Executive Summary

This project focused on performing a security audit and penetration testing of a basic cloud-hosted web application deployed on an AWS EC2 instance running Ubuntu Server. The primary objective was to identify potential security risks, analyze their severity, and implement remediation measures to strengthen the overall security posture.

We deployed a simple Flask-based web service on the EC2 instance and performed a baseline security review to examine open ports and exposed services. Using external scanning tools (Nmap) and internal system auditing (Lynis), we identified potential risks such as publicly accessible ports (22, 80, 5000). Additionally, we leveraged AWS Inspector, a cloud-native security assessment tool, which confirmed network reachability issues and highlighted possible package vulnerabilities.

Remediation steps included restricting **SSH (Port 22)** access to a single trusted IP, removing unnecessary inbound rules for **HTTP (Port 80)** and **Flask development (Port 5000)**, and performing full system updates to patch software vulnerabilities. After implementing these changes, we re-ran AWS Inspector, which reported **zero active findings**, confirming that the security risks had been successfully mitigated.

This project demonstrates how a simple cloud deployment can be effectively hardened by identifying and addressing key vulnerabilities, applying access control best practices, and leveraging both traditional security scanning tools and cloud-native solutions.

Step 1:Set up a basic cloud-hosted web app using AWS EC2

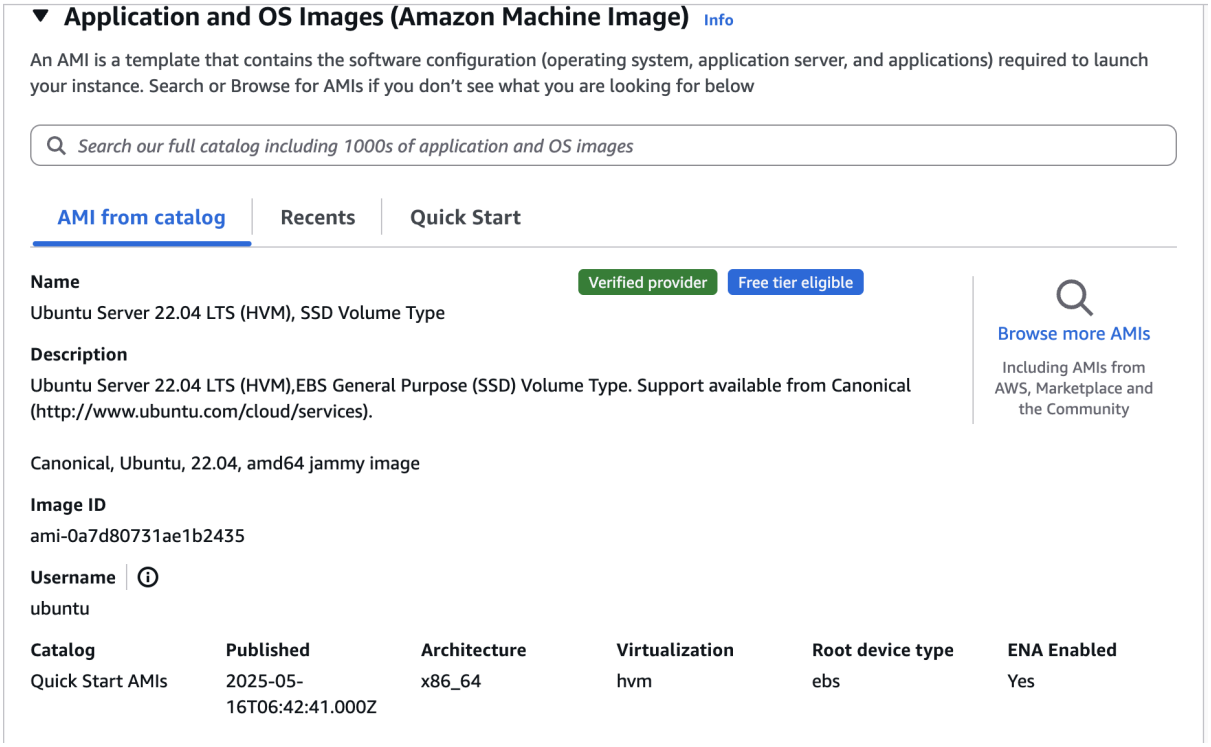
This security audit was conducted on a cloud-hosted web application deployed on AWS EC2. The goal was to simulate a real-world security assessment process by building, evaluating, and improving the security posture of a public-facing application.

