Google Gruyere Pentesting Report

Made By :
HAMAD MOIZ .CH
Muhammad Ahsan Ijaz
Usman Khaliq

Penetration Testing Report

Contents

1. Introduction

This report document hereby describes the proceedings and results of a Black Box security assessment conducted against **Google Gruyere**. The report hereby lists the findings and corresponding best practice mitigation actions and recommendations.

2. Objective

The objective of the assessment was to assess the state of security and uncover vulnerabilities in **Home of Google Gruyere** and provide with a final security assessment report comprising vulnerabilities, remediation strategy and recommendation guidelines to help mitigate the identified vulnerabilities and risks during the activity.

3. Scope

This section defines the scope and boundaries of the project.

Application Name	Home of Google Gruyere Web Application
URL	https://google-gruyere.appspot.com/

3.1. Assessment Attribute(s)

Parameter	Value
Starting Vector	External
Target Criticality	Critical
Assessment Nature	Cautious & Calculated
Assessment Conspicuity	Clear
Proof of Concept(s)	Attached wherever possible and applicable.

4. Risk Calculation and Classification

Following is the risk classification:

Info	Low	Medium	High	Critical
No direct threat to host/ individual user account. Sensitive information can be revealed to the adversary.	Vulnerabilities may not have public exploit (code) available or cannot be exploited in the wild. Vulnerability observed may not have high rate of occurrence. Patch workaround released by vendor.	Vulnerabilities may not have public exploit (code) available or cannot be exploited in the wild. Patch/ workaround not yet released by vendor.	Vulnerabilities which can be exploited publicly, workaround or fix/ patch available by vendor.	Vulnerabilities which can be exploited publicly, workaround or fix/ patch may not be available by vendor.

Table 1: Risk Rating

Summary

Outlined is a Black Box Application Security assessment for **Home of Acunetix Art Web Application**.

https://google-gruyere.appspot.com/

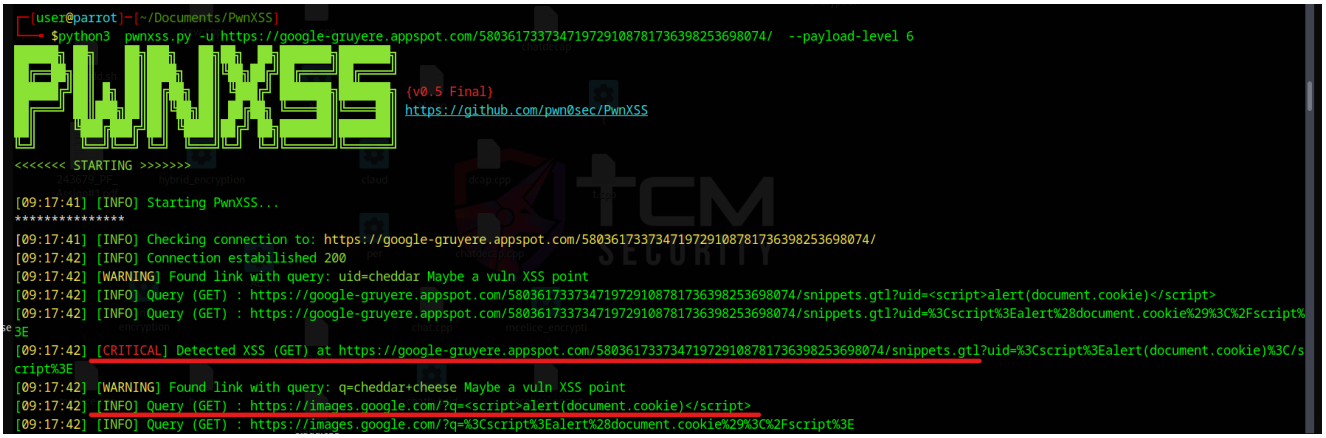
Following section illustrates **Detailed** Technical information about identified vulnerabilities.

Total: 20 Vulnerabilities

High	Medium	Low
12	8	2

5. XSS:

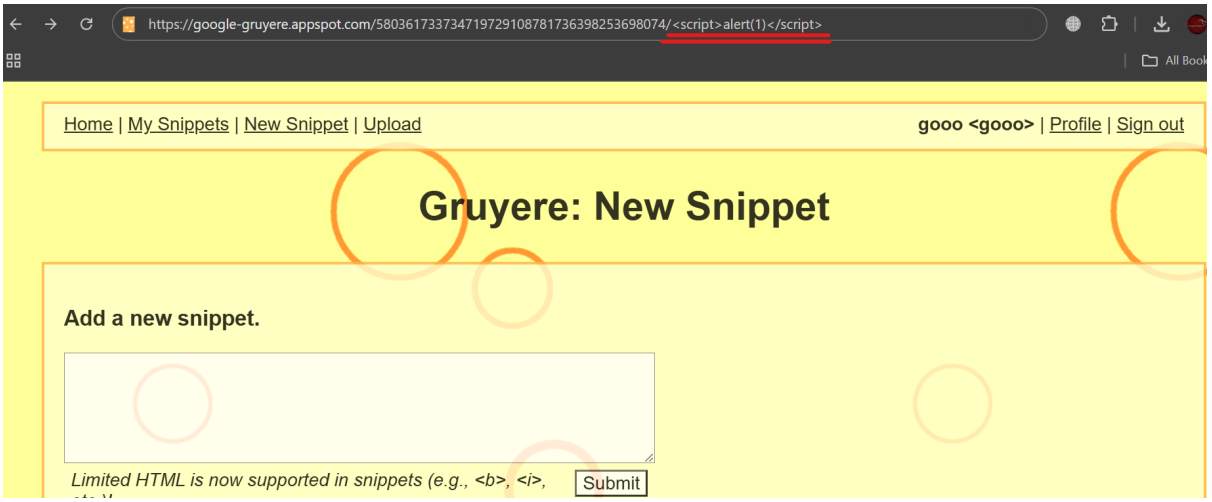
Finding the vulnerability through PwnXss:



5.1. Reflected XSS in the application.

Reference No:	Risk Rating:
WEB_XSS(r)	**Medium**
Tools Used:	
Browser,PwnXss	
Vulnerability Description:	
It was observed that in the search bar instead of search query if we inject JavaScript code then the JS code executes hence results into XSS	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
https://google-gruyere.appspot.com/	
Implications / Consequences of not Fixing the Issue	
An adversary having knowledge of JavaScript will be able to steal the user's credentials, hijack user's account, exfiltrate sensitive data and can access the client's computer.	
Suggested Countermeasures	
It is recommended to: • Filter input on arrival • Encode data on output • Use appropriate response headers • Use Content Security Policy (CSP) to reduce the severity of any existing XSS vulnerabilities	
References	
https://portswigger.net/web-security/cross-site-scripting	

Proof of concept:



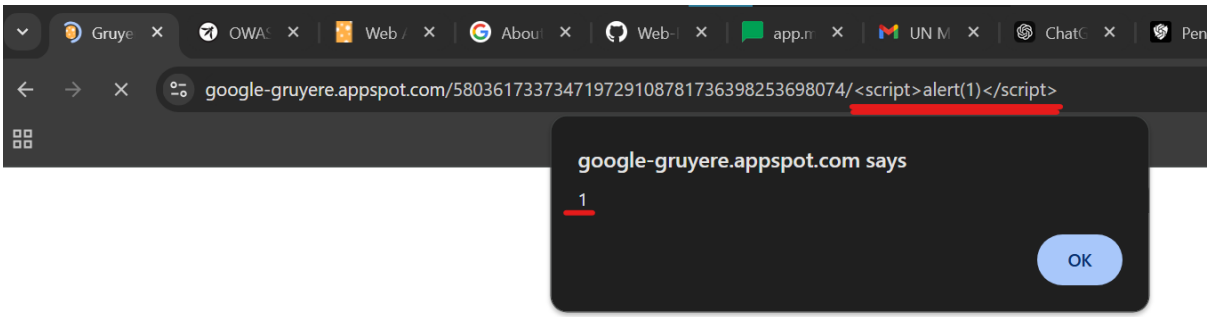


Fig 3: And hence we get to see the execution of our payload

URL #2 getting the user cookies:

Fig 1: Open the URL https://google-gruyere.appspot.com/5803617337347197291087817363982536980 74/<script>alert(document.cookie)</script>

