

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:

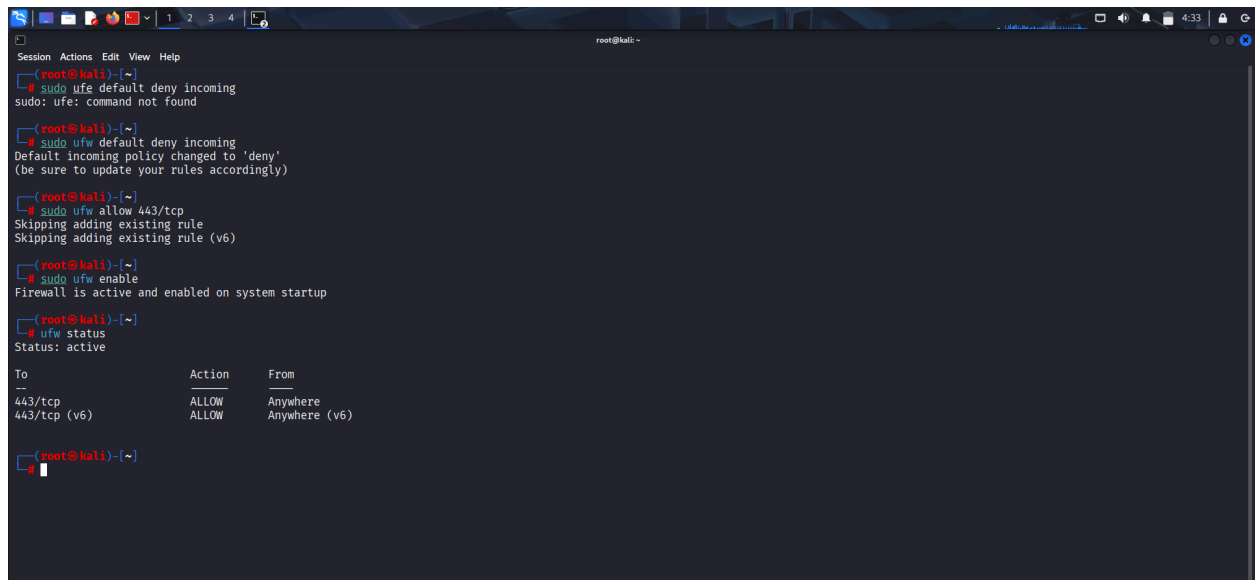
- The default policy was changed to 'deny'.
- The firewall was successfully activated.

- 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.



```

root@kali: ~
# sudo ufw default deny incoming
sudo: ufw: command not found

root@kali: ~
# sudo ufw default deny incoming
Default incoming policy changed to 'deny'
(be sure to update your rules accordingly)

root@kali: ~
# sudo ufw allow 443/tcp
Skipping adding existing rule
Skipping adding existing rule (v6)

root@kali: ~
# sudo ufw enable
Firewall is active and enabled on system startup

root@kali: ~
# ufw status
Status: active

To               Action       From
--               -
443/tcp          ALLOW        Anywhere
443/tcp (v6)     ALLOW        Anywhere (v6)

