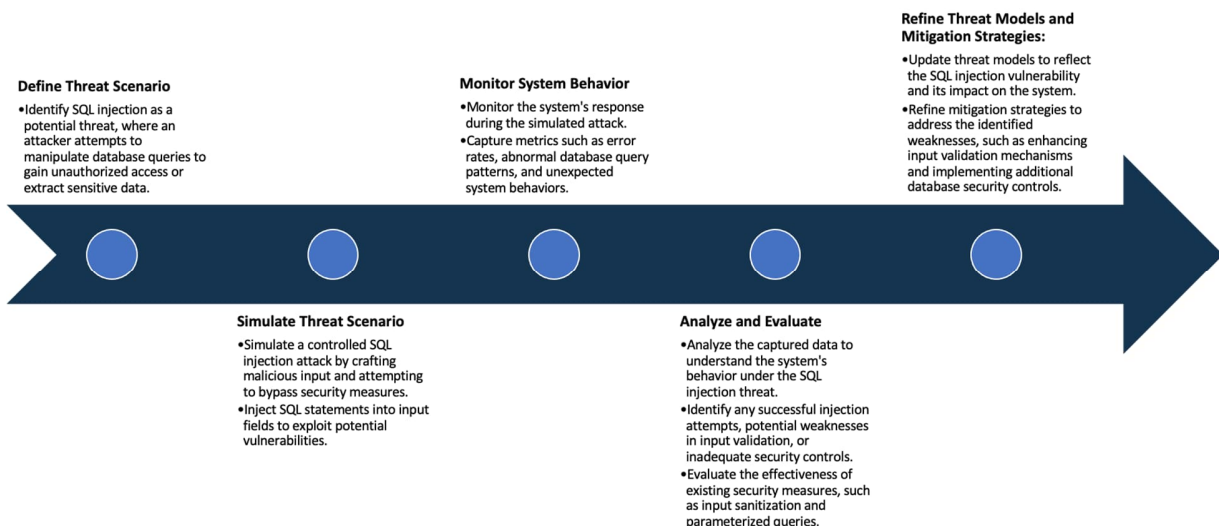


# Chaos Engineering – Hypothesis

Name:	Gowrav Kumar Baitharu Aripaka
Lab User ID:	23SEK3324_U02
Date:	11-01-2024
Application Name:	Juice Shop

Follow the below guidelines:

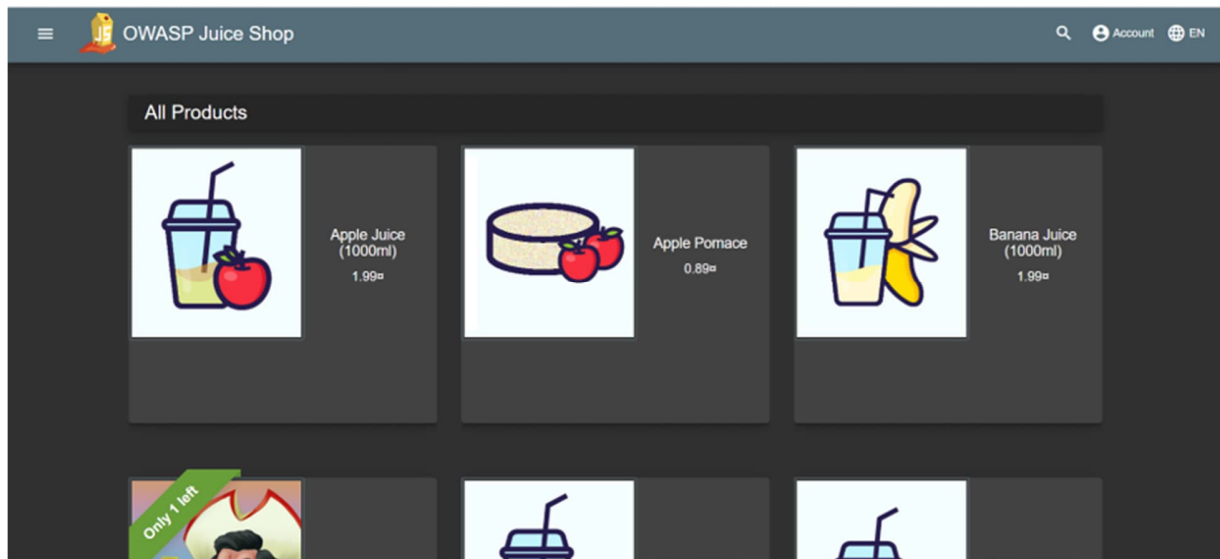
Define Hypotheses and Scenarios:	Inject Controlled Failure:	Measure System Behavior:	Learn and Iterate:
<ul style="list-style-type: none"> <li>Similar to Chaos Engineering, start by defining hypotheses and scenarios related to potential threats.</li> </ul>	<ul style="list-style-type: none"> <li>Introduce controlled failure scenarios that mimic potential attack vectors or vulnerabilities identified in Threat Modeling.</li> <li>Simulate failure conditions, such as network disruptions, component failures, or data breaches, to observe the system's response.</li> </ul>	<ul style="list-style-type: none"> <li>Capture and measure relevant system behavior metrics during the chaos experiments.</li> <li>Monitor the system's response to the injected failures, including performance metrics, error rates, and security-related indicators.</li> <li>Analyze and compare the observed behavior against the expected outcomes defined in the Threat Modeling process.</li> </ul>	<ul style="list-style-type: none"> <li>Learn from the results of the chaos experiments and iterate on the Threat Modeling process.</li> <li>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.</li> </ul>



