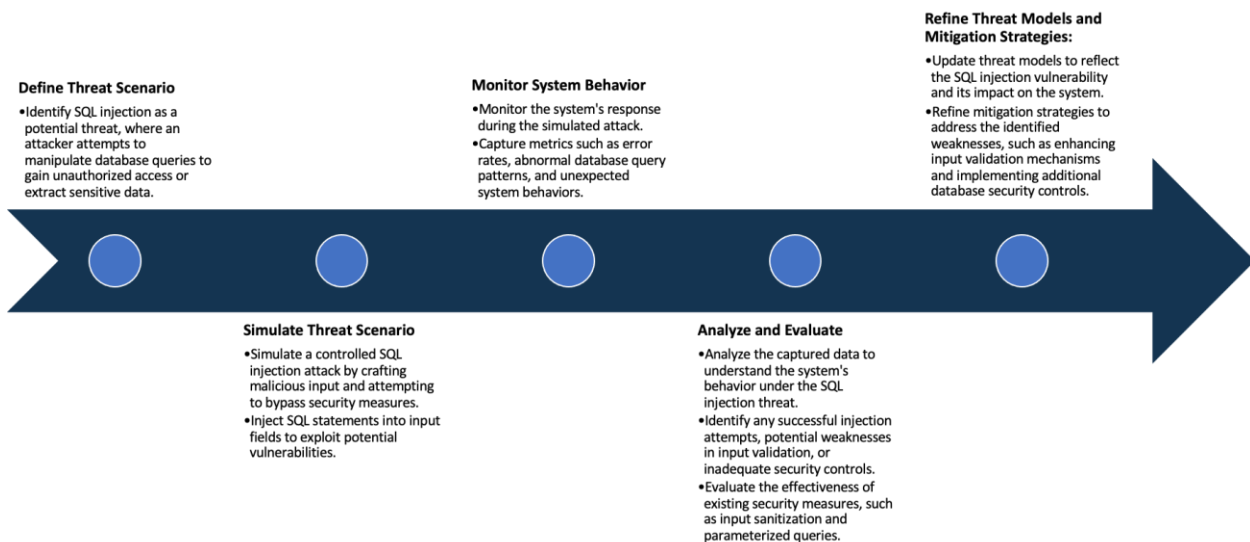


# Chaos Engineering - Report

Name:	Sunidhi Agrawal
Lab User ID:	23SEK3324_U11
Date:	09.01.2024
Application Name:	dvwps: Damn Vulnerable WordPress Site

## Follow the below guidelines:

Define Hypotheses and Scenarios:	Inject Controlled Failure:	Measure System Behavior:	Learn and Iterate:
<ul style="list-style-type: none"> <li>Similar to Chaos Engineering, start by defining hypotheses and scenarios related to potential threats.</li> </ul>	<ul style="list-style-type: none"> <li>Introduce controlled failure scenarios that mimic potential attack vectors or vulnerabilities identified in Threat Modeling.</li> <li>Simulate failure conditions, such as network disruptions, component failures, or data breaches, to observe the system's response.</li> </ul>	<ul style="list-style-type: none"> <li>Capture and measure relevant system behavior metrics during the chaos experiments.</li> <li>Monitor the system's response to the injected failures, including performance metrics, error rates, and security-related indicators.</li> <li>Analyze and compare the observed behavior against the expected outcomes defined in the Threat Modeling process.</li> </ul>	<ul style="list-style-type: none"> <li>Learn from the results of the chaos experiments and iterate on the Threat Modeling process.</li> <li>Analyze the observations and insights gained from the chaos experiments to refine the Threat Models. Update threat scenarios, adjust mitigation strategies, and improve security controls based on the lessons learned.</li> </ul>



