Task 1: Prevention Control Implementation

1. Description of the Preventive Control

The control implemented is a **Host-Based Firewall (UFW)** configuration on a Kali Linux system. This is a **Preventive control** designed to reduce the attack surface of the host. By enforcing a "Default Deny" posture, the system prevents unauthorized network connections and ensures that only pre-approved traffic (on Port 443) can reach the system services.

2. Tools & Configurations Used

- **Operating System:** Kali Linux
- **Tool:** ufw (Uncomplicated Firewall)
- **Configuration:**
    - **Default Policy:** Deny all incoming traffic.
    - **Permit Rule:** Allow TCP traffic on Port 443 (HTTPS).
    - **State:** Enabled on system startup.

3. Step-by-Step Implementation Process

1. **Set Global Policy:** Executed sudo ufw default deny incoming to ensure any connection attempt not explicitly permitted is blocked by default.
2. **Define Exceptions:** Executed sudo ufw allow 443/tcp to allow secure web traffic, following the principle of Least Privilege.
3. **Activation:** Executed sudo ufw enable to start the firewall service and commit the rules to the Linux kernel (iptables).
4. **Verification:** Executed ufw status to confirm the firewall is active and that only Port 443 is exposed to the network.

4. Evidence of Implementation

The attached screenshot (preventive-new-ufw.png) documents the terminal session where:

- The default policy was changed to 'deny'.
- The firewall was successfully activated.

- The status command verifies that traffic is **ALLOWED** for **443/tcp** from **Anywhere**, while all other traffic is implicitly denied.

5. Explanation of Prevention

This control prevents security incidents by **eliminating unauthorized entry points**. Without this control, high-risk services like SSH (22), Telnet (23), or SMB (445) might be left open to the network. By blocking these ports, we prevent attackers from:

- **Scanning/Enumerating** system services.
- **Brute-forcing** login credentials.
- **Exploiting unpatched vulnerabilities** in background services.

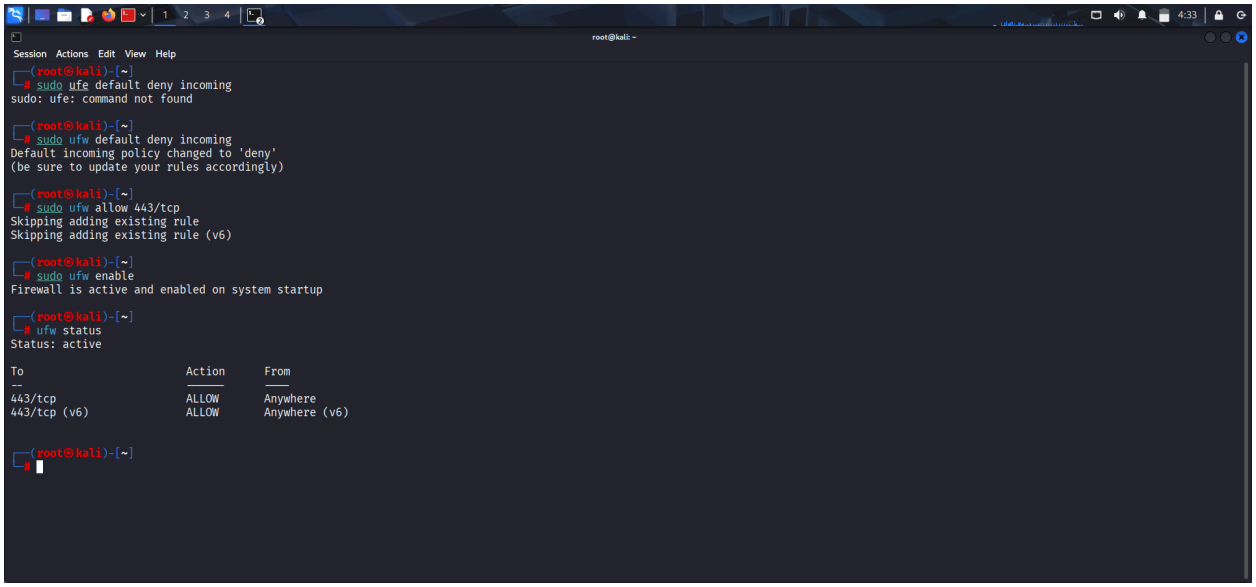

Figure 1. Firewall Configuration

Additional Preventive Control: File Permission Management

1. Description

This is a **Technical / Preventive Control** enforcing the **Principle of Least Privilege (PoLP)** through filesystem access controls.

