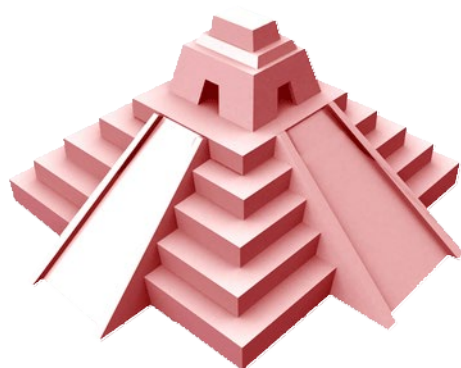


WEB PENETRATION TEST REPORT

BrainScribe Consulting



Author: Thu Duong

TARGET: GRUYERE

EXECUTIVE SUMMARY

This lab shows how a web application can be attacked using common security vulnerabilities, like Cross-Site Scripting vulnerabilities (XSS), Client-State Manipulation, Cross-Site Request Forgery (XSRF) and Cross-Site Script Inclusion (XSSI) by playing the role of a malicious hacker to find and exploit the security bugs on Gruyere web application.

Scope of work

This lab is conducted on Gruyere – a web application that allows users to publish snippets of text and store assorted files.

Project Objectives

By exploiting the Gruyere web application, we learn how hackers find security vulnerabilities and how they exploit web applications.

Timeline

The exploiting process is on Tuesday, March 8th, 2022, at 6:28 PM.

Summary of Findings

This lab found and exploited 10 vulnerabilities on Gruyere web application:

- File Upload XSS
- Reflected XSS
- Stored XSS
- Stored XSS via HTML Attribute
- Stored XSS via AJAX
- Reflected XSS via AJAX
- Elevation of Privilege
- Cookie Manipulation
- Cross-Site Request Forgery (XSRF)
- Cross-Site Script Inclusion (XSSI)

METHODOLOGY

To find out Gruyere's vulnerability, we have conducted **Cross-Site Scripting (XSS) attacks, Client-State Manipulation attacks, Cross-Site Request Forgery (XSRF) attacks, and Cross-Site Script Inclusion (XSSI) attacks.**

Cross-site scripting (XSS)

It is a way of injecting code (typically HTML or JavaScript) into the contents of a web page that isn't under the attacker's control. An attacker injects code into a victim's browser when they view such a page. Consequently, the attacker has been able to sidestep the browser's same origin policy, which allows him to steal private information from the victim.

Client-State Manipulation

These attacks take advantage of using cookies. As the HTTP protocol is stateless, a web server cannot automatically identify whether two requests are coming from the same user. Therefore, cookies were invented to solve this problem. Web sites generate cookies as part of their HTTP response (an arbitrary string). When the browser receives the cookie, it automatically sends it back to the site when the request is made again. Web sites can save state using cookies. Cookie technology allows Gruyere to remember who is logged in. Since the cookie is stored on the client side, it is susceptible to manipulation.

Cross-Site Request Forgery (XSRF)

Browsers always send along any cookies it has stored for a particular site, no matter where they are coming from. Also, web servers cannot tell the difference between a request initiated by a deliberate action by

the user and one initiated by a browser without user intervention. So, even if an authentication cookie is included in a request for some action, the site cannot determine whether the action was consciously initiated by the user. This can be exploited by an attacker so they can get the server to perform actions the user did not intend.

Script Inclusion (XSSI)

Browsers stop one domain's pages from viewing pages from other domains. They do not, however, prevent domain sites from referencing resources from other domains. They enable images to be rendered from other domains and scripts to be performed from other domains. A script that is included does not have its own security context. It runs in the context of the page that includes it in terms of security. As a result, any user information in that script will "leak."

Moreover, we found and exploited security flaws of Gruyere by using **Black Box Hacking** and **White Box Hacking**.

In **Black Box Hacking**, we try to uncover security issues by exploring with the program and altering input fields and URL parameters, causing application failures, and looking at HTTP requests and responses to guess server behavior in black box hacking. We don't have access to the source code but knowing how to examine it and seeing http headers is useful.

In **White Box Hacking**, we have to access to the source code and use automated or manual analysis to find bugs.

