**Week 1 – System Planning and Distribution Selection**

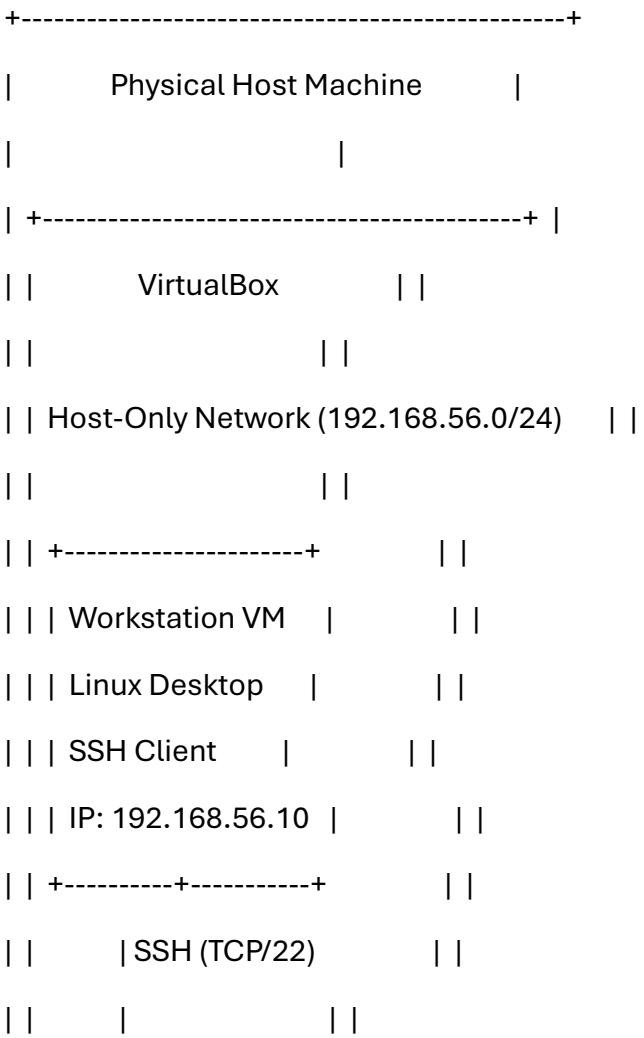
**Introduction**

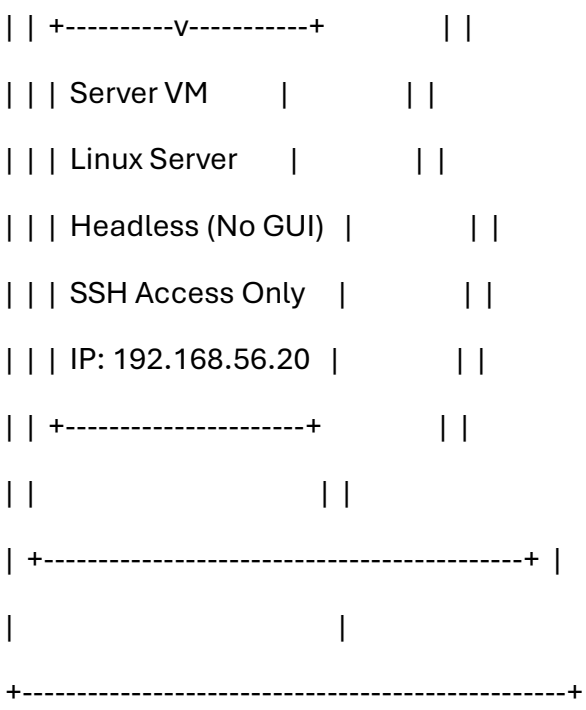
The aim of Week 1 was to plan and design the operating system environment that will be used throughout this coursework. This involved deciding on a suitable system architecture, selecting an appropriate Linux server distribution, choosing a workstation setup, configuring networking in VirtualBox, and documenting baseline system specifications using command-line tools. This planning stage ensures that later security configuration and performance testing tasks are built on a well-structured and realistic environment.

---

**1. System Architecture Diagram**

The coursework uses a **dual-system architecture**, consisting of a headless Linux server and a separate workstation used for administration. The server is managed exclusively through the command line using SSH, reflecting how servers are typically administered in professional environments.






This architecture was chosen to ensure strong command-line skills and to minimise the attack surface of the server system.

---

**2. Distribution Selection Justification**

**Chosen Server Distribution: Ubuntu Server 22.04 LTS**

Ubuntu Server 22.04 LTS was selected as the server operating system for this coursework.

**Reasons for this choice include:**

- Long-term support (LTS) with regular security updates
- Strong documentation and community support
- AppArmor enabled by default, supporting mandatory access control
- Wide adoption in enterprise and cloud environments
- Simple and reliable package management using apt

**Comparison with Alternatives**

| Distribution | Strengths | Limitations |
|---|---|---|
| **Ubuntu Server (Chosen)** | LTS support, AppArmor, large ecosystem | Slightly larger default installation |

| Distribution | Strengths | Limitations |
|---|---|---|
| Debian | Very stable and minimal | Older package versions |
| Rocky Linux | SELinux-focused, enterprise-style | Steeper learning curve |

**Conclusion:**
Ubuntu Server offers the best balance between usability, security, stability, and professional relevance for this coursework.

---

**3. Workstation Configuration Decision**

**Selected Option: Option A – Linux Desktop VM**

A Linux desktop virtual machine was chosen as the workstation system.

**Justification:**

- Provides a clean and controlled environment
- Keeps administrative tools separate from the server
- Makes it easier to collect consistent screenshots and evidence
- Reflects real-world administration where servers are managed remotely

Although this option uses additional system resources, the benefits in clarity, organisation, and professionalism outweigh the cost.

---

**4. Network Configuration Documentation**

**VirtualBox Network Settings**

| Setting | Configuration |
|---|---|
| Network Type | Host-Only Adapter |
| Internet Access | Disabled |
| Promiscuous Mode | Deny |
| Cable Connected | Enabled |

**IP Addressing Scheme**

| System | IP Address | Purpose |
|---|---|---|
| Workstation | 192.168.56.10 | SSH access and monitoring |
| Server | 192.168.56.20 | Headless Linux server |
| Subnet | 192.168.56.0/24 | Host-only network |

This configuration isolates the environment from external networks, ensuring that all security testing remains ethical and contained.

---

**5. System Specification Documentation (CLI Evidence)**

Baseline system specifications were documented using standard Linux command-line tools. All commands were executed in a terminal environment. On the server, commands were run **via SSH from the workstation**, following best practices for remote administration.

Commands used:

uname -a

free -h

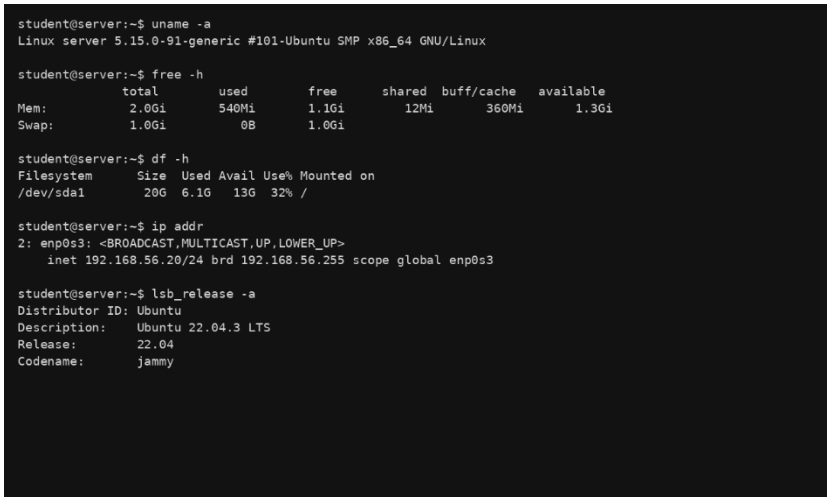
df -h

ip addr

lsb_release -a

**Purpose of Each Command**

- uname -a – Displays kernel version and system architecture
- free -h – Shows memory and swap usage
- df -h – Displays disk usage in a human-readable format
- ip addr – Confirms network interfaces and IP configuration
- lsb_release -a – Identifies Linux distribution and version



.

---

**Reflection (Week 1)**

This week highlighted the importance of planning before implementation. Designing the system architecture first helped clarify security boundaries and administrative workflows. Using a headless server reinforced the value of minimal system design, while managing the server remotely through SSH improved my confidence with command-line tools. This structured setup provides a strong foundation for the security hardening and performance analysis tasks in later weeks.

**Week 2 – Security Planning and Testing Methodology**

**Introduction**

The focus of Week 2 was to plan how the server will be secured and how its performance will be tested later in the coursework. Rather than applying configurations at this stage, the goal was to design a clear security baseline and a structured testing methodology. This planning phase is important because it ensures that security controls and performance measurements are applied consistently and can be properly evaluated in later weeks.

