

SAS (System Administration and Security)

Table of Contents

PART 1 - LOGBOOK	1
Introduction	1
1.HTTP	1
1.1 Overview	1
1.2 Implementation Steps	1
1.3 Analysis	4
2.MySQL/MariaDB Database.....	4
2.1 Overview	4
2.2 Implementation Steps	5
2.3 Analysis	8
3.Firewall Configuration	8
3.1 Overview	8
3.2 Implementation Steps	8
3.3 Analysis	11
Conclusion	11
PART 2 : HTTPD	12
Literature Review.....	12
Vulnerability Analysis	13
Risk Assessment	15
Hardening Procedure.....	17
Conclusion	25
References.....	26

PART 1 - LOGBOOK

Introduction

This logbook documents the installation and configuration of Linux services and mechanisms of HTTP, MySQL/MariaDB database, and Firewall on CentOS7 machine. This will help to understand how these services work and enhance the systems security.

1.HTTP

1.1 Overview

In this lab, the objective is to install and test the Apache HTTP Server (HTTPD). First, the Apache server package will be installed, and the service will be checked to make sure it's running correctly. Then, the configuration files will be reviewed. The firewall settings will also be configured and set up to allow web traffic, ensuring the server is accessible. After that, a simple HTML file is created and tested. Finally, the Apache access and error logs will be looked at to understand how the server handles and records requests.

1.2 Implementation Steps

I. Install and start HTTPD Service:

- The “**yum -y install httpd**” command downloads and installs Apache along with its dependencies. The “**-y**” flag allows automatic installation without confirming prompts.

```
[root@comp1475-centos7-25 ~]# yum -y install httpd
```

Figure 1 Installing Apache HTTPD

- To activate the Apache server and allow it to handle web requests we need to start the service with “**systemctl start httpd.service**” command.

```
[root@comp1475-centos7-25 ~]# systemctl start httpd.service
```

Figure 2 Starting Apache Service

- To check the status of the HTTPD Service we run the below command “**systemctl status httpd**”. In this case the service shows as “**ACTIVE**”.

```
[root@comp1475-centos7-25 ~]# systemctl status httpd
■ httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
   Active: active (running) since Tue 2025-02-25 15:56:57 GMT; 4 days ago
     Docs: man:httpd(8)
```

Figure 3 HTTPD Status

II. Check configuration Files

The “**httpd.conf**” file is the main configuration file for the Apache HTTP Server. It defines how the server runs, its security settings, where files are stored, logging mechanisms, and how requests are handled.

```
[root@comp1475-centos7-25 ~]# find / -name httpd.conf
/etc/httpd/conf/httpd.conf
/usr/lib/tmpfiles.d/httpd.conf
[root@comp1475-centos7-25 ~]# less /etc/httpd/conf/httpd.conf
```

Figure 4 HTTPD Config File

The Apache configuration file includes the following default directives and can be modified to increase security:

- Server Root : The server root is located at **/etc/httpd/**, where Apache stores essential configuration files and modules. It enhances security by restricting direct access to critical files.
- Less privileged Users and Groups : Apache runs under the **apache** user and group to improve security. This prevents unauthorized access to important system files, reduces the chances of attacks, and follows the "least privilege" rule.
- .htaccess and .htpasswd : .htaccess contains access control rule and .htpasswd stores user credentials. Properly securing these files can prevent unauthorized access to sensitive server configurations.
- Error Log : The error log is stored at “**/var/log/httpd/error_log**” where the server-related errors are recorded for debugging .
- Access Log : The access log is stored at “**/var/log/httpd/access_log**” where it records incoming requests details like IP, requested URL, response codes, etc.
- Listening ports :
The server listens on **port 80**, which is designated for **HTTP** traffic, thus ensuring seamless communication with client requests.
- Web Document :
Apache stores web content, like files and resources, in **/var/www/html/**. This keeps the file structure organized and separate from important server configurations thus leading to improved security.
- Server Admin:
The server admin email is used to provide contact information for users to report issues or get support. It appears in error messages and helps with debugging. The email is also used to receive automatic alerts about server problems.

III. Check Firewall Rules

The **iptables -L INPUT** command shows firewall rules for incoming traffic, deciding if packets are accepted or blocked. HTTP can be allowed by the **firewall-cmd --add-service=http** command. This helps improve security by only allowing necessary services and blocking unwanted connections, while also making it easier to manage the firewall.

```
[root@comp1475-centos7-25 /]# iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source                destination           ctstate RELATED,ESTABLISHED
ACCEPT     all  --  anywhere              anywhere
ACCEPT     all  --  anywhere              anywhere
INPUT_direct all  --  anywhere              anywhere
INPUT_ZONES all  --  anywhere              anywhere
DROP       all  --  anywhere              anywhere              ctstate INVALID
REJECT     all  --  anywhere              anywhere              reject-with icmp-host-prohibited
```

Figure 5 Check Firewall rule

```
[root@comp1475-centos7-25 /]# firewall-cmd --add-service=http
success
```

Figure 6 Add HTTP to Firewall rule

IV. Testing the HTTP Server

To verify if the HTTPD server is functioning correctly, a test file named index.html is created in the /var/www/html/ directory.

```
[root@comp1475-centos7-25 html]# cd /var/www/html
[root@comp1475-centos7-25 html]# echo "<h1>Apache is Working</h1>" > index.html
[root@comp1475-centos7-25 html]# ls
index.html
```

Figure 7 HTML File

To test the HTTPD server first Lynx, a text-based web browser, is installed. Lynx was then used to view the HTML file contents as a web page in the terminal. This shows that the HTTPD server is running as expected.

```
[root@comp1475-centos7-25 ~]# lynx localhost_
```

Figure 8 Accessing Localhost

```
Apache is Working
```

Figure 9 Web page displayed

V. Verifying Logs

- **Error Logs:** The error log shows messages about the server's operation and any issues. The below log file indicates no major errors and just system-level notifications, and that the server is running normally without any critical problems.

```
[root@comp1475-centos7-25 /]# less /var/log/httpd/error_log
[Sun Mar 02 03:16:01.767923 2025] [lbmethod_heartbeat:notice] [pid 32702] AH02282: No slotmem from mod_heartbeat
[Sun Mar 02 03:16:01.779801 2025] [mpm_prefork:notice] [pid 32702] AH00163: Apache/2.4.6 (CentOS) PHP/5.4.16 configured -- resuming normal operations
[Sun Mar 02 03:16:01.779822 2025] [core:notice] [pid 32702] AH00094: Command line: '/usr/sbin/httpd -D FOREGROUND'
/var/log/httpd/error_log (END)
```

