Chaos Engineering - Report

Name:	Sunidhi Agrawal
Lab User ID:	23SEK3324_U11
Date:	09.01.2024
Application Name:	OWASP Juice Shop

Follow the below guidelines:

Define Hypotheses and Scenarios:

- Similar to Chaos Engineering, start by defining hypotheses and scenarios related to potential threats.

Inject Controlled Failure:

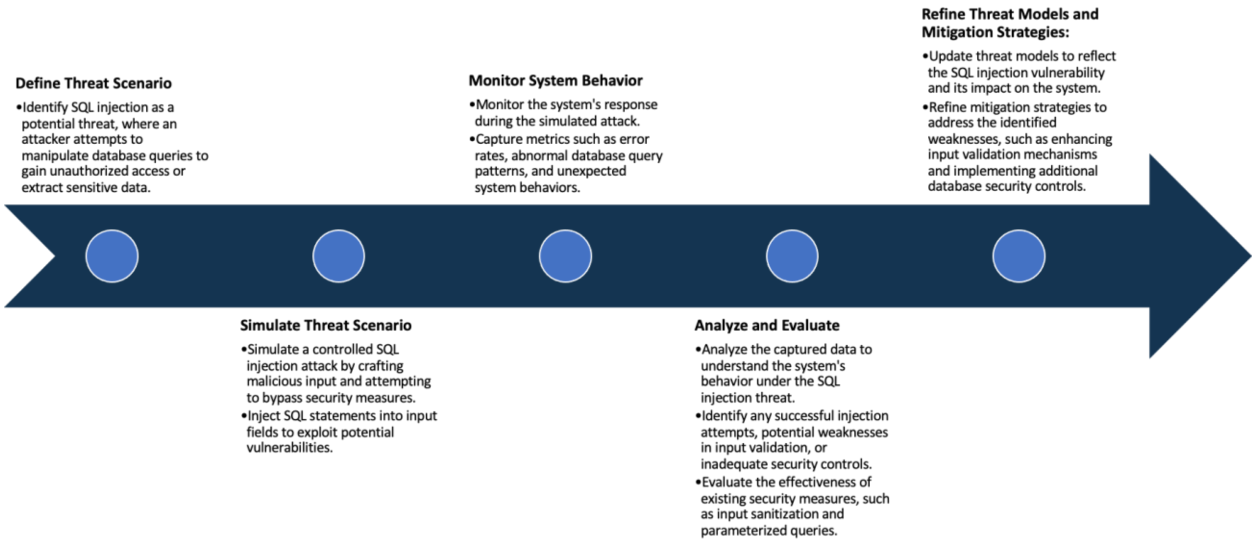
- Introduce controlled failure scenarios that mimic potential attack vectors or vulnerabilities identified in Threat Modeling.
- Simulate failure conditions, such as network disruptions, component failures, or data breaches, to observe the system's response.

Measure System Behavior:

- Capture and measure relevant system behavior metrics during the chaos experiments.
- Monitor the system's response to the injected failures, including performance metrics, error rates, and security-related indicators.
- Analyze and compare the observed behavior against the expected outcomes defined in the Threat Modeling process.

Learn and Iterate:

- Learn from the results of the chaos experiments and iterate on the Threat Modeling process.
- 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.



Define Threat Scenario

- Identify SQL injection as a potential threat, where an attacker attempts to manipulate database queries to gain unauthorized access or extract sensitive data.

Simulate Threat Scenario

- Simulate a controlled SQL injection attack by crafting malicious input and attempting to bypass security measures.
- Inject SQL statements into input fields to exploit potential vulnerabilities.

Monitor System Behavior

- Monitor the system's response during the simulated attack.
- Capture metrics such as error rates, abnormal database query patterns, and unexpected system behaviors.

Analyze and Evaluate

- Analyze the captured data to understand the system's behavior under the SQL injection threat.
- Identify any successful injection attempts, potential weaknesses in input validation, or inadequate security controls.
- Evaluate the effectiveness of existing security measures, such as input sanitization and parameterized queries.

