CS-375 CYBERSECURITY

ASSIGNMENT # 3

Ghulam Ishaq Khan Institute of Engineering Sciences & Technology, Topi, Swabi 23460, Pakistan

WEB APPLICATION VULNERABILITY ANALYSIS AND PREVENTION

By 2021083 (ALI NAUFIL)

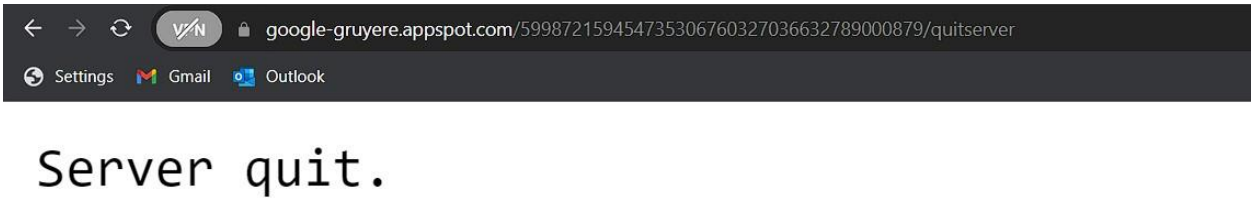
Vulnerability Name: DOS attack (server quitting)

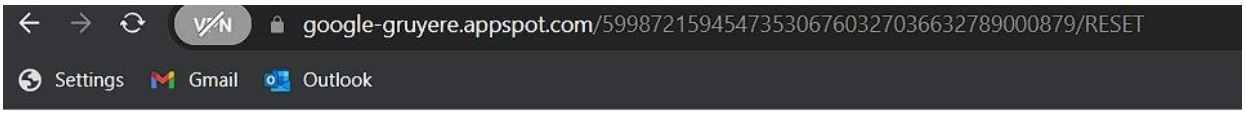
Explanation:
Denial of service attack to quit the server involves exploiting vulnerabilities in gruyere like path traversal or requesting "/quitserver" in the URL, causing the server to become unresponsive or crash.

URL of vulnerable webpage:
https://google-gruyere.appspot.com/599872159454735306760327036632789000879/quitserver
https://google-gruyere.appspot.com/599872159454735306760327036632789000879/RESET

Screenshot before vulnerability:

Screenshot after vulnerability:





Server reset to default values...

Observation:
We need to be logged in as an administrator for quitting server but here a non-administrator can also quit server and the case-sensitive check for protected URLs leads to a classic bug mismatch.

Prevention method:
We can prevent this by adding */quitserver* and */RESET* to the URLS which are only accessible to the administrators. We can also put security checks inside dangerous functions rather than outside them.

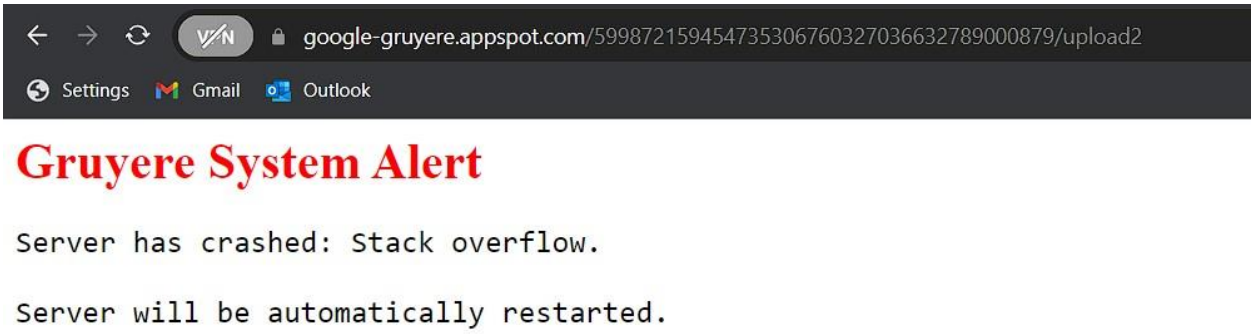
Vulnerability Name: DOS attack (server overloading)

Explanation:
A denial of service (DoS) attack to *overload the server* involves exploiting vulnerabilities like path traversal to create an infinite loop, causing the server to be overwhelmed with requests.

URL of vulnerable webpage:
https://google-gruyere.appspot.com/5998721594547353067603270366327890008 79/../resorces/menubar.gtl

Screenshot before vulnerability:

Screenshot after vulnerability:



Observation:
In gruyere every page includes the menubar.gtl template. We can exploit this by creating a file named menubar.gtl and upload it to *resources* directory using a path traversal attack i.e. Creating a user named *../resources.*

Prevention method:
We can prevent this by implementing some specific protections against path traversal.