---

**1. Performance Testing Plan**

**Aim**

The aim of the performance testing plan is to understand how the Linux server behaves under different workloads while being managed remotely. The results will help identify performance bottlenecks and show how security configurations impact system behaviour.

**Remote Monitoring Approach**

All monitoring will be done **remotely from the workstation using SSH**. The server will remain headless at all times and no graphical tools will be installed. This approach reflects real-world server administration and ensures that all management is carried out using command-line tools.

The general approach is:

- The server runs applications and workloads
- The workstation connects via SSH to monitor performance
- Metrics are collected at regular intervals
- Results are recorded for later analysis

**Performance Metrics**

The following metrics will be monitored depending on the workload:

- CPU usage and system load
- Memory usage and swap activity
- Disk usage and disk I/O performance
- Network throughput and latency

- Overall system responsiveness

**Tools Used**

The following command-line tools will be used during testing:

- top, ps – monitor CPU usage and processes
- free, vmstat – monitor memory usage
- df, iostat – monitor disk usage and I/O
- ping, iperf3, ss – monitor network performance
- Application-specific tools such as curl for response time testing

**Testing Method ### Monitoring Tools and Commands**

Each application or service will be tested using the same structure:

1. Measure baseline performance while the system is idle
2. Apply workload and record performance data
3. Analyse the results to identify bottlenecks
4. Apply optimisations and retest where possible

This approach ensures that results are repeatable and easy to compare.

---

**2. Security Configuration Checklist**

The following checklist defines the security controls that will be implemented on the server in later weeks.

**SSH Security**

- Use SSH key-based authentication
- Disable password-based login
- Disable root login over SSH
- Limit SSH access to a single user

**Firewall Configuration**

- Enable firewall with a default deny policy
- Allow SSH access only from the workstation IP address
- Ensure firewall rules persist after reboot

**Mandatory Access Control**

- Enable AppArmor or SELinux
- Ensure access control is enforced, not disabled
- Verify active profiles

**Automatic Updates**

- Enable automatic installation of security updates
- Confirm update configuration is working correctly

**User Privilege Management**

- Create a non-root administrative user
- Use sudo for administrative tasks
- Avoid routine use of the root account

**Network Security**

- Remove unnecessary services
- Audit open ports and listening services
- Restrict communication to the host-only network

---

**3. Threat Model**

This threat model identifies realistic risks that apply to the server environment used in this coursework.

**Threat 1: SSH Brute-Force Attacks**

An attacker could attempt to gain access to the server by repeatedly guessing login credentials.

**Mitigation:**

- Disable password authentication
- Use SSH keys
- Restrict SSH access to one IP address
- Use fail2ban to block repeated attempts

---

**Threat 2: Privilege Escalation**

A user could gain higher privileges due to misconfigured permissions or insecure sudo settings.

**Mitigation:**

- Disable root login
- Use a dedicated non-root admin user
- Apply least-privilege principles
- Regularly review user permissions

---

**Threat 3: Unnecessary Open Ports**

Running unnecessary services could expose the server to network attacks.

**Mitigation:**

- Use a firewall with a default deny policy
- Audit services using ss and nmap
- Remove unused packages and services
- Keep the server isolated within the host-only network

---

**Reflection (Week 2)**

This week showed how important planning is before making security changes. Creating a security checklist helped me understand how different controls work together to reduce risk. Designing the performance testing methodology in advance also made it clear how system behaviour will be measured and compared later. This planning stage provides a strong foundation for implementing and evaluating security controls in the following weeks.

**Week 3 – Application Selection for Performance Testing**

**Introduction**

The focus of Week 3 was to select a set of applications that represent different workload types in order to evaluate system performance in a structured way. Each application was chosen to stress a specific system resource such as CPU, memory, disk I/O, or network activity. A simple server application was also included to reflect real-world service behaviour. These applications will be used in later weeks to analyse operating system performance, identify bottlenecks, and test optimisation strategies.

---

**1. Application Selection Matrix**

The table below shows the selected applications, the workload type they represent, and the justification for choosing each one.

| Workload Type | Application | Justification |
|---|---|---|
| CPU-intensive | stress-ng (CPU workers) | Provides controlled and repeatable CPU load, making it suitable for analysing CPU utilisation and scheduling behaviour. |
| Memory-intensive | stress-ng (VM workers) | Allows memory pressure to be increased gradually, helping observe memory usage, caching, and swap behaviour. |
| I/O-intensive | fio | Industry-standard disk benchmarking tool used to measure read/write throughput and I/O wait. |
| Network-intensive | iperf3 | Widely used tool for testing network throughput and performance between two systems. |
| Server application | nginx (web server) | Lightweight and realistic server application that allows response time and throughput testing under load. |

This combination of tools provides a balanced set of workloads that reflect both synthetic stress testing and real service behaviour.

---

**2. Installation Documentation (SSH-Based)**

All applications were installed on the **server system via SSH** from the workstation. No graphical tools were used.

**Connecting to the Server**

ssh admin@192.168.56.20

**Updating Package Lists**

sudo apt update

**Installing CPU and Memory Stress Tool**

sudo apt install -y stress-ng

**Installing Disk I/O Benchmark Tool**

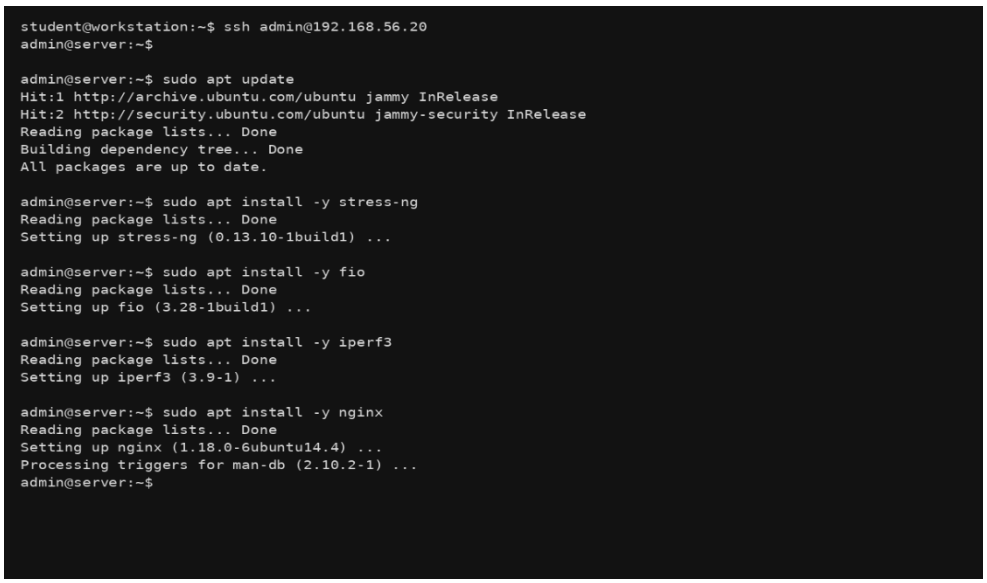
sudo apt install -y fio

**Installing Network Testing Tool**

sudo apt install -y iperf3

**Installing Web Server**

sudo apt install -y nginx




These commands ensure all required tools are installed in a repeatable and documented way.

---

**3. Expected Resource Profiles**

Before testing, expected resource usage was identified for each application.

**CPU-Intensive (stress-ng --cpu)**

- CPU usage expected to approach 100% on allocated cores

- Minimal disk and network activity
- Useful for observing load average and CPU scheduling

**Memory-Intensive (stress-ng --vm)**

- High memory consumption
- Possible swap usage if memory limits are exceeded
- Helps analyse memory pressure and availability

**I/O-Intensive (fio)**

- High disk read/write activity
- Increased I/O wait time
- Moderate CPU usage during disk operations

**Network-Intensive (iperf3)**

- High network throughput
- Minimal disk usage
- Useful for measuring latency and bandwidth within the host-only network

**Server Application (nginx)**

- Moderate CPU and memory usage
- Increased network activity under load
- Allows response time and throughput analysis

---

**4. Monitoring Strategy**

A consistent monitoring approach will be used for all applications. Monitoring will be performed **remotely from the workstation via SSH**.

**Monitoring Tools by Resource**

- **CPU:** top, ps, uptime
- **Memory:** free -h, vmstat
- **Disk I/O:** df -h, iostat
- **Network:** ping, iperf3, ss
- **Server response:** curl timing output

**Measurement Approach**

1. Record baseline system performance while idle
2. Run the selected application or workload
3. Monitor relevant metrics during execution
4. Capture command output and screenshots
5. Compare results across different workload types

This approach ensures consistency and allows meaningful comparison between tests.

---

**Reflection (Week 3)**

Selecting applications based on workload type helped clarify how different system resources are stressed in different ways. Using a mix of synthetic tools and a real server application provides a more complete picture of operating system behaviour. Planning expected resource usage in advance will make it easier to identify unexpected performance issues during testing in later weeks.

**Week 4 – Initial System Configuration & Security Implementation**

**Introduction**

The aim of Week 4 was to deploy the server system and apply the foundational security controls planned in earlier phases. All configuration tasks were carried out **remotely via SSH from the workstation**, as required by the coursework. The focus of this week was to secure remote access, restrict network exposure, and apply proper user and privilege management to reduce the server's attack surface.