Figure 10 Error Log

- **Access Log:** The access log records every request made to the server, including the IP address of the client, the requested URL, the response status code, date and time the request was made, response size etc. This helps to track which resources were accessed, by whom, and whether they were successfully delivered.

```
[root@comp1475-centos7-25 /]# less /var/log/httpd/access_log
::1 - - [02/Mar/2025:15:23:38 +0000] "GET / HTTP/1.0" 200 53 "-" "Lynx/2.8.8dev.15 libwww-FM/2.14 SSL-M/1.4.1 OpenSSL/1.0.1e-fips"
/var/log/httpd/access_log (END)
```

Figure 11 Access Log

1.3 Analysis

Setting up the Apache HTTPD server helps create a safe and efficient website. Installing and starting the server makes it ready to handle web traffic. Checking the system settings ensures it runs smoothly with multiple users. The configuration files control who can access the server, keep track of activity, and help fix problems if something goes wrong. Testing the server and checking logs confirm that everything is working properly. These steps keep the website safe, fast, and reliable.

2.MySQL/MariaDB Database

2.1 Overview

In this lab, the objective is to install, configure, and test the MariaDB database server. First, the MariaDB client and server packages will be installed to enable database interaction and hosting. The MariaDB service will then be started and checked to ensure it's running correctly. After that, the database will be logged in to, and user, databases, tables, and privileges will be provided to manage access and ensure secure operations.

2.2 Implementation Steps

I. Installing MariaDB

After logging in as root user we can perform installation process. The “**yum -y install mariadb**” command downloads and installs MariaDB client, which allows users to interact with the database. To enable database hosting and management “**yum -y install mariadb-server**” is installed, as the database service cannot run with this package. The “-y” flag allows automatic installation without confirming prompts.

```
[root@comp1475-centos7-25 ~]# yum -y install mariadb
```

Figure 12 Install MariaDB

```
[root@comp1475-centos7-25 ~]# yum -y install mariadb-server
```

Figure 13 Install MariaDB Server

II. Starting MariaDB service

To start MariaDB server the command “**systemctl start mariadb.service**” is used. It interacts with the database and perform tasks like retrieving, adding, or modifying data. It also makes the MariaDB service active and ready to be used, ensuring that the database is available for future operations or connections.

```
[root@comp1475-centos7-25 ~]# systemctl start mariadb.service
```

Figure 14 Starting the Service

To check the status of the MariaDB Service we run the below command “**systemctl status mariadb.service**”. In this case the service shows as “ACTIVE”.

```
[root@comp1475-centos7-25 ~]# systemctl status mariadb.service
■ mariadb.service - MariaDB database server
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; disabled; vendor preset: disabled)
   Active: active (running) since Tue 2025-02-25 15:14:30 GMT; 28min ago
   Process: 30416 ExecStartPost=/usr/libexec/mariadb-wait-ready $MAINPID (code=exited, status=0/SUCCESS)
```

Figure 15 Status of Maria DB

III. Reviewing Configuration File

The **my.cnf** file is the primary configuration file for MySQL/MariaDB and is located at “**/etc/my.cnf**”. The **mysqld_safe** contains safety and logging settings. The “**log-error=/var/log/mariadb/mariadb.log**” specifies the mysql log files and the “**pid-file=/var/run/mariadb/mariadb.pid**” stores the Process ID (PID) of the running mysqld process.

```
[root@comp1475-centos7-25 ~]# find / -name my.cnf
/etc/my.cnf
[root@comp1475-centos7-25 ~]# less /etc/my.cnf
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0
# Settings user and group are ignored when systemd is used.
# If you need to run mysqld under a different user or group,
# customize your systemd unit file for mariadb according to the
# instructions in http://fedoraproject.org/wiki/Systemd

[mysqld_safe]
log-error=/var/log/mariadb/mariadb.log
pid-file=/var/run/mariadb/mariadb.pid

#
# include all files from the config directory
#
!includedir /etc/my.cnf.d

(END)
```

Figure 16 Config File

IV. Logging in MYSQL and setting password

Once the database is active, the mysql command is used to connect to the MySQL shell for database interaction. By default, the root user is loaded, and root access does not require a password initially, which is a security risk. To fix this, the root password is updated using the “**UPDATE**” command and the new password is set to “**Group25**” as seen in Figure 17.

```
[root@comp1475-centos7-25 ~]# mysql
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 5.5.65-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> UPDATE mysql.user SET Password=PASSWORD('Group25') WHERE User='root';
Query OK, 4 rows affected (0.00 sec)
Rows matched: 4 Changed: 4 Warnings: 0
```

Figure 17 Set password for MariaDB

After updating the password, the database can now be accessed using the new password, as shown in Figure 18 The password will not be displayed on the screen during login. This ensures the database is secured with a password and cannot be accessed without proper authentication.

```
[root@comp1475-centos7-25 ~]# mysql -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
```

Figure 18 Login into MariaDB

V. Setting Up a MariaDB User, Database, and Table

The following commands show how to create a user, database, table, insert data, and grant access. These steps are essential for organizing, securing, and managing data efficiently, as seen in the figure 19.

- Create User: The command **Create user 'mariusz'@'localhost' identified by 'password'**; creates a new MySQL user and verifies their identity. The username 'mariusz'@'localhost' allows access only from the local machine, blocking remote connections. Users log in with a unique username and password, which helps prevent unauthorized access. Passwords are stored in an encrypted format, making them more secure since they cannot be read in plain text.
- Create Database: The command **CREATE DATABASE lamp**; creates a new database called lamp, which helps organize and manage data efficiently, improving security and performance. The command **USE lamp**; makes lamp the active database, so users can easily run queries and manage data without needing to specify the database every time.
- Create Table & insert Values: The command **CREATE TABLE person(Name VARCHAR(64), Age INT, address VARCHAR(128))**; creates a table to store data in an organized way, making it easier to manage and retrieve. The command **INSERT INTO person(Name, Age, address) VALUES ('Mariusz Pelc', 15, 'Poland')**; adds a new record for future use.
- Grant privileges/access: The command **GRANT SELECT ON person TO 'mariusz'@'localhost'**; gives the user 'mariusz' permission to view the data from the "person" table but not modify it. This helps control user privileges and secure the database.

```
MariaDB [(none)]> create user 'mariusz'@'localhost' identified by 'password';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> create database lamp;
Query OK, 1 row affected (0.01 sec)

MariaDB [(none)]> use lamp
Database changed
MariaDB [lamp]> create table person(Name Varchar(64),Age int, address varchar(128));
Query OK, 0 rows affected (0.00 sec)

MariaDB [lamp]> insert into person(Name,Age,address) values('Mariusz Pelc',15,'Poland');
Query OK, 1 row affected (0.00 sec)

MariaDB [lamp]> grant select on person to 'mariusz'@'localhost';
Query OK, 0 rows affected (0.00 sec)

MariaDB [lamp]> exit
Bye
```

Figure 19 Creating user, Database, table

```

MariaDB [lamp]> select * from person;
+-----+-----+-----+
| Name      | Age | address |
+-----+-----+-----+
| Mariusz Pelc | 15 | Poland |
+-----+-----+-----+
1 row in set (0.00 sec)

```

Figure 20 Displaying the table values

The SQL command **SELECT * FROM person;** retrieves all the data from the person table in the database.

