|  |  |
| --- | --- |
| Name: | Prajwal R |
| Lab User ID: | 23SEK3324\_U07 |
| Date: | 10/01/2024 |
| Application Name: | OWASP Juice Shop |

**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)

A screenshot of a computer

Description automatically generated

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 state for the OWASP Juice Shop refers to its expected and secure configuration, where the application operates without being compromised by security vulnerabilities. In this state, the OWASP Juice Shop should adhere to secure configurations, ensuring that sensitive operations such as session management, input validation, and secure communication protocols are appropriately implemented. Key security headers, including Content Security Policy (CSP), X-Frame-Options, and X-Content-Type-Options, should be correctly configured to enhance the overall security posture. The application should be devoid of common security vulnerabilities, such as Cross-Site Scripting (XSS), SQL Injection, Cross-Site Request Forgery (CSRF), and other OWASP Top Ten vulnerabilities. Authentication mechanisms should be robust, ensuring that users are authenticated securely, and authorization controls should be in place to restrict access to resources based on user privileges.

Furthermore, in the normal state, the OWASP Juice Shop should exhibit sound data security practices, safeguarding sensitive information with encryption and secure storage methodologies. The absence of default credentials and the implementation of strong authentication mechanisms contribute to a secure user authentication and authorization environment. The application should be free from known exploits, with regular updates and prompt application of security patches to mitigate emerging threats. This normal state aligns with functional and usable characteristics, emphasizing the importance of balancing security measures with the application's usability. By maintaining this secure baseline, the OWASP Juice Shop enhances its resilience against potential cyber threats and provides a trustworthy environment for users and developers alike.

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**

Things we are aware of but don’t understand.

Things we are aware of and understand.

942236

**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 and Trivy.

Below is the output of the issues found using the Synk tool:

A screenshot of a computer

Description automatically generated

Using the trivy to find the vulnerabilities the total are :

trivy image ec0465c6c0c2

Total: 18 (UNKNOWN: 0, LOW: 11, MEDIUM: 7, HIGH: 0, CRITICAL: 0)

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

CVE-203-46233: Asymmetric Private Key Exposure

Error Description:

The private key used for asymmetric encryption is exposed in the file /juice-shop/lib/insecurity.ts at line 23.

Mitigation:

Avoid storing private keys in source code repositories.

Store sensitive keys securely outside the source code.

Use environment variables or secure key management solutions to handle private keys.

Recommendation:

Remove the private key from the source code and store it securely.

Rotate the exposed key and update any dependencies that might be using it.

NSWG-ECO-428: Security Misconfiguration in Build Process

Error Description:

The private key exposure is exacerbated by copying the entire /juice-shop directory during the build process ('COPY /juice-shop . # buildkit').

Mitigation:

Avoid copying unnecessary sensitive files or directories during the build process.

Minimize the inclusion of unnecessary files in the final image.

Recommendation:

Review the Dockerfile and limit the copied content to essential files only.

Ensure that the Docker image doesn't include sensitive information.

CVE-2015-9235: Use of Placeholder Public Key

Error Description:

A placeholder public key is used in case the file encryptionkeys/jwt.pub is not found.

Mitigation:

Use a proper error handling mechanism to alert developers/administrators if the public key file is missing.

Avoid using placeholder keys in production.

Recommendation:

Implement proper error handling to prevent the use of placeholder keys.

Ensure the existence and correctness of the public key file during deployment.

CVE-2023-4806: Insufficient Key Management

Error Description:

The key management process lacks sufficient security controls, leading to potential vulnerabilities.

Mitigation:

Implement secure key management practices.

Regularly audit and rotate keys.

Recommendation:

Enhance the key management process with industry best practices.

Regularly review and update key management procedures to address evolving threats.

CWE-201: Information Exposure Through Sent Data

Error Description:

The application exposes sensitive information through sent data, particularly the private key in /juice-shop/lib/insecurity.ts transmitted during build.

Mitigation:

Encrypt sensitive information during transmission.

Use secure channels such as HTTPS to protect data in transit.

Recommendation:

Ensure that sensitive information, especially private keys, is transmitted securely.

Implement secure communication protocols to safeguard information during transmission.

CWE-592: Authentication Bypass Issues

Error Description:

The application might have authentication bypass vulnerabilities, as indicated by the use of a placeholder public key in case of file absence.

Mitigation:

Implement strong authentication mechanisms.

Conduct thorough testing, including penetration testing, to identify and fix authentication bypass issues.

Recommendation:

Strengthen authentication processes to prevent bypass vulnerabilities.

Regularly assess the application for potential authentication flaws.