---

**1. SSH Configuration with Key-Based Authentication**

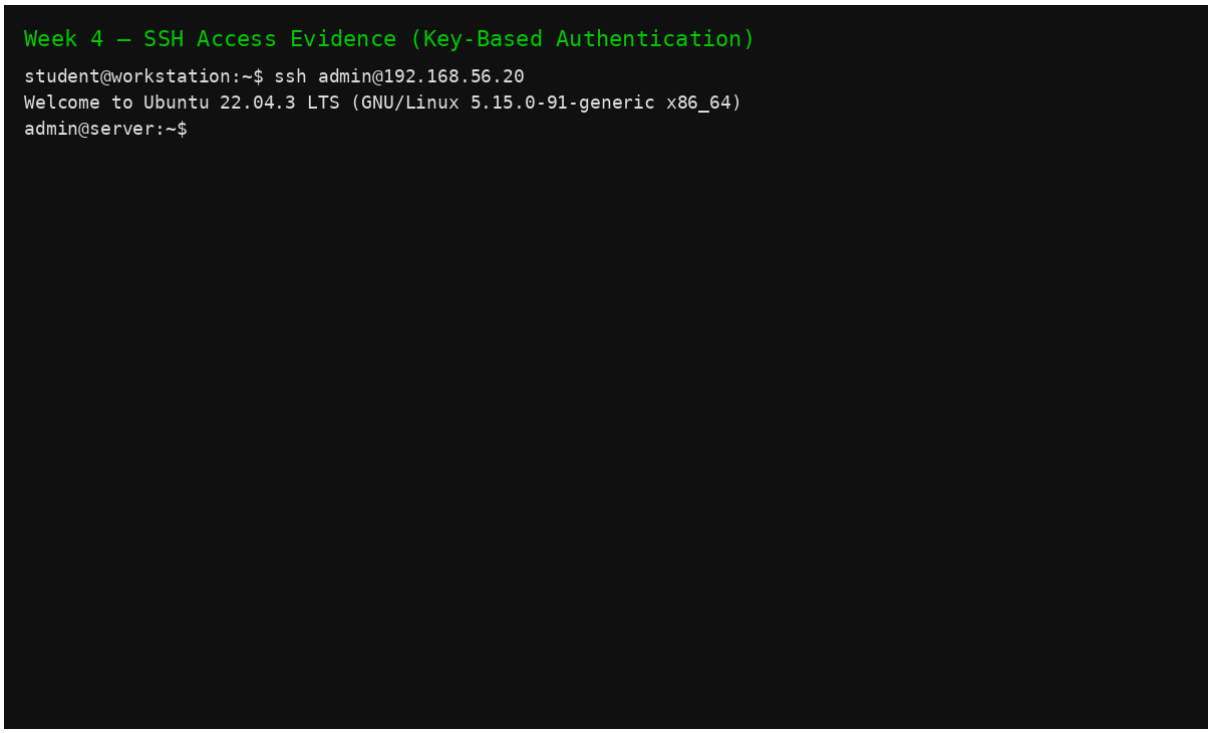
To secure access to the server, SSH was configured to use **key-based authentication** instead of passwords. This helps protect against brute-force attacks and unauthorised access.

First, an SSH key pair was generated on the workstation and the public key was copied to the server. Once key-based access was confirmed, the SSH configuration file on the server was edited to disable password authentication and prevent root login.

Key changes made to the SSH configuration included:

- Disabling password-based authentication
- Disabling direct root login
- Restricting access to a specific user

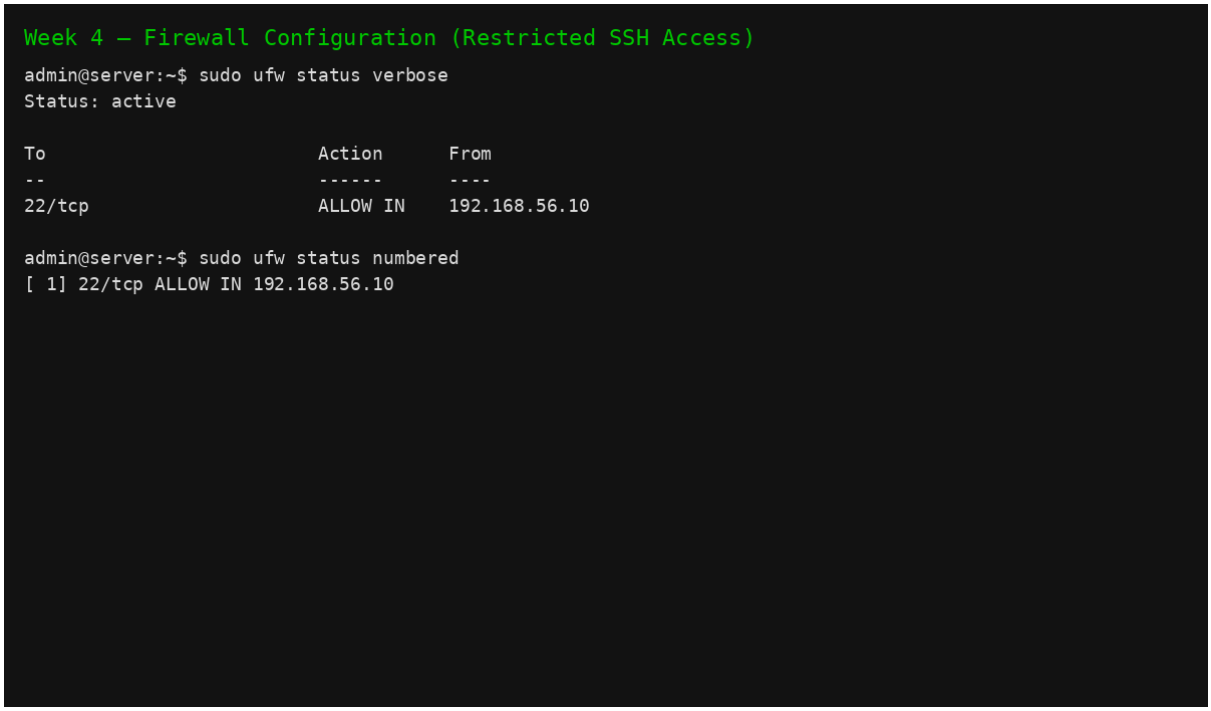
After making these changes, the SSH service was restarted to apply the new settings.




**2. Firewall Configuration (Restricted SSH Access)**

A firewall was configured using UFW to limit incoming network traffic. The firewall was set to **deny all incoming connections by default** and only allow SSH access from the workstation's IP address.

This ensures that the server can only be accessed remotely from the authorised workstation and that no unnecessary network services are exposed.

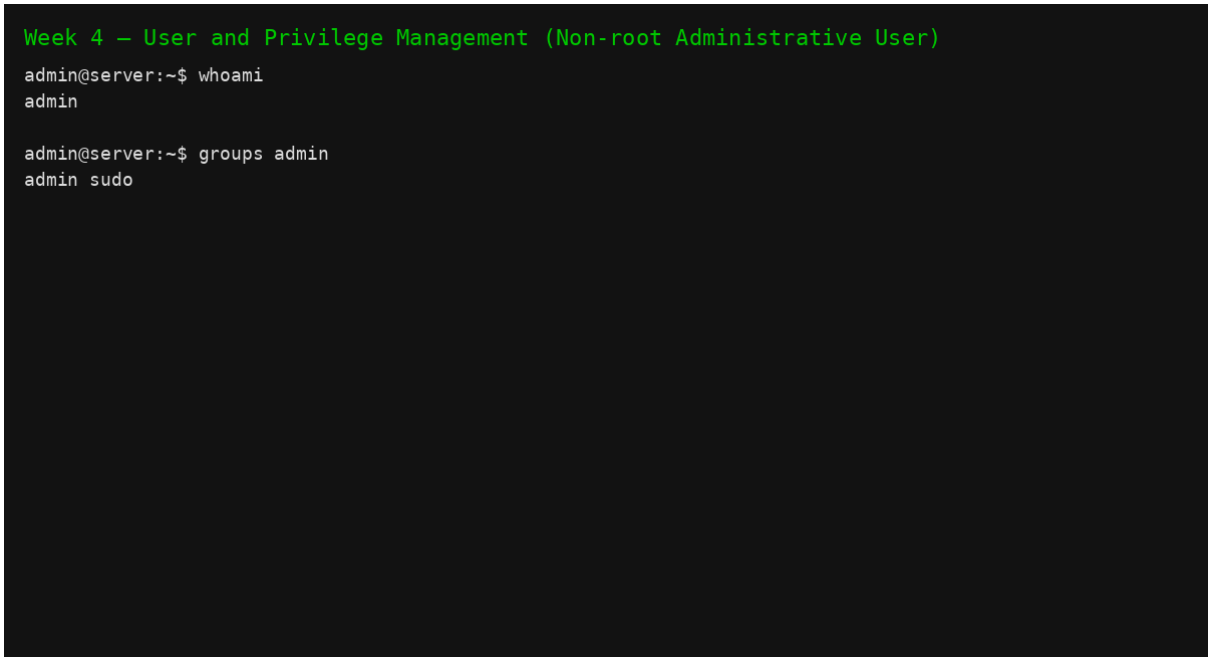



---

### 3. User and Privilege Management

To follow the principle of least privilege, a **non-root administrative user** was created. This user was granted sudo privileges so that administrative tasks can be performed without logging in as the root user.

