|  |  |
| --- | --- |
| Name: | **Subhasish Mukherjee** |
| Lab User ID: | **23SEK3324\_U05** |
| Date: | 10/01/24 |
| Application Name: | **OWASP / WRONG SECRETS** |

**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)

A screenshot of a computer

Description automatically generated

Ip add:8080

Web App

8080

The web server initiates, listening on specified ports, like IP address: 8080. A user interacts with the hosted website through a web browser.

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 initiates, listening on specified ports, like IP address: 8080. A user interacts with the hosted website through a web browser. The site comprises five components: Home, Challenges, GitHub, Stats, and About. The Challenges section displays various tasks, and upon user selection, redirects to dedicated challenge pages. These pages prompt users to submit solutions. If correct, the server reacts with a positive "Good Answer"; otherwise, a "Bad Luck, Try Again" message is displayed, accompanied by hints for assistance. This interactive platform engages users in problem-solving, fostering a dynamic and educational web experience. Additionally, users can explore GitHub for related resources and view stats related to challenges. Overall, the web server offers an interactive and educational environment with challenges, solutions, and community engagement elements.

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**

If we induce a CPU outage by simulating high load, then the system will automatically scale up to handle increased demand, maintaining performance and availability.

If the application experiences any kind of external attack, then its defense mechanisms will effectively mitigate the threat, preventing unauthorized access or data compromise.

**Unknown**

If a SQL injection attack is executed on the vulnerable Java application, then the application's security measures will may not prevent or mitigate the unauthorized database access attempt. Causing any kind of security issues.

Engineers simulate real-world unpredictability by randomly manipulating network bandwidth or introducing intermittent latency in various system components to test and ensure robustness under diverse and challenging conditions.

**Unknown**

**Known**

Experiment:

(Document your Preparation, Implementation, Observation and Analysis )

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

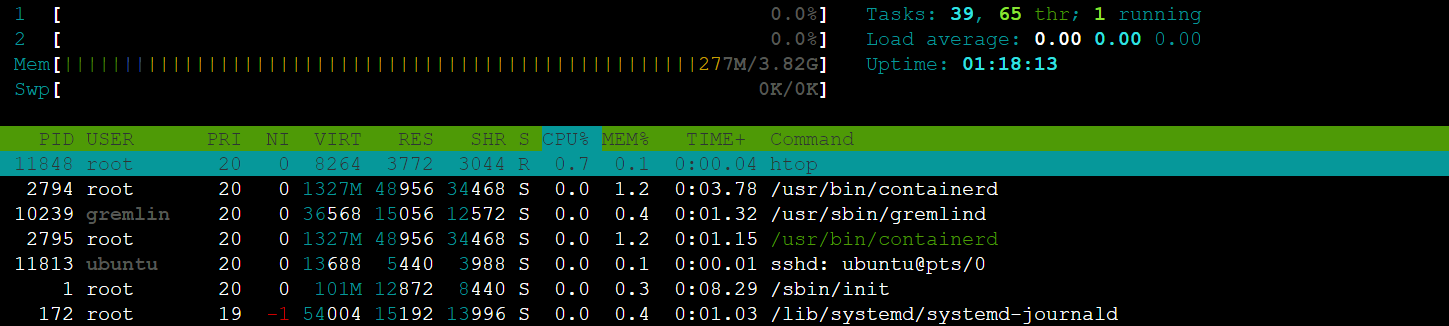
1. **GREMLIN**
2. **SNYK**
3. **OWASAP ZAP**

**Observation:**

**Gremlin: Using Gremlin**

Using gremlin, we can perform CPU utilization.

Before performing gremlin attack:

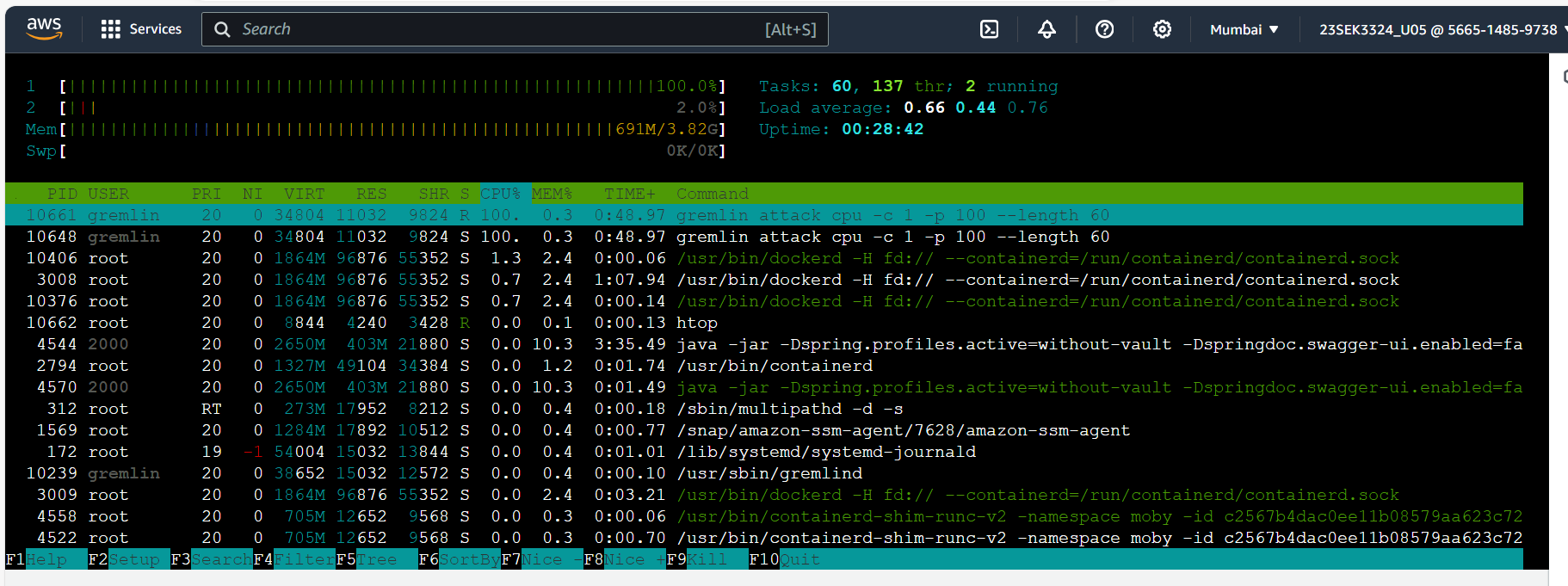


The CPU utilization is 0.7 %

After performing gremlin attacks

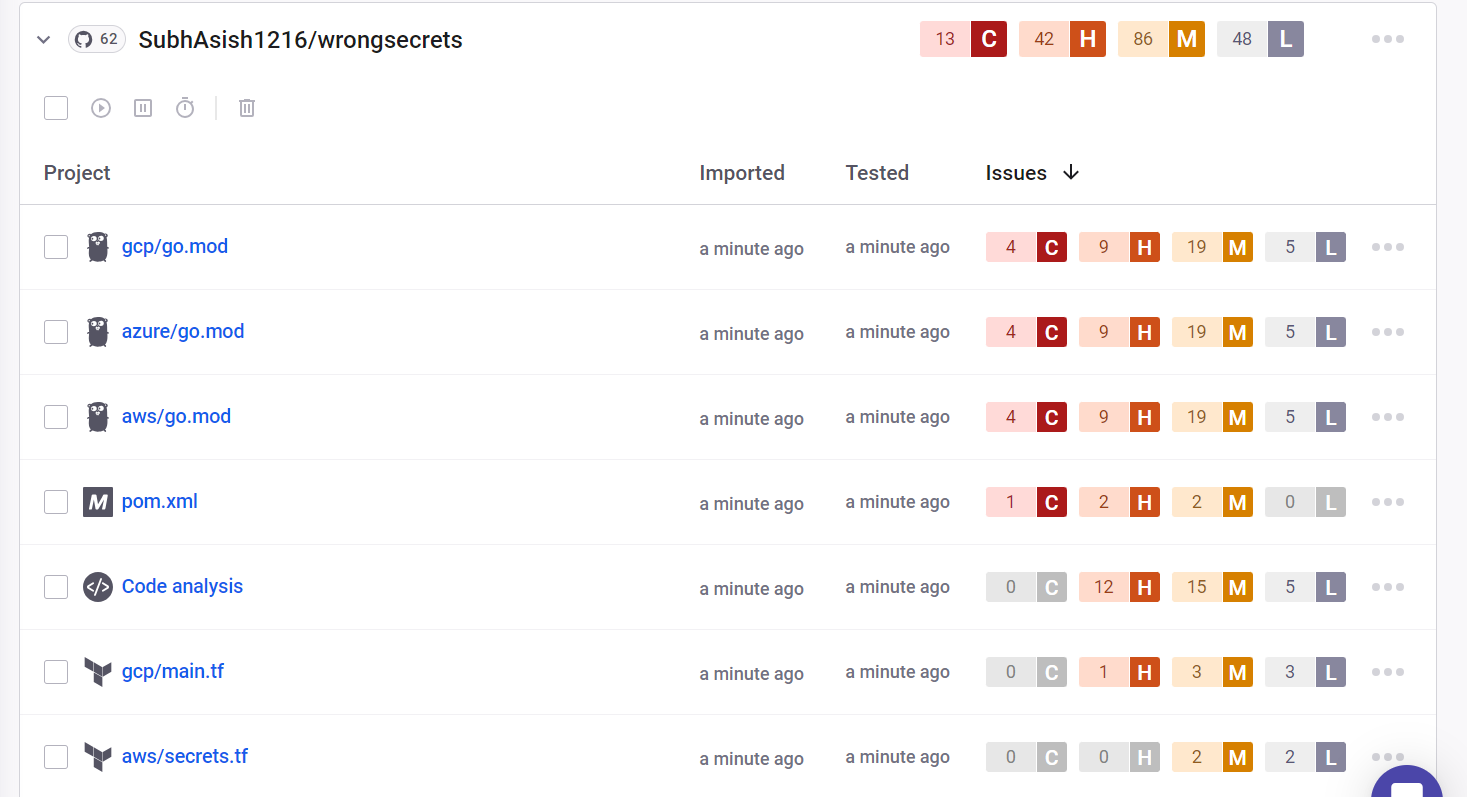
A graph with green and red lines

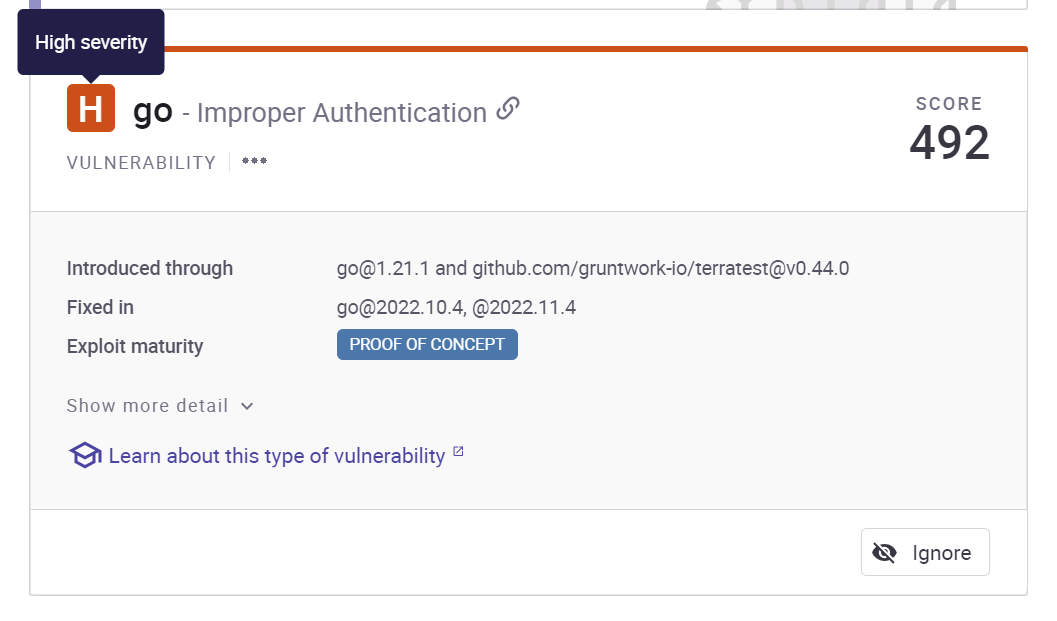
Description automatically generated



After gremlin CPU utilization attack, the CPU utilization has increased to 100%.

**SNYK: Performing SNYK**





**Broken access control:** Access control refers to the permissions structure that should be defined by the application. For example, your application may have separate roles for regular users and administrators. Regular users should not be able to obtain priviledged access, but administrators should!

When the access control of an application is broken, a regular user may be able to access functionality that is meant to be reserved for administrators, or perhaps they can access data that does not belong to them.