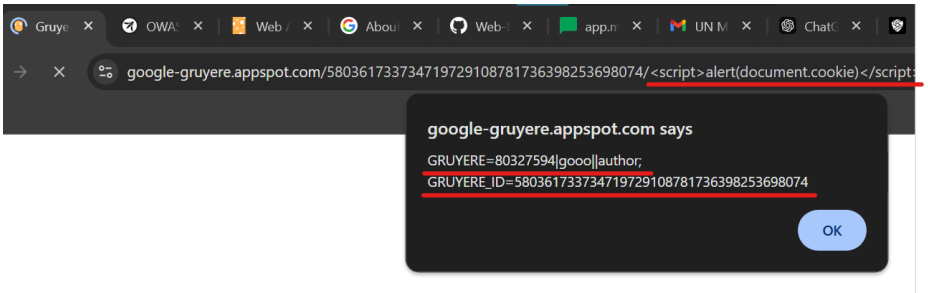


Fig 2: User ID and Cookie Fetched

5.2. Stored XSS in the Your Profile section.

Reference No:	Risk Rating:
WEB_Xss(s)	**High**
Tools Used:	
Browser	
Vulnerability Description:	
It was observed that in the your profile area instead of normal input if we execute JS code, then it gets stored in the server and hence it results into Stored XSS	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
https://google-gruyere.appspot.com/	
Implications / Consequences of not Fixing the Issue	
An adversary having knowledge of JavaScript will be able to steal the user's credentials, hijack user's account, exfiltrate sensitive data, can access the client's computer and even can redirect into other pages created by the adversary. And the impact will be faced by all users visiting the compromised page.	
Suggested Countermeasures	
It is recommended to: ● Filter input on arrival ● Encode data on output ● Use appropriate response headers ● Use Content Security Policy (CSP) to reduce the severity of any existing XSS vulnerabilities ● Using an Auto-Escaping Template System ● Using HTML Encoding	
References	
https://portswigger.net/web-security/cross-site-scripting https://blog.sqreen.com/stored-xss-explained/	

Proof of concept:

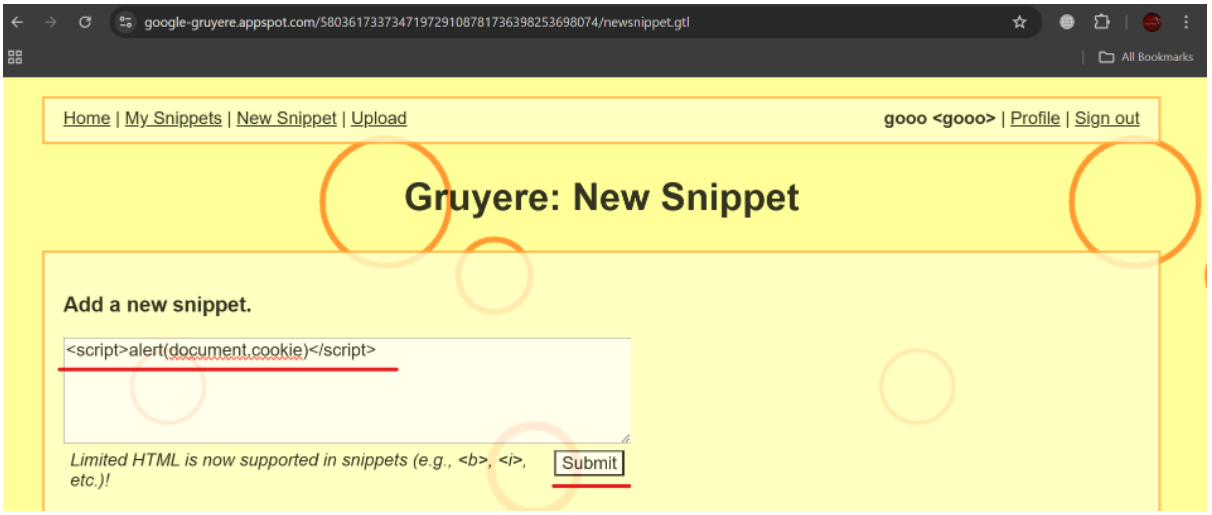


Fig 1: Visit the URL after Signing Up, Now the Script is stored in the Web Application

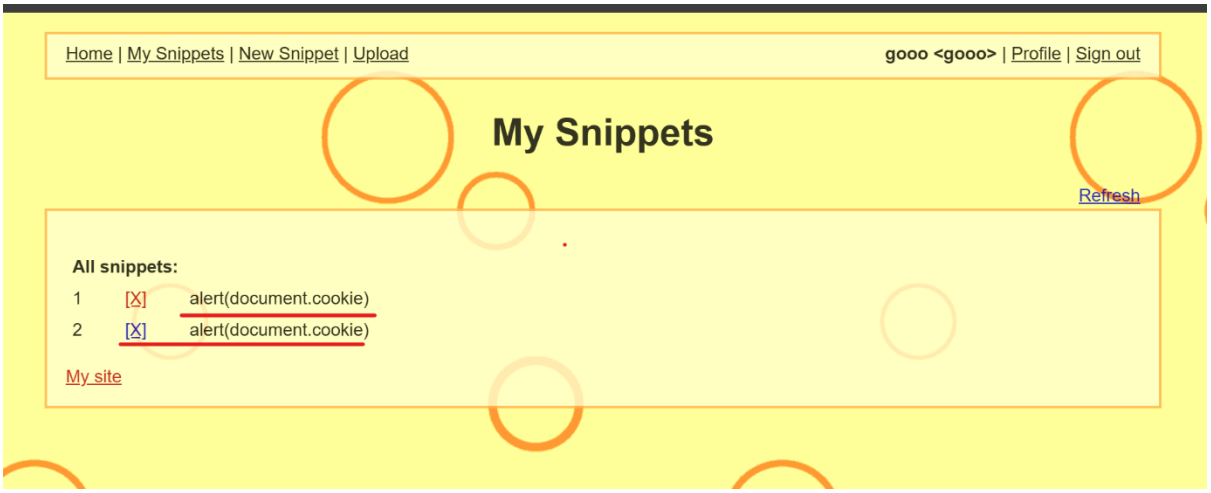


Fig 2: Type the Javascript code to all the field as any of them could be vulnerable to stored XSS and then click on the Update button

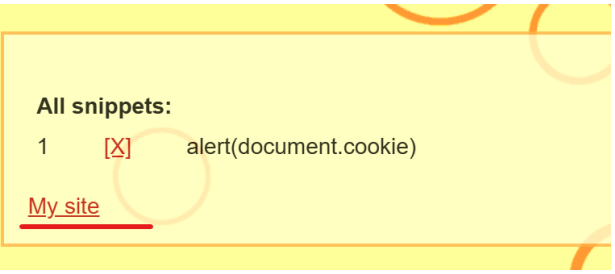
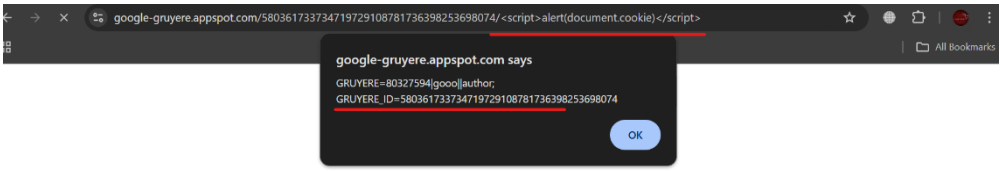
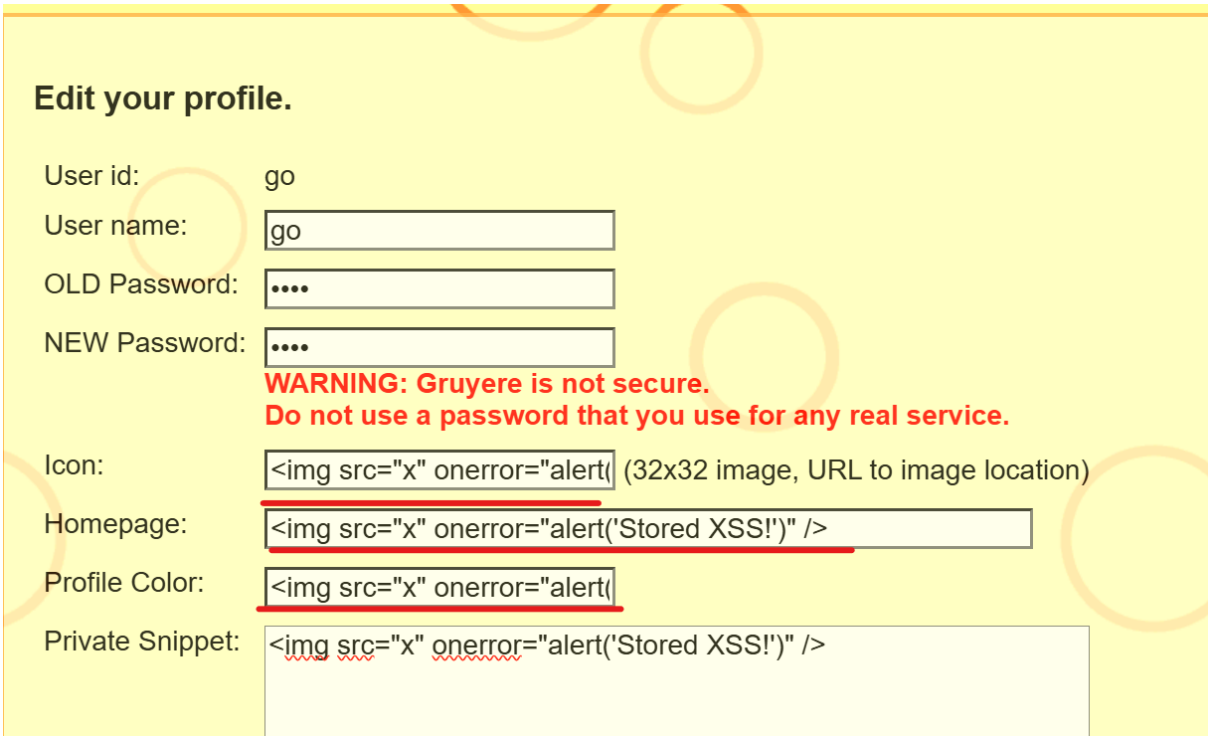


