|  |  |
| --- | --- |
| Name: | Ronit Gupta |
| Lab User ID: | 23SEK3324\_U12  528646462716 |
| Date: | 10/01/2024 |
| Application Name: | Browser Security - DVGM  https://git.logicalhacking.com/BrowserSecurity/DVGM |

**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)

Damn Vulnerable Grade Management implements a simplistic system for managing university grades. Students can upload assignments (pdf), view their grades for their assignments and lectures, download their grades as reports, and add comments to the grades which can be viewed by lecturers. The application knows three roles: *admins*, *lecturers*, and *students*.

* *Admins* can create new students, lecturers, and other admins. Admins can create new lectures, held by any lecturer. Admins can also create, view, and edit new grades for all lectures and students and can create, view, and edit comments.
* *Lecturers* can create new students. They can also create new lectures that are being held by them. Lecturers can can view grades for all students, but only enter new grades for their own students. Lecturers can see comments for all grades, but can not change any.
* *Students* can upload assignments (pdf). They can also view and comment on their grades for their assignments and overall lectures. For their convenience, they have the ability to filter their grade list by a lecturer name.
* All roles are able to log into the system. They can also reset their password by providing the answer to their chosen security question.

You are Peter, a student and you can log in with peter as username and football as password.

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.")



**Hypothesis**: Users will find value in a mobile task management application with an intuitive user interface.

**Rationale**: Prior market research and user surveys have identified a demand for a task management app. Competitor analysis indicates a need for a more user-friendly solution.

**Hypothesis**: Users will prefer feature X over feature Y in the task management app.

**Rationale**: While initial user feedback has highlighted a preference for simplicity, it's unclear whether users would prioritize a detailed task analytics feature (X) over a more streamlined task input method (Y).

**Known**

Things we are aware of but don’t understand.

Things we are aware of and understand.

**Hypothesis**: Users will discover new use cases and functionalities beyond basic task management.

**Rationale**: As users interact with the app, they might find innovative ways to use features that were initially designed for specific tasks. This may include collaborative features or integrations with other apps.

**Hypothesis**: Unexpected technical challenges may arise during integration with the latest Android OS version.

**Rationale**: While the team is confident in the app's compatibility, unforeseen changes in the Android OS may present challenges that were not anticipated. This could include issues related to new security protocols, device-specific nuances, or changes in user behavior with the latest OS.

**Unknown**

**Unknown**

**Known**

Things we are neither aware of nor understand.

Things we understand but are not aware of.

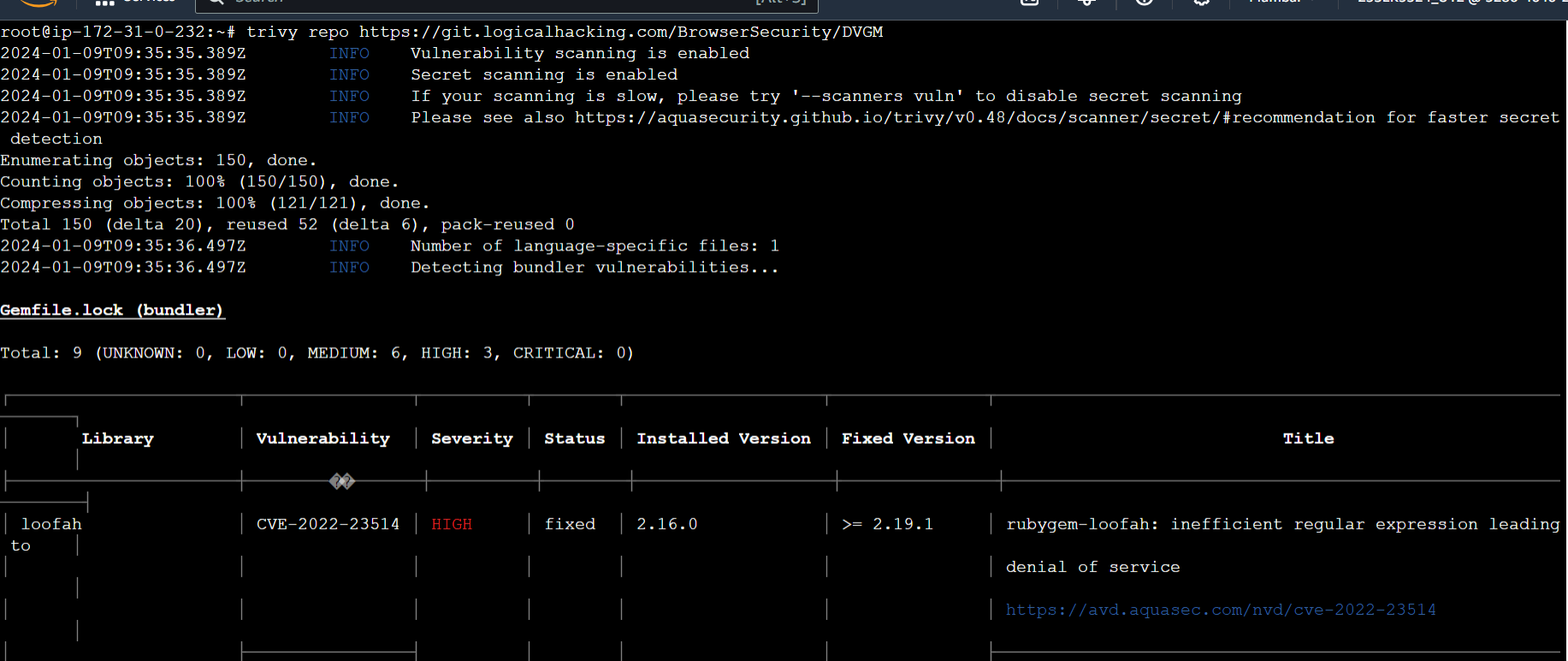
**Vulnerability analysis using trivy:**

root@ip-172-31-0-232:~# trivy repo https://git.logicalhacking.com/BrowserSecurity/DVGM