1.Set up a cloud-hosted web app using AWS

a. Launch an Amazon EC2 instance and set up a name as FlaskSecurityDemo

b. Setup Amazon Machine Image

We chose Ubuntu Server 22.04 LTS (Long Term Support) as the Amazon Machine Image (AMI) for my EC2 instance. This is because it is stable, lightweight, and widely supported. As an LTS version, it receives security updates and maintenance for five years, making it a reliable choice for server environments. Ubuntu has strong community support, excellent documentation, and works well with Python and Flask, which we use for my web development project.



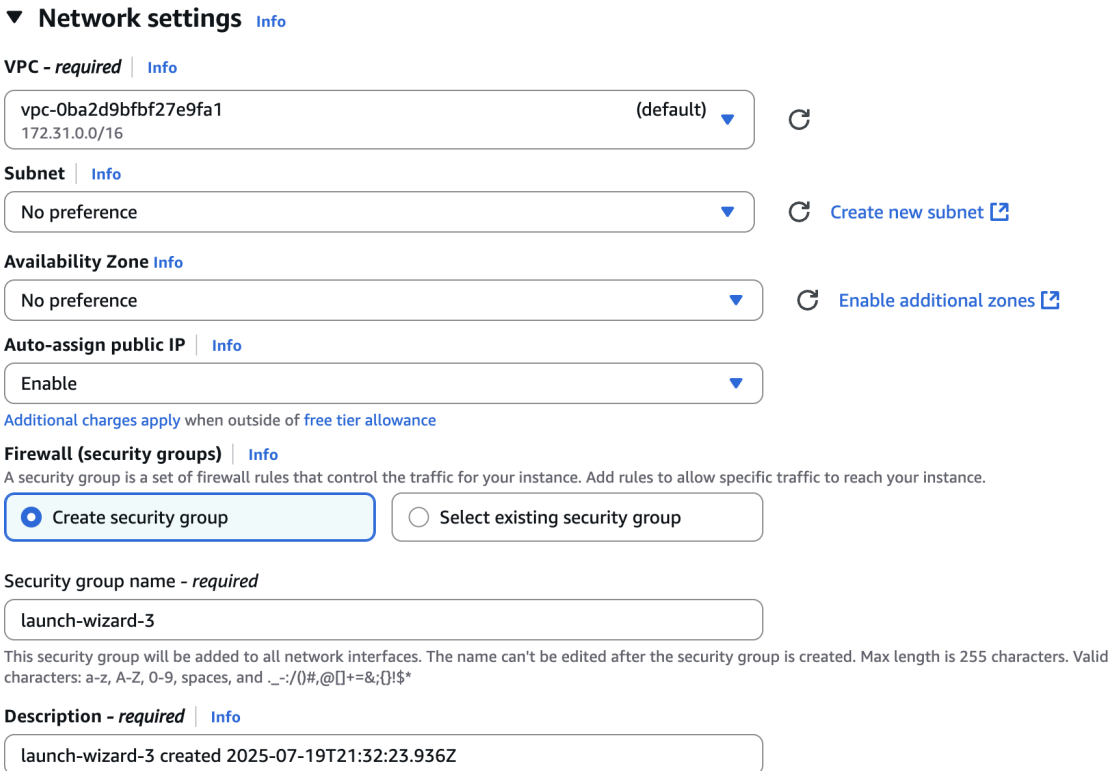
c. Network Settings

VPC: We used the default Virtual Private Cloud (vpc-0ba2d9bfbf27e9fa1), which provides a secure and isolated network environment.

Subnet: We allowed AWS to choose the subnet automatically by selecting "No preference."

Availability Zone: Set to "No preference" to let AWS place the instance in any zone.

Auto-assign Public IP: Enabled, so that the instance receives a public IPv4 address, allowing access from the internet.