CWE-400: Uncontrolled Resource Consumption

Error Description:

The application may suffer from uncontrolled resource consumption due to inefficient key management processes.

Mitigation:

Implement resource usage controls.

Optimize resource-intensive processes.

Recommendation:

Address key management inefficiencies to control resource consumption.

Regularly monitor and optimize resource usage.

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

Error Description:

The application may be vulnerable to cross-site scripting (XSS) attacks, considering the use of flash messages without proper input validation.

Mitigation:

Implement input validation and output encoding.

Avoid using user input directly in HTML without proper sanitization.

Recommendation:

Review and sanitize user input before displaying it to prevent XSS vulnerabilities.

Educate developers on secure coding practices related to input validation.

CWE-23: Relative Path Traversal

Error Description:

The application might be susceptible to relative path traversal vulnerabilities due to the handling of file paths.

Mitigation:

Avoid user-controlled input in file paths.

Implement proper input validation and enforce strict file access controls.

Recommendation:

Restrict file path inputs and validate them rigorously to prevent path traversal attacks.

Review and update file access controls to minimize the risk of unauthorized access.

Project Requirements:

* Docker: Install Docker using the below command :

apt install docker.io -y

* Kubernetes: Install Kubernetes on the master node and then setup the Kubernetes cluster with two worker nodes.

Setup:

The application can be executed via docker as well as Kubernetes and let me guide you through both the technologies.

Using Docker

Pull the repository into the local machine using the command

git clone <https://github.com/juice-shop/juice-shop.git>

To Build the image locally go to the cloned repo and execute the command:

docker build -t joyboy/juiceshop .

Or else pull the existing image using the command:

* docker pull bkimminich/juice-shop

Then run the local image or the pulled image using the command:

* docker run --rm -p 3000:3000 <Image Name>

Using Kubernetes:

Create a deployment.yaml file using the below content :

---

kind: Deployment

apiVersion: apps/v1

metadata:

name: juice-shop

spec:

template:

metadata:

labels:

app: juice-shop

spec:

containers:

- name: juice-shop

image: bkimminich/juice-shop

selector:

matchLabels:

app: juice-shop

...

Then apply the above configuration file using the below:

* kubectl create -f juice-shop-deployment.yaml

Then see the deployment using the command:

* kubectl get deployment

---

kind: Service

apiVersion: v1

metadata:

name: juice-shop

spec:

type: NodePort

selector:

app: juice-shop

ports:

- name: http

port: 8000

targetPort: 3000

...

Apply the above config file using the command:

* kubectl create -f juice-shop-service.yaml

Very if the service is deployed using the command:

* kubectl get svc juice-shop

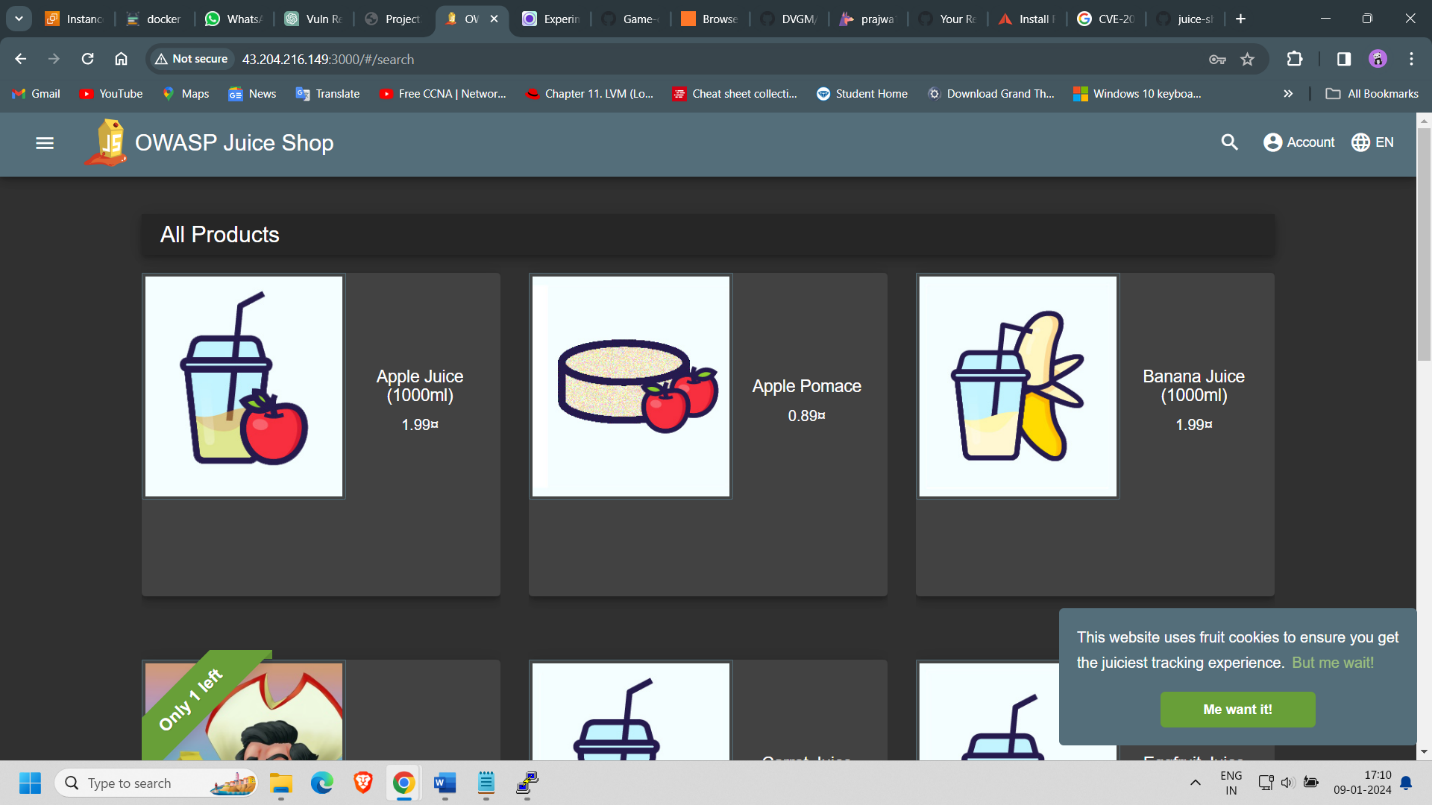
To see the endpoint where the application is deployed using the command:

* kubectl get ep juice-shop

When we type in the get-svc command we will be able to see the port to which the 3000 port has been mapped to copy that port and paste it along with public IP of the master node in the browser to see the output.

If the application is deployed using docker then copy the public IP of the machine the container is running in and use port 3000 to see the website.

The website looks like the below :



When the application is scanned using nuclei tool and OWASP ZAP using the below commands:

* docker run -t ghcr.io/zaproxy/zaproxy:stable zap-baseline.py -t <http://43.204.216.149:3000/#/>
* nuclei -u <http://43.204.216.149:3000/#/>

When the output of the issues is combined and analyzed below is the brief description about the project.

Report:

Exposed Server Information:

Vulnerability: Disclosure of server details in DNS records.

Risk Level: Low

Recommendation: Minimize server information exposure in DNS records for enhanced security.

Interaction with External Server:

Vulnerability: Interaction with an external server (oast.site).

Risk Level: Depends on the nature of interaction; further investigation needed.

Recommendation: Investigate the purpose of interaction and ensure it aligns with security policies.

Listener Detection and Juice Shop Recognition:

Vulnerability: Detection of specific listeners and recognition as the OWASP Juice Shop.

Risk Level: Low to Medium

Recommendation: Review listener configurations for unnecessary exposure and evaluate OWASP Juice Shop settings.

Web Fingerprinting:

Vulnerability: Web fingerprinting reveals the system as qm-system.

Risk Level: Low

Recommendation: Evaluate whether revealing the system type poses a security risk; consider obfuscation.

Recruitment Header Exposure:

Vulnerability: Exposure of recruitment-related headers.

Risk Level: Low

Recommendation: Review and potentially restrict headers related to recruitment to reduce unnecessary exposure.

Missing Security Headers:

Vulnerability: Several security headers are missing (Referrer Policy, Clear-Site-Data, Cross-Origin Resource Policy, Strict Transport Security, Content Security Policy, Permissions Policy, X-Permitted-Cross-Domain-Policies, Cross-Origin Embedder Policy, Cross-Origin Opener Policy).

Risk Level: High

Recommendation: Implement missing security headers to enhance web application security.

Robots.txt Endpoint:

Vulnerability: Presence of robots.txt.

Risk Level: Low

Recommendation: Ensure that robots.txt is appropriately configured to avoid information disclosure.

Security.txt Exposure:

Vulnerability: Exposure of security.txt with a contact email.

Risk Level: Low

Recommendation: Review and ensure the security.txt content aligns with security policies.

Missing Subresource Integrity (SRI):

Vulnerability: Lack of SRI for external scripts.

Risk Level: Medium

Recommendation: Implement SRI for external scripts to prevent unauthorized modifications.

Conclusion:

The web application has a mix of low to medium-severity vulnerabilities, with the absence of crucial security headers being the most critical issue.

Immediate attention should be given to implementing missing security headers to enhance the overall security posture.

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.