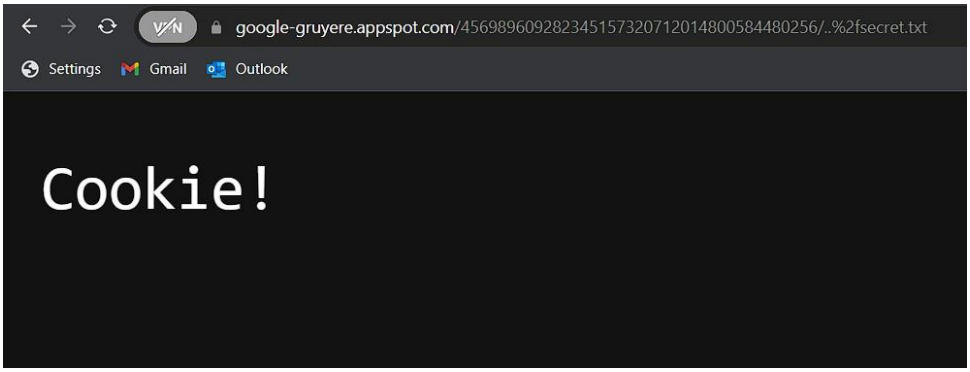
Vulnerability Name: Path traversal

Explanation:
A path traversal vulnerability allows attackers to access restricted directories by using "**../**" characters to navigate to files outside the intended directory structure, exposing sensitive information or executing unauthorized actions.

URL of vulnerable webpage:
https://google-gruyere.appspot.com/456989609282345157320712014800584480256/..%2fsecret.txt

Screenshot before vulnerability:

Screenshot after vulnerability:



Observation:
As **..** represents parent directory, so if we can inject **../** in a path *we can escape to the parent directory*. Similarly, an attacker can craft a URL that will traverse out of the installation directory to */etc*.

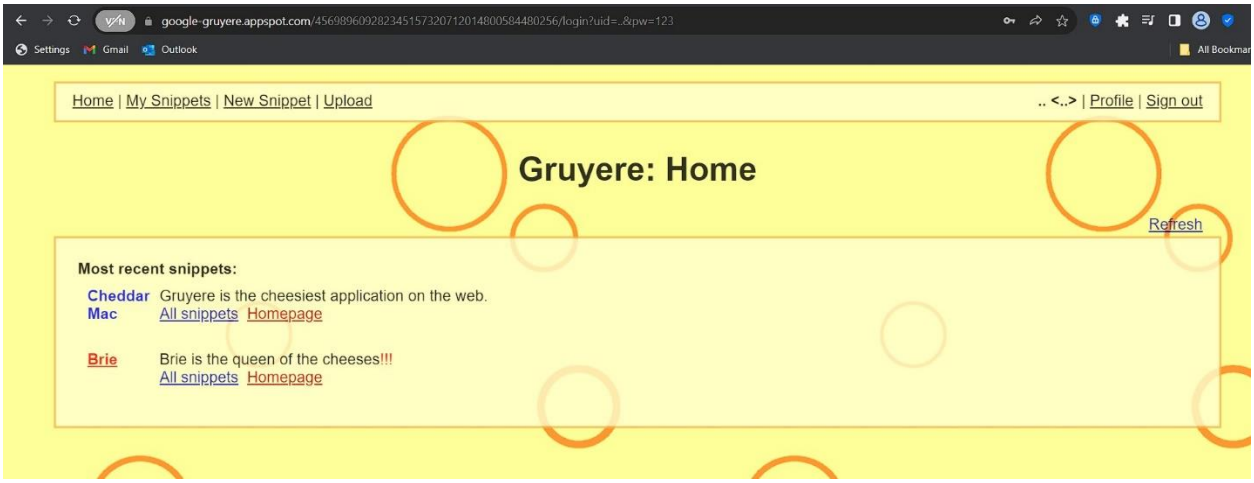
Prevention method:
We can prevent path traversal in gruyere by avoiding user-controlled file access and implementing strict input validation to restrict unauthorized directory access and file uploads.

Vulnerability Name: Code Execution

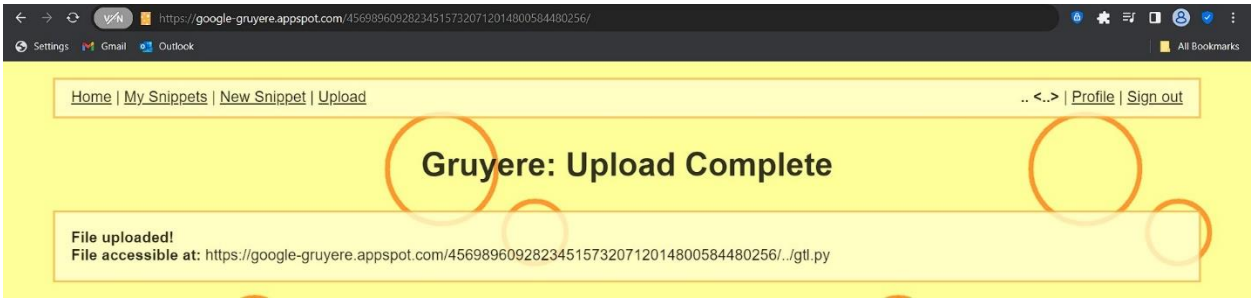
Explanation:
A code execution vulnerability allows attackers to execute arbitrary code on the server, potentially leading to unauthorized access, data breaches, and system compromise.

URL of vulnerable webpage:
https://google-gruyere.appspot.com/456989609282345157320712014800584480256/../gtl.py
https://google-gruyere.appspot.com/456989609282345157320712014800584480256/quitserver

Screenshot before vulnerability:



Screenshot after vulnerability:



Server quit.

Gruyere System Alert

```
Server is restarting
... please wait ...

Server has been 0wnd!
```

Observation:
To exploit vulnerability, we created a user .. and uploaded gtl.py. Then we quit the server. When the server restarts, our code runs, and attack was successful because gruyere has permission to both read and write files.

