|  |  |
| --- | --- |
| Name: | Prajwal R |
| Lab User ID: | 23SEK3324\_U07 |
| Date: | 10/01/2024 |
| Application Name: | Damn Vulnerable WordPress Site |

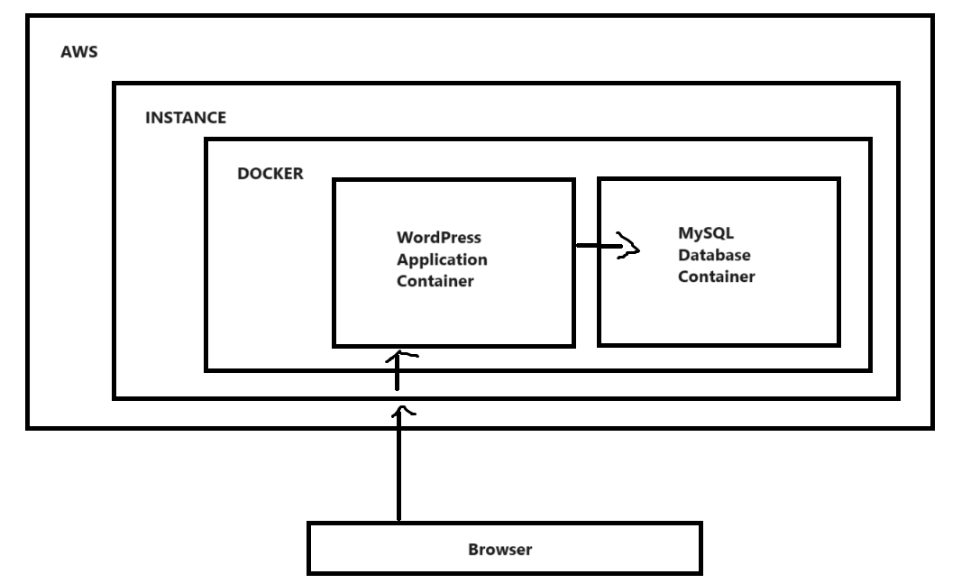
**Follow the below guidelines:**





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)



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 system's normal behavior in the Damn Vulnerable WordPress project involves ensuring the websites continues

availability, maintaining database connectivity, and monitoring resource utilization within acceptable limits. Regularly

tracking HTTP response codes confirms the website's responsiveness, while database queries and absence of

connection errors indicate proper interaction between the WordPress and MySQL containers.

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.")



Using the Gremlin when one of the pod is shutdown then in order to maintain the amount of replica the deployment service in the Kubernetes will deploy the new replica to match the amount of the replica

**Known**

Things we are aware of but don’t understand.

Things we are aware of and understand.

**Unknown**

**Unknown**

**Known**

Things we are neither aware of nor understand.

Things we understand but are not aware of.

Experiment:

(Document your Preparation, Implementation, Observation and Analysis )

This Project is done in the AWS instance, We use a Ubuntu 20.04 Machine. We first create the machine and then we attach to the machine and update the machine using “apt Update” and then we start to do the task.

At the first we scan the repository using the Synk tool. Below is the output of the issues found using the Synk tool:

A screenshot of a computer

Description automatically generated

Since the number of the issues is so much to explain in the document so I have explained some issues based on their scores and impact they hold on the project:

The application currently has a total of 0 critical, 8 high, 212 medium, and 183 low-severity issues. These vulnerabilities encompass various aspects, including code quality and security concerns.

Dockerfile Vulnerabilities:

CWE-193: Off-by-one Error:

An off-by-one error may lead to incorrect calculations or usage of maximum or minimum values, potentially causing unexpected behavior.

Mitigation: Ensure correct calculation and usage of maximum or minimum values to prevent off-by-one errors.

CWE-755: Improper Handling of Exceptional Conditions:

Improper handling of exceptional conditions can lead to unexpected issues or errors.

Mitigation: Implement proper handling mechanisms for exceptional conditions to ensure robustness and prevent unexpected behavior.

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'):

The vulnerability arises from the improper neutralization of special elements in SQL commands, exposing the application to potential SQL injection attacks.

Mitigation: Implement proper input validation and parameterized queries to neutralize special elements and prevent SQL injection.

Code Quality Issues:

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'):

Failure to neutralize user-controllable input before rendering web pages may lead to cross-site scripting (XSS) vulnerabilities.

Mitigation: Implement proper input validation and output encoding to neutralize user-controllable input and prevent XSS attacks.

