420-635-AB -NETWORK INSTALLATION AND ADMINISTRATION I

PROJECT 1
Report : Apache Web Server Configuration and Security

PREPARED BY: GUILLERMO PADILLA KEYMOLE

John Abbott College | Network Administration Program

Table of Contents

Introduction

The purpose of this project is to set up and configure a secure Apache web server on AlmaLinux. Apache is one of the most widely used web servers in the world, known for its flexibility and support for various modules and security features.

In this project, I will create a custom website and apply layered access control using both **user authentication** and **IP-based restrictions**. I will also configure directory permissions, implement secure areas with .htaccess, and enforce access based on subnets. Each task will be tested and validated from an Ubuntu client to simulate real-world access scenarios.

By the end of this project, I will have gained hands-on experience with Apache configuration files, virtual hosts, and web server security, all of which are essential skills in network and system administration.

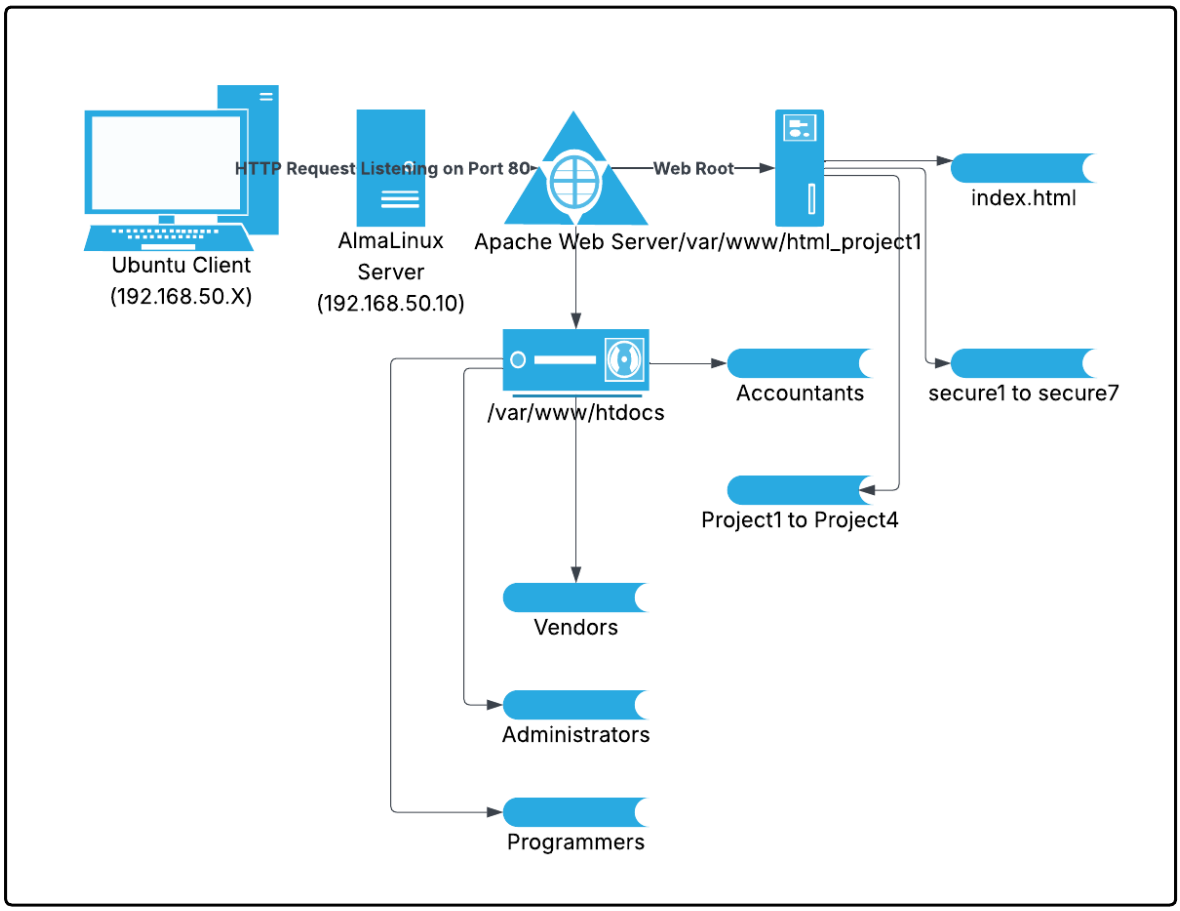


Figure 1 Topology showing the Ubuntu client connected to the Apache server with project directory structure.

TASK 1 – CREATING A WEBSITE

Install and Enable Apache Web Server

To begin building the web server, I installed the Apache HTTP server using dnf. I used the -y flag to skip confirmation prompts. After installation, I enabled the service to start on boot and then started it immediately. I verified that the service is active and running.

```
root@server07 ~ $ dnf install httpd -y
AlmaLinux 9 - AppStream
AlmaLinux 9 - AppStream
AlmaLinux 9 - BaseOS
AlmaLinux 9 - BaseOS
AlmaLinux 9 - Extras
AlmaLinux 9 - Extras
Extra Packages for Enterprise Linux 9 - x86_64
Extra Packages for Enterprise Linux 9 - x86_64
Dependencies resolved.
================================================================================
 Package                                Architecture            Version
================================================================================
Installing:
 httpd                                  x86_64                  2.4.62-1.el9_5.2
Installing dependencies:
 almalinux-logos-httpd                  noarch                  90.5.1-1.1.el9
 apr                                    x86_64                  1.7.0-12.el9_3
 apr-util                               x86_64                  1.6.1-23.el9
 apr-util-bdb                           x86_64                  1.6.1-23.el9
 httpd-core                             x86_64                  2.4.62-1.el9_5.2
 httpd-filesystem                       noarch                  2.4.62-1.el9_5.2
 httpd-tools                            x86_64                  2.4.62-1.el9_5.2
Installing weak dependencies:
 apr-util-openssl                       x86_64                  1.6.1-23.el9
 mod_http2                              x86_64                  2.0.26-2.el9_4.1
 mod_lua                                x86_64                  2.4.62-1.el9_5.2
```

```
root@server07 ~ $ systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
root@server07 ~ $ systemctl start httpd
root@server07 ~ $ systemctl status httpd
● httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset: disabled)
     Active: active (running) since Tue 2025-04-15 09:33:13 EDT; 7s ago
       Docs: man:httpd.service(8)
   Main PID: 3743 (httpd)
     Status: "Started, listening on: port 80"
```

Figure 2 Apache HTTP server installed and started on AlmaLinux.

Installing Terminator for Efficient Multitasking

Before continuing with Apache configuration, I installed **Terminator**, a terminal emulator that allows me to split the terminal window for easier multitasking.

```
root@server07 ~ $ dnf install terminator -y
Last metadata expiration check: 0:10:35 ago on Tue 15 Apr 2025 09:29:37 AM.
Dependencies resolved.
================================================================================
 Package                         Architecture              Version
================================================================================
Installing:
 terminator                      noarch                    2.1.3-1.el9
Installing dependencies:
 keybinder3                      x86_64                    0.3.2-13.el9
 python3-configobj               noarch                    5.0.6-25.el9
```

Figure 3 Installation of the Terminator package to enhance multitasking during Apache setup.

Note. This tool helps me monitor and run multiple commands side-by-side, which is especially useful for editing files, restarting services, and checking logs at the same time.

Backup the original Apache configuration file

To ensure that we have a safe copy of the original Apache configuration before making any modifications, we created a backup of the /etc/httpd/conf/httpd.conf file. This was done using the cp command, and the backup was named httpd.conf.original. This provides a fallback option in case configuration errors occur during the project setup.

```
root@server07 ~ $ cp /etc/httpd/conf/httpd.conf /etc/httpd/conf/httpd.conf.original
root@server07 ~ $ cd /etc/httpd/conf
conf/           conf.d/         conf.modules.d/
root@server07 ~ $ cd /etc/httpd/conf
root@server07 /etc/httpd/conf $ ll
total 40
-rw-r--r--. 1 root root 12005 Jan 21 16:19 httpd.conf
-rw-r--r--. 1 root root 12005 Apr 15 09:53 httpd.conf.original
-rw-r--r--. 1 root root 13430 Jan 21 16:24 magic
root@server07 /etc/httpd/conf $
```

Figure 4 Backup of the original Apache configuration file to httpd.conf.original.

Verifying Apache Web Server

Before modifying the Apache configuration, we confirmed the default installation was successful by accessing the default test page in a browser using: **http://localhost**

This loaded the AlmaLinux Test Page, confirming the Apache service is running correctly and serving content from the default directory /var/www/html.

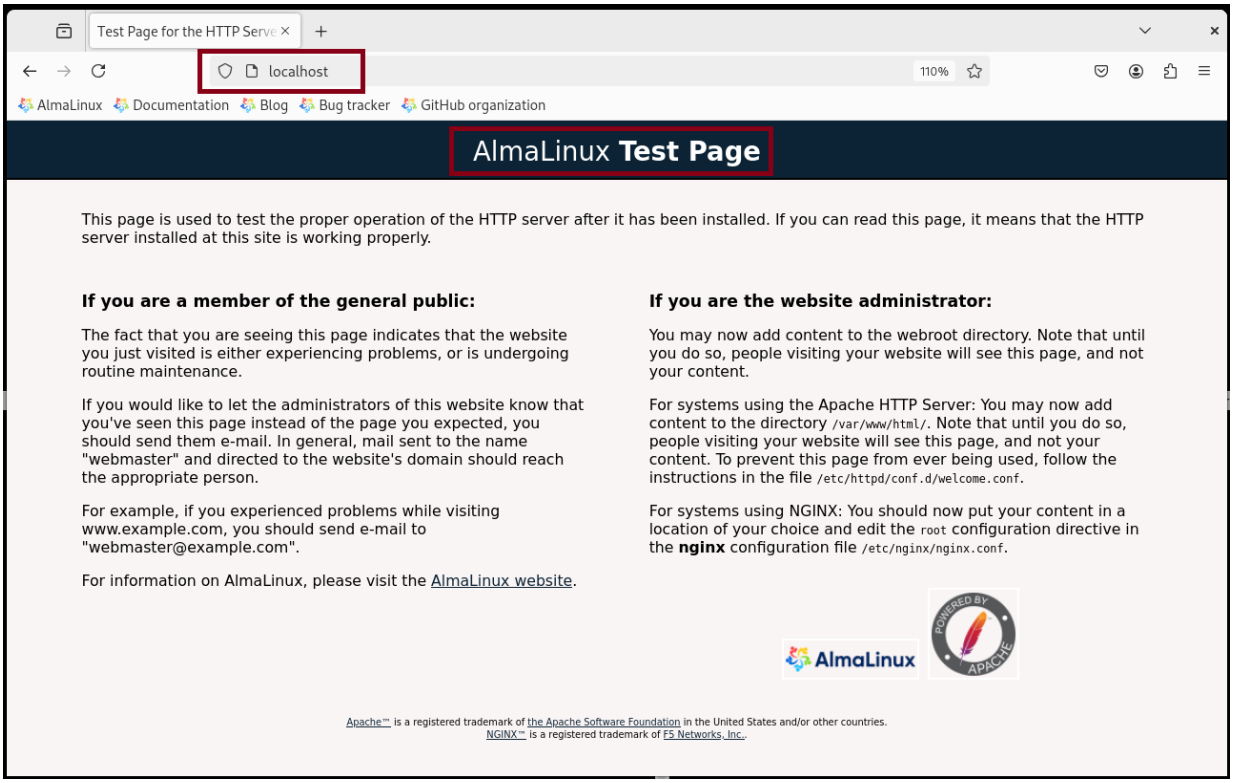


Figure 5 Successful loading of the default Apache test page before customization.

Apache Configuration Validation – IP Accessibility

To ensure the Apache web server is accessible via the server's IP address (192.168.50.10), I verified the configuration inside the httpd.conf file.

As shown in the figure, the following directive is present:

```
ServerRoot "/etc/httpd"
Listen 80
Include conf.modules.d/*.conf
User apache
Group apache
ServerAdmin root@localhost
```

Figure 6 Apache is configured to listen on port 80 (default HTTP), making the server accessible at http://192.168.50.10.

Note. This setting confirms that Apache is listening on port 80 for incoming HTTP requests on **all interfaces**, including 192.168.50.10. Therefore, clients on the same network can connect to the server using its IP address.

Firewall Configuration for Apache Server

To allow HTTP access to the Apache web server, I checked my active zone using firewall-cmd --get-active-zones and confirmed my network interface was part of the nm-shared zone. I then enabled access on port 80 using the following command:

```
root@server07 ~ $ firewall-cmd --get-active-zones
external
  interfaces: ens160
nm-shared
  interfaces: ens192
root@server07 ~ $ firewall-cmd --permanent --add-port=80/tcp --zone=nm-shared
success
root@server07 ~ $ firewall-cmd --reload
success
```

Figure 7 Port 80 opened in the nm-shared zone and firewall reloaded.

Note. This allows HTTP requests to reach the Apache web server hosted on the AlmaLinux Server VM.

Apache DocumentRoot and Directory Access Configuration

To ensure Apache serves web content from the custom directory /var/www/html_project1, I modified the existing Apache configuration file /etc/httpd/conf/httpd.conf.

```
root@server07 /etc/httpd/conf $ vim httpd.conf
```

```
DocumentRoot "/var/www/html_project1"

<Directory "/var/www">
    AllowOverride None
    # Allow open access:
    Require all granted
</Directory>

<Directory "/var/www/html_project1">
    Options Indexes FollowSymLinks
    AllowOverride None
</Directory>
```

Figure 8 Apache configuration file showing updated DocumentRoot and modified Directory block.

This configuration ensures that the server:

- Knows the correct location of the project files.
- Grants access to all clients.
- Enables directory listing and symbolic link following if needed.

Starting and Enabling the Apache (httpd) Service

To make sure the Apache web server is running and automatically starts at boot, I used the following systemctl commands:

```
root@server07 ~ $ systemctl enable --now httpd
root@server07 ~ $ systemctl status httpd
● httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled
     Active: active (running) since Tue 2025-04-15 09:33:13 EDT; 2h
       Docs: man:httpd.service(8)
   Main PID: 3743 (httpd)
```

Figure 9 Apache service (httpd) is active and enabled.

Creating and Securing the Web Root Directory

To match the custom DocumentRoot configuration, I created the directory /var/www/html_project1. For better workflow and security, I set the owner to my user account (gkeymole) and the group to apache, which is used by the Apache service.

This allows me to manage and edit files easily, while still ensuring the web server can serve them securely.

```
root@server07 ~ $ mkdir /var/www/html_project1
root@server07 ~ $ chown -R gkeymole:apache /var/www/html_project1
root@server07 ~ $ chmod -R 755 /var/www/html_project1
root@server07 ~ $
```

Figure 10 Project directory created with proper user and group ownership for editing and secure access.

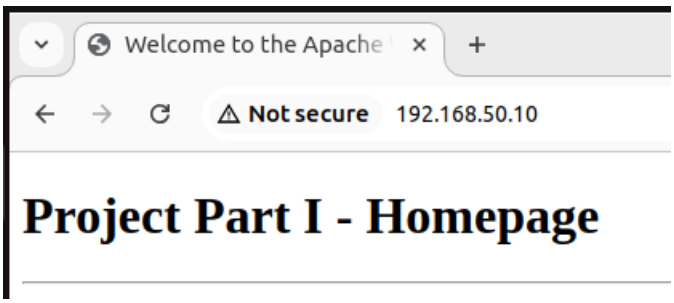
Creating and Testing the Homepage

To meet the project requirement of having a functional homepage served from /var/www/html_project1, I created the file index.html using the following command:

```
root@server07 ~ $ vim /var/www/html_project1/index.html
```

Figure 11 Editing the index.html homepage using vim in the /var/www/html_project1 directory.

