|  |  |
| --- | --- |
| Name: | Dhwanit Pandya |
| Lab User ID: | 23SEK3324\_U34 |
| Date: | 10-01-2024 |
| Application Name: | VulnerableJavaWebApplication |

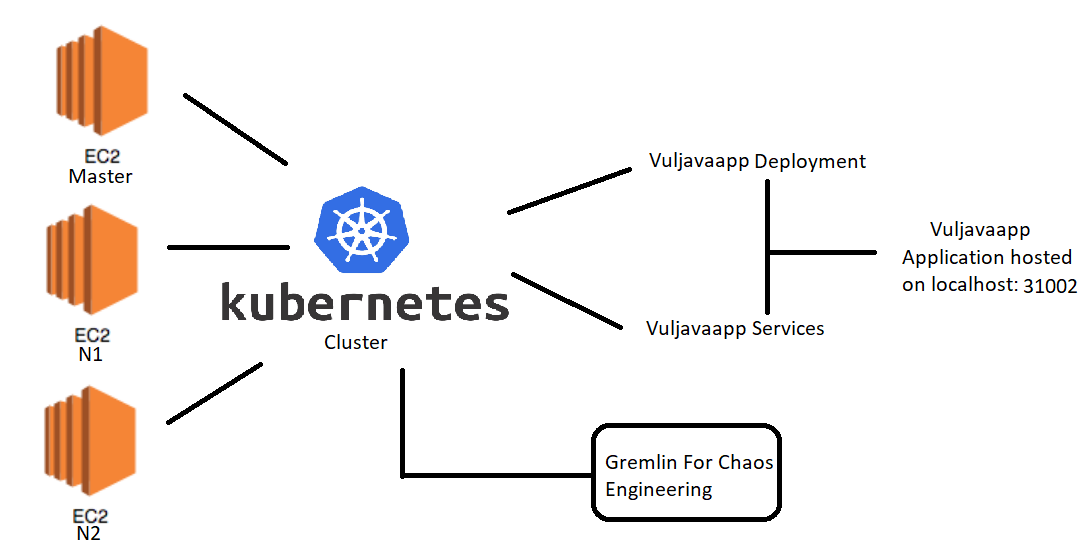
**Follow the below guidelines:**





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)



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 the steady state functionalities such as login / signup and working fine. Along with that reset password is also working as it should be.

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**

In case of node failure, we know replicas will be created but don’t know the time taken for it to do so.

If there is a node failure, then the pods start running on the active node using the replica.

**Unknown**

**Unknown**

**Known**

In case of node failure if replicas don’t get created due to some unknown reason

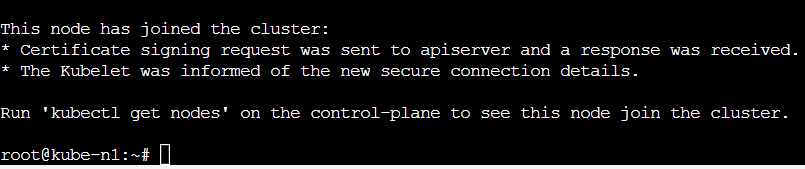
We replicas will be created but don’t know how much time it would take to restore the service.

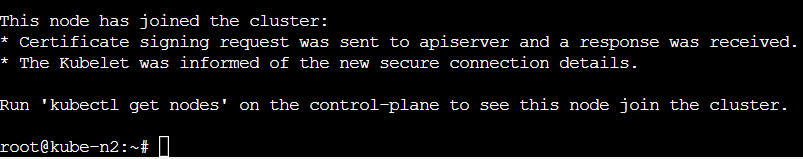
Experiment:

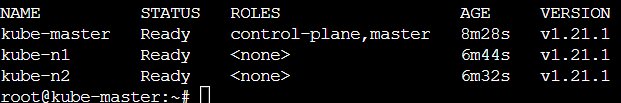
(Document your Preparation, Implementation, Observation and Analysis )

**Preparation:**

* Launched 3 virtual machines on AWS with Ubuntu 20.04
* Set up a Kubernetes cluster with master and two nodes (n1 & n2)







**Implementation:**

* Now creating a deployment using the following script which has replicas = 3

kind: Deployment

apiVersion: apps/v1

metadata:

name: vuljavaapp

spec:

replicas: 3

template:

metadata:

labels:

app: vuljavaapp

spec:

containers:

- name: vuljavaapp

image: dhwanit28/vuljapp

selector:

matchLabels:

app: vuljavaapp

* Now Creating a service using the following script

kind: Service

apiVersion: v1

metadata:

name: vuljavaapp

spec:

type: NodePort

selector:

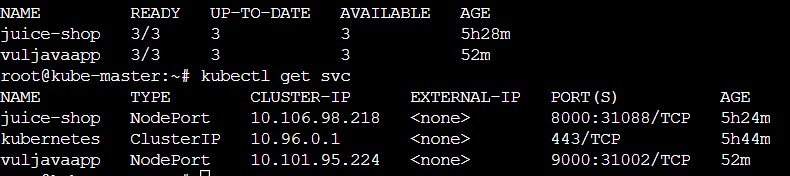
app: vuljavaapp

ports:

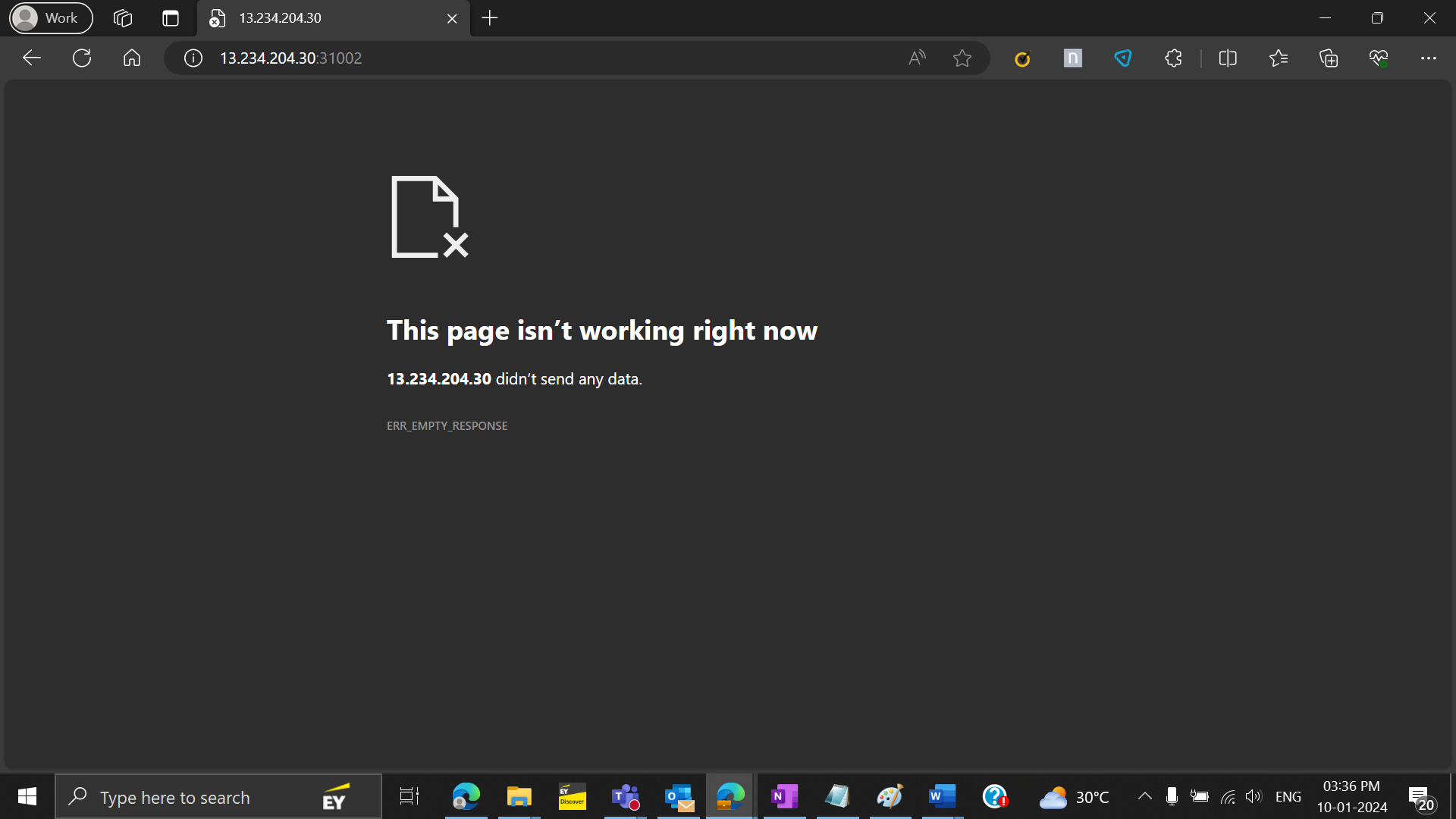
- name: http

port: 9000

targetPort: 9000

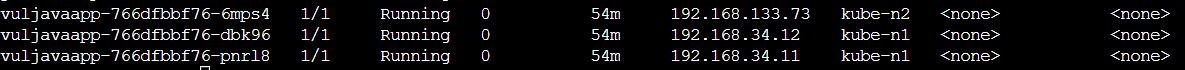


* **Now we can see the website gets hosted on localhost:31002 but not able to get any response**