Refine Threat Models and Mitigation Strategies:

- Update threat models to reflect the SQL injection vulnerability and its impact on the system.
- Refine mitigation strategies to address the identified weaknesses, such as enhancing input validation mechanisms and implementing additional database security controls.

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)

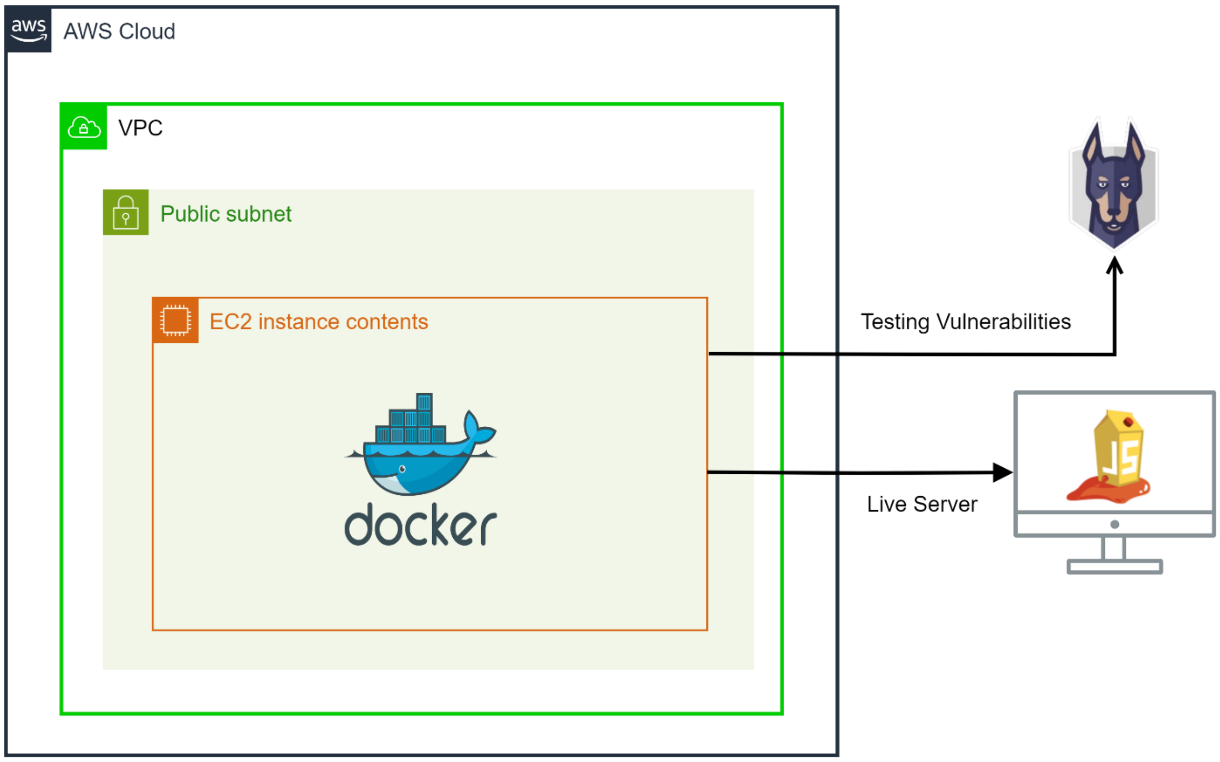
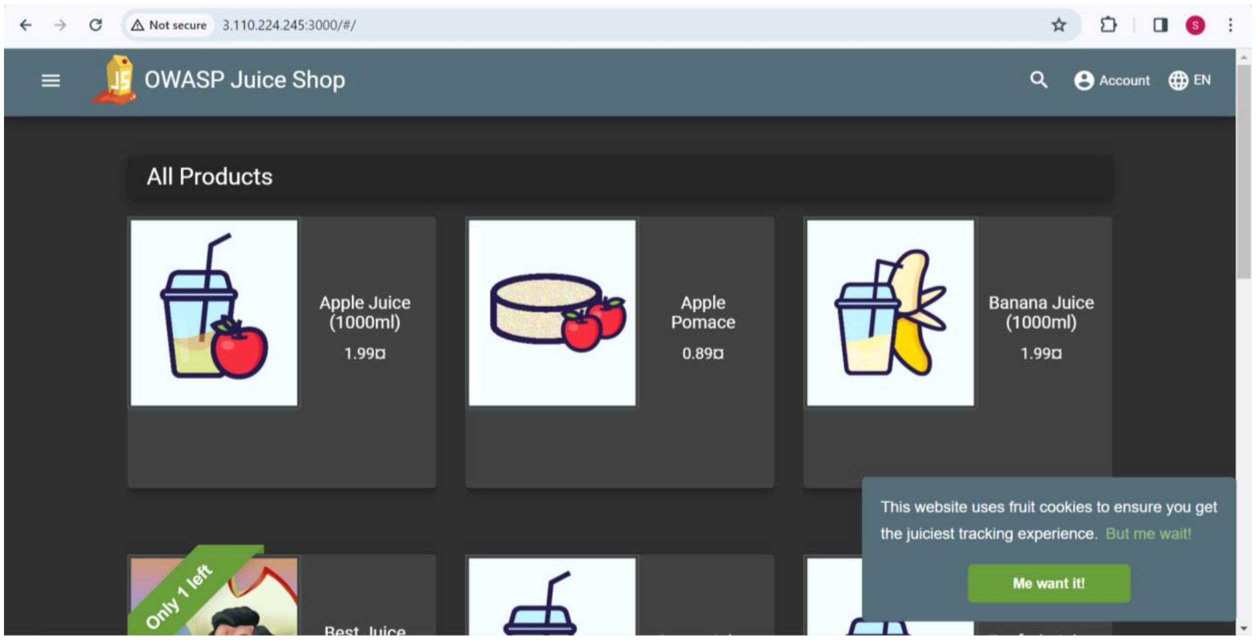


