INFORMATION SECURITY

ASSIGNMENT # 3

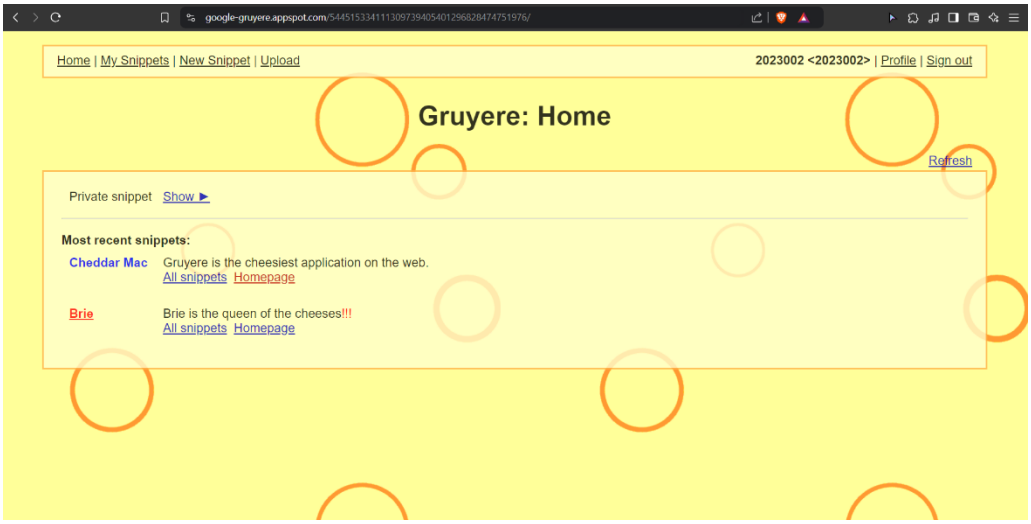
Web Application Vulnerability Analysis and Prevention Report

- **Name**: AAYAN RASHID (2023002)
- **Course**: CYS211
- **INSTRUCTOR:** DR. M. ZAIN SIDDIQI
- **Date**: 12/8/24

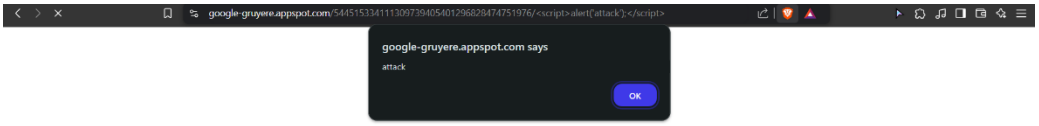
Table of Contents

Vulnerability 1: Reflected Cross-Site Scripting (XSS)

1. **Explanation**: Allows attackers to inject malicious scripts into web pages viewed by other users.
2. **URL of the Vulnerable Web Page**: https://google-gruyere.appspot.com/544515334111309739405401296828474751976/%3Cscript%3Ealert('attack');%3C/script%3E
3. **Screenshot of the Web App Before Exploiting the Vulnerability**:



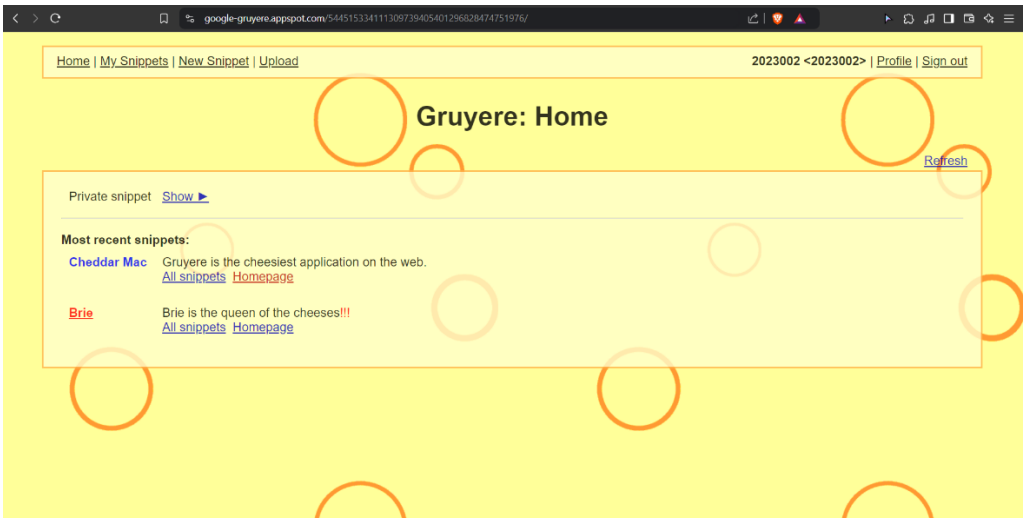
4. **Screenshot of the Web App Page After Exploiting the Vulnerability**:



