|  |  |
| --- | --- |
| Name: | Rishitha Pudi |
| Lab User ID: | 23SEK3324\_U01 |
| Date: | 09-01-2024 |
| Application Name: | Juice-shop |

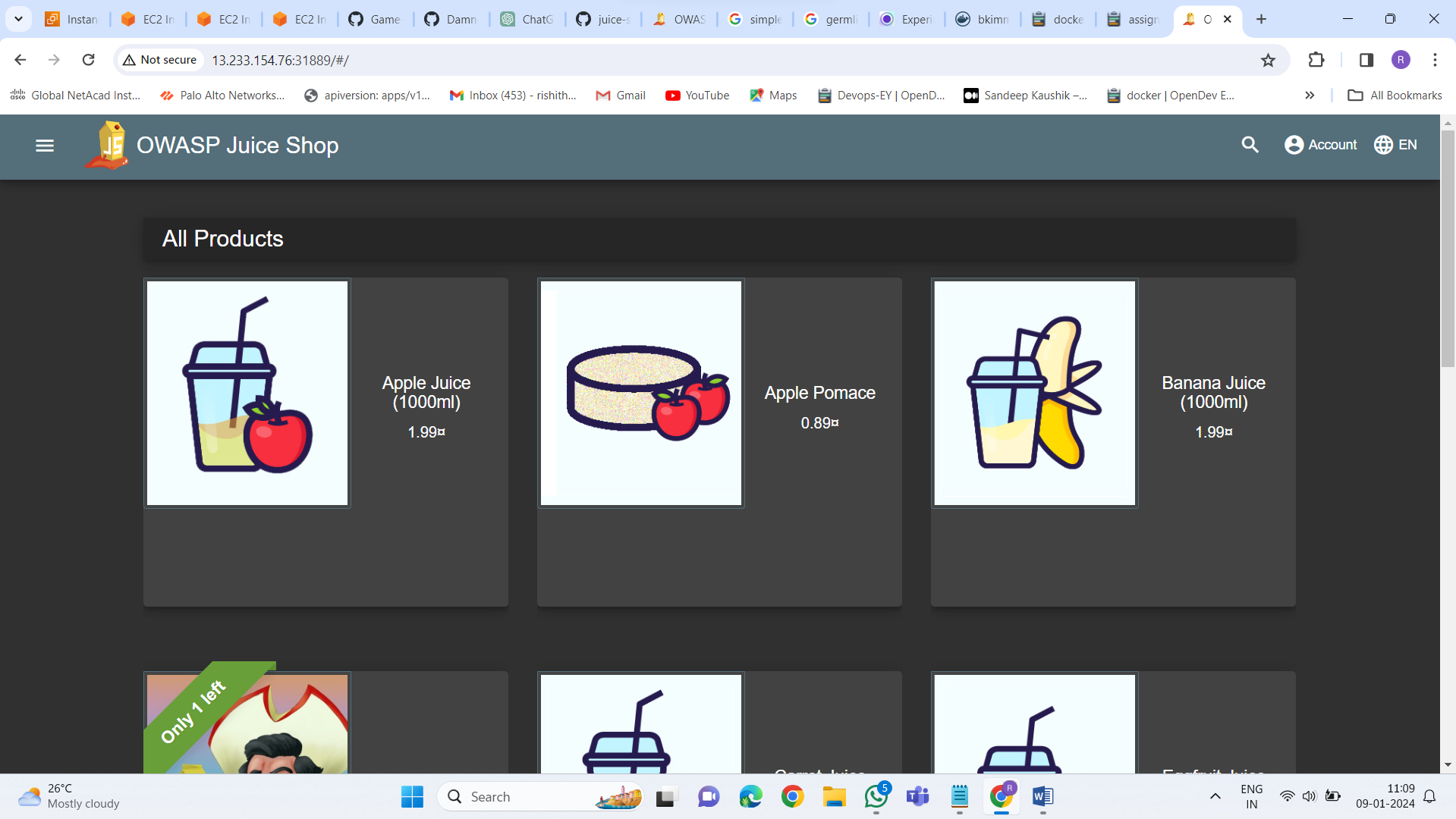
**Follow the below guidelines:**





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)



aws

Kubernet Cluster

Browser

Master

3.110.135.61:31889

13.233.154.7:31889

Kubernet service

Node 1

Node 2

13.233.154.76

3.110.135.61

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 server starts, listening on defined ports like ip address: 31889.