Figure 1:System Architecture



This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wait!

Figure 2:Live OWASP Juice Shop

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 your OWASP Juice Shop deployment, defining the steady state involves identifying normal behavior and establishing measurable outputs that indicate the system's health and proper functioning. Here are some key components to consider:

- Response Time:

- Normal Behavior: Requests are processed within an expected time frame, considering the application's complexity.
- Measurable Output: Track and analyze response times for different endpoints. Set thresholds and detect deviations from the baseline.

- Network Traffic:

- Normal Behavior: Consistent and expected levels of incoming and outgoing network traffic.
- Measurable Output: Use network monitoring tools to observe traffic patterns. Anomalies such as unexpected spikes or drops should be investigated.

- Database Query Performance:

- Normal Behavior: Database queries are executed efficiently without significant delays.
- Measurable Output: Monitor and analyze database query performance. Identify and address slow queries that could impact overall system performance.

- Authentication and Authorization Logs:

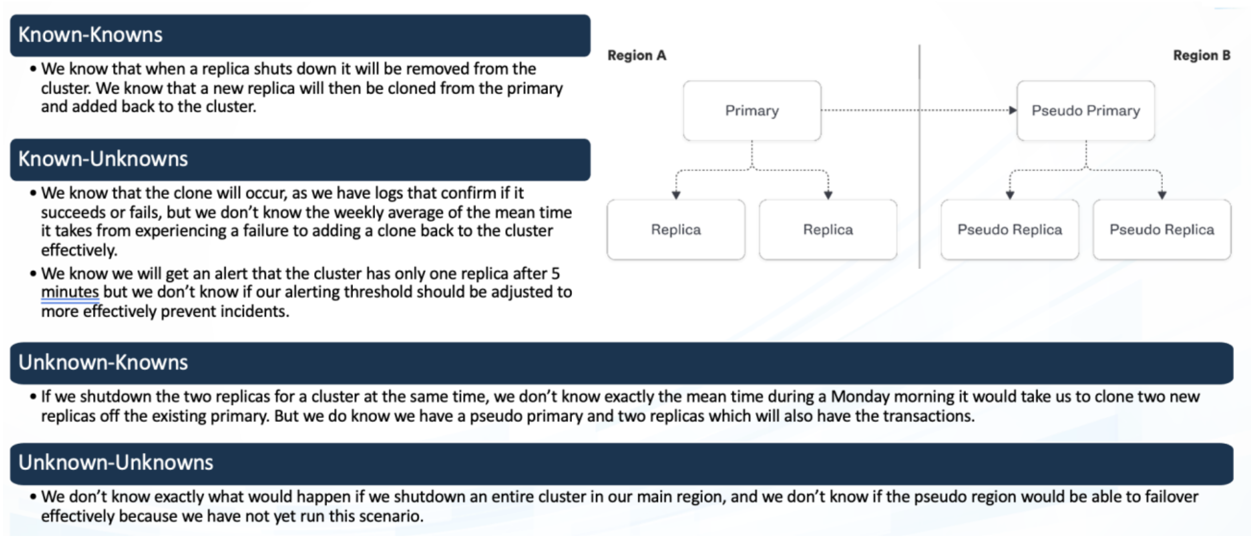
- Normal Behavior: Successful and authorized login attempts by legitimate users.
- Measurable Output: Regularly review authentication and authorization logs. Identify any unusual login patterns, multiple failed attempts, or unauthorized access.

- System Resource Utilization:

- Normal Behavior: Stable and balanced utilization of CPU, memory, and disk resources.
- Measurable Output: Use system monitoring tools to track resource usage. Set thresholds and detect any resource-intensive activities.

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	- **Known vulnerabilities:** OWASP Juice Shop is deliberately insecure and contains known vulnerabilities to help users practice identifying and exploiting security issues. - **Intended weaknesses:** The application intentionally includes vulnerabilities like SQL injection, cross-site scripting (XSS), security misconfigurations, and more.	- **Identified vulnerabilities with unknown solutions:** While you may know that certain vulnerabilities exist in OWASP Juice Shop, you might not be aware of all the possible ways to mitigate or fix them.