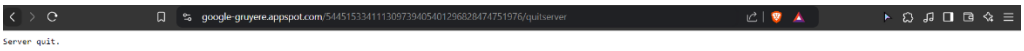
5. **Observation**: The malicious script executes within the context of the user's browser.
6. **Prevention Method**: Implement input validation and output encoding to prevent script injection.

Vulnerability 2: Denial of Service (DoS) Attack

1. **Explanation**: Overwhelms the web application with excessive requests, making it unavailable to users.
2. **URL of the Vulnerable Web Page**: https://google-gruyere.appspot.com/544515334111309739405401296828474751976/quitserver
3. **Screenshot of the Web App Before Exploiting the Vulnerability**:



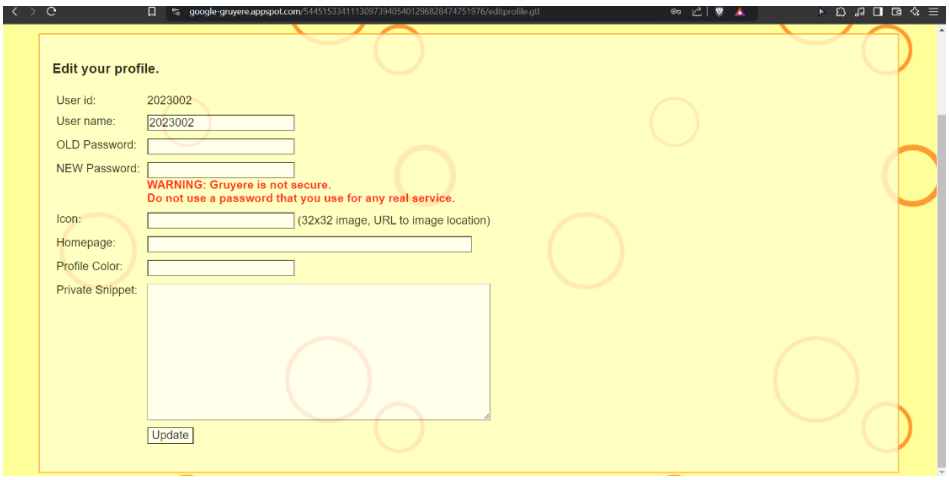
4. **Screenshot of the Web App Page After Exploiting the Vulnerability**:



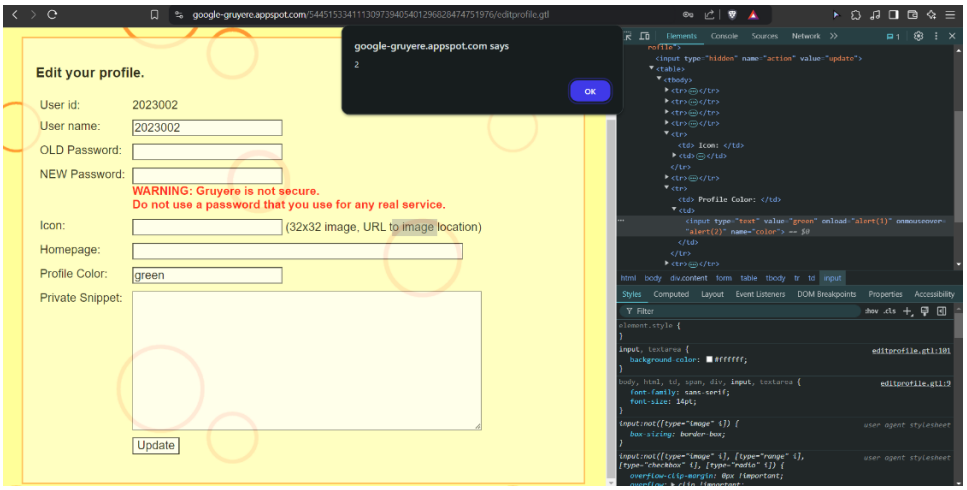
5. **Observation**: The application becomes unresponsive due to high traffic.
6. **Prevention Method**: Implement rate limiting and server-side input validation to mitigate DoS attacks.

