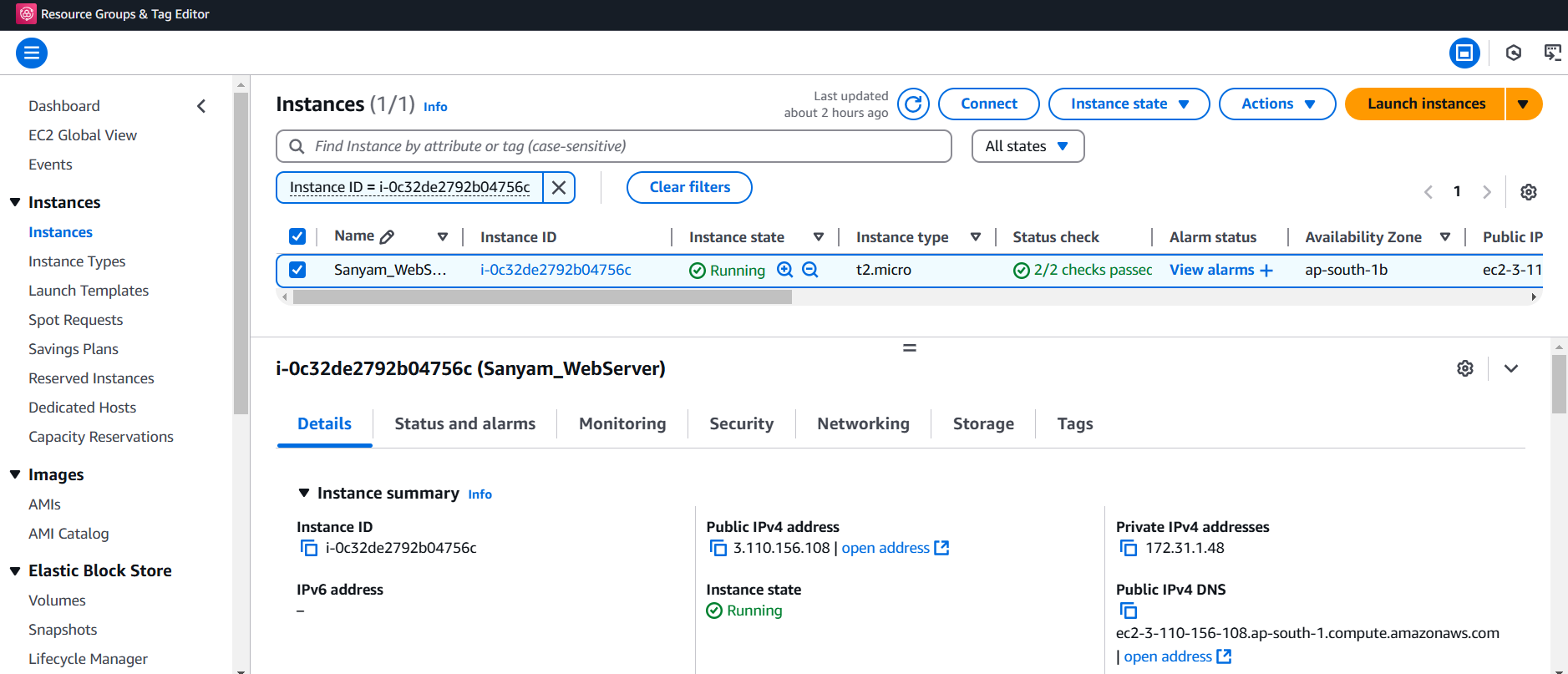
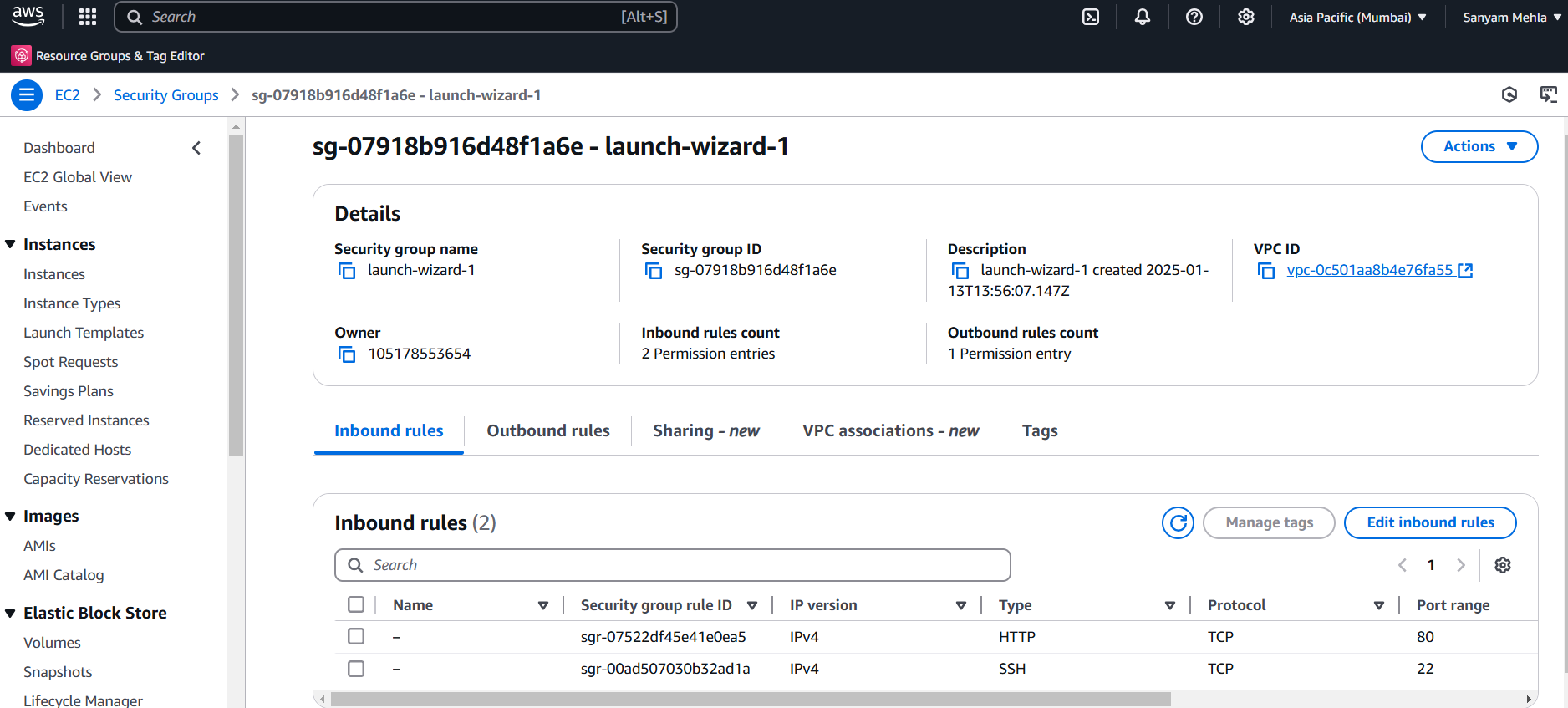
**Demonstrating and Mitigating SQL Injection in a Web Application**