After saving the file, I opened a web browser on my Ubuntu client and accessed the server via its IP address 192.168.50.10. The homepage loaded successfully, confirming the Apache server is reading from the correct DocumentRoot.




```
<!DOCTYPE html>
<html>
<head>
  <title>Welcome to the Apache Web Project</title>
</head>
<body>
  <h1>Project Part I - Homepage</h1>
  <hr>
```

Figure 12 Display of index.html homepage loaded via the Apache web server at 192.168.50.10

Homepage Highlights:

- Fully meets the title requirement: *Project Part I – Homepage*
- Includes hyperlinks to all tasks (Secure 1–7, Project 1–4, Role pages)
- All required HTML tags are present in the code
- Provides organized navigation for testing each web page
- Confirms Apache is correctly serving content from /var/www/html_project1

TASK 2 – AUTHENTICATION

Creating the secure1 Directory for Authentication Testing

To begin configuring the secure sections of the web project, I created a dedicated directory named secure1 under /var/www/html_project1. This directory will later be protected using **user-based access control** with Apache's <Directory> directive.

```
root@server07 ~ $ mkdir /var/www/html_project1/secure1
root@server07 ~ $
```

Figure 13 Creating the secure1 directory within the Apache project root for implementing user-based access restrictions.

Creating the User Authentication File

To protect the secure1 directory, I generated an .htpasswd file that stores the credentials for the user **user01**, using the password **secret.** The -c option creates the file (if it doesn't exist), and -b allows setting the password directly via the command line.

```
root@server07 ~ $ htpasswd -bc /var/www/.htpasswd user01 secret
Adding password for user user01
root@server07 ~ $
```

*Figure 14 Creating the .htpasswd file in /var/www/ with user **user01** using the htpasswd command.*

Configuring secure1 Directory (User-Based Access Control)

The first type of access control applied in this project is **simple user authentication**. I configured the secure1 directory so that **only user01** can access it, regardless of the client's IP address. This ensures access is tightly restricted to one authorized user.

```
<Directory "/var/www/html_project1/secure1">
    AuthType Basic
    AuthName "Restricted Area"
    AuthUserFile /var/www/.htpasswd
    Require user user01
</Directory>
```

Figure 15 Apache <Directory> configuration block restricting access to user01 only.

- **AuthType Basic:** Enables basic username/password login
- **AuthName:** This is the message that shows up in the login prompt
- **AuthUserFile:** Points to the file that stores the user credentials (.htpasswd)
- **Require user user01:** Only allows this specific user

To validate the changes before applying them, I used the httpd -t command to check the syntax of the Apache configuration file:

```
root@server07 ~ $ httpd -t
Syntax OK
root@server07 ~ $
```

Figure 16 Verifying Apache Configuration Syntax

Configuring secure2 Directory (User AND Subnet Access Control)

For the second directory, secure2, I created it inside /var/www/html_project1 to implement **compound access control** using the <RequireAll> directive.

```
root@server07 ~ $ mkdir /var/www/html_project1/secure2
root@server07 ~ $
```

Figure 17 Creating the secure2 directory.

This configuration restricts access to users who:

- Authenticate as user01, **AND**
- Are connecting from the **192.168.50.0/24** subnet

This logical AND condition means that both criteria must be satisfied for access to be granted.

```
<Directory "/var/www/html_project1/secure2">
    AuthType Basic
    AuthName "Restricted Access"
    AuthUserFile /var/www/.htpasswd
    Require valid-user
    <RequireAll>
        Require ip 192.168.50.0/24
        Require user user01
    </RequireAll>
</Directory>
```

Figure 18 Apache <Directory> block for secure2 using RequireAll to combine IP and user authentication.

I validated the syntax again before applying changes:

```
root@server07 ~ $ httpd -t
Syntax OK
root@server07 ~ $
```

Figure 19 Syntax validation of Apache configuration.

Then, I reloaded the service with systemctl reload httpd to apply the configuration without interrupting ongoing sessions.

```
gkeymole@server07 ~ $ sudo systemctl reload httpd
gkeymole@server07 ~ $
```

Configuring secure3 Directory (Logical OR Access Control)

The secure3 directory is configured to allow access if **either** of the following conditions is met:

- The client authenticates as user01, **OR**
- The request originates from the **192.168.50.0/24** subnet

This is achieved using the <RequireAny> directive, which implements **OR-based access control**. It's useful when you want to accommodate a specific user while also granting open access to trusted internal users based on IP.

```
root@server07 ~ $ mkdir /var/www/html_project1/secure3
root@server07 ~ $ httpd -t
Syntax OK
root@server07 ~ $ systemctl reload httpd
root@server07 ~ $
```

Figure 20 Secure3 Creation, Syntax check and reload for applying secure3 configuration.

```
<Directory "/var/www/html_project1/secure3">
    <RequireAny>
        Require ip 192.168.50.0/24
        AuthType Basic
        AuthName "Restricted Area"
        AuthUserFile /var/www/.htpasswd
        Require user user01
    </RequireAny>
</Directory>
```

Figure 21 Apache configuration using <RequireAny> to permit access from either a user or subnet.

Note. This configuration ensures that either the user or subnet condition is enough to allow access.

Configuring secure4 Directory (User-Based Access for a Different User)

The secure4 directory demonstrates how to apply **user-based access control** for a different user. This time, access is restricted to **user02** only. This approach is like secure1 but applied to a different user to demonstrate flexibility in user-specific configurations.

To begin, I created the secure4 directory under /var/www/html_project1:

```
root@server07 ~ $ mkdir /var/www/html_project1/secure4
root@server07 ~ $
```

Figure 22 Creating the secure4 directory inside the web root for Task 2 testing.

Next, I added user02 to the existing .htpasswd file located in /var/www/. This user was added with the same password "secret" to maintain consistency across test users.

```
root@server07 ~ $ htpasswd -b /var/www/.htpasswd user02 secret
Adding password for user user02
root@server07 ~ $
```

Figure 23 Adding user02 to the .htpasswd file using the htpasswd command

I then updated the Apache configuration to protect the secure4 directory using the <Directory> directive. Access was restricted **only** to user02, regardless of the subnet.

```
<Directory "/var/www/html_project1/secure3">
    AuthType Basic
    AuthName "Restricted Access"
    AuthUserFile /var/www/.htpasswd
    <RequireAny>
        Require ip 192.168.50.0/24
        Require user user01
    </RequireAny>
</Directory>
```

Figure 24 Apache <Directory> configuration to allow access to secure4 only for user02.

As always, I ran a syntax check before applying the changes:

```
root@server07 ~ $ httpd -t
Syntax OK
root@server07 ~ $
```

Figure 25 Verifying Apache configuration syntax before reloading.

Finally, I reloaded the Apache service to activate the updated configuration:

```
root@server07 ~ $ systemctl reload httpd
root@server07 ~ $
```

Figure 26 Reloading Apache service to apply secure4 access control.

Configuring secure5 Directory Using .htaccess for User-Based Access Control

To explore an alternative method of access control, I created the secure5 directory within the Apache project root. This directory is protected using an .htaccess file instead of the main Apache configuration file. The goal is to restrict access to the directory for user user01, regardless of the IP address.

```
root@server07 ~ $ mkdir /var/www/html_project1/secure5
root@server07 ~ $
```

Figure 27 Creating the secure5 directory under /var/www/html_project1.

To make this possible, I first modified the Apache configuration file /etc/httpd/conf/httpd.conf to allow .htaccess overrides specifically for this directory by adding the following:

```
<Directory "/var/www/html_project1/secure5">
    AllowOverride All
</Directory>
```

Figure 28 Allowing .htaccess override in Apache config for secure5.

Note. This tells Apache to respect any .htaccess settings placed inside the secure5 folder.

After saving the file, I reloaded the Apache configuration using the reload command to apply the changes without restarting the entire service.

Inside the /var/www/html_project1/secure5 directory, I created the .htaccess file with the following content:

```
root@server07 ~ $ vim /var/www/html_project1/secure5/.htaccess
```

```
AuthType Basic
AuthName "Secure Area"
AuthUserFile /var/www/.htpasswd
Require user user01
```

Figure 29 .htaccess file created inside secure5 to enable user-based access control for user01.

This configuration restricts access to user01, referencing the same credentials file created earlier.

This method demonstrates how Apache can delegate access control to .htaccess files, offering flexibility when modifying the main configuration is not preferred or possible.

Configuring secure6 Directory (User and Subnet-Based Access with .htaccess)

For secure6, the project requires enforcing access to the directory using both a specific user (user01) and a network restriction, but this time implemented through an .htaccess file.

To begin, I created the directory:

```
root@server07 ~ $ mkdir /var/www/html_project1/secure6
root@server07 ~ $
```

Figure 30 Creating the secure6 directory inside /var/www/html_project1.

Next, I created an .htaccess file within secure6 to define access control. It specifies that only user01 can access the directory **and only** if connecting from the 192.168.50.0/24 subnet. The .htaccess file contained the following rules using the <RequireAll> directive to enforce both the user and IP-based restriction:"

```
root@server07 ~ $ vim /var/www/html_project1/secure6/.htaccess
```

```
AuthType Basic
AuthName "Restricted Access"
AuthUserFile /var/www/.htpasswd
<RequireAll>
    Require user user01
    Require ip 192.168.50.0/24
</RequireAll>
```

Figure 31 Creating .htaccess file inside secure6 with combined user and IP subnet-based access control.

To allow Apache to process the .htaccess file, I modified the httpd.conf to include:

```
<Directory "/var/www/html_project1/secure6">
    AllowOverride All
</Directory>
```

Figure 32 Enabling .htaccess support in httpd.conf for the secure6 directory using AllowOverride All.

After confirming the configuration syntax using httpd -t and getting Syntax OK, I reloaded Apache to apply the changes.

```
root@server07 ~ $ httpd -t
Syntax OK
root@server07 ~ $ systemctl reload httpd
root@server07 ~ $
```

Figure 33 Validating Apache syntax and applying the new configuration

This configuration highlights how to delegate control using .htaccess files, which is especially useful when fine-tuning access to individual directories without editing the main config file directly.

Configuring secure7 Directory with Combined Access Rules (.htaccess)

For the final task in the authentication section, I created the **secure7** directory to practice applying access control based on **either user credentials** or **IP address**. This setup allows access **if the visitor is user01 or is connecting from the trusted 192.168.50.0/24 subnet**, demonstrating a logical OR condition in a simpler way using .htaccess.

I began by creating the directory:

```
root@server07 ~ $ mkdir /var/www/html_project1/secure7
root@server07 ~ $
```

Figure 34 Creating the secure7 directory under /var/www/html_project1.

Next, I created a .htaccess file inside this directory with the following configuration:

```
root@server07 ~ $ vim /var/www/html_project1/secure7/.htaccess
```

```
AuthType Basic
AuthName "Restricted Access"
AuthUserFile /var/www/.htpasswd
<RequireAny>
    Require ip 192.168.50.0/24
    Require user user01
</RequireAny>
```

Figure 35 .htaccess configuration inside secure7 allowing access to user01 or anyone from 192.168.50.0/24.

To allow Apache to interpret the .htaccess file in secure7, I updated the Apache configuration file /etc/httpd/conf/httpd.conf by adding a <Directory> block with the AllowOverride All directive:

```
<Directory "/var/www/html_project1/secure7">
    AllowOverride All
</Directory>
```

Figure 36 Apache <Directory> block added to enable .htaccess support for secure7.

Finally, I validated the Apache configuration syntax and reloaded the service to apply the new settings.

With this configuration, secure7 is now properly protected using an .htaccess file that allows access if **either** the authenticated user is **user01** or the client is connecting from the **192.168.50.0/24** subnet. This demonstrates a clean implementation of **OR-based logic** using the <RequireAny> directive, while the AllowOverride All directive in httpd.conf ensures the server reads and applies the .htaccess file.

Validation Task 2 - Authentication (Ubuntu Client)

Preparing secure Directories for Validation

Before beginning the validation of each secure directory's access controls, I created a basic index.html file inside every secureX directory (secure1 to secure7). This ensured that there was actual content to load during testing and that the Apache server could return a visible response when access was permitted. This approach helped clearly differentiate successful access (directory listing or HTML content) from denied access (403 or forbidden).

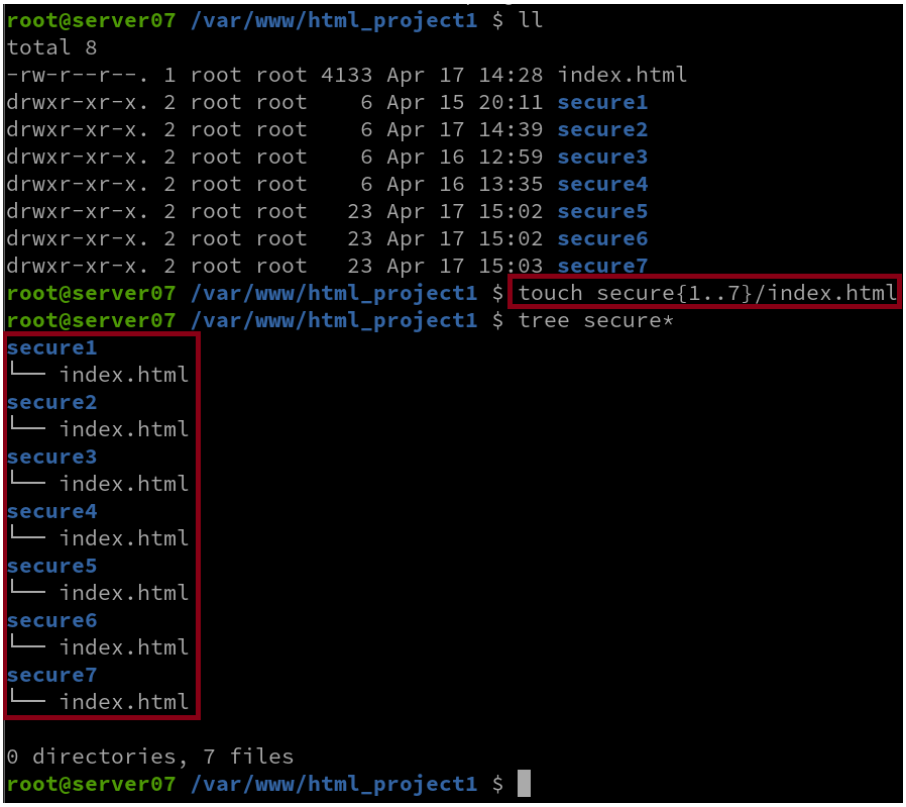


Figure 37 Creating minimal index.html files inside each secure directory to support access testing.

Example of a Secure Directory's index.html File

To make it easy to confirm when access was granted to a secure directory, I placed a simple but identifiable index.html file inside each one. The contents were kept minimal just a short message indicating which directory was accessed successfully.

Below is an example of the index.html file used in the secure2 directory:

```
<!DOCTYPE html>
<html>
<head>
  <title>Secure2 Validation Page</title>
</head>
<body>
  <h1>You have successfully accessed Secure2</h1>
</body>
</html>
```

Figure 38 Sample index.html file used in secure2 directory for access confirmation.

Preparing the Environment for Subnet-Based Access Validation

To simplify and centralize the validation of subnet-based access control across all secure directories, I adopted a browser-friendly strategy by assigning multiple IP addresses to the single network interface on both the **Ubuntu client** and the **AlmaLinux Apache server**.

This solution ensures that:

- The **client** can simulate requests from multiple subnets without reconfiguring the network each time.
- The **server** can accept requests to multiple destination IPs, allowing hyperlink-based validation through the index.html file.

