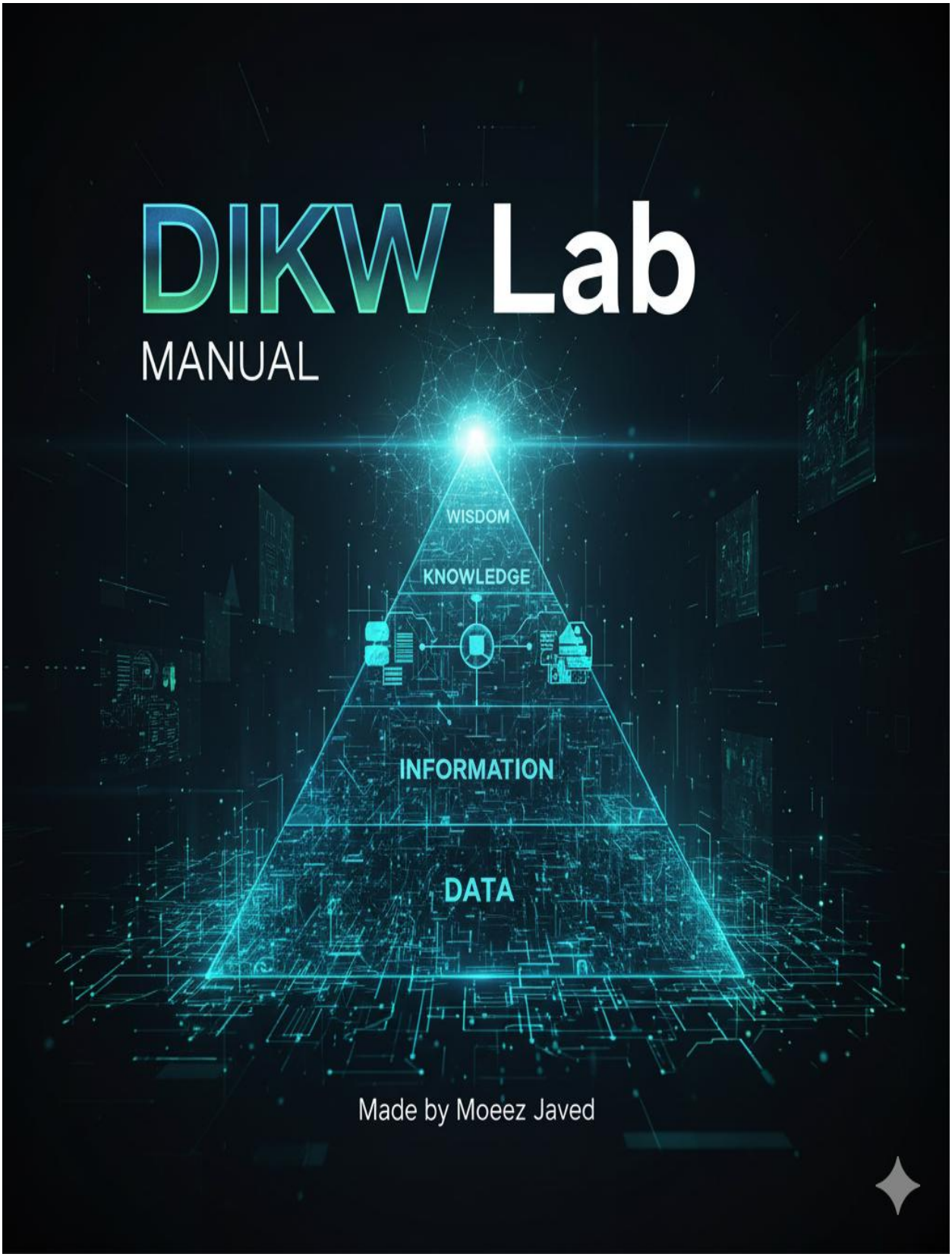
Made By Moeez Javed



Made By Moeez Javed

DIKW Lab Manual

Course: Applied Cybersecurity / SOC Analyst Labs
Duration: ~30–45 minutes (core walkthrough) + practice tasks
Goal: Teach students the DIKW (Data → Information → Knowledge → Wisdom) model using log analysis and a small sensor simulation.