Prevention method:
Always run your application with the *least privileges* it needs. Avoid passing *user input directly into commands* that evaluate arbitrary code. Implement *proper bounds checks* for non-safe languages.

Vulnerability Name: Configuration vulnerabilities

Explanation:
A configuration vulnerability arises from default settings or debug features left enabled, increasing the attack surface, and potentially leading to unauthorized access or data exposure.

URL of vulnerable webpage:
https://google-gruyere.appspot.com/456989609282345157320712014800584480256/dump.gtl

Screenshot before vulnerability:

Screenshot after vulnerability:


```
_cookie: {'is_admin': False, 'is_author': True, 'uid': u'..'}
_profile: {'is_author': 'True', 'name': '..', 'pw': '123'}
_db:    {'..': {'is_author': 'True', 'name': '..', 'pw': '123'},
         '2021083': {'is_author': 'True', 'name': '2021083', 'pw': '123'},
         'administrator': {'is_admin': True,
                           'is_author': False,
                           'name': 'Admin',
                           'private_snippet': 'My password is secret. Get it?',
                           'pw': 'secret',
                           'web_site': 'https://www.google.com/contact/'},
         'brie': {'color': 'red; text-decoration:underline',
                  'is_admin': False,
                  'is_author': True,
                  'name': 'Brie',
                  'private_snippet': 'I use the same password for all my accounts.',
                  'pw': 'briebrie',
                  'snippets': ['Brie is the queen of the cheeses<span style=color:red>!!!</span>'],
                  'web_site': 'https://news.google.com/news/search?q=brie'},
         'brie/../..': {'is_author': 'True', 'name': 'brie/../..', 'pw': '12345'},
         'cheddar': {'color': 'blue',
                     'is_admin': False,
                     'is_author': True,
                     'name': 'Cheddar Mac',
                     'private_snippet': 'My SSN is <a href="https://www.google.com/search?q=078-05-1120">078-05-1120</a>.'
                     'pw': 'orange',
                     'snippets': ['Gruyere is the cheesiest application on the web.',
                                  'I wonder if there are any security holes in this....'],
                     'web_site': 'https://images.google.com/?q=cheddar+cheese'},
         'sardo': {'color': 'red',
                   'is_admin': False,
                   'is_author': True,
                   'name': 'Miss Sardo',
                   'private_snippet': 'I hate my brother Romano.',
                   'pw': 'odras',
                   'snippets': [],
                   'web_site': 'https://www.google.com/search?q="pecorino+sardo"'}}
```


Observation:

To exploit vulnerability, we used the debug dump page *dump.gtl* to display the contents of the database and this exposes the users' passwords, name, id and other details.

Prevention method:

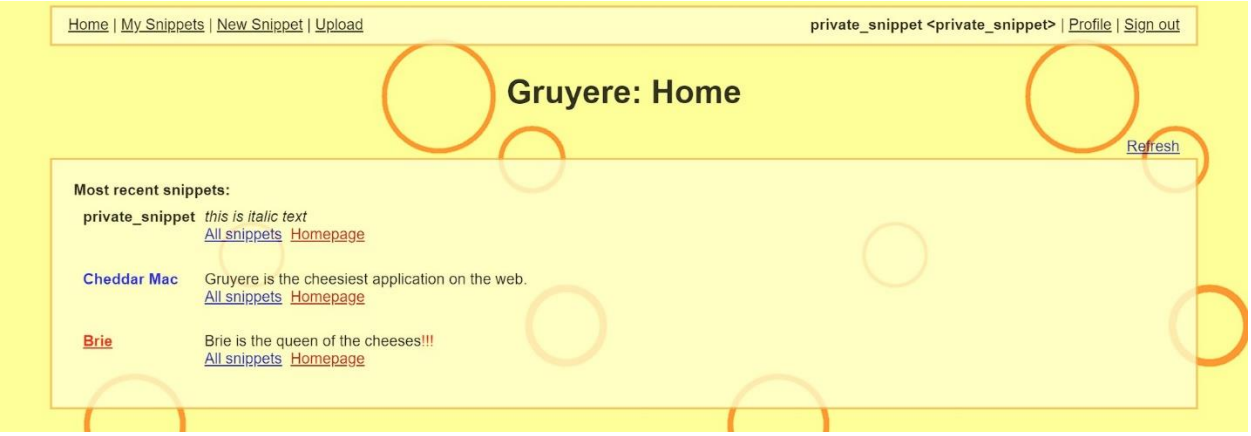
Always make sure debug features are not installed. Passwords should never be stored in cleartext. Instead use *password hashing*. Don't store user uploaded files in the same place as application files.