2.3 Analysis

Setting up MariaDB helps keep data safe and easy to use. Installing the client and server lets the system store and retrieve information properly. Starting the service turns the database on so it's ready for work. Checking settings makes sure everything runs smoothly and helps fix problems faster. Setting strict file permissions locks important data away from unwanted access. Changing the root password adds extra security. Creating users, databases, and tables keeps information organized. Giving users specific permissions controls who can see or change data. Each step helps keep the database safe, fast, and reliable.

3.Firewall Configuration

3.1 Overview

In this lab, we will focus on managing firewall settings to ensure network security. We will configure **firewall-cmd** and **iptables** to control which services and ports are open, block unauthorized traffic, and improve system performance. These steps help protect the network from potential threats while maintaining its functionality and efficiency.

3.2 Implementation Steps

I. Checking if the Firewall is Running

The command **“systemctl status firewalld”** check the firewall daemon status., if the firewall status is disabled then **“systemctl start firewalld”** command will enable the firewall.

```

[root@comp1475-centos7-25 ~]# systemctl status firewalld
■ firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2025-02-04 19:03:00 GMT; 2 weeks 0 days ago
     Docs: man:firewalld(1)

```

Figure 21 Firewall Daemon Status

II. Checking Current Firewall Configuration

Managing firewall settings is important for keeping a network secure, as it makes sure that only the necessary services and ports are open, while blocking unwanted traffic. This helps protect computers and networks from hackers and unauthorized access.

- `iptables -L`

The **iptables -L** command lists the current firewall rules that control how network traffic is allowed or denied. Iptables works by filtering data packets based on rules, such as which IP addresses or ports are allowed to send or receive data. The output of `iptables -L`, as seen in Figure 22, shows different chains like `INPUT_ZONES`, `IN_public`, and `IN_public_allow`, which help control how network traffic is managed. For example, the rule **ACCEPT tcp dpt:ssh ctstate NEW, UNTRACKED** allows SSH connections, while **ACCEPT tcp dpt:http ctstate NEW, UNTRACKED** allows HTTP traffic. Other chains, such as `IN_public_deny` and `IN_public_log`, are responsible for blocking and logging traffic to make sure that any unwanted or suspicious connections are either monitored or denied. This helps keep the network secure by only allowing necessary traffic while preventing unauthorized access.

```
target      prot opt source                destination
Chain FWD0_public_deny (1 references)
target      prot opt source                destination
Chain FWD0_public_log (1 references)
target      prot opt source                destination
Chain INPUT_ZONES (1 references)
target      prot opt source                destination
IN_public   all  --  anywhere              anywhere    [goto]
IN_public   all  --  anywhere              anywhere    [goto]
Chain INPUT_direct (1 references)
target      prot opt source                destination
Chain IN_public (2 references)
target      prot opt source                destination
IN_public_log all  --  anywhere              anywhere
IN_public_deny all  --  anywhere              anywhere
IN_public_allow all  --  anywhere              anywhere
ACCEPT      icmp --  anywhere              anywhere
Chain IN_public_allow (1 references)
target      prot opt source                destination
ACCEPT      tcp  --  anywhere              anywhere    tcp dpt:ssh ctstate NEW,UNTRACKED
ACCEPT      tcp  --  anywhere              anywhere    tcp dpt:http ctstate NEW,UNTRACKED
Chain IN_public_deny (1 references)
target      prot opt source                destination
Chain IN_public_log (1 references)
target      prot opt source                destination
Chain OUTPUT_direct (1 references)
target      prot opt source                destination
[root@comp1475-centos7-25 ~]#
```

Figure 22 Firewall rules

- firewall-cmd --list-all

The command **firewall-cmd --list-all** displays all active firewall rules, including zones, services, and allowed or denied ports. It allows HTTP (web traffic), SSH (remote access), and DHCPv6 (automatic IP assignment) while blocking other services. This improves security, controls network access, and makes firewall management easier by only allowing necessary connections.

```
[root@comp1475-centos7-25 /]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: ens192
  sources:
  services: dhcpv6-client http ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Figure 23 Firewall Rule

III. Adding Services

The commands **firewall-cmd --add-service=http** and **firewall-cmd --add-service=ftp** allows web traffic (HTTP) and file transfers (FTP) through the firewall. This helps in hosting websites, sharing files, and ensuring secure and controlled network access. It also simplifies firewall configuration and improves connectivity between servers and users. By only allowing necessary services, it enhances security and prevents unauthorized access while keeping network management simple and effective. After updating the rules, the **firewall-cmd --list-all** command will display these changes, as shown in Figure 25.

```
[root@comp1475-centos7-25 ~]# firewall-cmd --add-service=http
success
[root@comp1475-centos7-25 ~]# firewall-cmd --add-service=ftp
success
```

Figure 24 Adding Firewall rule

```
[root@comp1475-centos7-25 /]# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: ens192
  sources:
  services: dhcpv6-client ftp http ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Figure 25 Rule updated in firewall