Introduction — DIKW Model & Its Use

The **DIKW** model is a hierarchy used to transform raw observations into actionable decisions:

- **Data:** Raw, unprocessed facts (e.g., log lines, sensor readings).
- **Information:** Structured data that answers who/what/when/where (e.g., extracted IPs, timestamps, statuses).
- **Knowledge:** Patterns and meaning derived from information (e.g., repeated failed logins suggest brute-force behavior).
- **Wisdom:** Applying knowledge to make decisions that resolve problems (e.g., block attacker IPs, restrict log access).

In security operations, DIKW maps naturally to the workflow of a SOC analyst: collecting logs (Data), parsing and extracting fields (Information), identifying threats (Knowledge), and taking remediation or policy actions (Wisdom).

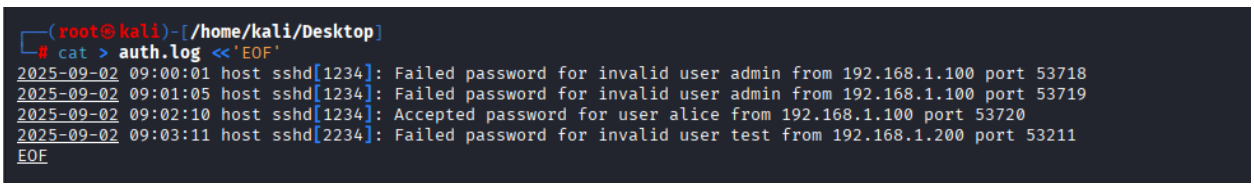
Walkthrough Task — DIKW Applied to auth.log

Step 1 — Data Collection (create a simulated auth.log)

Time: 2 minutes

Open a terminal and create a small auth.log using a here-document. Type the following exactly and press Enter after the final EOF line.

```
cat > auth.log <<'EOF'
2025-09-02 09:00:01 host sshd[1234]: Failed password for invalid user admin from 192.168.1.100 port 53718
2025-09-02 09:01:05 host sshd[1234]: Failed password for invalid user admin from 192.168.1.100 port 53719
2025-09-02 09:02:10 host sshd[1234]: Accepted password for user alice from 192.168.1.100 port 53720
2025-09-02 09:03:11 host sshd[2234]: Failed password for invalid user test from 192.168.1.200 port 532
11
EOF
```



This creates auth.log in your current directory containing four sample entries.

Made By Moeez Javed

Step 2 — Information Extraction with Python (7 minutes)

Write a Python script that parses the log and extracts timestamp, src_ip, and status (Accepted/Failed). Create walkthrough.py with the exact content below.

```
cat > walkthrough.py <<'PY'
#!/usr/bin/env python3
import re
from collections import defaultdict

logfile = 'auth.log'
pattern = re.compile(r"^(?P<timestamp>\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}).*?(Failed password|Accept
ed password).*from (?P<ip>\d+\.\d+\.\d+\.\d+).*(?:port (?P<port>\d+))")

results = []
with open(logfile, 'r') as f:
    for line in f:
        m = pattern.search(line)
        if m:
            status = 'Failed' if 'Failed password' in line else 'Accepted'
            results.append({'timestamp': m.group('timestamp'), 'ip': m.group('ip'), 'status': status})

# Print extracted information (Information layer)
print("Extracted entries:")
for r in results:
    print(f"{r['timestamp']}  {r['ip']}  {r['status']}")

# Aggregate per IP (helpful for knowledge layer)
counts = defaultdict(lambda: {'Failed': 0, 'Accepted': 0})
for r in results:
    counts[r['ip']][r['status']] += 1

print('\nSummary by IP:')
for ip, c in counts.items():
    print(f"{ip} -> Failed: {c['Failed']}, Accepted: {c['Accepted']}")

PY
```

Made By Moeez Javed

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# >....
from collections import defaultdict

logfile = 'auth.log'
pattern = re.compile(r"^(?P<timestamp>\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}).*?(Failed password|Accepted password).*from (?P<ip>\d+\.\d+\.\d+\.\d+).*(?:port (?P<port>\d
+))")

results = []
with open(logfile, 'r') as f:
    for line in f:
        m = pattern.search(line)
        if m:
            status = 'Failed' if 'Failed password' in line else 'Accepted'
            results.append({'timestamp': m.group('timestamp'), 'ip': m.group('ip'), 'status': status})