A user access a website hosted on this server via a web browser.

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

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.")



Chaos engineering hypothesis scenarios

You know a specific component can fail (e.g., database server overload). You also know the expected behavior (e.g., service degradation, automatic failover).

You know a new software update might cause regressions, but you're unsure which features or integrations might be affected. Chaos experiments involving the update can reveal these unknowns.

**Known**

942236

**Unknown**

You know a general category of potential issues (e.g., network fluctuations). However, you don't know the specific triggers or consequences.

Introducing resource starvation (memory, CPU) across different components can expose hidden dependencies and resource bottlenecks.

**Unknown**

**Known**

Experiment:

(Document your Preparation, Implementation, Observation and Analysis )

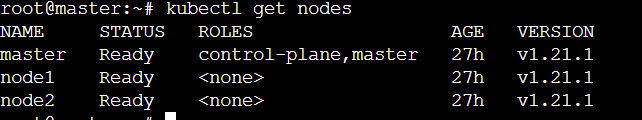
**Overview:** OWASP Juice Shop is an intentionally vulnerable web application developed by the Open

Web Application Security Project (OWASP). Now we are performing some security tools against

Application to check the vulnerabilities.

**Methodology:**

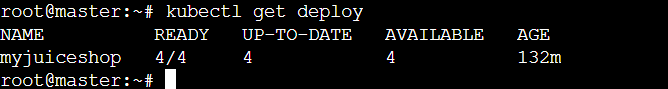
**Step 1**: set up the kubernet cluster, Deploy 3 EC2 instances master node 1 and node2 . Then step up kubernet cluster in all 3 nodes using kubernet



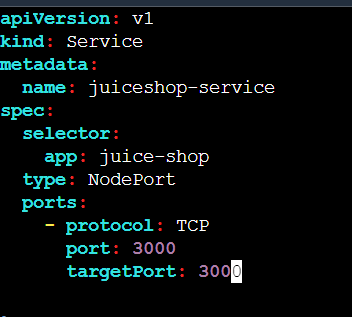
**Step 2** : When your nodes are in ready state. Deploy juice-shop application



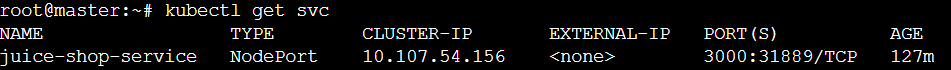
Using that command we can deploy juice shop application and create 4 replicas at a time