**Part 1:**

**1. Spin up an EC2 instance and install Nginx / Apache**

I launched an EC2 instance to host the nginx web application, configured as follows:

* **AMI**: Ubuntu 24.04
* **Instance Type**: t2.micro
* **Region**: ap-south-1 (Asia Pacific - Mumbai)
* **Key Pair**: \*\*\*\*\*\*.pem (used for secure EC2 connection)
* **Public IP**: 3.110.156.108
* **Security Groups**:
  + Inbound **SSH (port 22)** allowed from 0.0.0.0/0 for remote access using SSH Instance Connect.
  + Inbound **HTTP (port 80)** allowed from 0.0.0.0/0 to serve web traffic.

**Installing Nginx**

To set up the web server, I installed **Nginx** using the following command:

**sudo apt-get install nginx**

After installation, I verified the setup by visiting the EC2 public IP in the browser: http://3.110.156.108/. The default Nginx welcome page loaded successfully, confirming the server's proper functioning.  
  


**2. Create a login form with 1 input field & a submit button**

I created a simple login form using HTML, saved as page1.html:

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>Page 1 - SQL Injection affected Login Form</title>**

**</head>**

**<body>**

**<h1>Page 1 - Vulnerable Login Form</h1>**

**<form method="GET" action="#">**

**<label for="username">Username:</label>**

**<input type="text" id="username" name="username">**

**<button type="submit">Login</button>**

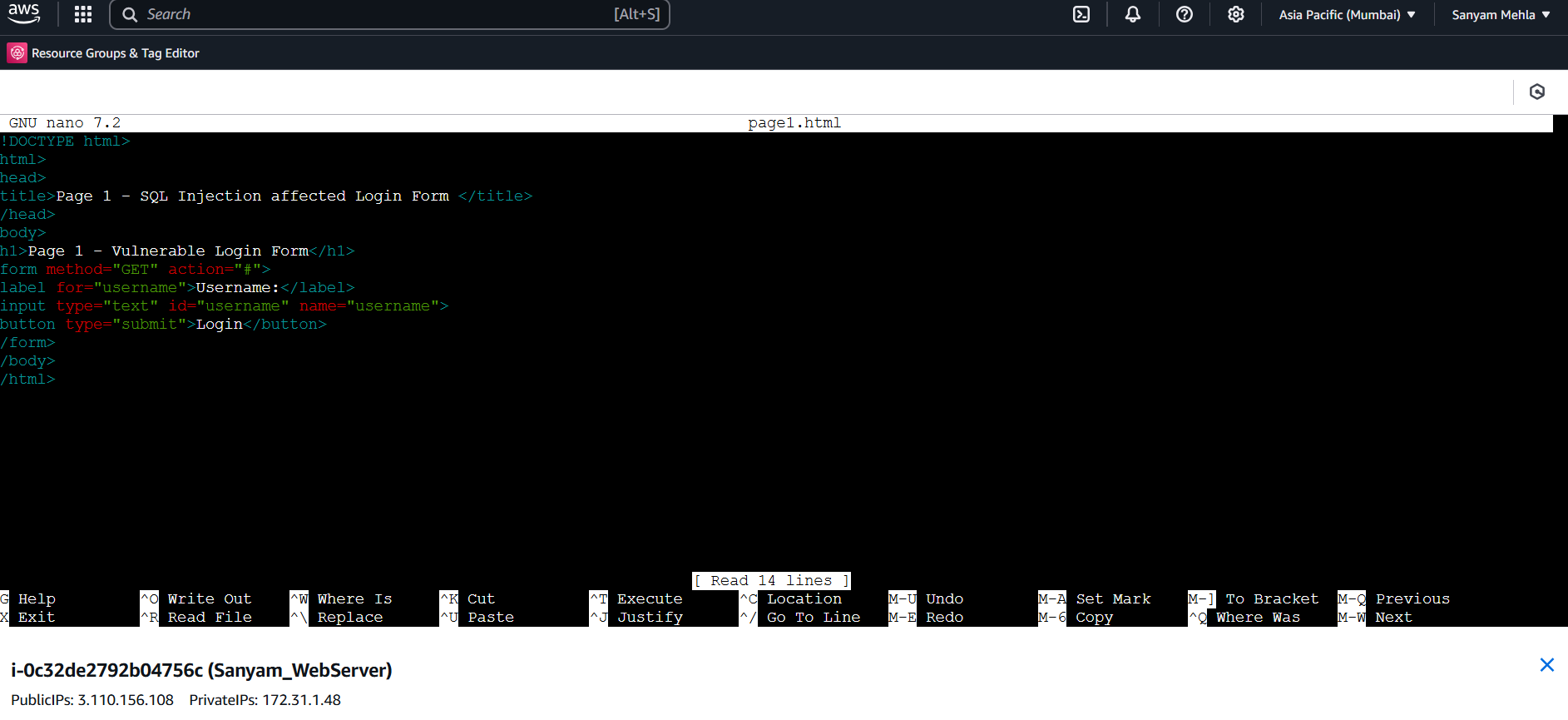
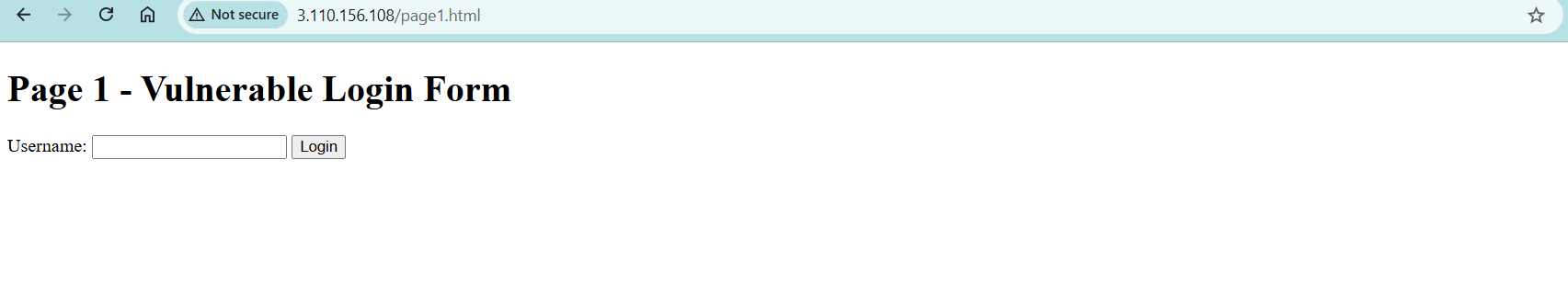
**</form>**