# Print extracted information (Information layer)
print("Extracted entries:")
for r in results:
    print(f"{r['timestamp']}  {r['ip']}  {r['status']}")

# Aggregate per IP (helpful for knowledge layer)
counts = defaultdict(lambda: {'Failed': 0, 'Accepted': 0})
for r in results:
    counts[r['ip']][r['status']] += 1

print('\nSummary by IP:')
for ip, c in counts.items():
    print(f"{ip} → Failed: {c['Failed']}, Accepted: {c['Accepted']}")
```

Make the script executable and run it:

chmod +x walkthrough.py
python3 walkthrough.py

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# chmod +x walkthrough.py
python3 walkthrough.py

Extracted entries:
2025-09-02 09:00:01  192.168.1.100  Failed
2025-09-02 09:01:05  192.168.1.100  Failed
2025-09-02 09:02:10  192.168.1.100  Accepted
2025-09-02 09:03:11  192.168.1.200  Failed

Summary by IP:
192.168.1.100 → Failed: 2, Accepted: 1
192.168.1.200 → Failed: 1, Accepted: 0
```

Expected output (Information layer) — you should see lines like:

Extracted entries:
2025-09-02 09:00:01 192.168.1.100 Failed
2025-09-02 09:01:05 192.168.1.100 Failed
2025-09-02 09:02:10 192.168.1.100 Accepted
2025-09-02 09:03:11 192.168.1.200 Failed

Summary by IP:
192.168.1.100 -> Failed: 2, Accepted: 1
192.168.1.200 -> Failed: 1, Accepted: 0

If you see this, you successfully transformed raw data into structured information.

Step 3 — Knowledge Application (5 minutes)

Look at the script output: 192.168.1.100 has two failures then one success within a short window. That pattern **may indicate a brute-force attempt** followed by a successful login.

To gather more context from the running system, check current user sessions and recent logins:

 # Show who is logged in now
who

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# who

kali     seat0        2025-09-23 23:40 (:0)
```

More verbose — show active sessions
w

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# w
 00:13:47 up 46 min,  1 user,  load average: 0.15, 0.70, 0.70
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
kali     -        -                23:40           0.00s  0.08s lightdm --session-child 13 24
```

Show recent logins (history)
last -a | head -n 10

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# last -a | head -n 10
root     pts/2        Tue Sep 23 23:48 - still logged in
kali     tty7         Tue Sep 23 23:40 - still logged in    :0
lightdm  tty7         Tue Sep 23 23:25 - 23:40  (00:15)     :0
lightdm  tty7         Tue Sep 23 06:54 - 06:54 (213503982+0 :0
kali     tty7         Tue Sep 23 06:54 - still logged in    :0
root     pts/1        Tue Sep 23 04:57 - still logged in
postgres pts/1        Wed Sep 10 01:57 - 01:57  (00:00)
postgres pts/1        Wed Sep 10 01:57 - 01:57  (00:00)
postgres pts/2        Wed Sep 10 01:56 - 01:56  (00:00)
postgres pts/2        Wed Sep 10 01:24 - 01:24  (00:00)
```

Use these commands to verify whether the Accepted login corresponds to an active session, its user, and which TTY.

Step 4 — Wisdom (Act to Remediate)

Having identified suspicious access, we now apply **wisdom**: implement defensive actions that secure confidentiality, integrity, and availability.

Confidentiality *— Restrict log access*

Set auth.log permissions so only the owner (admin/root) can read/write.

 # Change owner to root (if appropriate) and restrict perms
sudo chown root:root auth.log
sudo chmod 600 auth.log
ls -l auth.log

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# # Change owner to root (if appropriate) and restrict perms
sudo chown root:root auth.log
sudo chmod 600 auth.log
ls -l auth.log
-rw------- 1 root root 417 Sep 24 00:09 auth.log
```

This enforces that only root can read the file.