IP Assignment on the Ubuntu Client

Using the following nmcli command, I added all the required subnet IPs to the Ubuntu client's LAN1 interface:

```
gkeymole@client07:~$ sudo nmcli con mod LAN1 +ipv4.addresses 10.35.16.2/24 +ipv4.addresses 10.35.17.2/24 +ipv4.addresses 1
92.168.100.2/24 +ipv4.addresses 10.50.1.2/24 +ipv4.addresses 10.51.1.2/24 +ipv4.addresses 10.52.1.2/24 +ipv4.addresses 10.
53.1.2/24
gkeymole@client07:~$ sudo nmcli connection down LAN1 && sudo nmcli connection up LAN1
Connection 'LAN1' successfully deactivated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/8)
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/9)
gkeymole@client07:~$
```

Figure 39 Assigning Multiple Subnet IP Addresses to Ubuntu Client

IP Assignment on the Apache Server

On the AlmaLinux Apache server, I applied the following command to add additional IPs representing the expected subnets for validation:

```
root@server07 ~ $ nmcli con mod LAN1 +ipv4.addresses 10.35.16.1/24 +ipv4.addresses 10.35.17.1/24 +ipv4.addresses 192.1
68.100.1/24 +ipv4.addresses 10.50.1.1/24 +ipv4.addresses 10.51.1.1/24 +ipv4.addresses 10.52.1.1/24 +ipv4.addresses 10.
53.1.1/24
```

```
root@server07 ~ $ sudo nmcli con down LAN1 && sudo nmcli con up LAN1
Connection 'LAN1' successfully deactivated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/12)
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/13)
root@server07 ~ $
```

Figure 40 Assigning Multiple Subnet IP Addresses to the AlmaLinux Server

Benefit of This Setup

With this setup, I can now simulate requests from various subnets by simply clicking on the corresponding hyperlinks in the index.html file. The source IP used by the browser depends on the destination IP targeted, allowing seamless validation of subnet-based access rules.

Preparing the Homepage for Validation (index.html)

After setting up the required subnet IP addresses on both the Ubuntu client and the AlmaLinux Apache server, I configured the central homepage (index.html) located in /var/www/html_project1. This file serves as the main entry point for testing the different access controls implemented for each secure directory.

To do this, I edited the file to include hyperlinks pointing to each test case using different destination IPs. Each link is designed to simulate access from a specific subnet by directing the browser to request the resource through a different server IP.

This allowed me to perform all validation directly from the web browser on the Ubuntu client, without needing to manually reconfigure the network between tests.

Below is the HTML structure used in the file:

```
<!DOCTYPE html>
<html>
<head>
  <title>Welcome to the Apache Web Project</title>
</head>
<body>
  <h1>Project Part I - Homepage</h1>
  <hr>

  <!-- Task 2 - Authentication -->
  <h2>Task 2 - Authentication</h2>
  <ul>
    <li><a href="http://192.168.50.10/secure1">secure1 - user01 from any subnet (allow)</a></li>
    <li><a href="http://192.168.50.10/secure2">secure2 - user01 from 192.168.50.0/24 (allow)</a></li>
    <li><a href="http://10.35.16.1/secure2">secure2 - user01 from 10.35.16.0/24 (deny)</a></li>
    <li><a href="http://192.168.50.10/secure3">secure3 - user01 OR 192.168.50.0/24 (allow)</a></li>
    <li><a href="http://10.35.16.1/secure3">secure3 - not user01 AND not subnet (deny)</a></li>
    <li><a href="http://192.168.50.10/secure4">secure4 - user02 only (allow)</a></li>
    <li><a href="http://192.168.50.10/secure5">secure5 - user01 via .htaccess (allow)</a></li>
    <li><a href="http://192.168.50.10/secure6">secure6 - user01 from 192.168.50.0/24 (allow)</a></li>
    <li><a href="http://10.35.16.1/secure6">secure6 - user01 from 10.35.16.0/24 (deny)</a></li>
    <li><a href="http://192.168.50.10/secure7">secure7 - user01 OR 192.168.50.0/24 (allow)</a></li>
    <li><a href="http://10.35.16.1/secure7">secure7 - neither user01 nor subnet (deny)</a></li>
  </ul>
```

Figure 41 HTML structure of index.html showing subnet-specific validation links for Task 2

Validating Access Control for secure1

To verify the configuration for the **secure1** directory, I launched a web browser on my Ubuntu client (IP address: 192.168.50.20) and navigated to the homepage located at http://192.168.50.10.

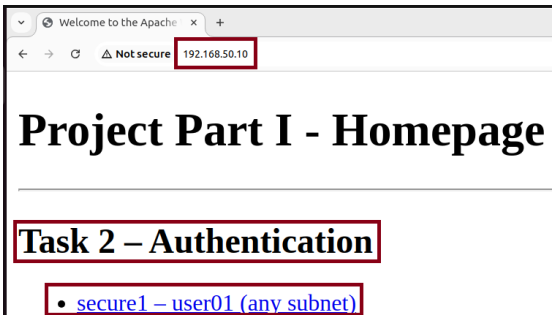


Figure 42 Homepage loaded from the Apache server, with links to all secure directories.

I clicked on the hyperlink for **secure1**, which led to the path http://192.168.50.10/secure1. As expected, the server prompted me for authentication.

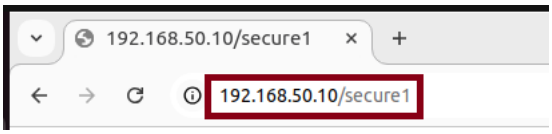


Figure 43 Browser redirected to secure1 and requesting authentication credentials.

I entered the credentials for **user01**, which had previously been defined in the .htpasswd file. Upon successful login, access was granted and I was redirected to the **Index of /secure1** directory page

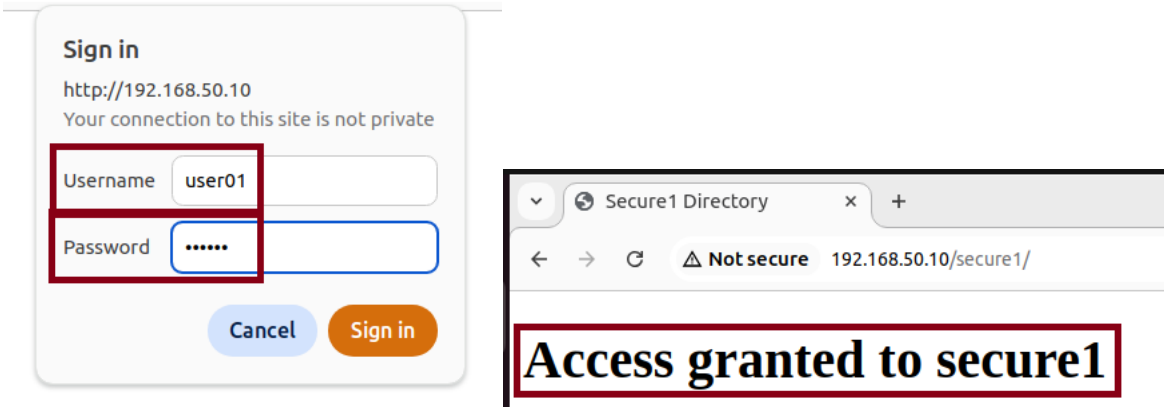


Figure 44 Apache authentication prompt for accessing secure1 & Successful access to secure1 confirmed by Apache directory listing page.

This confirmed that only **user01** was allowed to access this resource as intended in the configuration.

Validating Access Control for secure2

To test the combined user and subnet restriction applied to secure2, I remained on the same Ubuntu client (192.168.50.20), which is within the allowed **192.168.50.0/24** subnet. I clicked the hyperlink for secure2 on the homepage.

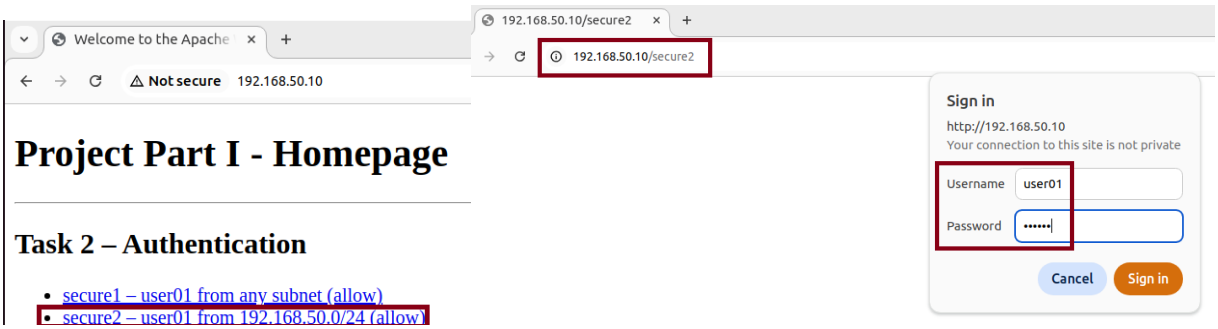


Figure 45 Browser prompt after accessing /secure2 from 192.168.50.0/24 subnet.

After entering valid credentials for user01, I was granted access and redirected to the secure2 directory listing, confirming that the user authentication and subnet check both succeeded.

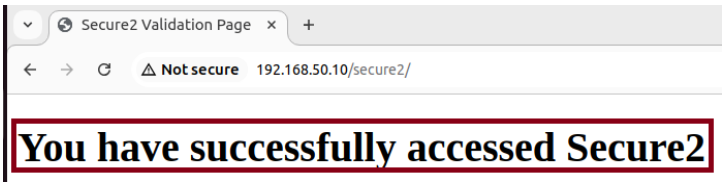


Figure 46 Directory listing shown after successful login from an allowed subnet.

To ensure the subnet restriction was working, I added a second IP address (10.35.16.1) to the client and repeated the test. This time, although the login prompt appeared, submitting the credentials resulted in an access denied message.

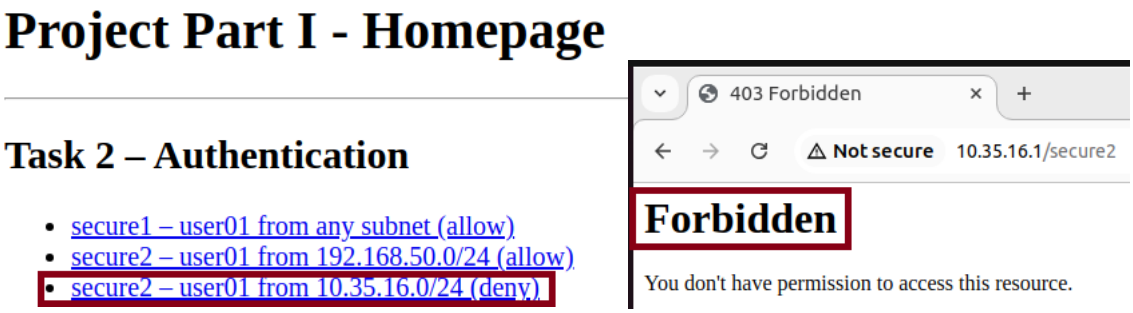


Figure 47 Attempt to access secure2 from unauthorized subnet (10.35.16.1) correctly results in a 403 Forbidden error

Validating Access Control for secure3

The secure3 directory was configured to allow access using a logical **OR** condition. This means access is permitted if:

- The user is authenticated as user01, **OR**
- The request originates from the trusted 192.168.50.0/24 subnet.

This demonstrates Apache's ability to grant access when **either** condition is satisfied using the <RequireAny> directive.

I first tested access using the Ubuntu client with IP 192.168.50.20, which falls within the allowed subnet. I clicked the appropriate hyperlink for secure3 in the homepage (targeting 192.168.50.10), entered the correct credentials, and was granted access.

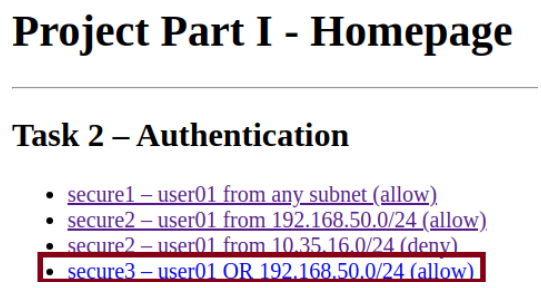


Figure 48 Homepage with hyperlink to secure3 from 192.168.50.10

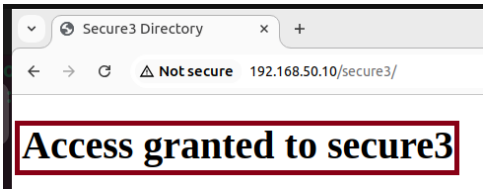


Figure 49 Access granted to secure3 from user01 via 192.168.50.0/24 subnet

Next, I validated that access would still be permitted from a **non-allowed subnet** as long as valid credentials for user01 were provided. I clicked the second secure3 validation link, targeting 10.35.16.1, and confirmed that despite the client being outside the trusted subnet, authentication as user01 allowed access.

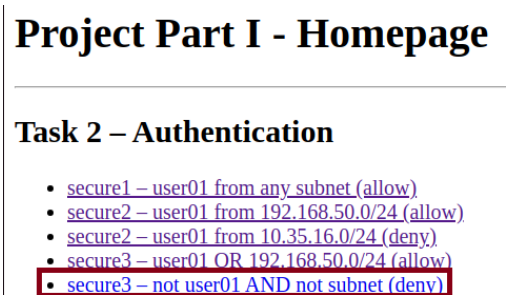


Figure 50 Homepage with hyperlink to secure3 using 10.35.16.1 (alternate subnet)

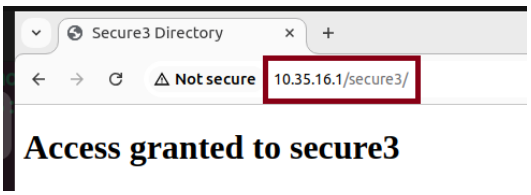


Figure 51 Access granted to secure3 via user authentication from unauthorized subnet

Validating Access Control for secure4

The **secure4** directory was configured to restrict access to a different user account "**user02**" showcasing how Apache can enforce user-based access control independently of IP address or subnet. This configuration ensures that **only user02** can access the directory, and all other users are denied.

To validate this setup, I used the Ubuntu client and clicked on the hyperlink to /secure4, which targeted the server at 192.168.50.10.

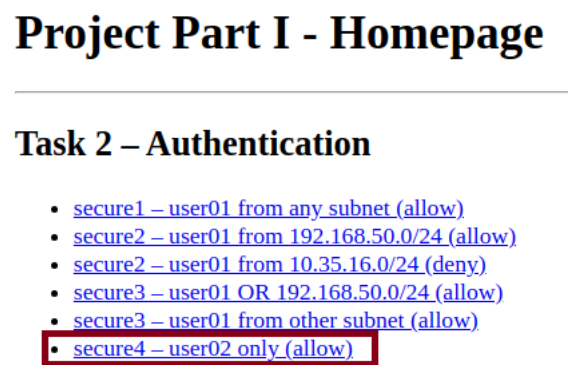


Figure 52 Homepage with hyperlink to secure4 for validating user02 access

As expected, I was prompted to enter credentials. I first entered the login details for **user01**, which were previously used in other tests. The server rejected the authentication, confirming that user01 did not have access.

