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

**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 diagram

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 a vulnerable Java application embodies a secure and well-configured environment, characterized by the absence of known security vulnerabilities and a robust implementation of security best practices. In this state, the application's configurations are diligently set to ensure secure session management, input validation, and effective error handling. Crucial security headers, such as Content Security Policy (CSP) and Strict-Transport-Security (HSTS), are appropriately configured to fortify the application against common web vulnerabilities.

Furthermore, the normal state necessitates the consistent application of up-to-date dependencies, frameworks, and libraries, mitigating risks associated with outdated software components. Regular updates and patches are crucial to address vulnerabilities and enhance the overall security posture of the Java application. Authentication mechanisms are implemented with a focus on strength, and proper authorization controls are in place to restrict access and actions based on user privileges.

The normal state also implies an absence of known exploits or vulnerabilities that could compromise the confidentiality, integrity, or availability of the application. The application's codebase adheres to secure coding practices, minimizing the likelihood of injection flaws, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and other common security pitfalls.

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 an 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

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

Description: The application fails to properly neutralize user-controllable input, making it susceptible to Cross-Site Scripting (XSS) attacks.

Mitigation: Implement input validation and use output encoding libraries to sanitize user input before rendering it in web pages. Employ Content Security Policy (CSP) headers to minimize the impact of XSS.

CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

Description: The application constructs OS commands using input from external sources without proper validation, exposing it to OS Command Injection vulnerabilities.

Mitigation: Utilize parameterized queries or prepared statements to prevent command injection. Validate and sanitize user input before using it in command construction. Employ application firewalls to detect and block malicious inputs.

CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

Description: The Docker file contains a vulnerability where input buffers are copied to output buffers without size verification, leading to classic buffer overflow issues.

Mitigation: Ensure all buffer operations have size checks to prevent overflow. Use safer functions for copying, such as strncpy or snprintf. Regularly update and patch software to fix known vulnerabilities.

CWE-787: Out-of-bounds Write

Description: The product writes data past the end or before the beginning of intended buffers, potentially causing out-of-bounds write vulnerabilities.

Mitigation: Implement bounds checking to ensure data is written within the allocated buffer size. Utilize secure coding practices and tools to identify and fix out-of-bounds write issues. Regularly update and patch software to address known vulnerabilities.

CWE-125: Out-of-bounds Read

Description: The product reads data past the end or before the beginning of intended buffers, potentially leading to out-of-bounds read vulnerabilities.

Mitigation: Implement bounds checking to ensure data is read within the allocated buffer size. Use secure coding practices and tools to identify and fix out-of-bounds read issues. Regularly update and patch software to address known vulnerabilities.

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.

Using Docker:

Clone the repository using the command:

* git clone <https://github.com/DataDog/vulnerable-java-application.git>

Then Create the image locally using the command execute the command within the repo that have been cloned:

* docker build -t joyboy/vulnerable-java-application .

Or we can pull the existing image using the command:

* docker pull ghcr.io/datadog/vulnerable-java-application

Then start the container using the command:

* docker run --rm -p 8000:8000 <Image name Local or the pulled one>

Using Kubernetes:

If you have a local image that you have built from the repo then we have to push it to the docker hub using the commands:

* docker login

Type in the UserName and the Authentication Token to successfully login to your docker hub

* docker tag local-image:tagname joyboy/vulnerable-java-application

Tag the image using the above command

* docker push joyboy/vulnerable-java-application

Use the above command to push the image

If you are using the docker image that is available then use the below command to run the application:

* kubectl run vulnerable-application --port=8000 --expose=true --image ghcr.io/datadog/vulnerable-java-application

Then use the command is used to create a secure tunnel between a local machine and a specified pod in a Kubernetes cluster :

* kubectl port-forward pod/vulnerable-application 8000

To find the vulnerabilities in the live application that has been created we us OWASP ZAP and nuclei

The combined report of both the tools are:   
Vulnerability Report:

Missing Anti-clickjacking Header:

Identified URL: http://13.232.48.232:8000/

Description: The application lacks the anti-clickjacking header (X-Frame-Options), making it susceptible to clickjacking attacks where an attacker could embed the site in a malicious frame.

Recommendation: Implement the X-Frame-Options header to prevent clickjacking.

X-Content-Type-Options Header Missing:

Identified URLs: http://13.232.48.232:8000/, http://13.232.48.232:8000/js/main.js

Description: The X-Content-Type-Options header is missing, which may expose the application to MIME-sniffing attacks.

Recommendation: Set the X-Content-Type-Options header to 'nosniff' to mitigate MIME-sniffing risks.