Fig 3: Hence the code gets executed and it's permanently stored in the server. Also it is found that the name field is vulnerable to stored XSS.

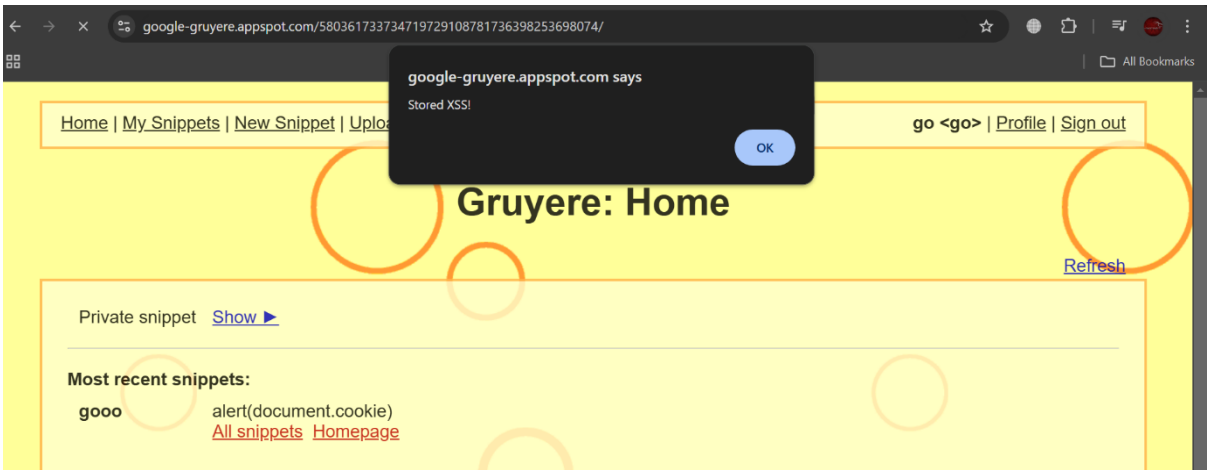


When Ever the My Site button is clicked the script in executed.

Stored XSS via HTML Attribute:



Whenever the page loads the it will allert



Changing the website layout through following payload

User id: shark

User name: shark

OLD Password: ....

NEW Password:

WARNING: Gruyere is not secure.
Do not use a password that you use for any real service.

Icon: <script> document.body.s (32x32 image, URL to image location)

Homepage: <script> document.body.style.backgroundColor = "red";

Profile Color: <script> document.body.s

Private Snippet:

```
<script>
  document.body.style.backgroundColor = "red";
  let hackedHeader = document.createElement("h1");
  hackedHeader.textContent = "Site I hacked";
  hackedHeader.style.color = "white";
  hackedHeader.style.textAlign = "center";
  document.body.prepend(hackedHeader);
</script>
```

google-gruyere.appspot.com/5860306530236166552593468022224349225 31/<script>%20%20%20%20%20document.b...

Introduction to Cyb... CompTIA Security+... YouTube WhatsApp Migadu Webmail All Bookmarks

Site I hacked

Home | My Snippets | New Snippet | Upload **shark <shark>** | Profile | Sign out

Invalid request: /

File Upload Xss:

Html file was uploaded and the accessed on the link:

5.3. AJAX:

Reference No:	Risk Rating:
Ajax-Dos	low
Tools Used:	
Browser,Brupsuit	
Vulnerability Description:	
It was observed that in the your profile area instead of normal input if we execute JS code, then it gets stored in the server and hence it results into Stored XSS	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
https://google-gruyere.appspot.com/	
Implications / Consequences of not Fixing the Issue	
An adversary having knowledge of JavaScript will be able to steal the user's credentials, hijack user's account, exfiltrate sensitive data, can access the client's computer and even can redirect into other pages created by the adversary. And the impact will be faced by all users visiting the compromised page.	
Suggested Countermeasures	
It is recommended to: • Filter input on arrival • Encode data on output • Use appropriate response headers • Use Content Security Policy (CSP) to reduce the severity of any existing XSS vulnerabilities • Using an Auto-Escaping Template System	

• Using HTML Encoding
References
https://portswigger.net/web-security/cross-site-scripting https://blog.sqreen.com/stored-xss-explained/

DoS via AJAX

First of all, let's sign in using my Gruyere account "Forensicxs"

Forensicxs

We can see the snippets. Clicking on "refresh", we see the response corresponding to the snippets content

Forensicxs

Then, let's create a user "**private_snippet**", and create several snippets

```
1  HTTP/2 200 OK
2  Cache-Control: no-cache
3  Content-Type: text/html
4  Pragma: no-cache
5  X-Xss-Protection: 0
6  X-Cloud-Trace-Context: ba216539a16d1e407d25b52ef7508802
7  Vary: Accept-Encoding
8  Date: Sat, 18 Sep 2021 15:45:38 GMT
9  Server: Google Frontend
10 Content-Length: 230
11 Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-T051=":443";
   ma=2592000,h3-Q050=":443"; ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443";
   ma=2592000,quic=":443"; ma=2592000; v="46,43"
12
13
14 _feed((
15
16
17
18
19 {
20 "private_snippet":
21   ""
22
23
24
25 ,"cheddar":
26   "Gruyere is the cheesiest application on the web."
27 ,"Forensicxs":
28   "
29 I can hack this !"
30 ,"brie":
31   "Brie is the queen of the cheeses<span style=color:red>!!!</span>"
32 }
33
34 ))
35
```

All snippets:

1 Again another DoS via AJAX

2 Another DoS via AJAX

3 DoS via AJAX

private_snippet's site

Here is the response. The snippets of the other users have been deleted

```
1  HTTP/2 200 OK
2  Cache-Control: no-cache
3  Content-Type: text/html
4  Pragma: no-cache
5  X-Xss-Protection: 0
6  X-Cloud-Trace-Context: 99671b53c46df4230c5a3e3dcf66961d
7  Vary: Accept-Encoding
8  Date: Sat, 18 Sep 2021 15:16:27 GMT
9  Server: Google Frontend
10 Content-Length: 119
11 Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-T051=":443";
   ma=2592000,h3-Q050=":443"; ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443";
   ma=2592000,quic=":443"; ma=2592000; v="46,43"
12
13
14 _feed((
15
16
17 [
18 "private_snippet"
19
20 ,"
21 Again another DoS via AJAX"
22
23 ,"
24 Another DoS via AJAX"
25
26 ,"
27 DoS via AJAX"
28
29 ]
30
31
32
33 ))
34
```

