|  |  |
| --- | --- |
| Name: | Sandeep Chowdary |
| Lab User ID: | 23SEK3324\_U03 |
| Date: | 09-01-2024 |
| Application Name: | [VulnerableJavaWebApplication](https://github.com/rafaelrpinto/VulnerableJavaWebApplication) |

**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)

AWS infra

user

Docker

https://<ip address>:9000

browser

container

Web app

browser

https://<ip address>:9000

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 web application is running on port number 9000 with https protocol. The user interface is not so good and there

are not much feature in the application. The application is loading properly taking very less time. The response time

is also good. The buttons are working properly in the application. The application responding to the requests from

the web browser which means the server is working properly in the normal conditions. The application running on a

single instance and in a single container. The container is running properly. The overall performance of the application

is good under normal conditions.

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**

While the system is assumed to handle a certain level of load, there may be unknown thresholds beyond which performance degrades unexpectedly

Users have appropriate access rights based on their roles, and unauthorized access attempts are appropriately restricted

**Unknown**

**Unknown**

**Known**

Cloud service outages may have unknown consequences on the system's performance and stability, potentially revealing dependencies or interactions that were not apparent under normal circumstances.

While we assume that the system scales efficiently, there may be inefficiencies or bottlenecks in resource scaling that impact performance under certain conditions

1. Users have appropriate access rights based on their roles, and unauthorized access attempts are appropriately restricted

2. While the system is assumed to handle a certain level of load, there may be unknown thresholds beyond which performance

degrades unexpectedly

3. While we assume that the system scales efficiently, there may be inefficiencies or bottlenecks in resource scaling

that impact performance under certain conditions

4. Cloud service outages may have unknown consequences on the system's performance and stability, potentially revealing

dependencies or interactions that were not apparent under normal circumstances.

Experiment:

(Document your Preparation, Implementation, Observation and Analysis )

1)Application overview: The application uses Spring Boot and an embedded H2 database that resets every time it

starts. If you break it just restart and everything will be reset. The application will run on HTTPS port 9000.

2)Making the application live: Created an ec2 instance with ubuntu os.

The application is made live using the docker containers. Following are the docker Commands used to run the application.

# gets the code

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

cd VulnerableJavaWebApplication

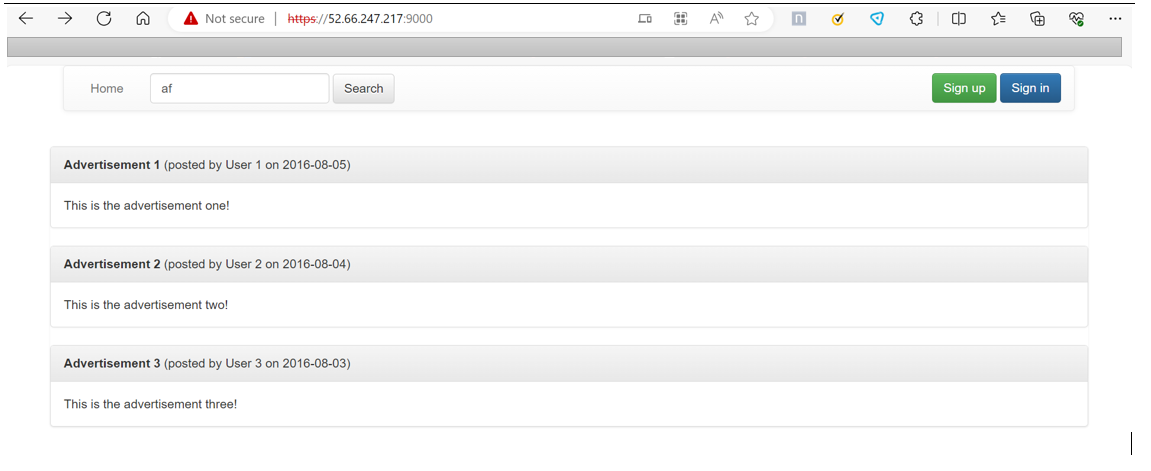
# creates the docker image

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

3)Checking the Application is live on port 9000:

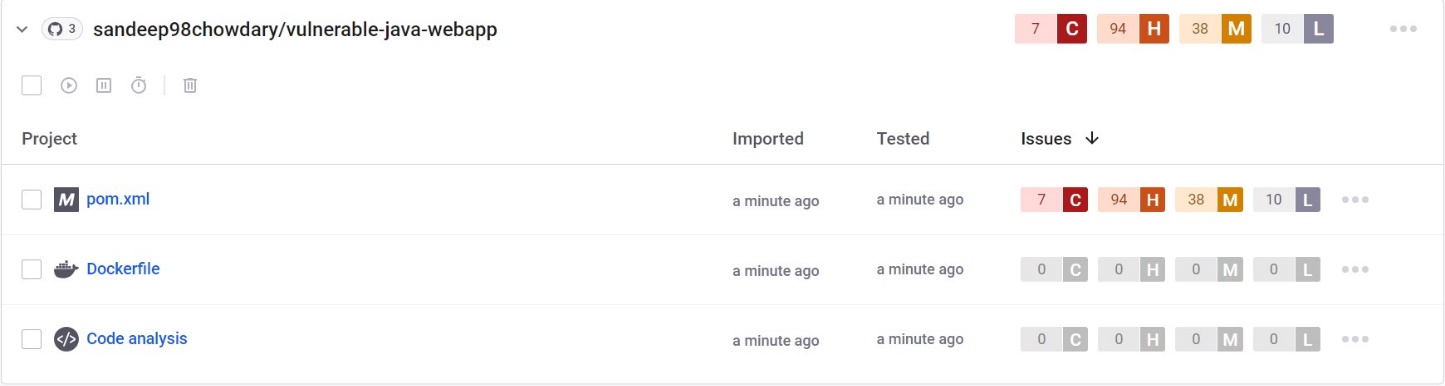


4) Scanning the vulnerabilities of the web application using the snyk tool:

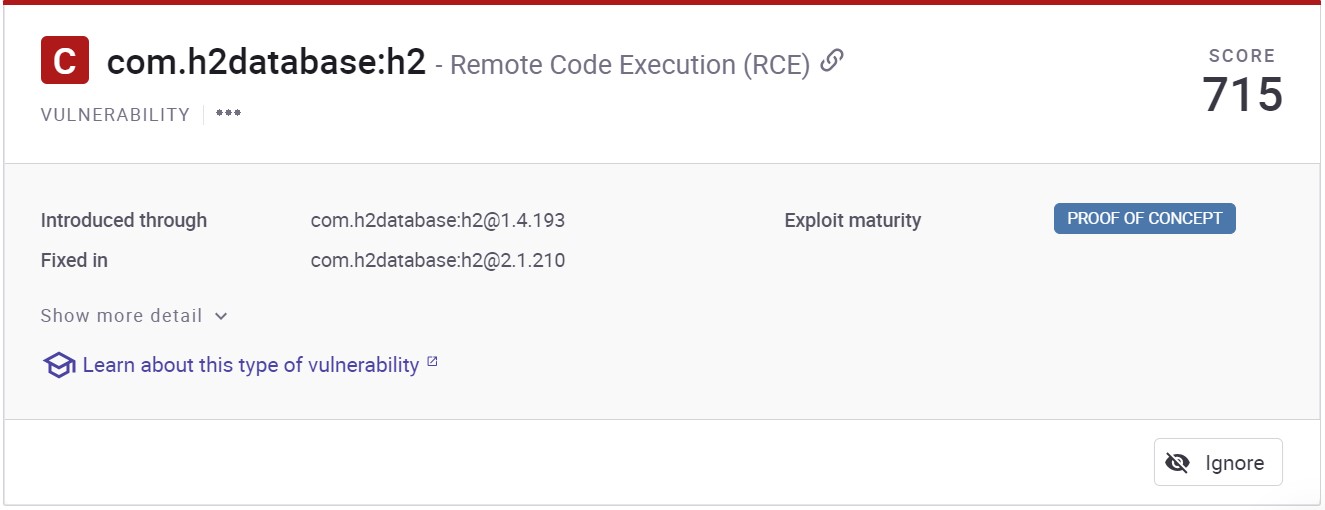
In the snyk I imported the git repo and snyk scanned the git repo for the vulnerability:

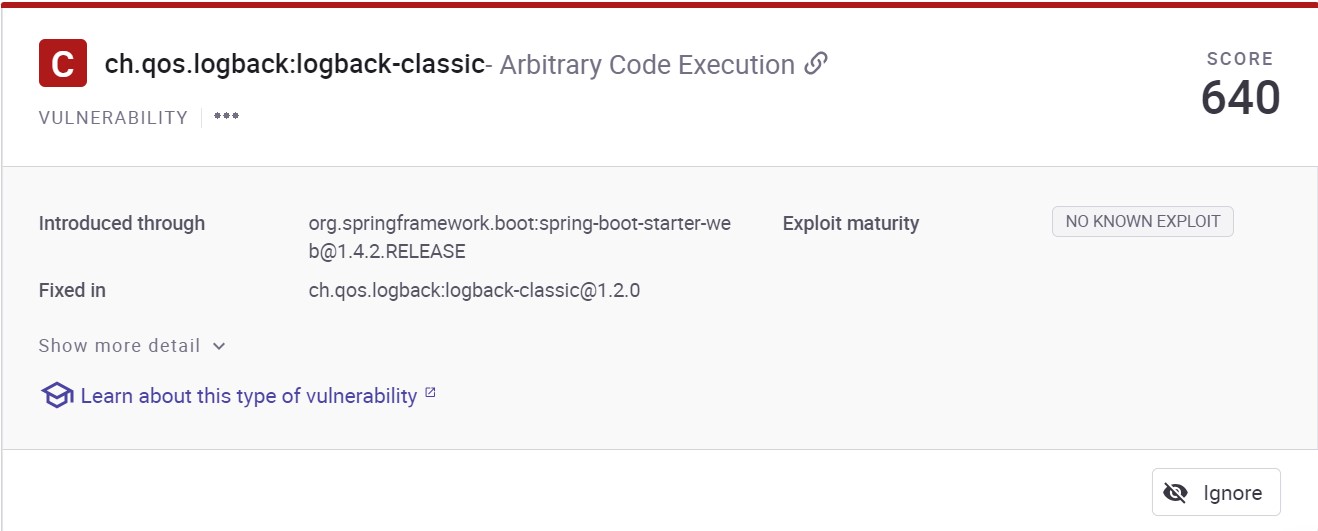
There are vulnerabilities only in the pom.xml file.

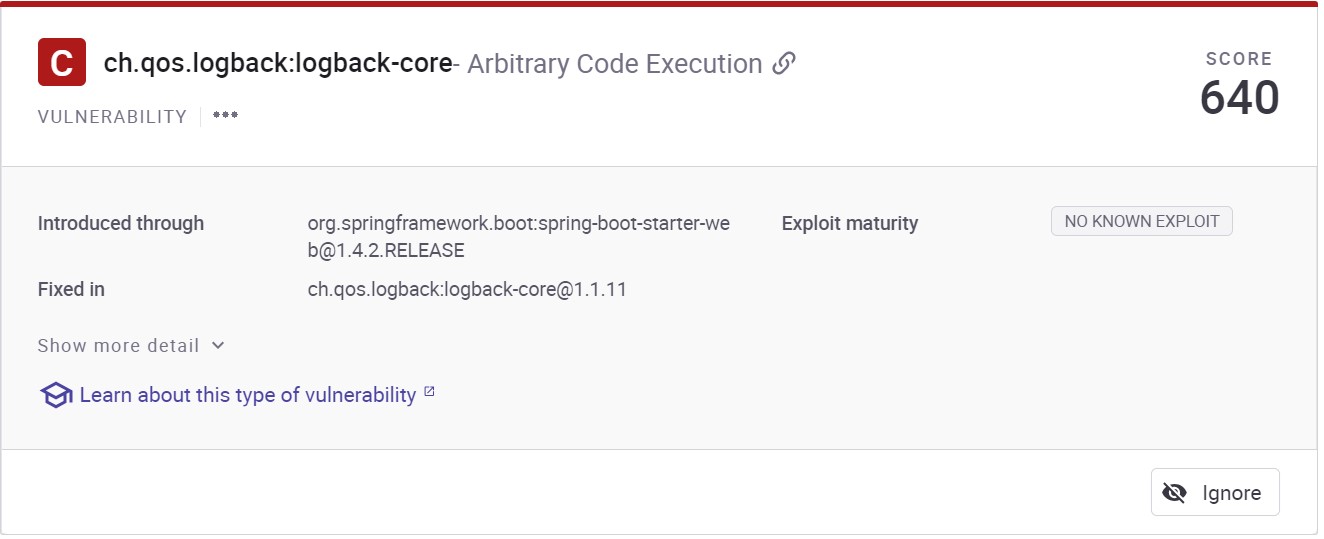
Critical=7, high= 94, medium=38, low=10



Few critical vulnerabilities:



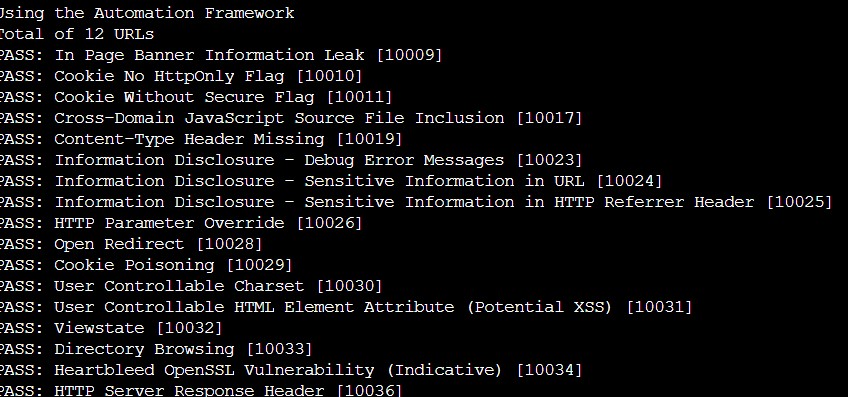


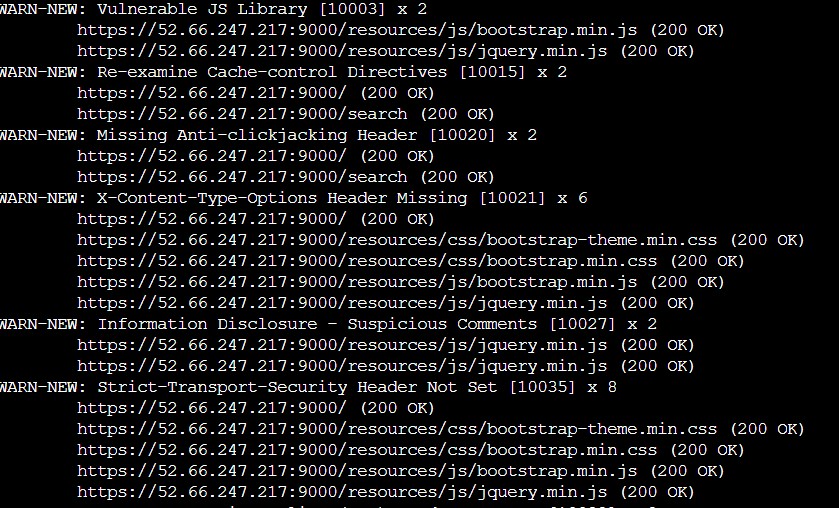


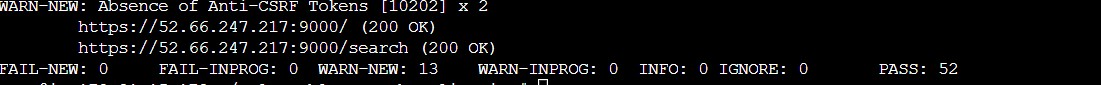
5) Scanning the vulnerabilities of the web application using the ZAP tool:

Used the ZAP baseline scanner to check the vulnerabilities:









Alerts and solutions:

Alert Id 10003

Risk Medium

vul: The identified library ExampleLibrary, version x.y.z is vulnerable.

sol: Please upgrade to the latest version of ExampleLibrary.

Alert Id 10015

Risk: Informational

vul: The cache-control header has not been set properly or is missing, allowing the browser and proxies to cache content. For static assets like css, js, or image files this might be intended, however, the resources should be reviewed to ensure that no sensitive content will be cached.

sol: For secure content, ensure the cache-control HTTP header is set with "no-cache, no-store, must-revalidate". If an asset should be cached consider setting the directives "public, max-age, immutable".

Alert Id 10020-1

Risk: Medium

vul: The response does not include either Content-Security-Policy with ‘frame-ancestors’ directive or X-Frame-Options to protect against ‘ClickJacking’ attacks.

sol: Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app. If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive.

Alert Id 10027

Risk Informational

vul: The response appears to contain suspicious comments which may help an attacker. Note: Matches made within script blocks or files are against the entire content not only comments.

sol: Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.

Alert Id 10038-1

Risk Medium

vul: Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.

sol: Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.

Alert Id: 10063-2

Risk: Low

vul: The header has now been renamed to Permissions-Policy.

sol: Ensure that your web server, application server, load balancer, etc. is configured to set the Permissions-Policy header instead of the Feature-Policy header.

Alert Id 10054

Risk Low

vul: A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a ‘cross-site’ request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks.

sol: Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies.