Unknown	- **Undiscovered vulnerabilities:** There may be additional security issues in OWASP Juice Shop that are not widely known or documented. Regular security assessments and updates from the OWASP community can help address these.	- **Emerging threats**: As with any software, there might be new, unforeseen vulnerabilities or attack vectors that have not been discovered or documented yet. Staying informed about the latest security trends and updates is crucial to mitigating these unknown risks.

When working with OWASP Juice Shop or any intentionally vulnerable application, it's essential to consider both the known vulnerabilities (to practice and learn) and the potential for unknown issues that may arise during deployment or usage. Regularly updating the application and keeping an eye on security communities can help address emerging threats and ensure a more secure environment.

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.

The objective of implementing chaos engineering in OWASP Juice Shop is to evaluate the application's resilience under controlled chaotic conditions, identify potential vulnerabilities, and enhance overall security. By introducing deliberate chaos, we aim to simulate real-world scenarios and proactively address any weaknesses in the application.

Hypotheses:

- Injection Vulnerabilities Resilience:

- Assumption: OWASP Juice Shop effectively mitigates injection vulnerabilities.
- Prediction: Introducing simulated injection attacks will reveal the application's ability to resist unauthorized database access.

- Cross-Site Scripting (XSS) Handling:

- Assumption: OWASP Juice Shop can prevent XSS attacks.
- Prediction: Simulating XSS attacks will assess the application's robustness in blocking malicious scripts and protecting user interactions.

Experiment:

(Document your Preparation, Implementation, Observation and Analysis)

Project Overview:
The primary objective of this project is to deploy OWASP Juice Shop using Docker and conduct a vulnerability assessment. The goal is to familiarize oneself with the application's intentionally insecure nature, understand its vulnerabilities, and practice identifying and exploiting security issues. Through this process, we aim to enhance our knowledge of web application security and improve our ability to recognize and mitigate common vulnerabilities.