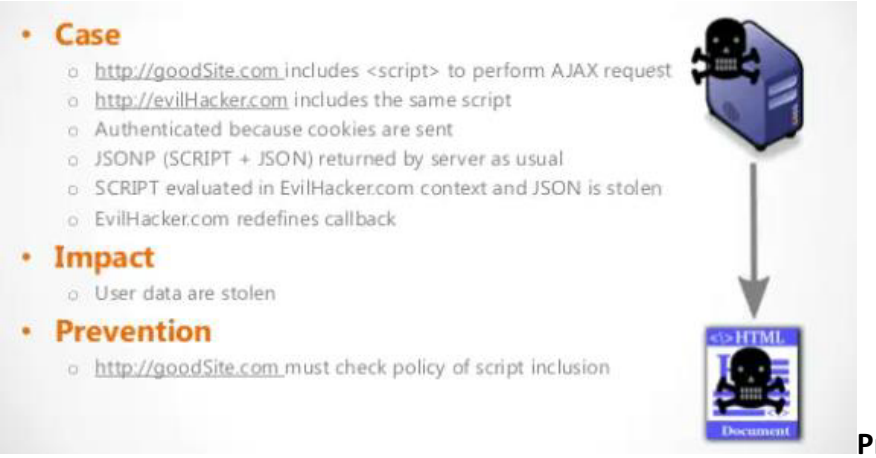
private_snippet

The flaw here is the structure of the response. We see the construction here

```
45      Private Snippet:
46    </td><td>
47      <input type='text' name='private_snippet'
48          value='{{_profile.private_snippet:text}}'>
49  </td></tr>
```

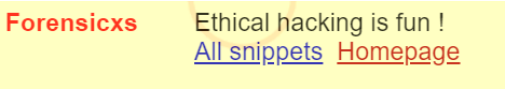
6. Cross Site Script Inclusion (XSSI)

XSSI is a client-side attack similar to Cross Site Request Forgery (CSRF) but has a different purpose. Where CSRF uses the authenticated user context to execute certain state-changing actions inside a victim's page (reset password, etc.), XSSI instead uses JavaScript on the client side to leak sensitive data from authenticated sessions

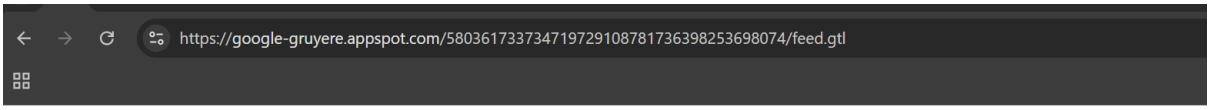


Principles of an XSSI

Let's follow the example provided by Google. Here is my private snippet on my home page. This is the "sensitive information" that we are going to leak



Following JavaScript code for XSSI



_feed(({ "private_snippet": "" ,"cheddar": "Gruyere is the cheesiest application on the web." ,"brie": "Brie is the queen of the cheeses!!!" }))

```html
<!DOCTYPE html>
<html>

<script>function feed(s) {
    alert("Your private snippet is: " + s['private_snippet']);
}
</script>

<script src="https://google-gruyere.appspot.com/6290099538641962524917807181022069070 7/feed.gtl"></script>

</body>
</html>
```

7. Path traversal

Path traversal	**High** [redacted]
Tools Used:	
curl	
Vulnerability Description:	
A web application vulnerability that allows an attacker to access files and directories outside of the web root folder	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis & Automated Analysis	
Implications / Consequences of not Fixing the Issue	
attacker can steal our critical information from files or directories	
Suggested Countermeasures	
• we need to prevent access to files outside the resources directory. Validating file paths is a bit tricky as there are various ways to hide path elements like "../" or "~" that allow escaping out of the resources folder. The best protection is to only serve specific resource files.	

Proof of concept:

1. Path Traversal

Information disclosure via path traversal

7.1.

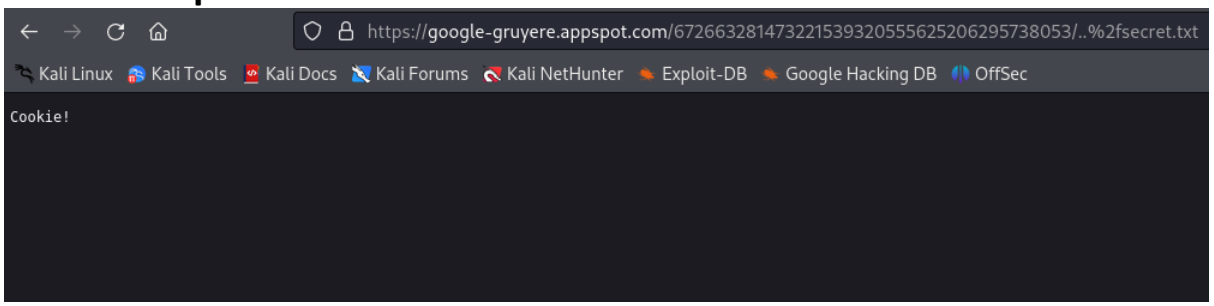
```
┌──(ahsan㉿ahsan)-[~]
└─$ gobuster dir -u https://google-gruyere.appspot.com/6726632814732215393205556252062957380 53/ \
-w txtfiles \
--exclude-length 2262

===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     https://google-gruyere.appspot.com/6726632814732215393205556252062957380 53/
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                txtfiles
[+] Negative Status codes:   404
[+] Exclude Length:          2262
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/error_log.txt        (Status: 200) [Size: 2239]
/credentials.txt      (Status: 200) [Size: 2241]
/secret.txt           (Status: 200) [Size: 2236]
/config.txt           (Status: 200) [Size: 2236]
/api_keys.txt         (Status: 200) [Size: 2238]
/passwords.txt        (Status: 200) [Size: 2239]
/keys.txt             (Status: 200) [Size: 2234]
/backup.txt           (Status: 200) [Size: 2236]
/access_log.txt       (Status: 200) [Size: 2240]
/debug.txt            (Status: 200) [Size: 2235]
/database.txt         (Status: 200) [Size: 2238]
/settings.txt         (Status: 200) [Size: 2238]
/env.txt              (Status: 200) [Size: 2233]
```

After finding a secret.txt file then try a path traversal attack to retrieve the content of a file . in this scenario, I used an encoding technique to bypass server

curl tool

```
┌──(ahsan㉿ahsan)-[~]
└─$ curl -X GET https://google-gruyere.appspot.com/6726632814732215393205556252062957380 53/..%2fsecret.txt

Cookie!
```

7.2. URL snap

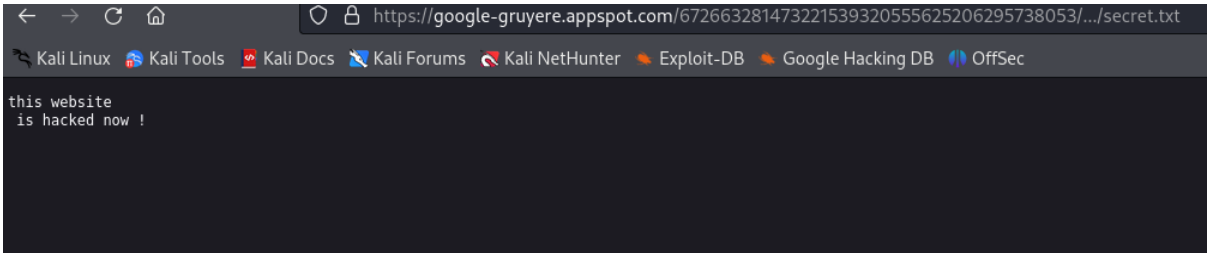


8. Data tampering via path traversal

First I create a new user named .. and create the same file secret.txt I put some new information into a file that I will show later) upload new secret.txt file .

I log in as a user .. and upload a file secret.txt.

After navigating to the file path now the data of file secret.txt has been tampered successfully

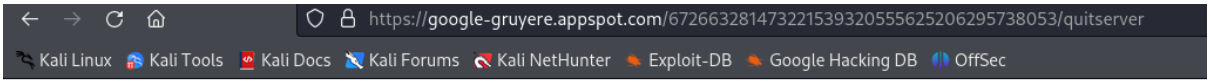


Denial of service	**High** [redacted]
Tools Used:	
none	
Vulnerability Description:	
A denial of service (DoS) attack is an attempt to make a server unable to service ordinary requests. A common form of DoS attack is sending more requests to a server than it can handle. The server spends all its time servicing the attacker's requests that it has very little time to service legitimate requests	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Implications / Consequences of not Fixing the Issue	
A DoS attack can make a website or service inaccessible to legitimate users. This can hinder business operations and customer experience	
Suggested Countermeasures	
• .Ensure the application can handle large payloads gracefully to prevent crashes. • Limit the number of concurrent sessions a user can maintain.	

8.1.1. DoS - Quit the Server

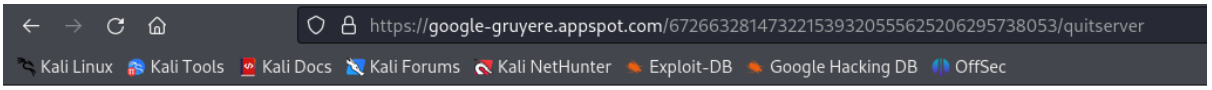
The simplest form of denial of service is shutting down a service.

mostly server protects against non-administrators accessing certain URLs but the list includes `/quit` instead of the actual URL `/quitserver`.