Content Security Policy (CSP) Header Not Set:

Identified URL: http://13.232.48.232:8000/

Description: The Content Security Policy (CSP) header is not configured, leaving the application more vulnerable to various code injection attacks.

Recommendation: Implement a strong and restrictive CSP header to protect against code injection.

Storable and Cacheable Content:

Identified URLs: http://13.232.48.232:8000/, http://13.232.48.232:8000/js/main.js, http://13.232.48.232:8000/robots.txt, http://13.232.48.232:8000/sitemap.xml

Description: Some content is storable and cacheable, potentially exposing sensitive information.

Recommendation: Adjust caching settings to prevent sensitive information from being stored or cached.

Permissions Policy Header Not Set:

Identified URLs: http://13.232.48.232:8000/, http://13.232.48.232:8000/js/main.js

Description: The Permissions Policy header is not set, leaving the application more susceptible to security vulnerabilities.

Recommendation: Configure and set a strong Permissions Policy header to control and limit browser features.

Modern Web Application:

Identified URL: http://13.232.48.232:8000/

Description: The application is identified as a modern web application.

Recommendation: Stay updated with modern security best practices and ensure the application follows secure coding standards.

Absence of Anti-CSRF Tokens:

Identified URL: http://13.232.48.232:8000/

Description: The application lacks anti-CSRF tokens, making it vulnerable to CSRF attacks.

Recommendation: Implement anti-CSRF tokens to protect against Cross-Site Request Forgery attacks.

Potential Risks:

Clickjacking: Without the anti-clickjacking header, attackers might trick users into performing unintended actions by embedding the application in malicious frames.

MIME-sniffing Attacks: Absence of the X-Content-Type-Options header may expose the application to MIME-sniffing attacks.

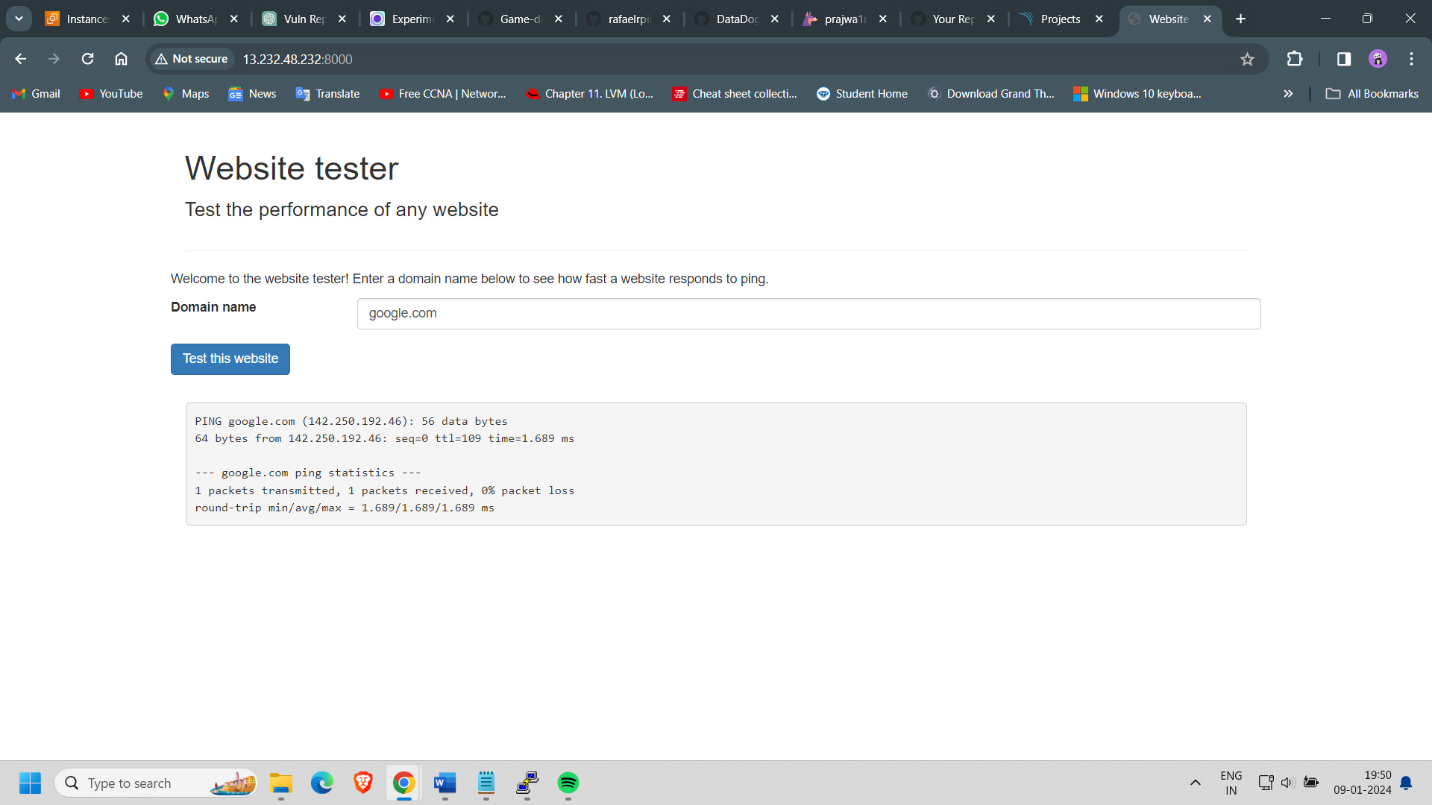
Code Injection: Lack of a Content Security Policy (CSP) header increases the risk of code injection attacks.