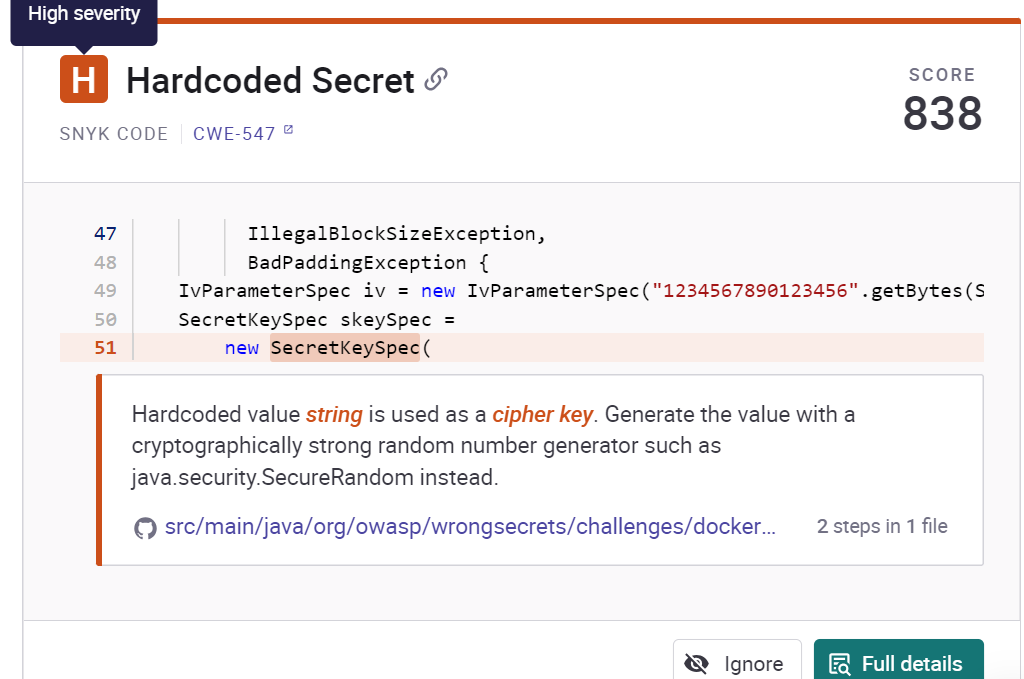
**Solution:** Mitigate broken access control by implementing robust access permission mechanisms. Clearly define and enforce user roles, ensuring that each role has only the necessary privileges. Regularly audit and test access controls to identify and rectify vulnerabilities. Utilize proper session management and authentication mechanisms to verify user identities. Employ access control lists and well-defined policies to restrict user actions and data access, enhancing overall application security.

**The vulnerability in `com.h2database:h2` library exposes a risk of Remote** Code Execution (RCE). Attackers can exploit this flaw to execute arbitrary code on the target system remotely, potentially leading to unauthorized access, data manipulation, or service disruption. Mitigate by updating the H2 database library to the latest version, regularly applying security patches, and adopting secure coding practices to prevent code execution vulnerabilities.

**Solution**: In JavaScript, it's crucial to avoid using eval(), setTimeout(), setInterval(), and the Function constructor, particularly with user input. These functions can introduce security vulnerabilities. Instead, opt for safer alternatives. For instance, replace a vulnerable eval() usage with a direct method, like transforming a string with name.toUpperCase(). This practice prevents potential code injection risks associated with dynamic data passed into functions, ensuring a more secure application against malicious input.

The **vulnerability in `org.apache.commons:commons-compress`** library involves improper input validation. Without thorough validation, attackers can exploit this flaw by providing malicious input, leading to various security issues. To address this, ensure the library is updated to the latest version, regularly apply security patches, and implement strict input validation checks to prevent malicious input, enhancing the overall security of the application.

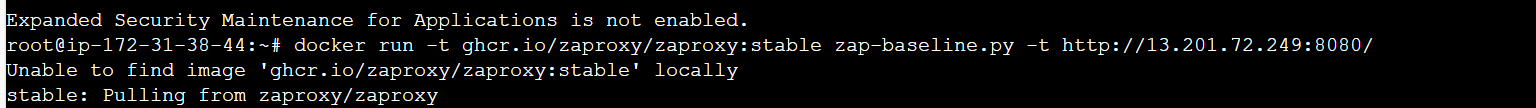
**Solution:** Mitigate the Improper Input Validation vulnerability in `org.apache.commons:commons-compress` by updating the library to the latest version and regularly applying security patches. Implement strict input validation routines to validate and sanitize user inputs, preventing malicious data from causing potential security threats. Regularly monitor and audit the application for vulnerabilities to ensure a robust defense against improper input handling.



