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

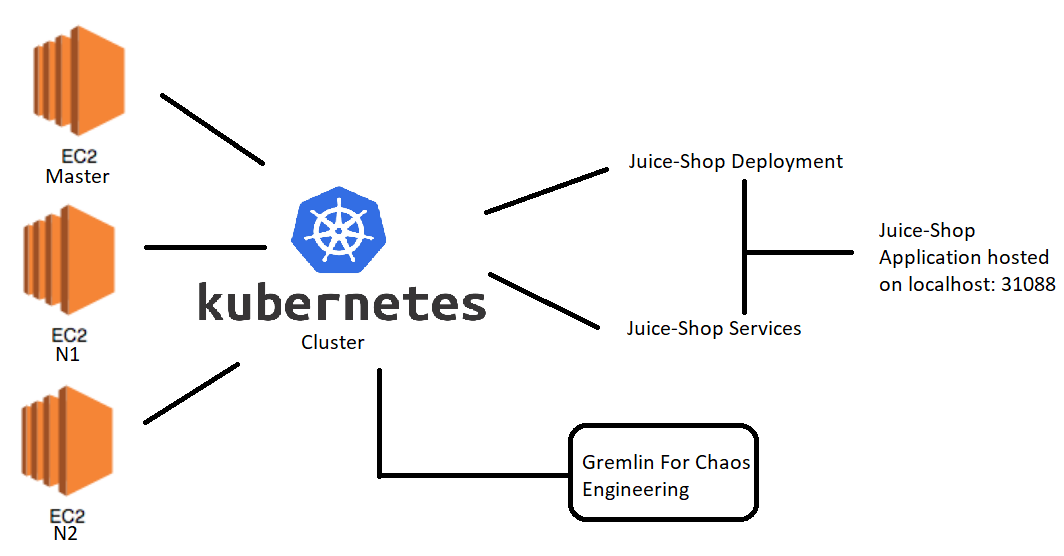
**Follow the below guidelines:**





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. User is able to see the production information when they click on the product.

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

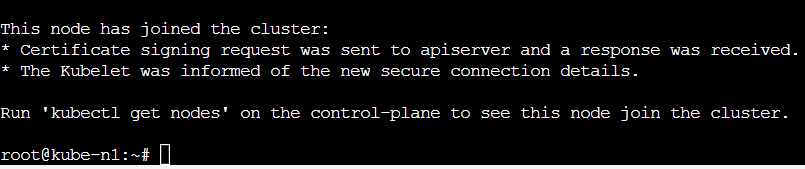
We replicas will be created but don’t know how much time it would take to restore the service.

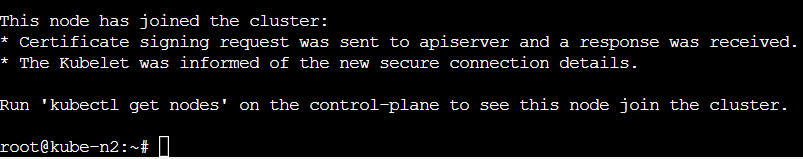
Experiment:

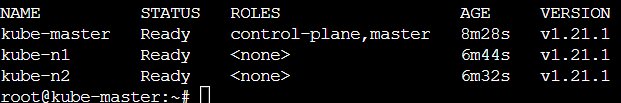
(Document your Preparation, Implementation, Observation and Analysis )

**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: juice-shop

spec:

replicas: 3

template:

metadata:

labels:

app: juice-shop

spec:

containers:

- name: juice-shop

image: bkimminich/juice-shop

selector:

matchLabels:

app: juice-shop

* Now Creating a service using the following script

kind: Service

apiVersion: v1

metadata:

name: juice-shop

spec:

type: NodePort

selector:

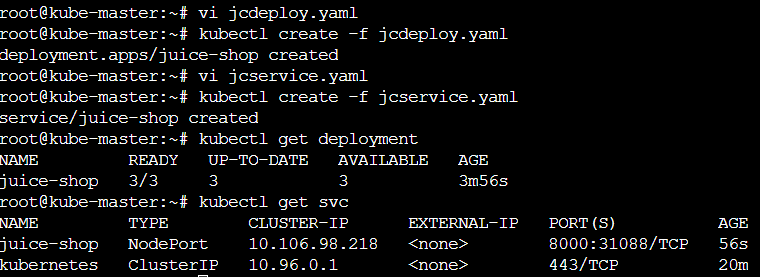
app: juice-shop

ports:

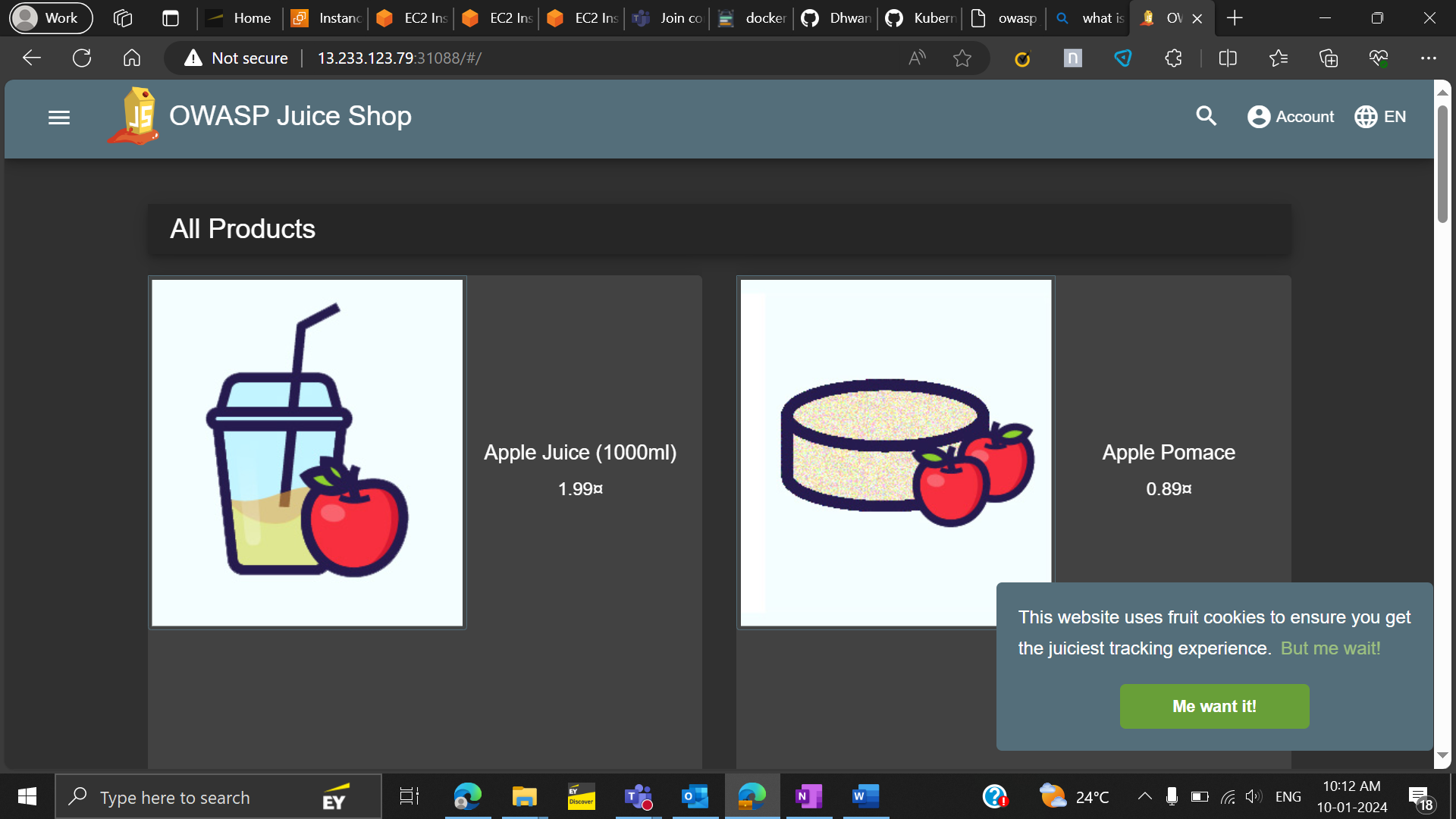
- name: http

port: 8000

targetPort: 3000

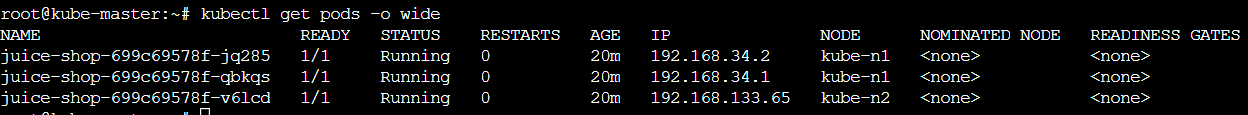


* **Now we can see the website gets hosted on localhost:31088**

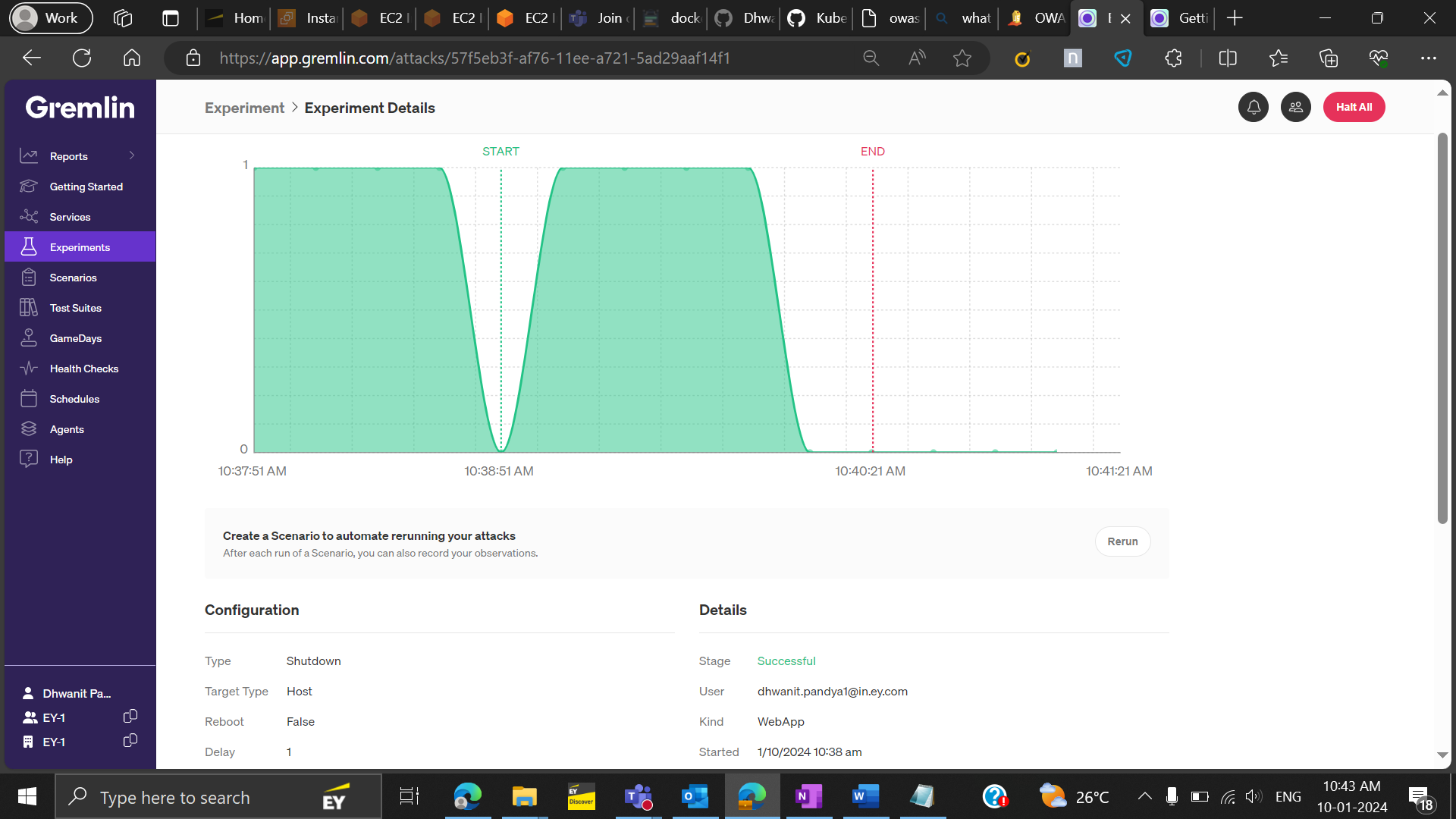


**Observation and Analysis**

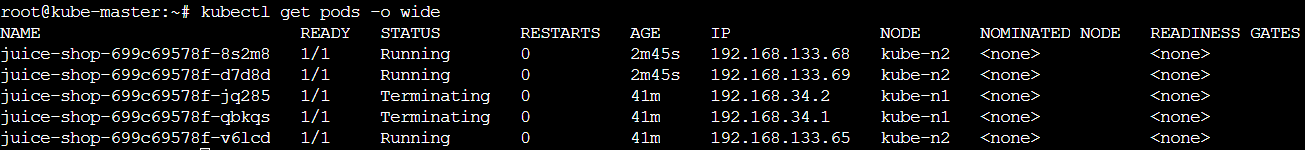
* **Performing chaos engineering on this cluster**
* **Initially 2 pods of juice-shop runs on node 1 and 1 pod runs on node 2**

