|  |  |
| --- | --- |
| Name: | Rishitha Pudi |
| Lab User ID: | 23SEK3324\_U01 |
| Date: | 10-01-2024 |
| Application Name: | OWASP-wrong-secret |

**Follow the below guidelines:**





System Architecture:

(Understand the system and document the physical and logical architecture of the system, use the shapes and icons to capture the system architecture)

to capture the system architecture)

EC2

Docker

Registry

Docker Host

Client

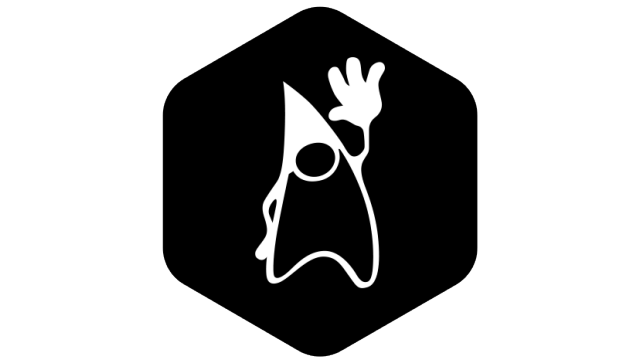
container

Docker Pull

Image

Docker

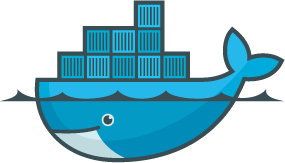
Daemon

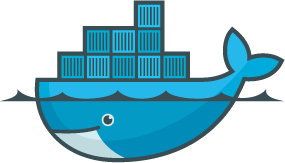


Images





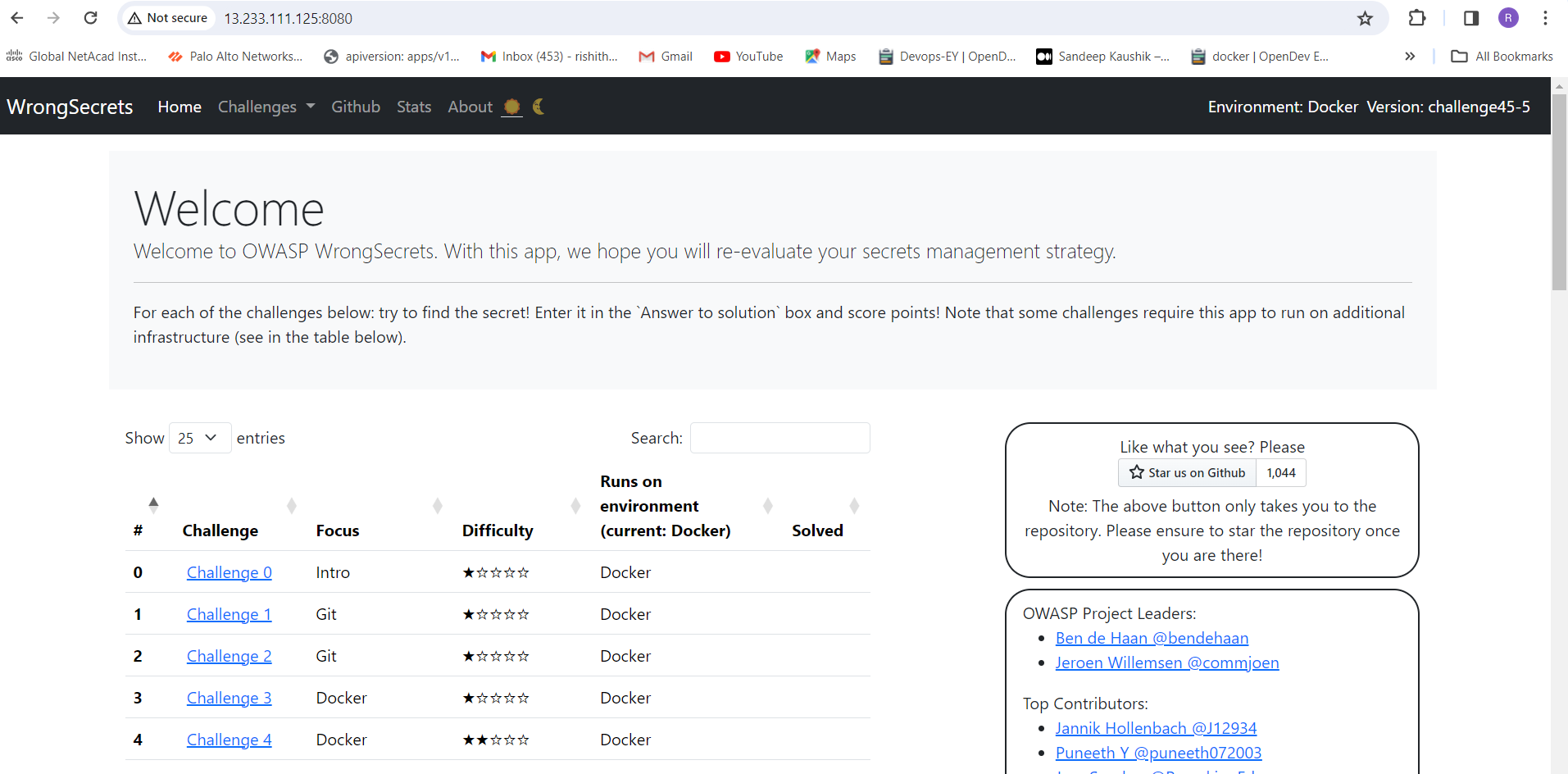




Docker Build

Docker pull

13.233.111.123



Define system’s normal behavior:

(Define the steady state of the system is defined, thereby defining some measurable outputs which can indicate the system’s normal behavior)

The web server starts, listening on defined ports like ip address: 8080.

A user accesses a website hosted on this server via a web browser.

Web server is having 5 components Home, challenges, GitHub, stat and About

Different challenges are displayed on the page, if user select anyone challenge it will redirect into another page