Vulnerability 3: Cross-Site Scripting (XSS) via HTML attribute

1. **Explanation**: Allows an attacker to execute arbitrary JavaScript code in the context of a user's browser.
2. **URL of the Vulnerable Web Page**: https://google-gruyere.appspot.com/544515334111309739405401296828474751976/editprofile.gtl
3. **Screenshot of the Web App Before Exploiting the Vulnerability**:



4. **Screenshot of the Web App Page After Exploiting the Vulnerability**:

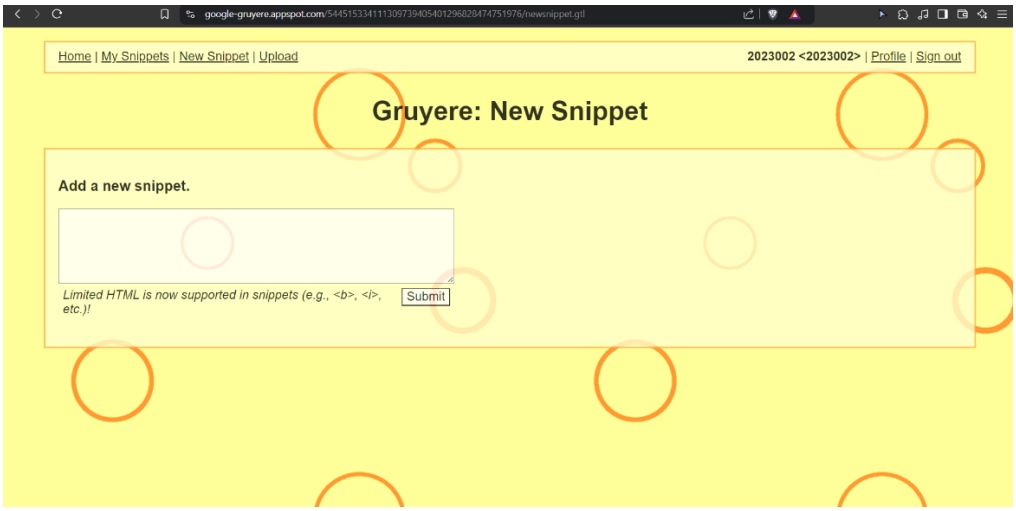


5. **Observation**: The script executes when the user interacts with the input field, indicating an XSS vulnerability.
6. **Prevention Method**: Validate and sanitize inputs, encode outputs, and implement a Content Security Policy (CSP) to restrict the sources from which scripts can be loaded.

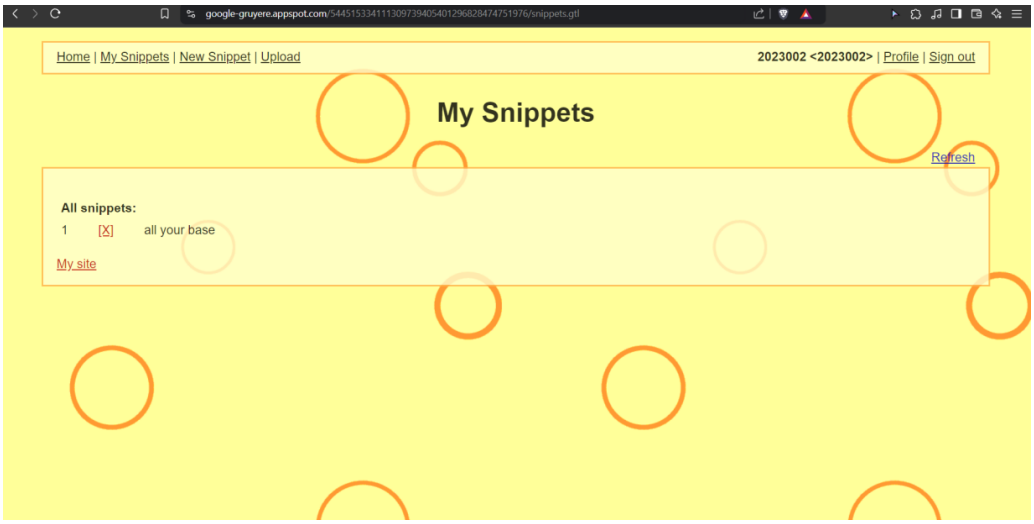
Vulnerability 4: Cross-Site Scripting (XSS) via AJAX

Explanation: Allows an attacker to execute arbitrary JavaScript code by injecting a snippet into the application.