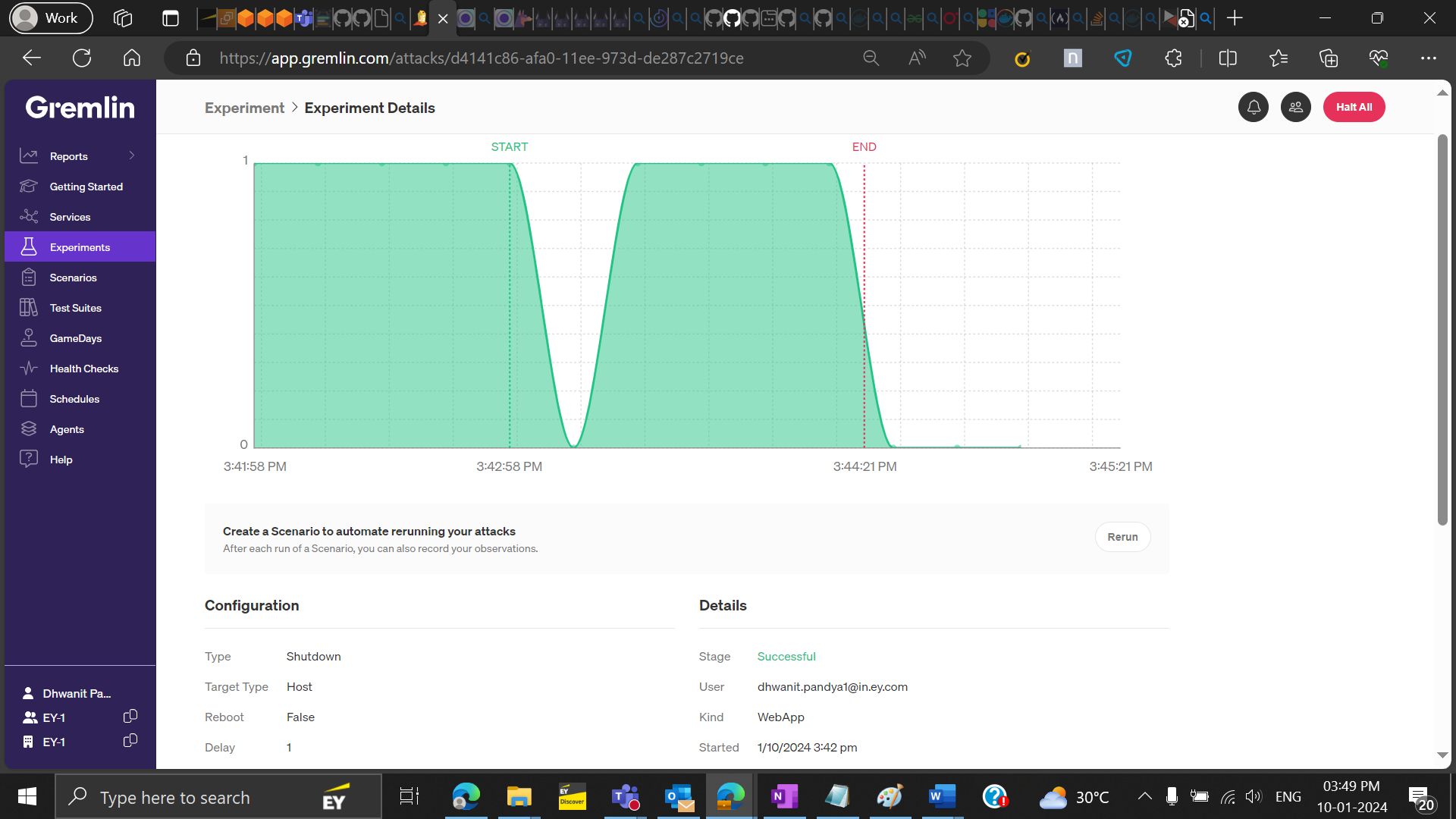


**Observation and Analysis**

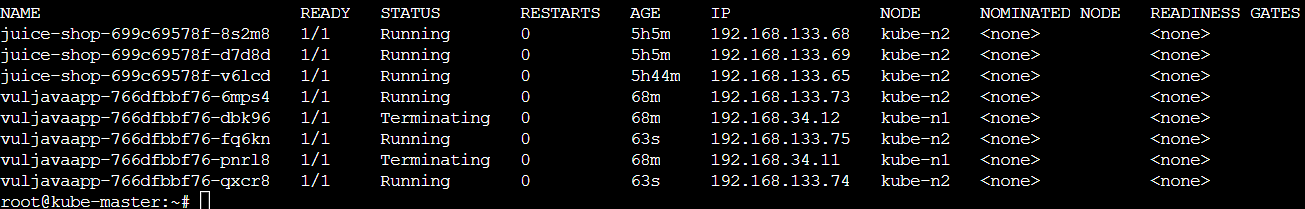
* **Performing chaos engineering on this cluster**