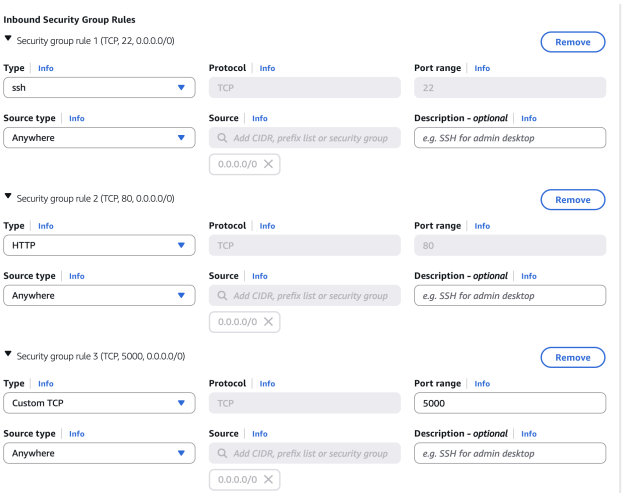
d. Setup Security Group

To ensure proper access and functionality of the EC2 instance and the deployed web application, three inbound security group rules were configured.

The first rule allows remote login to the EC2 instance via SSH on port 22 from any IP address.

The second rule opens port 80 for HTTP traffic, allowing users to access the default web page of the application.

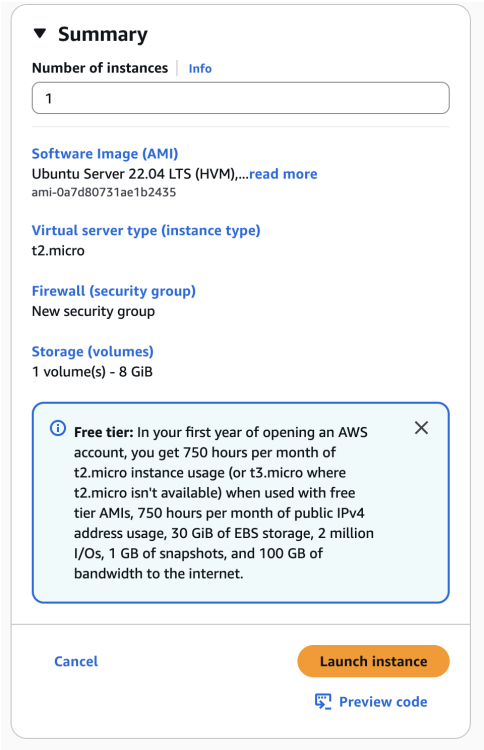
The third rule enables access to port 5000, which is the default port used by Flask applications, allowing users to interact with the Flask-based web service.



e. Launch Instance and Get Summary

After setup steps, we clicked ‘Launch Instance’. Then an EC2 instance was successfully launched using the Ubuntu Server 22.04 LTS (HVM) Amazon Machine Image (AMI). After that, we got a summary list. According to the summary, we can see that the instance type selected is t2.micro, which is eligible for AWS Free Tier and provides 1 vCPU and 1 GiB of memory — suitable for lightweight web applications and testing.

The instance was configured with a new security group that allows inbound traffic on ports 22 (SSH), 80 (HTTP), and 5000 (Flask default). A public IPv4 address (44.204.74.240) was automatically assigned, enabling internet access to the instance.



Instance summary for i-0a5742f3c081ead57 (FlaskSecurityDemo) Info

Connect | Instance state | Actions

Updated less than a minute ago

Instance ID
i-0a5742f3c081ead57

IPv6 address
–

Hostname type
IP name: ip-172-31-90-123.ec2.internal

Answer private resource DNS name
IPv4 (A)

Auto-assigned IP address
44.204.74.240 [Public IP]

IAM Role
–

IMDSv2
Optional
EC2 recommends setting IMDSv2 to required | Learn more

Operator
–

Public IPv4 address
44.204.74.240 | open address

Instance state
Running

Private IP DNS name (IPv4 only)
ip-172-31-90-123.ec2.internal

Instance type
t2.micro

VPC ID
vpc-0ba2d9bfbf27e9fa1

Subnet ID
subnet-0e89bc607149f0911

Instance ARN
arn:aws:ec2:us-east-1:730335547474:instance/i-0a5742f3c081ead57

Private IPv4 addresses
172.31.90.123

Public DNS
ec2-44-204-74-240.compute-1.amazonaws.com | open address

Elastic IP addresses
–

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more

Auto Scaling Group name
–

Managed
false

Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

Instance details Info

AMI ID
ami-0a7d80731ae1b2435

AMI name
ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20250516

Stop protection
Disabled

Instance reboot migration
Default (On)