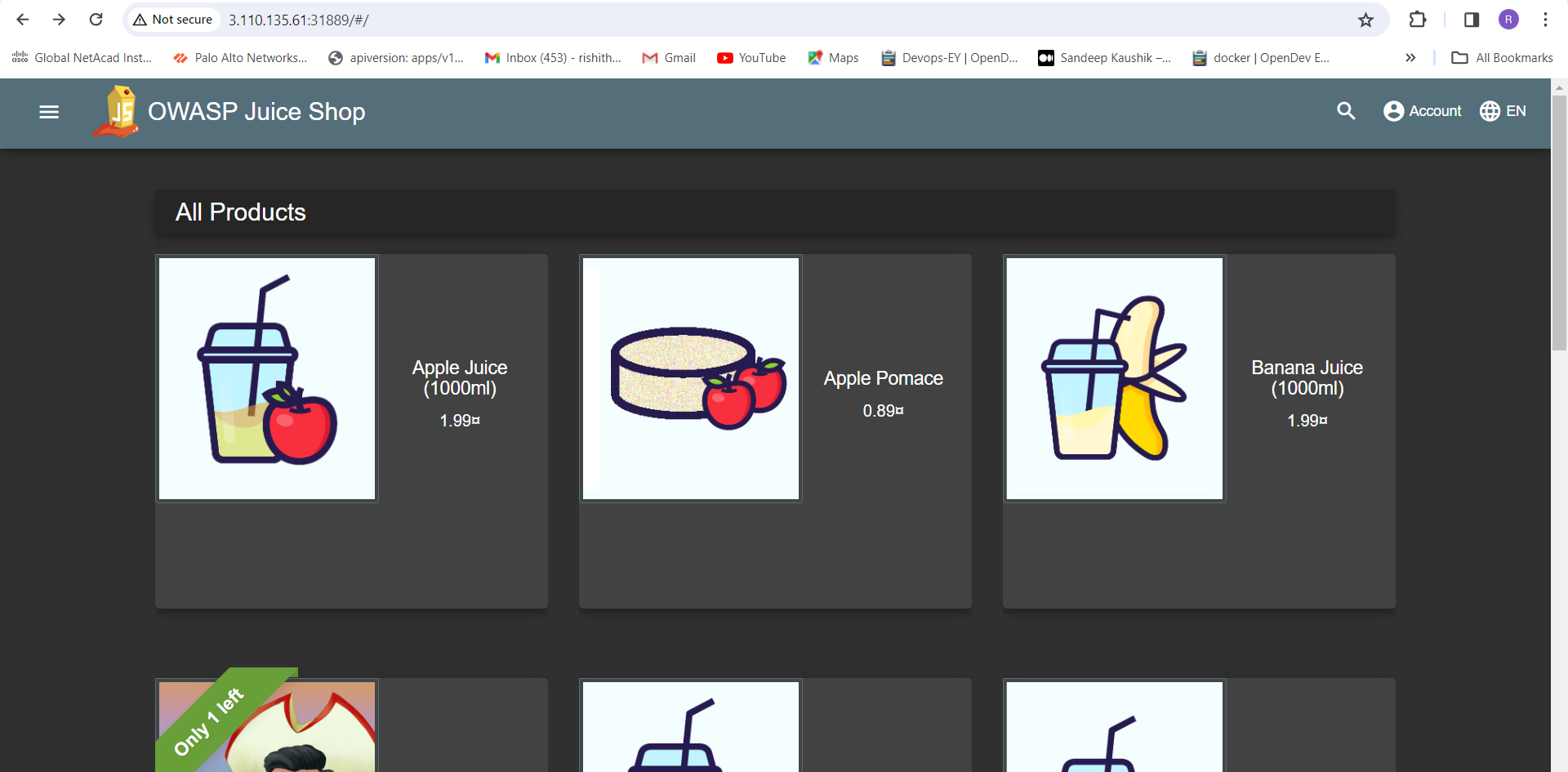
**Step 3**: creating service to expose our application



That yawl file creates the service, using apply command we can apply that command.



**Step 4**: live that application



**Tools**: The following are the tools used in this project to do vulnerability analysis

1. OWSAP ZEP
2. SNYK
3. Germlin

**Observation:**

**OWSAP ZEP:**



Using that command u can perform security analysis over your machine after –t u are supposed to give url of our live

Application.

**Analysis 1:**



Out of 94 test juice-shop only pass 56 tests and 9 are warn –new

**Cross-Domain JavaScript source file inclusion:** The page includes one or more script files from a third-party domain.

**Solution**: is Ensure JavaScript source files are loaded from only trusted sources, and the sources can't be controlled by end users of the application.

**Information Disclosure suspicious comments:** 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.

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

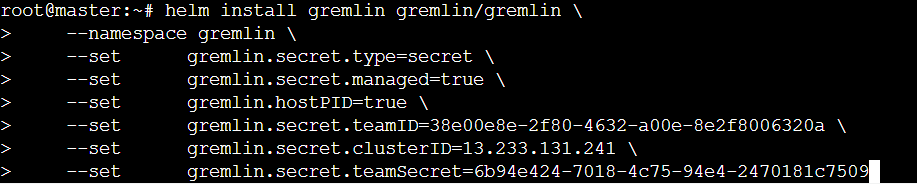
**Content security policy:** Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attack.  
**Solution:** Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.