6) **Performing Chaos engineering by using Gremlin:**

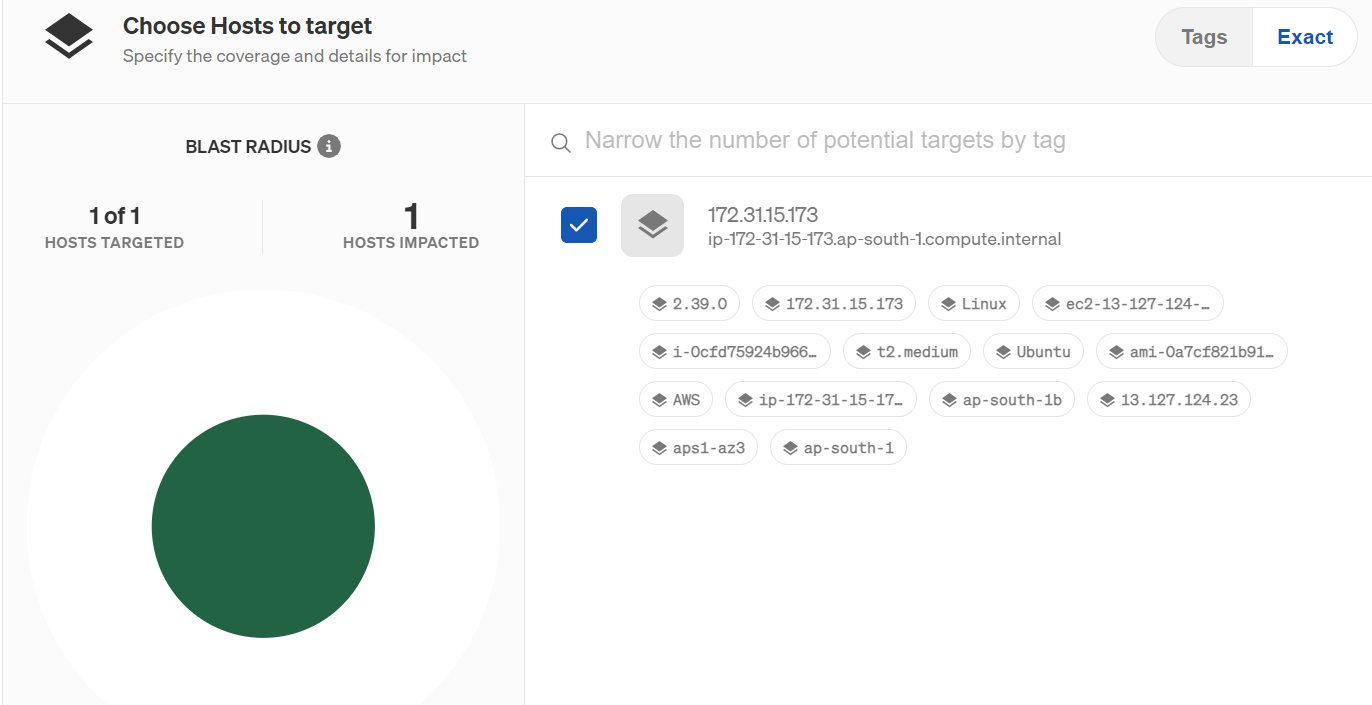
With the help of gremlin UI based web application I have performed a simple experiment to check the behavior of