Gruyere System Alert

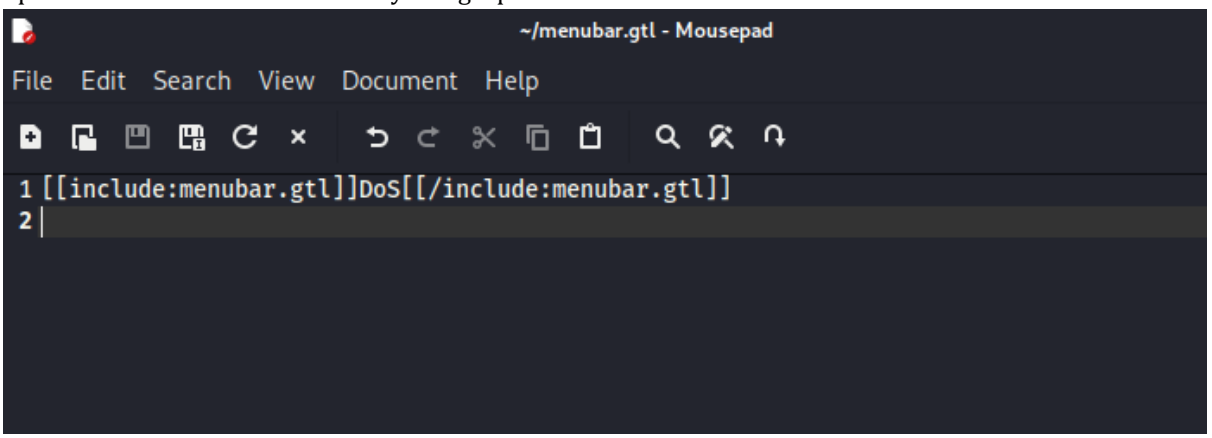
```
Server is restarting
... please wait ...

Done.

Please hit refresh.
```



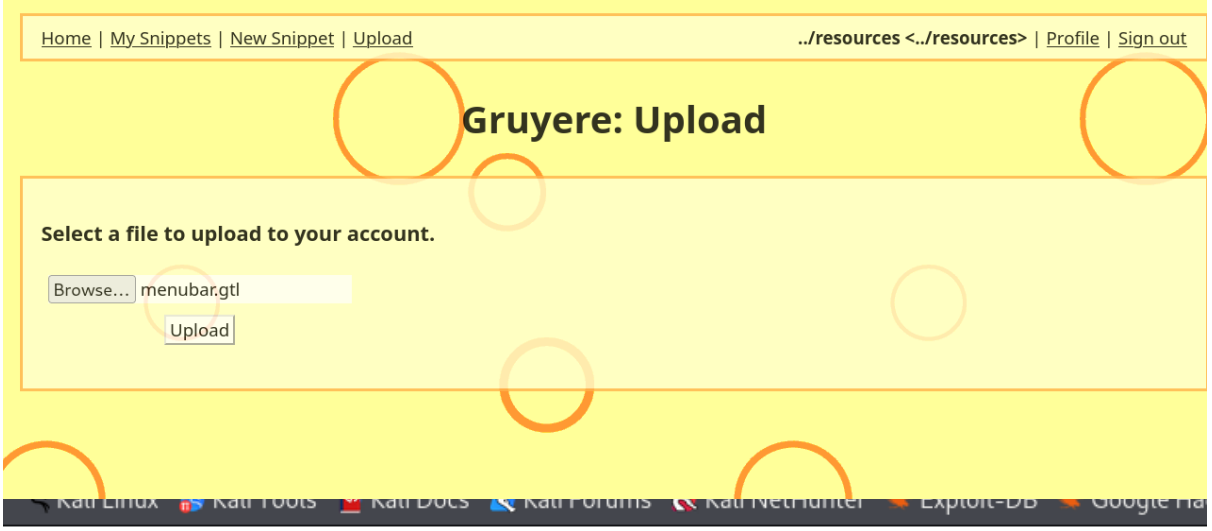

```
Server quit.
```

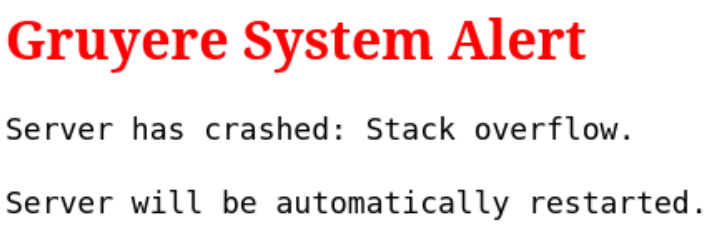
8.1.2. DoS - Overloading the Server

Every page includes the menubar.gtl template.
So I create a file named menubar.gtl containing: [[include:menubar.gtl]]DoS[[/include:menubar.gtl]]and upload it to the resources directory using a path traversal attack



after signing in as user `../resources` and uploading file `menubar.gtl`server is crashed





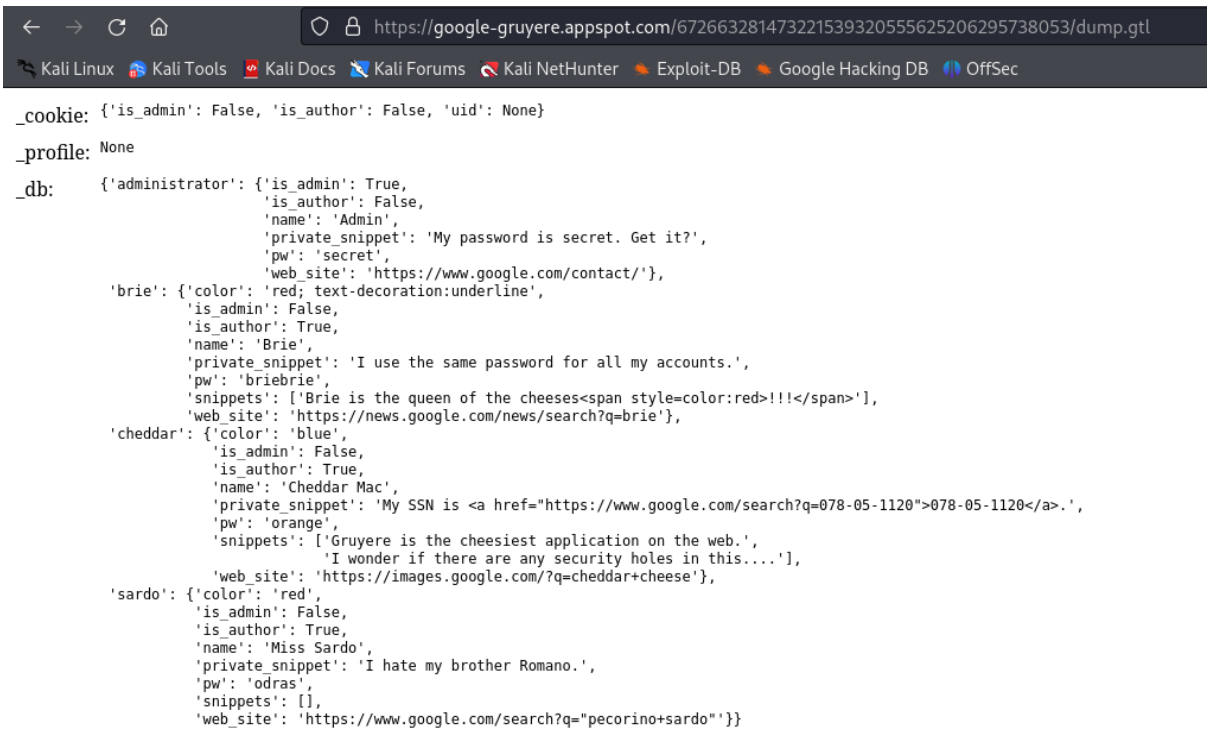
9. Configuration Vulnerabilities

Configuration Vulnerabilities	**High** [redacted]
Tools Used:	
none	
Vulnerability Description:	
This is particularly an issue with third party software where an attacker has easy access to a copy of the same application or framework you are running. Hackers know the default account names and passwords. For example, looking at the contents of `data.py` you know that there's a default administrator account named 'admin' with the password 'secret'.	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Implications / Consequences of not Fixing the Issue	
Exposure of sensitive data such as server configurations, API keys, database details, or source code.Enables attackers to craft targeted attacks or exploit known vulnerabilities	
Suggested Countermeasures	