Caching Sensitivity: Storable and cacheable content might expose sensitive information if improperly stored or cached.

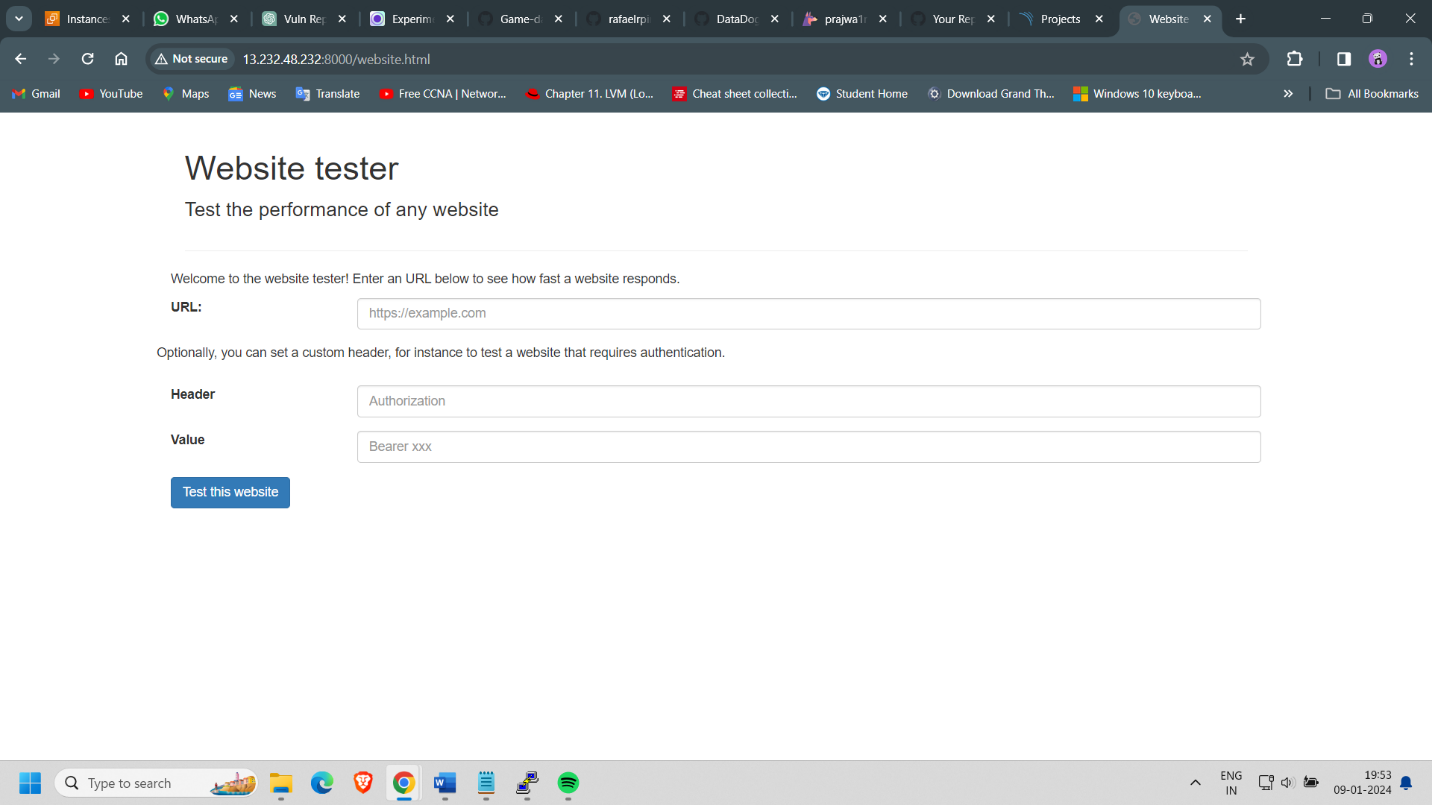
Permissions Policy Control: Absence of the Permissions Policy header may lead to inadequate control over browser features, increasing the risk of security vulnerabilities.