Stop-hibernate behavior
Disabled

State transition reason
–

State transition message
–

Owner
730335547474

Current instance boot mode
legacy-bios

Answer RBN DNS hostname IPv4
Enabled

Monitoring
disabled

Allowed image
-

Launch time
Sat Jul 19 2025 14:36:45 GMT-0700 (Pacific Daylight Time) (14 minutes)

Instance auto-recovery
Default

AMI Launch index
0

Credit specification
standard

Usage operation
RunInstances

Enclaves Support
–

Allow tags in instance metadata
Disabled

Platform details
Linux/UNIX

Termination protection
Disabled

AMI location
amazon/ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20250516

Lifecycle
normal

Key pair assigned at launch
my-key

Kernel ID
–

RAM disk ID
–

Boot mode
uefi-preferred

Use RBN as guest OS hostname
Disabled

f. Update & Install Python 3

To prepare the EC2 instance for running Python applications, the system was first updated using the command *sudo apt update*, which refreshed the package list and ensured access to the latest available software versions.

```
ubuntu@ip-172-31-90-123:~$ sudo apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe Translation-en [5652 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 c-n-f Metadata [286 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [217 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse Translation-en [112 kB]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 c-n-f Metadata [8372 B]
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [2757 kB]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [437 kB]
Get:13 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [4038 kB]
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [729 kB]
Get:15 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1224 kB]
Get:16 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [302 kB]
Get:17 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 c-n-f Metadata [28.7 kB]
Get:18 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages [59.5 kB]
Get:19 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/multiverse Translation-en [14.2 kB]
Get:20 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 c-n-f Metadata [592 B]
Get:21 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages [68.8 kB]
Get:22 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/main Translation-en [11.4 kB]
Get:23 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/main amd64 c-n-f Metadata [392 B]
Get:24 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/restricted amd64 c-n-f Metadata [116 B]
Get:25 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages [30.0 kB]
Get:26 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/universe Translation-en [16.5 kB]
Get:27 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/universe amd64 c-n-f Metadata [672 B]
Get:28 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports/multiverse amd64 c-n-f Metadata [116 B]
Get:29 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [2510 kB]
Get:30 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [374 kB]
Get:31 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [3877 kB]
91% [31 Packages 2257 kB/3877 kB 58%]                              440 kB/s 8sGet:32 http://security.ubuntu.com/ubuntu jammy-security/restricted Translation-en [699 kB]
Get:33 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [991 kB]
Get:34 http://security.ubuntu.com/ubuntu jammy-security/universe Translation-en [215 kB]
Get:35 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 c-n-f Metadata [21.7 kB]
Get:36 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [40.3 kB]
Get:37 http://security.ubuntu.com/ubuntu jammy-security/multiverse Translation-en [8908 B]
Get:38 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 c-n-f Metadata [368 B]
Fetched 39.2 MB in 21s (1895 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
80 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Next, Python's package manager pip was installed using the command *sudo apt install python3-pip -y*.

```
ubuntu@ip-172-31-90-123:~$ sudo apt install python3-pip -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  build-essential bzip2 cpp cpp-11 dpkg-dev fakeroot fontconfig-config
  fonts-dejavu-core g++ g++-11 gcc gcc-11 gcc-11-base javascript-common
  libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl
  libasan6 libatomic1 libc-dev-bin libc-devtools libc6 libc6-dev libcc1-0
  libcrypt-dev libdeflate0 libdpkg-perl libexpat1-dev libfakeroot
  libfile-fcntllock-perl libfontconfig1 libgcc-11-dev libgd3 libgomp1 libisl23
  libitm1 libjbig0 libjpeg-turbo8 libjpeg8 libjs-jquery libjs-sphinxdoc
  libjs-underscore liblsan0 libmpc3 libnsl-dev libpython3-dev libpython3.10
  libpython3.10-dev libpython3.10-minimal libpython3.10-stdlib libquadmath0
  libstdc++-11-dev libtiff5 libtirpc-dev libtsan0 libubsan1 libwebp7 libxpm4
  linux-libc-dev lto-disabled-list make manpages-dev python3-dev python3-wheel
  python3.10 python3.10-dev python3.10-minimal rpcsvc-proto zlib1g-dev
```

This command installed pip3 along with several required development tools and Python dependencies. The installation was successful, and Python 3 was confirmed to be available. Then, we tried to run a Python script *(app.py)*, but the system returned an error indicating that the file did not exist. This means the environment was ready, but the Flask application file still needed to be created.

Therefore, we re-run the pip installation command to confirm that python3-pip was already installed and up to date.

