Knupfer F. (2024). Authentication log monitoring.[online]. Newrelic. Available at: <https://newrelic.com/blog/best-practices/authentication-log-monitoring#:~:text=Security%20monitoring%3A%20Authentication%20logs%20allow,that%20may%20indicate%20security%20breaches> (Accessed 29 June 2024)

Kuchling A.M. (N.D). Regular Expression HOWTO.[online]. Python. Available at: <https://docs.python.org/3/howto/regex.html> (Accessed 01 July 2024)

Matt B. (2017). Torvalds Tuesday:auth.log. [online]. Medium. Available at: <https://bromiley.medium.com/torvalds-tuesday-sample-post-bde3d5c6d05e> (Accessed: 01 July 2024)

n.a(2018).Regular Expressions (Regex). [online]. NTU. Available at: [https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html#:~:text=The%20%5Cs%20\(lowercase%20s%20\),%2C%20i.e.%2C%20non%2Dwhitespace.](https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html#:~:text=The%20%5Cs%20(lowercase%20s%20),%2C%20i.e.%2C%20non%2Dwhitespace.) (Accessed 29 June 2024)

n.a(2023).Brute-Force Attack. [online]. sosafe.Avaliable at: <https://sosafe-awareness.com/glossary/brute-force-attack/#:~:text=Everyone%2C%20from%20individuals%20to%20businesses,identity%20theft%20to%20malware%20dissemination.> (Accessed 02 July 2024)

PeanutsMonkey (2011). \b work when using regular expressions?.[online]. Available at: <https://stackoverflow.com/questions/7605198/how-does-b-work-when-using-regular-expressions> (Accessed 29 June 2024)

Reddy R. (2024). Why is user access review important?.[online]. Zluri. Available at: <https://www.zluri.com/blog/why-is-user-access-review-important/#:~:text=Conducting>

[%20user%20access%20reviews%20is.%2C%20vendors%2C%20and%20third%20parties](#) (Accessed 29 June 2024)

Shea S. (2023). Best practices to conduct a user access review.[online]. TechTarget. Available at:

<https://www.techtarget.com/searchsecurity/answer/How-to-conduct-a-periodic-user-access-review-for-account-privileges> (Accessed: 1 July 2024)

Sheldon R.(2023). sudo(su'do').[online]. TechTarget. Available at:

<https://www.techtarget.com/searchsecurity/definition/sudo-superuser-do#:~:text=Sudo%20is%20a%20command%2Dline,run%20under%20their%20regular%20accounts> (Accessed 29 June 2024)

User4647247 (2015).Python regex match month, date, time.[online]. Stackoverflow. Available at:

<https://stackoverflow.com/questions/30490087/python-regex-match-month-date-time> (Accessed: 01 July 2024)

Yegulalp S. (2021). Unleash the power of python regular expression. [online]. Infoworld. Available at:

<https://www.infoworld.com/article/3608409/unleash-the-power-of-python-regular-expressions.html#:~:text=Match%20objects%20in%20Python%20regex,the%20match%20from%20the%20string>. (Accessed 01 July 2024)