Another way to write firewall rules is through iptables commands, like **iptables -A INPUT -p tcp --dport 22 -j ACCEPT** for SSH, **iptables -A INPUT -p tcp --dport 80 -j ACCEPT** for HTTP, and **iptables -A INPUT -p tcp --dport 21 -j ACCEPT** for FTP. iptables gives more detailed control over traffic, making it better for advanced setups. Unlike firewall-cmd, which simplifies things, iptables requires manual configuration but offers more flexibility.

```
[root@comp1475-centos7-25 ~]# systemctl stop firewalld
[root@comp1475-centos7-25 ~]# iptables -A INPUT -p tcp --dport 22 -j ACCEPT
[root@comp1475-centos7-25 ~]# iptables -A INPUT -p tcp --dport 80 -j ACCEPT
[root@comp1475-centos7-25 ~]# iptables -A INPUT -p tcp --dport 21 -j ACCEPT
[root@comp1475-centos7-25 ~]# iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Figure 26 Adding firewall rule using iptables command

3.3 Analysis

Managing firewall settings is crucial for network security. Checking if the firewall is running ensures it's actively protecting the system. By using **firewall-cmd** or **iptables**, we control which services and ports are open, blocking unnecessary traffic and reducing the risk of unauthorized access. Allowing only essential services, like HTTP or FTP, enhances security while maintaining system functionality. **firewall-cmd** simplifies the process with predefined services, while **iptables** offers granular control, making it suitable for more advanced setups. Both methods protect the system from potential threats, improving performance by preventing harmful or unnecessary traffic from accessing the network.

Conclusion

In this lab, we successfully set up and configured Apache HTTPD, MariaDB, and firewall rules, each contributing to the overall security and functionality of the system. Installing Apache HTTPD and MariaDB ensured the proper hosting of web services and secure database management, while configuring firewall settings using **firewall-cmd** and **iptables** protected the network by controlling access to critical services. By only allowing essential traffic, the system's vulnerability to unauthorized access was minimized, ensuring both performance and security. It reinforced the importance of careful configuration, monitoring, and proactive security measures in maintaining a secure environment.

PART 2 : HTTPD

Literature Review

Apache HTTP Server, also known as Apache httpd, is a popular open-source web server. It is used to host websites, support static and dynamic content, and act as a reverse proxy. Apache is known for being flexible, secure, and modular, making it a popular choice in both enterprise and shared hosting environments. According to data from the Apache Software Foundation's 2009 report, Apache powers over 106 million websites worldwide, and in the 2009 Reader's Choice Awards, 89% of Linux Journal readers chose it as their favourite web server.

One big advantage of Apache is its modular design, which lets users add or remove features as need. Apache includes Multi-Processing Modules (MPMs) that help it handle multiple requests more efficiently. For example, the Event MPM, which improves performance by managing connections better. Apache also supports dynamic modules like `mod_rewrite` for URL redirection and `mod_ssl` for encryption. It also provides strong authentication features and better load balancing.

Security is a major concern when running a web server, and Apache has several built-in security features. It supports TLS encryption and works with `mod_security` to protect against online threats. However, if Apache is not configured properly, it can become vulnerable to attacks. Common issues include incorrect file permissions, weak login settings, and exposed directories. To keep Apache secure, it's important to disable unused modules, use `.htaccess` files to limit access, set up firewall rules, and regularly update the software to fix security problems.

Apache performs well, especially when handling dynamic content with modules like `mod_php`. It also supports `.htaccess` files, which allow users to make changes to server settings. This makes Apache a good option for shared hosting and older websites.

Overall, Apache is a reliable and powerful web server, but it needs proper setup and maintenance to run smoothly. By following good security practices and using the right modules, users can take full advantage of Apache's features.

Vulnerability Analysis

1. **Lack of SSL/TLS Encryption**

Based on Apache's default configuration the server is currently running traffic over HTTP (port 80) instead of HTTPS (port 443) and does not have SSL/TLS enabled. SSL/TLS is required for encrypting communication between clients and servers. Without it, sensitive information, such as login credentials, financial data, and personal details are transmitted in plaintext, making it easily accessible to attackers performing Man-in-the-Middle (MITM) attacks. The vulnerability can be discovered using tools like Nessus, OpenVAS, or manual methods such as curl to detect missing or misconfigured SSL/TLS settings. Attackers can also use packet sniffers like Wireshark to intercept and manipulate this unencrypted traffic. The severity level of this vulnerability is high as this attack can lead to data theft, unauthorized access, loss of confidentiality, and potential compliance violations.

Mitigation Steps:

To mitigate this vulnerability following steps can be followed:

- SSL/TLS should be enabled on Apache by installing and configuring the mod_ssl module and having a valid certificate from a trusted Certificate Authority.
- Configuring Apache to automatically redirect all HTTP traffic to HTTPS. This ensures that all communication is encrypted, even if users initially try to access the site over HTTP. This can be done in Apache's .htaccess or httpd.conf file.
- Also disabling port 80 to prevent any unencrypted HTTP traffic from being served. This will force all communication to take place over HTTPS, ensuring encryption for all data.

2. **Weak Firewall Configuration**

The default firewall configuration accepts all incoming traffic, which can lead to critical services like HTTP/HTTPS (ports 80/443) being exposed to attacks. This makes the server vulnerable to various threats, including unauthorized access, denial of service, and exploitation of unpatched vulnerabilities in web services. Attackers can use these open ports to search for weaknesses in services like web servers, taking advantage of outdated or poorly configured HTTP or HTTPS settings. These open ports might also allow unauthorized access to sensitive systems and databases. Attackers could scan the open ports, exploit known vulnerabilities in the services this can lead to unauthorized access, data breaches, denial of service, exploitation of weaknesses, and potential system compromise. Due to which the severity level of the vulnerability is high.

Mitigation Steps:

- Disable or block unnecessary open ports to reduce the attack surface and limit access to essential services only.
- Configuring the firewall settings to allow only necessary traffic, by restricting incoming requests to critical services like HTTP/HTTPS to trusted IP addresses.
- Implementing WAF to filter and monitor HTTP traffic, blocking malicious requests and protecting against common web application attacks.

3. Information Disclosure

Apache's default configuration reveals detailed version information of the server in response headers. This includes version numbers of Apache itself, as well as installed modules and other sensitive server information. Attackers can use this information to target known vulnerabilities associated with the version. For example, CVE-2014-0226 and CVE-2014-0118 refer to vulnerabilities in Apache HTTP Server versions prior to 2.4.10. The severity level of this vulnerability is medium as this can lead to server being compromised, gaining unauthorized access etc

Mitigation Steps:

This vulnerability can be mitigated by implementing the following steps:

- Modifying the Apache configuration file httpd.conf to prevent Apache from revealing version information in HTTP response headers by using ServerTokens and ServerSignature.
- Ensuring that Apache is regularly updated to the latest version with security patches to protect it against known vulnerabilities.

4. Lack of Security Headers