Tools Used:
Docker:

- Purpose: Docker is utilized for containerization, providing a consistent and isolated environment for running OWASP Juice Shop.
- Explanation: Docker enables the creation of isolated containers, ensuring consistency across different environments.

Snyk:

- Purpose: Snyk is used for automated vulnerability assessment, focusing on identifying and reporting vulnerabilities in the dependencies of OWASP Juice Shop.
- Explanation: Snyk is integrated into the project to scan and analyze the application's dependencies for security vulnerabilities.

Web Browser:

- Purpose: web browser is used to interact with the OWASP Juice Shop web application and assess its vulnerabilities.
- Explanation: Browsing to http://localhost:3000 allows access to the Juice Shop interface.

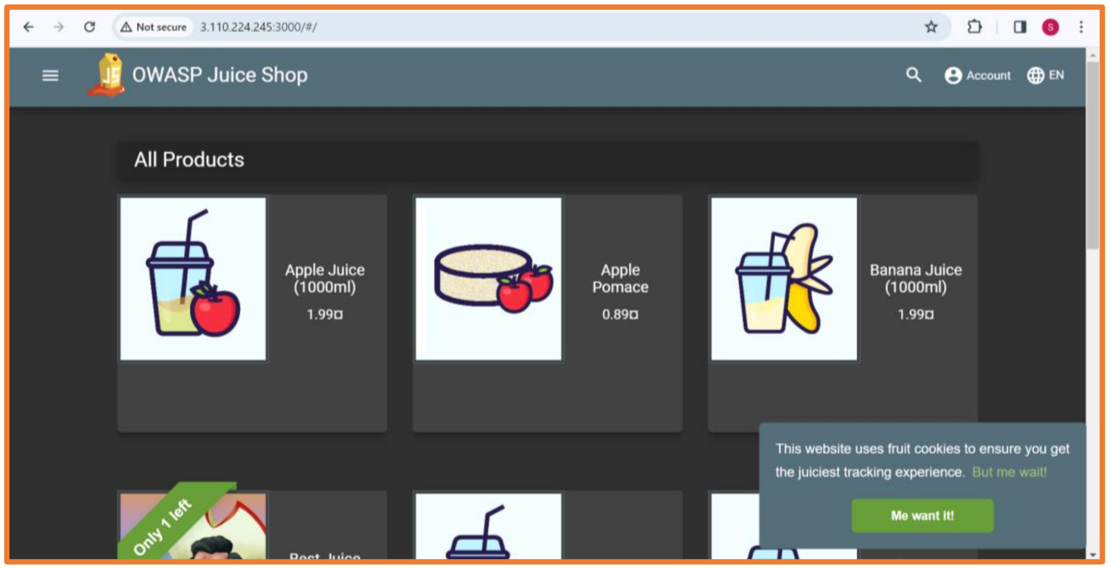
Implementation:

1. **Docker Installation:**

- **Command:**
  - Execute the '**docker pull bkimminich/juice-shop**' command to fetch the OWASP Juice Shop Docker image.
  - Run the '**docker run --rm -p 3000:3000 bkimminich/juice-shop**' command to launch the Juice Shop container, mapping port 3000 on the host to port 3000 in the container.

2. **Accessing OWASP Juice Shop:**

- Open a web browser and navigate to http://localhost:3000.
- The Juice Shop interface should be accessible, providing an interactive environment to explore and test various security vulnerabilities.