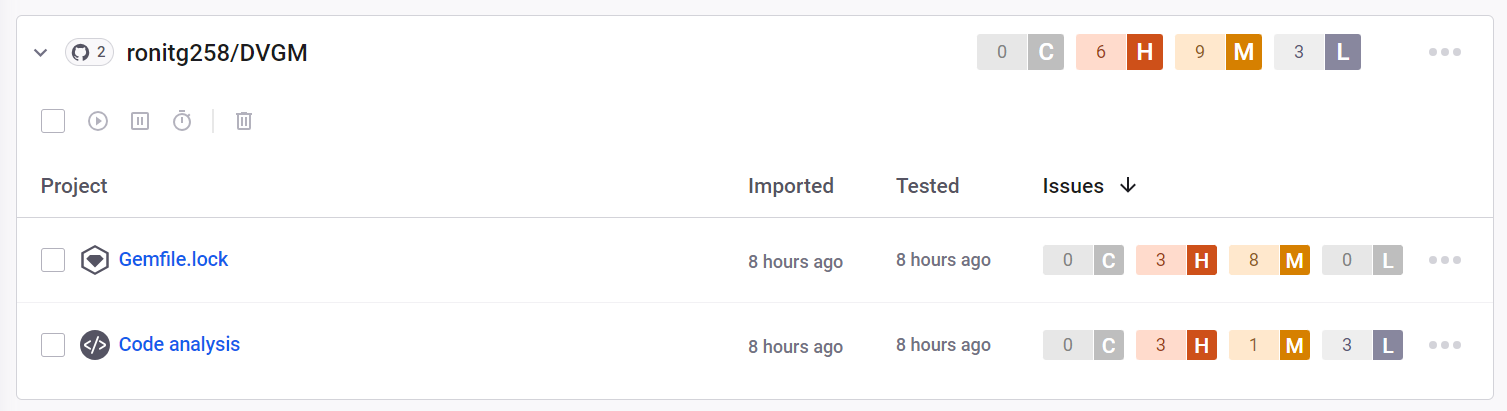
-----

Gemfile.lock (bundler)

Total: 9 (UNKNOWN: 0, LOW: 0, MEDIUM: 6, HIGH: 3, CRITICAL: 0)



**Vulnerability analysis using snyk:**



**Some critical vulnerability analysis:**

1. **SQL Injection:** SQL injection is a type of cyber-attack where an attacker inserts malicious SQL code into input fields or parameters in a web application, aiming to manipulate the application's SQL query.

**Mitigation Techniques:**

\* Use parameterized queries or prepared statements in your code. This ensures that user input is treated as data, not executable code.

\* Validate and sanitize user inputs on the server-side. Ensure that the input adheres to expected formats and reject any input that looks suspicious.

2. **Use of Password Hash with Insufficient Computational Effort:** This weakness makes it easier for attackers to crack passwords using brute-force attacks, dictionary attacks, or other methods that exploit the low computational cost of the hashing algorithm.

**Mitigation Techniques:**

\* Use unique salts for each password before hashing. Salting adds randomness to each password, even if they are the same, resulting in different hash values. This prevents attackers from using precomputed tables (rainbow tables) for common passwords.

\* Enforce strong password policies that encourage users to choose complex and unique passwords. This makes it more challenging for attackers to guess or crack passwords.

3**. Improperly Controlled Modification of Dynamically Determined Object Attributes:** The "Improperly Controlled Modification of Dynamically-Determined Object Attributes" is a security vulnerability where an attacker can manipulate or modify object attributes dynamically, leading to unintended behavior in an application. This vulnerability often arises when user inputs are used to determine object attributes, and insufficient validation or controls are in place.

**Mitigation Techniques:**

\* Implement whitelisting for dynamically determined object attributes. Define a set of acceptable values, and only allow inputs that match these predefined values.

\* Limit the set of properties that can be dynamically modified. Only expose those properties that are necessary and safe for modification based on the application's logic.

**4. Security Misconfigurations**: Insecure default settings, open ports, unnecessary services, or excessive permissions can lead to unauthorized access or exposure of sensitive information.

**Mitigation Techniques:**

\* Regularly audit and review security configurations, including default settings.

\* Employ tools for automated scanning and identify and remediate misconfigurations.

\* Follow the principle of least privilege for user accounts and services.

Experiment:

(Document your Preparation, Implementation, Observation and Analysis )

**Steps:**

1. We need to install the required dependencies that are Ruby 3.1 (and Raild 7) and bundler.  
\curl -sSL https://get.rvm.io | bash -s stable

source ~/.rvm/scripts/rvm

rvm requirements

rvm install 3.1

rvm use 3.1 –default

ruby -v

2. Clone the repository using **git clone https://git.logicalhacking.com/BrowserSecurity/DVGM.git**

3. After cloning the repository, install the dependencies like bundle

4.cd DVGM

bundle install --path vendor/bundle

5. Now, start the server:

bin/rails server

6. Now, open your browser, go to http://localhost:3000, and start exploring!

**Output Snapshots:**

\* Output was not live on browser due to some unknown error or issue so took a snapshot by executing curl command on the local.  
**curl http://localhost:3000**