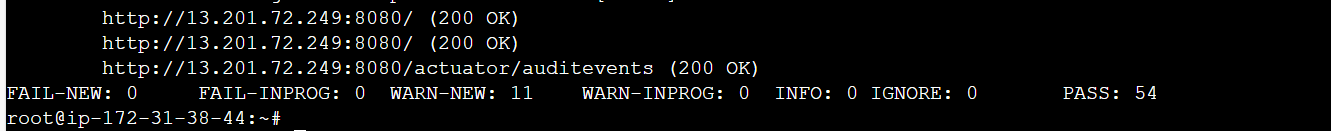
The issue involves the use of a hardcoded string as a cipher key, a critical security flaw. When a static value is employed as a key, it exposes cryptographic systems to predictable attacks. To address this, it's recommended to generate the key using a cryptographically secure random number generator like `java.security.SecureRandom`. This ensures the key's unpredictability, enhancing the strength of encryption and fortifying the system against potential exploits based on hardcoded key values.

**Solution**: Address the vulnerability of using a hardcoded value as a cipher key by generating keys using a robust random number generator like `java.security.SecureRandom`. Replace static keys with dynamically generated, cryptographically strong values to enhance encryption security. Regularly update and rotate keys to minimize risks. Adopt best practices in key management to fortify the system against potential attacks exploiting hardcoded values.

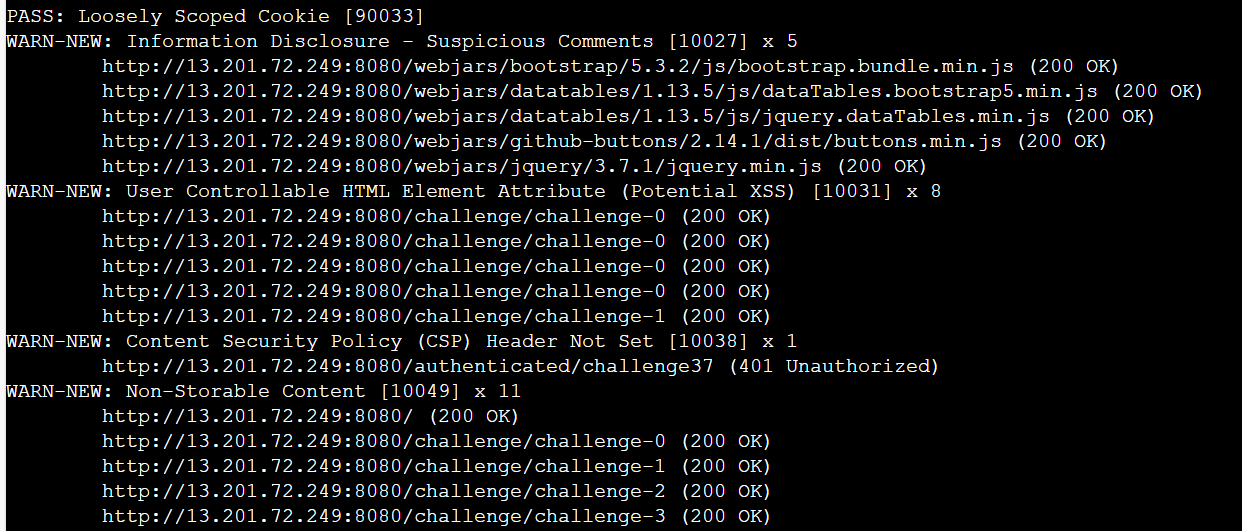
**OWASP ZAP:**



After performing ZAP, getting this result:



Warn new: 11 and pass test: 54.



**Explaining this security issues with solution:**

A screenshot of a computer screen

Description automatically generated

**User Controllable HTML Element Attribute (Potential XSS) [10031] x 8:** This check looks at user-supplied input in query string parameters and POST data to identify where certain HTML attribute values might be controlled. This provides hot-spot detection for XSS (cross-site scripting) that will require further review by a security analyst to determine exploitability.

**Solution**: Validate all input and sanitize output it before writing to any HTML attributes.

**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.

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