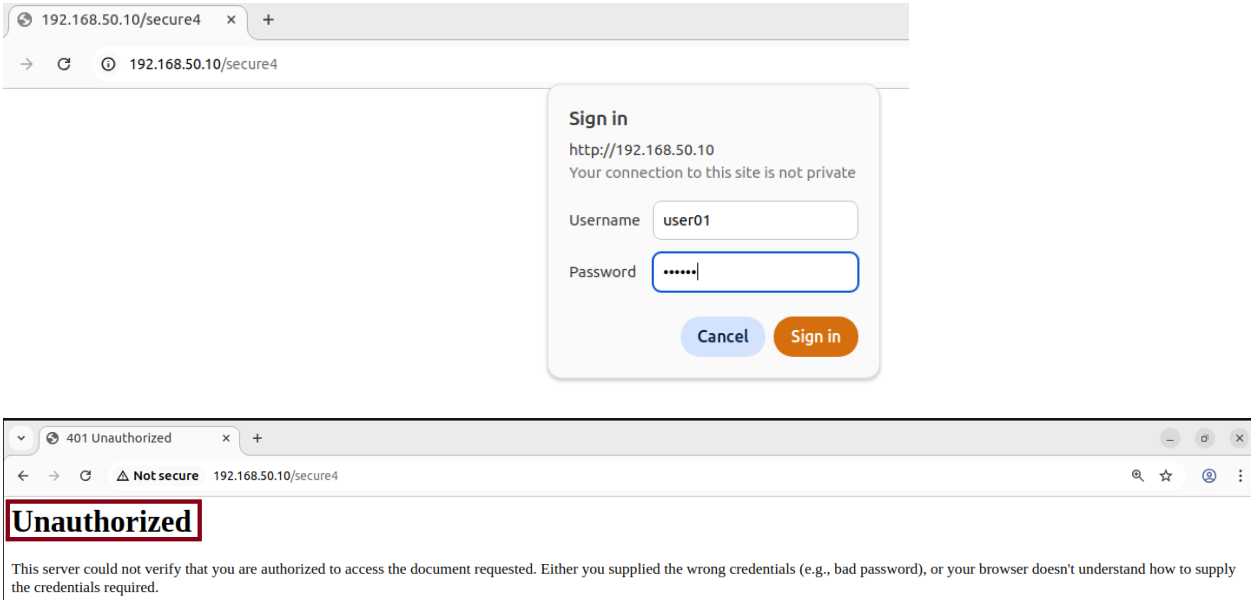


Figure 53 Access denied when attempting to authenticate to secure4 as user01

Next, I entered the credentials for **user02**, which were created during configuration with the same password “secret”. This time, access was granted, and the page displayed the secure4 index content, confirming that user-based authentication was working as intended.

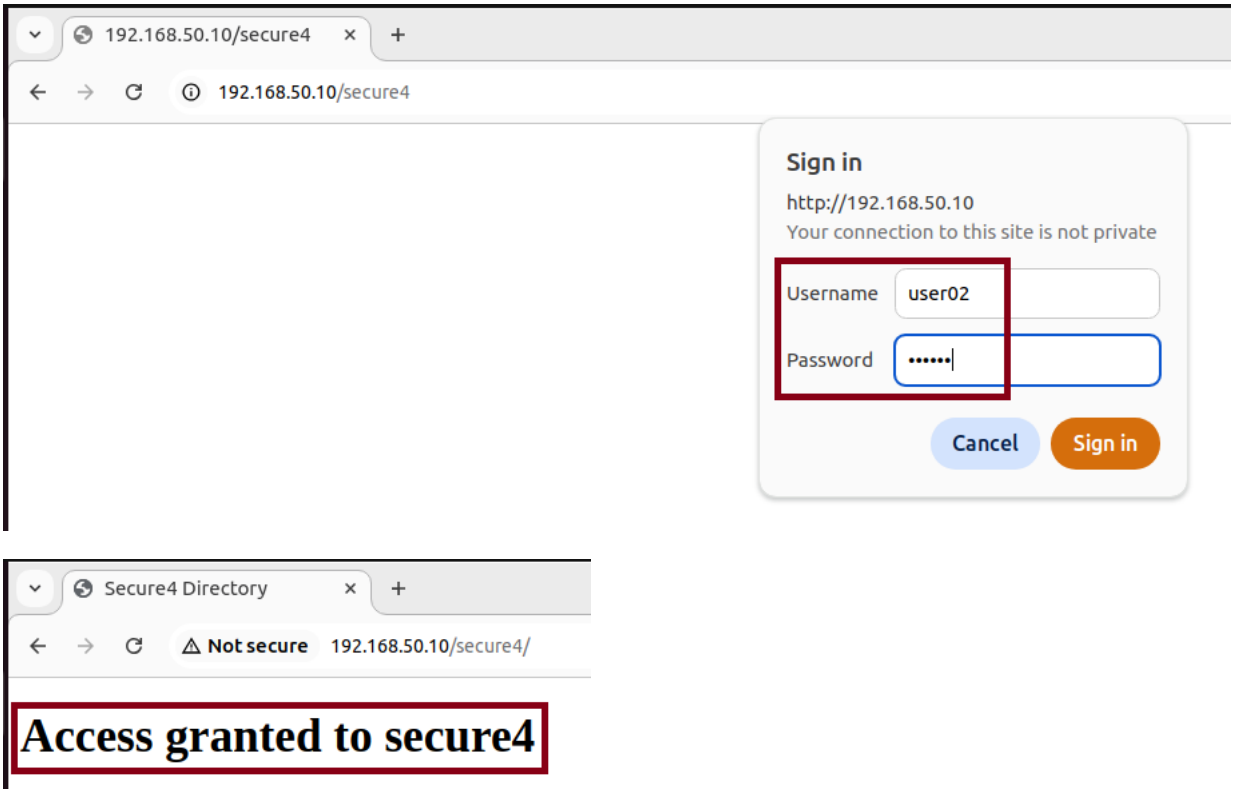


Figure 54 Access granted to secure4 after successful authentication as user02

This test validates that Apache correctly enforces access restrictions based on the specified user account using the Require user directive.

Validating Access Control for secure5

The **secure5** directory demonstrates how Apache can enforce user-based access restrictions using a local **.htaccess file**, rather than placing the access rules in the main configuration file. This is useful when administrators prefer decentralized control or lack access to the global Apache configuration.

In this case, access to /secure5 was restricted to **user01**. I initiated the validation by clicking on the hyperlink to /secure5 from the homepage.

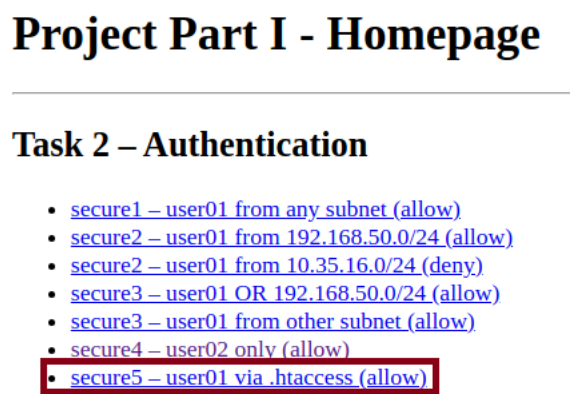


Figure 55 Homepage with hyperlink pointing to secure5 for .htaccess-based user restriction

The server prompted for authentication. I entered the credentials for **user01**, which were previously defined in the .htpasswd file.

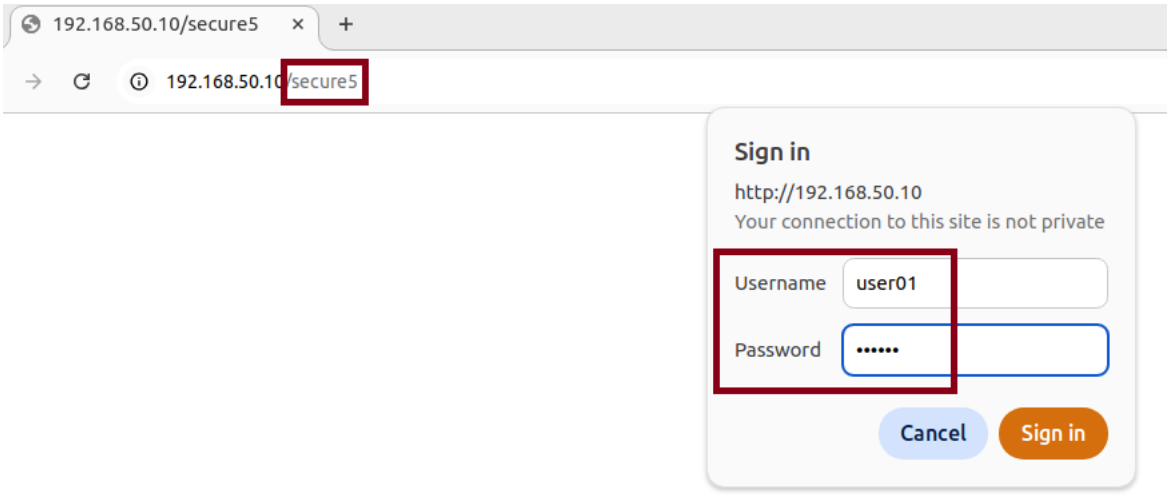


Figure 56 Apache authentication challenge triggered by .htaccess for secure5

After successfully authenticating, I was granted access to the directory and shown the contents of the secure5 page, confirming that the .htaccess configuration was respected and working correctly.

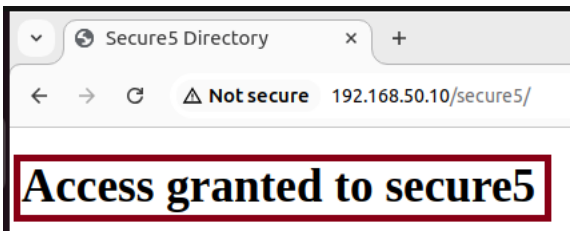


Figure 57 Access granted to secure5 using .htaccess validation for user01

This test confirms that Apache properly honors the .htaccess directives and allows user-based access without needing to alter the main httpd.conf file.

Validating Access Control for secure6

The **secure6** directory was configured to enforce **both user authentication and subnet restriction**, but this time using a **.htaccess file** rather than the Apache main configuration. The goal was to ensure access is granted **only if** the client is both:

- Authenticated as **user01**, and
- Connecting from the **192.168.50.0/24** subnet

To begin the test, I clicked the hyperlink to /secure6 from the project homepage while my Ubuntu client was operating from the **192.168.50.20** IP address (inside the allowed subnet).

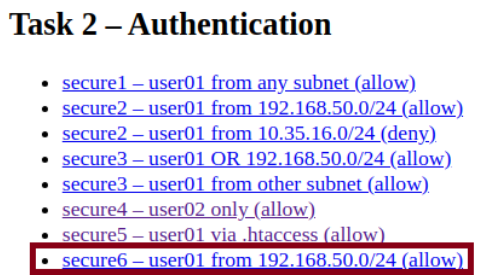


Figure 58 Homepage with hyperlink to secure6 using .htaccess and subnet restriction

Upon clicking, the browser requested credentials as expected. After entering **user01**'s credentials, I was granted access.

Figure 59 Successful access to secure6 from subnet 192.168.50.0/24 with valid user credentials

Next, I tested access denial by simulating a request from a **different subnet**, specifically from **10.35.16.1**, which is not within the allowed range. Although the browser still prompted for credentials, the server responded with a **403 Forbidden** error after submission, confirming the subnet restriction was enforced.

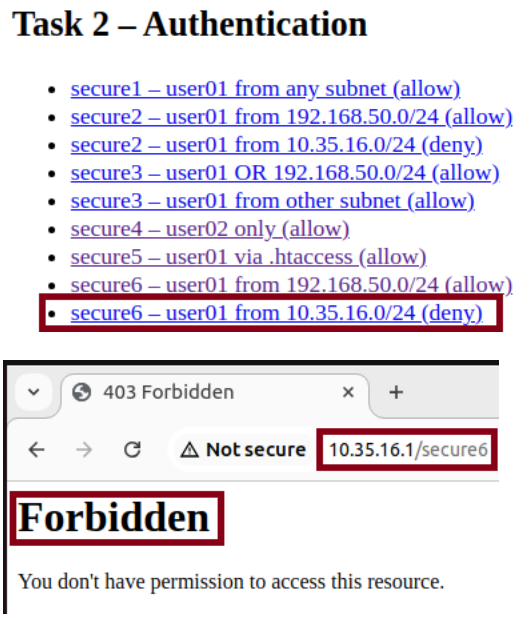


Figure 60 Denied access to secure6 from an unauthorized subnet despite correct credentials (403 Forbidden)

This confirmed that the **RequireAll** logic defined in the .htaccess file worked properly, and both authentication **and** subnet membership were required to gain access.

Validating Access Control for secure7

The **secure7** directory demonstrates how to apply **logical OR access control** using a **.htaccess file**. Access is permitted if the client:

- Authenticates as **user01**, **or**
- Connects from the **192.168.50.0/24** subnet

This flexible approach allows for broader access while still maintaining control over who can enter.

I first tested access **from a valid subnet (192.168.50.20)**. Without providing credentials, the server allowed me in directly, confirming that subnet access alone was sufficient.

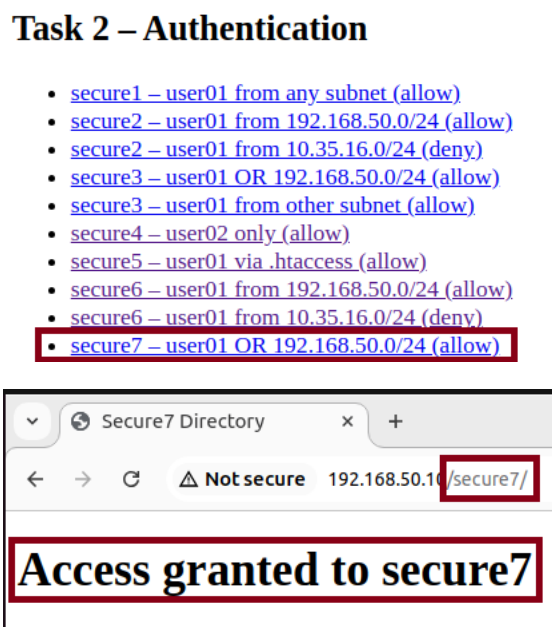


Figure 61 Access granted to secure7 from subnet 192.168.50.0/24 without user authentication

Next, I changed the client's source IP to **10.35.16.1**, which is **not part of the allowed subnet**. This time, I was prompted for credentials. After entering the correct credentials for **user01**, I was successfully granted access.

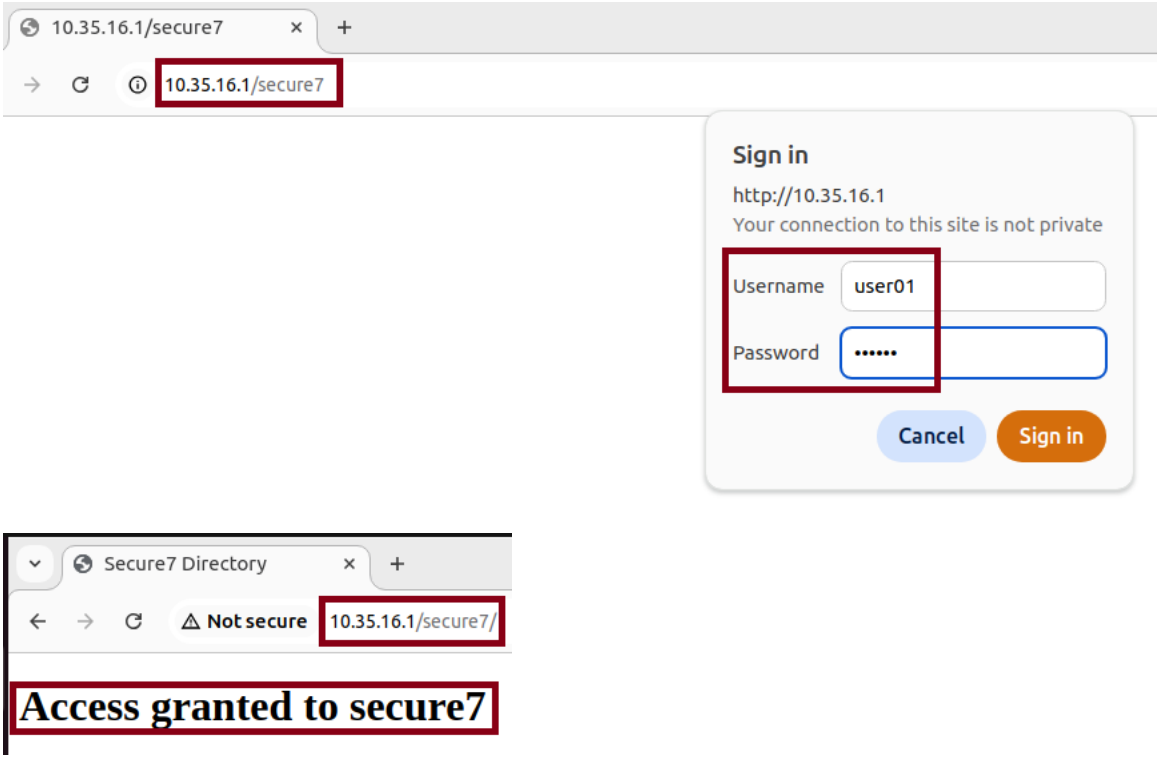


Figure 62 Access granted to secure7 from an invalid subnet using valid user01 credentials

This validation confirmed that the **OR condition** was correctly implemented in the .htaccess file, and **either** subnet-based access **or** correct authentication would allow entry.