# Chaos Engineering - Report

## 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)

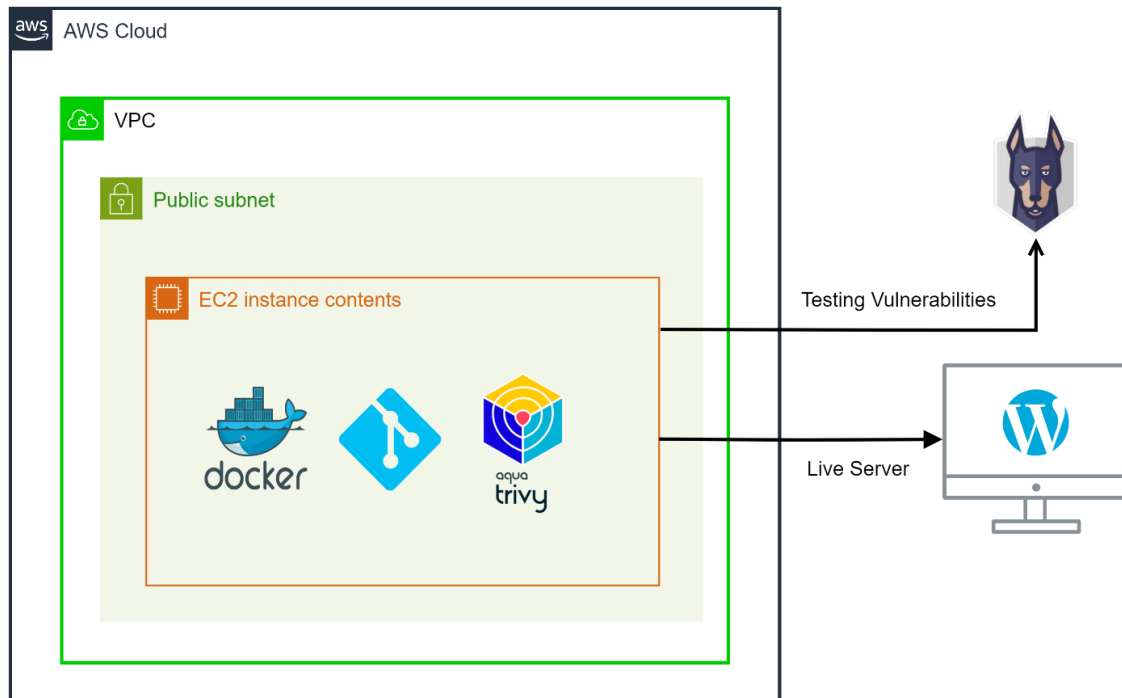


Figure 1: System Architecture

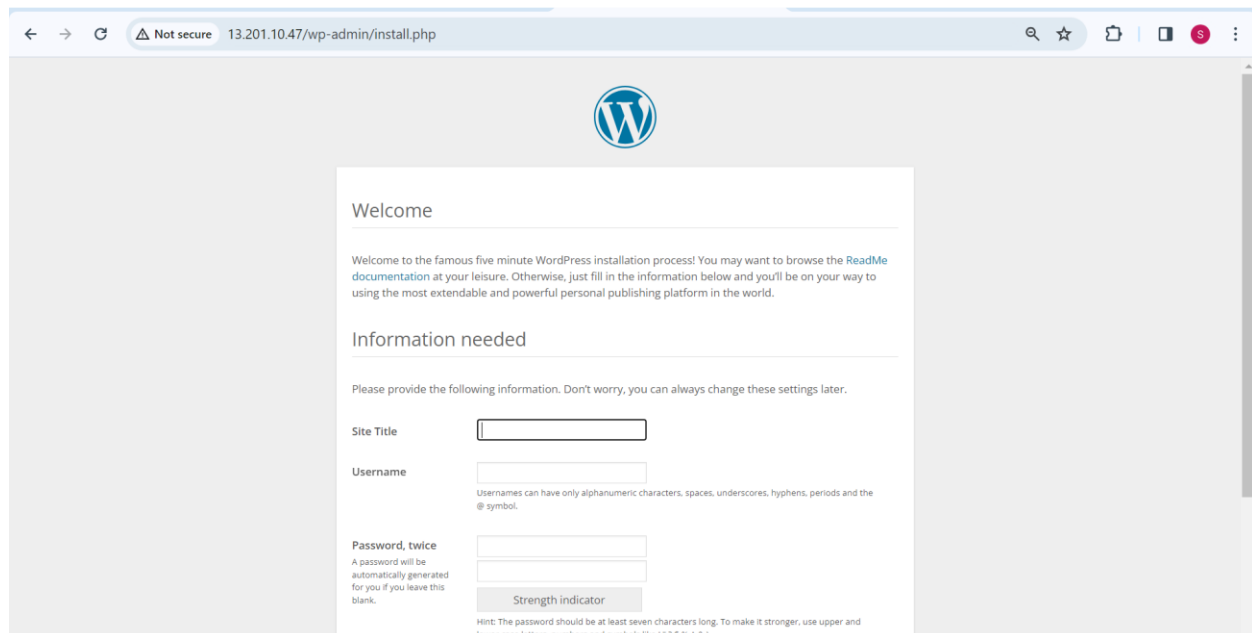


Figure 2: Live Web WordPress Server

# Chaos Engineering - Report

## 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 normal behavior of a system refers to its typical or expected state when operating under standard conditions.