At that page it will ask us to write solution for the challenge, if u are answer is correct it will react good answer.

Or else bad luck try again. It do have some hints in it

Hypothesis:

(During an experiment, we need a hypothesis for comparing to a stable control group, and the same applies here too. If there is a reasonable expectation for a particular action according to which we will change the steady state of a system, then the first thing to do is to fix the system so that we accommodate for the action that will potentially have that effect on the system. For eg: "If one of our database servers fails, our service will automatically switch to a backup server, and users will not experience any downtime or data loss.")



Chaos engineering hypothesis scenarios

**Known**

We can perform an experiment about CPU utilization. We can intentionally do that and check system behaviour

If application experience any DDos attack, we cannot expect that attack but we know what steps to take to secure our system

**Unknown**

**Unknown**

**Known**

The Analyses and experiments you can do when data from multiple sources is combined in one place. But sometimes we are not sure how to mitigate.

The data you want to track but can’t currently

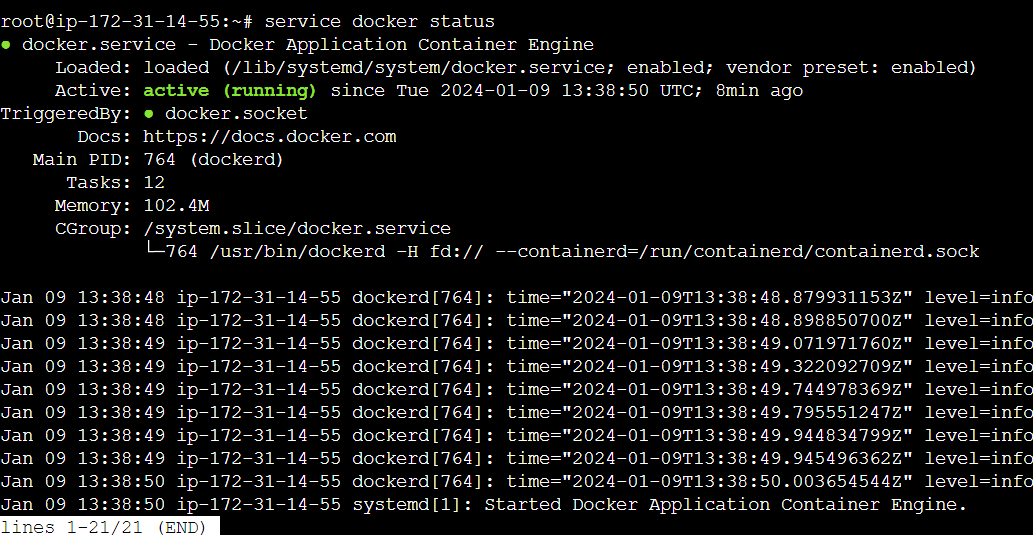
Experiment:

(Document your Preparation, Implementation, Observation and Analysis )

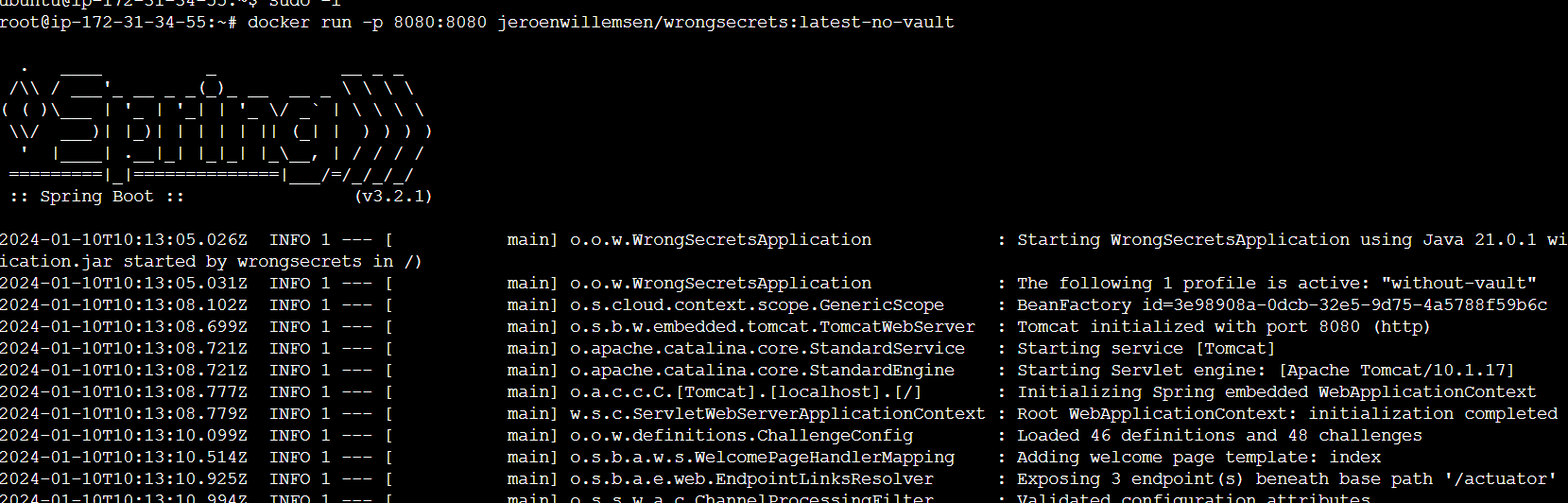
**Overview:** The game is packed with real life examples of how to not store secrets in your software. Each of these examples is captured in a challenge, which you need to solve using various tools and techniques. 