```
ubuntu@ip-172-31-90-123:~$ python3 app.py
python3: can't open file '/home/ubuntu/app.py': [Errno 2] No such file or directory
ubuntu@ip-172-31-90-123:~$ sudo apt install python3-pip -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-pip is already the newest version (22.0.2+dfsg-1ubuntu0.6).
0 upgraded, 0 newly installed, 0 to remove and 74 not upgraded.
```

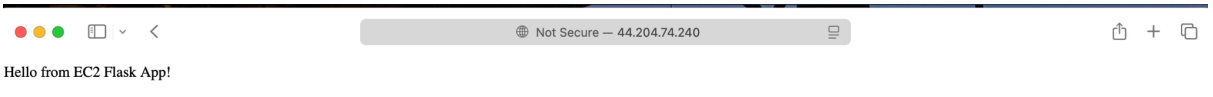
g. Install & Run Flask

We use the command *pip3 install flask* to install Flask. After installation, a simple Flask application was created using *nano app.py*, and the app was launched by the command *python3 app.py*. Then, we can see the server started successfully, listening on 0.0.0.0:5000, meaning it was accessible from the internet via the EC2 instance's public IP.

```
ubuntu@ip-172-31-90-123:~$ pip3 install flask
Defaulting to user installation because normal site-packages is not writeable
Collecting flask
  Downloading flask-3.1.1-py3-none-any.whl (103 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 103.3/103.3 KB 2.5 MB/s eta 0:00:00
Collecting blinker>=1.9.0
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting click>=8.1.3
  Downloading click-8.2.1-py3-none-any.whl (102 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 102.2/102.2 KB 9.4 MB/s eta 0:00:00
Collecting jinja2>=3.1.2
  Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.9/134.9 KB 13.1 MB/s eta 0:00:00
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting markupsafe>=2.1.1
  Downloading MarkupSafe-3.0.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
Collecting werkzeug>=3.1.0
  Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 224.5/224.5 KB 18.1 MB/s eta 0:00:00
Installing collected packages: markupsafe, itsdangerous, click, blinker, werkzeug, jinja2, flask
  WARNING: The script flask is installed in '/home/ubuntu/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed blinker-1.9.0 click-8.2.1 flask-3.1.1 itsdangerous-2.2.0 jinja2-3.1.6 markupsafe-3.0.2 werkzeug-3.1.3
ubuntu@ip-172-31-90-123:~$ nano app.py
ubuntu@ip-172-31-90-123:~$ python3 app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.90.123:5000
Press CTRL+C to quit
81.28.158.59 - - [19/Jul/2025 21:47:56] "GET / HTTP/1.1" 200 -
81.28.158.59 - - [19/Jul/2025 21:47:56] "GET /favicon.ico HTTP/1.1" 404 -
81.28.159.41 - - [19/Jul/2025 21:48:48] "GET / HTTP/1.1" 200 -
81.28.159.41 - - [19/Jul/2025 21:48:48] "GET /favicon.ico HTTP/1.1" 404 -
81.28.159.41 - - [19/Jul/2025 21:49:02] "GET / HTTP/1.1" 200 -
81.28.159.41 - - [19/Jul/2025 21:49:18] "GET / HTTP/1.1" 200 -
81.28.159.41 - - [19/Jul/2025 21:49:18] "GET /favicon.ico HTTP/1.1" 404 -
```

h. Application Output

After successfully launching the Flask application on the EC2 instance, we accessed it through the instance's public IP address in a web browser. The page displayed the message "Hello from EC2 Flask App!", confirming that the Flask server was running correctly and responding to HTTP requests from the internet.



2. Technical Implementation Details

a. Cloud Setup

Cloud Provider: Amazon Web Services (AWS)
Region: US East (N. Virginia)
Instance Type: t2.micro (Free Tier Eligible)

AMI: Ubuntu Server 22.04 LTS (64-bit x86)
Public IPv4 Address: 44.204.74.240
Instance Name: FlaskSecurityDemo

b. Security Group Configuration (Firewall)

Rule	Type	Port Range	Source	Purpose
1	SSH	22	Anywhere	Allow remote login to the EC2 instance using SSH
2	HTTP	80	Anywhere	Allow access to the default web page of the web application
3	Custom TCP	5000	Anywhere	Allow access to the Flask application default port

c. Flask Web Application Deployment

Language & Framework: Python 3, Flask
Deployment Steps:
Installed Python & Flask via apt and pip3;
Created and ran app.py with Flask's default development server;
Application is bound to 0.0.0.0:5000 and publicly accessible
Application Output:
URL: http://44.204.74.240:5000
Page content: "Hello from EC2 Flask App!"

3. Conclusion

In Phase 1 of the project, we successfully deployed a basic Flask application on an EC2 instance, configured initial firewall settings, and ensured public accessibility via port 5000. This foundational setup will support further stages of vulnerability scanning, risk analysis, and remediation.

Step 2: Baseline Security Review

In this phase, we performed a manual baseline security inspection of the cloud-hosted EC2 instance before any vulnerability scanning. The objective was to identify potentially exposed services, unnecessary open ports, and insecure default configurations that could be exploited.

1. Open Ports & Exposed Services

An initial analysis was performed to identify all listening ports on the EC2 instance to understand the network attack surface.

Command Used: sudo ss -tuln