CWE-532: Insertion of Sensitive Information into Log File: Privacy Leak:

Inserting sensitive information into log files poses a privacy leak risk, potentially providing valuable guidance to attackers.

Mitigation: Avoid logging sensitive information or ensure proper access controls and encryption for log files to prevent privacy leaks.

CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute:

Not setting the 'Secure' attribute for sensitive cookies in HTTPS sessions may expose them in plaintext over HTTP.

Mitigation: Ensure that sensitive cookies in HTTPS sessions have the 'Secure' attribute set to prevent exposure over insecure channels.

By addressing these vulnerabilities and implementing the recommended mitigations, the application can enhance its overall security posture and minimize the risk of potential exploits. Regularly monitor and update security measures to stay resilient against evolving threats.

Project Requirements:

* Docker: Install Docker using the below command :

apt install docker.io -y

Setup:

The project is started by cloning the GitHub repo onto the local machine

git clone <https://github.com/vianasw/dvwps.git>

The project requires a Database for the website to be working so we first create the database :

FROM ubuntu:14.04

MAINTAINER Prajwal R <prajwa1rg0d@gmail.com>

VOLUME /var/lib/mysql

CMD ["true"]

We save the above content in a file called as ‘dockerfile’ and create a docker image from the above using the command :

docker build -t joyboy/mysql\_datastore .

Then we create a data volume container from the above image using the command :

docker run --name wp\_data username/mysql\_datastore

The purpose of this container is to act as a data volume container, providing persistent storage for the MySQL database.

Then we create a MySQL database container :

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

This command creates a Docker container named "wp\_db" using the "mysql:5.6" image.

* The -p 3306:3306 flag maps the container's MySQL port (3306) to the host machine's port 3306, allowing external access.
* The --volumes-from wp\_data flag links the data volume container "wp\_data" to the "wp\_db" container, ensuring persistent storage for the MySQL data.
* The -v /root/dvwps/configs/:/etc/mysql/conf.d flag mounts the local directory "/root/dvwps/configs/" to the container's "/etc/mysql/conf.d" directory, allowing custom MySQL configurations.
* The -e flags set environment variables for MySQL, including the root password, a WordPress user, its password, and the WordPress database name.
* The -d flag runs the container in detached mode (in the background).

Then we go to the repo that we have clone using ‘cd’ and then we create a docker image from the dockerfile that has been given in the repo before that we edit some path of the files that need to be copied from the host machine into image in the dockerfile and execute the below command.   
  
docker build -t joyboy/dvwps .

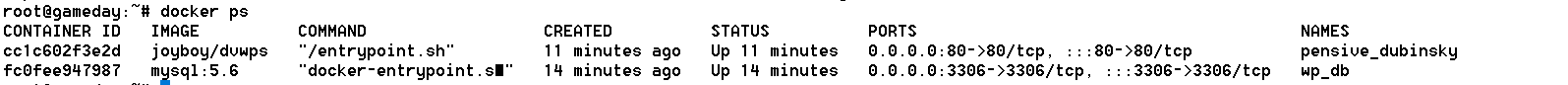
Then we execute the below command:

docker run -d --link wp\_db:mysql -p 80:80 -e DB\_NAME='wordpress' -e DB\_USER='wordpressuser' -e DB\_HOST='172.31.2.251' -e DB\_PASSWORD='password' joyboy/dvwps

* -d: Runs the Docker container in detached mode, meaning it runs in the background, allowing you to use the terminal for other commands.
* --link wp\_db:mysql: Establishes a link between the running container named "wp\_db" (assumed to be the MySQL container) and the new container being created. This allows the new container to access the MySQL container using the alias "mysql."
* -p 80:80: Maps port 80 from the host machine to port 80 on the container. This is typically done for web applications to make them accessible on the standard HTTP port.
* -e DB\_NAME='wordpress': Sets the environment variable DB\_NAME inside the container to 'wordpress'. This likely specifies the name of the WordPress database.
* -e DB\_USER='wordpressuser': Sets the environment variable DB\_USER inside the container to 'wordpressuser'. This likely specifies the username for accessing the WordPress database.
* -e DB\_HOST='172.31.2.251': Sets the environment variable DB\_HOST inside the container to '172.31.2.251'. This is likely the IP address where the MySQL database is hosted.
* -e DB\_PASSWORD='password': Sets the environment variable DB\_PASSWORD inside the container to 'password'. This likely specifies the password for accessing the WordPress database.
* joyboy/dvwps: Specifies the Docker image to be used for creating the container, in this case, 'joyboy/dvwps.'