**Permission policy header not set:** Permissions Policy Header is an added layer of security that helps to restrict from unauthorized access or usage of browser/client features by web resources. This policy ensures the user privacy by limiting or specifying the features ofthe browsers can be used by the web resources.  
**Solution**: Ensure that your web server, application server, load balancer, etc. is configured to set the Permissions-Policy header.

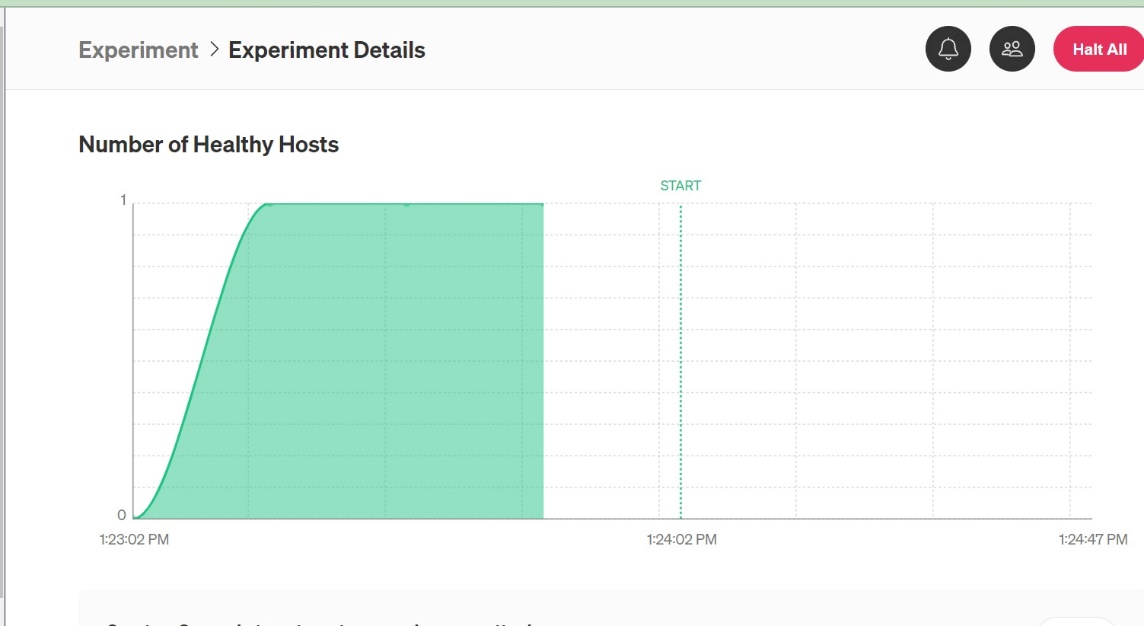
**Dangerous Js function:** A dangerous JS function seems to be in use that would leave the sitevulnerable.   
**Solution:** See the references for security advice on the use of these functions.

**Analysis 2**: using germline

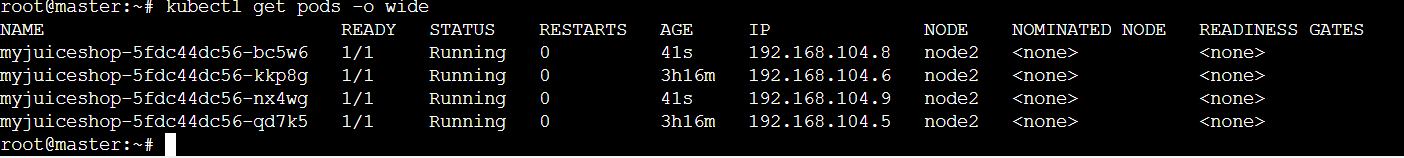
We can set up the agent using the following commands



Performing the experiment of shutting my one node to check the reliability of the system,