Vulnerability Name: AJAX vulnerabilities

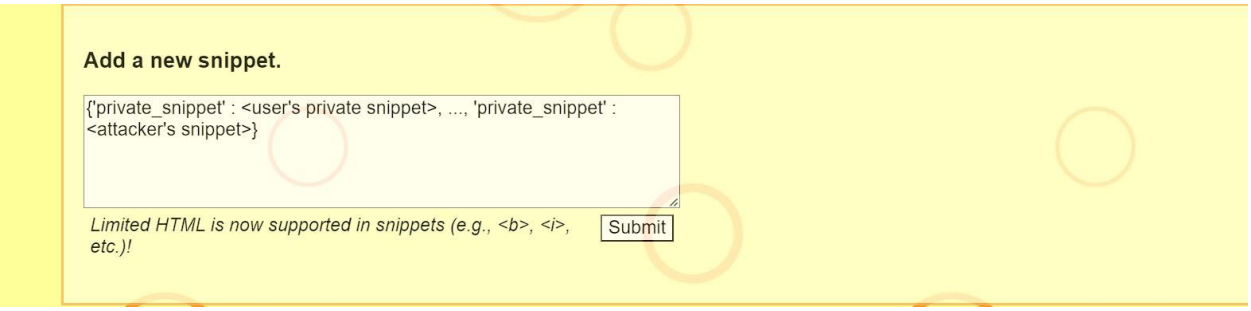
Explanation:
AJAX vulnerabilities can allow attackers to manipulate parts of the application unexpectedly, potentially leading to unauthorized modifications.

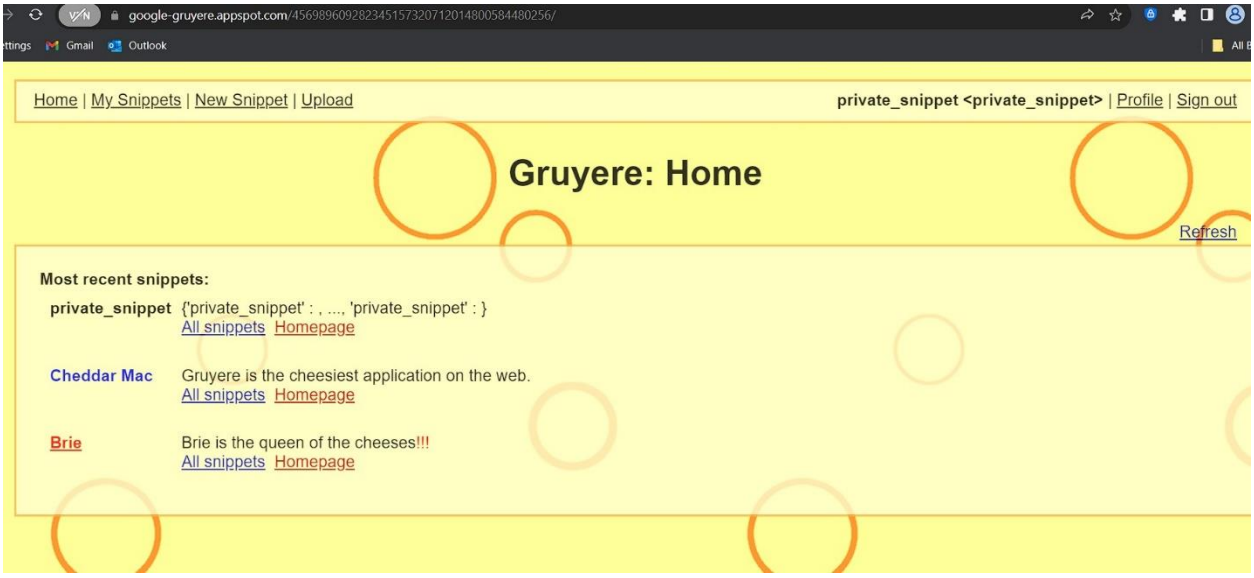
URL of vulnerable webpage:
https://google-gruyere.appspot.com/456989609282345157320712014800584480256/snippets.gtl
https://google-gruyere.appspot.com/456989609282345157320712014800584480256/

Screenshot before vulnerability:



Screenshot after vulnerability:





Observation:
We exploited vulnerability by creating a user named ***private_snippet*** and created an italic html snippet for user. Then we created a ***Json snippet*** which replaced user's snippet and hided the user snippet.

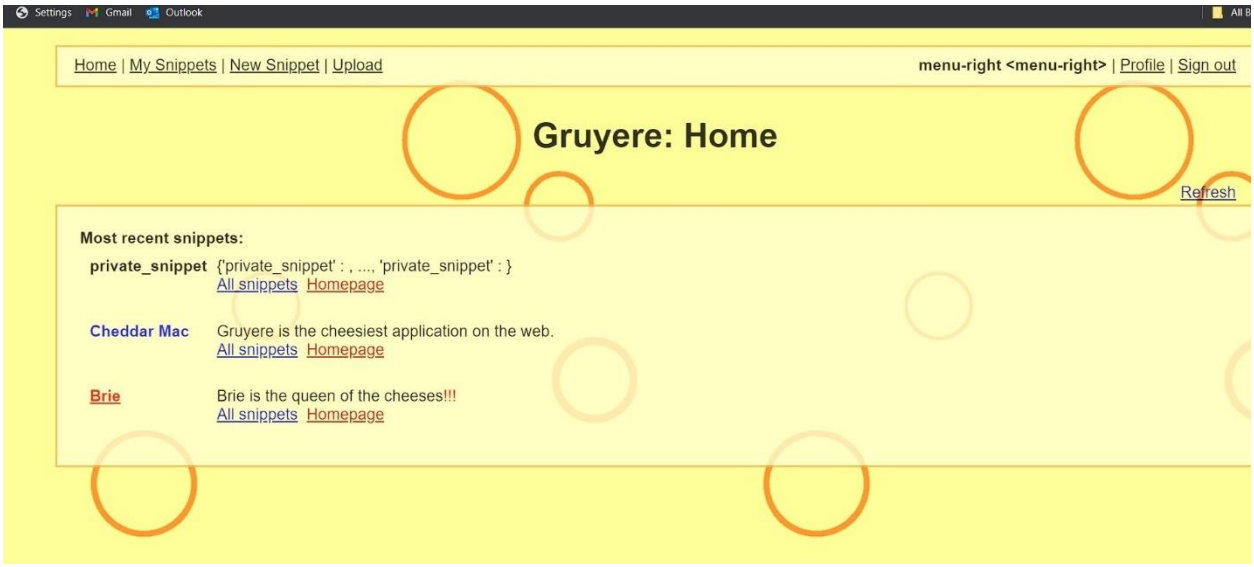
Prevention method:
The AJAX code needs to make sure that the data only goes where it's supposed to go. The flaw here is that the JSON structure is not robust.