**Methodology :**

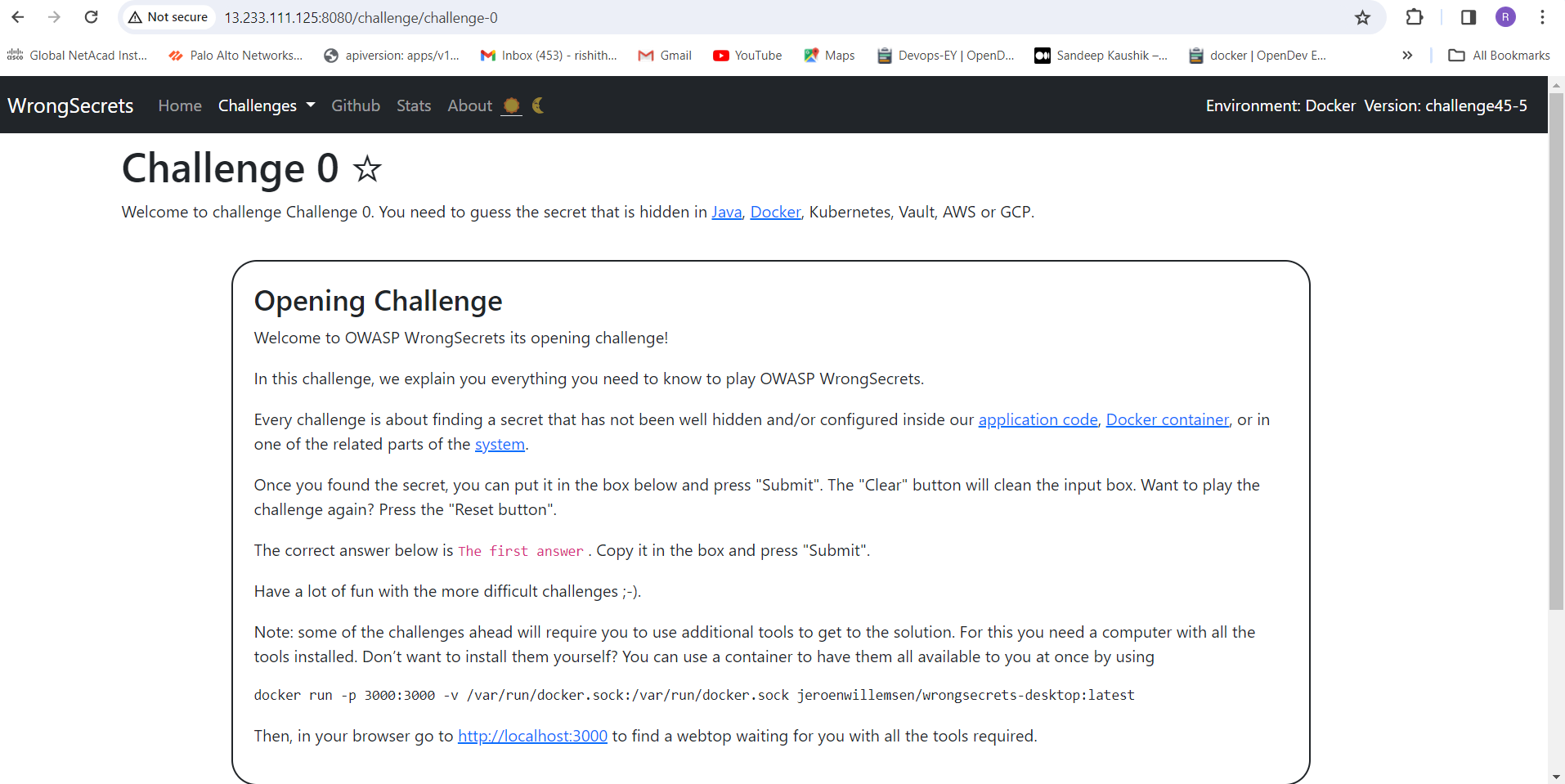
**Step1 :** create an Ec2 instance, install docker to that virtual machine. Make sure docker is running.