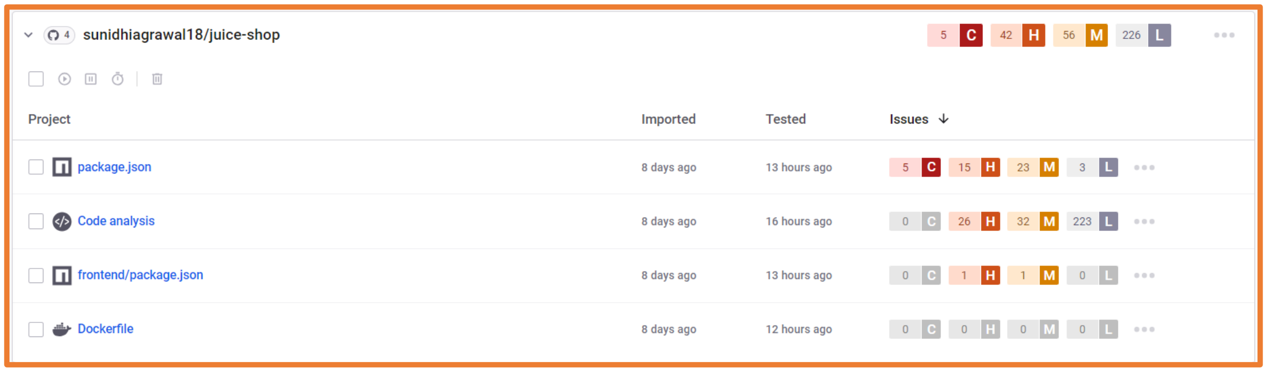


This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wait!

Figure 3: OWASP Juice Shop

3. **Vulnerability Assessment with Snyk:**

- Integrate Snyk into the project to scan the OWASP Juice Shop dependencies for vulnerabilities.
- Analyze the Snyk report to identify and understand the severity of any vulnerabilities detected.



sunidhiagrawal18/juice-shop — 5 C 42 H 56 M 226 L

Project	Imported	Tested	Issues
package.json	8 days ago	13 hours ago	5 C 15 H 23 M 3 L
Code analysis	8 days ago	16 hours ago	0 C 26 H 32 M 223 L
frontend/package.json	8 days ago	13 hours ago	0 C 1 H 1 M 0 L
Dockerfile	8 days ago	12 hours ago	0 C 0 H 0 M 0 L

Figure 4: Vulnerability Assessment by Snyk

Observation and Analysis:

Vulnerabilities Identified in the Project:

- vm2 Remote Code Execution (RCE) - CVE-2023-37903:

- **Description**: In versions up to and including 3.9.19 of the vm2 module, there is a vulnerability where the Node.js custom inspect function can be exploited by attackers to escape the sandbox and execute arbitrary code.

- ansi-regex Regular Expression Denial of Service (ReDoS) - CVE-2021-3807

- **Description:** The ansi-regex module is susceptible to an Inefficient Regular Expression Complexity issue, potentially leading to Regular Expression Denial of Service (ReDoS) attacks.

- jsonwebtoken Authentication Bypass - CVE-2015-9235

- **Description:** In versions of the jsonwebtoken module before 4.2.2, there is a vulnerability that allows attackers to bypass token verification. An attacker could send a token digitally signed with a symmetric algorithm instead of the expected asymmetric key, leading to an authentication bypass.

- lodash Prototype Pollution - CVE-2019-10744

- **Description** Versions of lodash lower than 4.17.12 are vulnerable to Prototype Pollution. The function defaultsDeep could be manipulated to add or modify properties of Object.prototype using a constructor payload.

- engine.io Denial of Service (DoS) - CVE-2022-41940

- **Description:** Engine.IO, the implementation of a transport-based bi-directional communication layer for Socket.IO, is vulnerable to a Denial of Service (DoS) attack. A specially crafted HTTP request can trigger an uncaught exception on the Engine.IO server, potentially causing the Node.js process to crash.

Conclusion:

The deployment of OWASP Juice Shop through Docker, coupled with vulnerability assessments with Snyk, enhances our understanding of web application security. Snyk's automated scanning provides an efficient means to identify and prioritize vulnerabilities, allowing for a more comprehensive and proactive approach to securing the application. This hands-on experience contributes to continuous improvement in security awareness and knowledge, reinforcing good security practices in web application development and maintenance.