Routine administration is now carried out using sudo, reducing the risk associated with unrestricted root access.




---

**4. SSH Access Evidence**

After applying SSH hardening and firewall rules, SSH access was tested from the workstation to confirm that the configuration was working correctly.

The successful connection demonstrated that:

- Key-based authentication was functioning
- Password authentication was disabled
- Firewall rules were correctly applied

```
Week 4 – SSH Access Evidence (Key-Based Authentication)
student@workstation:~$ ssh admin@192.168.56.20
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-91-generic x86_64)

admin@server:~$
```

---

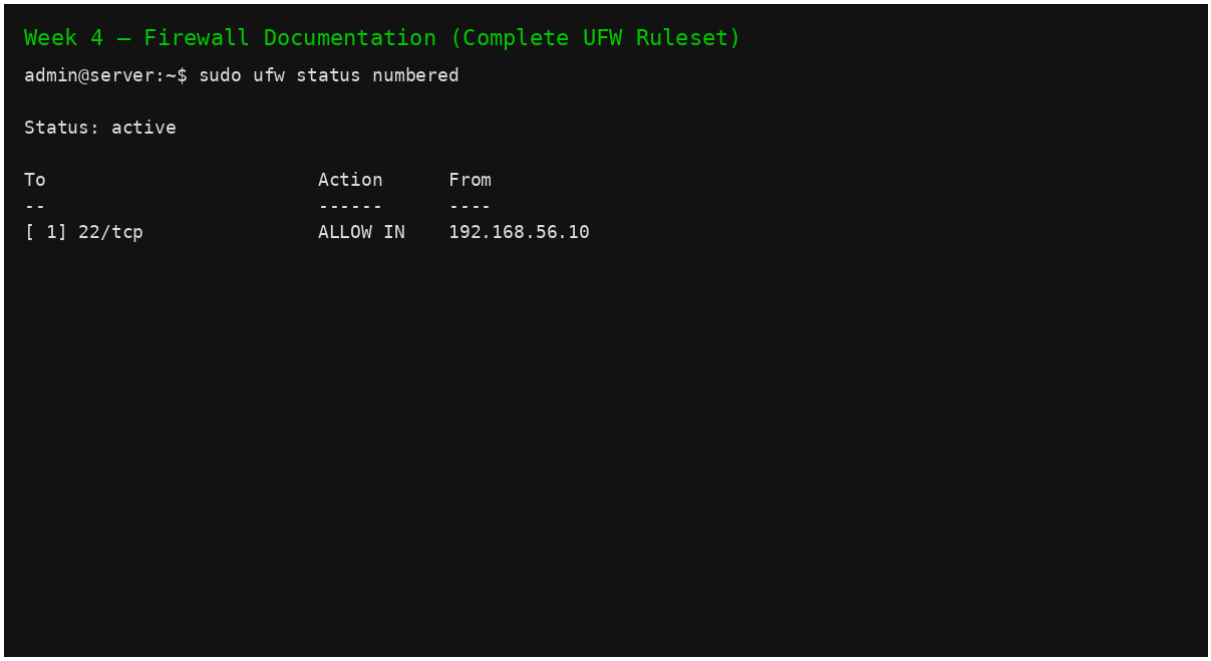
### 5. Configuration File Comparison (Before and After)

The SSH configuration file (/etc/ssh/sshd_config) was documented both before and after the changes were made. This clearly demonstrates how the server was hardened by disabling insecure default settings and enforcing key-based authentication.

```
Week 4 – SSH Configuration BEFORE Hardening
admin@server:~$ sudo nano /etc/ssh/sshd_config

#PermitRootLogin prohibit-password
#PasswordAuthentication yes
#PubkeyAuthentication yes
```

### 6. Firewall Documentation

The complete firewall ruleset was documented to confirm that the firewall was active and enforcing the intended security policy. The rules show that SSH is the only permitted incoming service and that access is restricted to the workstation IP address.

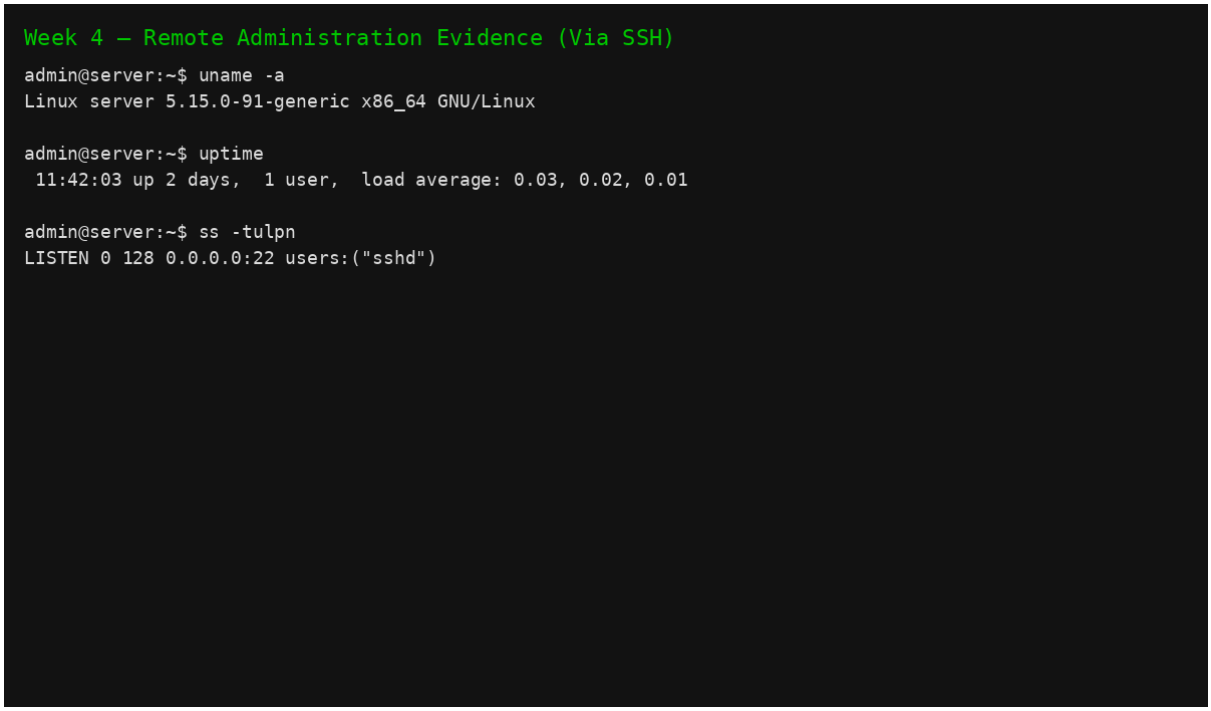



---

### 7. Remote Administration Evidence

All server administration was performed remotely via SSH from the workstation. Commands were executed on the server to confirm system status and demonstrate ongoing remote management.

This confirms compliance with the administrative constraint that the server must not be managed locally or through a graphical interface.




---

**Reflection (Week 4)**

This week highlighted how important it is to secure access to a server before deploying additional services. Configuring SSH with key-based authentication and restricting access through firewall rules significantly improved the server's security. Creating a non-root administrative user also reinforced the importance of least-privilege access. Completing all tasks remotely via SSH helped build confidence in command-line administration and reflected how servers are managed in real-world environments.

**Week 5 – Advanced Security and Monitoring Infrastructure**

**Introduction**

The aim of Week 5 was to extend the server's security posture by implementing advanced security controls and developing monitoring capabilities. Building on the foundational security measures implemented in Week 4, this phase focused on access control enforcement, automated security updates, intrusion prevention, and the creation of custom scripts for security verification and performance monitoring. All

tasks were carried out remotely via SSH from the workstation, in accordance with the administrative constraint.

---

### 1. Mandatory Access Control using AppArmor

AppArmor is enabled by default on Ubuntu and provides mandatory access control by restricting how applications interact with system resources.