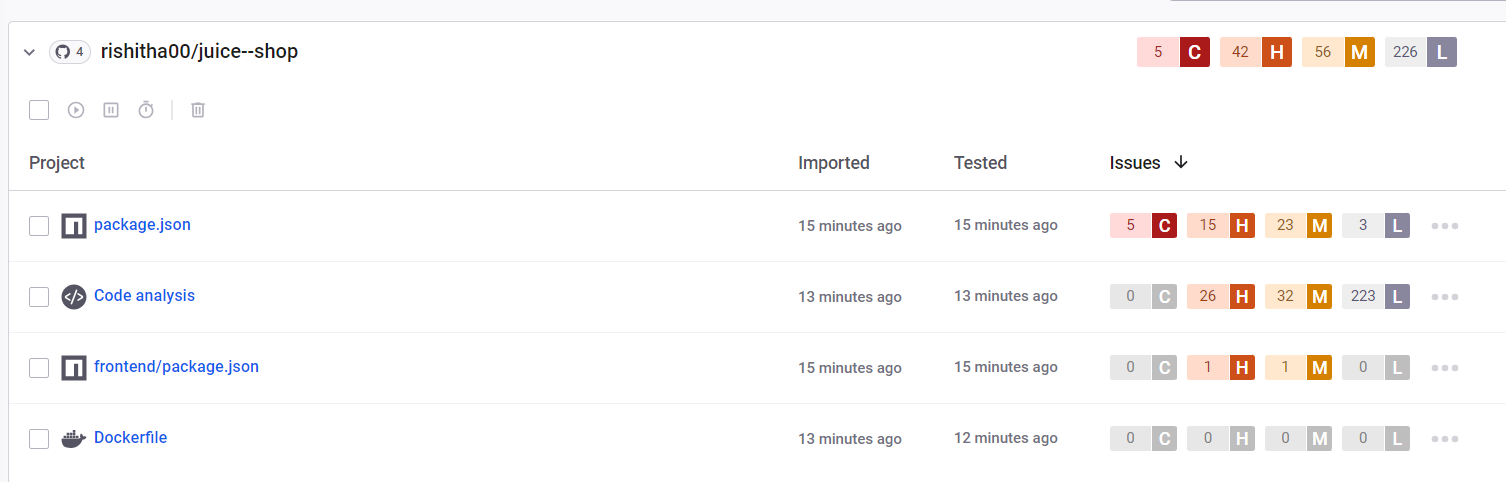


If completing successfully completing the experiment we can check reliability their.



All the nodes are working on the node2 itself.

**Analysis 3:** Using synk



The following are the causes for the vulnerability and their solutions

**Uninitialized Memory Exposure:** Sensitive information could include data that is sensitive in and of itself (such as credentials or private messages), or otherwise useful in the further exploitation of the system (such as internal file system structure).

**Solution:** Ensure that any possibly sensitive data specified in the requirements is verified with designers to ensure that it is either a calculated risk or mitigated elsewhere. Any information that is not necessary to the functionality should be removed in order to lower both the overhead and the possibility of security sensitive data being sent.

## Regular Expression Denial of Service (ReDoS): The product does not properly control the allocation and maintenance of a limited resource, thereby enabling an actor to influence the amount of resources consumed, eventually leading to the exhaustion of available resources.

## Solution: Ensure that protocols have specific limits of scale placed on them. Ensure that all failures in resource allocation place the system into a safe posture.

## Authentication Bypass: When an actor claims to have a given identity, the product does not prove or insufficiently proves that the claim is correct.

## Solution: The following code intends to ensure that the user is already logged in. If not, the code performs authentication with the user-provided username and password. If successful, it sets the loggedin and user cookies to "remember" that the user has already logged in. Finally, the code performs administrator tasks if the logged-in user has the "Administrator" username, as recorded in the user cookie.

## Arbitrary Code Execution: The product constructs all or part of a code segment using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the syntax or behavior of the intended code segment.

## Solution: Use dynamic tools and techniques that interact with the product using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The product's operation may slow down, but it should not become unstable, crash, or generate incorrect results.

## Command Injection: The product constructs all or part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.

## Solution: Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. These will help the programmer encode outputs in a manner less prone to error.