2. Implementation Evidence

- **Task:** Restrict target_file to -rwxrw---- (Octal **760**).
- **Outcome:** The command sudo chmod 760 target_file was successfully executed.
- **Observation:** When a standard ls was attempted, the system returned "Permission denied," proving the control effectively blocked unauthorized access.
- **Verification:** Using sudo ls -l confirmed the permissions were correctly set to **rwx (Owner)** and **rw- (Group)**, with zero access for Others.

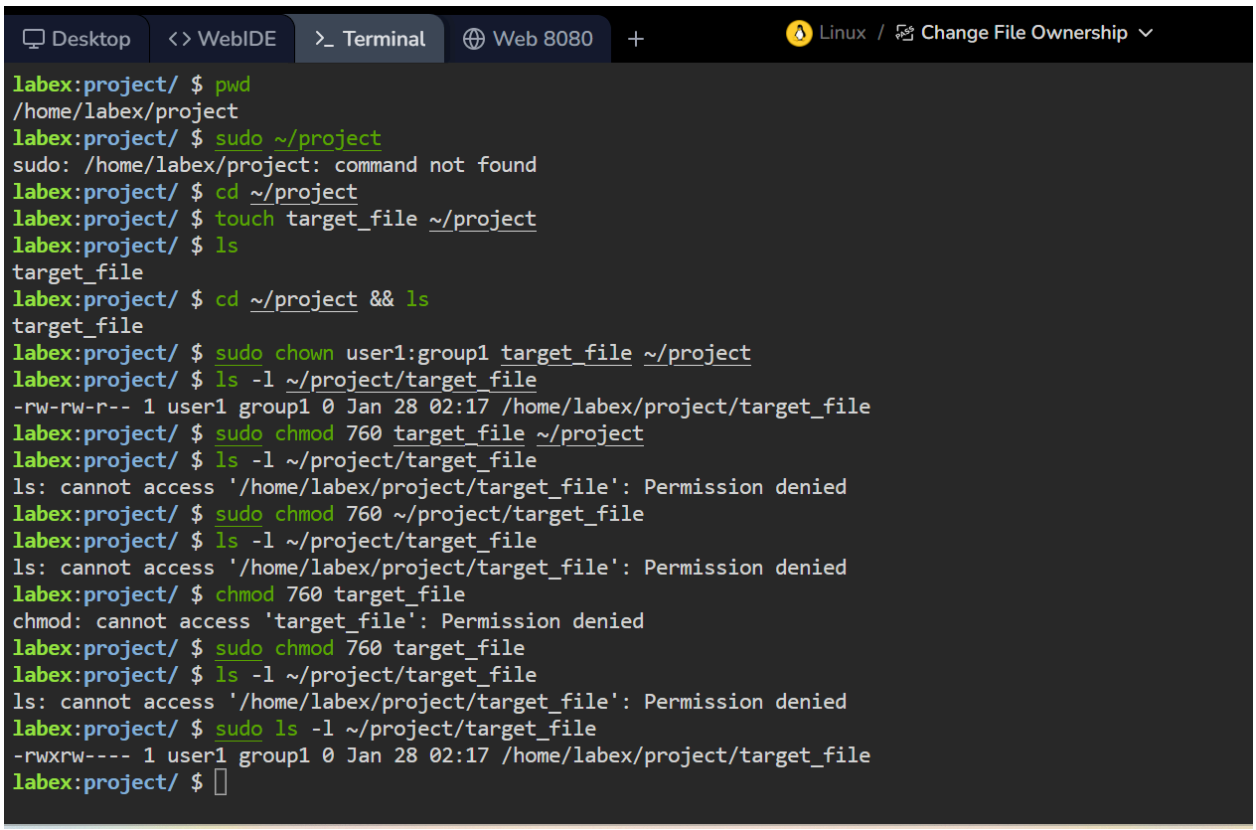

Figure 2. File Permission Management

Task 2: Detective Control Implementation (Fail2Ban)

1. Description

The control implemented is **Log-Based Intrusion Detection** using **Fail2Ban**. This is a **Detective control** designed to monitor system logs for signs of suspicious activity, specifically brute-force attacks against the SSH service.