```
sudo ss -tuln

^Cubuntu@ip-172-31-90-123:~sudo ss -tulnln
Netid State  Recv-Q Send-Q       Local Address:Port   Peer Address:Port Process
udp   UNCONN 0      0            127.0.0.53%lo:53         0.0.0.0:*
udp   UNCONN 0      0       172.31.90.123%eth0:68         0.0.0.0:*
udp   UNCONN 0      0                127.0.0.1:323        0.0.0.0:*
udp   UNCONN 0      0                    [::1]:323           [::]:*
tcp   LISTEN 0      4096         127.0.0.53%lo:53         0.0.0.0:*
tcp   LISTEN 0      128                0.0.0.0:22         0.0.0.0:*
tcp   LISTEN 0      128                   [::]:22            [::]:*
```

Findings:

The scan produced the following key findings:

Port	Protocol	Service	Exposed to Public?	Description
22	TCP	SSH	Yes	Publicly accessible for remote login.
5000	TCP	Flask	Not running	Will be exposed if the *app.py* is running.
53, 68, 323	UDP	DNS, DHCP, NTP	No (local only)	System-related background services.

Conclusion:

- **Only port 22 (SSH)** was exposed at the time of inspection.
- Flask on port 5000 will be exposed when the web app is running, but no unexpected ports were found open.
- No database ports (e.g., 3306, 5432) or administrative panels were exposed.

Recommendations:

- **Restrict SSH Access:** Configure AWS security groups to limit access to port 22 from trusted IP ranges only.
- **Use a Reverse Proxy:** Manage public access to the Flask application using a firewall or reverse proxy, especially once testing is complete.
- **Periodic Scanning:** Regularly scan for newly opened ports after any system updates or deployments.

2 Running Services

An audit of all active system services was conducted to ensure no unauthorized or unnecessary processes were running.

Command Used: `systemctl list-units --type=service --state=running`

```
ubuntu@ip-172-31-90-123:~$ systemctl list-units --type=service --state=running
  UNIT                                      LOAD   ACTIVE SUB     DESCRIPT>
  acpid.service                             loaded active running ACPI eve>
  chrony.service                            loaded active running chrony,  >
  cron.service                              loaded active running Regular  >
  dbus.service                              loaded active running D-Bus Sy>
  getty@tty1.service                        loaded active running Getty on>
  multipathd.service                        loaded active running Device-M>
  networkd-dispatcher.service               loaded active running Dispatch>
  packagekit.service                        loaded active running PackageK>
  polkit.service                            loaded active running Authoriz>
  rsyslog.service                           loaded active running System L>
  serial-getty@ttyS0.service                loaded active running Serial G>
  snap.amazon-ssm-agent.amazon-ssm-agent.service loaded active running Service  >
  snapd.service                             loaded active running Snap Dae>
  ssh.service                               loaded active running OpenBSD  >
  systemd-journald.service                  loaded active running Journal  >
  systemd-logind.service                    loaded active running User Log>
  systemd-networkd.service                  loaded active running Network  >
  systemd-resolved.service                  loaded active running Network  >
  systemd-udevd.service                     loaded active running Rule-bas>
  unattended-upgrades.service               loaded active running Unattend>
  user@1000.service                         loaded active running User Man>