**Step 2:** run the following command to live that application



**Step 3:** live the application using the defined port



**Tools:**

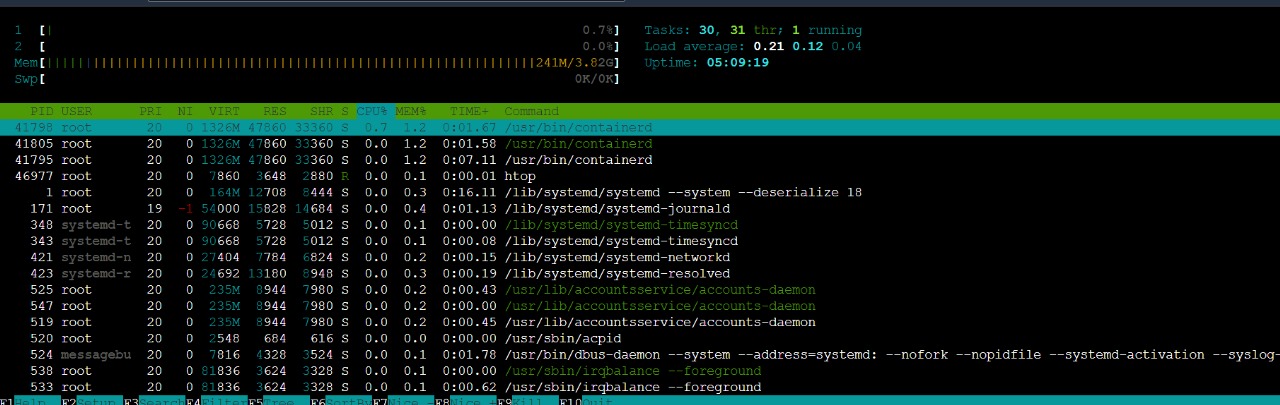
1. Gremlin
2. Horusec
3. Trivy

**Observation:**

**Analysis 1:** using Gremlin

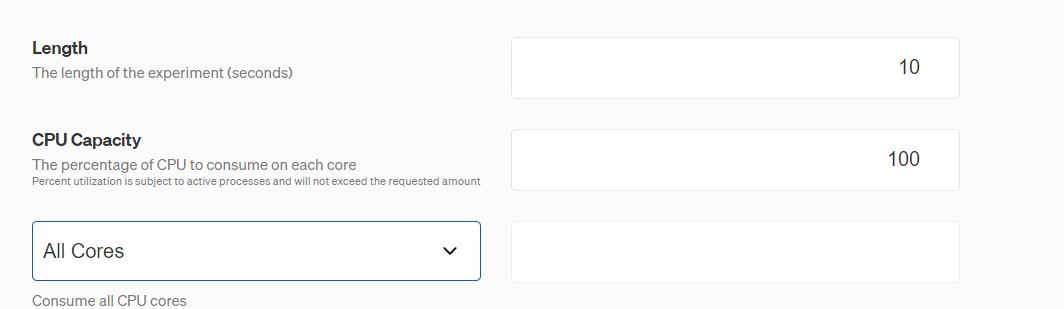
Using gremlin we can perform CPU utilization attack

The following pic is the CPU utilization range before germline attack



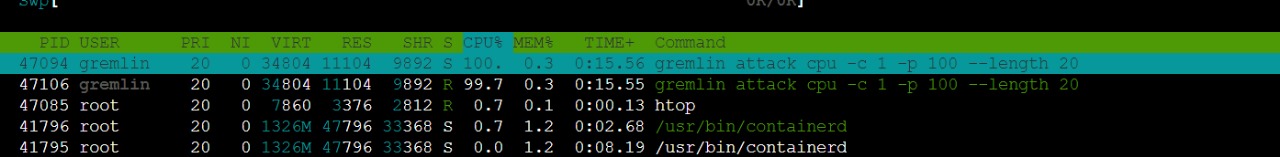
The CPU utilization percentage is 0.7 %

Login to gremlin , add the agent and then perform the experiment





After the attack is successful using “htop” command u can check CPU utilization in your VM



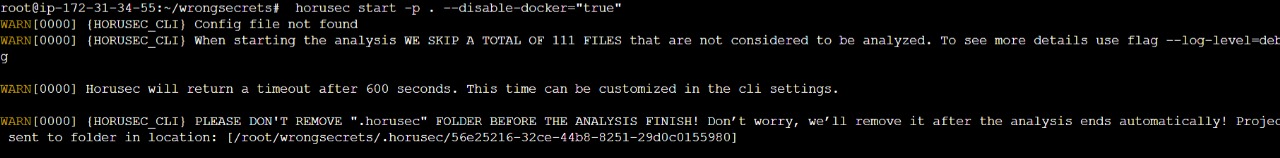
CPU utilization was increased to 100 % in the above picture