TASK 3 – ACCESSIBILITY

Preparing Project Directories for Access Control

To begin Task 3, I created four subdirectories within the /var/www/html_project1 directory: Project1, Project2, Project3, and Project4. These directories are used to apply and validate access control rules based on subnet and file-type restrictions.

```
root@server07 ~ $ mkdir /var/www/html_project1/Project{1..4}
root@server07 ~ $
```

Figure 63 Creating directories for accessibility testing in /var/www/html_project1.

Creating Web Pages for Testing

Each project directory contains a simple HTML file named after the directory (e.g., project1.html in Project1). These files are used to verify directory accessibility and file-level access controls from different client subnets.

```
root@server07 ~ $ touch /var/www/html_project1/Project1/project1.html /var/www/html_project1/Project2/project2.html /var/www/
html_project1/Project3/project3.html /var/www/html_project1/Project4/project4.html
```

Figure 64 Creating test HTML pages for each project directory.

To validate file-level restrictions, I also created additional test files, such as secret.doc, test.html, image.gif and log.txt depending on the directory-specific rule.

```
root@server07 /var/www/html_project1 $ tree /var/www/html_project1/
/var/www/html_project1/
├── index.html
├── Project1
│   ├── project1.html
│   ├── readme.txt
│   ├── secret.doc
│   ├── secret.html
│   └── test.txt
├── Project2
│   ├── image.png
│   ├── manual.txt
│   ├── project2.html
│   ├── readme.txt
│   └── sample.txt
├── Project3
│   ├── graphic.gif
│   ├── image.gif
│   ├── log.txt
│   ├── picture.gif
│   └── project3.html
└── Project4
    ├── note.txt
    ├── project4.html
    ├── report.txt
    └── test.html
```

Figure 65 Adding special test files to validate file-based access restrictions across directories.

Configuring Apache for Directory Listing

To ensure directory contents are shown when accessed, Apache was configured with the +Indexes option for the /var/www/html_project1 directory. The mod_autoindex module is enabled by default on AlmaLinux.

```
root@server07 ~ $ httpd -M 2>/dev/null | grep autoindex
 autoindex_module (shared)
root@server07 ~ $
```

```
<Directory "/var/www/html_project1">
    Options +Indexes
    AllowOverride None
    Require all granted
</Directory>
```

Figure 66 Enabling directory listing by configuring Apache with Options +Indexes.

Blocking All .txt Files Globally

A global <FilesMatch> directive was added to deny access to any .txt file across the server, except for requests from the 192.168.50.0/24 subnet. This rule was placed outside any <Directory> block to apply universally.

```
<FilesMatch "\.txt$">
    <RequireAny>
        Require ip 192.168.50.0/24
        Require all denied
    </RequireAny>
</FilesMatch>
```

Figure 67 Global .txt file access restriction with an exception for 192.168.50.0/24.

Project1 – Subnet-Specific Access with File Restriction

Project1 is configured to be accessible from three subnets: 192.168.50.0/24, 10.35.16.0/24, and 10.35.17.0/24. However, files named secret.* must be blocked **only** for the 10.35.16.0/24 and 10.35.17.0/24 networks. This is achieved using a nested <FilesMatch> directive within a <RequireAll> block.

```
<Directory "/var/www/html_project1/Project1">
    <RequireAny>
        Require ip 192.168.50.0/24
        Require ip 10.35.16.0/24
        Require ip 10.35.17.0/24
    </RequireAny>
<FilesMatch "^secret\..*">
        <RequireAll>
            Require all granted
            Require not ip 10.35.16.0/24
            Require not ip 10.35.17.0/24
        </RequireAll>
    </FilesMatch>
</Directory>
```

Figure 68 Project1 allows access from three networks and denies secret. files to external subnets only.*

Project2 – Subnet Blacklist for Directory and File-Level Access

Project2 is accessible from all networks **except** 10.35.16.0/24, which is explicitly denied. The configuration combines <RequireAny> to allow listed networks, and <RequireAll> to apply a logical AND condition with a not clause for subnet denial.

```
<Directory "/var/www/html_project1/Project2">
    <RequireAny>
        Require ip 192.168.50.0/24
        Require ip 10.35.17.0/24
        Require ip 192.168.100.0/24
    </RequireAny>
<RequireAll>
        Require all granted
        Require not ip 10.35.16.0/24
    </RequireAll>
</Directory>
```

Figure 69 Project2 combines allow and deny directives to exclude only the 10.35.16.0/24 subnet while permitting access from other sources.

Project3 – Subnet-Based Access with .gif File Restriction

Project3 is configured to allow access from the local 192.168.50.0/24 and external 192.168.100.0/24 subnets. However, all .gif files must be **blocked only for the 192.168.100.0/24 network**, while still visible to the 50.0 subnet. This is done using a <FilesMatch> block with a Require not ip inside a <RequireAll> clause.

```
<Directory "/var/www/html_project1/Project3">
    <RequireAny>
        Require ip 192.168.50.0/24
        Require ip 192.168.100.0/24
    </RequireAny>
<FilesMatch "\.gif$">
        <RequireAll>
            Require all granted
            Require not ip 192.168.100.0/24
        </RequireAll>
    </FilesMatch>
</Directory>
```

Figure 70 Project3 configured to allow access from two networks, while .gif files are hidden from 192.168.100.0/24 only.

Project4 – Subnet Whitelist with test.html File Restriction

Project4 is accessible from the following three networks: 192.168.50.0/24, 10.35.16.0/24, and 192.168.100.0/24. However, one specific file — test.html — must be hidden from the 10.35.16.0/24 and 192.168.100.0/24 subnets. The configuration uses a <Files> directive for this filename, combined with a <RequireAll> block to exclude the restricted networks.

```
<Directory "/var/www/html_project1/Project4">
<RequireAny>
        Require ip 192.168.50.0/24
        Require ip 10.35.16.0/24
        Require ip 192.168.100.0/24
    </RequireAny>
<Files "test.html">
        <RequireAll>
            Require all granted
            Require not ip 10.35.16.0/24
            Require not ip 192.168.100.0/24
        </RequireAll>
    </Files>
</Directory>
```

Figure 71 Project4 permits all required subnets, while selectively blocking the test.html file from external networks.

Validation Task 3 - Accessibility (Ubuntu Client)

Task 3 - Project 1:

Project1 – Access from 192.168.50.0/24 (Allow)

To begin validating Project1, I simulated access from the 192.168.50.0/24 subnet, which is explicitly allowed in the Apache configuration using a <RequireAny> directive. Using my Ubuntu client with IP 192.168.50.20, I clicked the corresponding hyperlink to load the directory listing.

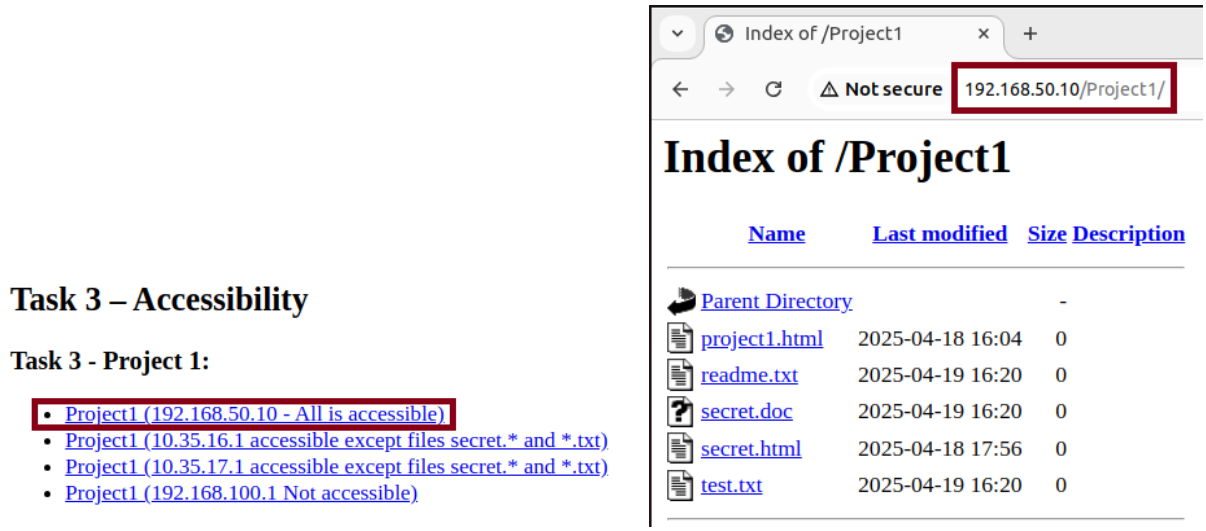


Figure 72 Accessing /Project1 from IP 192.168.50.20 (192.168.50.0/24) – All contents are visible.

Project1 – Access from 10.35.16.0/24 (Allow, with File Restrictions)

Next, I validated access from the 10.35.16.0/24 subnet using IP 10.35.16.2. The Apache configuration allows this subnet to access the directory, but denies files matching secret.*.

Figure 73 Accessing /Project1 from IP 10.35.16.2 – Directory listing is visible but secret.html and secret.doc are blocked.

Project1 – Access from 10.35.17.0/24 (Allow, with File Restrictions)

I continued validation from the 10.35.17.0/24 subnet using IP 10.35.17.2. Like the previous case, this subnet is allowed but should be restricted from accessing files named secret.*.

Figure 74 Accessing /Project1 from IP 10.35.17.2 – Directory is accessible; secret files are denied.

Project1 – Access from 192.168.100.0/24 (Deny)

Finally, I tested a restricted subnet by accessing Project1 from IP 192.168.100.1. According to the configuration, this subnet is **not listed** in the <RequireAny> block and should be completely denied.

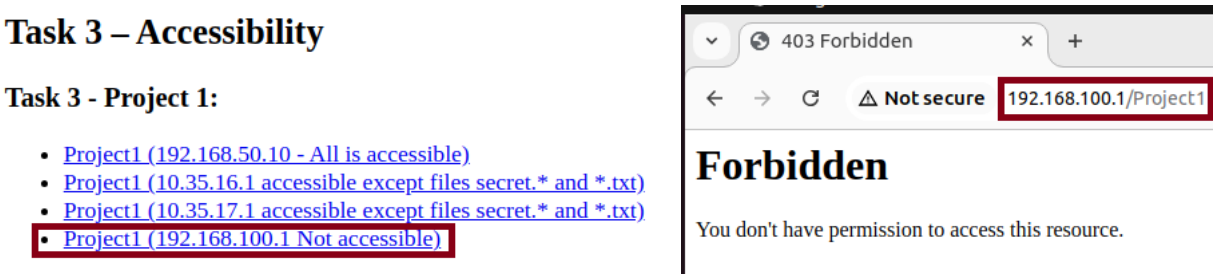


Figure 75 Accessing /Project1 from IP 192.168.100.1 results in a 403 Forbidden error.

Task 3 - Project 2:

Project2 – Access from 192.168.50.0/24 (Allow)

To begin validating Project2, I accessed the link from the local subnet 192.168.50.0/24 using the Ubuntu client with IP 192.168.50.20. This subnet is included in the <RequireAny> block and is allowed full access.

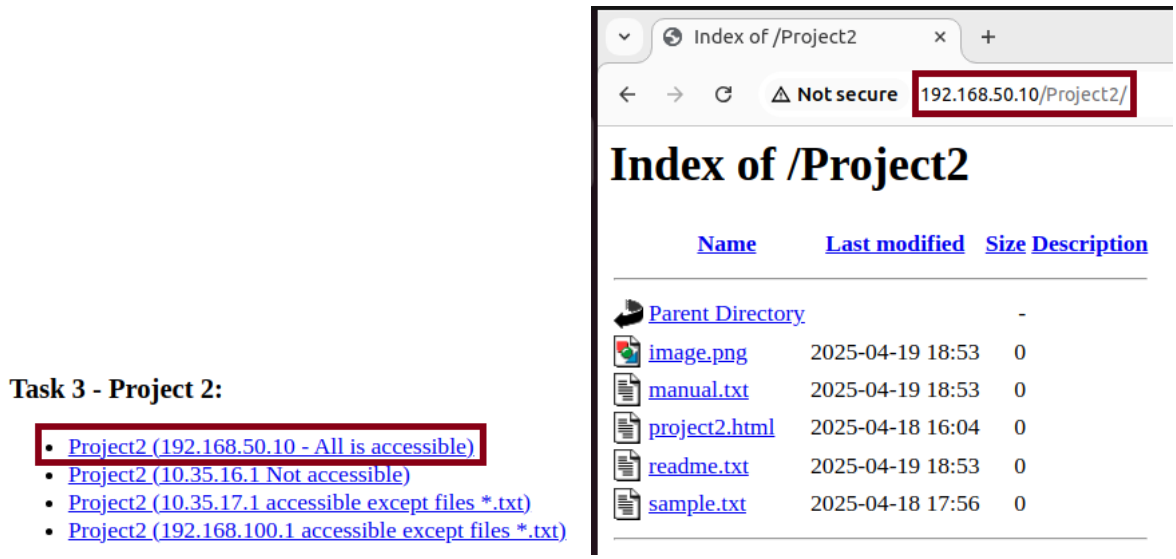


Figure 76 Accessing /Project2 from IP 192.168.50.20 – Directory listing and all files are visible.

Project2 – Access from 10.35.16.0/24 (Deny)

Next, I validated the behavior for subnet 10.35.16.0/24, which is **explicitly denied** using <RequireAll> and a Require not ip directive. Accessing from IP 10.35.16.2 correctly resulted in a denial.

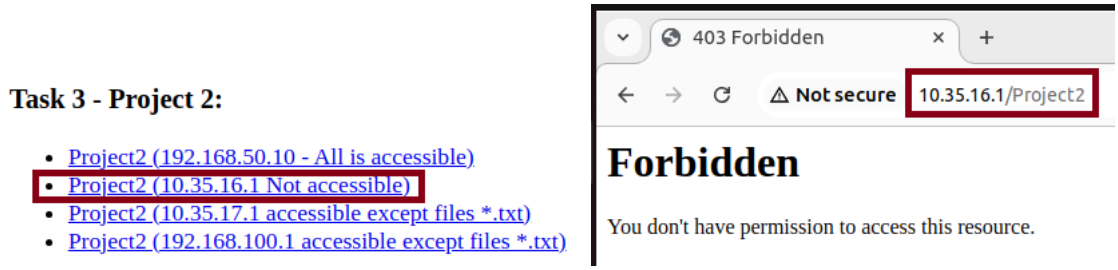


Figure 77 Accessing /Project2 from IP 10.35.16.2 – 403 Forbidden error is shown, as intended.

Project2 – Access from 10.35.17.0/24 (Allow, with File Restriction)

The subnet 10.35.17.0/24 is included in the <RequireAny> block and is granted access. However, global rules block .txt files for all networks **except** 192.168.50.0/24, so .txt files should be hidden.