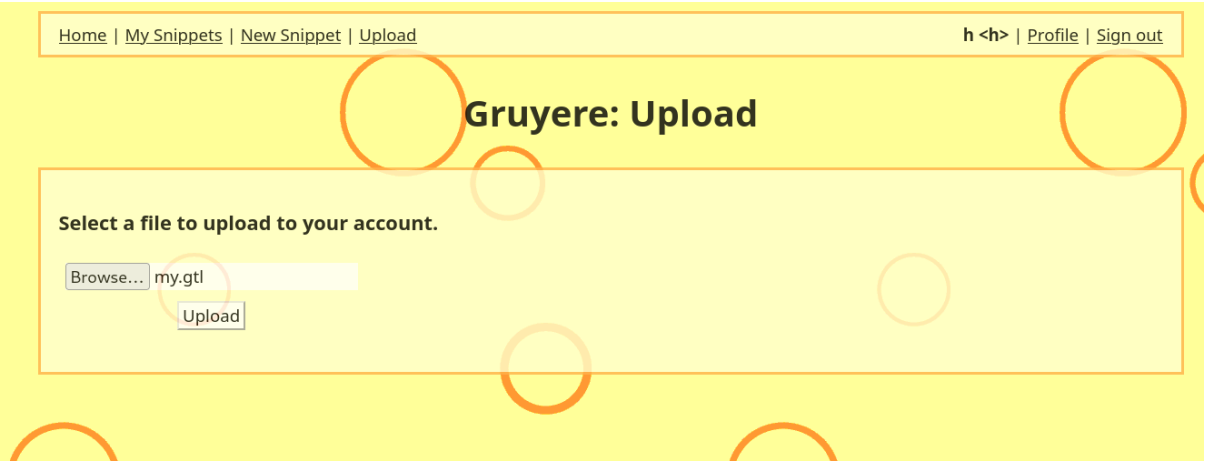
- Restrict access to sensitive files (e.g., `.env`, `config.php`).
- Remove unnecessary comments or sensitive data in code or configurations.
- Harden configurations based on best practices (e.g., OWASP guidelines).

10. Information Disclosure

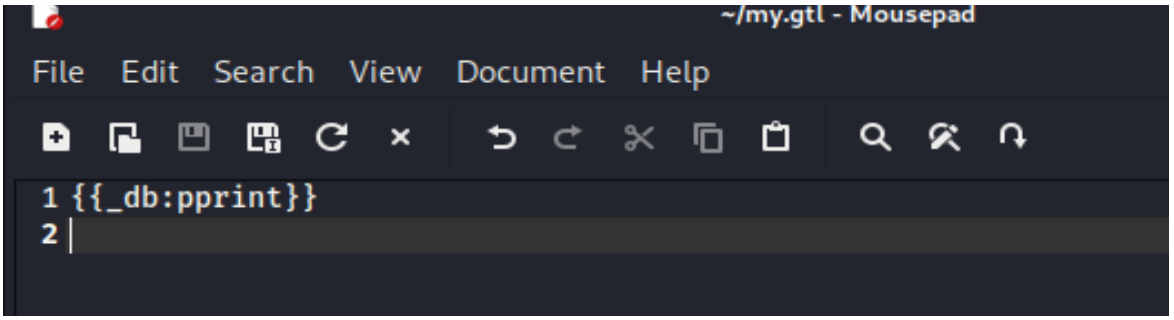
After looking at all the files installed with Gruyere. I found a file `dump.gtl` this file expose the information of all users along with passwords further this file exposes the data of database

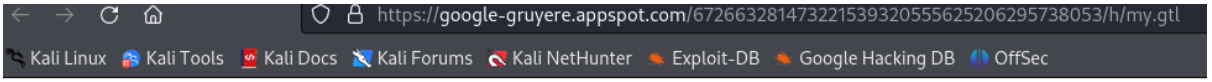


Gruyere allows the user to upload files of any type, including `.gtl` files. So the attacker can simply upload their own copy of `dump.gtl` or a similar file **i**



Here I create a file with the same gtl extension and add this code for retreiving the data from the database .






```
{'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'},
 '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'},
 'h': {'is_author': 'True', 'name': 'h', 'pw': 'h'},
 '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"'}}
```

11. Client-State Manipulation

11.1. Privilege Escalation

Reference No:	Risk Rating:
WEB_VUL_01	**High**
Tools Used:	
Browser	
Vulnerability Description:	
Privilege escalation is the act of exploiting a bug, a design flaw, or a configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user. The result is that an application or user with more privileges than intended by the application developer or system administrator can perform unauthorized actions.	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
https://googlegruyere.appspot.com/saveprofile?action=new&uid=Usman+Khaliq&pw=1122&is_author=True	
Implications / Consequences of not Fixing the Issue	
An adversary having knowledge about Privilege escalation could easily become admin from normal user.	
Suggested Countermeasures	
It is recommended to implement below control for mitigating the Privilege escalation: ● Input Validation ● Url Encryption	
References	
https://en.wikipedia.org/wiki/Privilege_escalation https://www.eccouncil.org/cybersecurity-exchange/penetration-testing/privilege-escalations-attacks/	

Proof of concept:

Manual Analysis:

Create a new user with default privileges.

Now User has Created Successfuly

This is the file that is visible we can see that its checking with if statement that If cookies are present are not in request.

```
</head>

<body>
[[include:menubar.gtl]][[/include:menubar.gtl]]
<div>
<h2>Gruyere: Profile</h2>
</div>

<div class='content'>
[[if:_cookie.is_admin]]
  <h3>Add a new account or edit an existing account.</h3>
[[/if:_cookie.is_admin]]
[[if:!_cookie.is_admin]]
  <h3>Edit your profile.</h3>
[[/if:!_cookie.is_admin]]
[[if:_message]]
<div class='message'>{{_message}}</div>
[[/if:_message]]

<form method='get' action='/{{_unique_id}}/saveprofile'>
<input type='hidden' name='action' value='update'>
<table>
  <tr><td>
    User id:
  </td><td>
[[if:_cookie.is_admin]]
  [[if:uid]]
    <input type='hidden' name='uid' value='{{uid.0}}'>
    {{uid.0}}
  [[/if:uid]]
  [[if:!uid]]
    <input type='hidden' name='uid' value='{{_cookie.uid}}'>
    {{_cookie.uid}}
  [[/if:!uid]]
[[/if:_cookie.is_admin]]
[[if:!_cookie.is_admin]]
  [[if:_cookie.uid]]
    {{_cookie.uid}}
  [[/if:_cookie.uid]]
  [[if:!_cookie.uid]]
      &lt;not logged in&gt;
  [[/if:!_cookie.uid]]
[[/if:!_cookie.is_admin]]
  </td></tr>
<tr><td>
    User name:
  </td><td>
    <input type='text'
      value='[[if:uid]]{{_db.*uid.name:text}}[[/if:uid]][[if:!uid]]{{_profile.name:text}}[[/if:!uid]]'
      name='name' maxlength='16'>
```

https://google-gruyere.appspot.com/4663372806515774152532181602870044189 06/saveprofile?action=update&is_admin=True

Home | My Snippets | New Snippet | Upload Usman Khaliq <Us

Gruyere: Home

Most recent snippets:

Cheddar Mac Gruyere is the cheesiest application on the web.
All snippets Homepage

Brie Brie is the queen of the cheeses!!!
All snippets Homepage

Home | My Snippets | New Snippet | Upload yes <yes> | Manage this server | Profile | Sign out

Gruyere: Home

Refresh

Most recent snippets:

Cheddar Mac Gruyere is the cheesiest application on the web.
All snippets Homepage