1. **URL of the Vulnerable Web Page**: https://google-gruyere.appspot.com/544515334111309739405401296828474751976/snippets.gtl
2. **Screenshot of the Web App Before Exploiting the Vulnerability**:



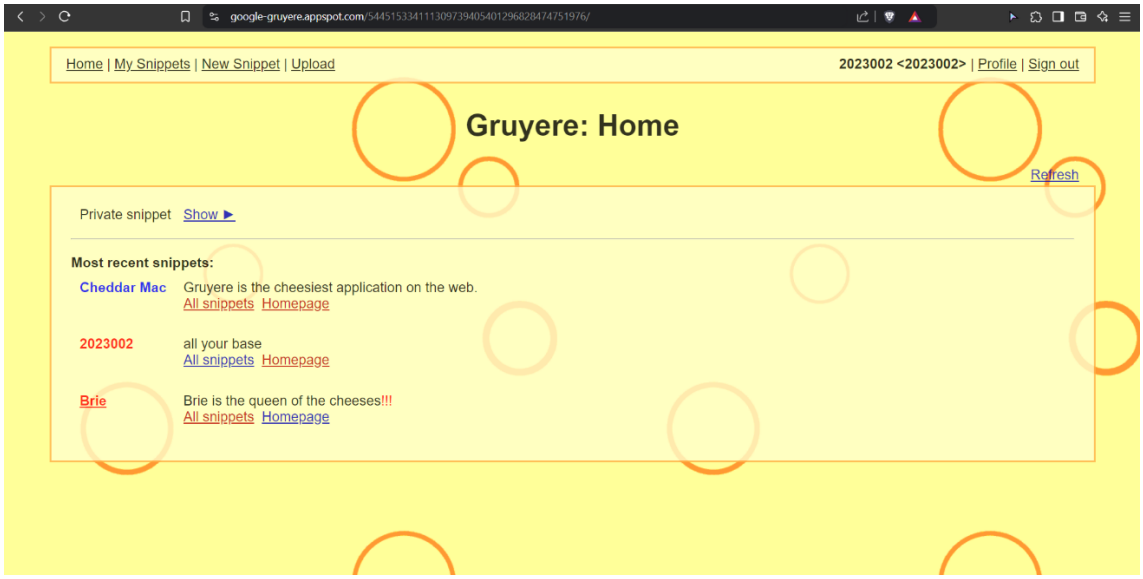
3. **Screenshot of the Web App Page After Exploiting the Vulnerability**:



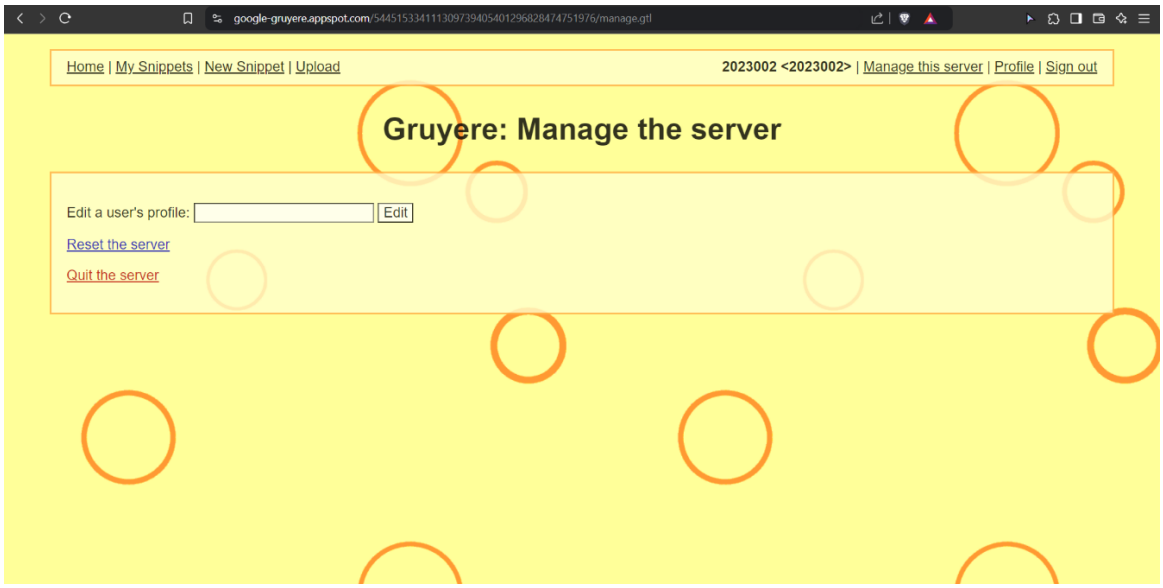
4. **Observation**: The script executes when the snippet is processed, indicating an XSS vulnerability.
5. **Prevention Method**: Ensure that all text has correctly escaped when rendered in the JSON response, avoid using eval for JSON parsing, and implement proper sanitation and encoding techniques.