In the context of a "dvwps: Damn Vulnerable WordPress Site" deployed on Docker running on an Ubuntu EC2 machine, defining the system's normal behaviour involves understanding the typical, expected state of the system when operating under standard conditions. Here are some aspects to consider when defining the normal behaviour of the system:

---

### **1. Website Availability:**

- **Steady State:** The website should be consistently accessible without interruptions.
- **Measurable Output:** Monitor the website's response time and ensure that it remains within acceptable limits.

### **2. WordPress Logs:**

- **Steady State:** WordPress application logs should not show unusual error rates or unexpected activities.
- **Measurable Output:** Regularly check WordPress logs for errors, warnings, or suspicious activities. Abnormal patterns may indicate potential security threats.

### **3. Security Monitoring:**

- **Steady State:** Regular security scans or monitoring should be in place to detect vulnerabilities or potential threats.
- **Measurable Output:** Use security tools to perform vulnerability assessments and regularly check for updates and patches.

### **4. User Authentication:**

- **Steady State:** User authentication and authorization mechanisms should function as expected.
- **Measurable Output:** Monitor login attempts and user activity. Unusual login patterns, multiple failed login attempts, or unauthorized access may indicate abnormal behaviour.

# Chaos Engineering - Report

## Hypothesis:

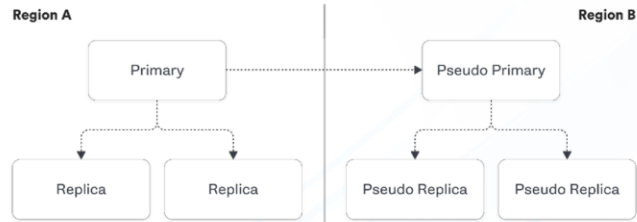
(During an experiment, we need a hypothesis for comparing to a stable control group, and the same applies here too. If there is a reasonable expectation for a particular action according to which we will change the steady state of a system, then the first thing to do is to fix the system so that we accommodate 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.")

### Known-Knowns

- We know that when a replica shuts down it will be removed from the cluster. We know that a new replica will then be cloned from the primary and added back to the cluster.

### Known-Unknowns

- We know that the clone will occur, as we have logs that confirm if it succeeds or fails, but we don't know the weekly average of the mean time it takes from experiencing a failure to adding a clone back to the cluster effectively.
- We know we will get an alert that the cluster has only one replica after 5 minutes but we don't know if our alerting threshold should be adjusted to more effectively prevent incidents.



### Unknown-Knowns

- If we shutdown the two replicas for a cluster at the same time, we don't know exactly the mean time during a Monday morning it would take us to clone two new replicas off the existing primary. But we do know we have a pseudo primary and two replicas which will also have the transactions.

### Unknown-Unknowns

- We don't know exactly what would happen if we shutdown an entire cluster in our main region, and we don't know if the pseudo region would be able to failover effectively because we have not yet run this scenario.

Known	<ul style="list-style-type: none"> <li>- <b>Vulnerabilities in WordPress:</b> The dvwps intentionally contains known vulnerabilities.</li> <li>- <b>Docker Deployment:</b> The project utilizes Docker for containerization, providing isolation for the WordPress site.</li> <li>- <b>Ubuntu EC2 Machine:</b> The WordPress site is hosted on an Ubuntu EC2 instance.</li> <li>- <b>Vulnerability Finding Tools:</b> Specific tools or scripts are known and will be used for experiments, i.e., Trivy &amp; Snyk</li> <li>- <b>Initial System State:</b> The initial state of the system, including the WordPress version and configurations, is known.</li> </ul>	<ul style="list-style-type: none"> <li>- <b>Chaos Impact Magnitude:</b> The exact impact of chaos experiments on the Damn Vulnerable WordPress site is unknown.</li> <li>- <b>Resilience Levels:</b> The system's resilience against different types of failures is uncertain.</li> <li>- <b>Interaction with Vulnerabilities:</b> Potential interactions between known vulnerabilities in the WordPress site are not fully understood.</li> <li>- <b>Network Behaviour:</b> The behavior of the network during chaos experiments is not precisely predicted.</li> </ul>
Unknown	<ul style="list-style-type: none"> <li>- <b>Undiscovered Vulnerabilities:</b> There may be additional vulnerabilities in the WordPress site that were not intentionally introduced.</li> <li>- <b>Dependency Interactions:</b> Unforeseen interactions between Docker, Ubuntu, and WordPress dependencies may influence system behaviour.</li> </ul>	<ul style="list-style-type: none"> <li>- <b>Emergent Behaviours:</b> Unpredicted behaviours may emerge due to the combination of vulnerabilities, Docker, and chaos experiments.</li> <li>- <b>System Collapse Scenarios:</b> Unanticipated scenarios leading to system collapse or severe degradation.</li> <li>- <b>New Attack Vectors:</b> The chaos experiments might reveal new attack vectors that were not initially considered.</li> </ul>

---

In the context of chaos engineering, a chaos hypothesis is a statement that predicts how a system will respond to a particular event or experiment. It is a crucial part of the chaos engineering process as it provides a basis for comparison between the normal, stable state of the system (control group) and the state during the chaos experiment.