2. Implementation & Configuration

- **Configuration File:** The /etc/fail2ban/jail.local file was configured to define the detection parameters for the sshd jail.
- **Parameters:**
    - enabled = true: Activated the monitoring for SSH.
    - maxretry = 3: Set the threshold to 3 failed attempts.
    - findtime = 600: Monitoring window of 10 minutes.
    - bantime = 3600: A 1-hour ban for detected attackers.
- **Backend:** Configured to use systemd to monitor the system journal for authentication failures.

3. Simulation & Verification (Attack verf.png & ssh c2.png)

- **The Simulation:** A brute-force attack was simulated by attempting to SSH into the host (192.168.0.4) from a secondary terminal using the invalid user attacker_test.
- **Detection Evidence:** * The journalctl -u ssh logs confirm the capture of multiple "Failed password" and "Invalid user" events originating from the local IP.
    - The system recorded: Connection closed by invalid user attacker_test 192.168.0.4.

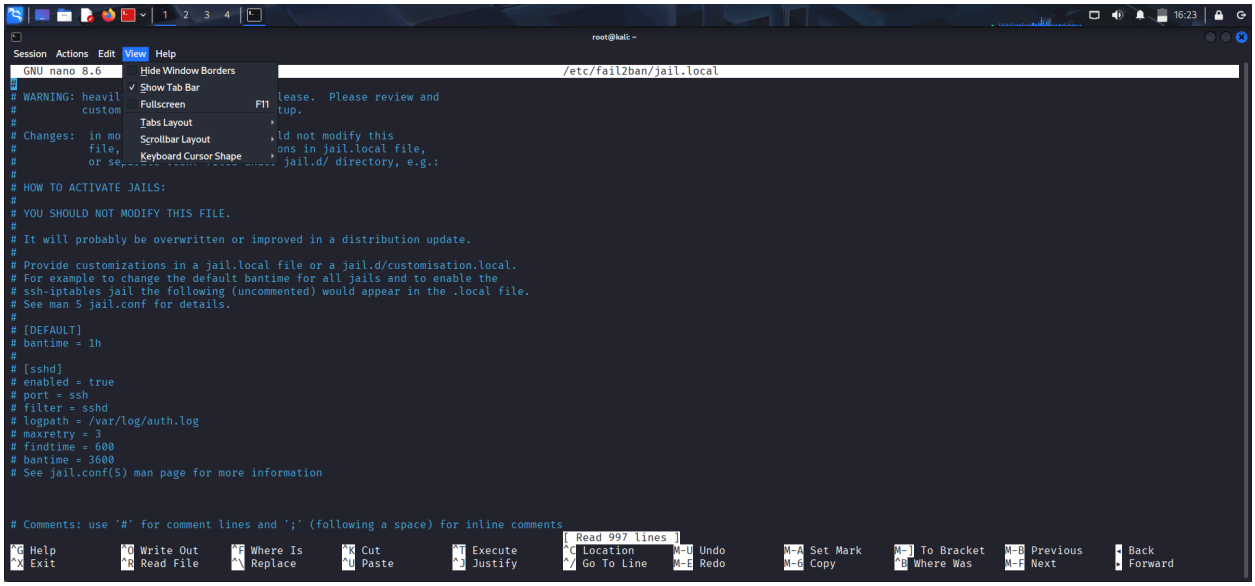

Fig 3. Setting login Parameters

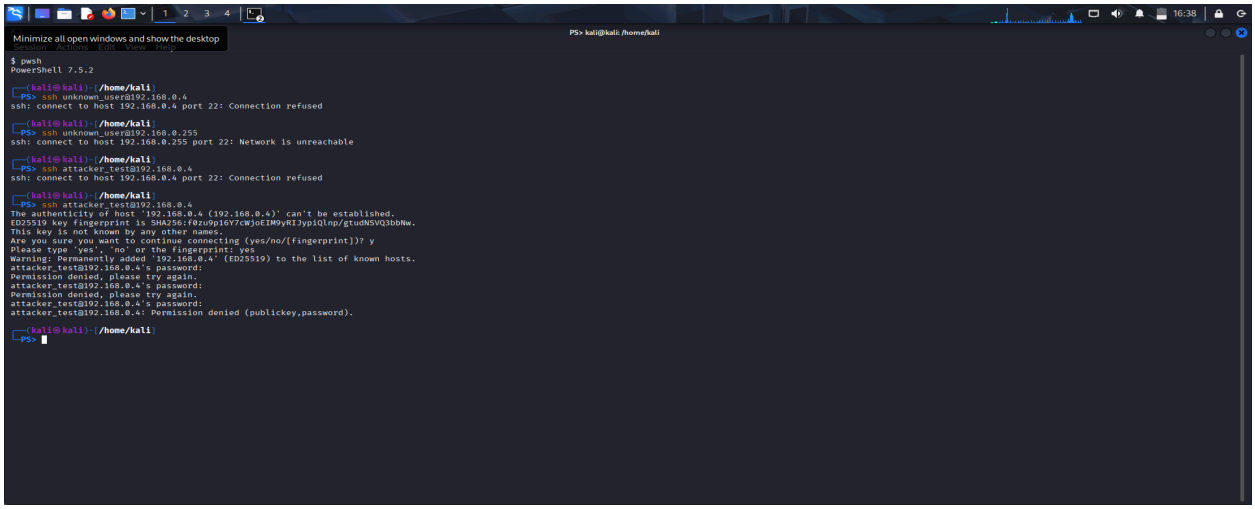

Figure 4. Ssh Attack

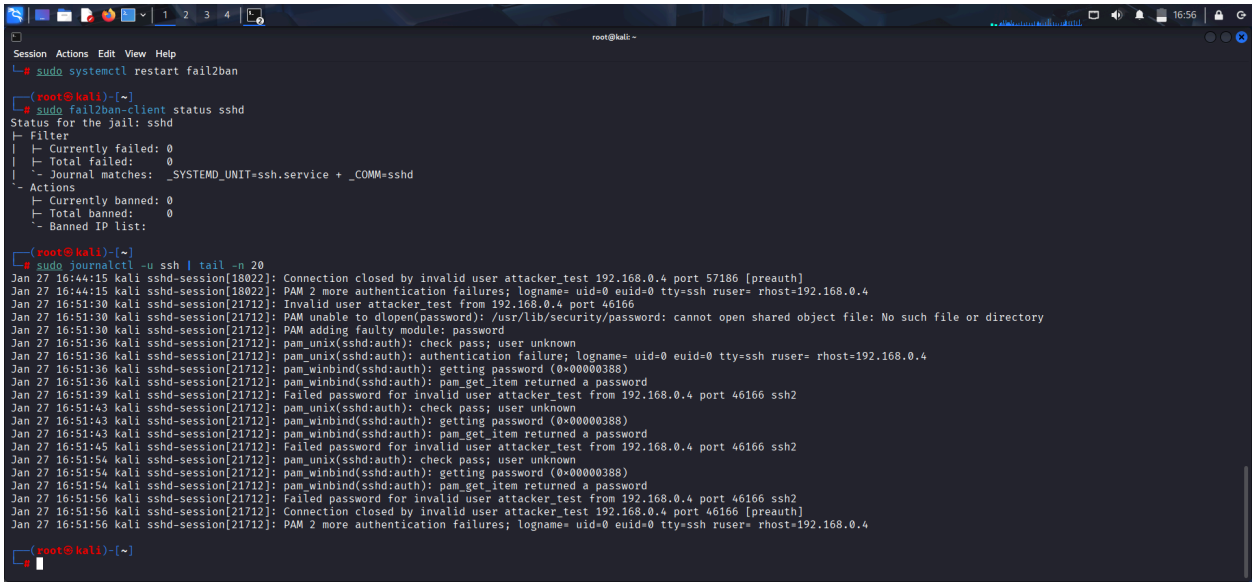

Figure 5. Attack Verification

Task 3: Directive Control Implementation

1. Description of the Directive Control

The control implemented is a **Mandatory Password Complexity Policy**. This is a directive control because it establishes the "rules of engagement" for users, mandating specific behaviors (choosing strong passwords) to maintain system security.