Apache's default configuration does not include important HTTP security headers in its HTTP response configuration. These headers are important for mitigating various web vulnerabilities such as cross-site scripting (XSS), clickjacking, and content sniffing. The absence of headers like X-Content-Type-Options, X-XSS-Protection, Content-Security-Policy (CSP), and Strict-Transport-Security (HSTS) makes the server vulnerable to attacks. This misconfiguration can be easily discovered and exploited by using tools like Nessus or OpenVAS. Without these headers, the web server is vulnerable to attacks that could compromise the security, privacy, and availability of the web page making the severity level as medium.

Mitigation Steps:

This vulnerability can be mitigated by implementing the following steps:

- Configuring Apache to include essential HTTP security headers, such as X-Content-Type-Options, X-XSS-Protection, Content-Security-Policy (CSP), and Strict-Transport-Security (HSTS), to protect against XSS, clickjacking, and content sniffing attacks

5. Securing Web content Files

This vulnerability occurs when sensitive files on a web server, like configuration files (e.g., .htaccess, .env), logs, or user-uploaded files, are exposed and can be accessed by unauthorized users. This can lead to data breaches and attackers can obtain sensitive information like database credentials, API keys, or password. Attackers can exploit this vulnerability by manually inspecting the server, using tools like Nessus, Burp Suite, or simply by scanning for exposed files through URL enumeration. This occurs due to incorrect file permissions or misconfiguration of server settings. The severity level of this vulnerability is medium as this can lead to data breaches, application compromise, loss of sensitive information and legal consequences.

Mitigation Steps:

- Ensure sensitive files are not publicly accessible by setting restrictive file permissions and access controls.
- Enable and configure SELinux to enforce security policies that prevent unauthorized access to sensitive files and processes. SELinux can enforce strict access controls on files and limit the damage caused by potential attacks.

Risk Assessment

1. Lack of SSL/TLS Encryption

Risk Impact

The lack of SSL/TLS encryption means that data transmitted between users and the website is not secured, making it vulnerable to interception and attacks. The most common type of attack is Man-in-the-Middle (MITM) attack, where the attacker intercepts communication between a client and a server, where they can steal or alter sensitive data. Another type of attack is Session hijacking where the attacker can take control of an active user session and try to gain unauthorized access to a person's account and sensitive information. This can lead to data breaches, stealing of user credentials and payment details, or other confidential information. This puts the company's goals of protecting customer trust, securing

data at risk and can lead to compliance issue with PCI-DSS, GDPR, ISO 27001, and NIST.

Risk Level

The risk level of this vulnerability is high due to the lack of SSL/TLS as it increases the risk of data breaches, unauthorized access, and compliance violations.

2. Weak Firewall Configuration

Risk Impact

Having default firewall configuration exposes the systems to multiple attacks. If firewall rules are not configured correctly, or if they do not filter the right kinds of traffic, it can expose the network to various attacks like port scanning, Denial of Service (DoS), brute force attacks, SQL injection, Man-in-the-Middle (MITM) attacks, and Remote Code Execution (RCE). This could lead to it can lead to data breaches, financial loss, operational disruptions, such as system downtime or loss of services. It can also lead to PCI DSS compliance violations leading to legal penalties , fines and damage to the company's reputation.

Risk Level

This is a high risk because it makes the system vulnerable to many types of attacks and threats, like unauthorized access, data breaches, denial of service attacks, exploiting services, or scanning the network.

3. Information Disclosure

Risk Impact

In Apache's default configuration details such as version information of the server is revealed in the response headers. The most common risk is that the attackers can exploit known vulnerabilities targeting the exposed information. For example, CVE-2014-0226 and CVE-2014-0118 refer to vulnerabilities in Apache HTTP Server versions prior to 2.4.10. This can lead to targeted attacks, unauthorized access, operational disruption, and data breaches.

Risk Level

The risk level of this vulnerability is medium as it increases the risk of an attacker using known exploits related to the version and modules.

4. Lack of Security Headers

Risk Impact

The absence of security headers increases vulnerability to web applications by exposing them to various attacks like XSS, clickjacking, content sniffing, MITM attacks, and data theft. This puts user privacy at risk and allows hackers to exploit sensitive information. It can also lead to non-compliance issues with regulations like PCI-DSS, GDPR, and ISO 27001 leading to legal and financial consequences.

Risk Level

The risk level of this vulnerability is Medium as it leaves the application exposed to different types of attacks, including data manipulation, unauthorized access, and content injection. These attacks can compromise both the security of users and the integrity of the web application, causing significant business and reputational damage

5. Securing Web content Files

Risk Impact

This vulnerability occurs when sensitive files on a web server are not properly secured and can be accessed by unauthorized users. If attackers gain access to these files, they can steal confidential data, change the website's content, or even execution of malicious code on the server. This can lead to consequences, such as data loss, which may harm business operations, damage the organization's reputation, and lead to legal issues due to non-compliance with data protection regulations and potential customer trust loss.

Risk Level

The risk level is medium because while unauthorized access to sensitive files can lead to significant damage to systems security.

Hardening Procedure

1. Installing TLS/SSL

SSL/TLS (Secure Sockets Layer/Transport Layer Security) is essential for encrypting data transmitted between clients and servers. Enabling SSL/TLS, ensures that sensitive data such as passwords, personal information, and payment details are securely transmitted.

Step1: Installing and Configuring SSL Certificate

SSL certificate verifies the authenticity of the website, proving that the server is legitimate, and the connection is secure. The certificate also enables encryption, protecting data in transit. The command “ sudo certbot –apache” is used to install the open-source certificate from certbot.

Step2: Configuring SSL/TLS

To handle HTTPS traffic the SSL/TLS certificate details are configured in httpd.conf file. The following directives should be added to enable SSL/TLS:

- i. **SSLEngine on** enables SSL for the server, ensuring encrypted connections.
- ii. **SSLCertificateFile** points to the SSL certificate file.
- iii. **SSLCertificateKeyFile** specifies the private key associated with the SSL certificate.

Step3: Redirect All HTTP Traffic to HTTPS

By redirecting all HTTP traffic to HTTPS ensures that all communications are encrypted, even if users try to access the site using the non-secure HTTP protocol. This can be done using the `mod_rewrite` module in Apache, which forces HTTP connections to upgrade to HTTPS. The following directives are written in `httpd.conf` file:

- i. **RewriteEngine On** which enables the `mod_rewrite` to process the rules.
- ii. **RewriteCond %{HTTPS} off** checks if the connection is not using HTTPS.
- iii. **RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]** redirects the user to the HTTPS version of the page.

This reduces the risk of man-in-the-middle (MITM) attacks.

Step4: Disable Port 80 (HTTP)

Disabling port 80 prevents any unencrypted HTTP traffic from reaching the server. Port 80 is used for HTTP traffic, which is unencrypted. Allowing traffic on this port exposes the server to risks such as man-in-the-middle (MITM) attacks, where attackers can intercept and manipulate unencrypted data. This can be done via the `httpd.conf` file as well as firewall. The following firewall command can be executed for blocking port 80 (HTTP) and accepting HTTPS port 443:

```
iptables -A INPUT -p tcp --dport 80 -j REJECT  
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

2. Enhancing Firewall Configuration

Firewall is essential for securing a server by controlling incoming and outgoing network traffic. It helps protect the server by blocking unwanted traffic and only allowing web traffic, which reduces the chances of unauthorized access. The firewall also limits the risk of attacks like DDoS, brute-force, and port scanning. It also keeps track of suspicious activity through logs, allowing quick action if needed.

Step1: Add HTTPS Service to the Firewall

The command `firewall-cmd --permanent --add-service=https` allows only HTTPS traffic to the server. This helps keep the server safer by blocking other services and reducing unauthorized access. It makes sure that only necessary traffic is allowed. .

```
[root@comp1475-centos7-25 ~]# firewall-cmd --permanent --add-service=https  
success
```

Figure 27 Add HTTPS service

Step2: Enable Logging and Limit HTTPS Connections

The following command adds a rule to the firewall to log HTTP traffic with the prefix "HTTP_REQUEST" and restricts the number of HTTP connections to 10 requests per minute from the same IP: **firewall-cmd --permanent --add-rich-rule='rule family="ipv4" service name="https" log prefix="HTTPS Dropped " level="info" limit value="10/m" accept'**. Limiting the number of requests from a single IP can help prevent attacks like web flooding attacks, scraping or repeated login attempts before they impact server performance or security. It also helps the server run smoothly by not letting it get overloaded with too much traffic, so real users can access it without problems. These setting help keep the server safe, stable, and running well.

```
[root@comp1475-centos7-25 ~]# firewall-cmd --permanent --add-rich-rule='rule family="ipv4" service name="https" log prefix="HTTPS Dropped: " level="info" limit value="10/m" accept'
success
```

Figure 28 Firewall rule

3. Preventing Information Disclosure

When securing a web server like Apache, one of the important steps is to hide details about the server, such as its version number and which modules are installed. This is important because if attackers can see this information, they might try to exploit known weaknesses in that version of Apache or in a specific module. Reducing the apache information makes it more difficult for attackers to exploit known vulnerabilities, thereby strengthening the server's overall security posture.

Step1:Check Default Server response

First, we used the `curl -I http://localhost` command to check the response headers sent by the Apache server. Since Apache was using its default configuration, the response headers included detailed information such as the Apache version, the operating system, and the modules enabled as seen in figure 29.

```
[root@comp1475-centos7-25 ~]# curl -I http://localhost
HTTP/1.1 200 OK
Date: Thu, 06 Mar 2025 16:24:11 GMT
Server: Apache/2.4.6 (CentOS) PHP/5.4.16
Last-Modified: Wed, 19 Feb 2025 16:41:55 GMT
ETag: "35-62e816f764192"
Accept-Ranges: bytes
Content-Length: 53
Content-Type: text/html; charset=UTF-8
```

Figure 29 Default Apache header response

Step2: Disable the Server Details

All configuration changes related to Apache can be made in the httpd.conf file. To enhance security, we modified this file by adding the following directives:

- **ServerTokens Prod:** This setting ensures that the Server header in HTTP responses only displays "Apache" without revealing the version number, operating system, or loaded modules. This prevents attackers from easily identifying specific vulnerabilities associated with a particular Apache version.
- **ServerSignature Off:** This setting disables the display of server information on Apache-generated error pages (such as 404 Not Found or 500 Internal Server Error). Without this setting, error pages might expose details like the Apache version and operating system, which could be exploited by attackers.

```
#Disabel Server Signature and Version Discolsure
ServerTokens Prod
ServerSignature Off
```

Figure 30 config changes in httpd.conf file

Step 3: Restarting Apache

After saving the changes in httpd.conf, we need to restarted Apache (httpd) to apply the new configuration settings. This ensures that the new security changes take effect.

```
[root@comp1475-centos7-25 /]# systemctl restart httpd
```

Figure 31 Restart Apache

Step 4: Verifying the Security Changes

After making the changes, and checking the response headers again, the headers no longer showed the Apache version details. This confirms that the security changes were successfully implemented. The updated response headers, as seen in Figure 32, now show only minimal server information, making the server safer from attacks. By removing the Apache version and module information, it becomes harder for attackers to find weaknesses in the server. This improves the security of the Apache server.

```
[root@comp1475-centos7-25 /]# curl -I http://localhost
HTTP/1.1 200 OK
Date: Thu, 06 Mar 2025 16:33:52 GMT
Server: Apache
Last-Modified: Wed, 19 Feb 2025 16:41:55 GMT
ETag: "35-62e816f764192"
Accept-Ranges: bytes
Content-Length: 53
Content-Type: text/html; charset=UTF-8
```

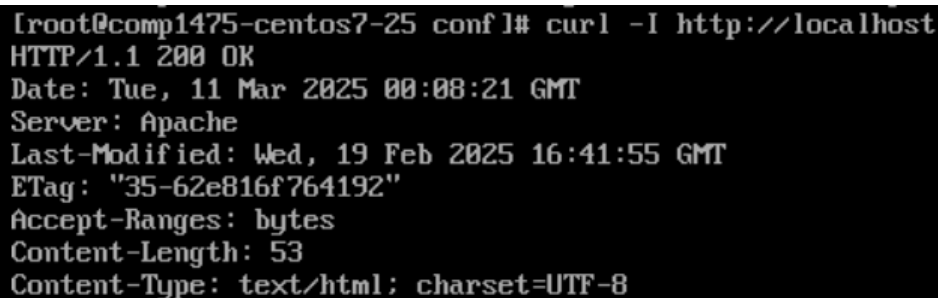
Figure 32 Apache header response after changes

4. Enhancing Web Security with HTTP Headers

The default apache configuration does not have security headers configured in HTTP response which makes them vulnerable to MITM attacks, XSS, clickjacking, and MIME-sniffing allowing attackers to steal data or inject malicious scripts. Configuring HTTP Security Headers or Response Security Header will help to protect the web applications from various attacks by instructing the browser on how to handle specific security-related behaviours.