**Analysis 2:** Using Horusec

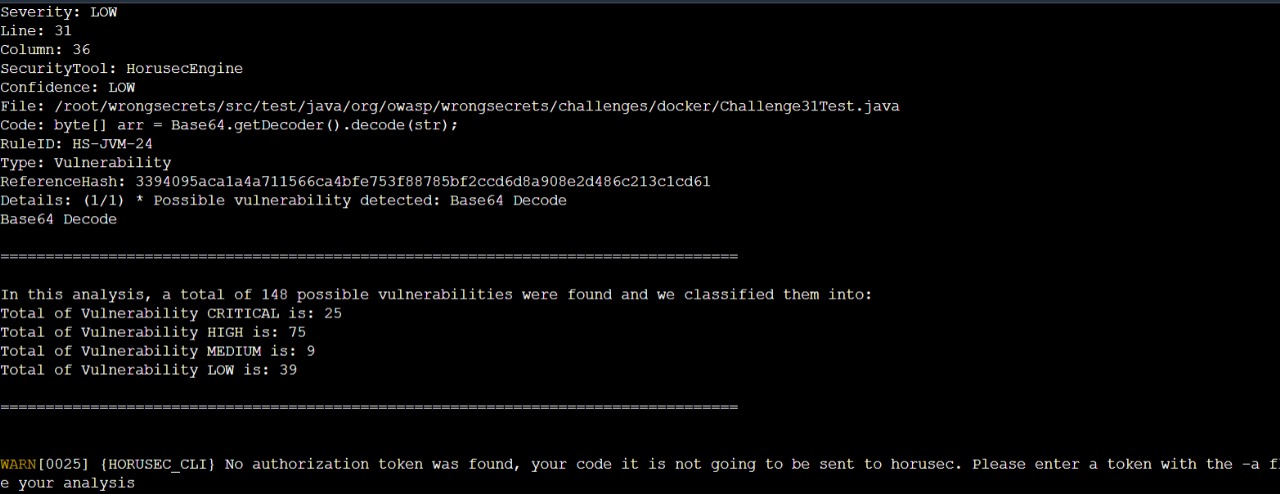
Clone the repo into the virtual machine



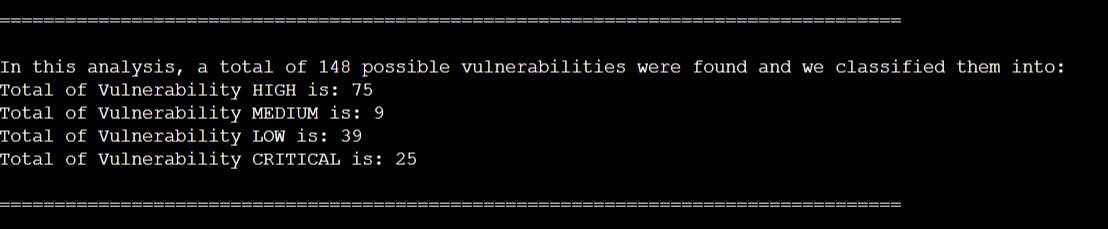
go into that directory perform to horusec scanning.



This command will scan the repository



This will give detailed analysis of the errors in that code



This picture shows the detailed analysis

High: 75

Medium: 9

Low: 39

Critical: 25

The following are some high vulnerability’s in the repo

**Possible vulnerability detected**: Crypto import

Sometimes, data might require preprocessing or cleaning before importing into Cyto. Ensure that your data is free from errors, duplicates, or inconsistencies that might hinder the import process.

**Possible vulnerability detected:** Weak block mode for Cryptographic Hash Function

Identify the specific cryptographic hash function or block mode being utilized in your application or system. Certain outdated or weakened hash functions (such as MD5 or SHA-1) might be susceptible to vulnerabilities due to advancements in attacks.

**Possible vulnerability detected:** Bad hexadecimal concatenation

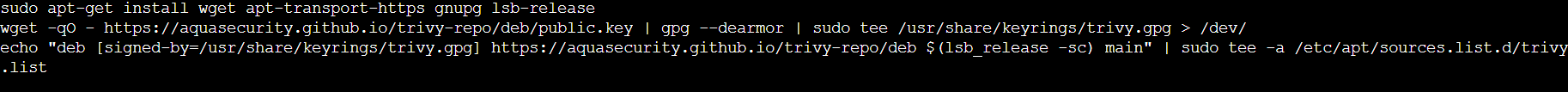
Adhere to secure coding guidelines and best practices when handling sensitive data, including hexadecimal values. This includes input validation, output encoding, and following secure data handling practices.

**Possible vulnerability detected**: Unsafe hash equals

Keep your software libraries, frameworks, and dependencies up-to-date. Apply security patches and updates provided by the respective vendors to address known vulnerabilities in hashing algorithms or comparison methods