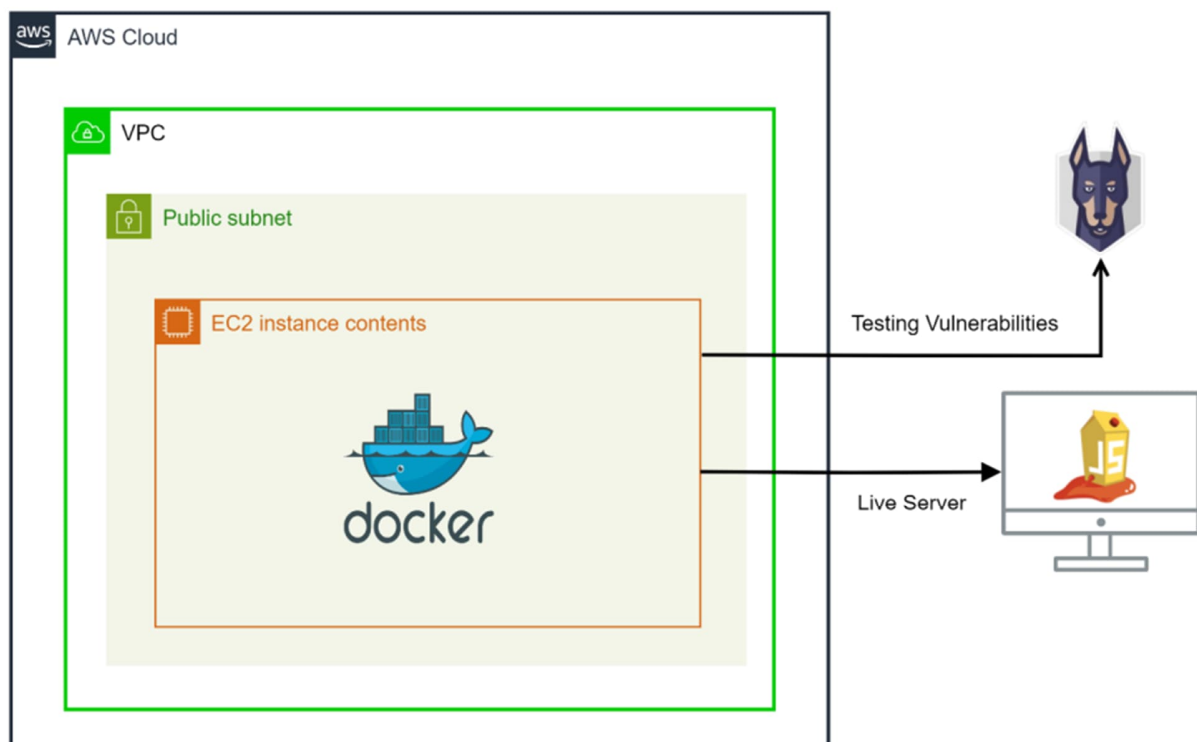
# Chaos Engineering – Hypothesis

## 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)



1. Live Website



2. System Architecture

# Chaos Engineering – Hypothesis

## 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)

### Kubernetes Cluster overview:

Cluster Configuration: 1 Master node, 2 Worker node, Master Node responsible for the control plane and managing the cluster state.

---

**Application Deployment:** OWASP Juice Shop, deployed using a Kubernetes Deployment with 2 replicas.

---

Pods are distributed across two worker nodes.