```

Figure 1. Firewall Configuration

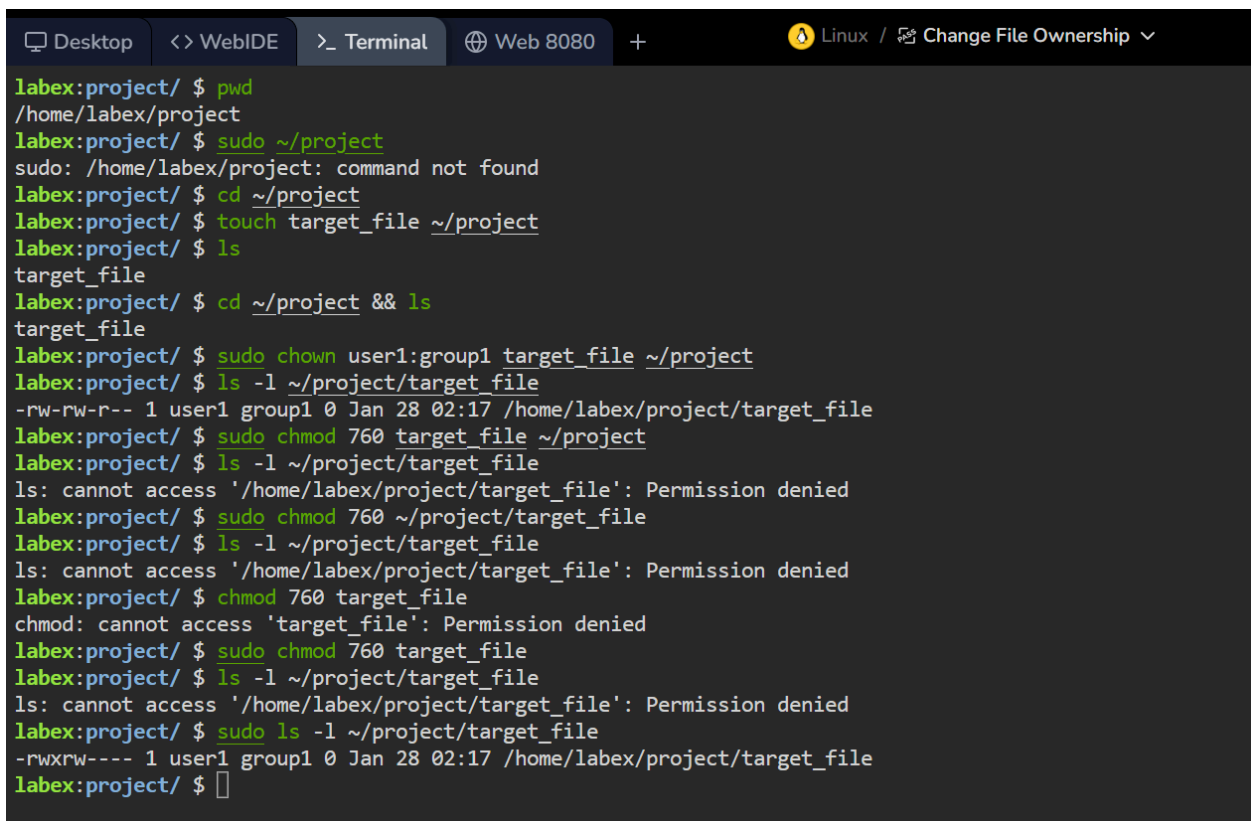
Additional Preventive Control: File Permission Management

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.

A screenshot of a terminal window with a dark background. The terminal shows a series of commands and their outputs. The user is in the directory /home/labex/project. They create a file named target_file, then use 'ls' to see its permissions (-rw-rw-r--). They then use 'sudo chown user1:group1 target_file' to change ownership. Next, they use 'ls -l' to see the permissions (-rw-rw-r--). They then use 'sudo chmod 760 target_file' to set permissions to -rwxrw----. Finally, they use 'ls -l' to verify the permissions, which are now -rwxrw----. The terminal output is as follows:

```
labex:project/ $ pwd
/home/labex/project
labex:project/ $ sudo ~/project
sudo: /home/labex/project: command not found
labex:project/ $ cd ~/project
labex:project/ $ touch target_file ~/project
labex:project/ $ ls
target_file
labex:project/ $ cd ~/project && ls
target_file
labex:project/ $ sudo chown user1:group1 target_file ~/project
labex:project/ $ ls -l ~/project/target_file
-rw-rw-r-- 1 user1 group1 0 Jan 28 02:17 /home/labex/project/target_file
labex:project/ $ sudo chmod 760 target_file ~/project
labex:project/ $ ls -l ~/project/target_file
ls: cannot access '/home/labex/project/target_file': Permission denied
labex:project/ $ sudo chmod 760 ~/project/target_file
labex:project/ $ ls -l ~/project/target_file
ls: cannot access '/home/labex/project/target_file': Permission denied
labex:project/ $ chmod 760 target_file
chmod: cannot access 'target_file': Permission denied
labex:project/ $ sudo chmod 760 target_file
labex:project/ $ ls -l ~/project/target_file
ls: cannot access '/home/labex/project/target_file': Permission denied
labex:project/ $ sudo ls -l ~/project/target_file
-rwxrw---- 1 user1 group1 0 Jan 28 02:17 /home/labex/project/target_file
labex:project/ $
```

