|  |  |
| --- | --- |
| Name: | Mazhar Hussain C K |
| Lab User ID: | 23SEK3324\_U06 |
| 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)

EC2 Instance

Docker Engine

Java Web App Container

Access through

Ip and port number

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)

in its typical state, a vulnerable Java web application might exhibit security gaps and configuration issues that could compromise its integrity. This state might include known security vulnerabilities and less-than-optimal implementation of security best practices. Configuration settings might lack robustness, leading to potential lapses in session management, input validation, and error handling. Critical security headers like Content Security Policy (CSP) and Strict-Transport-Security (HSTS) might be inadequately configured, leaving the application exposed to common web vulnerabilities.

Moreover, maintaining outdated dependencies, frameworks, and libraries could heighten risks associated with using components that lack essential security updates. The absence of regular updates and patches might leave the application vulnerable to exploitation, undermining its overall security posture. Authentication mechanisms might be less stringent, and authorization controls may not be as effective in restricting access and actions based on user privileges.

This state may contain known exploits or vulnerabilities, potentially posing threats to the confidentiality, integrity, or availability of the application. The codebase might lack adherence to secure coding practices, potentially increasing the risk of injection flaws, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and other common security vulnerabilities.

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 )

The project is done on a aws ec2 machine running on ubuntu,after successful installation of docker we implement a

Code that has been provided in the readme file

# gets the code

git clone https://github.com/rafaelrpinto/VulnerableJavaWebApplication

cd VulnerableJavaWebApplication

# creates the docker image

docker build -t vulnerable-java-application:0.1 .

# creates/starts the container

docker run --name vulnerable-java-application -p 9000:9000 -d vulnerable-java-application:0.1

This will clone the repository and then create a docker image from the repository dockerfile which is then used for creating the container.

After successful deployment we can access the web app through ip and portnumber 9000,remember to specify https:// as this runs in https

A screenshot of a computer

Description automatically generated

Now we have to do vulnerability scanning on this here we have used snyk for repository scanning and ZAP baseline scanning for scanning the live web app one is a static code testing while the other one is a dynamic code testing

**Scanning using snyk**

A screenshot of a computer

Description automatically generated

There are many number of errors in the repository mainly in pom file.Some of them are

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

**Description:** The application is vulnerable to XSS attacks due to inadequate neutralization of user-controlled input during web page generation. **Mitigation:** Implement input validation and employ output encoding libraries to sanitize user input before displaying it on web pages. Additionally, utilize Content Security Policy (CSP) headers to minimize the impact of XSS vulnerabilities.

**CWE-78: OS Command Injection**

**Description:** The application is at risk of OS Command Injection due to constructing OS commands without proper validation from external input. **Mitigation:** Mitigate the risk by using parameterized queries or prepared statements to prevent command injection. Validate and sanitize user input before incorporating it into command construction. Employ application firewalls to detect and block malicious inputs.

**CWE-120: 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:** Prevent buffer overflow by ensuring all buffer operations include size checks. Use safer functions like strncpy or snprintf for copying. Regularly update and patch software to address known vulnerabilities.

**CWE-787: Out-of-bounds Write**

**Description:** The product writes data beyond intended buffer limits, potentially causing out-of-bounds write vulnerabilities. **Mitigation:** Implement bounds checking to confine data within allocated buffer sizes. Use secure coding practices and tools to detect and resolve out-of-bounds write issues. Regularly update and patch software to tackle known vulnerabilities.

**CWE-125: Out-of-bounds Read**

**Description:** The product reads data outside intended buffer boundaries, potentially leading to out-of-bounds read vulnerabilities. **Mitigation:** Ensure bounds checking to restrict data reading within allocated buffer sizes. Adopt secure coding practices and tools to identify and fix out-of-bounds read issues. Regularly update and patch software to address known vulnerabilities.

**Using ZAP Baseline Scan**

Using baseline scan we found many vulnerabilities

A screenshot of a computer program

Description automatically generated

A computer screen shot of a computer code

Description automatically generated

**X-Content-Type-Options Header Missing**

**Description:** The absence of the X-Content-Type-Options header exposes the application to MIME-sniffing attacks, potentially compromising content interpretation.

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

**Content Security Policy (CSP) Header Not Set**

**Description:** The application lacks a configured Content Security Policy (CSP) header, leaving it more vulnerable to diverse code injection attacks.

**Recommendation:** Strengthen the application's security posture by implementing a robust and restrictive CSP header to counter code injection threats effectively.

**Missing Anti-Clickjacking Header**

**Description:** The application lacks the necessary anti-clickjacking header (X-Frame-Options), making it vulnerable to clickjacking attacks.

**Recommendation:** Implement the X-Frame-Options header to prevent clickjacking and secure user interactions effectively.

**Storable and Cacheable Content**

**Description:** Some content within the application is storable and cacheable, potentially exposing sensitive information.

**Recommendation:** Adjust the caching settings to prevent the storage or caching of sensitive data, bolstering overall security measures and safeguarding user privacy.