Vulnerability 5: Elevation of Privilege (EoP) via Unauthorized Account Modification

1. **Explanation**: Allows an attacker to convert a regular user account into an administrator account without proper authorization.
2. **URL of the Vulnerable Web Page**: http://google-gruyere.appspot.com/544515334111309739405401296828474751976saveprofile?action=update&is_admin=True
3. **Screenshot of the Web App Before Exploiting the Vulnerability**:



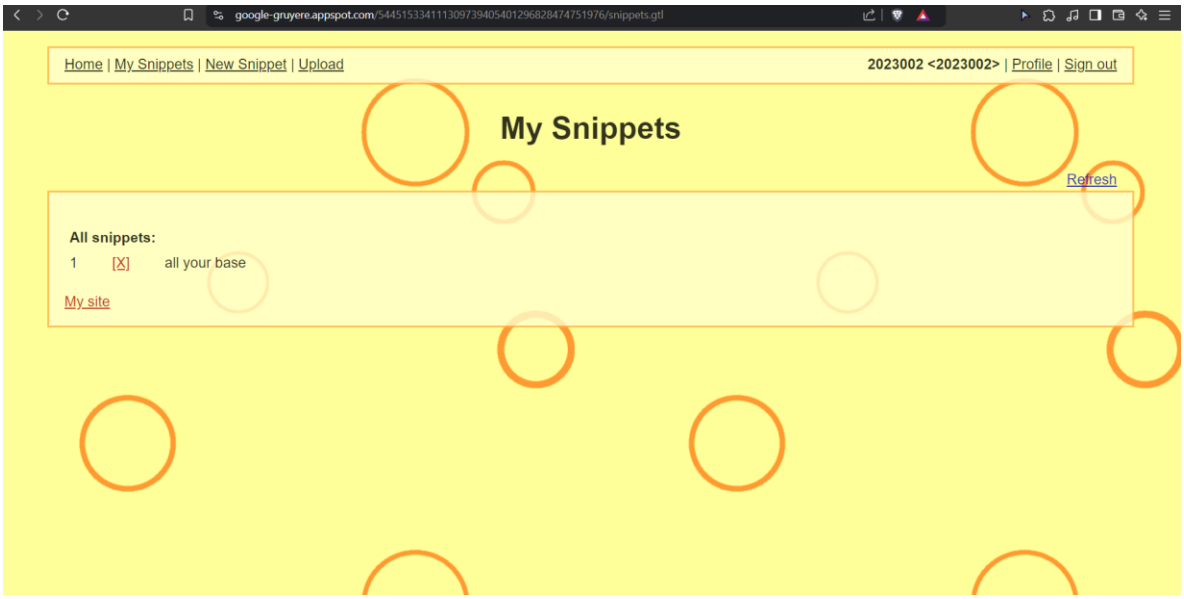
4. **Screenshot of the Web App Page After Exploiting the Vulnerability**:



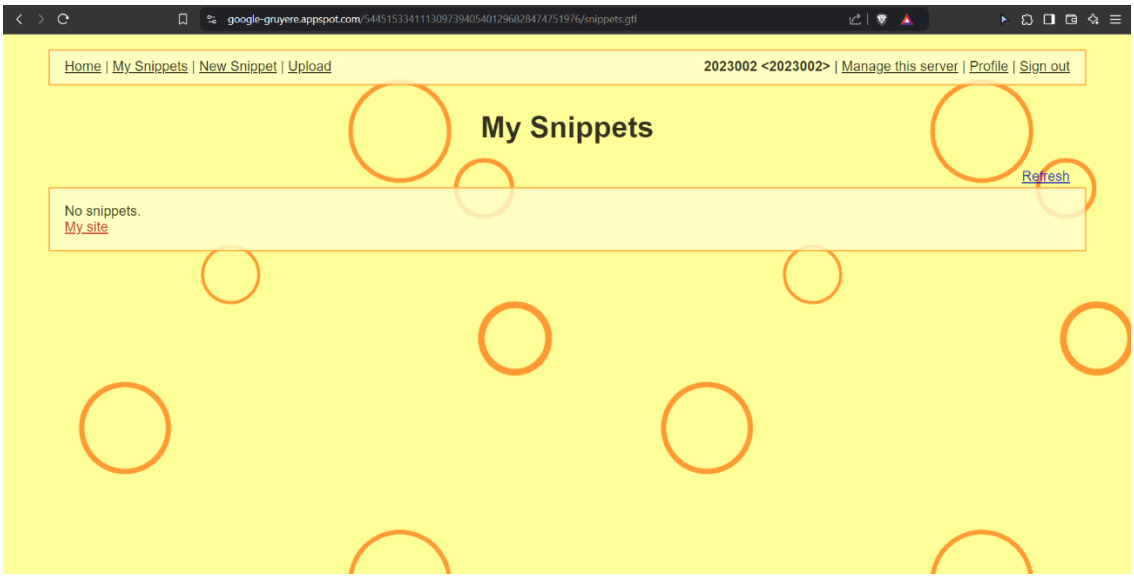
5. **Observation**: After sending the unauthorized request, the user account is marked as an administrator, indicated by the presence of the "Manage this server" link upon re-login.
6. **Prevention Method**: Implement proper server-side validation to ensure that only authorized users can modify account roles. Avoid relying solely on client-side checks.

Vulnerability 6: Cross-Site Request Forgery (CSRF) via Snippet Deletion

1. **Explanation**: Allows an attacker to perform state-changing actions on behalf of a logged-in user without their knowledge.
2. **URL of the Vulnerable Web Page**: http://google-gruyere.appspot.com/544515334111309739405401296828474751976/deletesnippet?index=0
3. **Screenshot of the Web App Before Exploiting the Vulnerability**:



4. **Screenshot of the Web App Page After Exploiting the Vulnerability**:

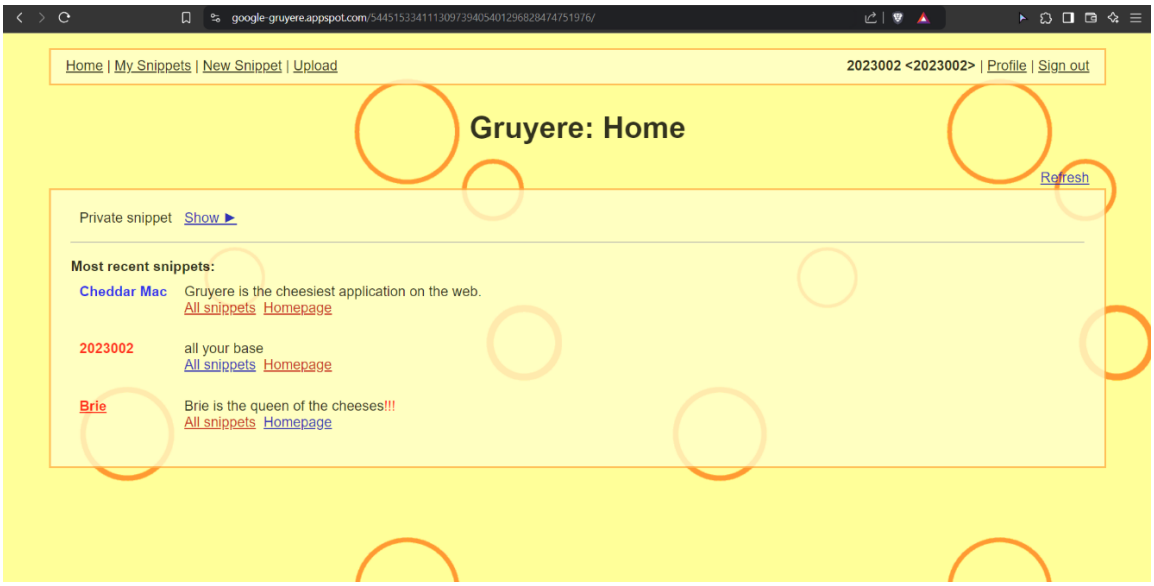


5. **Observation**: The snippet is deleted without the user's consent, demonstrating a CSRF vulnerability.
6. **Prevention Method**: Change the deletesnippet request to use the POST method, implement CSRF tokens, and avoid using eval to parse JSON.