**Analysis 3:** Using Trivy

first set up the trivy in the instance by using following commands

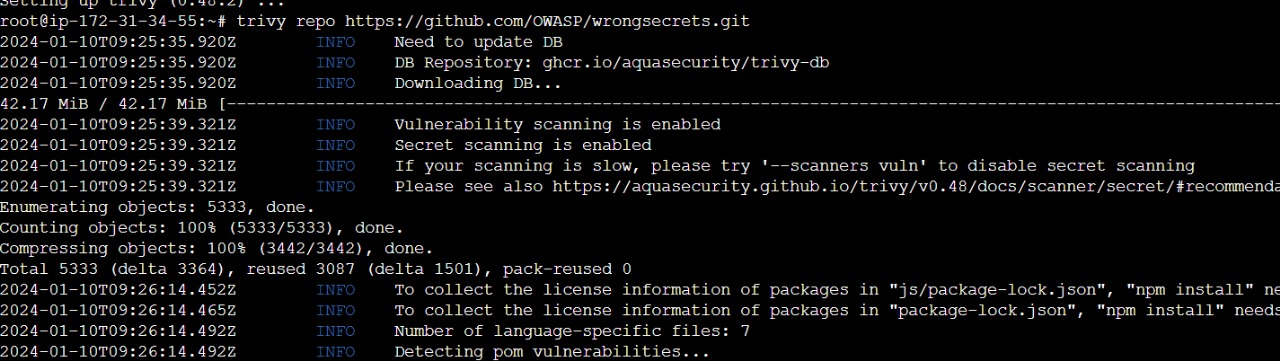


After install trivy

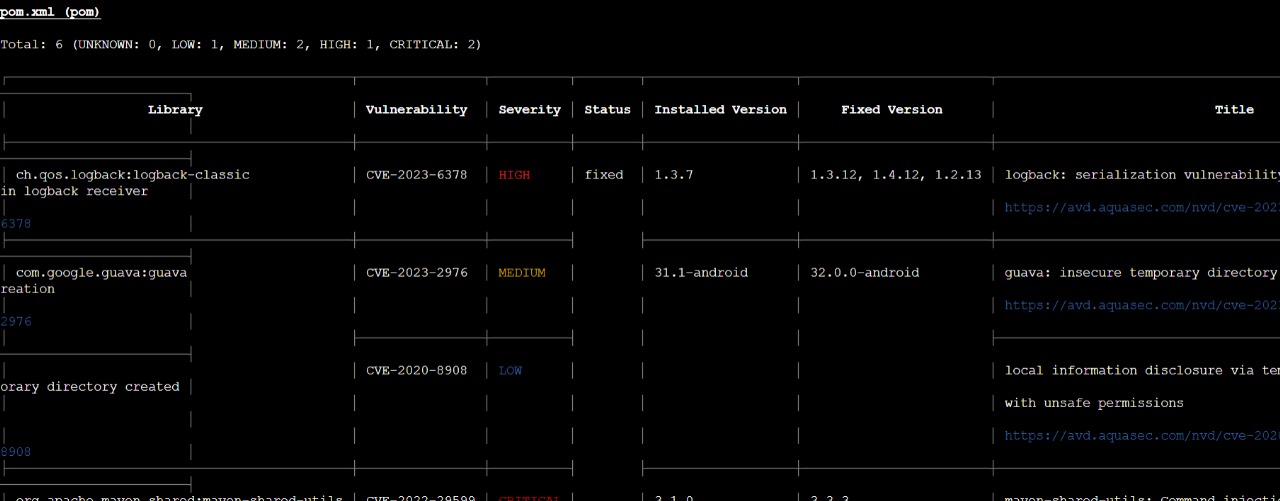
Clone the repo into the local machine



Use the following command to perform scanning in the repo



After the scanning completed it will generate the result as followin



Trivy finds vulnerability from that git repo

Critical: 2

High: 1

Medium: 2

Low: 1

The following are the vulnerability causes and their solutions

**Deserialization of Untrusted Data**: A serialization vulnerability in log back receiver component part of log back version 1.4.11 allows an attacker to mount a Denial-Of-Service attack by sending poisoned data. The product deserializes untrusted data without sufficiently verifying that the resulting data will be valid.

**Solution:** Make fields transient to protect them from deserialization. An attempt to serialize and then deserialize a class containing transient fields will result in NULLs where the transient data should be. This is an excellent way to prevent time, environment-based, or sensitive variables from being carried over and used improperly.

**Incorrect Permission Assignment for Critical Resource:** The product specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.

**Solution:** OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java Security Manager allows the software to specify restrictions on file operations.

**Origin Validation Error:** The product does not properly verify that the source of data or communication is valid.

**Solution:** The update in Apache Maven 3.8.1+ alters the default behavior to no longer follow HTTP-based repository references by default. Instead, it encourages the use of HTTPS (SSL/TLS) for repository URLs, promoting a more secure communication channel.

**Improper Certificate Validation:** Bouncy Castle For Java before 1.74 is affected by an LDAP injection vulnerability. The vulnerability only affects applications that use an LDAP Crestor from Bouncy Castle to validate X.509 certificates. During the certificate validation process, Bouncy Castle inserts the certificates Subject Name into an LDAP search filter without any escaping, which leads to an LDAP injection vulnerability.

**Solution:** Upgrade to Bouncy Castle version 1.74 or newer. This version contains fixes and patches to address the LDAP injection vulnerability. Updating to the latest secure version is the most effective way to mitigate this issue