Step1: Check Default Server response

First, we used the `curl -I http://localhost` command to check the response headers sent by the Apache server. Since Apache was using its default configuration, the response headers do not have any HTTP Security Headers as seen in Figure 33.



```
[root@comp1475-centos7-25 conf]# curl -I http://localhost
HTTP/1.1 200 OK
Date: Tue, 11 Mar 2025 00:08:21 GMT
Server: Apache
Last-Modified: Wed, 19 Feb 2025 16:41:55 GMT
ETag: "35-62e816f764192"
Accept-Ranges: bytes
Content-Length: 53
Content-Type: text/html; charset=UTF-8
```

Figure 33 Apache response without HTTP Security headers

Step2: Adding Security Headers

To make the web server more secure, we added a few important security headers in the `httpd.conf` file. These headers help protect against different types of cyberattacks.

1. X-Content-Type-Options

Directive: Header always set X-Content-Type-Options "nosniff"

This prevents the browser from sniffing the content type of files, ensuring that they are interpreted according to the Content-Type header. It helps block MIME-sniffing attacks, reducing the risk of unauthorized content execution.

2. X-Frame-Options

Directive: Header always set X-Frame-Options "SAMEORIGIN"

This prevents the website from being displayed inside an iframe on another site. It helps stop clickjacking attacks, where hackers try to trick users into clicking on hidden elements like buttons without realizing it, potentially leading to unauthorized actions.

3. Content-Security-Policy (CSP)

Directive: Header always set Content-Security-Policy "default-src 'self';"

This controls where content like scripts, images, and styles can be loaded from. By restricting it to trusted sources, it helps prevent Cross-Site Scripting

(XSS) and content injection attacks, where attackers try to insert harmful code into the webpage.

4. X-XSS-Protection

Directive: Header always set X-XSS-Protection "1; mode=block"

This enables the browser's built-in protection against XSS attacks. If a harmful script is detected, the browser automatically blocks it before it can do any damage.

```
#Security Headers

Header always set X-Content-Type-Options "nosniff"
Header always set X-Frame-Options "SAMEORIGIN"
Header always set Content-Security-Policy "default-src 'self';"
Header always set X-XSS-Protection "1; mode=block"
```

Figure 34 Updating Config File

Step 3: Restarting Apache

After saving the changes in httpd.conf, we need to restart Apache (httpd) to apply the new configuration settings. This ensures that the new security changes take effect.

```
[root@comp1475-centos7-25 /]# systemctl restart httpd
```

Figure 35 Restart Apache

Step 4: Verifying the Security Changes

After implementing the changes and verifying the response headers once again the security headers are successfully included in the response. These headers add an extra layer of protection, making the web server much harder to exploit. They help keep users safe from common vulnerabilities, reduce the risk of attacks and ensure that only trusted content is loaded. By enforcing these security measures, unauthorized access and data manipulation are reduced, improving the overall security of the application and the users.

```
[root@comp1475-centos7-25 conf]# curl -I http://localhost
HTTP/1.1 200 OK
Date: Tue, 11 Mar 2025 00:09:46 GMT
Server: Apache
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Content-Security-Policy: default-src 'self';
X-XSS-Protection: 1; mode=block
Last-Modified: Wed, 19 Feb 2025 16:41:55 GMT
ETag: "35-62e816f764192"
Accept-Ranges: bytes
Content-Length: 53
Content-Type: text/html; charset=UTF-8
```

Figure 36 HTTP response headers

5. Securing Files with SELinux Context

Files stored in the web directory `/var/www/html/` other than web content files should be secured in case the web server is compromised. Access to these sensitive files is restricted using SELinux, preventing attackers from reading confidential files, even if they gain access to the web server. This makes the system more secure by isolating important data from the web service. It also prevents unauthorized file access, privilege escalation, and information disclosure by limiting access to sensitive files, even if the web server is compromised.

Step 1: Create a File and Check its SELinux Security Context

In the `/var/www/html/` directory, a file named `credential.txt` is created to store a username and password. Running the `ls -lZ` command displays the file's details, file permission and its SELinux security context. By default, the file's context is set to `httpd_sys_content_t`, which allows the web server to access and serve it. This context is typically used for files that need to be publicly available to the web server. If the `httpd` service were compromised, an attacker could access any file with the `httpd_sys_content_t` context, causing sensitive data being exposed

```
[root@comp1475-centos7-25 ~]# cd /var/www/html
[root@comp1475-centos7-25 html]# ls
cred.html  index.html
[root@comp1475-centos7-25 html]# echo " Admin credentials -> Username: root, password:toor" > confidential.txt
[root@comp1475-centos7-25 html]# ls
confidential.txt cred.html index.html
[root@comp1475-centos7-25 html]# ls -lZ
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 confidential.txt
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 cred.html
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 index.html
```

Figure 37 creating a txt file and checking its status

Step 2: Verify File Accessibility in a Web Browser

Upon verifying the `credential.txt` file in the web browser using the `lynx` command, we were able to see the file contents, as shown in Figure 39.

```
[root@comp1475-centos7-25 html]# lynx http://localhost/confidential.txt_
```

Figure 38 Accessing the txt file via http

```
Admin credentials -> Username: root, password:toor
```

Figure 39 TXT file details visible

Step 3: Change the SELinux Context to Restrict Access

To secure sensitive data, the SELinux context of the confidential.txt file is changed to admin_home_t using the following command: **chcon -t admin_home_t /var/www/html/confidential.txt**. The **chcon** command is used to change the SELinux security context of a file or folder. The -t option specifically sets the type of the security context. By assigning the admin_home_t type to the file, it ensures that only administrative users can access it, while blocking the web server from accessing the file. This is an important step in securing sensitive data, as it prevents the file from being publicly accessible, even if the web server is compromised. As shown in Figure 41, when attempting to access the file, a 403 Forbidden error is received, indicating that access to the file is denied due to the restricted permissions set by the updated SELinux context.

```
[root@comp1475-centos7-25 html]# chcon -t admin_home_t confidential.txt
[root@comp1475-centos7-25 html]# ls -lZ
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 confidential.txt
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 cred.html
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 index.html
[root@comp1475-centos7-25 html]# _
```

Figure 40 Adding security measures

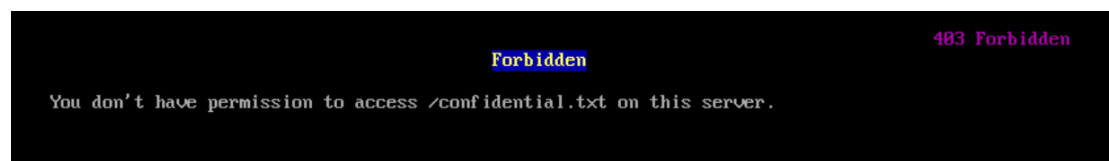


Figure 41 Access denied to File

Step 4: Checking log Files

Reviewing server logs is a critical security practice that helps detect potential threats and misconfigurations.