Brie Brie is the queen of the cheeses!!!
All snippets Homepage

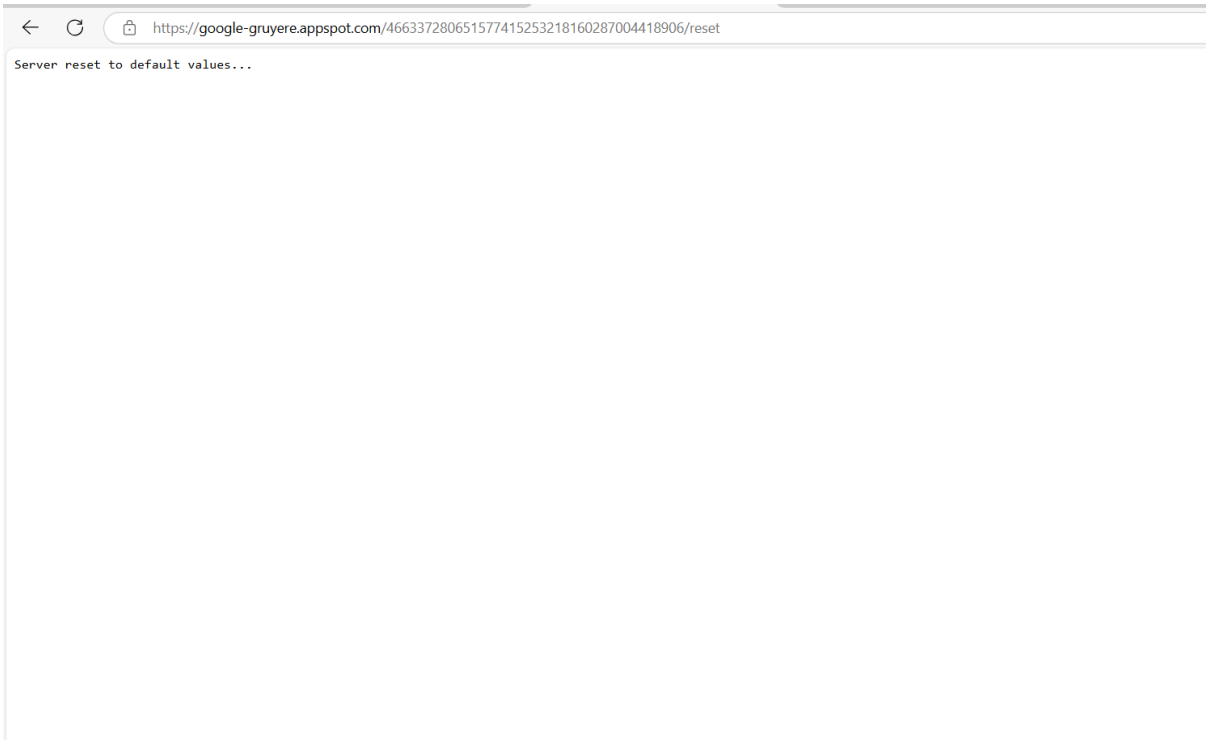
Home | My Snippets | New Snippet | Upload yes <yes> | Manage this server | Profile | Sign out

Gruyere: Manage the server

Edit a user's profile: [ ] Edit

Reset the server

Quit the server



11.2. Cookie Manipilation

Reference No:	Risk Rating:
WEB_VUL_02	**High**
Tools Used:	
Browser	
Vulnerability Description:	
This vulnerability allows attackers to alter the cookie's content and impersonate other users by encoding their modified data back into the cookie. This attack involves stealing a user's cookie to gain unauthorized access to their account within an application.	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
https://googlegruyere.appspot.com/saveprofile	
Implications / Consequences of not Fixing the Issue	
This attack involves stealing a user's cookie to gain unauthorized access to their account within an application.	
Suggested Countermeasures	
Use Secure Flags Validate and Sanitize Inputs Implement SameSite Attribute Encrypt Cookie Data	
References	
PortSwigger Web Security Academy - DOM-based Cookie Manipulation	

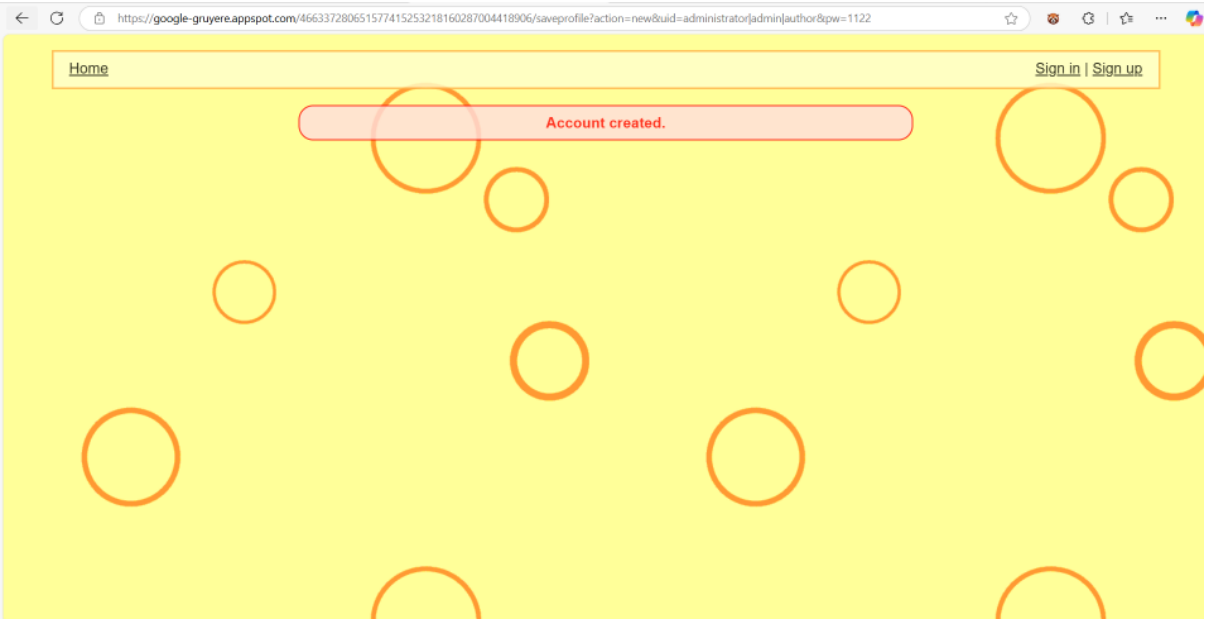
Improving Web Application Security: Threats and Countermeasures

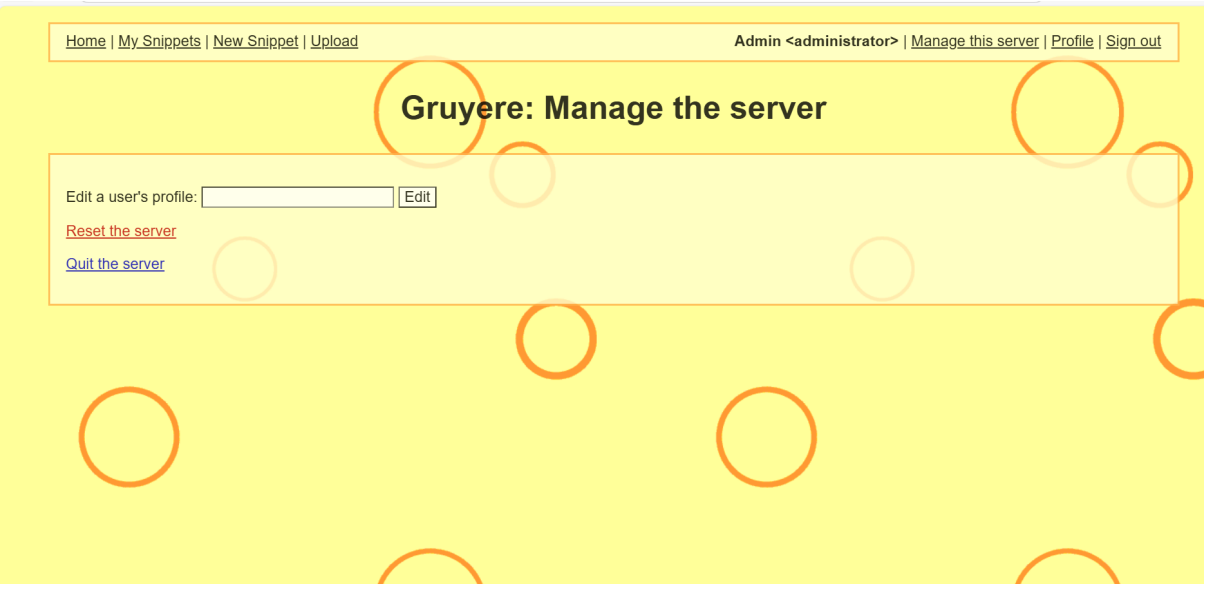
TuringSecure - Cookie Manipulation (DOM-Based)