In this project if the web server hosting the Damn Vulnerable WordPress Site (DVWPS) encounters a sudden increase in traffic or experiences a simulated attack, the system will maintain its functionality and uptime. We expect that, despite potential disruptions, the DVWPS will be resilient and capable of handling the stress, ensuring that users can still access the site without encountering downtime or loss of data. This hypothesis is grounded in the assumption that our implemented chaos experiments will trigger responses and safeguards within the system, such as auto-scaling or failover mechanisms, preventing significant disruptions and maintaining the expected level of service reliability.

# Chaos Engineering - Report

## Experiment:

(Document your Preparation, Implementation, Observation and Analysis)

### **Project Overview:**

The goal of this project is to assess the resilience of a web server hosting a vulnerable WordPress site by leveraging Chaos Engineering principles. The project utilizes Docker to containerize the Damn Vulnerable WordPress site (DVWPS) and related services. Finding vulnerabilities is also a main objective of this project, the project utilizes Trivy and Snyk for finding and accessing vulnerabilities.

### **Tools Used:**

#### Docker:

- Purpose: Containerization tool used to package the DVWPS and its dependencies.
- Explanation: Docker enables the creation of isolated containers, ensuring consistency across different environments.

#### GitHub:

- Purpose: Version control and source code management.
- Explanation: Cloning the DVWPS repository from GitHub ensures the use of the latest version of the vulnerable WordPress site.

#### MySQL Docker Image:

- Purpose: Database service for the WordPress site.
- Explanation: The MySQL Docker image is used to create a containerized MySQL database for storing WordPress data.

#### Trivy:

- Purpose: Trivy is utilized in this project for vulnerability scanning within the Docker containers.
- Explanation: By running Trivy on your container images, it analyzes the installed packages and compares them against known vulnerability databases.

#### Snyk:

- Purpose: Snyk is employed in this project to address security concerns by identifying and mitigating vulnerabilities in both the Docker container images and project dependencies.
- Explanation: The tool contributes to the overall goal of creating a secure and resilient environment for the Damn Vulnerable WordPress site and its supporting infrastructure.

#### Chaos Engineering:

- Purpose: To introduce controlled chaos and observe system behavior.
- Explanation: Chaos experiments are designed to uncover vulnerabilities, improve system resilience, and enhance overall reliability.

## Implementation:

### 1. Cloning the DVWPS Repository:

- **Command:**  
`git clone https://github.com/vianasw/dvwps.git`  
`cd dvwps`
- **Explanation:** Cloning the DVWPS repository from GitHub to obtain the necessary files for the vulnerable WordPress site.

### 2. Building MySQL Datastore Docker Image:

- **Command:**  
`vi Dockerfile.mysql_datastore`
- **Explanation:** Creating a Dockerfile for the MySQL datastore, specifying the base image, and configuring the volume. The docker build command is then used to build the MySQL Datastore Docker image.

### 3. Running MySQL Datastore Container:

- **Command:**  
`docker run --name wp_data sunidhiagrawal/mysql_datastore`
- **Explanation:** Creating a Docker container named wp\_data using the MySQL Datastore image to store MySQL data.