CSRF Attacks: The absence of anti-CSRF tokens exposes the application to Cross-Site Request Forgery attacks.   
  
Then type in the public IP of the machine with port 8000 and you will be able to see the application and if the application is deployed using Kubernetes find the port and link it to the public IP to view the page.

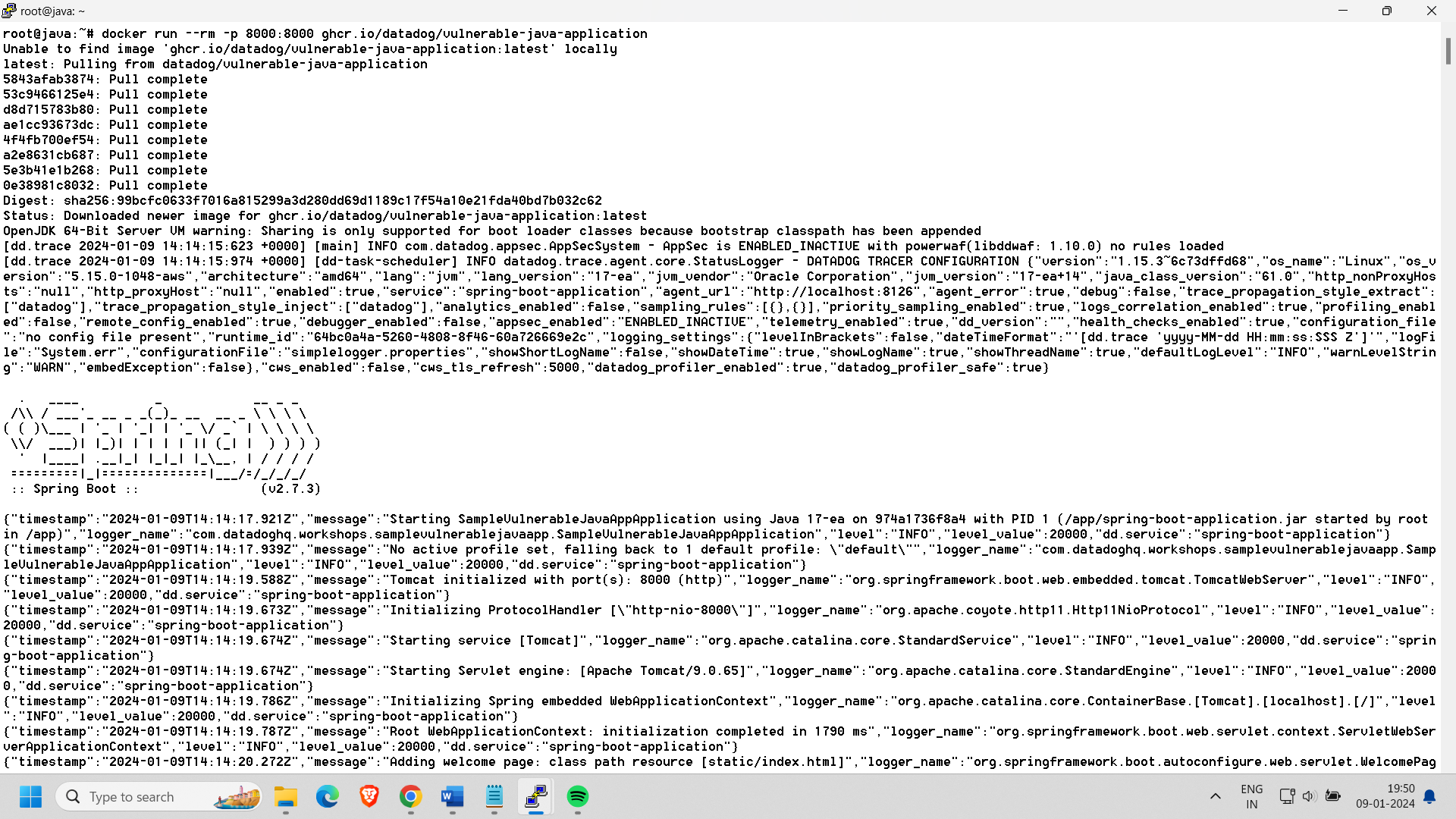
The Image of the page looks like below :



The website has two pages one looks like the above which is a default page and below is another one that can be viewed using the publicip:port/website.html which looks like the below image



If we run the application using the docker the result looks like:



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.