lines 1-23
```

Findings

The audit categorized the running services as follows:

- **Expected Core Services:** Standard Linux services such as ssh.service, cron.service, rsyslog.service, systemd-* were all running as expected.
- **AWS Management Agents:** The snap.amazon-ssm-agent was active, which is standard for managing EC2 instances.
- **Optional Services:** Two services, multipathd.service and packagekit.service, were running but are not essential for the web application's functionality.

Conclusion:

No suspicious or third-party daemons were found.

Recommendations:

- **Disable Unused Services:** To reduce the potential attack surface, disable unnecessary services like multipathd.
- **Retain Security-Hardening Services:** Keep unattended-upgrades.service enabled to ensure automatic application of security patches.
- **Regular Audits:** Periodically review the list of running services and compare it against a secure baseline to detect any unauthorized changes.

Step 3: Run vulnerability scans using tools like Nmap.

A basic security evaluation was conducted on an EC2 instance hosting a Flask web application. The assessment involved scanning the server for open ports using Nmap and performing a system audit with lynis. Several vulnerabilities and misconfigurations were identified, including missing firewall rules, weak authentication settings, and outdated packages.

```
ubuntu@ip-172-31-90-123:~$ sudo nmap -sS -Pn 44.204.74.240
Starting Nmap 7.80 ( https://nmap.org ) at 2025-07-19 22:25 UTC
Nmap scan report for ec2-44-204-74-240.compute-1.amazonaws.com (44.204.74.240)
Host is up (0.00085s latency).
Not shown: 997 filtered ports
PORT     STATE  SERVICE
22/tcp   open   ssh
80/tcp   closed http
5000/tcp closed upnp

Nmap done: 1 IP address (1 host up) scanned in 4.87 seconds
```

An Nmap scan was performed to determine which network services were exposed to the internet. The scan revealed that **port 22 (SSH)** was open, while other common ports such as **80 (HTTP)** and **5000 (UPnP)** were closed. The presence of an open SSH port indicates that the system is remotely accessible, and while this is often necessary, it should be tightly secured using firewall rules, SSH key authentication, and potentially IP whitelisting to minimize unauthorized access.

Step 4: Vulnerability Logging and Severity Analysis

A system-level audit was conducted using Lynis version 3.0.7. The audit identified several vulnerabilities and areas requiring attention. One critical issue was the presence of outdated or vulnerable software packages, which could be exploited if not patched. In addition, the system had no active firewall rules despite having the iptables module loaded. This significantly increases the server's attack surface by allowing unrestricted access to all network services.

Here is the command we used:
sudo apt install lynis -y
sudo lynis audit system
The result showed:

```
-[ Lynis 3.0.7 Results ]-

Warnings (2):
----------------------------
! Found one or more vulnerable packages. [PKGS-7392]
    https://cisofy.com/lynis/controls/PKGS-7392/

! iptables module(s) loaded, but no rules active [FIRE-4512]
    https://cisofy.com/lynis/controls/FIRE-4512/

Suggestions (49):
----------------------------
* This release is more than 4 months old. Check the website or GitHub to see if there is an update available. [LYNIS]
    https://cisofy.com/lynis/controls/LYNIS/

* Install libpam-tmpdir to set $TMP and $TMPDIR for PAM sessions [DEB-0280]
    https://cisofy.com/lynis/controls/DEB-0280/

* Install apt-listbugs to display a list of critical bugs prior to each APT installation. [DEB-0810]
    https://cisofy.com/lynis/controls/DEB-0810/

* Install apt-listchanges to display any significant changes prior to any upgrade via APT. [DEB-0811]
    https://cisofy.com/lynis/controls/DEB-0811/

* Install fail2ban to automatically ban hosts that commit multiple authentication errors. [DEB-0880]
    https://cisofy.com/lynis/controls/DEB-0880/