**</body>**

**</html>**

The form was created using the nano editor on the EC2 instance:

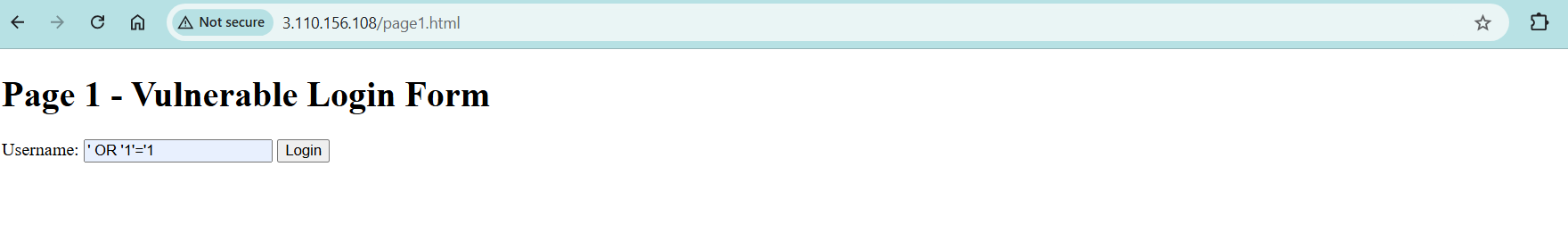
**sudo nano /var/www/html/page1.html**

The form is intentionally simple, without any input validation or sanitization, making it vulnerable to SQL injection.  
  
  


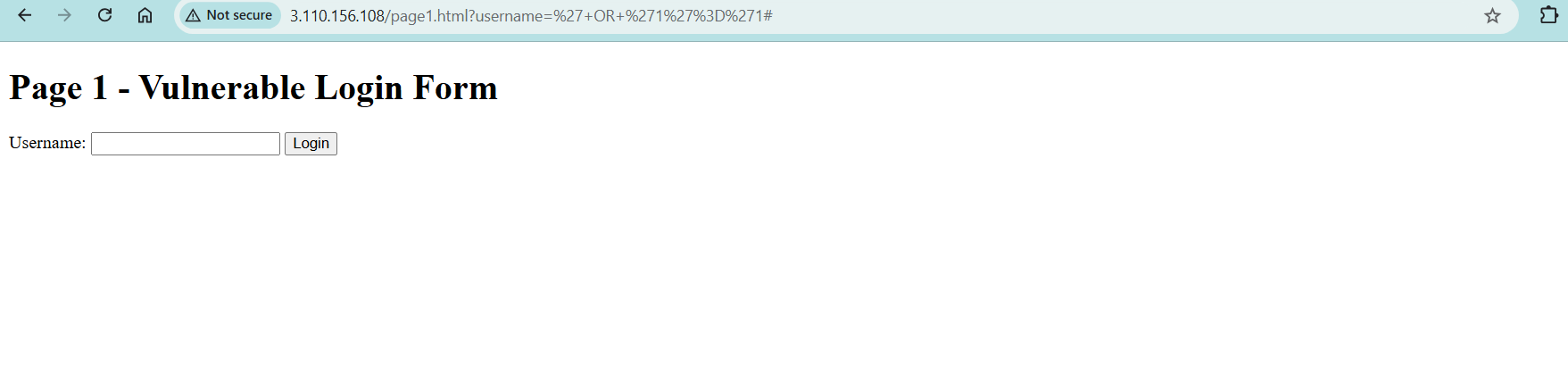
**3. The form should be exploitable via SQL injection attack**

The login form was tested to demonstrate the SQL injection vulnerability:

* **Before Submission**: Entering a malicious payload like ' OR 1=1 -- in the username field.



* **After Submission**: Upon submitting the payload, the SQL injection successfully executed, and the URL directly displayed the user-provided input, confirming the vulnerability without showing any errors.



**Part 2**

1. **Mitigate the SQL injection attack by developing the rules in ModSecurity WAF**

To secure the application, I configured **ModSecurity** on the Nginx web server. This was done using guidance from the following resources:

* [Linode Documentation: Securing Nginx with ModSecurity](https://www.linode.com/docs/guides/securing-nginx-with-modsecurity/)
* [Core Rule Set (CRS) Documentation](https://github.com/coreruleset/coreruleset)