Pod 1: Worker Node 1 (IP:15.23.331.123)

Pod 2: Worker Node 2 (IP:3.103.254.22)

---

**Service Port:** Service is configured to open port 30371.

---

**Pod Status:** Both Pods are running and healthy. Application instances are distributed across the worker nodes.

---

**Connectivity:** Pods on different worker nodes ensure high availability and fault tolerance.

---

**Access URL:** The Juice Shop application can be accessed externally using the worker nodes IP addresses and the specified port.

---

**Example for Worker Node 1:** `http:// 15.23.331.123:30371`

---

**Example for Worker Node 2:** `http:// 3.103.254.22:30371`

---

**Browser Connectivity:** Users can access the Juice Shop application via a web browser.

---

### Juice Shop workings:

---

**Product Catalog:** Users can browse through a variety of products listed on the website,

---

**User Authentication:** Users can create accounts, log in, and perform actions that involve user authentication.

---

Users can browse through a catalog of juices, view their details, and add them to a virtual cart.

---

Users can create accounts, log in, and manage their profiles, including reviewing products on the website

---

**The application having several pages like contact, company, and feedback.**

---

**There are several APIs available for programmatic access to juice shop data and functionality.**

---

**Administrators can manage users, products, orders, and website settings.**

# Chaos Engineering – Hypothesis

## Hypothesis:

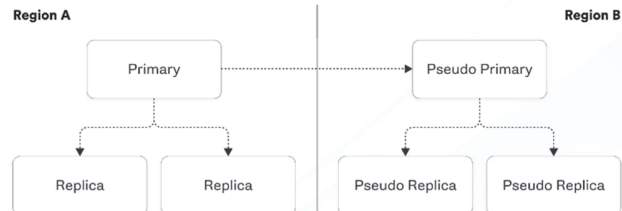
(During an experiment, we need a hypothesis for comparing to a stable control group, and the same applies here too. If there is a reasonable expectation for a particular action according to which we will change the steady state of a system, then the first thing to do is to fix the system so that we accommodate 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	<p>You know a specific component can fail (e.g., database server overload). You also know the expected behavior (e.g., service degradation, automatic failover).</p>	<p>The system comprehensively understands resource limits like CPU, memory, and storage under regular circumstances. However, unforeseen challenges may arise if these limits are unexpectedly surpassed, potentially leading to performance issues or system disruptions.</p>
Unknown	<p>You know a general category of potential issues (e.g., network fluctuations). However, you don't know the specific triggers or consequences</p>	<p>Shutting down the entire Kubernetes cluster, including the main control hub, is uncertain, and we're not sure what might happen.</p>
	Known	Unknown

# Chaos Engineering – Hypothesis

## Experiment:

(Document your Preparation, Implementation, Observation and Analysis )

---

**Using some tools for security analysis on this Juice Shop application.**

---

1. OWASP ZAP

---

2. SNYK

---

3. Gremlin

## Observation:

### OWASP ZAP:

---

```
root@master:~# docker run -t ghcr.io/zaproxy/zaproxy:stable zap-baseline.py -t http://15.207.221.39:30371
```

---

By using this command, we are performing zap base line command on our live application.

---

Analysis and result:

---

```
http://15.207.221.39:30371/vendor.js (200 OK)
FAIL-NEW: 0 FAIL-INPROG: 0 WARN-NEW: 9 WARN-INPROG: 0 INFO: 0 IGNORE: 0 PASS: 56
```

---

Out of 94 tests it has passed only 56 tests and 9 are new. It has ignored 0 tests.

---

By performing this ZAP test, I have got several security issues.

---

[Cross-Domain JavaScript Source File Inclusion \(Issue ID: 10017\)](#) is a security vulnerability where an attacker can include JavaScript files from an external domain, compromising the integrity of a web application. This allows the attacker to execute malicious scripts within the victim's browser, potentially leading to unauthorized access, data theft, or other security breaches. Implementing proper input validation and secure coding practices can help mitigate this risk.

---



---

**Solution:** Mitigate Cross-Domain JavaScript Source File Inclusion by validating and restricting the inclusion of JavaScript files to trusted domains. Implement proper input validation and use Content Security Policy (CSP) headers to define approved sources for scripts, reducing the risk of malicious file inclusions.

---

## Chaos Engineering – Hypothesis

---

**Information Disclosure - Suspicious Comments (Issue ID: 10027)** refers to a security vulnerability where potentially sensitive information is exposed through suspicious comments in the source code of a software application. These comments may unintentionally reveal details about the system, passwords, or other critical information, posing a risk of unauthorized access or exploitation. Developers should conduct thorough code reviews and remove or secure any suspicious or unnecessary comments to prevent information disclosure.