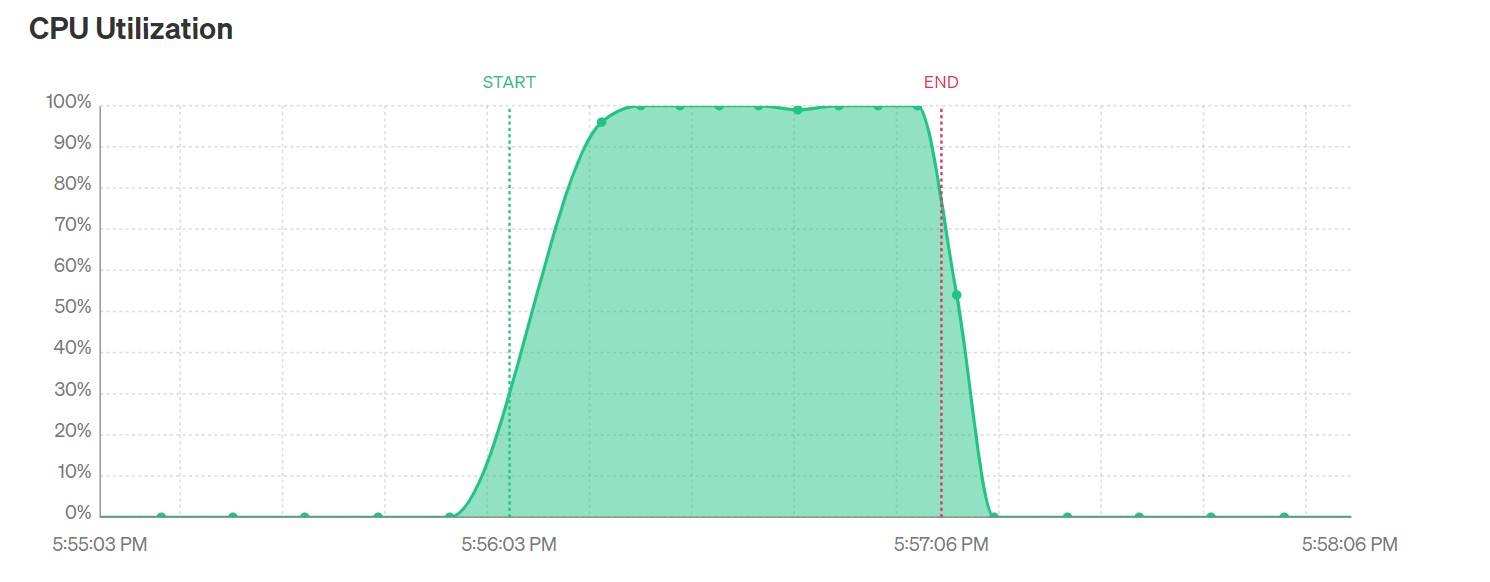
the system in a disruptive environment.

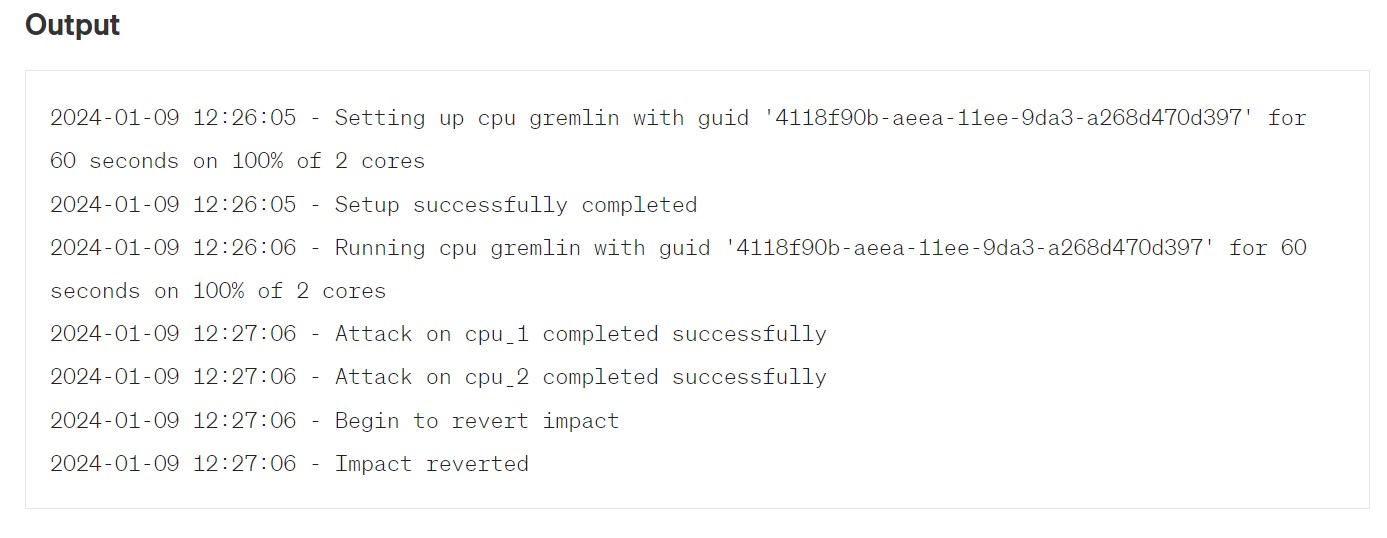
For this I logged in to my gremlin account.

I installed the gremlin agent over the host machine and attached the machine to the gremlin and performed the

the following experiments.







While the CPU utilization experiment was performed the web application did not respond properly as it has

not automatically scaled up. So there should be a autoscaling capability to the host machine.