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`.


```
root@kali: ~  
# sudo systemctl restart fail2ban  
[root@kali] ~  
# sudo fail2ban-client status sshd  
Status for the jail: sshd  
- Filter  
- Currently failed: 0  
- Total failed: 0  
- Journal matches: _SYSTEMD_UNIT=ssh.service + _COMM=sshd  
- Actions  
- Currently banned: 0  
- Total banned: 0  
- Banned IP list:  
[root@kali] ~  
# sudo journalctl -u ssh | tail -n 20  
Jan 27 16:44:15 kali sshd-session[18022]: Connection closed by invalid user attacker_test 192.168.0.4 port 57186 [preauth]  
Jan 27 16:44:15 kali sshd-session[18022]: PAM 2 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=192.168.0.4  
Jan 27 16:51:30 kali sshd-session[21712]: Invalid user attacker_test from 192.168.0.4 port 46166  
Jan 27 16:51:30 kali sshd-session[21712]: PAM unable to dlopen(password): /usr/lib/security/password: cannot open shared object file: No such file or directory  
Jan 27 16:51:30 kali sshd-session[21712]: PAM adding faulty module: password  
Jan 27 16:51:36 kali sshd-session[21712]: pam_unix(sshd:auth): check pass; user unknown  
Jan 27 16:51:36 kali sshd-session[21712]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=192.168.0.4  
Jan 27 16:51:36 kali sshd-session[21712]: pam_winbind(sshd:auth): getting password (0=00000388)  
Jan 27 16:51:36 kali sshd-session[21712]: pam_winbind(sshd:auth): pam_get_item returned a password  
Jan 27 16:51:39 kali sshd-session[21712]: Failed password for invalid user attacker_test from 192.168.0.4 port 46166 ssh2  
Jan 27 16:51:43 kali sshd-session[21712]: pam_unix(sshd:auth): check pass; user unknown  
Jan 27 16:51:43 kali sshd-session[21712]: pam_winbind(sshd:auth): getting password (0=00000388)  
Jan 27 16:51:43 kali sshd-session[21712]: pam_winbind(sshd:auth): pam_get_item returned a password  
Jan 27 16:51:45 kali sshd-session[21712]: Failed password for invalid user attacker_test from 192.168.0.4 port 46166 ssh2  
Jan 27 16:51:54 kali sshd-session[21712]: pam_unix(sshd:auth): check pass; user unknown  
Jan 27 16:51:54 kali sshd-session[21712]: pam_winbind(sshd:auth): getting password (0=00000388)  
Jan 27 16:51:54 kali sshd-session[21712]: pam_winbind(sshd:auth): pam_get_item returned a password  
Jan 27 16:51:56 kali sshd-session[21712]: Failed password for invalid user attacker_test from 192.168.0.4 port 46166 ssh2  
Jan 27 16:51:56 kali sshd-session[21712]: Connection closed by invalid user attacker_test 192.168.0.4 port 46166 [preauth]  
Jan 27 16:51:56 kali sshd-session[21712]: PAM 2 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=192.168.0.4  
[root@kali] ~  
#
```