Made By Moeez Javed

Integrity — Hash the log for tamper detection

Create a SHA-256 hash of the file and store it for later comparison.

sha256sum auth.log > auth.log.sha256
cat auth.log.sha256

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# sha256sum auth.log > auth.log.sha256
cat auth.log.sha256
2c99265e9b369abf623bb9e02b1acc0629b8d9f2aff3ee47ac920b0ad77bf1ca  auth.log
```

Later you can verify with sha256sum -c auth.log.sha256.

Availability *— Block the offending IP (iptables)*

Block the IP address to prevent further attacks.

 # Add DROP rule for the offending IP (replace with the suspicious IP)
sudo iptables -A INPUT -s 192.168.1.100 -j DROP

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# sudo iptables -A INPUT -s 192.168.1.100 -j DROP
```

 # Save rules (Debian/Ubuntu)
sudo apt -y install iptables-persistent
sudo netfilter-persistent save

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# sudo apt -y install iptables-persistent
sudo netfilter-persistent save

iptables-persistent is already the newest version (1.0.23).
Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 113
run-parts: executing /usr/share/netfilter-persistent/plugins.d/15-ip4tables save
run-parts: executing /usr/share/netfilter-persistent/plugins.d/25-ip6tables save
```

View the iptables rules and confirm the block:

sudo iptables -L -n -v --line-numbers

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# sudo iptables -L -n -v --line-numbers
Chain INPUT (policy ACCEPT 1 packets, 76 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 DROP       all  --  *      *       192.168.1.100        0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 DOCKER-USER  all  --  *      *       0.0.0.0/0            0.0.0.0/0
2        0     0 DOCKER-ISOLATION-STAGE-1  all  --  *      *     0.0.0.0/0            0.0.0.0/0
3        0     0 ACCEPT     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0           ctstate RELATED,ESTABLISHED
4        0     0 DOCKER     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
5        0     0 ACCEPT     all  --  docker0 !docker0  0.0.0.0/0           0.0.0.0/0
6        0     0 ACCEPT     all  --  docker0 docker0  0.0.0.0/0            0.0.0.0/0

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination

Chain DOCKER (1 references)
num   pkts bytes target     prot opt in     out     source               destination

Chain DOCKER-ISOLATION-STAGE-1 (1 references)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 DOCKER-ISOLATION-STAGE-2  all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
2        0     0 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain DOCKER-ISOLATION-STAGE-2 (1 references)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 DROP       all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
2        0     0 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0
```

To remove the rule later (if needed), note the line number from the previous output and delete it:

```
# Example: delete rule number 3 from INPUT chain
sudo iptables -D INPUT 3
```

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# sudo iptables -D INPUT 3
iptables: Index of deletion too big.
```

Practice Tasks (for students)

Practice Task 1 — University Student Portal (DIKW exercise)

Scenario: The portal's auth.log contains:

Login attempt from 192.168.1.100 at 2025-09-02 09:00: failed
Login attempt from 192.168.1.100 at 2025-09-02 09:01: failed
Login attempt from 192.168.1.100 at 2025-09-02 09:03: success

Exercise: 1. Create a simulated auth.log with those entries (use Step 1 method above). 2. Run walkthrough.py to extract information. 3. Produce the knowledge summary and explain why this looks like a brute-force attempt. 4. Apply wisdom: - chmod 600 auth.log to protect confidentiality. - sha256sum auth.log > auth.log.sha256 to protect integrity. - Use iptables to block 192.168.1.100 to protect availability.

Include screenshots or terminal copy of each command and its output.

Practice Task 2 — Temperature Sensor Simulation

Goal: Write and run a Python program that: periodically generates simulated temperature readings, logs them to temp.log, and triggers an alert when temperature > 30°C.

Create temp_sensor.py:

```
cat > temp_sensor.py <<'PY'
#!/usr/bin/env python3
import time
import random
from datetime import datetime

LOGFILE = 'temp.log'
THRESHOLD = 30.0

with open(LOGFILE, 'a') as f:
    for i in range(20):
        temp = round(random.uniform(20.0, 35.0), 1)
        ts = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        line = f"{ts} Temperature: {temp} C\n"
        f.write(line)
        f.flush()
        print(line.strip())
        if temp > THRESHOLD:
            alert = f"{ts} ALERT: temperature {temp}C exceeds threshold {THRESHOLD}C\n"
            f.write(alert)
```

