Project Name

SOC-SIEM-Incident-Detection-System

STEP 1: Log Collection & Understanding

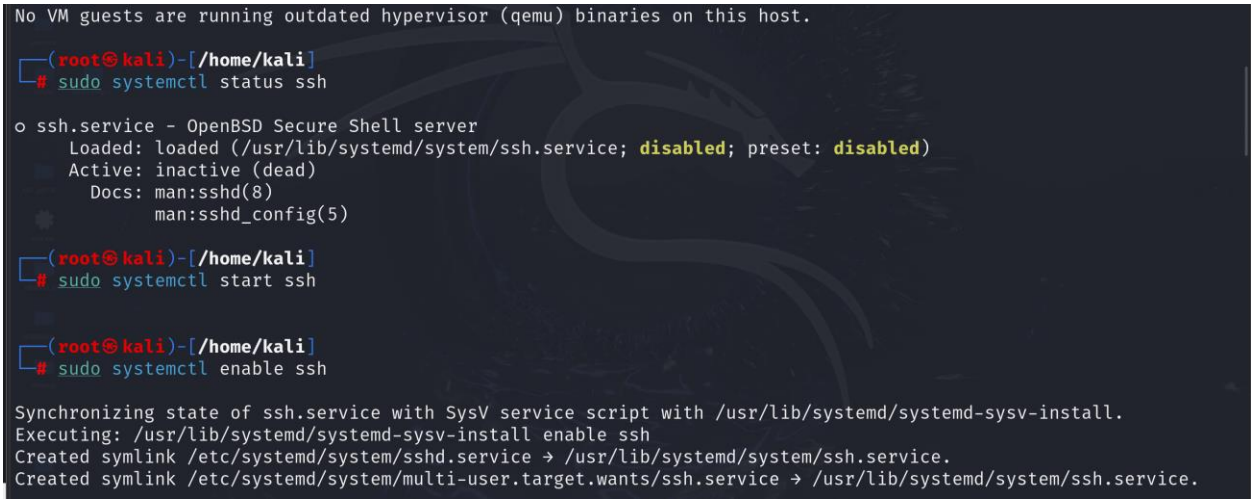
Collect sample logs like:

- SSH login attempts
- Web server access logs

Solution:

1. SSH Service Setup

- Screenshot showing:
    - systemctl start ssh
    - systemctl enable ssh
    - systemctl status ssh showing it's active



2. SSH Connection Attempts

- Screenshot showing:
    - ssh kali@192.168.58.128
    - Host authenticity prompt
    - Failed password attempts
    - Permission denied message

```
┌──(root㉿kali)-[/home/kali]
└─# ssh kali@192.168.58.128/24

ssh: Could not resolve hostname 192.168.58.128/24: Name or service not known

┌──(root㉿kali)-[/home/kali]
└─# ssh kali@192.168.58.128

The authenticity of host '192.168.58.128 (192.168.58.128)' can't be established.
ED25519 key fingerprint is SHA256:8AATQrkmYCvK2NMGSQPbIKLJnNfNYhPSlnK3mUx5NEs.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: y
Please type 'yes', 'no' or the fingerprint: y
Please type 'yes', 'no' or the fingerprint: y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '192.168.58.128' (ED25519) to the list of known hosts.
kali@192.168.58.128's password:
Permission denied, please try again.
kali@192.168.58.128's password:
Connection closed by 192.168.58.128 port 22
```