Vulnerability Name: Phishing attack by ajax vulnerability.

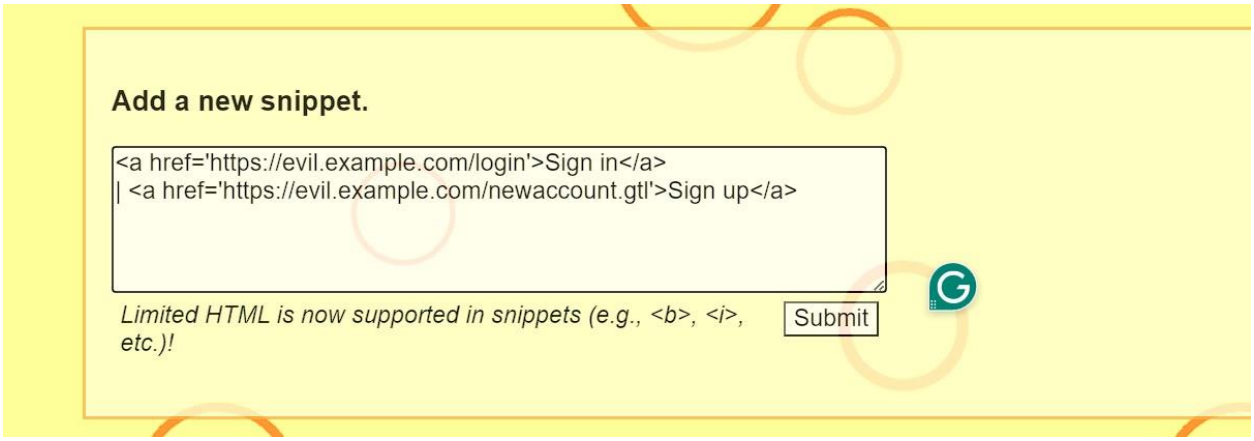
Explanation:
A phishing attack via AJAX vulnerability involves injecting code to manipulate the Document Object Model and create deceptive links that lead users to a phishing site, exploiting user trust.

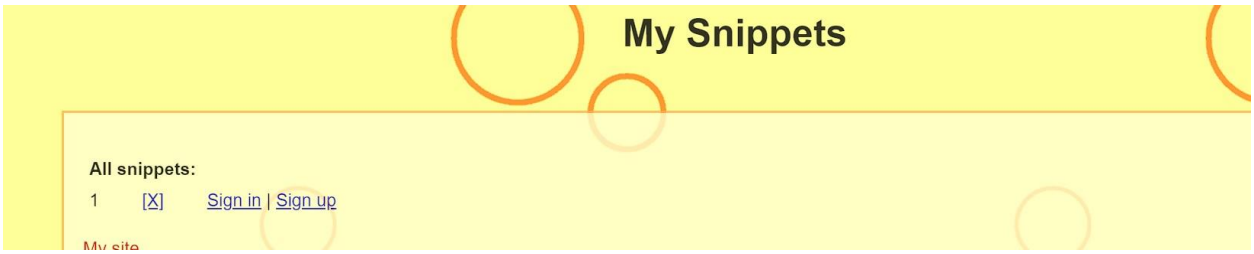
URL of vulnerable webpage:
https://google-gruyere.appspot.com/456989609282345157320712014800584480256/snippets.gtl

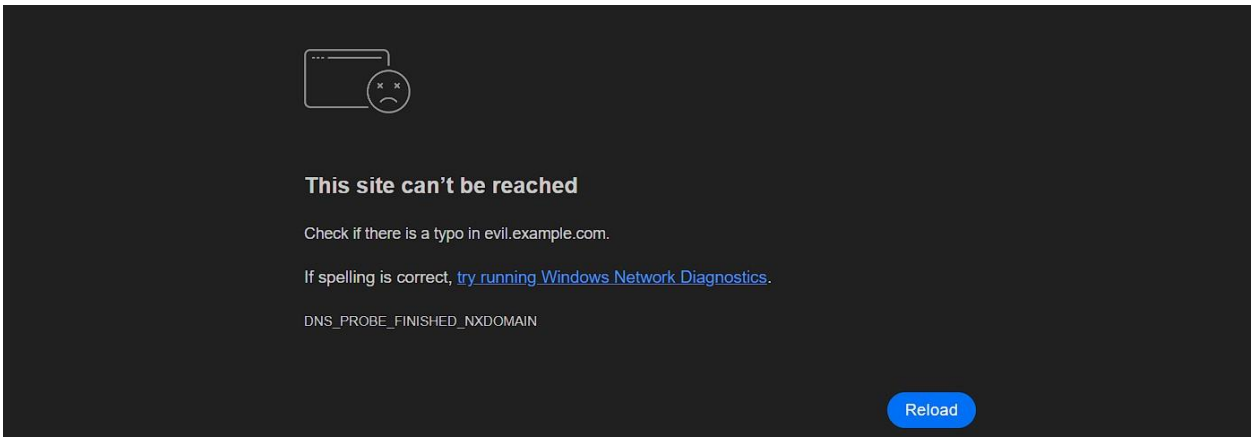
Screenshot before vulnerability:



Screenshot after vulnerability:







Observation:
To exploit vulnerability, we created a user named ***menu-right*** and published a snippet that looks exactly like the right side of the menu bar. When the user clicks on the button he is redirected to another site.

Prevention method:
The process of modifying DOM needs to be made more robust. The user values used in DOM element identifiers must be checked properly and we should use our own identifiers rather than user values.

Vulnerability Name: Elevation of privilege (Client state manipulation)

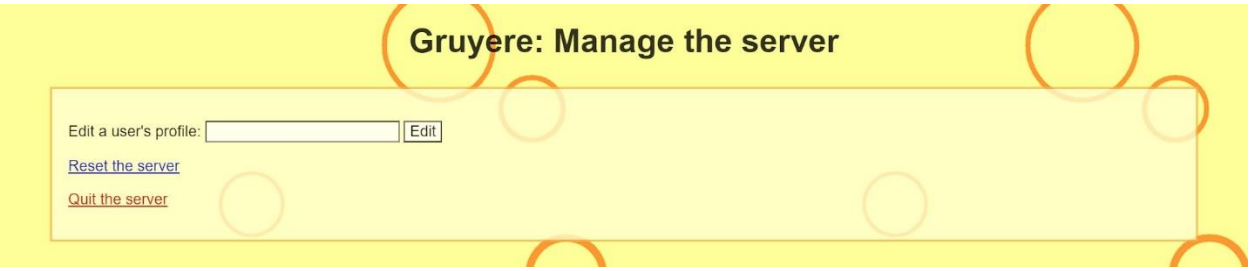
Explanation:
Elevation of privilege through client-state manipulation involves unauthorized manipulation of client-side data to gain elevated access or permissions leading to account takeovers or unauthorized actions.

URL of vulnerable webpage:

- https://google-gruyere.appspot.com/4569896092823451573207120148005844802 56/saveprofile?action=update&is_admin=True
- https://google-gruyere.appspot.com/4569896092823451573207120148005844802 56/manage.gtl

Screenshot before vulnerability:

Screenshot after vulnerability:



Observation:
To exploit vulnerability, we converted our account to being an administrator by issuing *action=update&is_admin=True* in the URL.

Prevention method:
There must be proper validation on the server side that checks that request is authorized. Server must be checked for authorization at the time the request is received.

Vulnerability Name: Cookie manipulation

Explanation:
Cookie manipulation vulnerability arises due to the client-side storage of cookies, making them susceptible to unauthorized modification.

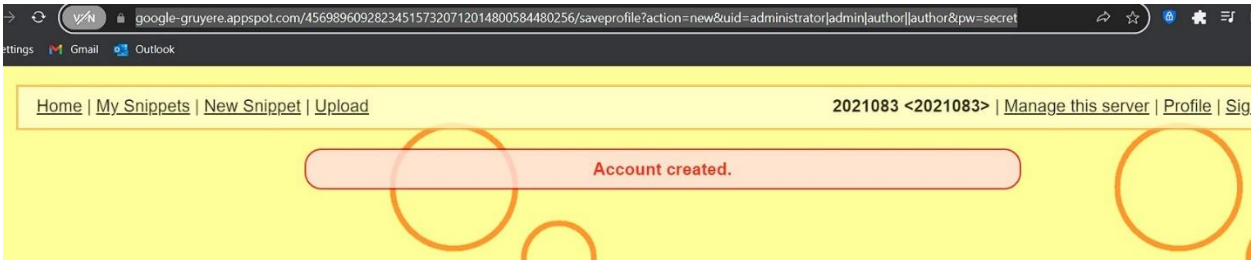
URL of vulnerable webpage:

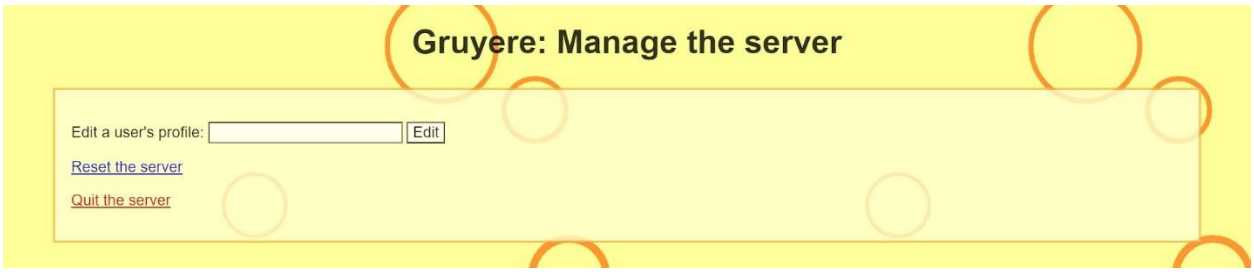
- https://google-gruyere.appspot.com/4569896092823451573207120148005844802 56/saveprofile?action=new&uid=administrator|admin|author||author&pw=secret

- https://google-gruyere.appspot.com/4569896092823451573207120148005844802 56/manage.gtl

Screenshot before vulnerability:

Screenshot after vulnerability:





Observation:
We can exploit this vulnerability by issuing a cookie for someone else's account by creating a new account. When we log into this account, it will issue a cookie which logs us as an administrator.