* **Initially 2 pods of juice-shop runs on node 1 and 1 pod runs on node 2**

****

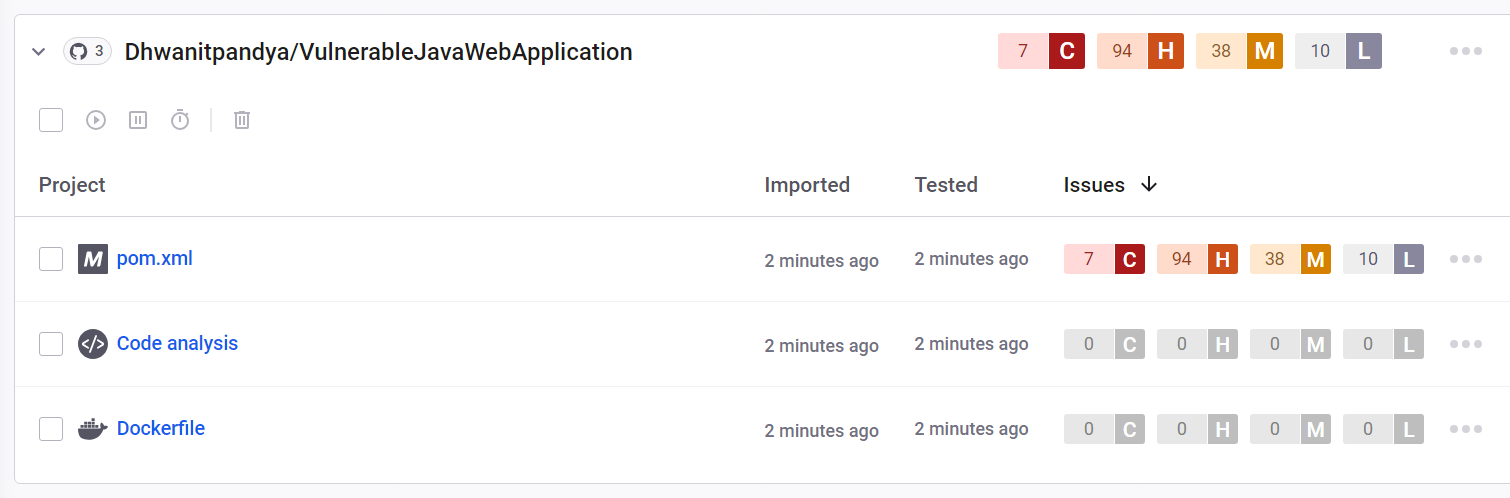
* **Now we run an experiment to shut down node 1**



