Zehra Syeda Sarwat
CyberArch, HW 7
May 7 2025

Git Repo: https://github.com/zehrasyedaGW/CyberArchHW7.git

Objective

Purpose of this exercise as I understand is to set up my environment and deploy and test a vulnerable Flask app inside a Docker container on AWS EC2 instance, analyze and identify threats and vulnerabilities, then harden the system against common security threats. A detailed report that provides step by step on what I did, what I learned, screen shots for every step, errors I experienced and how I fixed them, is also being submitted in a separate doc called "Details Assignment 7". I couldn't put everything in here because of length limit for this report.

Approach

I took the assignment step by step, and completed it over a period of days. My videos are spotty but I made sure I screen shot every step as I did it because I wanted to remember what I am doing, why I am doing it, and what the command prompts looked like and what it returned. I created the detailed report for my own learning so I could go back to it when needed but also submitting it here so you have all the details on what I did, why, what I learned.

The After folder has files from the instance.

The Before folder has files that I initially downloaded from Git. I ended up modifying them and forgot I had to save them and so had to download them again.

The Process Folder has files on my local computer that I used to edit, and then upload to my instance.

Step 1 – Environment set up

I set up my EC2 instance. Created and connected to Amazon Linux using SSH but had to use Putty to be able to use Win SCP. I did this because using GUI is much simpler to move files around and it enabled speed. I converted the key from Pem to ppk to be able to do this.

I uploaded Flask ap, launched it and validated end points. All screen shots and command r in the detail doc. I tested browser access and fixed errors throughout the journey. I did an initial scan and make scan failed due to docker scout issues. I then had to switch to Trivy to scan the docker image and was able to do so.

Step 2 – Threats assessment/scanning

Primarily I found the below through threat modeling and scanning. There were other vulnerabilities already existing in the set up such as hard coded passwords, use of eval, and more which are discussed in the next section.

a. Ingress rule allows traffic from anywhere on port 22. This is a security risk as anyone can connect. This can lead to Spoofing, Information disclosure, or denial of service attacks. This can be mitigated by restricting access to trusted Ips only. MITRE T1133 (External Remote Services), NIST AC-4, SC-7
b. SSH port appears open to anyone and everyone on the internet. Similar to above this can lead to Spoofing, Information disclosure, or Elevation of privileges. This can be mitigated using VPN or secure connections that are limited. MITRE T1078 (Valid Accounts), T1068, T1611; NIST AC-17, IA-2

Step 3 – Mitigation steps taken

1. Code Remediation (Flask App):
    a. Hardcoded Passwords: Replaced with environment variable (os.environ.get("APP_PASSWORD"))
    b. Use of eval(): Replaced with ast.literal_eval() to prevent code injection
    c. Input Validation: Improved validation in app.py and corrected syntax issues in docker-compose.yml
    d. Flask Binding: Changed host to 127.0.0.1 to limit access to localhost only
2. Docker Security:
    a. Minimal Base Image: Used python:3.9-alpine for smaller attack surface
    b. Non-root User: Ensured app runs as appuser using adduser and addgroup
    c. Health Checks: Added HEALTHCHECK directive in Dockerfile and Compose for resilience
    d. Multi-Stage Builds: Attempted but couldn't complete due to persistent build errors
3. Compose Hardening:
    a. Security Controls Added:
        i. read_only, security_opt, mem_limit, pids_limit
        ii. Restricted exposed ports to 127.0.0.1
        iii. Moved secrets into a .env file to prevent leakage

Step 4 – Security Architecture

1. Threat Modeling:

    a. I performed the Stride analysis by getting the Stride file from git from another folder in week 7
    b. I mapped threats to MITRE and NIST
    c. I generated the file which shows the mapping clearly
2. Architecture Diagram:
    a. I developed a script to build the diagram. This became a problem as I had to install graphviz to be able to generate the diagram using the python script. I kept running out of space (even before I was generating the diagram I was running out of space) so had to do major clean up. Then I was bale to generate the diagram under deliverables.
3. Hardening:
    a. I was able to harden the systems using the security fixes and injected best practices. This included controls below:
        i. Access Control: Restricted SSH and Flask access to trusted Ips and local host
        ii. Boundary Protection: Ingress rules updated to resuce external exposure
        iii. Least Privilege: Enforced non root docker user and secure credential handling
        iv. Hardening: Dockerfile and compose enhancements, health checks, environment file use.

Learnings:

1. Learned how to set up the system, build it, assess it, harden it. Loads of errors and issues that came up that I figured out!!!
2. Important to avoid hard coded passwords so threat actors cant gain access.
3. Always use minimal images to make docker smaller, more secure and faster to build and run.
4. Restrict network access and only allow known Ips to connect to manage more access control.
5. Loads of more learnings in the Detail doc!