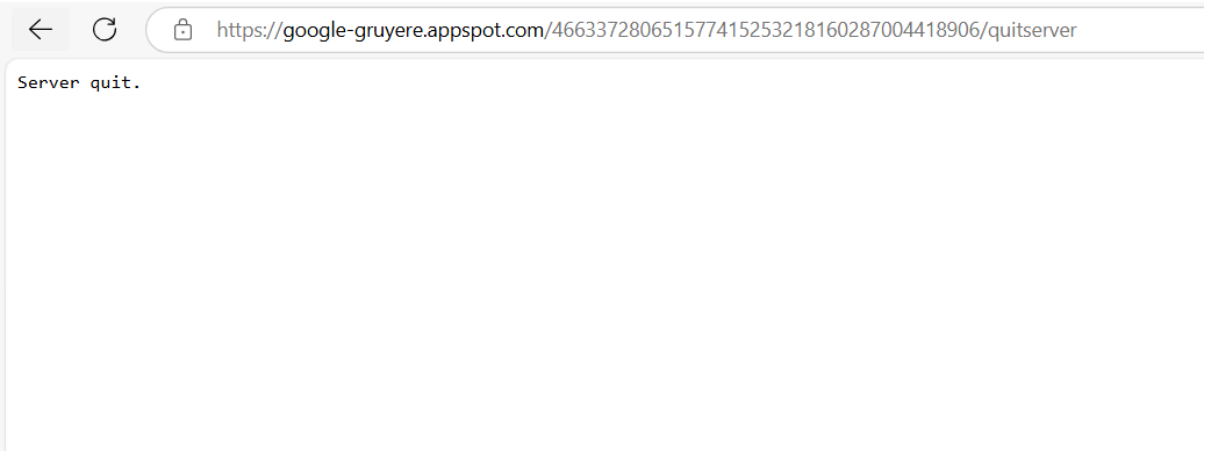
Proof of concept:

Manual Analysis:
We can simple map the new administrator user with the existing user to user their cookies and and created a new admin user we can utilize the given command.

https://google -groyere-appspot.com/saveprofile?action=new&uid=administrator|admin|author&pw=1122







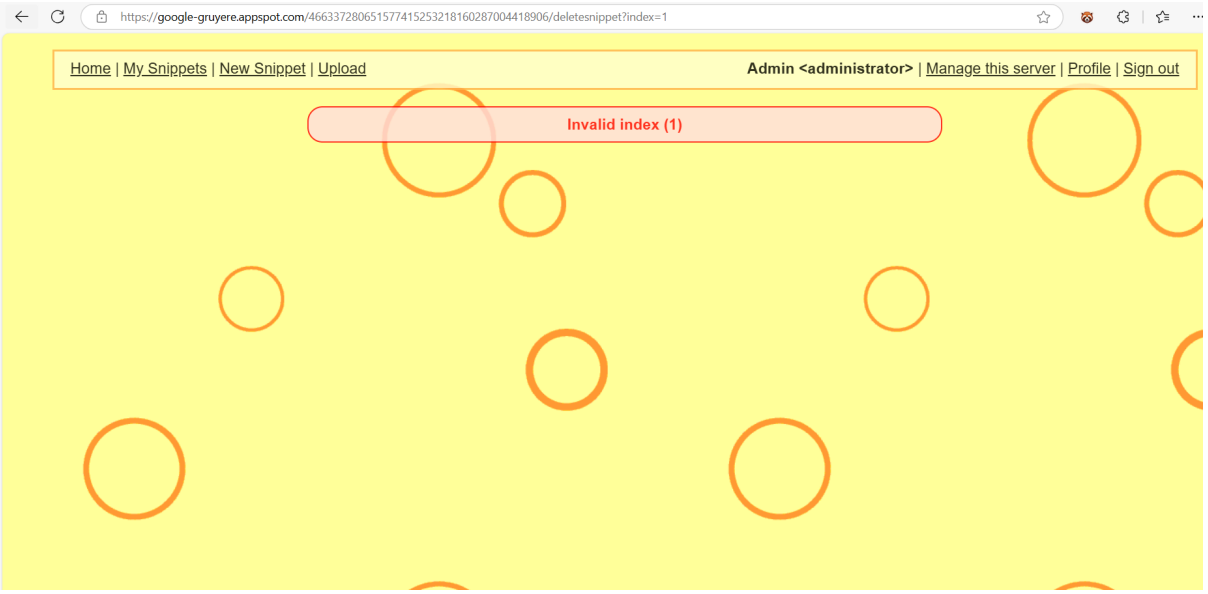
11.3. XSRF Cross-site Request Forgery.

Reference No:	Risk Rating:
WEB_VUL_03	**Medium**
Tools Used:	
Browser	
Vulnerability Description:	
When a browser makes requests to a site, it always sends along any cookies it has for that site, regardless of where the request comes from. Additionally, web servers generally cannot distinguish between a request initiated by a deliberate user action (e.g., user clicking on "Submit" button) versus a request made by the browser without user action (e.g., request for an embedded image in a page). Therefore, if a site receives a request to perform some action (like deleting a mail, changing contact address), it cannot know whether this action was knowingly initiated by the user — even if the request contains authentication cookies. An attacker can use this fact to fool the server into performing actions the user did not intend to perform.	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
https://googlegruyere.appspot.com/4663372806515774152532181602870044189 06/deletesnippet?index=1	
Implications / Consequences of not Fixing the Issue	
An attacker can use this fact to fool the server into performing actions the user did not intend to perform.	
Suggested Countermeasures	
It is recommended to: • Use CSRF Tokens • Synchronizer Token Pattern • Double Submit Cookies • SameSite Cookie Attribute • Custom Request Headers	
References	
https://owasp.org/www-community/attacks/csrf	
https://en.wikipedia.org/wiki/Cross-site_request_forgery	

Proof of concept:

```
[---]            XSRFProbe, A Cross Site Request Forgery Audit Toolkit          [---]
[---]                                                                           [---]
[---]                      ~  Author : Pinaki Mondal  ~                         [---]
[---]                     ~  github.com / 0xInfection  ~                        [---]
[---]                                                                           [---]
[---]                          ~  Version 2.3.0  ~                              [---]

[!] Testing site google-gruyere.appspot.com status...
[+] Site seems to be up!
[!] Testing 4663372806515774152532181602870044189 06/ endpoint status...
[+] Endpoint seems to be up!
[*] Preparing the request...
[*] Processing the GET Request...
[!] Trying to parse response...
[!] Checking endpoint request validation via Referer Checks...

+-------------------------------------+
|   Referer Based Request Validation   |
+-------------------------------------+

[!] Making request on normal basis...
[*] Preparing the request...
[*] Processing the GET Request...
```

```
[!] Making request on normal basis...
[*] Preparing the request...
[*] Processing the GET Request...
[*] Setting generic headers...
[!] Making request with Tampered Referer Header...
[*] Preparing the request...
[*] Processing the GET Request...
[-] Endpoint Referer Validation Not Present!
[-] Heuristics reveal endpoint might be VULNERABLE to Origin Based CSRFs...
[+] Possible CSRF Vulnerability Detected : https://google-gruyere.appspot.com/4663372806515774152532181602870044189 06/!
[+] Possible Vulnerability Type: No Referer Based Request Validation
[!] Confirming the vulnerability...
[!] Confirming endpoint request validation via Origin Checks...

+-------------------------------------+
|   Origin Based Request Validation   |
+-------------------------------------+

[!] Making request on normal basis...
[*] Preparing the request...
[*] Processing the GET Request...
[*] Setting generic headers...
[!] Making request with Tampered Origin Header...
[*] Preparing the request...
[*] Processing the GET Request...
[-] Endpoint Origin Validation Not Present!
[-] Heuristics reveal endpoint might be VULNERABLE to Origin Based CSRFs...
[+] Possible CSRF Vulnerability Detected : https://google-gruyere.appspot.com/4663372806515774152532181602870044189 06/!
[!] Possible Vulnerability Type: No Origin Based Request Validation

[!] Retrieving all forms on https://google-gruyere.appspot.com/4663372806515774152532181602870044189 06/...
```



11.4. Remote Code Execution.

Reference No:	Risk Rating:
WEB_VUL_04	**High** [redacted]
Tools Used:	
Browser, Python	
Vulnerability Description:	
If an attacker can execute arbitrary code remotely on your server, it's usually game over. They may be able to take control over the running program or potentially break out the process to open a new shell on the computer. From here, it's usually not hard to compromise the entire machine the server is running on.	
Vulnerability Identified by / How It Was Discovered	
Manual Analysis	
Vulnerable URLs / IP Address	
https://google-gruyere.appspot.com/4663372806515774152532181602870044189 06/upload2	
Implications / Consequences of not Fixing the Issue	
An adversary having knowledge of JavaScript will be able to steal the user's credentials, hijack user's account, exfiltrate sensitive data, can access the client's computer and even can redirect into other pages created by the adversary. And the impact will be faced by all users visiting the compromised page.	
Suggested Countermeasures	
It is recommended to: • Least Privilege: • Application Level Checks: • Bounds Checks:	
References	
https://www.imperva.com/learn/application-security/remote-code-execution/ https://www.lakera.ai/blog/remote-code-execution https://medium.com/@anandrishav2228/remote-code-execution-rce-an-in-depth-guide-with-practical-7082a7e17e97	

12. Proof of concept:

There is no Filtering of the file you can upload any file to the server after that execute that file and then code is executed on the server.