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

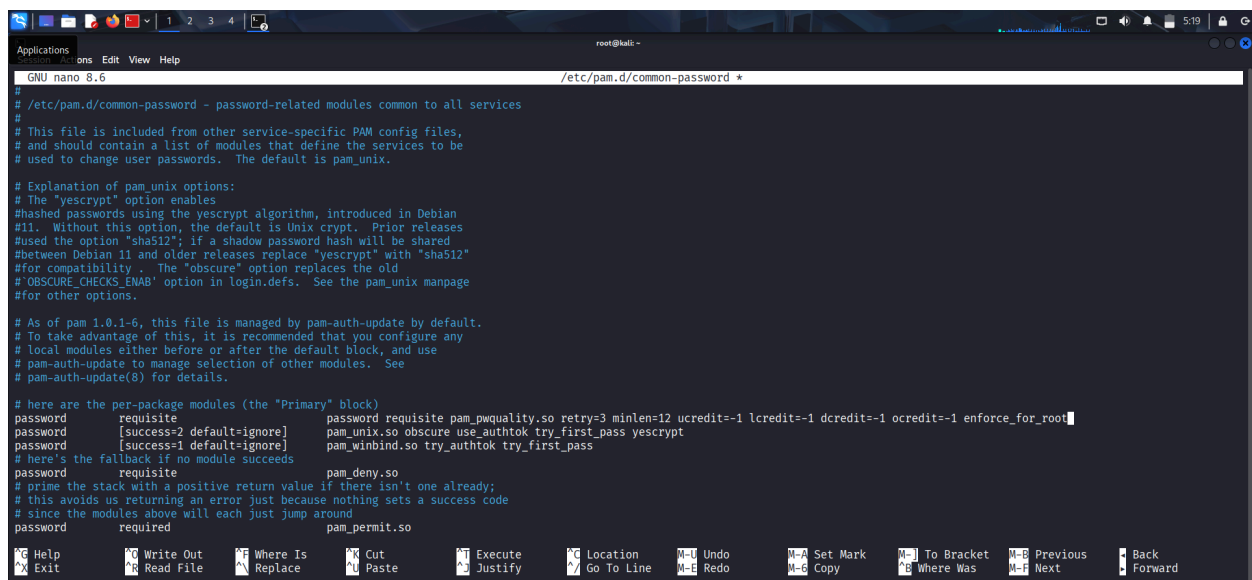
1. **File Access:** Opened the PAM configuration file using the Nano text editor with root privileges.
2. **Directive Injection:** Modified the `pam_pwquality.so` line to include strict complexity requirements.
3. **Policy Enforcement:** Saved the configuration to immediately apply the directive to all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



```
GNU nano 8.6 /etc/pam.d/common-password
#
# /etc/pam.d/common-password - password-related modules common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of modules that define the services to be
# used to change user passwords. The default is pam_unix.

# Explanation of pam_unix options:
# The "yescrypt" option enables
# hashed passwords using the yescrypt algorithm, introduced in Debian
# 11. Without this option, the default is Unix crypt. Prior releases
# used the option "sha512"; if a shadow password hash will be shared
# between Debian 11 and older releases replace "yescrypt" with "sha512"
# for compatibility. The "obscure" option replaces the old
# OBSOLETE_CHECKS_BNAB option in login.defs. See the pam_unix manpage
# for other options.

# As of pam 1.0.1-6, this file is managed by pam-auth-update by default.
# To take advantage of this, it is recommended that you configure any
# local modules either before or after the default block, and use
# pam-auth-update to manage selection of other modules. See
# pam-auth-update(8) for details.

# here are the per-package modules (the "Primary" block)
password      requisite      password      requisite      pam_pwquality.so  retry=3 minlen=12 ucredit=-1 lcredit=-1 dcredit=-1 enforce_for_root
password      [success=2 default=ignore]  pam_unix.so  obscure use_authtok try_first_pass yescrypt
password      [success=1 default=ignore]  pam_unix.so  obscure use_authtok try_first_pass yescrypt
# here's the fallback if no module succeeds
password      requisite      pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
password      required      pam_permit.so
```

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.