**Key Steps**:

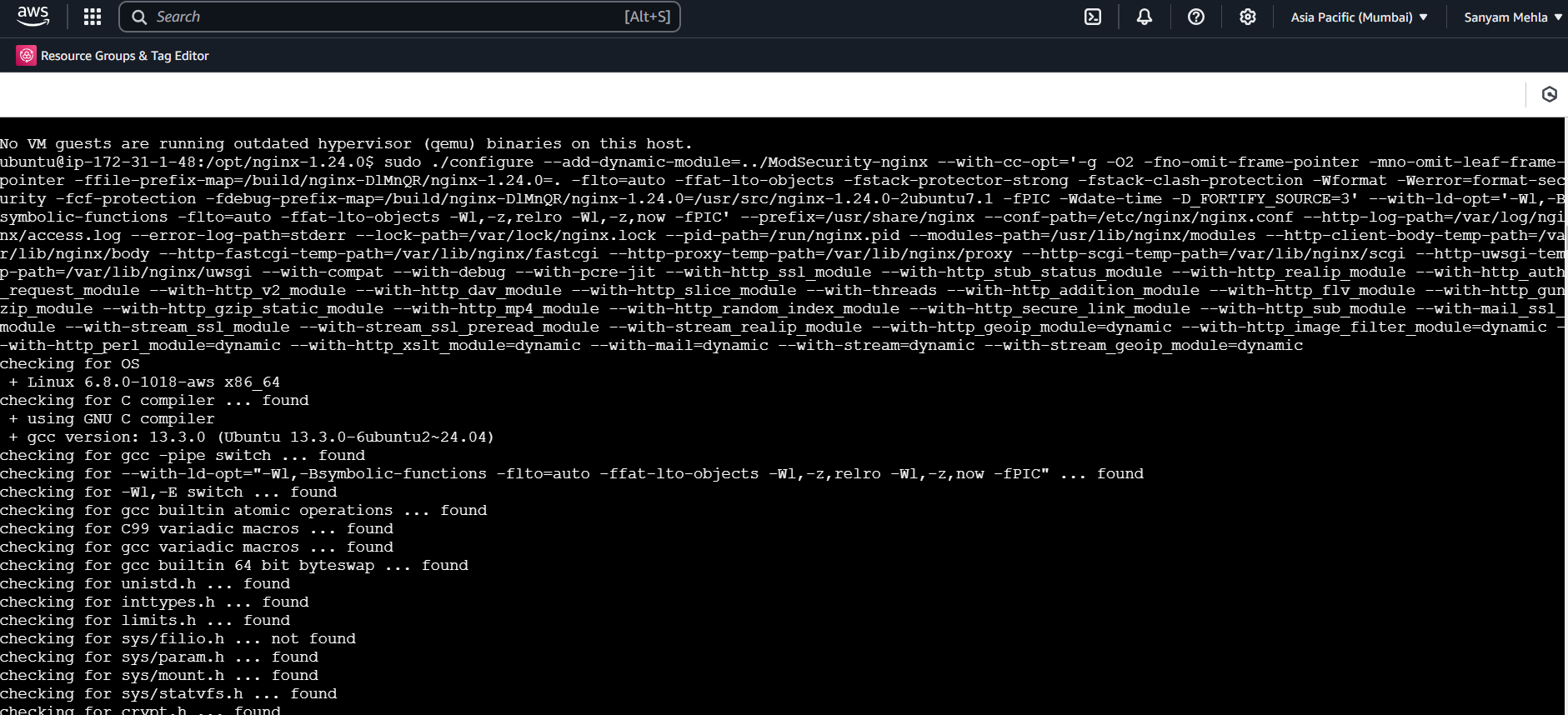
Here are the updated key steps for installing and configuring ModSecurity on an Nginx server based on the provided references:

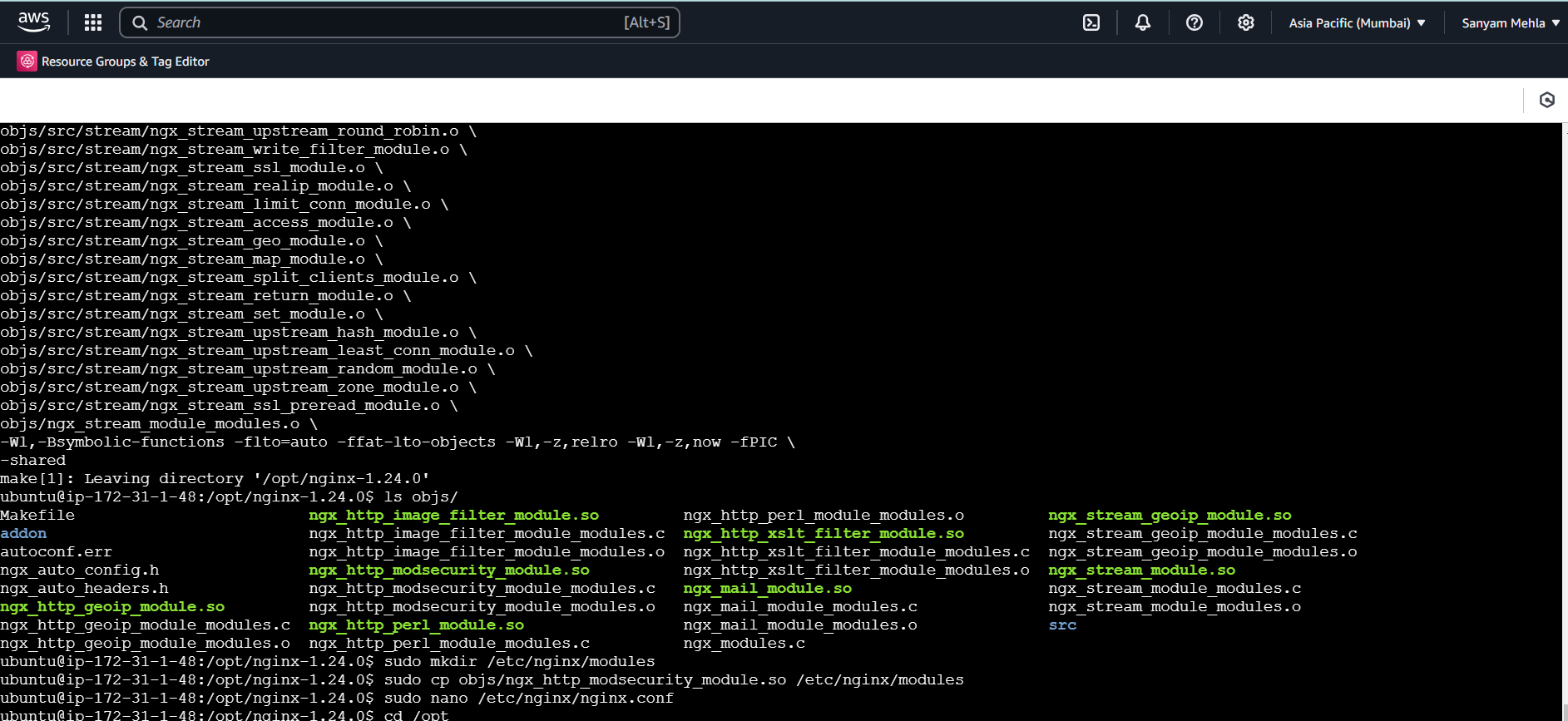
**Key Steps:**

1. **Install Dependencies**:
   * Installed necessary libraries and tools(have shown only few in below command) for ModSecurity:

sudo apt-get install libmodsecurity3 libmodsecurity-dev build-essential git

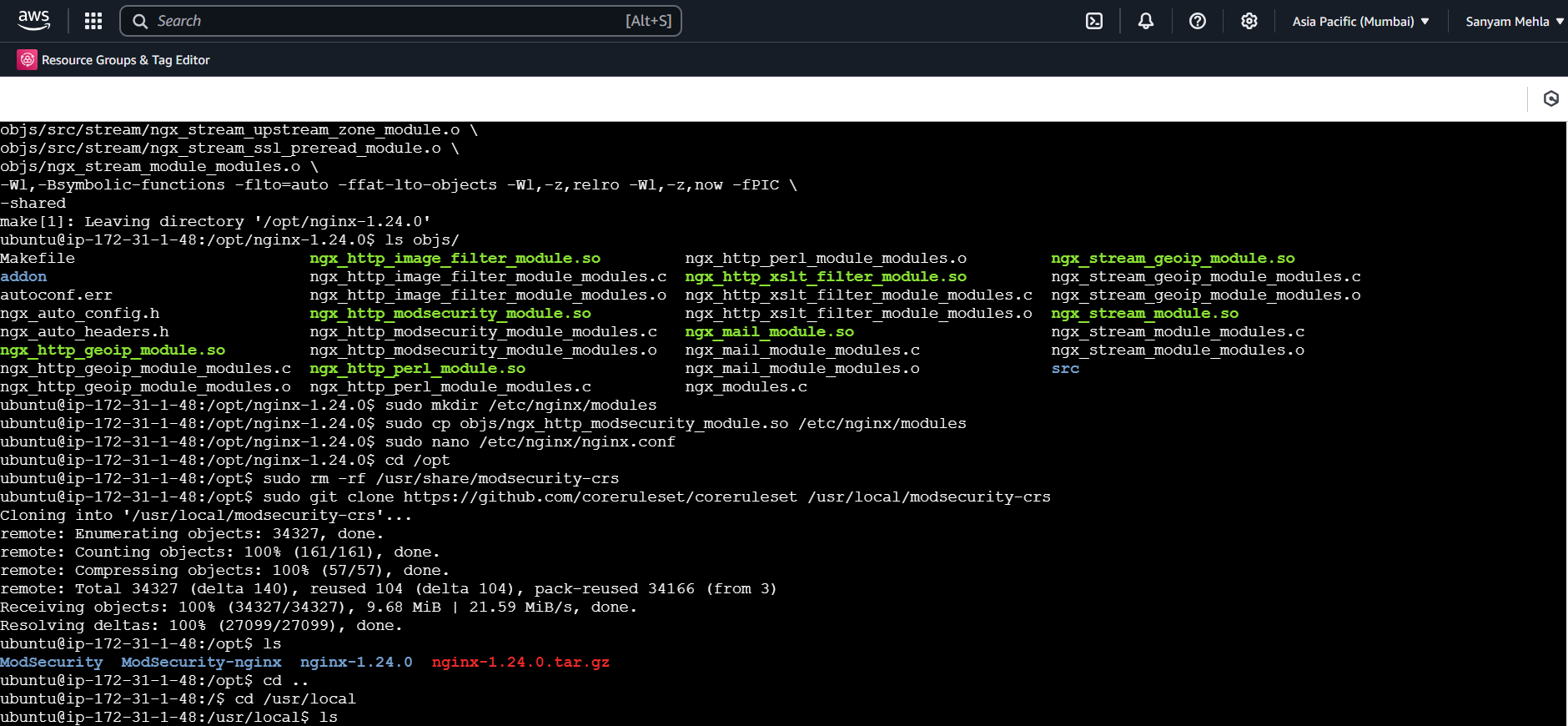
1. **Clone and Build ModSecurity Nginx Connector**:
   * Cloned the ModSecurity Nginx connector repository:
   * sudo git clone --depth 1 https://github.com/SpiderLabs/ModSecurity-nginx.git
2. **Integrate ModSecurity with Nginx**:
   * Installed and built Nginx with ModSecurity support (using Nginx source and the connector):
     + Followed the build process as described in the [SpiderLabs documentation](https://github.com/SpiderLabs/ModSecurity-nginx).

****

****

1. **Download and Configure OWASP Core Rule Set (CRS)**:
   * Cloned the OWASP CRS repository:

**git clone https://github.com/coreruleset/coreruleset.git /etc/nginx/modsecurity-crs**

****

* + Linked the default CRS setup file:

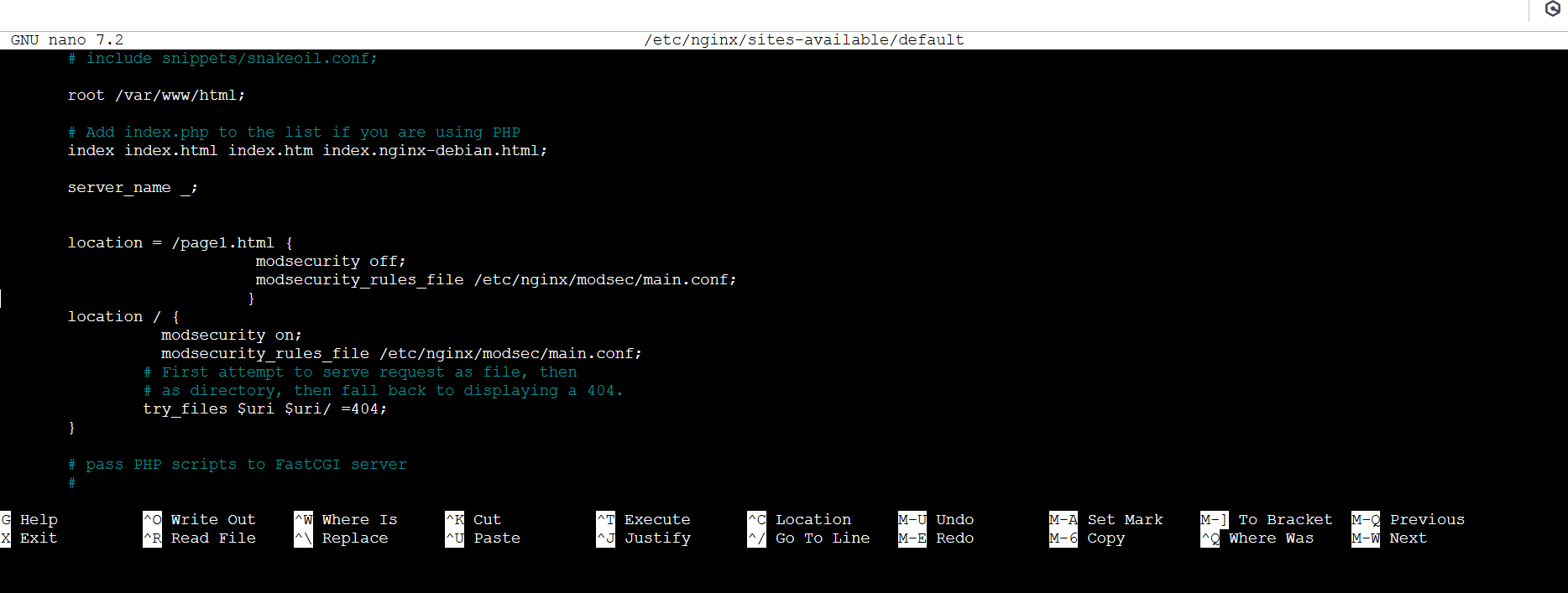
**sudo mv /usr/local/modsecurity-crs/crs-setup.conf.example /usr/local/modsecurity-crs/crs-setup.conf**

1. **Enable ModSecurity in Nginx Configuration**:
   * Edited the Nginx configuration file to enable ModSecurity:

**nano /etc/nginx/nginx.conf**

Added the following inside the http block: To enable ModSecurity, I updated the Nginx configuration file (/etc/nginx/sites-available/default). I added modsecurity on; and specified the path to the ModSecurity rules file (/etc/nginx/modsec/main.conf) for requests. Separate configurations were used for vulnerable and secure pages.

**modsecurity on;**

**modsecurity\_rules\_file /etc/nginx/modsec/main.conf;**

1. **Test and Verify Setup**:
   * Tested the Nginx configuration:

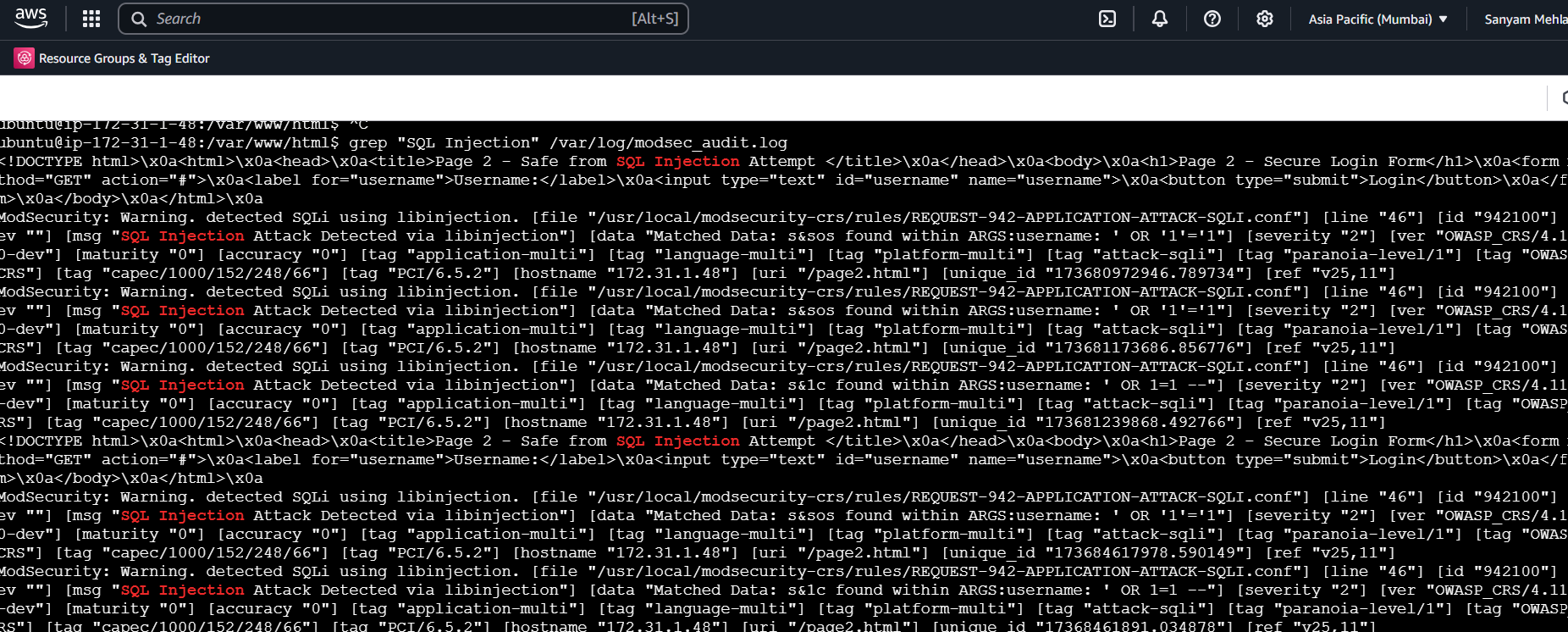
**sudo nginx -t**

* + Restarted Nginx:

**sudo systemctl restart nginx**

* + Verified that SQL injection attempts were blocked by checking logs using command for identifying SQL Injection attempts:

**grep "SQL Injection" /var/log/modsec\_audit.log**



**2. This form should have adequate controls for SQL injection attack and should not be exploitable**

I created below secure form (page2.html):