DETAIL

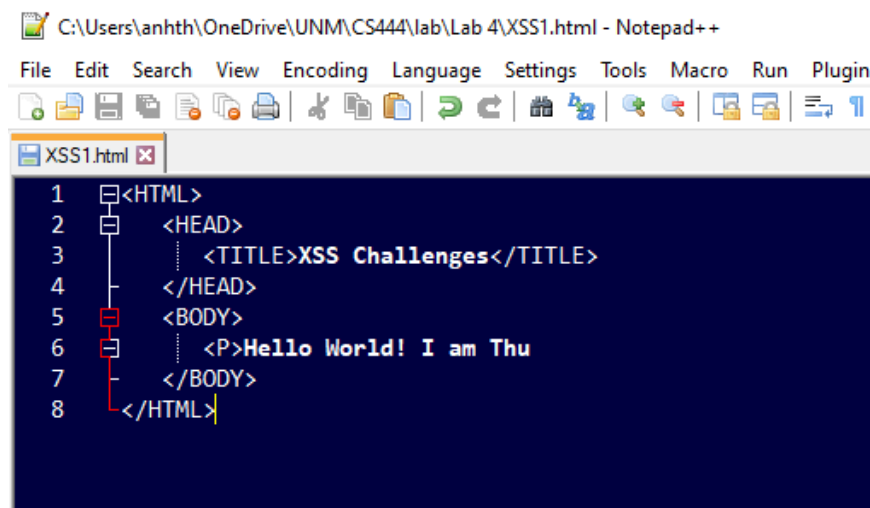
FINDINGS

VULNERABILITIES

1. File Upload XSS

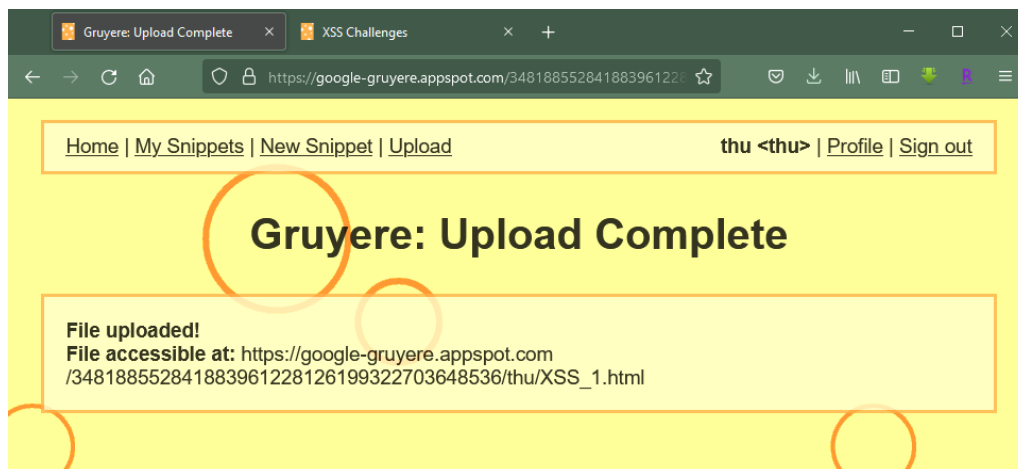
If we can get our own script to execute on a page when it's viewed by another user, we basically detect an XSS vulnerability. So, I tested whether we could upload a file that allows us to execute arbitrary script on the `google-gruyere.appspot.com` domain.

First, I created a html file with script as follow:



```
C:\Users\anhth\OneDrive\UNM\CS444\lab\Lab 4\XSS1.html - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugin
XSS1.html
1 <HTML>
2 <HEAD>
3 <TITLE>XSS Challenges</TITLE>
4 </HEAD>
5 <BODY>
6 <P>Hello World! I am Thu
7 </BODY>
8 </HTML>
```

Then, I uploaded HTML file on Gruyere.



Whenever the user enters that link, it prompts to the page like this.