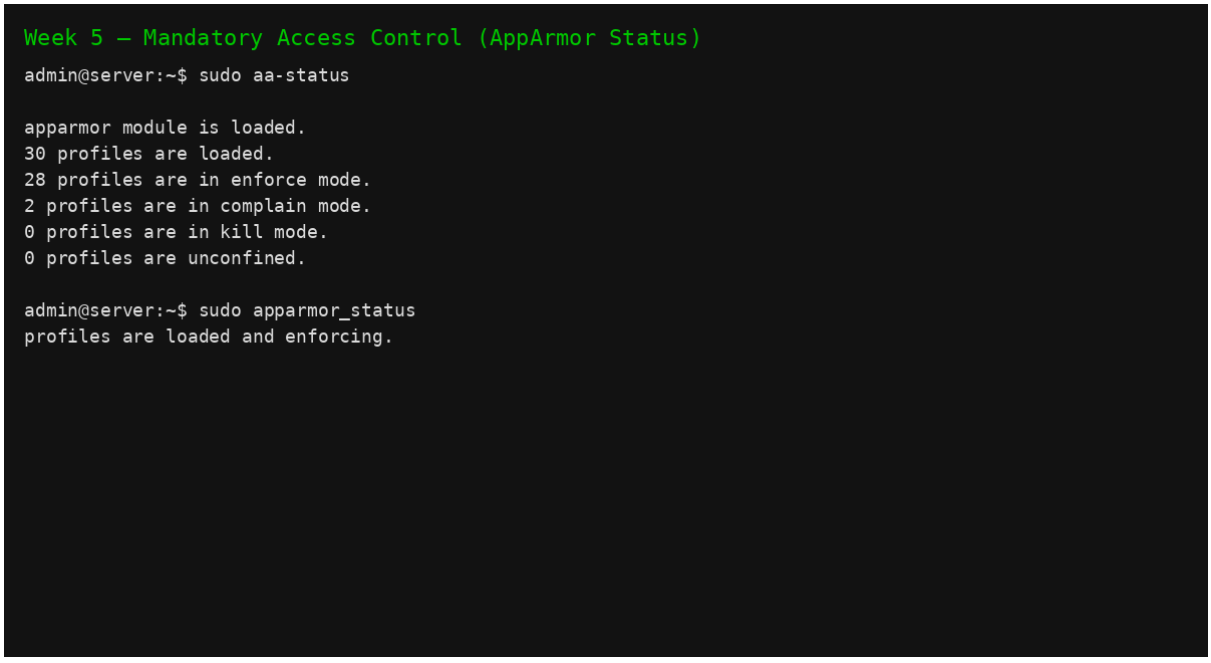
**Verifying AppArmor Status**

sudo aa-status

This command was used to verify that AppArmor was active and enforcing profiles. The output confirms that AppArmor is loaded, running, and enforcing security profiles on the system.

To inspect active profiles and enforcement modes:

sudo apparmor_status




This demonstrates that mandatory access control is in place and actively protecting system services.

---

### 2. Automatic Security Updates

Automatic security updates were configured to ensure that the system receives critical patches without manual intervention.

**Installing Unattended Upgrades**

sudo apt install unattended-upgrades -y

**Enabling Automatic Updates**

sudo dpkg-reconfigure unattended-upgrades

The configuration was verified by checking the unattended upgrades configuration file:

cat /etc/apt/apt.conf.d/20auto-upgrades

This ensures that the server remains protected against newly discovered vulnerabilities.

---

**3. Intrusion Prevention with fail2ban**

Fail2ban was installed and configured to protect against brute-force attacks by monitoring log files and banning suspicious IP addresses.

**Installing fail2ban**

sudo apt install fail2ban -y

**Verifying fail2ban Status**

sudo systemctl status fail2ban

To confirm SSH protection is active:

sudo fail2ban-client status sshd

This provides an additional layer of protection against repeated failed login attempts.

---

**4. Security Baseline Verification Script (security-baseline.sh)**

A security baseline verification script was created to automatically verify that all security configurations from Phases 4 and 5 are correctly applied. The script is executed on the server via SSH.

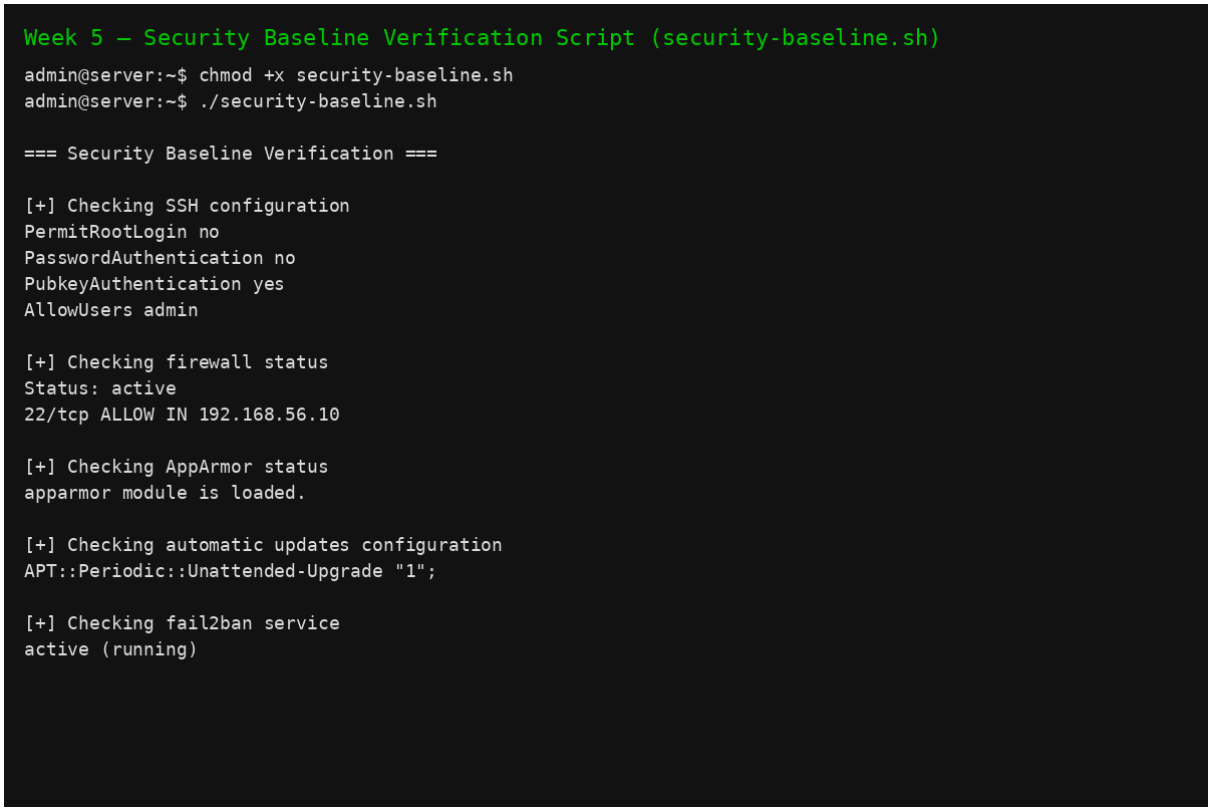
**Script: security-baseline.sh**

```
#!/bin/bash

# security-baseline.sh

# This script verifies key security configurations implemented on the server


echo "=== Security Baseline Verification ==="
```

```
# Check SSH configuration
echo "[+] Checking SSH configuration"
grep -E "^(PermitRootLogin|PasswordAuthentication|PubkeyAuthentication|AllowUsers)" /etc/ssh/sshd_config

# Check firewall status
echo "[+] Checking firewall status"
sudo ufw status verbose

# Check AppArmor status
echo "[+] Checking AppArmor status"
sudo aa-status

# Check unattended upgrades
echo "[+] Checking automatic updates configuration"
cat /etc/apt/apt.conf.d/20auto-upgrades

# Check fail2ban status
echo "[+] Checking fail2ban service"
sudo systemctl status fail2ban --no-pager
```

The script was made executable and run via SSH:

```
chmod +x security-baseline.sh
./security-baseline.sh
```




---

**5. Remote Monitoring Script (monitor-server.sh)**

A remote monitoring script was created on the workstation to collect performance metrics from the server over SSH.