---

**Solution:** Mitigate Information Disclosure - Suspicious Comments (Issue ID: 10027) by conducting regular code audits and removing unnecessary or sensitive comments. Implement strict commenting guidelines to avoid inadvertently disclosing confidential information.

---

---

**Content Security Policy (CSP) Header Not Set (Issue ID: 10038)** indicates a security vulnerability where web pages lack proper CSP headers, allowing for increased risk of cross-site scripting (XSS) attacks, repeated 11 times.

---

**Solution:** To mitigate Content Security Policy (CSP) Header Not Set (Issue ID: 10038), implement and enforce a strict CSP header in web servers. Specify trusted sources for scripts, styles, and other resources to prevent XSS attacks. Regularly review and update the CSP policy as needed.





---

---

SNYK: Performing SNYK

---

## Chaos Engineering – Hypothesis

Project	Imported	Tested	Issues ↓
<input type="checkbox"/>  package.json	13 minutes ago	13 minutes ago	5 <b>C</b> 15 <b>H</b> 23 <b>M</b> 3 <b>L</b> ...
<input type="checkbox"/>  Code analysis	11 minutes ago	11 minutes ago	0 <b>C</b> 26 <b>H</b> 32 <b>M</b> 223 <b>L</b> ...
<input type="checkbox"/>  frontend/package.json	13 minutes ago	13 minutes ago	0 <b>C</b> 1 <b>H</b> 1 <b>M</b> 0 <b>L</b> ...
<input type="checkbox"/>  Dockerfile	13 minutes ago	13 minutes ago	0 <b>C</b> 0 <b>H</b> 0 <b>M</b> 0 <b>L</b> ...

---

After performing SNYK there are some critical, high, medium, and low vulnerabilities.

---

### Uninitialized Memory Exposure:

---

Uninitialized Memory Exposure is a security vulnerability where a program or application inadvertently accesses and exposes data from uninitialized memory. This occurs when software fails to initialize or clear allocated memory before use, leading to the disclosure of sensitive information. Attackers can exploit this flaw to retrieve uninitialized data, potentially revealing passwords, cryptographic keys, or other confidential information. To address this vulnerability, developers should ensure proper initialization of all memory, conduct thorough code reviews, and implement static analysis tools to identify and rectify potential risks.

---

**Solution:** Mitigate Uninitialized Memory Exposure by adopting secure coding practices. Initialize all variables and memory locations before use to prevent unintended data exposure. Conduct thorough code reviews. Utilize memory-safe programming languages that automatically handle memory initialization, reducing the likelihood of this security flaw. Regularly update and patch software to address any identified issues and enhance overall system security.

---

### Vulnerability: Regular Expression Denial of Service (ReDoS)

---

Improper control of a limited resource allocation allows an actor to influence resource consumption, potentially leading to resource exhaustion.

## Chaos Engineering – Hypothesis

---

Regular Expression Denial of Service (ReDoS) is a security vulnerability where an attacker exploits inefficient regular expressions to cause a system to excessively backtrack while parsing input. This can lead to a denial of service as the system becomes overwhelmed, taking an excessive amount of time to process the malicious input, resulting in performance degradation or unresponsiveness.

---

**Solution:** Implement specific scale limits for protocols.

---

Ensure all resource allocation failures put the system in a safe state to prevent exhaustion.

---

Mitigate ReDoS vulnerabilities by optimizing regular expressions for efficiency, avoiding nested quantifiers and excessive backtracking. Use tools like regex analyzers to identify potential bottlenecks. Employ input validation to restrict user input complexity, limiting the impact of malicious patterns. Regularly update software dependencies to benefit from security patches and improvements in regular expression processing.

---

---

### **Vulnerability: Arbitrary Code Execution**

---

Arbitrary Code Execution is a critical security flaw allowing attackers to execute unauthorized commands on a system. Exploiting vulnerabilities in software, attackers inject and run malicious code, potentially gaining control over the targeted system. This poses significant risks, such as unauthorized access, data breaches, or system compromise. Preventive measures include robust input validation, code reviews, and regularly updating software to patch known vulnerabilities, enhancing overall system security against arbitrary code execution attacks.