<!DOCTYPE html>

<html>

<head>

<title>Page 2 - Safe from SQL Injection Attempt</title>

</head>

<body>

<h1>Page 2 - Secure Login Form</h1>

<form method="GET" action="#">

<label for="username">Username:</label>

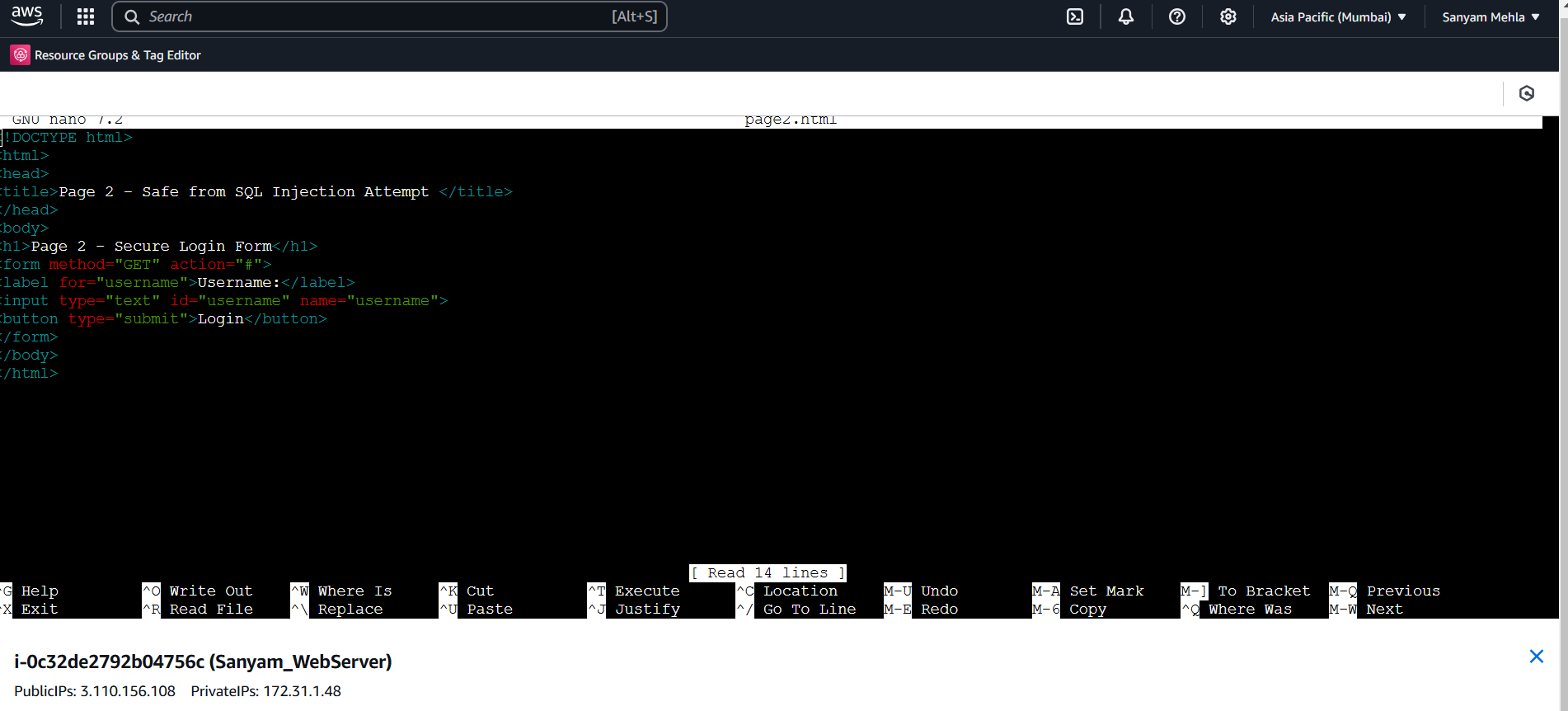
<input type="text" id="username" name="username">