* **We observe the following: even if node 1 shuts down then replica takes care of that and the pods that used to run on node 1 started running on node 2**

****

* Performed vulnerability analysis of the following repo using the snyk tool and found:

****

* Out of these Vulnerabilities some critical ones are and their fixes are mentioned below:

1. **Vul : ch.qos.logback:logback-classic- Insecure deserialization**

**Impact:** [may allow an attacker to manipulate serialized objects and pass harmful data into the application code](https://www.bing.com/ck/a?!&&p=4f85db4458f5dab7JmltdHM9MTcwNDg0NDgwMCZpZ3VpZD0xNjViMzI5ZS1kZjJmLTZhNjktMGE0Yy0yMWZjZGU4MjZiMTAmaW5zaWQ9NTcyOA&ptn=3&ver=2&hsh=3&fclid=165b329e-df2f-6a69-0a4c-21fcde826b10&psq=impact+of+Insecure+deserialization&u=a1aHR0cHM6Ly9wb3J0c3dpZ2dlci5uZXQvd2ViLXNlY3VyaXR5L2Rlc2VyaWFsaXphdGlvbg&ntb=1). [Insecure deserialization can lead to **denial of service, arbitrary code execution, or privilege escalation**](https://www.bing.com/ck/a?!&&p=4944b3dbeab78a1fJmltdHM9MTcwNDg0NDgwMCZpZ3VpZD0xNjViMzI5ZS1kZjJmLTZhNjktMGE0Yy0yMWZjZGU4MjZiMTAmaW5zaWQ9NTczMQ&ptn=3&ver=2&hsh=3&fclid=165b329e-df2f-6a69-0a4c-21fcde826b10&psq=impact+of+Insecure+deserialization&u=a1aHR0cHM6Ly93d3cuYWN1bmV0aXguY29tL2Jsb2cvYXJ0aWNsZXMvd2hhdC1pcy1pbnNlY3VyZS1kZXNlcmlhbGl6YXRpb24v&ntb=1)

**Fix**:

* Introduce digital signatures and other integrity checks to stop malicious object creation or other data interfering.
* Run deserialization code in low privilege environments.
* Keep a log with deserialization exceptions and failures.
* Use language-agnostic methods for deserialization such as JSON, XML, or YAML.
* Use safer API which avoids the use of the interpreter.

1. **Vul : org.apache.tomcat.embed:tomcat-embed-core-Information Exposure**

**Impact:** If the send file processing completed quickly, it was possible for the Processor to be added to the processor cache twice. This could result in the same Processor being used for multiple requests which in turn could lead to unexpected errors and/or response mix-up.

**Fix**: It enables a potential attacker to understand the state of the login function, and could allow an attacker to discover a valid username by trying different values until the incorrect password message is returned. In essence, this makes it easier for an attacker to obtain half of the necessary authentication credentials.

1. **Vul : org.springframework:spring-beans-Remote Code Execution**

**Impact:** Thisvulnerability allows an attacker to perform remote code execution on an application server running a vulnerable configuration, giving them full access to the compromised server.

**Fix**: here are some fixes:

1. Upgrade Spring Framework to a version equal to or greater than 5.2.20 or 5.3.18.
2. If you are using Spring Boot directly, upgrade to a version equal to or greater than 2.6.6.
3. **Vul : org.apache.tomcat.embed:tomcat-embed-core-Remote Code Execution**

**Impact:** It allows an attacker to inject malicious code into an application through a user input field, which is then executed on the fly. Can result in a total loss of integrity, availability, and confidentiality within the application. An attacker may also abuse a code injection vulnerability to execute terminal commands on that server and pivot to adjacent systems.

**Fix**: here are some fixes:

* + - Avoid the use of dangerous functions
    - Reconsider the need for dynamic code execution
    - Lock down the interpreter
    - Utilize a static analysis tool

1. **Vul : socket.io-parser-Denial of Service (DoS)**

**Impact:** ReDoS attack attempts to slow down or even render an application unavailable. processing of the malicious string exhausts the computing power or memory available, thus impacting the application's performance and, in certain circumstances, causing a denial of service (or DoS).

**Fix**: Avoid using regex for user input validation. Closely review and analyze all patterns before implementation to ensure they do not contain any evil regex patterns.