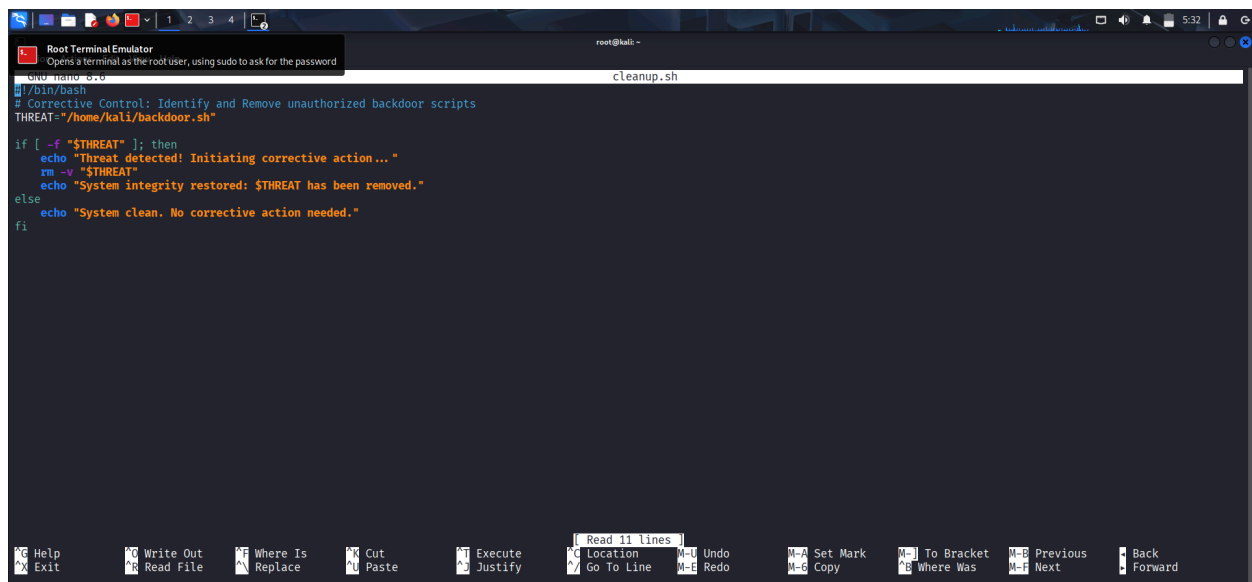
4. Evidence of Implementation

- **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.



```

root@kali: ~
└─$ cat cleanup.sh
#!/bin/bash
# Corrective Control: Identify and Remove unauthorized backdoor scripts
THREAT="/home/kali/backdoor.sh"

if [ -f "$THREAT" ]; then
    echo "Threat detected! Initiating corrective action..."
    rm -v "$THREAT"
    echo "System integrity restored: $THREAT has been removed."
else
    echo "System clean. No corrective action needed."
fi