Prevention method:
We can prevent cookie manipulation by using SSL encryption, secure session management, regular cookie clearing, unique encrypted cookies, and avoiding storing sensitive data in cookies.

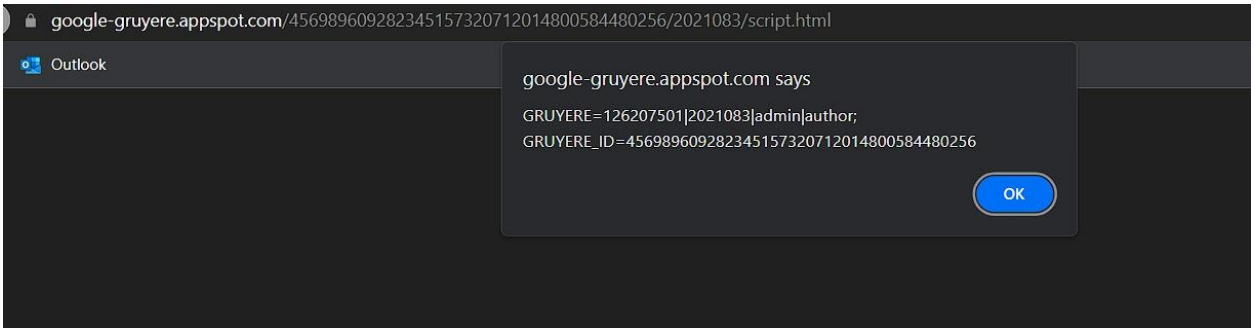
Vulnerability Name: Cross-site Scripting (File upload XSS)

Explanation:
Exploiting the file upload XSS vulnerability in gruyere involves uploading malicious HTML files to execute scripts, compromising the application's security, and leading to unauthorized code execution on the server or client-side.

URL of vulnerable webpage:
https://google-gruyere.appspot.com/456989609282345157320712014800584480256/2021083/script.html

Screenshot before vulnerability:

Screenshot after vulnerability:



Observation:
To exploit the file upload XSS vulnerability we uploaded a HTML file with a script that steals cookies, like ***<script>alert(document.cookie)</script>*** enabling attackers to access sensitive user information.

Prevention method:
To prevent this vulnerability, we can host the user content on a separate domain so that the script won't have access to any content from our domain.

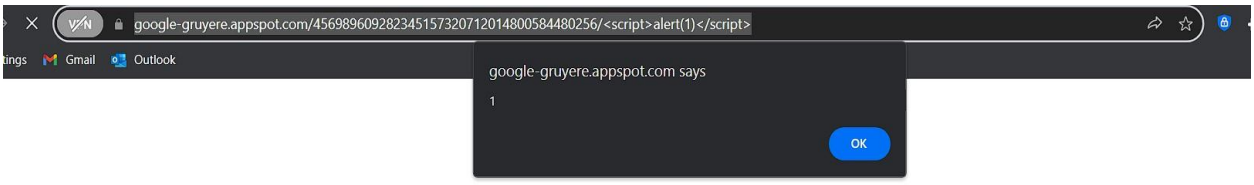
Vulnerability Name: Cross-site Scripting (Reflected XSS)

Explanation:
A reflected XSS vulnerability occurs when unvalidated user input is included in the server's response without proper sanitization, allowing attackers to craft malicious URLs that execute scripts when clicked.

URL of vulnerable webpage:
https://google-gruyere.appspot.com/456989609282345157320712014800584480256/%3Cscript%3Ealert(1)%3C/script%3E

Screenshot before vulnerability:

Screenshot after vulnerability:



Observation:
To exploit the reflected XSS vulnerability, we can craft a malicious script like ***<script>alert(1)</script>*** and persuade a victim to click on it, triggering the execution of the embedded script in the victim's browser.

Prevention method:
To prevent this vulnerability, we can escape user input displayed in error messages by adding text modifier to the template rendering the message, ensuring that user input is properly sanitized.