****

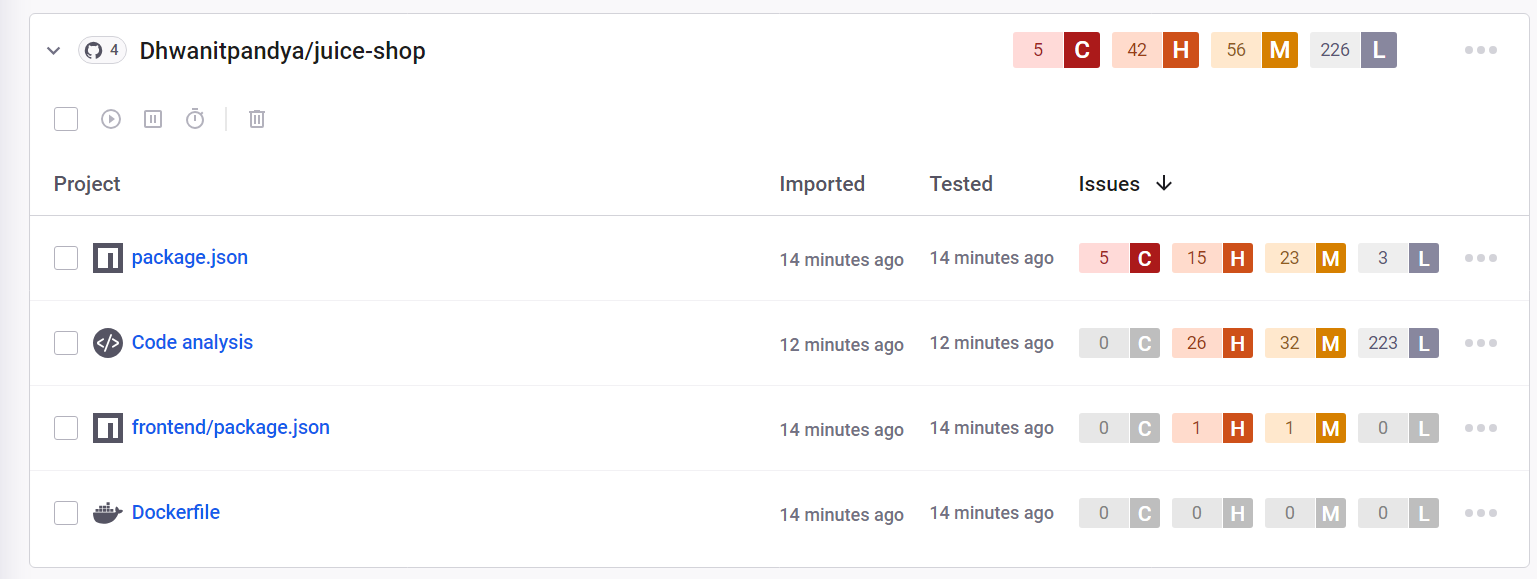
* **Now we run an experiment to shut down node 1**



* **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 : Cross-site Scripting (XSS)**

**Impact:** XSS allows hackers to inject malicious JavaScript into a web application. Such injections are extremely dangerous from the security perspective, and can lead to:

* Stealing sensitive information, including session tokens, cookies or user credentials
* Injecting multiple types of malware (e.g. worms) into the website
* Changing the website appearance to trick users into performing undesirable actions

**Fix**: The first step to mitigate XSS is to find all places in your code where this pattern occurs. Then the most important XSS mitigation step is to escape your HTML output. To do that, you should HTML-encode all dangerous characters in the user-controlled data before injecting that data into your HTML output.

1. **Vul : sanitize-html-Arbitrary Code Execution**

**Impact:** 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 : vm2 Remote Code Execution (RCE)**

**Impact:** 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 : vm2 Sandbox Bypass**

**Impact:** Affected versions of this package are vulnerable to Sandbox Bypass by abusing an unexpected creation of a host object based on the maliciously crafted specification of Proxy. Exploiting this vulnerability allows an attacker to gain remote code execution rights on the host running the sandbox via the Function constructor.

**Fix**: No fix as such but to prevent sandbox bypass attacks, organizations can take several measures, such as keeping their security tools and sandbox updated with the latest patches and signatures, using multiple layers of defense, such as endpoint protection, network security, and threat intelligence, educating their employees about the risks of opening suspicious emails, links, or attachments, and implementing security policies that restrict access to sensitive data and critical systems.

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.