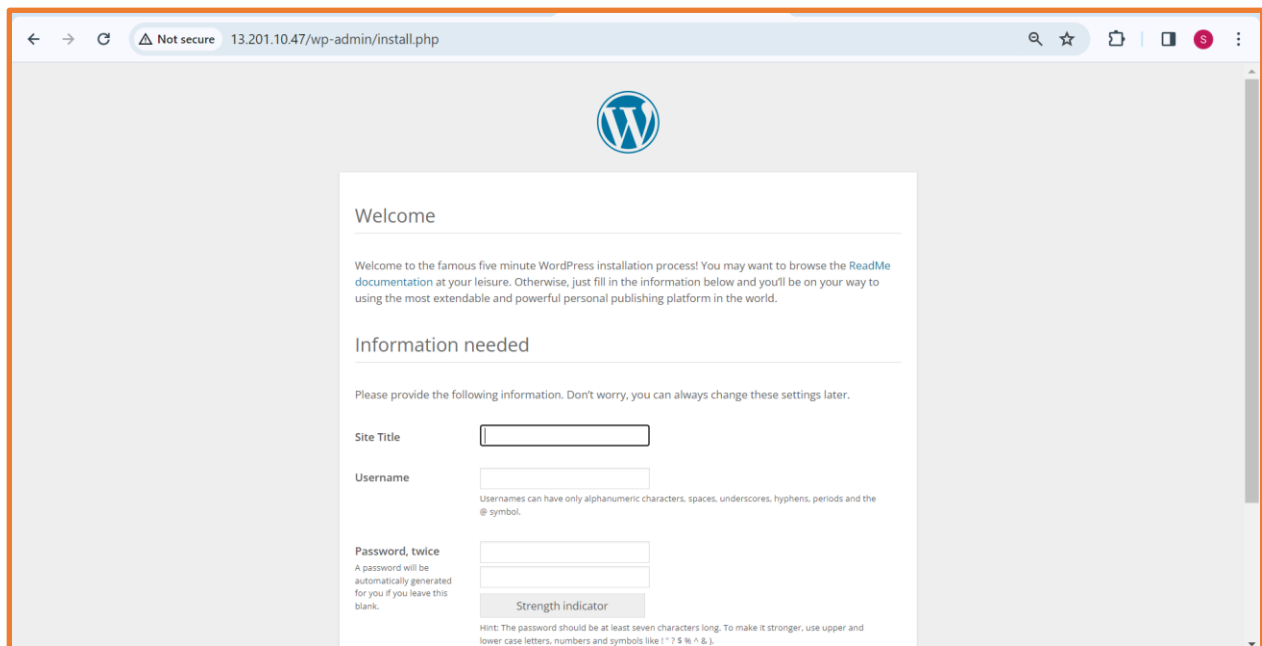
### 4. Running MySQL Database Container:

- **Command:**  
`docker run --name wp_db -p 3306:3306 --volumes-from wp_data -v /root/dvwps/configs:/etc/mysql/conf.d -e MYSQL_ROOT_PASSWORD=password -e MYSQL_USER=wordpressuser -e MYSQL_PASSWORD=password -e MYSQL_DATABASE=wordpress -d mysql:5.6`
- **Explanation:** Creating a Docker container named wp\_db using the MySQL image, exposing port 3306, and configuring MySQL settings. This container links to the wp\_data container to ensure data persistence.

# Chaos Engineering - Report

## 5. Running DVWPS Container:

- **Command:**  
`docker run -d --link wp_db:mysql -p 80:80 -e DB_NAME='wordpress' -e DB_USER='wordpressuser' -e DB_HOST='wp_db' -e DB_PASSWORD='password' vianasw/dvwps`
- **Explanation:** Creating a Docker container running DVWPS, linking it to the MySQL database container, and exposing port 80 to make the WordPress site accessible.



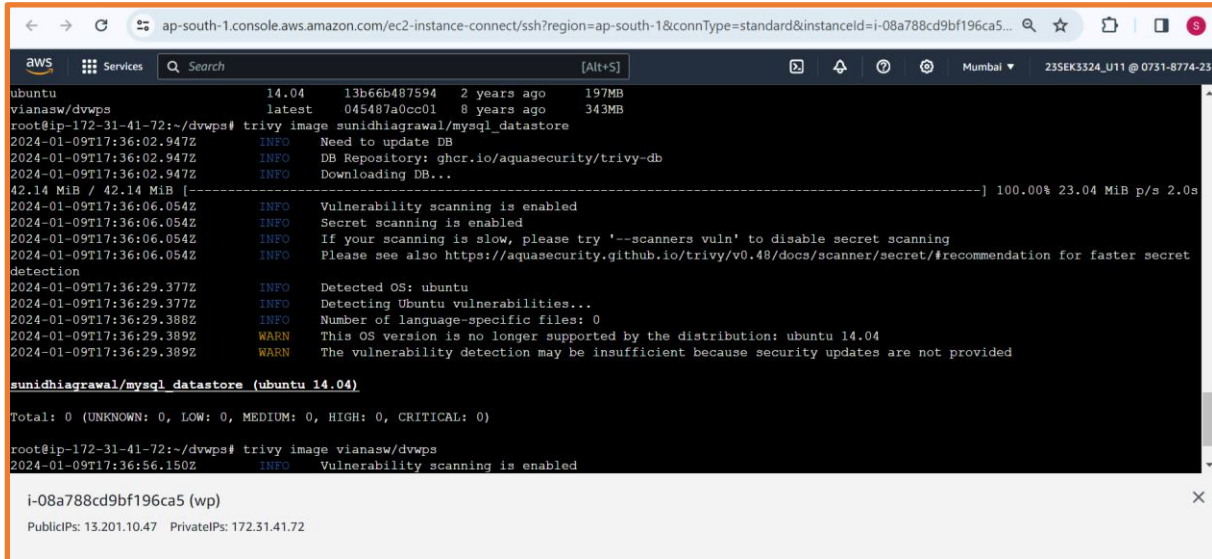
## 6. Vulnerability Scanning with Trivy:

- **Command:**  
`trivy image sunidhiagrawal/mysqlidatabase`  
`trivy image vianasw/dvwps`  
`trivy image mysql(oracle 8.9)`
- **Explanation:** Integrate Trivy into the Docker workflow to scan container images for vulnerabilities. Identify and analyze potential security issues within the Dockerized environment.



# Chaos Engineering - Report

**Number of Vulnerabilities:** TOTAL: 0 (UNKOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 0, CRITICAL: 0)



```

aws
Services
Search
[Alt+S]
Mumbai 23SEK3324_U11 @ 0731-8774-236

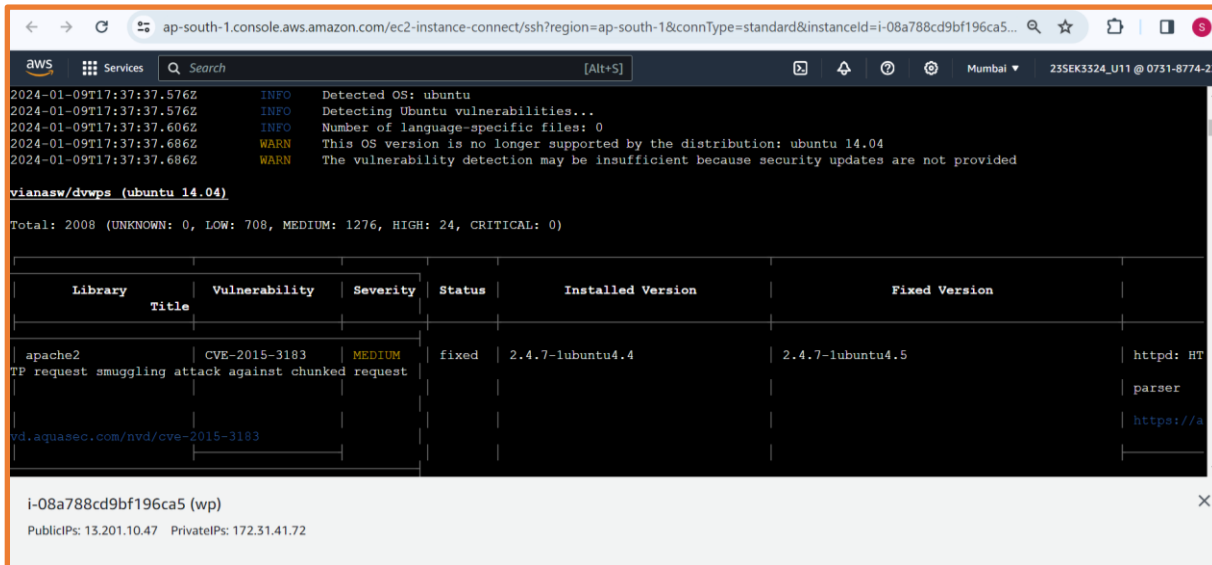
ubuntu 14.04 13b66b487594 2 years ago 197MB
vianasw/dwvps latest 045487a0cc01 8 years ago 343MB
root@ip-172-31-41-72:~/dwvps# trivy image sunidhiagrawal/mysql_datastore
2024-01-09T17:36:02.947Z INFO Need to update DB
2024-01-09T17:36:02.947Z INFO DB Repository: ghor.io/aquasecurity/trivy-db
2024-01-09T17:36:02.947Z INFO Downloading DB...
42.14 MiB / 42.14 MiB [-----] 100.00% 23.04 MiB p/s 2.0s
2024-01-09T17:36:06.054Z INFO Vulnerability scanning is enabled
2024-01-09T17:36:06.054Z INFO Secret scanning is enabled
2024-01-09T17:36:06.054Z INFO If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2024-01-09T17:36:06.054Z INFO Please see also https://aquasecurity.github.io/trivy/v0.48/docs/scanner/secret/#recommendation for faster secret
detection
2024-01-09T17:36:29.377Z INFO Detected OS: ubuntu
2024-01-09T17:36:29.377Z INFO Detecting Ubuntu vulnerabilities...
2024-01-09T17:36:29.388Z INFO Number of language-specific files: 0
2024-01-09T17:36:29.389Z WARN This OS version is no longer supported by the distribution: ubuntu 14.04
2024-01-09T17:36:29.389Z WARN The vulnerability detection may be insufficient because security updates are not provided

sunidhiagrawal/mysql_datastore (ubuntu 14.04)
Total: 0 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 0, CRITICAL: 0)
root@ip-172-31-41-72:~/dwvps# trivy image vianasw/dwvps
2024-01-09T17:36:56.150Z INFO Vulnerability scanning is enabled

i-08a788cd9bf196ca5 (wp)
PublicIPs: 13.201.10.47 PrivateIPs: 172.31.41.72
  