- Access Logs help identify unauthorized access attempts and unusual traffic patterns as seen in figure 42
- Error Logs reveal denied requests and misconfigurations that could indicate tampering, or privilege escalation attempts as seen in Figure 43.
- Audit Logs, which record SELinux security decisions, are essential for investigating access denials. They log details like timestamps, process IDs, the exact reasons a file or process was blocked etc as seen in Figure 44.

Regular log analysis helps identify attack patterns, detect brute-force attempts, and ensure security policies are working as intended. This allows administrators to proactively respond to security threats before they escalate.

```

[root@comp1475-centos7-25 html]# less /var/log/httpd/access_log
::1 - - [10/Mar/2025:12:11:12 +0000] "HEAD / HTTP/1.1" 200 - "-" "curl/7.29.0"
::1 - - [10/Mar/2025:14:28:19 +0000] "HEAD / HTTP/1.1" 200 - "-" "curl/7.29.0"
::1 - - [10/Mar/2025:15:03:21 +0000] "HEAD / HTTP/1.1" 200 - "-" "curl/7.29.0"
::1 - - [10/Mar/2025:15:09:28 +0000] "HEAD / HTTP/1.1" 200 - "-" "curl/7.29.0"
::1 - - [10/Mar/2025:15:33:28 +0000] "GET /confidential.txt HTTP/1.0" 200 52 "-" "Lynx/2.8.8dev.15 libwww-FM/2.14 SSL-MM/1.4.1 OpenSSL/1.0.1e-fips"
::1 - - [10/Mar/2025:15:35:19 +0000] "GET /confidential.txt HTTP/1.0" 403 218 "-" "Lynx/2.8.8dev.15 libwww-FM/2.14 SSL-MM/1.4.1 OpenSSL/1.0.1e-fips"
/var/log/httpd/access_log (END)

```

Figure 42 Access logs

```

[Mon Mar 10 15:35:19.298401 2025] [core:error] [pid 1451] (13)Permission denied: [client ::1:41956]
AH00035: access to /confidential.txt denied (filesystem path '/var/www/html/confidential.txt') becau
se search permissions are missing on a component of the path

```

Figure 43 Error logs

```

time->Mon Mar 10 15:35:19 2025
type=PROCTITLE msg=audit(1741620919.297:134): proctitle=2F7573722F7362696E2F6874747064002D44464F5245
47524F554E44
type=SYSCALL msg=audit(1741620919.297:134): arch=c000003e syscall=4 success=no exit=-13 a0=56267c9d2
928 a1=7fffa7f4c270 a2=7fffa7f4c270 a3=7fa6bb76d772 items=0 ppid=1449 pid=1451 auid=4294967295 uid=4
8 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48 fsgid=48 tty=(none) ses=4294967295 comm="httpd" ex
e="/usr/sbin/httpd" subj=system_u:system_r:httpd_t:s0 key=(null)
type=AVC msg=audit(1741620919.297:134): avc: denied { getattr } for pid=1451 comm="httpd" path="/
var/www/html/confidential.txt" dev="dm-0" ino=33607700 scontext=system_u:system_r:httpd_t:s0 tcontex
t=unconfined_u:object_r:admin_home_t:s0 tclass=file permissive=0
time->Mon Mar 10 15:35:19 2025
type=PROCTITLE msg=audit(1741620919.297:135): proctitle=2F7573722F7362696E2F6874747064002D44464F5245
47524F554E44
type=SYSCALL msg=audit(1741620919.297:135): arch=c000003e syscall=6 success=no exit=-13 a0=56267c9d2
a10 a1=7fffa7f4c270 a2=7fffa7f4c270 a3=fffffe0 items=0 ppid=1449 pid=1451 auid=4294967295 uid=48 gi
d=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48 fsgid=48 tty=(none) ses=4294967295 comm="httpd" exe="/
usr/sbin/httpd" subj=system_u:system_r:httpd_t:s0 key=(null)
type=AVC msg=audit(1741620919.297:135): avc: denied { getattr } for pid=1451 comm="httpd" path="/
var/www/html/confidential.txt" dev="dm-0" ino=33607700 scontext=system_u:system_r:httpd_t:s0 tcontex
t=unconfined_u:object_r:admin_home_t:s0 tclass=file permissive=0

```

Figure 44 Audit Logs

Conclusion

The default Apache server configuration had several vulnerabilities, such as lack of SSL/TLS encryption, risk to DoS/DDoS attacks, weak firewall settings, information disclosure, missing security headers, and insecure web content file permissions. The server's overall security was enhanced by the hardening procedures, which decreased these vulnerabilities. By enabling SSL/TLS encryption, securing the web content files, and implementing security headers, the system is now protected against attacks such as data interception, unauthorized access, and web-based exploits. The improved firewall configuration and protection against DoS/DDoS attacks help reduce the chances of service disruptions. As a result, the server is more secure from threats, keeping the web services safe and available. These improvements align the server with industry security standards, providing a robust defence against evolving cyber threats. The result is a safer and more reliable Apache setup, reducing risks and improving efficiency.

References

Apache (2019). *Security Tips - Apache HTTP Server Version 2.4*. [online] Apache.org. Available at: https://httpd.apache.org/docs/2.4/misc/security_tips.html.

Nist.gov. (2015). *NVD - CVE-2014-0226*. [online] Available at: <https://nvd.nist.gov/vuln/detail/CVE-2014-0226>

Nist.gov. (2015). *NVD - CVE-2014-0118*. [online] Available at: <https://nvd.nist.gov/vuln/detail/CVE-2014-0118>

Sally (2009). *The Apache HTTP Server Project Does It Again - The World's #1 Web Server Tops Linux Journal's Reader's Choice Awards - The Apache Software Foundation Blog*. [online] The Apache Software Foundation Blog. Available at: https://news.apache.org/foundation/entry/the_apache_http_server_project

Yoachimik, O. and Pacheco, J. (2025). *Record-breaking 5.6 Tbps DDoS attack and global DDoS trends for 2024 Q4*. [online] The Cloudflare Blog. Available at: <https://blog.cloudflare.com/ddos-threat-report-for-2024-q4/>.