* Set a password on GRUB boot loader to prevent altering boot configuration (e.g. boot in single user mode without password) [BOOT-5122]
    https://cisofy.com/lynis/controls/BOOT-5122/
```

The audit also pointed out weaknesses in the system's authentication configuration. Default or weak authentication settings were detected, suggesting the need for improved password policies and the implementation of multi-factor authentication (MFA).

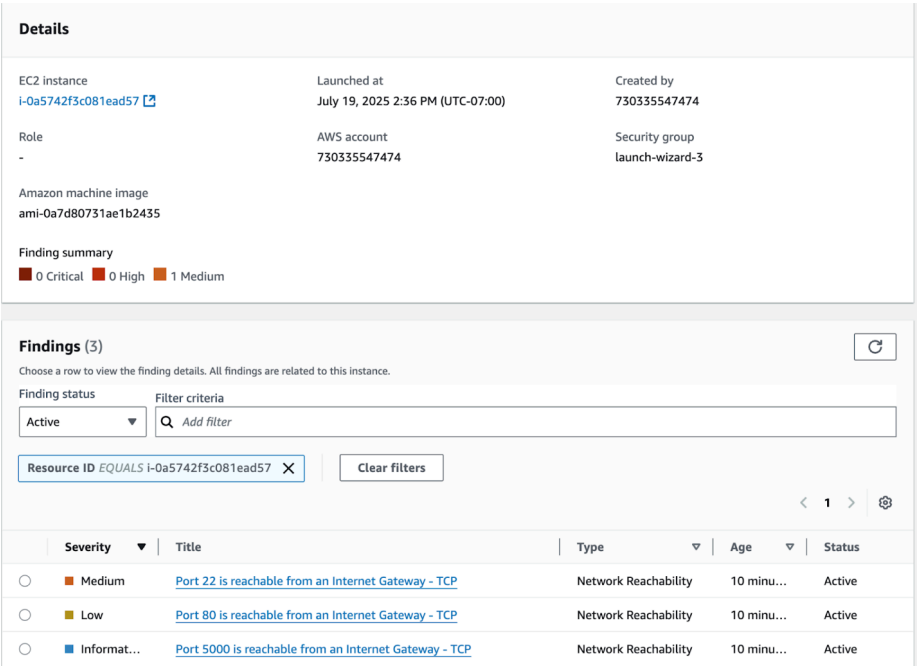
The findings indicate that while the system is operational, it lacks critical security controls. Immediate actions should include patching vulnerable packages, configuring a firewall, hardening SSH access, and applying secure logging and file system practices. These steps will greatly improve the overall security posture of the server and help protect it against both external and internal threats.

Step 5: Use Cloud-Native Tools to Verify Additional Risks

To complement the manual scanning tools (Nmap and Lynis), we used AWS Inspector, a native AWS cloud security service, to evaluate the EC2 instance for additional risks. After enabling AWS Inspector and assigning the required IAM role and Systems Manager agent, a full security scan was performed. The scan focused on both network reachability and system vulnerabilities, leveraging Amazon's continuously updated vulnerability database.

The scan identified three major network reachability findings: Port 22 (SSH) was publicly accessible from the internet, Port 80 (HTTP) was publicly accessible despite no production web service running, and Port 5000 (Flask development port) was open and reachable. In addition, AWS Inspector detected multiple package vulnerabilities, including one critical and several high-severity issues caused by outdated system libraries and kernel components. These findings highlighted potential attack vectors, including brute-force attempts on SSH and possible exploitation of unpatched packages.

Severity	Title
Medium	Port 22 is reachable from an Internet Gateway (SSH)
Low	Port 80 is reachable from an Internet Gateway (HTTP)
Informational	Port 5000 is reachable from an Internet Gateway (UPnP/Custom App)



Step 6: Propose and Implement Remediation Steps for Key Vulnerabilities

Based on the Inspector findings, a remediation plan was designed and implemented. For network reachability issues, unnecessary ports were closed, and access was restricted according to the principle of least privilege. Specifically, the inbound rules for **Port 80 (HTTP)** and **Port 5000 (Flask)** were removed entirely since these services were not required for production. The inbound rule for **Port 22 (SSH)** was updated to allow access only from a single trusted IP address, effectively preventing unauthorized login attempts from unknown networks.

For the package vulnerabilities, we applied a full system upgrade and patching process, updating all packages and the Linux kernel using the following commands:

sudo apt update && sudo apt full-upgrade -y && sudo reboot

This ensured that the latest security patches were installed and active.

After implementing these changes, we triggered a new AWS Inspector scan to validate the remediation. The subsequent scan showed **no active findings** for open ports, confirming that the network attack surface was significantly reduced. Additionally, the number of package vulnerabilities decreased after patching, further improving the security posture of the EC2 instance.

```
ubuntu@ip-172-31-90-123:~$ sudo apt update
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
ubuntu@ip-172-31-90-123:~$ sudo apt full-upgrade -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
ubuntu@ip-172-31-90-123:~$
```