Then open the Public IP of the machine in the browser to see the website at the first we get a wordpress admin page set the title, name and email of the author

When you type in the command ‘docker ps’ you see the below output



Type in ‘netstat -tulnp’ you see the port that are working and listening on the machine

A close-up of a test

Description automatically generated

Below is the image how the page looks like

A screenshot of a computer

Description automatically generated

Below is the wordpress admin application

A screenshot of a computer

Description automatically generated

Using the OWASP ZAP we try to find the vulnerabilites in the app using the below command :

docker run -t ghcr.io/zaproxy/zaproxy:stable zap-baseline.py -t http://43.204.216.149/

Vulnerability Report:

Summary:

IP Address: 43.204.216.149

Domain: ec2-43-204-216-149.ap-south-1.compute.amazonaws.com

Web Server: Apache/2.4.7 (Ubuntu)

PHP Version: 5.5.9

WordPress Version: 3.8.1

Issue: Missing Permissions Policy header.

Mitigation: Add a Permissions Policy header to control which features can be used in the browser.

Issue: Missing X-Frame-Options header.

Mitigation: Set X-Frame-Options to DENY to prevent clickjacking attacks.

Issue: Missing X-Permitted-Cross-Domain-Policies header.

Mitigation: Add X-Permitted-Cross-Domain-Policies with a secure configuration.

Issue: Missing Referrer Policy header.

Mitigation: Implement Referrer Policy to control how much referrer information is sent with requests.

Issue: Missing Strict-Transport-Security header.

Mitigation: Enable Strict-Transport-Security to force the use of secure connections.

Content-Security-Policy:

Issue: Missing Content-Security-Policy header.

Mitigation: Implement a Content-Security-Policy to mitigate various types of attacks, including XSS.

Cross-Origin-Embedder-Policy:

Issue: Missing Cross-Origin-Embedder-Policy header.

Mitigation: Add Cross-Origin-Embedder-Policy to control how a document loads cross-origin resources.

Issue: Missing Cross-Origin-Opener-Policy header.

Mitigation: Implement Cross-Origin-Opener-Policy for better control over cross-origin pages.

Issue: Missing Cross-Origin-Resource-Policy header.

Mitigation: Add Cross-Origin-Resource-Policy to control how resources can be shared cross-origin.

Issue: Missing X-Content-Type-Options header.

Mitigation: Enable X-Content-Type-Options to prevent browsers from interpreting files as a different MIME type.

Issue: Missing Clear-Site-Data header.

Mitigation: Implement Clear-Site-Data to clear browsing data for a given origin.

Issue: Mixed content found in scripts.

Mitigation: Ensure all script resources are loaded over HTTPS to prevent mixed content issues.

Issue: Mixed content found in link resources.

Mitigation: Load all link resources over HTTPS to avoid mixed content vulnerabilities.

WordPress Plugins:

Info: Vulnerable Akismet plugin version 2.5.9. Consider updating to the latest version.

WordPress Directories:

Info: Directory listing found in /wp-content/uploads/ and /wp-includes/.

Additional Information:

SSH Authentication Methods:

Info: SSH server allows public key authentication.

Recommendations:

Implement missing security headers on the web server.

Ensure all resources are loaded over HTTPS to prevent mixed content.

Regularly update WordPress and its plugins to the latest versions.

Disable directory listing to enhance security.

Consider implementing multi-factor authentication for SSH.

Remember to conduct regular security assessments and follow best practices to maintain a secure online presence.

Using Gremlin the below test is done:

Shutdown Experiment:

Observation: The application experienced a shutdown, resulting in the system being unavailable.

Analysis: The shutdown experiment effectively demonstrated that the system is susceptible to a loss of availability, which could be caused by unexpected outages or intentional disruptions.

Recommendation: To enhance resilience, consider implementing redundancy, failover mechanisms, and proactive monitoring to minimize downtime.

Blackhole Experiment:

Observation: The application remained functional despite the blackhole experiment, indicating resilience to network disruptions.

Analysis: The system demonstrated robustness in handling network blackholes, suggesting that it may have effective error handling or that it is well-insulated from network disruptions.

Recommendation: Continue to evaluate and enhance network resilience. Ensure proper error handling and implement fallback mechanisms for critical network operations.

Latency Experiment:

Observation: The application remained operational during the latency experiment, suggesting tolerance to delays in response times.

Analysis: The system's ability to handle increased response times indicates a certain degree of tolerance to latency, potentially due to optimized resource allocation or asynchronous processing.

Recommendation: Continue monitoring and optimizing performance to ensure the application can gracefully handle variations in response times.