<button type="submit">Login</button>

</form>

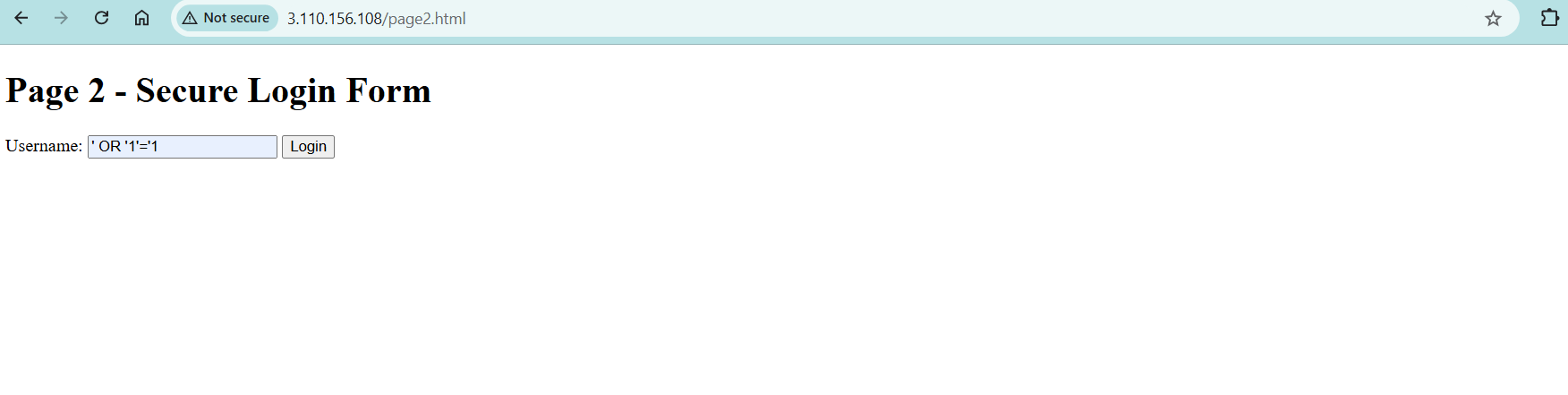
</body>

</html>

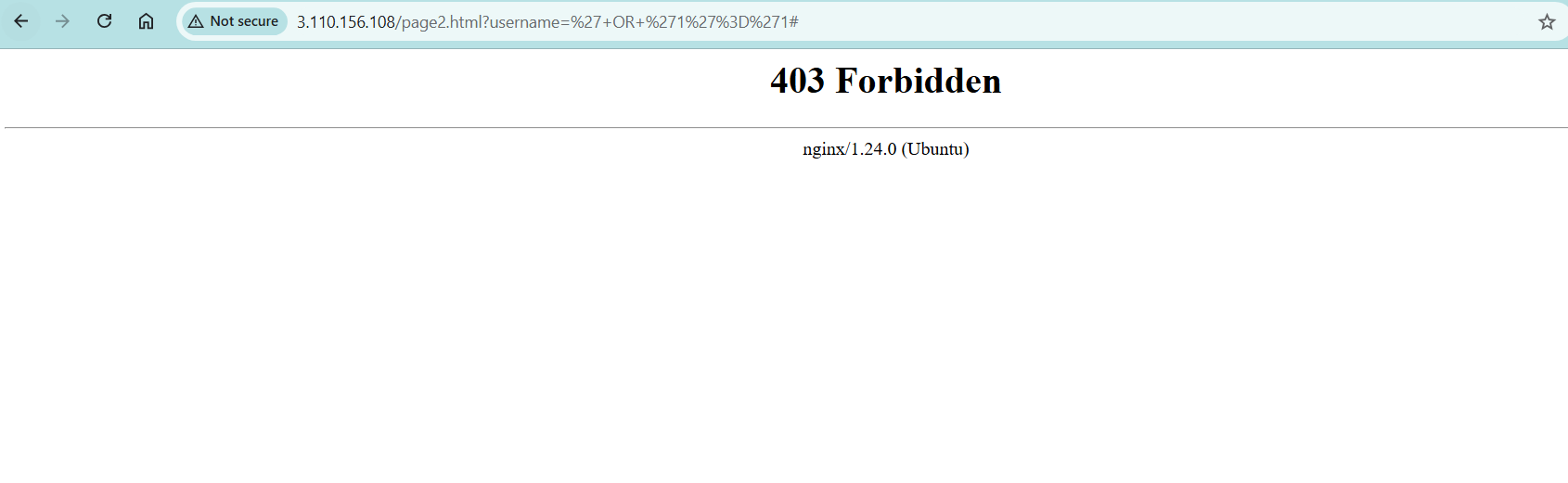


* The form was tested, and ModSecurity successfully blocked SQL injection attempts.

**Before Submission**: Entering a malicious payload like ' OR 1=1 -- in the username field.



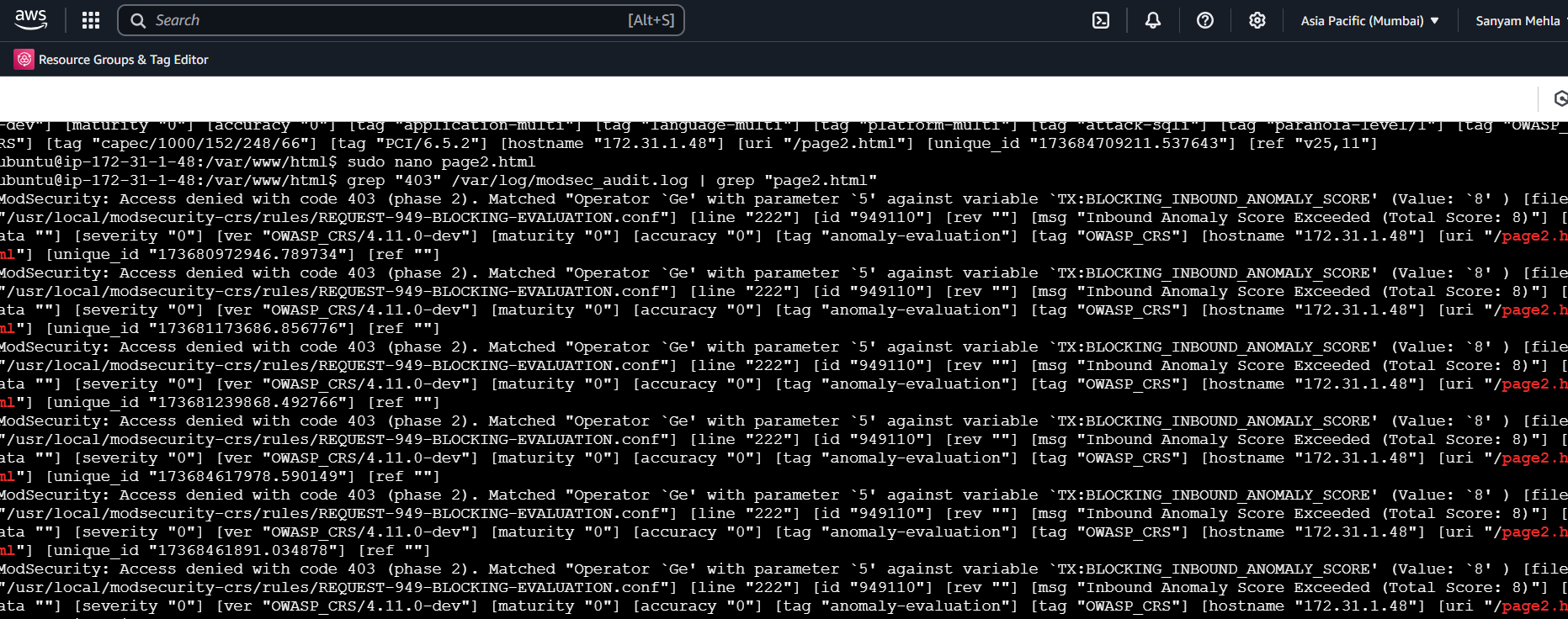
**After Submission**: Upon submitting the payload, the SQL injection successfully blocked, and the URL directly displayed 403 error.



To verify that ModSecurity effectively blocks unauthorized requests to page2.html, I filtered the audit logs using the command:

**grep "403" /var/log/modsec\_audit.log | grep "page2.html"**

This confirmed that SQL injection attempts and other malicious requests were successfully detected and blocked, resulting in 403 responses, as shown in the screenshot.



**Submission Details**

* **Exploitable Form**: <http://3.110.156.108/page1.html>
* **Non-Exploitable Form**: <http://3.110.156.108/page2.html>