Vulnerability Name: Cross-site Scripting (Stored XSS)

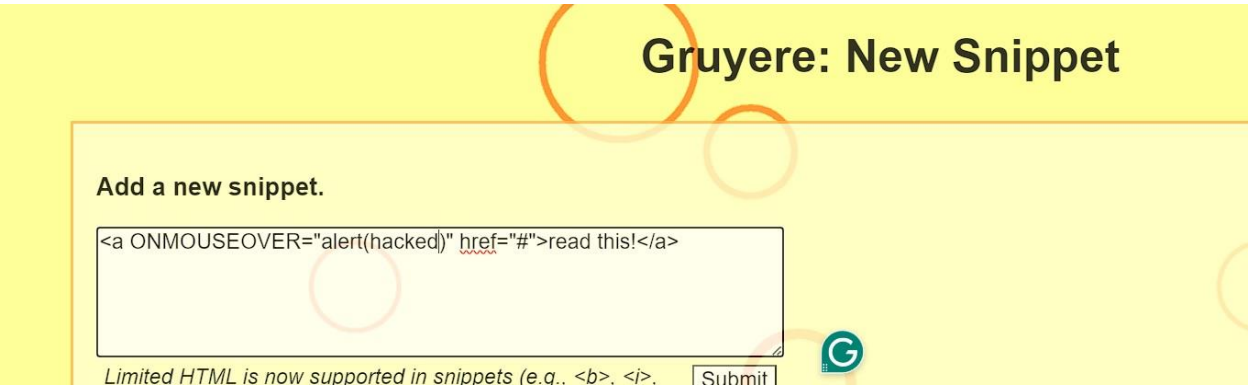
Explanation:
A stored XSS vulnerability occurs when an attacker stores malicious scripts within the application, which are then served to other users, leading to execution of unauthorized code, and compromising security.

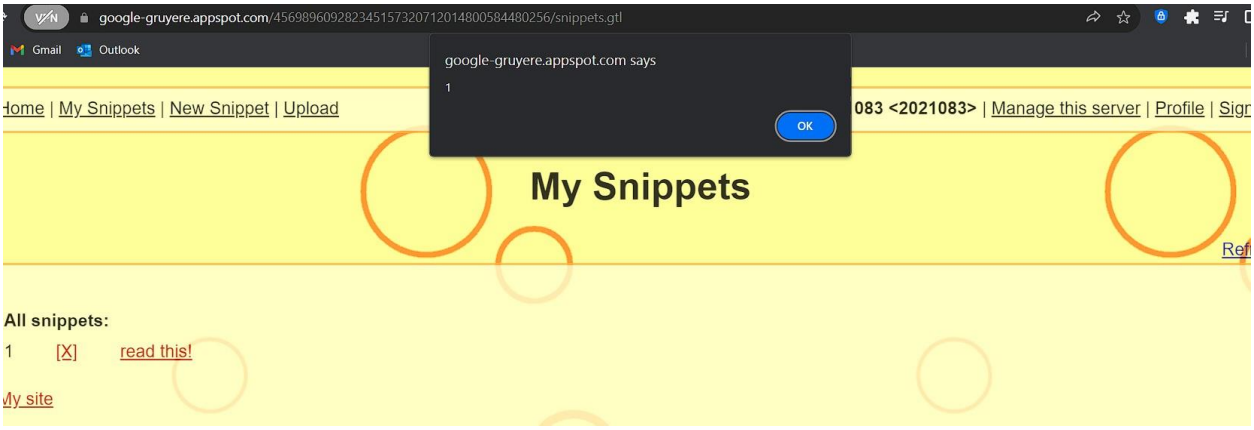
URL of vulnerable webpage:

https://google-gruyere.appspot.com/456989609282345157320712014800584480256/snippets.gtl

Screenshot before vulnerability:

Screenshot after vulnerability:





Observation:
To exploit the vulnerability, we used snippet like ***read this!*** as the input, taking benefit of inadequate HTML sanitization where certain attributes like onmouseover are not properly restricted.

Prevention method:
We can efficiently sanitize HTML by parsing input into an intermediate DOM structure, enforcing strict whitelists for allowed tags and attributes, and rigorously sanitizing URLs and CSS attributes if permitted.

Vulnerability Name: Cross-site Scripting (Reflected XSS via AJAX)

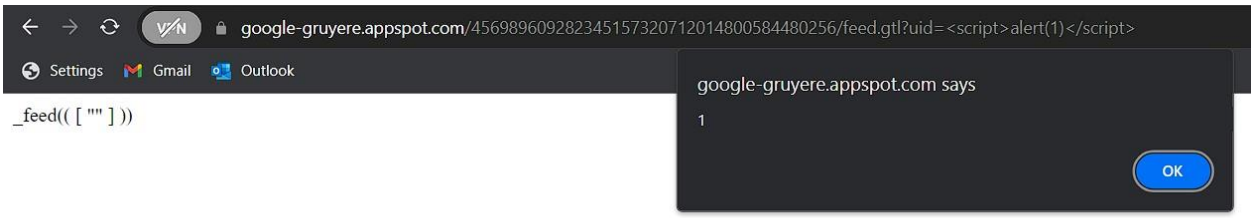
Explanation:
Reflected XSS via AJAX involves injecting malicious scripts into AJAX responses, triggering script execution when users interact with specific features, leading to unauthorized actions or data theft.

URL of vulnerable webpage:

https://google-gruyere.appspot.com/456989609282345157320712014800584480256/feed.gtl?uid=%3Cscript%3Ealert(1)%3C/script%3E

Screenshot before vulnerability:

Screenshot after vulnerability:



Observation:
The vulnerability is exploited, by enticing a victim to click on the compromised link, triggering execution of the script due to gruyere returning all gtl files as text/html, allowing the script to execute successfully.

Prevention method:
We need to ensure JSON content is not misinterpreted as HTML by replacing ***< and >*** with JavaScript escapes ***\x3c and \x3e*** and setting the response content type to application/JavaScript.