Figure 78 Accessing /Project2 from IP 10.35.17.2 – Directory is accessible; .txt files are hidden.

Project2 – Access from 192.168.100.0/24 (Allow, with File Restriction)

Lastly, I accessed Project2 from the subnet 192.168.100.0/24 using IP 192.168.100.1. This subnet is allowed by the <RequireAny> rule, but like the previous case, .txt files are denied.

Figure 79 Accessing /Project2 from IP 192.168.100.1 – Directory is accessible; .txt files are not shown.

Task 3 - Project 3:

Project3 – Access from 192.168.50.0/24 (Allow)

Accessing /Project3 from the local subnet 192.168.50.0/24 using IP 192.168.50.20 is explicitly allowed by the <RequireAny> rule in the configuration. Directory contents are fully visible, including .gif files, since this subnet is not restricted by the <FilesMatch> block.

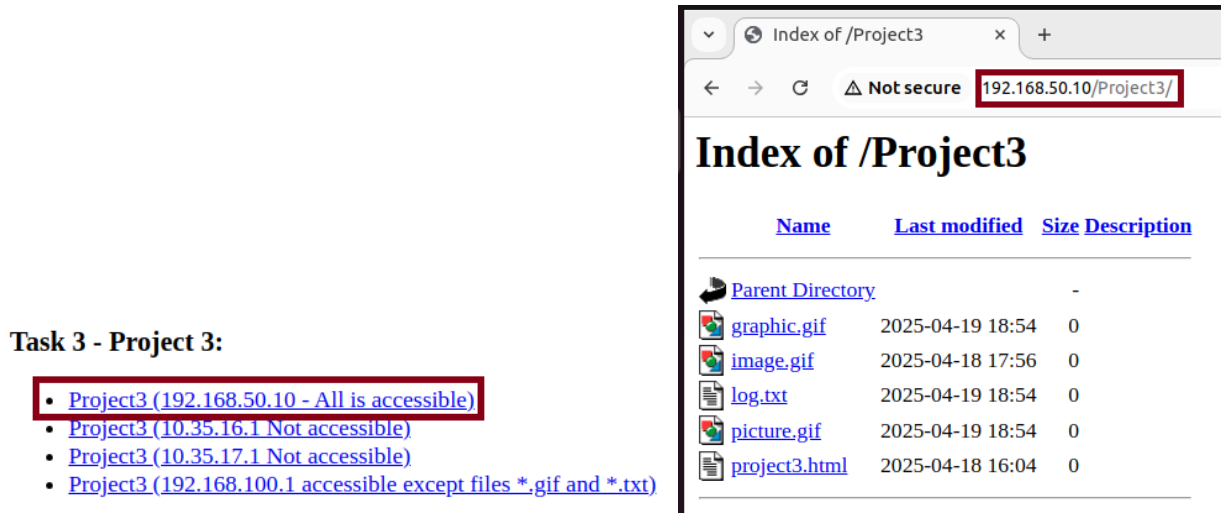


Figure 80 Accessing /Project3 from IP 192.168.50.20 – All files, including .gif, are accessible.

Project3 – Access from 10.35.16.0/24 (Deny)

Subnet 10.35.16.0/24 is not part of the allowed IP list for Project3. Accessing from IP 10.35.16.2 resulted in a **403 Forbidden**, confirming that access is properly denied.

Figure 81 Accessing /Project3 from IP 10.35.17.2 – Forbidden due to subnet restriction.

Project3 – Access from 10.35.17.0/24 (Deny)

Similarly, access from 10.35.17.0/24 is not allowed. When attempting to open the link using IP 10.35.17.2, Apache returned a **403 Forbidden** error.

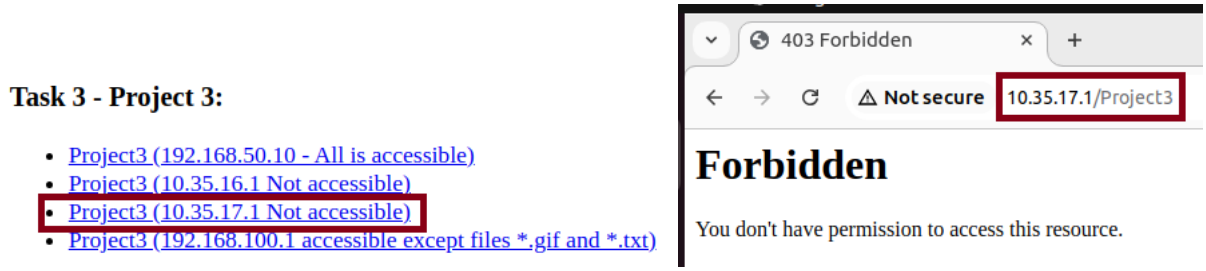


Figure 82 Accessing /Project3 from IP 10.35.17.2 – Forbidden due to subnet restriction.

Project3 – Access from 192.168.100.0/24 (Allow with File Restriction)

The subnet 192.168.100.0/24 is explicitly permitted. However, .gif files are denied using a <FilesMatch "\.gif$"> block with a Require not ip 192.168.100.0/24 rule. As expected, .gif files are **hidden**, while other files are visible.

Figure 83 Accessing /Project3 from IP 192.168.100.1 – Directory is visible, .gif files are not.

Task 3 - Project 4:

Project4 – Access from 192.168.50.0/24 (Allow All)

Access from the local subnet 192.168.50.0/24 is fully permitted. The Ubuntu client with IP 192.168.50.20 loaded all files in the /Project4 directory, including test.html, confirming unrestricted access.

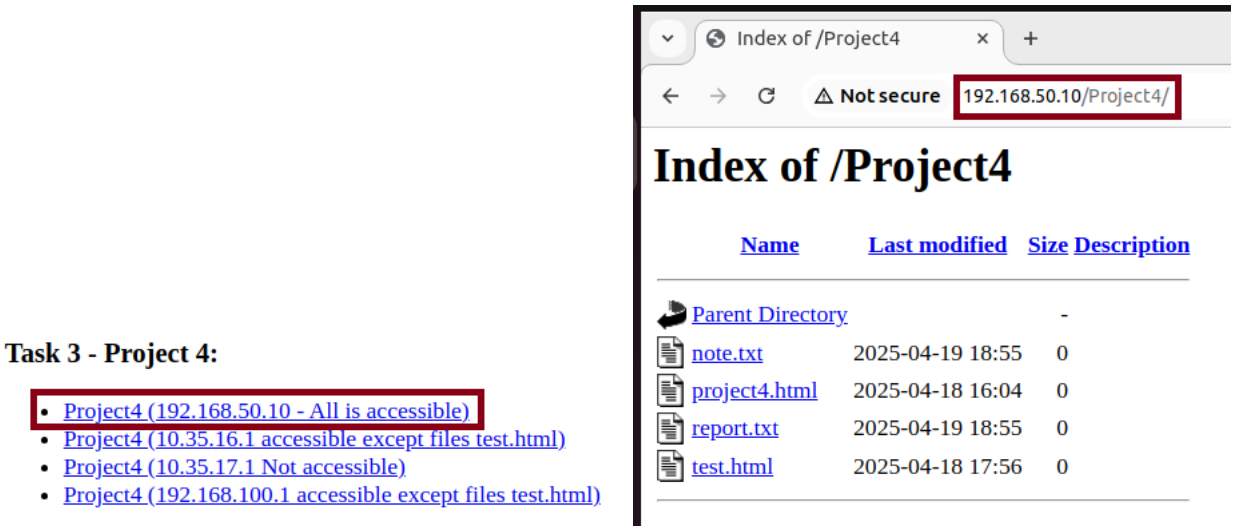


Figure 84 Accessing /Project4 from IP 192.168.50.20 – All files including test.html are accessible.

Project4 – Access from 10.35.16.0/24 (Allow with File Restriction)

This subnet is allowed by the <RequireAny> block. However, the Apache configuration contains a <Files "test.html"> restriction that denies access **only** to test.html for both 10.35.16.0/24 and 192.168.100.0/24. As expected, the directory is visible, but test.html does not appear.

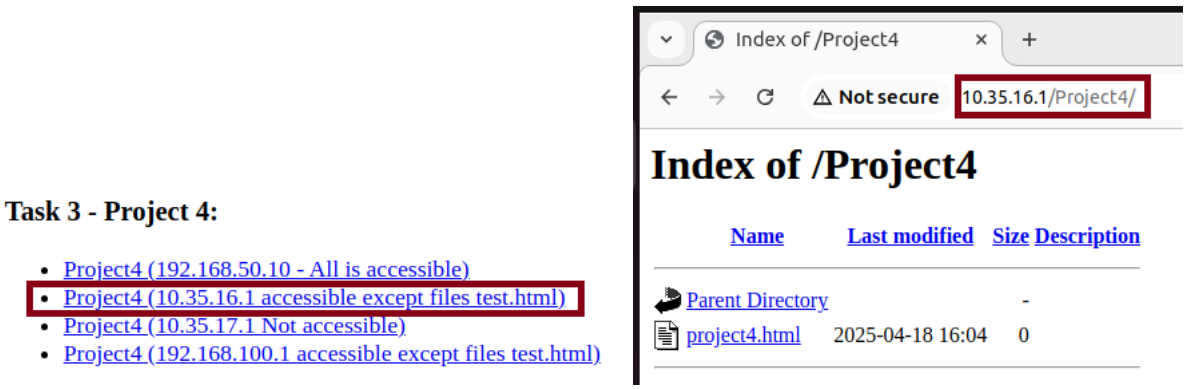


Figure 85 Accessing /Project4 from IP 10.35.16.2 – Directory loads, test.html is hidden.

Project4 – Access from 10.35.17.0/24 (Deny)

This subnet is **not included** in the <RequireAny> block for /Project4, and therefore access is completely denied. A **403 Forbidden** error is returned.

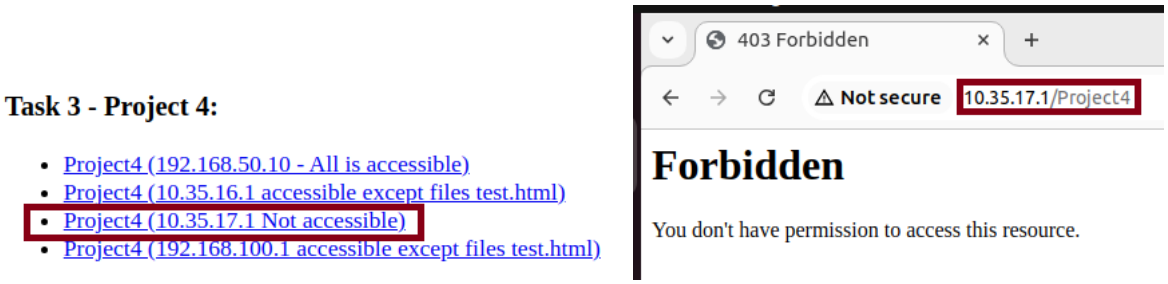


Figure 86 Accessing /Project4 from IP 10.35.17.2 – Access denied as expected.

Project4 – Access from 192.168.100.0/24 (Allow with File Restriction)

Subnet 192.168.100.0/24 is allowed in the <RequireAny> directive. However, as with 10.35.16.0/24, test.html is blocked by the <Files> rule. The Ubuntu client with IP 192.168.100.1 was able to see all files **except** test.html.

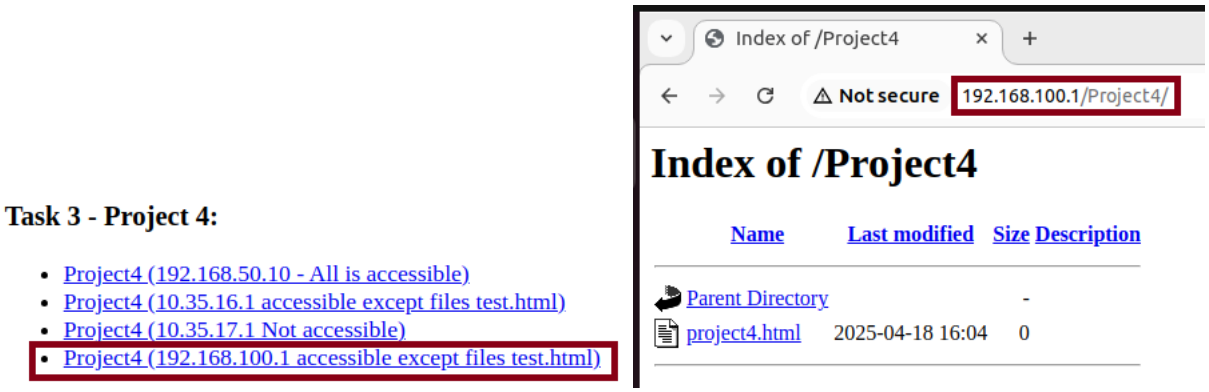


Figure 87 Accessing /Project4 from IP 192.168.100.1 – Directory accessible, test.html is blocked.

TASK 4 – AUTHORIZATION

Creating Departmental Directories and Placeholder Files

To begin Task 4, I created four department-specific directories under /var/www/htdocs: vendors, accountants, administrators, and programmers. Each directory was populated with placeholder content to validate access controls and restrictions based on subnet and file types.

```
root@server07 ~ $ mkdir -p /var/www/htdocs/{vendors,accountants,administrators,programmers}
```

```
/var/www/htdocs/
├── accountants
│   ├── accountants.html
│   ├── Photo1.gif
│   ├── Photo1.jpg
│   └── test.html
├── administrators
│   ├── administrators.html
│   ├── Photo1.gif
│   └── Photo1.jpg
├── programmers
│   ├── Photo1.gif
│   ├── Photo1.jpg
│   ├── programmers.html
│   ├── test.gif
│   └── test.jpg
└── vendors
    ├── doc.txt
    ├── Photo1.gif
    ├── Photo1.jpg
    └── vendors.html
```

Each directory contains:

- **An empty index.html file to allow directory listing.**
- **test.gif and test.jpg image files to test file-based access restrictions.**
- **A .html file named after the department for targeted access control testing.**

Figure 88 Creating and verifying directory structure and test files for authorization testing.

Setting Ownership and Permissions

To ensure Apache could access and serve these files correctly, I applied the following permissions, these settings provide read and execute access to the Apache service while maintaining system-level ownership:

```
root@server07 /var/www/htdocs $ sudo chown -R root:root /var/www/htdocs
sudo chmod -R 755 /var/www/htdocs
```

Figure 89 Setting permissions to allow directory access by Apache.

Configuring Aliases for Web Access

To make each department's directory accessible via a browser using friendly URLs, I created the following Alias directives in Apache:

```
Alias /vendors "/var/www/htdocs/vendors"
Alias /accountants "/var/www/htdocs/accountants"
Alias /administrators "/var/www/htdocs/administrators"
Alias /programmers "/var/www/htdocs/programmers"
```

Figure 90 Apache Alias directives for departmental websites.

Authorization: Apache Access Rules

To complete Task 4, we created web directories for each department under /var/www/htdocs and configured Apache access rules based on subnet and file type. The goal was to ensure that each group could only access permitted content according to project specifications.

General Configuration for /var/www/htdocs

To ensure that the department websites could be accessed properly via directory listing, I configured the /var/www/htdocs parent directory with the following settings:

- **“Options +Indexes”** enables directory listing so that files in each department’s folder are visible in the browser.
- **“AllowOverride None”** prevents the use of .htaccess files, centralizing all access control within the httpd.conf.
- **“Require all granted”** allows global access to the base directory, which is later restricted in each subdirectory based on subnet rules.

This base configuration was necessary to allow Apache to properly serve the subdirectories under /var/www/htdocs, where each department's site resides.

```
<Directory "/var/www/htdocs">
    Options +Indexes
    AllowOverride None
    Require all granted
</Directory>
```

Figure 91 Apache configuration enabling access and listing for /var/www/htdocs, the base directory for department websites.