2. Policies & Configurations Used

- **Configuration File:** /etc/pam.d/common-password
- **Module:** pam_pwquality.so
- **Enforced Parameters:**
  - minlen=12: Minimum of 12 characters.
  - ucredit=-1, lcredit=-1, dcredit=-1, ocredit=-1: Mandatory use of uppercase, lowercase, digits, and symbols.
  - enforce_for_root: Ensures even administrative accounts must follow the directive.

3. Step-by-Step Implementation Process

1. **File Access:** Opened the PAM configuration file using the Nano text editor with root privileges.
2. **Directive Injection:** Modified the pam_pwquality.so line to include strict complexity requirements.
3. **Policy Enforcement:** Saved the configuration to immediately apply the directive to all future password changes.

4. Evidence of Implementation

As shown in **directive-password.png**, the configuration file has been updated to include the specific string of requirements. This visual evidence confirms that the system will now reject any password that does not meet the 12-character minimum or lacks the required character diversity.

5. Explanation of User Direction

This control directs user behavior by providing immediate feedback during password creation. If a user attempts to use a weak password, the system will refuse the change and "direct" them to choose a more complex one. This significantly reduces the risk of **credential cracking** and **brute-force attacks**.

Figure 6. Setting password Parameters

Task 4: Corrective Control Implementation

1. Description of the Corrective Control

This control is a **Corrective control** designed to restore system integrity after a security event has been detected. It utilizes an automated shell script to identify, locate, and permanently remove unauthorized "backdoor" files or scripts. Unlike preventive controls that block access, this control "corrects" the situation by cleaning up the system once a compromise is identified.

2. Tools & Remediation Techniques Used

- **Operating System:** Kali Linux
- **Language:** Bash Scripting
- **Commands:** `rm -v` (Verbose removal), `if-then` logical operators, and `ls` for verification.
- **Target:** Unauthorized script located at `/home/kali/backdoor.sh`.

3. Step-by-Step Implementation Process

1. **Threat Simulation:** Created a dummy malicious file using `touch` and populated it with a string representing malicious code.
2. **Script Development:** Authored a bash script named `cleanup.sh` containing logic to check for the file's existence.
3. **Permissions:** Applied execution rights to the script using `chmod +x cleanup.sh` to allow it to run as a system tool.
4. **Execution:** Ran the script with root privileges (`sudo`). The script successfully detected the target, executed the removal command, and provided status output.
5. **Verification:** Attempted to list the file using `ls -l` to confirm it was no longer present on the filesystem.

4. Evidence of Implementation

- **Script Logic (Clean-up.png):** The screenshot shows the source code of the corrective control, highlighting the automated detection logic and the remediation command (`rm -v "$THREAT"`).

- **Remediation Action (corrective.png):** The terminal output records the script being triggered. It explicitly states: *"Threat detected! Initiating corrective action..."* followed by confirmation that the backdoor script was removed.
- **Final Verification (corrective.png):** The final command `ls -l backdoor.sh` results in a *"No such file or directory"* error, proving the system has been successfully restored to its "known good" state.

5. Explanation of System Restoration

This control restores the system by **eliminating the persistence** of a threat. In a real-world scenario, if an attacker managed to bypass a firewall (Prevention) and was caught by an alert (Detective), this Corrective control would be the final stage of the incident response. It ensures that even if a breach occurs, the unauthorized modifications are purged, preventing the attacker from maintaining a foothold in the environment.