---

**Solution:** mitigate Arbitrary Code Execution, employ secure coding practices such as input validation and sanitization. Implement least privilege principles to restrict system access for processes and users. Regularly update and patch software to address known vulnerabilities. Utilize web application firewalls and intrusion detection systems to monitor and block malicious activities. Employ application-level controls, like sandboxing, to contain and limit the impact of any successful exploitation, enhancing overall system resilience.

---

---

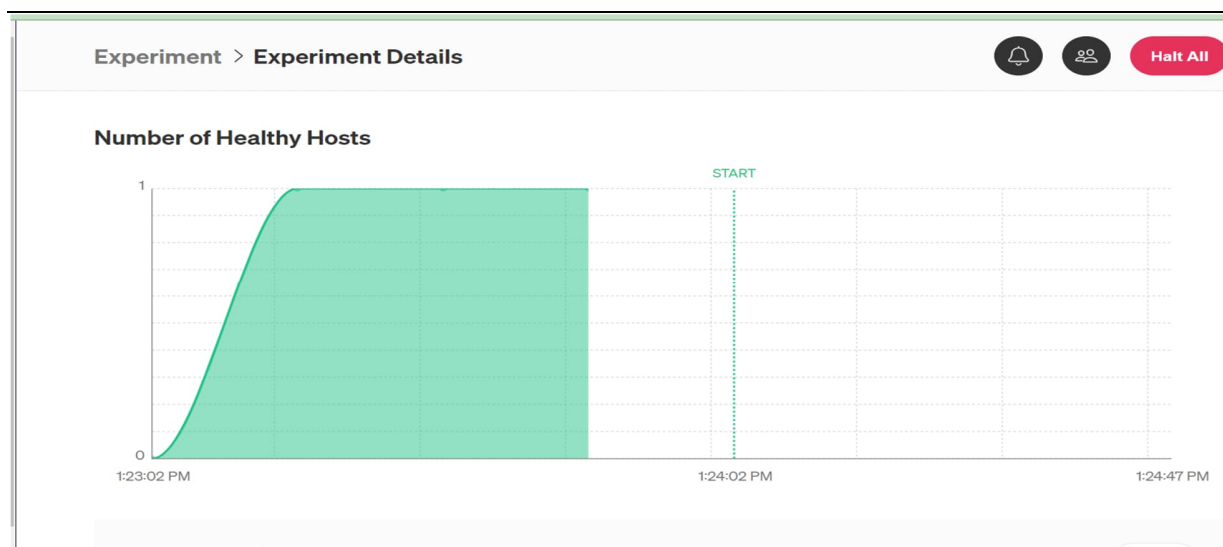


## Chaos Engineering – Hypothesis

### Gremlin: Using Gremlin

In the Kubernetes cluster with one master and two worker nodes, I have deployed the Juice Shop application using a deployment with two replicas. Additionally, there's a service exposing the application on port 30371, allowing connectivity through a web browser. With worker nodes having IP addresses 3.109.54.229 and 15.207.221.39, the Juice Shop instances are distributed across these nodes. In a fault-tolerance test, when worker1 was intentionally stopped using Gremlin, the Kubernetes system seamlessly redirected traffic to worker2. This demonstrated the **fault tolerance and high availability features of Kubernetes, as the application dynamically adapted to the failure by smoothly transitioning to the available worker node without disrupting the service, ensuring uninterrupted access to the Juice Shop through the browser.**

### Performing Gremlin attack:



### After Gremlin Attack:

## Chaos Engineering – Hypothesis

```

NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE             NOMINATED NODE   READINESS GATES
ice-shop-deployment-b46455fdc-j9zrn 1/1     Running   0           148m  192.168.189.65  worker2          <none>            <none>
ice-shop-deployment-b46455fdc-z9s6z 1/1     Running   1           148m  192.168.235.133 worker1          <none>            <none>
root@master:~# kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE             NOMINATED NODE   READINESS GATES
ice-shop-deployment-b46455fdc-j9zrn 1/1     Running   0           151m  192.168.189.65  worker2          <none>            <none>
ice-shop-deployment-b46455fdc-t2k5k 1/1     Running   0           37s   192.168.189.68  worker2          <none>            <none>
ice-shop-deployment-b46455fdc-z9s6z 1/1     Terminating 1           151m  192.168.235.133 worker1          <none>            <none>
root@master:~# kubectl get nodes
NAME      STATUS    ROLES                  AGE   VERSION
master    Ready     control-plane,master   159m  v1.21.1
worker1   NotReady  <none>                 157m  v1.21.1
worker2   Ready     <none>                 158m  v1.21.1
root@master:~#

```