GOOGLE GURYERE

Ali M Abdelhamid AA2100274

KH4063CEM

Abstract

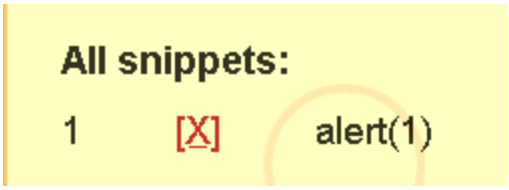
This report will contain the answer to the required questions as well as their mitigation process that happens after exploiting them. And then explain the flaws and the security posture as well as the security Impact, all this will be supported by screenshots of the vulnerabilities.

XSS Challenges

There are multiple XSS questions that are presented, the first one is whether you can upload a file that allows you to execute desired code onto the website's domain, the answer is yes, due to gruyere allowing the upload of HTML files and due to its low security posture. Here's an example: create a new text document with "<script>alert('hello') </script>" as the contents, then save it as an HTML file. The file is then uploaded to the gruyere, and it will create a new URL that if visited will execute the HTML file

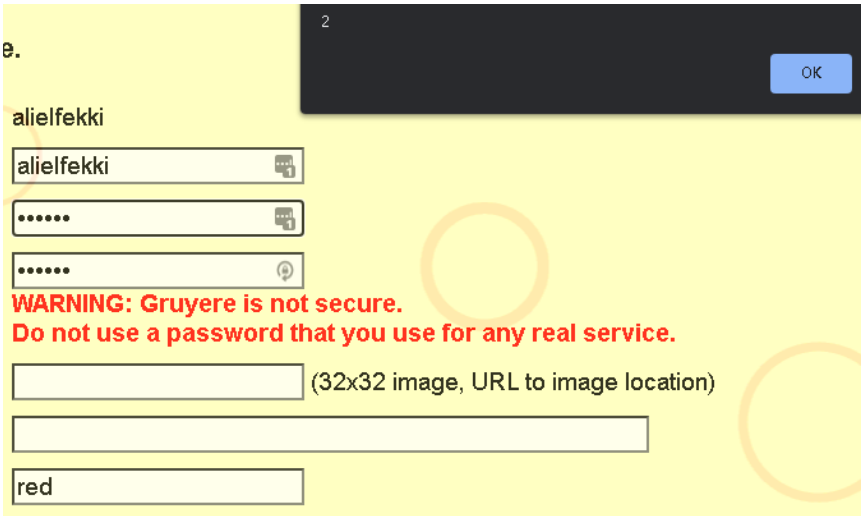
To fix a user uploading an HTML file that can be uploaded you need to host the content on another separate domain to prevent the script from accessing any content from the main domain. And to prevent the reflected XSS attack, you need to escape the user input that is displayed in error messages, using an autoescaping feature would be the simplest fix to this issue.

The second problem is to put a custom script on a snippet that will then view it back to a separate user, doing that is very simple, the only thing required to do is to write the script in the text area of the snippet, the script is going to be: "<script>alert(1) </script>" the result will be



To mitigate this issue, you need to use a certain tag called "_SanitizeTag" but the only issue is that this tag is case sensitive, and HTML is not, therefore typing in capital will still work. To fix this you need to use strict whitelisted text, tags, and attributes.

Following that is injecting script into the HTML aspect of the website and changing the color of a profile, you can easily do that by going to the profile tab then typing this "red' onload='alert (1)' onmouseover='alert (2)" now the text in the profile tab is viewed as red, and every time you refresh or hover over it, an alert pops up showing 2



To mitigate this issue all you need to implement is input validation, where it bypasses all double and single quotes to skip the function and print it as usual.

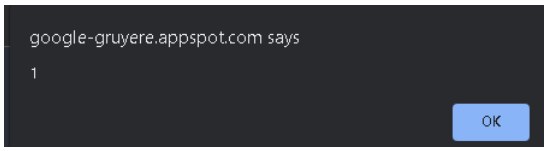
To exploit the next question which is an attack that utilizes a bug in the AJAX code of gruyere is to create a new snippet with the contents of

```
all <span style=display:none>"
+ (alert(1),"")
+ "</span>your base
```

then revisit the feed.gtl file which will view all the snippets that are currently on the site. To prevent this from happening, the usage of JavaScript eval must stop, since it's very dangerous. And if used a JSON parser should be used with it since it ensures the string does not contain any invalid content.

The next attack is creating a URL that when accessed a script will execute utilizing one of AJAX features. To do that you simply need to take the provided URL and modify it a little so it would like that: https://google-gruyere.appspot.com/4911980295498636398801269975204814599 18/feed.gtl?uid=%3Cscript%3Ealert(1)%3C/script%3E

And then prompt the user to click on it. It will then execute the script and the output will be



To prevent this, you need to make it certain that the browser does not interpret JSON code as HTML. To do that you need to modify the JS part of the code to implement JavaScript escapes \x3c and \x3e instead of <>

Client-State Manipulation

Elevation of Privilege

The first request is to change the account into an admin account, since it said any account, I chose to change "Brie". By visiting the editprofile.gtl you can see all the users and their profile information. There is a string called "is_admin" all you need to do is change that string from False to True. To do that you need to create a URL with the string "/saveprofile?action=update&is_admin=True&uid=alielfekki" then sign-out and resign in and you'll see a manage this server button due to being an administrator.

| Manage this server |

The problem here is that there is absolutely no validation towards whether this request is authorized or not, the fix for this is to check for authorization on the server right at the time where the request is made.