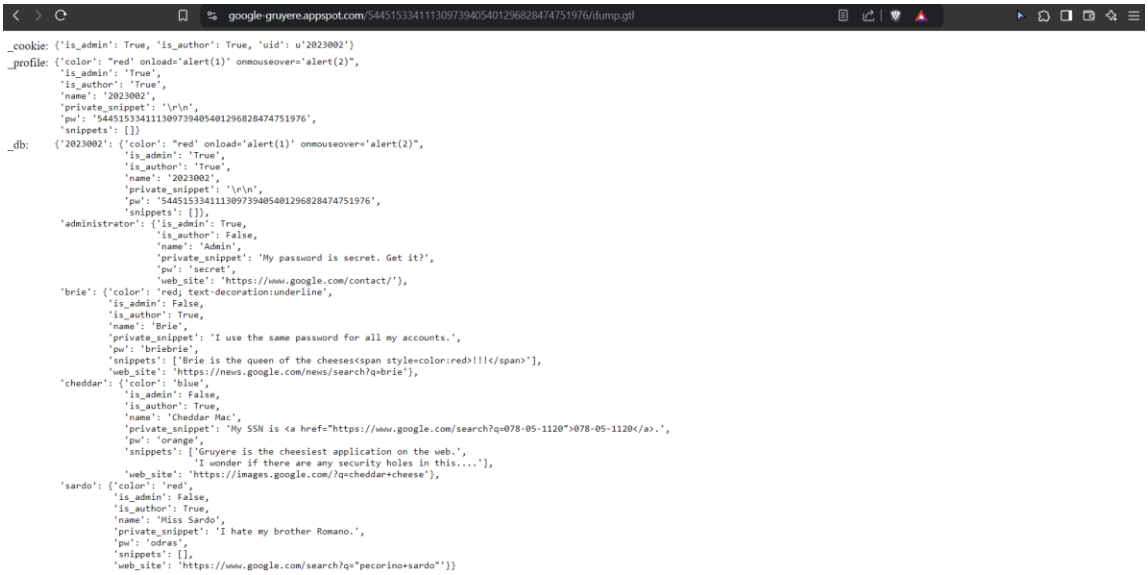
Vulnerability 7: Information Disclosure via Debug Dump Page

1. **Explanation**: Allows an attacker to read the contents of the database, exposing sensitive information such as user passwords.
2. **URL of the Vulnerable Web Page**: https://google-gruyere.appspot.com/544515334111309739405401296828474751976/dump.gtl

3. **Screenshot of the Web App Before Exploiting the Vulnerability**:



4. **Screenshot of the Web App Page After Exploiting the Vulnerability**:

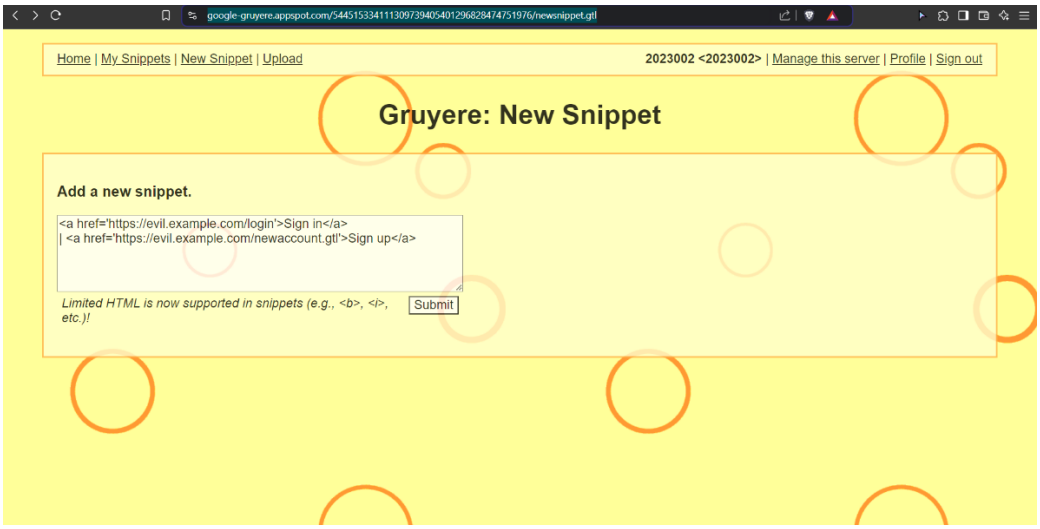


5. **Observation**: The database contents, including user passwords in cleartext, are displayed, demonstrating an information disclosure vulnerability.