Vendors Website Configuration

The vendors directory was configured to allow access only to specific subnets, meeting the rule: **“Vendors can access only their own website.”**

To meet this requirement:

- The **<RequireAny>** block allows access to the vendors’ website from the Vendors subnet 10.50.1.0/24, as well as from the Administrators 10.52.1.0/24 and Programmers 10.53.1.0/24, which aligns with the validation links that check shared access (while preventing access from Accountants 10.51.1.0/24).

Additionally, based on the project validation:

“Accessible to Programmers (10.53.1.0/24) but not *.gif and *.jpg files.”

- A **<FilesMatch>** directive is used to match .gif and .jpg files.
- Inside that block, access is explicitly allowed only from 10.50.1.0/24 (Vendors) and 10.52.1.0/24 (Administrators), effectively blocking access to image files for the Programmers’ subnet.

This satisfies the validation requirements where programmers can access the website, but not the .gif or .jpg files within it.

```
<Directory "/var/www/htdocs/vendors">
    <RequireAny>
        Require ip 10.50.1.0/24
        Require ip 10.52.1.0/24
        Require ip 10.53.1.0/24
    </RequireAny>

    <FilesMatch "\.(gif|jpg)$">
        <RequireAny>
            Require ip 10.50.1.0/24
            Require ip 10.52.1.0/24
        </RequireAny>
    </FilesMatch>
</Directory>
```

Figure 92 Access control for /vendors configured to restrict HTML media access from programmers while allowing full access to vendors and administrators.

Accountants Website Configuration

The accountants directory was configured to follow two key requirements:

“Accountants can access only the accountants' website and must not be able to view any *.html files.”
“Other departments like administrators and programmers should have partial access as described in the validations.”

Directory Access Options

To allow users to browse the contents of the directory without being redirected to an index page, and to prevent the use of .htaccess for overriding configurations, the following options were applied:

- Options +Indexes allows Apache to display a **file listing** of the directory if no homepage is loaded.
- AllowOverride None disables override rules from .htaccess to enforce centralized access control via httpd.conf.
- DirectoryIndex disabled ensures Apache does **not auto-load index.html**, which supports the project rule where accountants should not view HTML files.

```
<Directory "/var/www/htdocs/accountants">
    Options +Indexes
    AllowOverride None
    DirectoryIndex disabled

    <RequireAny>
        Require ip 10.51.1.0/24
        Require ip 10.52.1.0/24
        Require ip 10.53.1.0/24
    </RequireAny>

    <FilesMatch "\.html$">
        <RequireAll>
            Require all granted
            Require not ip 10.51.1.0/24
        </RequireAll>
    </FilesMatch>

    <FilesMatch "\.(gif|jpg)$">
        <RequireAny>
            Require ip 10.51.1.0/24
            Require ip 10.52.1.0/24
        </RequireAny>
    </FilesMatch>
</Directory>
```

Figure 93 Configuration for the /accountants directory using <FilesMatch> to restrict .html visibility for accountants and image file access for unauthorized clients. Directory options ensure visible file listings and centralized configuration.

Directory Access Rule

The **<RequireAny>** directive allows access to the accountants' directory from three specific subnets:

- 10.51.1.0/24 – Accountants
- 10.52.1.0/24 – Administrators
- 10.53.1.0/24 – Programmers

This ensures that multiple departments can access the directory contents depending on the access level applied to individual file types.

HTML Restriction for Accountants

The .html file restriction is implemented using **a <FilesMatch "\.html$">** block. Inside it:

- Require all granted permits access by default.
- Require not ip 10.51.1.0/24 denies .html files only to the Accountants subnet.

This satisfies the requirement that **accountants can access the directory but not its .html files**.

GIF/JPG Shared Access

A second **<FilesMatch>** block restricts access to .gif and .jpg files only to:

- 10.51.1.0/24 (Accountants)
- 10.52.1.0/24 (Administrators)

This aligns with validation requirements to block image files from other unauthorized subnets (like vendors).

Programmers Website Configuration

The programmers directory must fulfill this requirement:

"Programmers must be able to access their own website and to view all department websites except their *.gif or *.jpg files."

```
<Directory "/var/www/htdocs/programmers">
    <RequireAny>
        Require ip 10.52.1.0/24
        Require ip 10.53.1.0/24
    </RequireAny>
</Directory>
```

Figure 94 Apache configuration for the /programmers directory allowing access to programmers and administrators only.

Directory Access Rule

This rule is implemented using a <RequireAny> directive that allows access to the directory from:

- 10.53.1.0/24 – Programmers (primary access)
- 10.52.1.0/24 – Administrators (granted visibility to all departments)

This configuration enables both programmers and administrators to view the programmers' directory and its content, except for files restricted in other <FilesMatch> blocks applied elsewhere.

Note: This specific <Directory> block does **not** contain any <FilesMatch> directive, because **the restrictions for programmers apply to other department websites** (e.g., vendors, accountants), not to their own directory.

Thus, no .gif or .jpg restriction is necessary inside the programmers directory itself.

Administrators Website Configuration

This directory must satisfy the following rule:

"Administrators must be able to view all department websites. Programmers can access all websites except *.gif and *.jpg files."

Directory Access Rule

The <RequireAny> block allows access from:

- 10.52.1.0/24 – Administrators
- 10.53.1.0/24 – Programmers

This satisfies the rule that both groups can access the administrators' website.

File-Specific Restriction

To restrict image file types (.gif and .jpg) **only from programmers**, we use a <FilesMatch> directive with:

```
<RequireAll>
    Require all granted
    Require not ip 10.53.1.0/24
</RequireAll>
```

This means:

- Everyone is granted access,
- Except clients from the 10.53.1.0/24 subnet (programmers) who are denied access **only to image files**.

This allows administrators full access while blocking image files from programmers as required.

```
<Directory "/var/www/htdocs/administrators">
    <RequireAny>
        Require ip 10.52.1.0/24
        Require ip 10.53.1.0/24
    </RequireAny>

    <FilesMatch "\.(gif|jpg)$">
        <RequireAll>
            Require all granted
            Require not ip 10.53.1.0/24
        </RequireAll>
    </FilesMatch>
</Directory>
```

Figure 95 Apache configuration for the /administrators directory, restricting .gif and .jpg access from programmers but allowing full access to administrators.

Validation Task 4 - Authorization (Ubuntu Client)

Validations: Vendors Website

vendors “access from 10.50.1.0/24 (allowed)”

To validate access for the Vendors department, I configured the Ubuntu client with IP 10.50.1.2, which belongs to the 10.50.1.0/24 subnet. I opened a web browser and navigated to: **“http://10.50.1.1/vendors/”**

The directory listing was displayed, showing the contents of /var/www/htdocs/vendors. This confirms that vendors can access their own website, as intended by the project requirements.

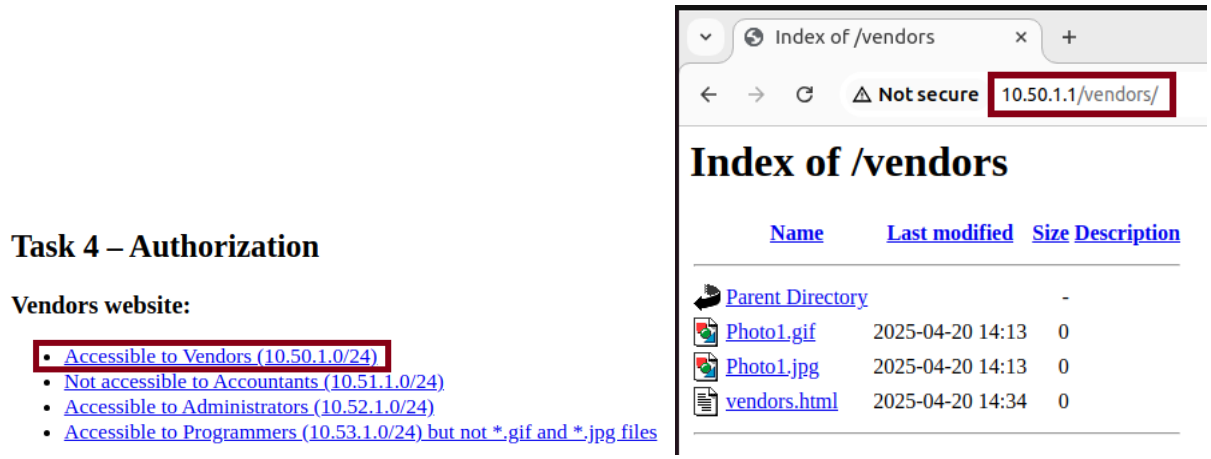


Figure 96 Accessing /vendors/ from the Vendors subnet (10.50.1.0/24).

vendors “access from 10.51.1.0/24 (not allowed)”

Next, I changed the Ubuntu client's IP to 10.51.1.2 (Accountants subnet) and tried to access the same URL: **“http://10.50.1.1/vendors/”**

This resulted in a **403 Forbidden** error, verifying that access is blocked for clients outside the Vendors subnet.

Figure 97 403 Forbidden when accessing /vendors/ from 10.51.1.0/24 (Accountants).

vendors “access from 10.52.1.0/24 (Administrators – allowed)”

To validate that **administrators** can access the vendors’ site, I assigned the Ubuntu client the IP address 10.52.1.2 (Administrators subnet) and opened: **http://10.50.1.1/vendors/**

The directory listing loaded successfully, and I was able to view all files without restriction. This confirms that administrators are correctly allowed full access to the vendors' website.

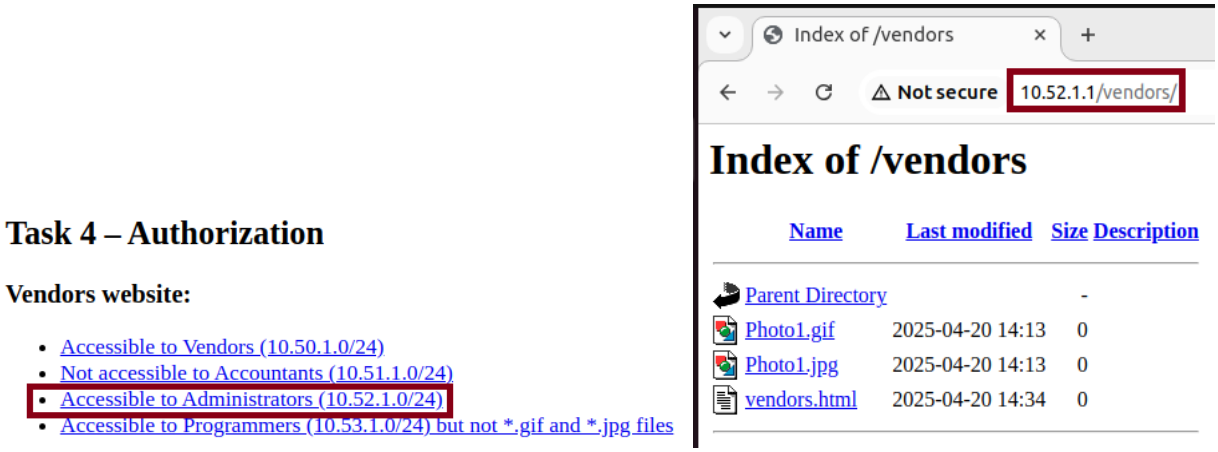


Figure 98 Administrators (10.52.1.0/24) successfully accessed the vendors’ website and all content.

vendors “access from 10.53.1.0/24 (Programmers – allowed, but not *.gif or *.jpg)”

Next, I changed the client IP to 10.53.1.2 (Programmers subnet) and accessed the same site: **http://10.50.1.1/vendors/**

The directory listing loaded correctly. All non-image files were accessible, but when attempting to open .gif or .jpg files, Apache returned a **403 Forbidden** error. This confirms that the restriction for programmers is correctly applied.

Figure 99 Programmers (10.53.1.0/24) accessed vendors’ website but were blocked from .gif and .jpg files.

Validation: Accountants Website

accountants “Not accessible to Vendors (10.50.1.0/24)”

To test this restriction, I set the Ubuntu client’s IP to 10.50.1.2 (Vendors subnet) and accessed: **http://10.51.1.1/accountants/**

The browser returned a **403 Forbidden** message. This confirms that **vendors are completely denied access** to the accountants’ directory, as required by the access rules.

Figure 100 Access to /accountants denied for the vendors subnet (10.50.1.0/24).

accountants “Accessible to Accountants (10.51.1.0/24) but not .html files”

With the client IP set to 10.51.1.2, I accessed the accountants’ URL. The directory loaded successfully, but all .html files were not visible in the listing directory.

This behavior validates that **accountants can access the directory** but **cannot view .html content**, as required.

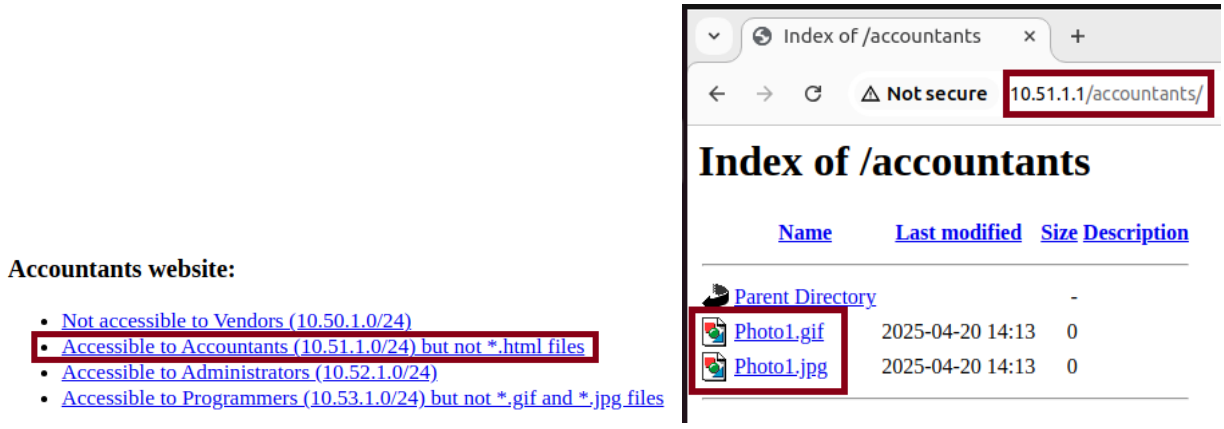


Figure 101 Accountants subnet (10.51.1.0/24) can browse the directory but is denied access to .html files.

accountants “Accessible to Administrators (10.52.1.0/24)”

Using IP 10.52.1.2, I verified that administrators have **full access** to the /accountants directory and can open all file types, including .html, .gif, and .jpg.