```

Figure 3: trivy image sunidhiagrawal/mysql\_database

**Number of Vulnerabilities:** TOTAL: 2008 (UNKOWN: 0, LOW: 708, MEDIUM: 1276, HIGH: 24, CRITICAL: 0)



```

aws
Services
Search
[Alt+S]
Mumbai 23SEK3324_U11 @ 0731-8774-236

2024-01-09T17:37:37.576Z INFO Detected OS: ubuntu
2024-01-09T17:37:37.576Z INFO Detecting Ubuntu vulnerabilities...
2024-01-09T17:37:37.606Z INFO Number of language-specific files: 0
2024-01-09T17:37:37.686Z WARN This OS version is no longer supported by the distribution: ubuntu 14.04
2024-01-09T17:37:37.686Z WARN The vulnerability detection may be insufficient because security updates are not provided

vianasw/dwvps (ubuntu 14.04)
Total: 2008 (UNKNOWN: 0, LOW: 708, MEDIUM: 1276, HIGH: 24, CRITICAL: 0)

Library Title Vulnerability Severity Status Installed Version Fixed Version
-----
apache2 CVE-2015-3183 MEDIUM fixed 2.4.7-1ubuntu4.4 2.4.7-1ubuntu4.5
TP request smuggling attack against chunked request
vd.aquasec.com/nvd/cve-2015-3183 https://a

i-08a788cd9bf196ca5 (wp)
PublicIPs: 13.201.10.47 PrivateIPs: 172.31.41.72
  
```

Figure 4: trivy image vianasw/dwvps

# Chaos Engineering - Report

**Number of Vulnerabilities:** TOTAL: 5 (UNKOWN: 0, LOW: 0, MEDIUM: 4, HIGH: 1, CRITICAL: 0)

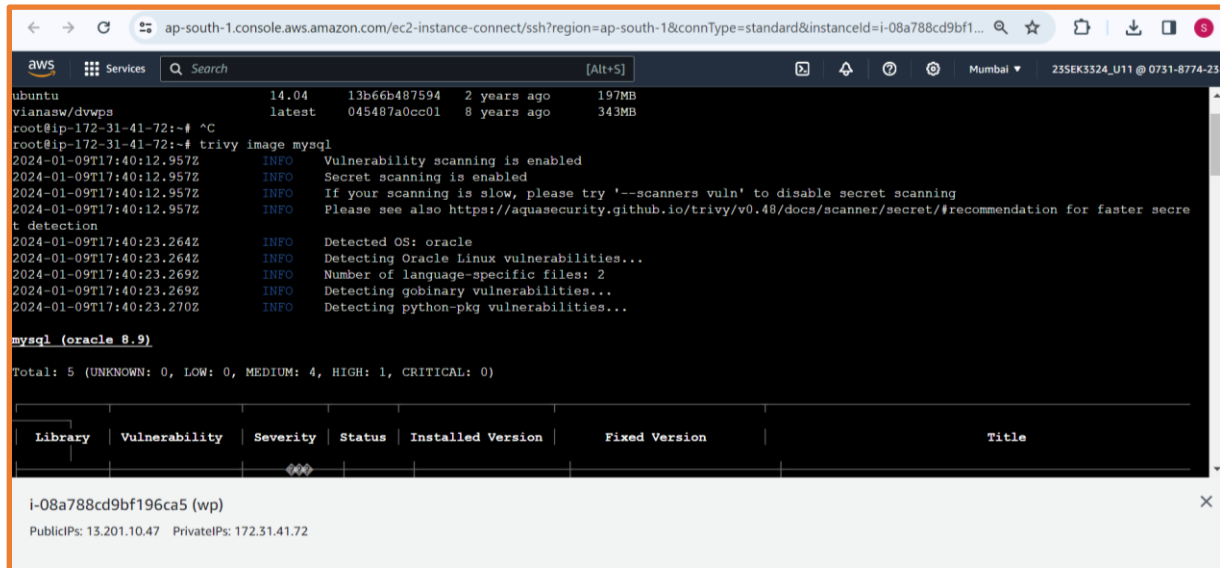


Figure 5: trivy image mysql(oracle 8.9)

## 7. Security Analysis with Snyk:

- **Explanation:** Utilize Snyk to scan project dependencies and Docker container images for known vulnerabilities. Receive continuous monitoring alerts and remediation guidance to enhance security.