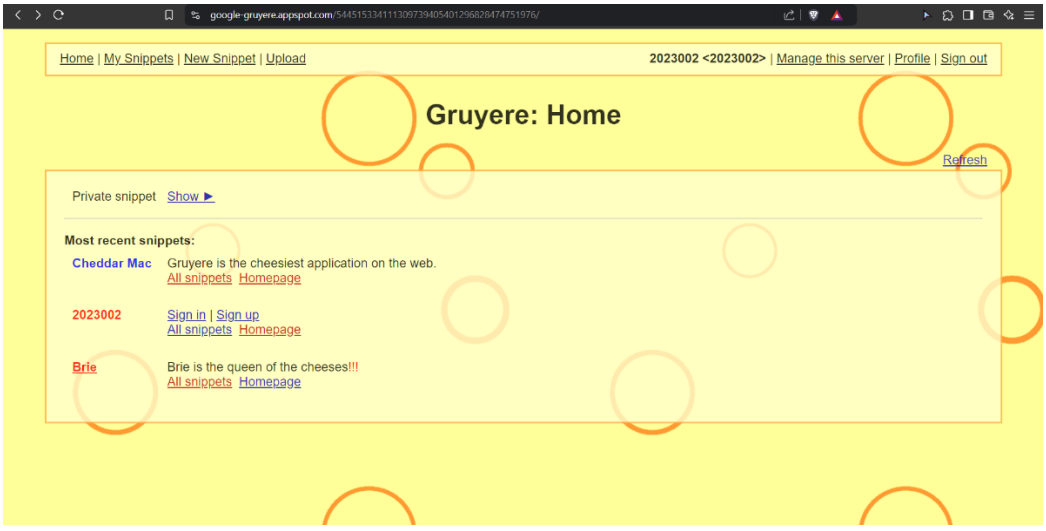
6. **Prevention Method**: Remove debug features like dump.gtl from production environments. Implement password hashing to securely store user passwords

Vulnerability 8: User Interface Spoofing via Menu Bar Snippet

1. **Explanation:** Allows an attacker to trick users into clicking malicious links by spoofing the legitimate menu bar.
2. **URL of the Vulnerable Web Page: https://google-gruyere.appspot.com/5445153341113097394054012968284747519 76/#**
3. **Screenshot of the Web App Before Exploiting the Vulnerability:**



4. **Screenshot of the Web App Page After Exploiting the Vulnerability:**



5. **Observation: The spoofed menu bar contains malicious links, demonstrating a user interface spoofing vulnerability.**
6. **Prevention Method: Ensure that user-generated content cannot conflict with or override existing DOM elements. Implement secure element identifiers and avoid using user values as identifiers.**

Conclusion:

This report provides a thorough analysis of multiple vulnerabilities identified in the Gruyere web application. Each vulnerability represents significant security risks that attackers could exploit to gain unauthorized access, disrupt services, or compromise sensitive information. The report outlines detailed exploitation steps for each vulnerability and offers actionable recommendations to mitigate these risks.

Key Takeaways:

- Input Validation and Output Encoding: Critical for preventing cross-site scripting (XSS) attacks.
- Rate Limiting and Input Validation: Essential to mitigate denial-of-service (DoS) attacks and ensure system stability.
- Proper Server-Side Validation: Necessary to prevent unauthorized account modifications and enforce secure access control mechanisms.
- Implementation of CSRF Tokens: Vital for defending against cross-site request forgery (CSRF) attacks.
- Secure Password Storage: Employing cryptographic hashing for password storage enhances protection against information disclosure.
- Avoiding the Use of eval: Ensures safer handling of JSON and other inputs, reducing the risk of code injection vulnerabilities.
- Secure Element Identifiers: Prevents user interface spoofing and ensures the integrity of UI components.