```
        f.flush()
        print(alert.strip())
    time.sleep(1)
PY
```

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# cat > temp_sensor.py <<'PY'
#!/usr/bin/env python3
import time
import random
from datetime import datetime

LOGFILE = 'temp.log'
THRESHOLD = 30.0

with open(LOGFILE, 'a') as f:
    for i in range(20):
        temp = round(random.uniform(20.0, 35.0), 1)
        ts = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        line = f"{ts} Temperature: {temp} C\n"
        f.write(line)
        f.flush()
        print(line.strip())
        if temp > THRESHOLD:
            alert = f"{ts} ALERT: temperature {temp}C exceeds threshold {THRESHOLD}C\n"
            f.write(alert)
            f.flush()
            print(alert.strip())
        time.sleep(1)
PY
```

Make executable and run:

```
chmod +x temp_sensor.py
python3 temp_sensor.py
```

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# chmod +x temp_sensor.py
python3 temp_sensor.py

2025-09-24 00:16:47 Temperature: 28.2 C
2025-09-24 00:16:48 Temperature: 23.3 C
2025-09-24 00:16:49 Temperature: 32.0 C
2025-09-24 00:16:49 ALERT: temperature 32.0C exceeds threshold 30.0C
2025-09-24 00:16:50 Temperature: 21.5 C
2025-09-24 00:16:51 Temperature: 21.6 C
2025-09-24 00:16:52 Temperature: 20.5 C
2025-09-24 00:16:53 Temperature: 23.3 C
2025-09-24 00:16:54 Temperature: 33.8 C
2025-09-24 00:16:54 ALERT: temperature 33.8C exceeds threshold 30.0C
2025-09-24 00:16:55 Temperature: 28.5 C
2025-09-24 00:16:56 Temperature: 23.0 C
2025-09-24 00:16:57 Temperature: 30.1 C
2025-09-24 00:16:57 ALERT: temperature 30.1C exceeds threshold 30.0C
2025-09-24 00:16:58 Temperature: 22.5 C
2025-09-24 00:16:59 Temperature: 26.9 C
2025-09-24 00:17:00 Temperature: 27.5 C
2025-09-24 00:17:01 Temperature: 30.4 C
2025-09-24 00:17:01 ALERT: temperature 30.4C exceeds threshold 30.0C
2025-09-24 00:17:02 Temperature: 22.5 C
2025-09-24 00:17:03 Temperature: 20.3 C
2025-09-24 00:17:04 Temperature: 28.2 C
2025-09-24 00:17:05 Temperature: 24.4 C
2025-09-24 00:17:06 Temperature: 28.2 C
```

What to submit: - temp.log showing all readings and any ALERT lines. - A short note recommending an action (e.g., enable cooling, send notification) when threshold crossed.

Helpful Commands Summary

- Create file with here-doc: cat > filename <<'EOF' then paste content then EOF
- Run Python script: python3 script.py
- Check active sessions: who, w, last
- Hash files: sha256sum file > file.sha256

- Change permissions: chmod 600 file
- Block IP (iptables): sudo iptables -A INPUT -s 1.2.3.4 -j DROP
- View iptables: sudo iptables -L -n -v --line-numbers

Grading Rubric (suggested)

- **Data creation & scripts provided** (30%) — correct auth.log, walkthrough.py, temp_sensor.py files.
- **Output correctness** (30%) — parsed output, temp alerts in temp.log.
- **DIKW reasoning** (20%) — explanation identifying brute-force behavior and correct application of confidentiality/integrity/availability controls.
- **Presentation** (20%) — screenshots, organized report, commands listed.

Instructor Notes & Tips

- If iptables-persistent refuses to install, instruct students to manually save rules or use sudo iptables-save > /etc/iptables/rules.v4 on Debian/Ubuntu.
- If students lack root, use ufw (if available) or ask instructor for elevated privileges.
- Encourage students to timestamp screenshots and include command history.

End of DIKW Walkthrough Lab Manual