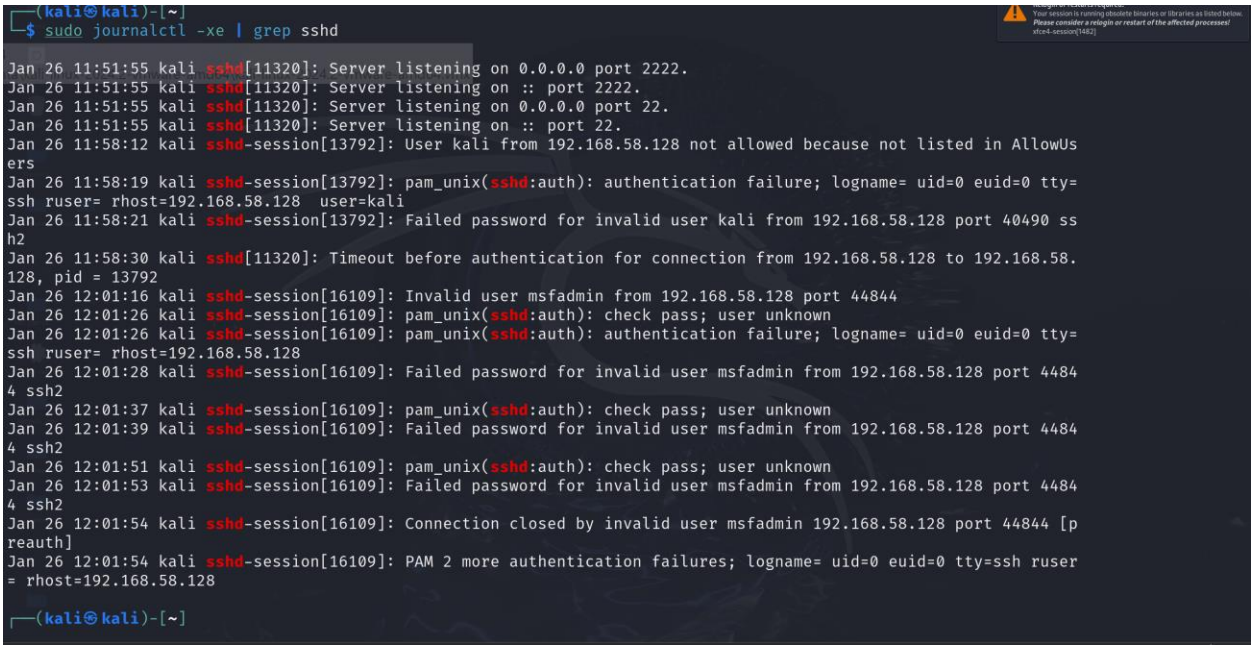
3. User Management

- Screenshot showing:
    - adduser kali (already exists)
    - passwd kali (password updated)

```
┌──(root㉿kali)-[/home/kali]
└─# cat /etc/passwd | grep kali


kali:x:1000:1000:,,,:/home/kali:/usr/bin/zsh
rescueuser:x:1002:1002:kali,kali,kali,kali,kali:/home/rescueuser:/bin/bash

┌──(root㉿kali)-[/home/kali]
└─# sudo adduser kali

fatal: The user `kali' already exists.

┌──(root㉿kali)-[/home/kali]
└─# sudo passwd kali

New password:
Retype new password:
passwd: password updated successfully
```

```
┌──(root㉿kali)-[/home/kali]
└─# su kali
┌──(kali㉿kali)-[~]
└─$ sudo journalctl | grep "Failed password"

[sudo] password for kali:
Sorry, try again.
[sudo] password for kali:
Jan 26 11:58:21 kali sshd-session[13792]: Failed password for invalid user kali from 192.168.58.128 port 40490 ss
h2
Jan 26 12:01:28 kali sshd-session[16109]: Failed password for invalid user msfadmin from 192.168.58.128 port 4484
4 ssh2
Jan 26 12:01:39 kali sshd-session[16109]: Failed password for invalid user msfadmin from 192.168.58.128 port 4484
4 ssh2
Jan 26 12:01:53 kali sshd-session[16109]: Failed password for invalid user msfadmin from 192.168.58.128 port 4484
4 ssh2
```

View SSH Logs



STEP 2: Log Parsing (Python)

You extract:

- IP address
- Timestamp
- Event type (login failed / success)

Solution:

Extract IP Address

```
┌──(kali㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:e7:70:55 brd ff:ff:ff:ff:ff:ff
    inet 192.168.58.128/24 brd 192.168.58.255 scope global dynamic noprefixroute eth0
       valid_lft 1636sec preferred_lft 1636sec
    inet6 fe80::9291:95e5:9bd2:3292/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: br-6d7c97741749: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:1d:33:33:7d brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-6d7c97741749
       valid_lft forever preferred_lft forever
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:aa:65:8a:23 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever

┌──(kali㉿kali)-[~]
└─$ python3
Python 3.11.9 (main, Apr 10 2024, 13:16:36) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import re
>>> def extract_ip(log):
...     return re.findall(r'\d+\.\d+\.\d+\.\d+', log)
...
>>> log = "Jan 10 10:23:45 kali sshd: Failed password for root from 192.168.58.128 port 22 ssh2"
>>> extract_ip(log)
['192.168.58.128']
>>>
```

Extract Timestamp

```
┌──(kali㉿kali)-[~]
└─$ python3
Python 3.11.9 (main, Apr 10 2024, 13:16:36) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import re
>>> r'^[A-Z][a-z]{2}\s+\d+\s+\d+:\d+:\d+'
'^[A-Z][a-z]{2}\\s+\\d+\\s+\\d+:\\d+:\\d+'
>>> def extract_timestamp(log):
...     match = re.search(r'^[A-Z][a-z]{2}\s+\d+\s+\d+:\d+:\d+', log)
...     return match.group() if match else None
...
>>> import re
>>>
>>> logs = [
...     "Jan 10 10:23:45 server sshd[123]: Accepted password for user",
...     "Feb 5 08:12:01 server sshd[456]: Failed password for root",
...     "Invalid log line without timestamp"
... ]
>>>
>>> for log in logs:
...     print(extract_timestamp(log))
...
Jan 10 10:23:45
Feb 5 08:12:01
None
>>> log = "Jan 10 10:23:45 server sshd[123]: Accepted password for user"
>>> extract_timestamp(log)
'Jan 10 10:23:45'
```

Extract Event Type

```
>>> def extract_event(log):
...     if "Failed password" in log:
...         return "FAILED_LOGIN"
...     elif "Accepted password" in log:
...         return "SUCCESS_LOGIN"
...     else:
...         return "OTHER"
...
>>> extract_event(log)
'SUCCESS_LOGIN'
```

Combine All in parse_ssh_log()

```
>>> import re
>>>
>>> def extract_ip(log):
...     match = re.search(r'[0-9]+(?:\.[0-9]+){3}', log)
...     return [match.group()] if match else None
...
>>> def extract_timestamp(log):
...     match = re.search(r'^[A-Z][a-z]{2}\s+\d+\s+\d+:\d+:\d+', log)
...     return match.group() if match else None
...
>>> def extract_event(log):
...     return log.split(":")[-1].strip() if ":" in log else None
...
>>> def parse_ssh_log(log):
...     return {
...         "timestamp": extract_timestamp(log),
...         "ip_address": extract_ip(log)[0] if extract_ip(log) else None,
...         "event": extract_event(log)
...     }
...
>>> log = "Jan 10 10:23:45 server sshd[123]: Accepted password for 192.168.1.100 port 22 ssh2"
>>> print(parse_ssh_log(log))
{'timestamp': 'Jan 10 10:23:45', 'ip_address': '192.168.1.100', 'event': 'Accepted password for 192.168.1.100 port 22 ssh2'}
```

```
>>> logs = [
...     "Jan 10 10:23:45 kali sshd: Failed password for root from 192.168.1.10 port 22 ssh2",
...     "Jan 10 10:25:01 kali sshd: Accepted password for user from 192.168.1.15 port 22 ssh2"
... ]
>>>
>>> parsed_logs = [parse_ssh_log(log) for log in logs]
>>> parsed_logs
[{'timestamp': 'Jan 10 10:23:45', 'ip_address': '192.168.1.10', 'event': 'Failed password for root from 192.168.1.10 port 22 ssh2'}
, {'timestamp': 'Jan 10 10:25:01', 'ip_address': '192.168.1.15', 'event': 'Accepted password for user from 192.168.1.15 port 22 ssh
2'}]
```