Figure 7. Identify and Remove Backdoor scripts

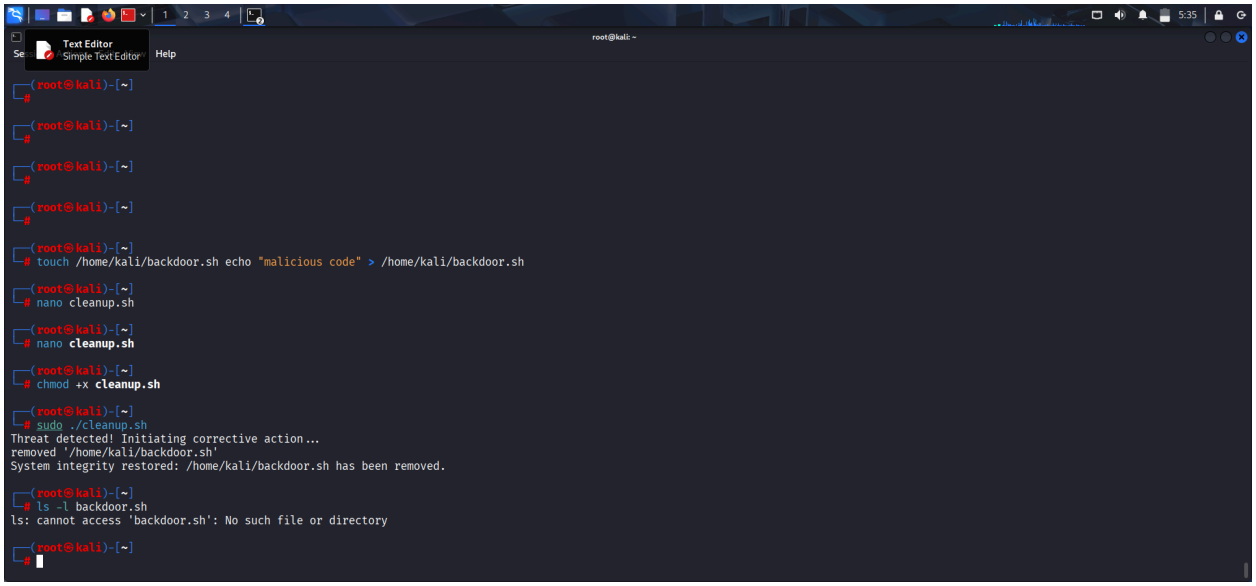

Figure 8. Cleaning up Threat

Task 5: Active Footprinting of the Local Network

1. Description of the Footprinting Process

Active footprinting is a **Detective / Reconnaissance** activity used to map a network and identify potential attack vectors. By interacting directly with systems, a Security Analyst can determine which assets are live, what ports are listening, and what services are exposed.

2. Tools & Methods Used

- **Operating System:** Kali Linux
- **Primary Tool: Nmap** (Network Mapper)
- **Techniques Applied:**
    - **Host Discovery (Ping Sweep):** Used to identify all active IP addresses in the subnet.
    - **Service Enumeration (-sV):** Probed ports to determine specific service versions.
    - **OS Detection (-O):** Analyzed TCP/IP stack fingerprints to identify target operating systems.

3. Step-by-Step Implementation & Results

A. Identification of Live Hosts

I performed a network-wide scan of the local subnet to identify active systems.

- **Command:** `sudo nmap -sn 192.168.1.0/24`
- **Findings:** As shown in **nmap.png**, the scan discovered **16 live hosts** within the subnet, including devices from Samsung, Apple, and Intel, as well as the local Kali instance (`192.168.1.200`).

B. Identification of Open Ports & Running Services

I conducted a deep scan on a specific target discovered during host discovery (`192.168.1.138`).

- **Command:** `sudo nmap -sS -sV -O 192.168.1.138`
- **Findings:** The scan targeted a **Redmi-Note-13**. The results in **nmap.png** show that while the host is live, all 1000 scanned ports returned a `closed` state.

4. Summary of Findings

Host IP	Device Type	Status	Security Posture
192.168.1.138	Redmi-Note -13	Live	**High Hardening:** All common ports closed/stealthed.
192.168.1.200	Kali Linux	Live	Localhost (Scanning Source).

5. Explanation of Security Value

Footprinting allows an analyst to perform **Asset Inventory** and **Vulnerability Assessment**. Identifying closed ports on the target (192.168.1.138) indicates a strong security posture where

no unnecessary services are exposed. For the analyst, this confirms that the target has a minimal attack surface.

Figure 9. Scanning the network

Final Conclusion

The combination of these controls creates a resilient security posture. **Prevention** (Firewall/Permissions) stops the majority of attacks; **Directive** (Passwords) ensures user-level security; **Detection** (Fail2Ban) alerts us to bypasses; and **Correction** (Cleanup scripts) handles the aftermath of a breach.