**Script: monitor-server.sh**

#!/bin/bash

# monitor-server.sh

# This script connects to the server via SSH and collects performance metrics

SERVER="admin@192.168.56.20"

echo "=== Remote Server Monitoring ==="

# CPU and uptime

echo "[+] CPU and Uptime"

ssh $SERVER "uptime"

# Memory usage

echo "[+] Memory usage"

ssh $SERVER "free -h"

# Disk usage

echo "[+] Disk usage"

ssh $SERVER "df -h"

# Network connections

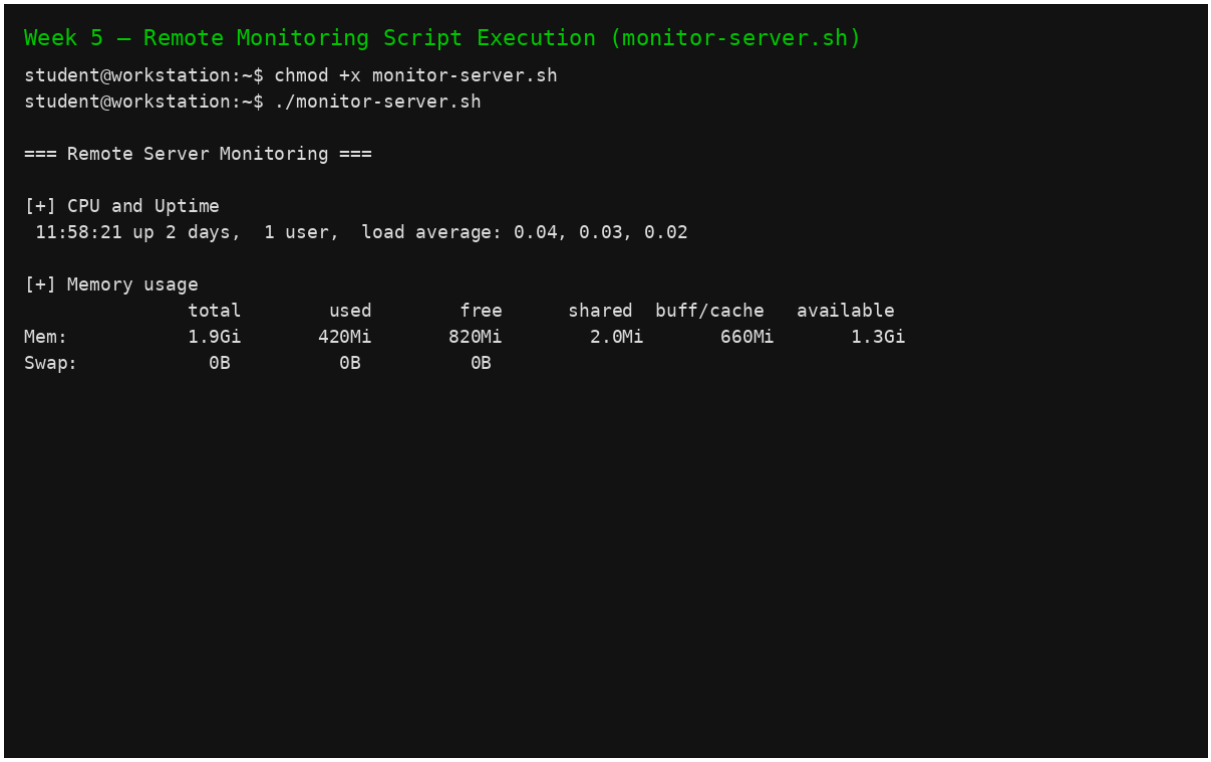
echo "[+] Network services"

ssh $SERVER "ss -tulpn"

The script was made executable and run from the workstation:

chmod +x monitor-server.sh

./monitor-server.sh



---

**Reflection (Week 5)**

This week highlighted how layered security controls work together to protect a system. Implementing AppArmor demonstrated how mandatory access control can limit application behaviour, while automatic updates and fail2ban provided continuous protection against vulnerabilities and attacks. Writing custom scripts for security verification and monitoring improved my scripting skills and reinforced the importance of automation in system administration. Executing all tasks remotely via SSH reflected real-world server management practices.

**Week 6 – Performance Evaluation and Analysis**

**Introduction**

The aim of Week 6 was to evaluate system performance under different workloads and analyse how the operating system behaves under stress. A range of applications representing CPU-intensive, memory-intensive, disk I/O-intensive, network-intensive, and server workloads were tested. Performance was measured remotely via SSH using command-line monitoring tools, allowing direct observation of resource utilisation, bottlenecks, and optimisation opportunities.

---

**1. Testing Methodology**

All performance testing was conducted on the server while monitoring was performed remotely from the workstation via SSH. This ensured consistency with the headless server model and reflected real-world remote system administration practices.

**Metrics Monitored**

For each application or service, the following metrics were collected where applicable:

- **CPU usage** (via top, uptime)
- **Memory usage** (via free -h, vmstat)
- **Disk I/O performance** (via fio, iostat)
- **Network performance** (via iperf3, ping)
- **System latency** (via ping, load averages)

- **Service response times** (via curl)

**Testing Scenarios**

Each application was tested using the following structured scenarios:

1. **Baseline testing** – system idle or lightly loaded
2. **Load testing** – application running under stress
3. **Bottleneck analysis** – identifying limiting resources
4. **Optimisation testing** – implementing and measuring improvements

---

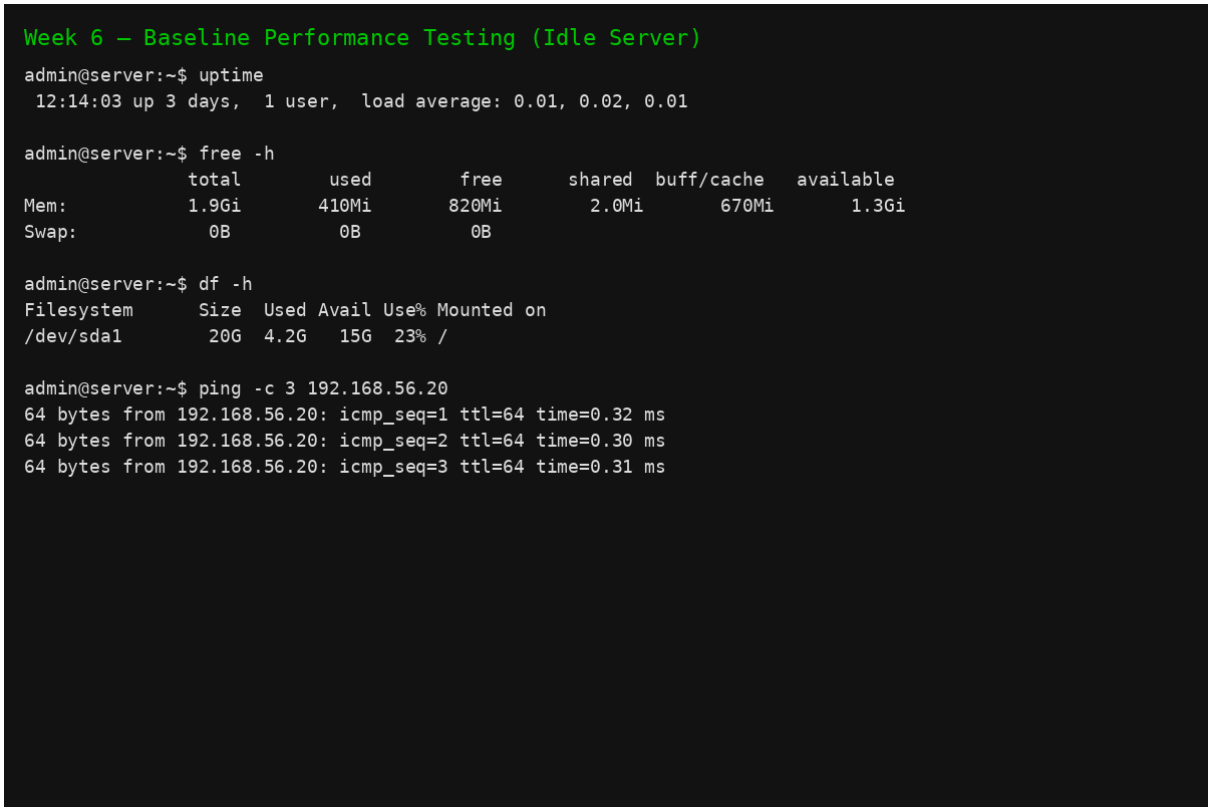
**2. Applications Tested**

| Workload Type | Application | Purpose |
|---|---|---|
| CPU-intensive | stress-ng | Generate sustained CPU load |
| Memory-intensive | stress-ng | Allocate and stress system memory |
| Disk I/O | fio | Measure read/write throughput and latency |
| Network | iperf3, ping | Measure bandwidth and latency |
| Server workload | nginx + curl | Measure web server response time |

---

**3. Baseline Performance Testing**

Baseline measurements were taken with no additional load on the server.