STEP 3: Threat Detection Rules

Detect:

- Brute-force attempts
- Suspicious IPs
- Repeated failures

Solution:

Sample Parsed Logs (Input)

```
┌──(kali㉿kali)-[~]
└─$ python3
Python 3.11.9 (main, Apr 10 2024, 13:16:36) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import re
>>> parsed_logs = [
...     {'timestamp': 'Jan 10 10:23:45', 'ip_address': '192.168.1.10', 'event': 'FAILED_LOGIN'},
...     {'timestamp': 'Jan 10 10:23:50', 'ip_address': '192.168.1.10', 'event': 'FAILED_LOGIN'},
...     {'timestamp': 'Jan 10 10:24:01', 'ip_address': '192.168.1.10', 'event': 'FAILED_LOGIN'},
...     {'timestamp': 'Jan 10 10:24:10', 'ip_address': '192.168.1.10', 'event': 'FAILED_LOGIN'},
...     {'timestamp': 'Jan 10 10:24:20', 'ip_address': '192.168.1.10', 'event': 'FAILED_LOGIN'},
...     {'timestamp': 'Jan 10 10:25:01', 'ip_address': '192.168.1.15', 'event': 'SUCCESS_LOGIN'}
... ]
```

Rule 1 — Detect Repeated Failed Logins

```
>>> from collections import Counter
>>> failed_ips = [
...     log['ip_address']
...     for log in parsed_logs
...     if log['event'] == 'FAILED_LOGIN'
... ]
>>>
>>> failed_count = Counter(failed_ips)
>>> failed_count
Counter({'192.168.1.10': 5})
```

Rule 2 — Detect Brute-Force Attack

```
>>> BRUTE_FORCE_THRESHOLD = 5
>>>
>>> brute_force_ips = [
...     ip for ip, count in failed_count.items()
...     if count ≥ BRUTE_FORCE_THRESHOLD
... ]
>>>
>>> brute_force_ips
['192.168.1.10']
```

Rule 3 — Flag Suspicious Ips

```
>>> alerts = []
>>>
>>> for ip in brute_force_ips:
...     alerts.append({
...         "ip_address": ip,
...         "threat": "Brute Force Attack",
...         "failed_attempts": failed_count[ip]
...     })
...
>>> alerts
[{'ip_address': '192.168.1.10', 'threat': 'Brute Force Attack', 'failed_attempts': 5}]
```

Suspicious Success After Failures

```
>>> for log in parsed_logs:
...     print(log['event'], log['ip_address'])  # Debug line
...     if log['event'] == 'SUCCESS_LOGIN' and log['ip_address'] in brute_force_ips:
...         print(f"⚠ ALERT: Successful login after brute-force from {log['ip_address']}")
...
FAILED_LOGIN 192.168.1.10
FAILED_LOGIN 192.168.1.10
FAILED_LOGIN 192.168.1.10
FAILED_LOGIN 192.168.1.10
FAILED_LOGIN 192.168.1.10
SUCCESS_LOGIN 192.168.1.15
>>> brute_force_ips = ['192.168.1.10']
>>>
>>> parsed_logs = [
...     {'timestamp': 'Jan 10 10:23:45', 'ip_address': '192.168.1.10', 'event': 'FAILED_LOGIN'},
...     {'timestamp': 'Jan 10 10:25:01', 'ip_address': '192.168.1.10', 'event': 'SUCCESS_LOGIN'}
... ]
>>>
>>> for log in parsed_logs:
...     if log['event'] == 'SUCCESS_LOGIN' and log['ip_address'] in brute_force_ips:
...         print(f"⚠ ALERT: Successful login after brute-force from {log['ip_address']}")
...
⚠ ALERT: Successful login after brute-force from 192.168.1.10
```