```

Figure 7. Identify and Remove Backdoor scripts

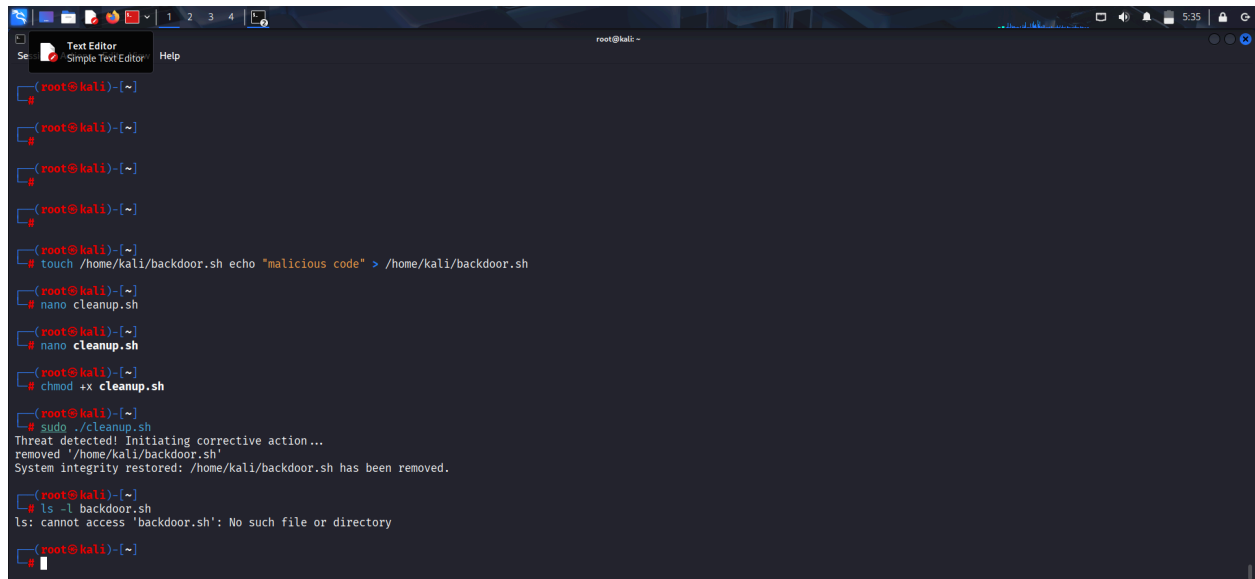
A screenshot of a Kali Linux terminal window. The terminal shows a series of commands and their outputs. The user is in the root directory of a Kali machine. They first create a file named 'backdoor.sh' in the /home/kali directory using the 'touch' command. Then, they use 'nano' to edit 'cleanup.sh'. After saving the file, they use 'chmod +x cleanup.sh' to make it executable. Finally, they run 'sudo ./cleanup.sh'. The output of this command shows a 'Threat detected! Initiating corrective action...' message, followed by the removal of '/home/kali/backdoor.sh' and a confirmation that 'System integrity restored: /home/kali/backdoor.sh has been removed.' The user then runs 'ls -l backdoor.sh', which returns an error: 'ls: cannot access 'backdoor.sh': No such file or directory'.

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.



```
root@kali: ~  
Session Actions Edit View Help  
Host is up (0.33s latency).  
MAC Address: EC:9B:F3:66:04:C7 (Samsung Electro-mechanics(Thailand))  
Nmap scan report for Revision-PC (192.168.1.173)  
Host is up (0.11s latency).  
MAC Address: 48:D7:05:E2:79:0D (Apple)  
Nmap scan report for DESKTOP-NDIHA41 (192.168.1.177)  
Host is up (0.10s latency).  
MAC Address: 84:3A:4B:AB:A8:CC (Intel Corporate)  
Nmap scan report for DESKTOP-PNF6LS7 (192.168.1.181)  
Host is up (0.34s latency).  
MAC Address: 6C:88:14:09:63:D4 (Intel Corporate)  
Nmap scan report for DESKTOP-BBR7N7V (192.168.1.182)  
Host is up (0.10s latency).  
MAC Address: 60:67:20:DE:91:C8 (Intel Corporate)  
Nmap scan report for Dear-Mike (192.168.1.192)  
Host is up (0.0012s latency).  
MAC Address: 38:8A:F8:22:62:E9 (Intel Corporate)  
Nmap scan report for Kali (192.168.1.200)  
Host is up.  
Nmap done: 256 IP addresses (16 hosts up) scanned in 7.55 seconds  
  
root@kali: ~  
# sudo nmap -ss -sV -O 192.168.1.138  
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-27 06:20 EST  
Nmap scan report for Redmi-Note-13 (192.168.1.138)  
Host is up (0.026s latency).  
All 1000 scanned ports on Redmi-Note-13 (192.168.1.138) are in ignored states.  
Not shown: 1000 closed tcp ports (reset)  
MAC Address: CA:45:34:AC:76:47 (Unknown)  
Too many fingerprints match this host to give specific OS details  
Network Distance: 1 hop  
  
OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .  
Nmap done: 1 IP address (1 host up) scanned in 5.59 seconds  
  
root@kali: ~
```

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.