Cookie Manipulation

Now I need to get Gruyere to issue me a cookie of another user's account. To do that all you need to do is create a new account and set the username as "foo|admin|author" once you log in as this user, you will be granted access to user foo as an administrator, this can also be called an elevation of privilege attack. | Manage this server |

To fix this issue all that needs to be done is add restrictions in the username input tab and be careful how they are handled. In this particular case, the cookie parsing code is the issue since it allows malformed cookies.

Cross-site Request Forgery

In order to get a user to delete one of the snippets all they need to do is visit a malicious URL linked to Guryere, this URL will prompt guryere to delete on of their snippets. The URL is going to be: https://google-gruyere.appspot.com/4911980295498636398801269975204814599 18/deletesnippet?index=0

Further more, you can set the guryere website icon to this link so they would further believe it. On the other hand, to fix this issue, the /deletesnippet tag should be change to a POST request since this is a state changing action.

Cross Site Script Inclusion

An attack to read someone elses private snippets is also simple. All that is required to do is create an HTML file with these contents

```
<script>
function _feed(s) {
  alert("Your private snippet is: " + s['private_snippet']);
}
</script>
<script src="https://google-gruyere.appspot.com/4911980295498636398801269975204814599 18/feed.gtl"></script>
```

Once the user accesses the HTML file, the script located in feed.gtl will start to execute, and allows the attacker to read the victims snippets.

To fix this, you to first use an XSRF token to make sure that the JSON results that are containing confidential data are only returned to the original user's page.

Path Traversal

To access the secret.txt file all that needs to be is create a URL that leads directly to the text file. Most websites use /../ as an added security option but most browsers buffer that out since it doesn't provide any security level. The URL should be https://google-gruyere.appspot.com/4911980295498636398801269975204814599 18/../secret.txt. To fix that issue, you cannot change file permissions since the gruyere admins should still be able to access it. The best fix to this is to validate file paths and hide the path element like ../ and ~. On the other hand, to replace the secret.txt file all you need to do is create a new user called .. and then upload your new secret.txt file. The fix to this has been discussed earlier in the username input exploitation problem, where you must limit certain characters from being usernames.

Denial of service

To shut down the server there is a very simple DOS attack, add a simple /quitserver to the end of the URL, the default in all websites should require you to be an administrator to perform this but this case you don't. `Server quit.` To fix this issue all that needs to be done add /reset and /quitserver to the protected URLS that only admins can access.

Another simple way to DDOS the server is to overload it with request, all that should be done is create a new file and call it menubar.gtl and the contents of this site should be "[[include:menubar.gtl]]DoS[[/include:menubar.gtl]]" you should then utilize a path traversal attack that was used earlier and plant the file in the ../resources. An important though is after performing this exploit you'll need to press the reset button which is this URL https://google-gruyere.appspot.com/resetbutton/4911980295498636398801269975204814599 18

To fix this certain path traversal attacks should be combatted.

Code execution

In order to execute a code a copy of gtl.py should be made and then the extra suspicious code should be added, and the file should be reuploaded by a user with the username of .. or directly to ../gtl.py. you should then cause the server to quit using the method discussed above which is the serverquit method. And when it restarts the code will be run.

To fix this the privileges should be as minimal as possible since right now a non-administrator user read and written in the files of the server.

Configuration Vulnerabilities

Here the contents of a database should be read off a running server using the exploitation of the configuration vulnerability. To use that exploit the debug dump page called dump.gtl could be used. To visit this file, you should add /dump.gtl to the end of gruyere URL. There is a simple solution to this issue, which is simply disabling the debug features and deleting the dump.gtl file.

Another issue arises after doing that, an attacker can still undo the actions done above and still perform the attack, this can easily be exploited since the attacker can simply upload their own dump.gtl file from the computer and still perform the attack. To fix these 3 things should be implemented, 1st, only files that are part of the gruyere should be used as a template, next, set a whitelist regarding the types of files that could be uploaded. And lastly, do not store the uploaded files in the same location as the application source file. Even after all these fixes, an attacker can still perform the attack, due to a certain malfunction in the gruyere template, it reparses expanded variables especially when expanding a block, it parses it as a template then returns the variable to normal. In order to bypass the fixes, the attacker should add {{_db:pprint}} to their private snippet. To fix this the administrators should simply modify the code of the set template so it would never reparse the inserted variables.

AJAX vulnerabilities

There is a certain attack that can be made that will prevent users from seeing their private snippets on the main page. To exploit this attack, the attackers needs to create a new user called private_snippet and create at least a single snippet. The JSON response to this snippet would be a user's private snippet. The flaw here is that the JSON structure is no the best there is, and the AJAX code needs to be made sure that the data goes only where it's supposed to go.

The final issue here, is that there is a way to change the sign in link in the upper right corner of the page. To exploit this, you need to create a user called menu-right and then publish a private snippet containing this code “Sign in

| Sign up”

In order to fix this you have to ensure that there is no conflict in the strings such as the prefix for user values like id=”user=” once the user clicks on the sign in button they will be redirected to evil.example.com