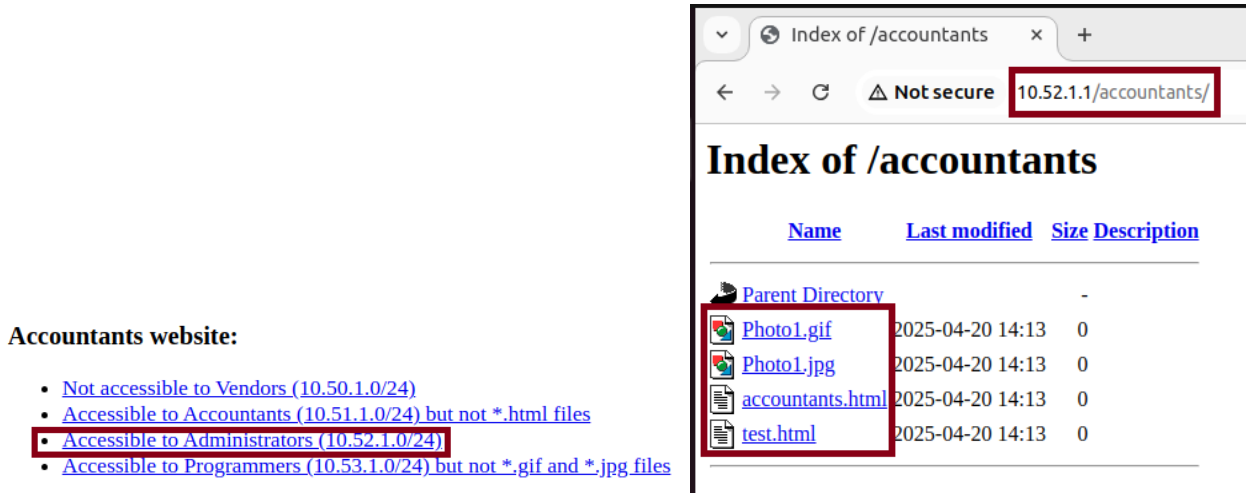


Figure 102 Administrators (10.52.1.0/24) successfully accessed all content in the /accountants directory.

accountants “Accessible to Programmers (10.53.1.0/24) but not .gif and .jpg files”

With the client set to 10.53.1.2, I confirmed that **programmers can access the directory** and view .html files. However,.gif or .jpg files were not visible within the listing directory, confirming that **image file access is properly blocked**.

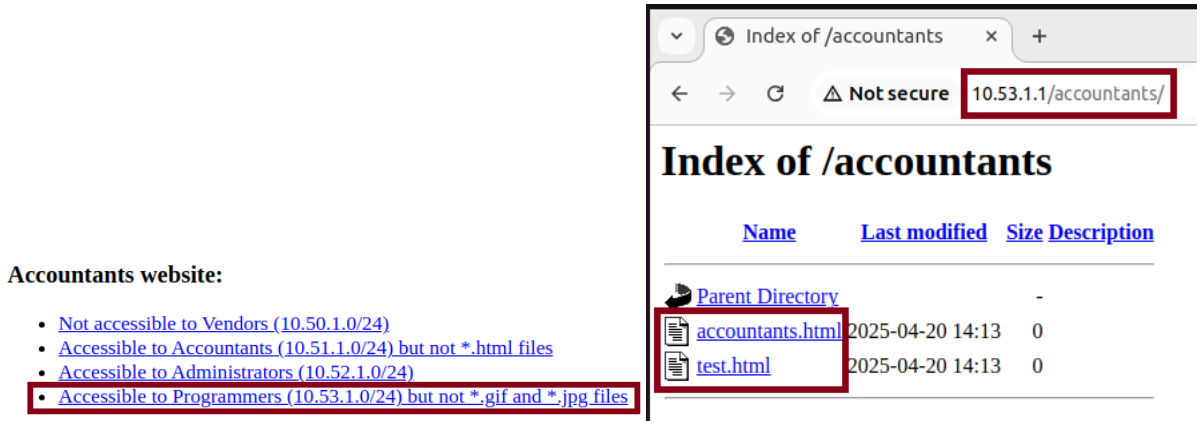


Figure 103 Programmers (10.53.1.0/24) accessed /accountants but are restricted from viewing image files.

Validation: Programmers Website

programmers “Not accessible to Vendors (10.50.1.0/24)”

With the Ubuntu client configured to IP 10.50.1.2, I accessed the programmers’ alias: http://10.53.1.1/programmers/

The page returned a **403 Forbidden** error, confirming that clients from the **vendors subnet (10.50.1.0/24)** cannot access the programmers’ directory.

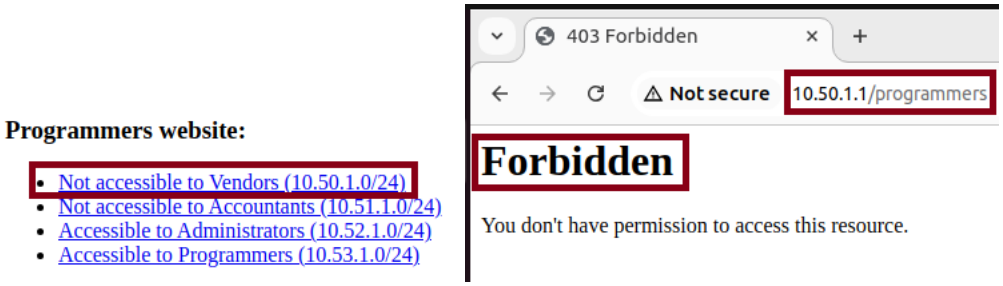


Figure 104 Vendors subnet denied access to /programmers (as expected).

programmers “Not accessible to Accountants (10.51.1.0/24)”

I changed the IP address to 10.51.1.2 (Accountants subnet) and retried accessing the programmers’ site. The directory was **inaccessible**, resulting in a **403 Forbidden**, as expected by the rule that blocks all access to accountants.

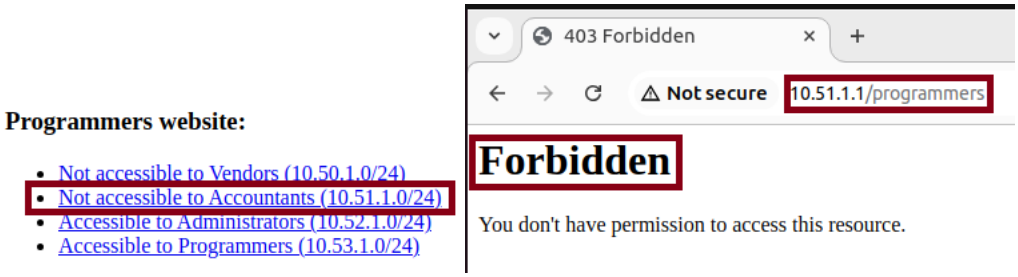


Figure 105 Access from accountants (10.51.1.0/24) denied for /programmers.

programmers “Accessible to Administrators (10.52.1.0/24)”

I set the client IP to 10.52.1.20 and accessed /programmers. The full directory listing was visible, and **all file types were accessible**, including .html, .gif, and .jpg.

Figure 106 Administrators (10.52.1.0/24) successfully accessing /programmers and all its contents.

programmers “Accessible to Programmers (10.53.1.0/24) but not .gif and .jpg files”

From the client IP 10.53.1.2, I accessed the directory. The programmers could view the directory listing and **access most content**, but **files ending in .gif or .jpg were completely hidden** from the listing. They **did not appear at all**, meaning Apache's access rules **prevented even listing them**.

This confirms that access to the directory is granted, while .gif and .jpg files are effectively hidden for this subnet.

Figure 107 Programmers (10.53.1.0/24) can access /programmers but image files are not listed or accessible.

Validation: Administrators Website

administrators “Not accessible to Vendors (10.50.1.0/24)”

While using an Ubuntu client set to IP 10.50.1.2 from the vendors subnet, I attempted to access: **http://10.52.1.1/administrators/**
The request returned a 403 Forbidden error, confirming that vendors are denied access to the administrators' directory.

Figure 108 Vendors subnet correctly denied access to /administrators.

administrators “Not accessible to Accountants (10.51.1.0/24)”

Switching to IP 10.51.1.2, I verified that **accountants also receive a 403 Forbidden** when attempting to access /administrators. This aligns with the rule to deny access to anyone outside the specified subnets.

Figure 109 Accountants (10.51.1.0/24) correctly blocked from administrators’ website.

administrators “Accessible to Administrators (10.52.1.0/24)”

With client IP 10.52.1.20, the administrators’ alias loaded correctly, and **all files** (index.html, test.gif, test.jpg, etc.) were **visible and accessible**.

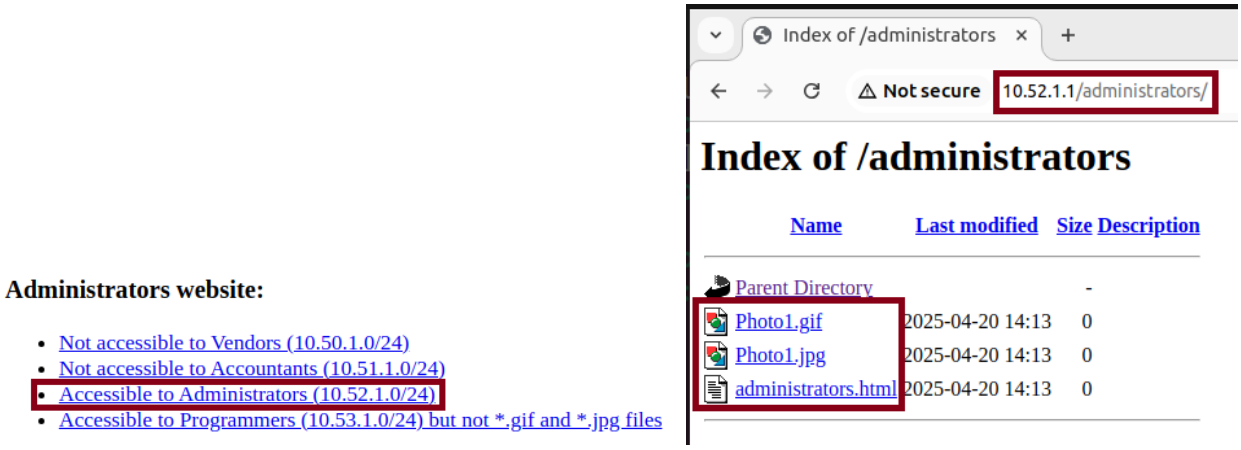


Figure 110 Full directory listing and file access confirmed for 10.52.1.0/24.

administrators “Accessible to Programmers (10.53.1.0/24) but not *.gif and .jpg files”*

From the IP 10.53.1.20 (programmers subnet), the directory opened successfully. However, .gif and .jpg image files were **not listed** at all due to Apache’s access restriction. This confirmed that programmers can access the site, but **image files are hidden** and inaccessible from their subnet.

Figure 111 Programmers subnet (10.53.1.0/24) accessing /administrators – image files not visible.

“Log Evidence”

To strengthen the evidence of successful access control implementation, select Apache access log entries were captured to support each task in the project. Below are four validation scenarios, each aligned to a specific task and requirement. All logs are sourced from /var/log/httpd/access_log.

Validation: Task 1 (Homepage Access)

To verify that the Apache homepage was accessible from the 192.168.50.0/24 subnet, a GET request was made from IP 192.168.50.20 to /. The access log shows a **200 OK** response.

```
192.168.50.20 - - [20/Apr/2025:18:21:15 -0400] "GET / HTTP/1.1" 200 5287 "-" "Mozilla/5.0 (X11; Linux x86_64) Apple
WebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36"
192.168.50.20 - - [20/Apr/2025:18:22:07 -0400] "-" 408 - "-" "-"
root@server07 /etc/httpd/conf $
```

Figure 112 Figure 105 Access to homepage (index.html) from subnet 192.168.50.0/24.

Validation: Task 2 (Authentication to secure1)

Access to the secure1 directory was tested. The first entry shows a denied request (401 Unauthorized), followed by a successful login (200 OK) by user01 from 192.168.50.20.

```
root@server07 /etc/httpd/conf $ grep "/secure1" /var/log/httpd/access_log
192.168.50.20 - - [20/Apr/2025:18:35:23 -0400] "GET /secure1 HTTP/1.1" 401 381 "http://192.168.50.10/" "Mozilla/5.0
 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36"
192.168.50.20 - user01 [20/Apr/2025:18:35:28 -0400] "GET /secure1 HTTP/1.1" 301 237 "http://192.168.50.10/" "Mozill
a/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36"
192.168.50.20 - user01 [20/Apr/2025:18:35:28 -0400] "GET /secure1/ HTTP/1.1" 200 134 "http://192.168.50.10/" "Mozil
la/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36"
root@server07 /etc/httpd/conf $
```

Figure 113 Successful and failed authentication attempts to /secure1 from subnet 192.168.50.0/24.

Validation – Task 3 (Project1 restriction)

Project1 access was denied to the 192.168.100.0/24 subnet. The log indicates **403 Forbidden** errors when 192.168.100.2 attempted access. This confirms that subnet-based restrictions were enforced correctly.

```
root@server07 /etc/httpd/conf $ grep "/Project1" /var/log/httpd/access_log | grep "192.168.100"
192.168.100.2 - - [20/Apr/2025:15:12:46 -0400] "GET /Project1 HTTP/1.1" 403 199 "http://192.168.50.10/" "Mozilla/5.
0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36"
192.168.100.2 - - [20/Apr/2025:15:12:46 -0400] "GET /favicon.ico HTTP/1.1" 404 196 "http://192.168.100.1/Project1"
"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36"
192.168.100.2 - - [20/Apr/2025:18:36:13 -0400] "GET /Project1 HTTP/1.1" 403 199 "http://192.168.50.10/" "Mozilla/5.
0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36"
root@server07 /etc/httpd/conf $
```

Figure 114 Denied access to Project1 from unauthorized subnet 192.168.100.0/24.

Validation: Task 4 (Restricted file visibility for programmers)

From subnet 10.53.1.0/24, a GET request was made to /administrators. The log shows a successful 200 OK response for the directory listing, but .gif and .jpg files were not listed or fetched—demonstrating correct file-based restriction as required by Task 4.

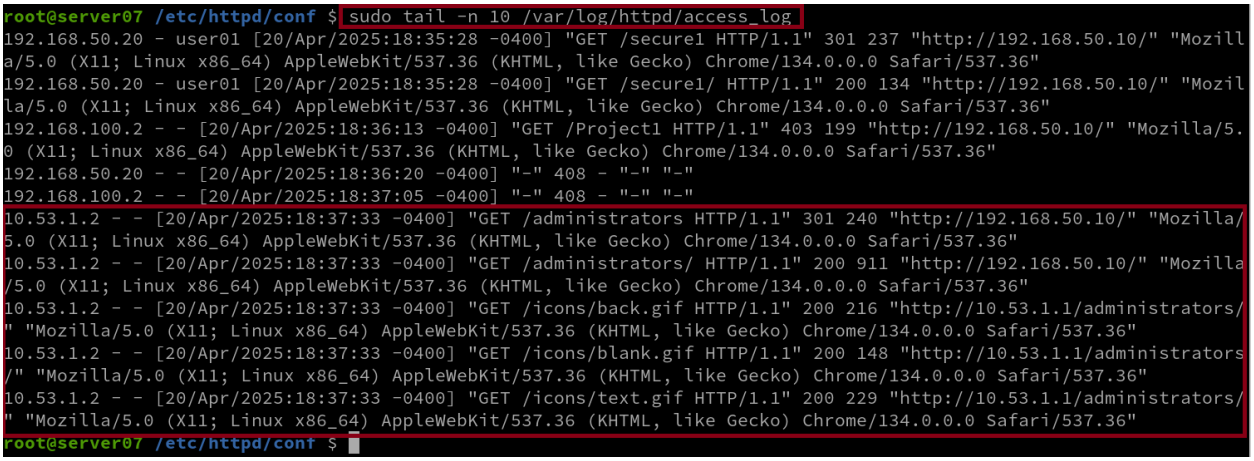


Figure 115 Directory listing allowed for 10.53.1.0/24 (programmers) under /administrators, while .gif and .jpg files are not visible or accessed.

Conclusion

This project provided a hands-on opportunity to learn how to install, configure, and secure an Apache web server in a real-world scenario. By working through each task, I gained a deeper understanding of how directory permissions, authentication, subnet restrictions, and file-level access control can be used to protect web content and limit visibility based on user roles or network location.

- Throughout the project, I learned how to:
- Set up a custom DocumentRoot and homepage for a website.
- Create secure directories with user- and subnet-based access using <Directory>, <RequireAll>, and .htaccess files.
- Implement restrictions based on file types and client networks to control visibility and enforce security policies.
- Use Apache logs to validate and confirm access behavior in different testing scenarios.

Each configuration was tested and validated successfully using multiple client IPs, ensuring the rules worked exactly as intended. This project helped me apply Apache directives in a meaningful way and better understand how access control works in network and web server environments.