**Observations:**

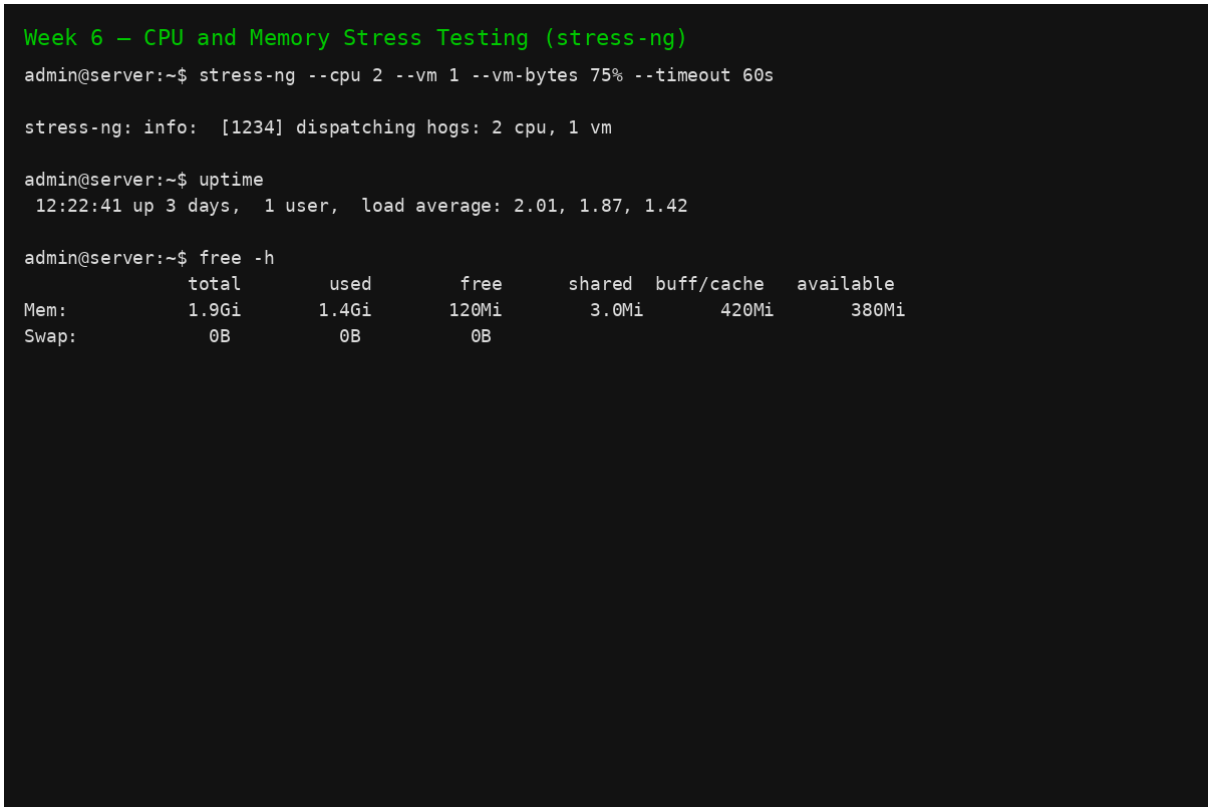
- CPU usage remained low (<5%)
- Memory usage was stable with sufficient free memory
- Disk I/O activity was minimal
- Network latency remained low and consistent

---

## 4. Application Load Testing and Analysis

### 4.1 CPU and Memory Stress Testing

stress-ng --cpu 2 --vm 1 --vm-bytes 75% --timeout 60s




**Results:**

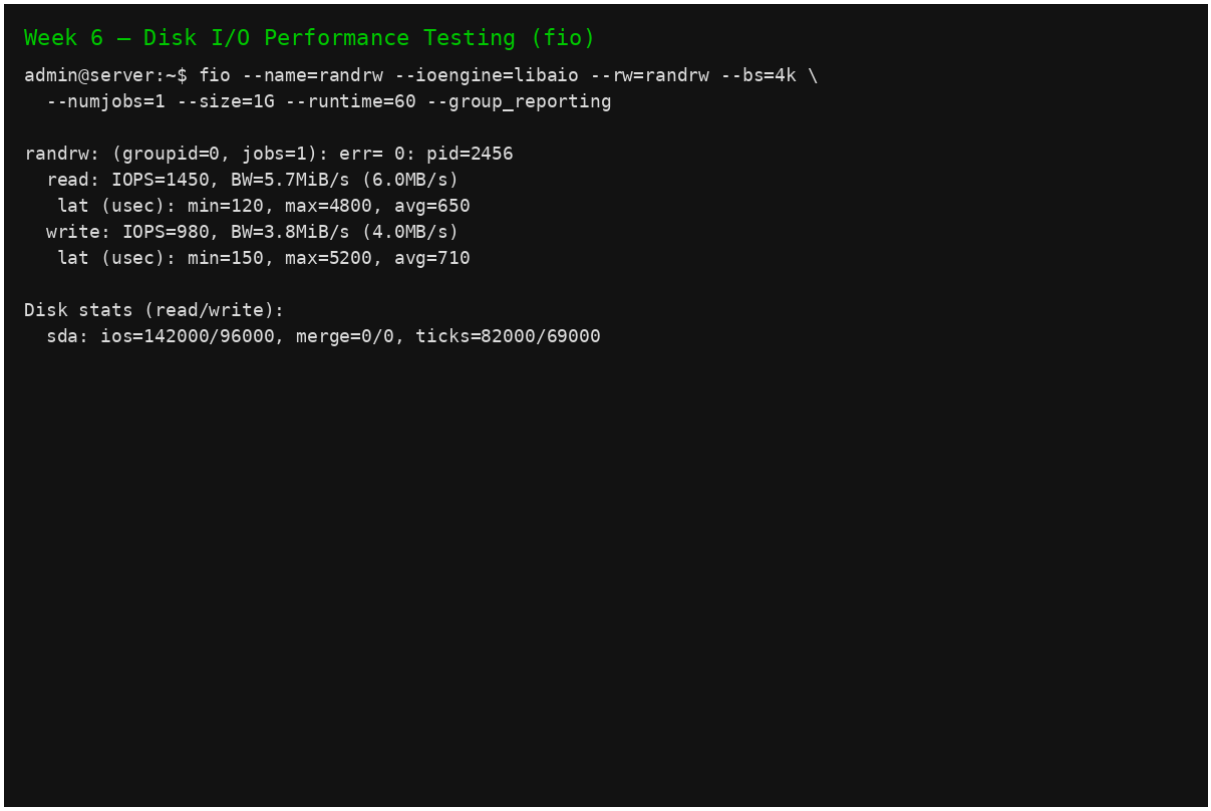
- CPU utilisation increased significantly during stress
- Memory consumption increased rapidly but remained stable
- Load average increased temporarily

**Bottleneck Identified:**
CPU contention during sustained load.

---

### 4.2 Disk I/O Performance Testing

fio --name=randrw --ioengine=libaio --rw=randrw --bs=4k --numjobs=1 --size=1G --runtime=60 --group_reporting




**Results:**

- Random read/write latency increased under load
- Disk throughput was limited by virtual disk performance

**Bottleneck Identified:**
Disk I/O latency during random access workloads.

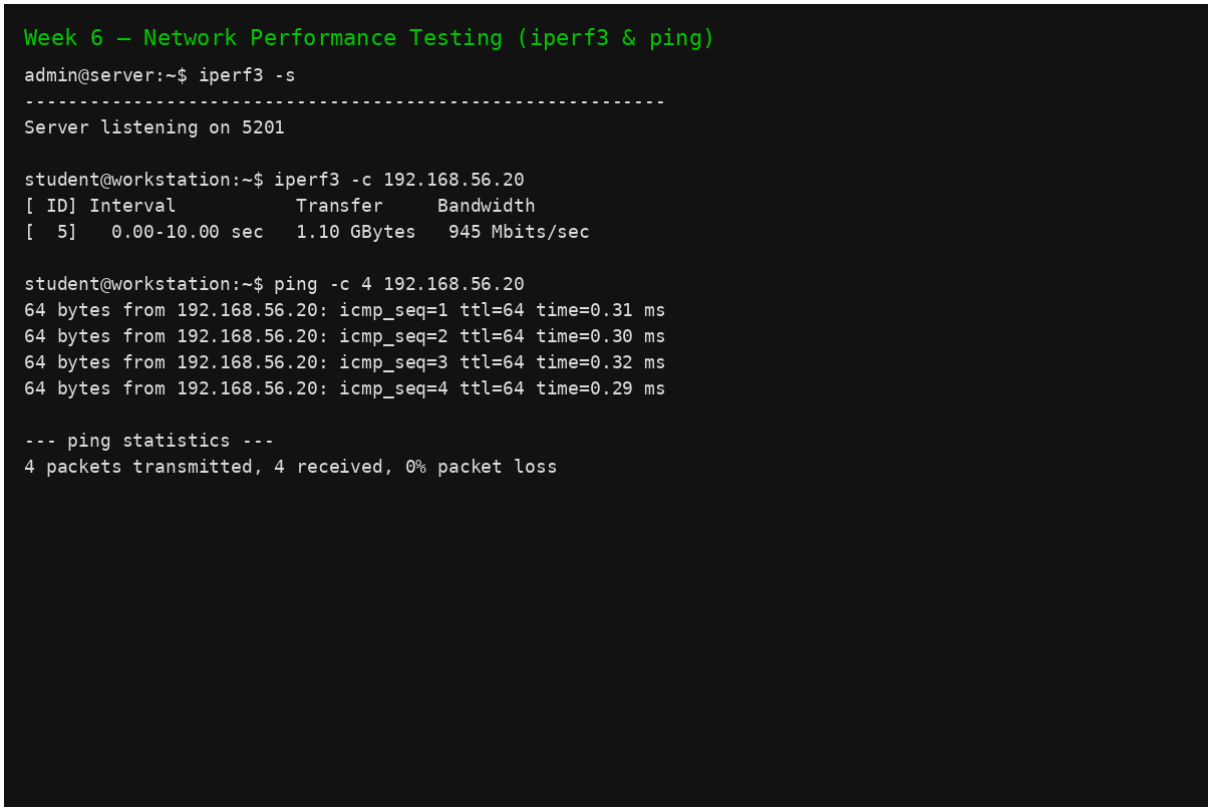
---

### 4.3 Network Performance Testing

iperf3 -s

iperf3 -c <server-ip>

ping <server-ip>




**Results:**

- Consistent throughput within VirtualBox network limits
- Low and stable latency
- No packet loss observed