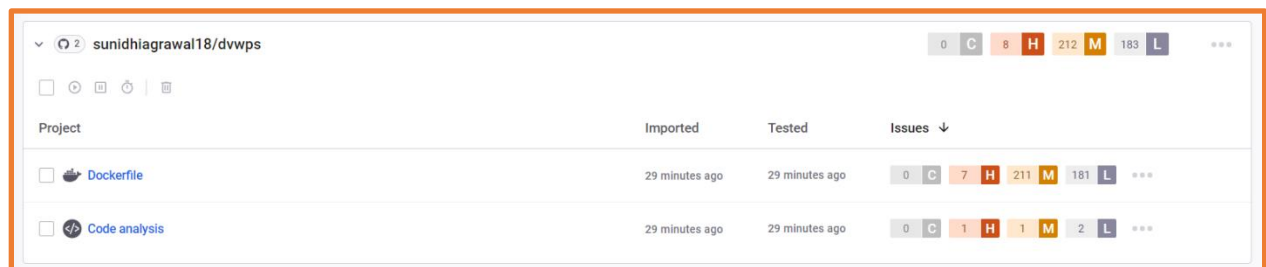


Figure 6: Snyk Vulnerability scan of DVWPS

## Observation and Analysis:

### Vulnerabilities Identified in the Project:

#### - Cross-Site Scripting (XSS) - CWE-79:

- **Description:** A cross-site scripting attack occurs when an attacker tricks a web application into accepting a request from a trusted source. By escaping the web application's context, the attacker delivers malicious data alongside trusted dynamic content.

#### - Sudo Off-by-one Error - CVE-2021-3156:

- **Description:** Sudo before version 1.9.5p2 contains an off-by-one error leading to a heap-based buffer overflow. This vulnerability allows privilege escalation to root through the "sudoedit -s" command with a command-line argument ending with a single backslash character.

#### - Expat/libexpat1 Exposure of Resource to Wrong Sphere - CVE-2022-25236:

- **Description:** Expat (libexpat) before version 2.4.5 allows attackers to insert namespace-separator characters into namespace URIs, potentially leading to exposure of resources to the wrong sphere.

#### - Expat/libexpat1 Improper Encoding or Escaping of Output - CVE-2022-25235:

- **Description:** Expat (libexpat) before version 2.4.5 lacks certain validation of encoding, specifically missing checks for whether a UTF-8 character is valid in each context, potentially resulting in improper encoding or escaping of output.

#### - OpenSSL Loop with Unreachable Exit Condition ('Infinite Loop') - CVE-2022-0778:

- **Description:** The BN\_mod\_sqrt() function in OpenSSL contains a bug that can cause an infinite loop for non-prime moduli while computing a modular square root.

#### - Cyrus SASL2/libsasl2-modules-db SQL Injection - CVE-2022-24407:

- **Description:** In Cyrus SASL version 2.1.17 through 2.1.27 before 2.1.28, the plugins/sql.c component does not escape the password for a SQL INSERT or UPDATE statement, potentially leading to SQL injection.

These vulnerabilities represent a range of security issues, including XSS, heap-based buffer overflow, improper handling of encoding, infinite loop, and SQL injection. Addressing these vulnerabilities is crucial for ensuring the security and resilience of the Damn Vulnerable WordPress site and its supporting infrastructure. It is recommended to apply patches, updates, or other remediation measures to mitigate these risks. Additionally, continuous monitoring and proactive security practices should be implemented to prevent the introduction of such vulnerabilities in the future.

## **Conclusion:**

The Damn Vulnerable WordPress Site (DVWPS) chaos engineering project has provided valuable insights into the resilience and vulnerabilities of a web application deployed on Docker within an Ubuntu EC2 environment. Through a systematic series of chaos experiments, the project aimed to assess the system's response to controlled disruptions, identify weaknesses, and enhance overall understanding of the interplay between known vulnerabilities, containerization, and infrastructure components.