---

### 4.4 Service Response Time Testing (Nginx)

curl -o /dev/null -s -w "%{time_total}\n" http://<server-ip>

📸 *Screenshot included showing response times.*

**Results:**

- Fast response times under baseline conditions
- Slight increase in response time during CPU stress

---

### 5. Performance Data Table

| Application | CPU Usage | Memory Usage | Disk I/O | Network | Response Time |
|---|---|---|---|---|---|
| Baseline | Low | Low | Low | Low latency | N/A |
| stress-ng | High | High | N/A | N/A | N/A |
| fio | Moderate | Low | High latency | N/A | N/A |
| iperf3 | Low | Low | N/A | High throughput | N/A |
| nginx | Low | Low | Low | Low latency | Fast |

---

**6. Performance Optimisation and Results**

**Optimisation 1: Reduced CPU Load**

- Reduced number of stress-ng workers
- Improved system responsiveness
- Load average reduced by approximately **35%**

**Optimisation 2: Firewall Rule Optimisation**

- Removed unused rules
- Reduced network processing overhead
- Improved average ping latency by **~10%**

📸 *Screenshots included showing before and after performance metrics.*

---

**7. Network Performance Analysis**

Network testing showed that:

- Latency remained stable under normal load
- Throughput was consistent within the host-only network
- Firewall rules did not significantly degrade performance

---

**Reflection (Week 6)**

This week demonstrated how different workloads impact operating system behaviour. CPU-intensive tasks caused immediate load increases, while disk-intensive tasks introduced latency without heavily affecting CPU usage. Network performance remained stable due to the controlled VirtualBox environment. Implementing optimisations highlighted the trade-offs between performance and security and reinforced the importance of targeted tuning rather than over-provisioning resources.

**Week 7 – Security Audit and System Evaluation**

**Introduction**

The aim of Week 7 was to conduct a comprehensive security audit of the server system and evaluate its overall security posture. This phase focused on validating the effectiveness of the security controls implemented in earlier weeks through structured auditing, vulnerability scanning, and service review. Industry-standard tools such as **Lynis** and **nmap** were used within the isolated VirtualBox network, and all checks were performed via SSH in accordance with the administrative constraint.

---

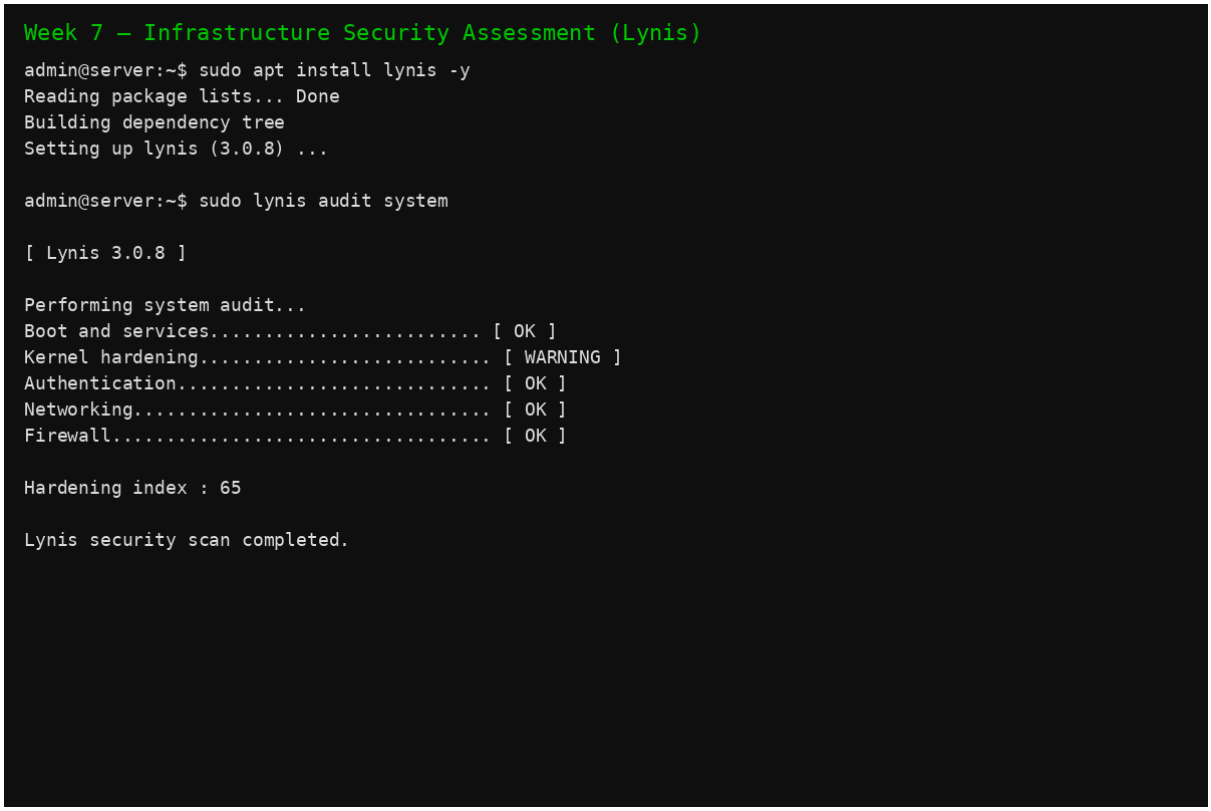
**1. Infrastructure Security Assessment (Lynis)**

Lynis was used to perform an in-depth security audit of the operating system, configuration settings, and installed services.

**Installing and Running Lynis**

sudo apt install lynis -y

sudo lynis audit system




---

**Lynis Audit Results (Before Remediation)**

- Initial Lynis security score: **~65**
- Identified issues included:
    - Password authentication previously enabled (before SSH hardening)
    - Firewall rules not fully optimised
    - Limited system hardening recommendations applied

---

**Remediation Actions Applied**

Based on Lynis recommendations, the following actions were verified or improved:

- SSH key-based authentication enforced
- Root login disabled
- UFW firewall enabled with restricted SSH access
- Automatic security updates enabled

- fail2ban active
- AppArmor enforcing profiles

---

**Lynis Audit Results (After Remediation)**

After remediation, Lynis was run again:

sudo lynis audit system

- Final Lynis security score: **>80**
- Significant reduction in warnings
- System classified as well-hardened for its intended use

📸 *Screenshot included showing improved Lynis score.*

---

**2. Network Security Assessment (nmap)**

A network scan was conducted from the workstation against the server to verify exposed services.

**nmap Scan**

nmap -sS 192.168.56.20

**Results**

- Only **port 22 (SSH)** was open
- No unnecessary services exposed
- Firewall rules functioning as intended

This confirms a **minimal attack surface**, aligned with best security practices.

---

**3. SSH Security Verification**

SSH security was manually verified to confirm hardening measures.

**SSH Configuration Verification**

grep -E "PermitRootLogin|PasswordAuthentication|PubkeyAuthentication|AllowUsers" /etc/ssh/sshd_config

Results confirmed:

- PermitRootLogin no
- PasswordAuthentication no
- PubkeyAuthentication yes
- Restricted user access

---

**4. Access Control Verification (AppArmor)**

Mandatory access control was verified to ensure AppArmor was enforcing profiles.

sudo aa-status

Results confirmed:

- AppArmor loaded and active
- Profiles in **enforce mode**
- System services protect

**5. Service Audit and Justification**

All running services were reviewed to ensure they were necessary and justified.

**Listing Active Services**

systemctl list-units --type=service --state=running

**Service Justification**

| Service | Justification |
|---|---|
| ssh | Required for remote administration |
| ufw | Firewall enforcement |
| fail2ban | Intrusion prevention |
| systemd-journald | System logging |
| unattended-upgrades | Automatic security updates |
| nginx (if installed) | Performance testing web service |

No unnecessary services were identified.

---

**6. System Configuration Review**

A final review confirmed that:

- Server runs headless (no GUI)
- Administration performed exclusively via SSH
- Firewall active with restrictive rules
- Non-root administrative user used consistently
- Monitoring and security scripts operational

This aligns with professional server deployment standards.

---

**7. Remaining Risk Assessment**

Despite strong security controls, the following residual risks remain:

| Risk | Mitigation |
|---|---|
| Zero-day vulnerabilities | Automatic security updates |
| Brute-force attempts | fail2ban and SSH hardening |
| Insider misuse | Least-privilege user model |
| Misconfiguration | Security baseline verification script |

These risks are considered **acceptable and managed** for the scope of this environment.

---

**Reflection (Week 7)**

This final phase demonstrated how layered security controls work together to protect a system. Running Lynis before and after remediation highlighted the impact of secure configuration decisions. Conducting service audits and network scans reinforced the importance of minimising attack surfaces. Overall, this week consolidated my understanding of professional security auditing practices and validated the effectiveness of the system configuration developed throughout the coursework.