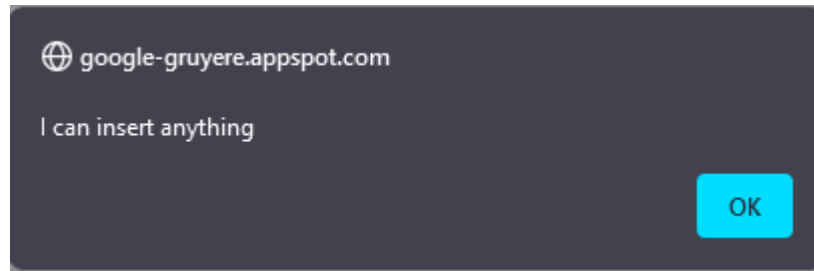


2. Reflected XSS

Reflected XSS attacks usually occur when the server inserts the attack verbatim or incorrectly escapes or sanitizes the response from the request itself (frequently the URL). Attacks are initiated when the victim visits a malicious URL created by the attacker.

To conduct a reflected XSS attack, I created a URL that when victim clicks on, it will execute a script.

[https://google-gruyere.appspot.com/348188552841883961228126199322703648536/%3Cscript%3Ealert\(%22I%20can%20insert%20anything%22\)%3C/script%3E](https://google-gruyere.appspot.com/348188552841883961228126199322703648536/%3Cscript%3Ealert(%22I%20can%20insert%20anything%22)%3C/script%3E)



3. Stored XSS

In a **stored XSS** attack, the attacker stores the attack in the application (e.g., in a snippet) and the victim triggers the attack by browsing to a page on the server that renders the attack, by not properly escaping or sanitizing the stored data.

To find a stored XSS, I put a script in a place where Gruyere will serve it back to another user.

A web form with a yellow background and orange circular patterns. The title is 'Add a new snippet.' Below it is a text area containing the code '<script>Hello</script>'. At the bottom left, there is a message: 'Limited HTML is now supported in snippets (e.g., , <i>, etc.)!'. To the right of this message is a 'Submit' button.

Most recent snippets:

Cheddar Gruyere is the cheesiest application on the web.
Mac [All snippets](#) [Homepage](#)

thu Hello
[All snippets](#) [Homepage](#)

Brie Brie is the queen of the cheeses!!!
[All snippets](#) [Homepage](#)

4. Stored XSS via HTML Attribute

We can also do XSS by injecting a value into an HTML attribute. Now, we try to inject a script by setting the color value in a profile.

I choose the color preference for the username by:

```
green' onload='alert(1)' onmouseover='alert(2)'
```

Gruyere: Profile

Edit your profile.

User id:

User name:

OLD Password:

NEW Password:

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

Icon: (32x32 image, URL to image location)

Homepage:

Profile Color:

Private Snippet:

And the result is:



5. Stored XSS via AJAX

We find an XSS attack that uses a bug in Gruyere's AJAX code. The attack should be triggered when you click the refresh link on the page.

To exploit, I put this in my snippet.

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

Run `curl` on `https://google-gruyere.appspot.com/348188552841883961228126199322703648536/feed`.
gt1 and look at the result.

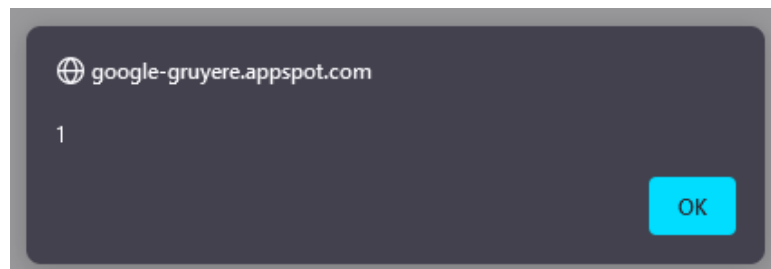
```
_feed(( { "private_snippet": "all your base" ,"cheddar": "Gruyere is the cheesiest application on the web." ,"thu":
"all your base" ,"brie": "Brie is the queen of the cheeses!!!" } ))
```

It includes each user's first snippet into the response. This entire response is then evaluated on the client side which then inserts the snippets into the document.

6. Reflected XSS via AJAX

We find a URL that when clicked on will execute a script using one of Gruyere's AJAX features.

[https://google-gruyere.appspot.com/348188552841883961228126199322703648536/feed.gtl?uid=%3Cscript%3Ealert\(1\)%3C/script%3E](https://google-gruyere.appspot.com/348188552841883961228126199322703648536/feed.gtl?uid=%3Cscript%3Ealert(1)%3C/script%3E)



It renders as

```
_feed((["<script>alert(1)</script>"])))
```

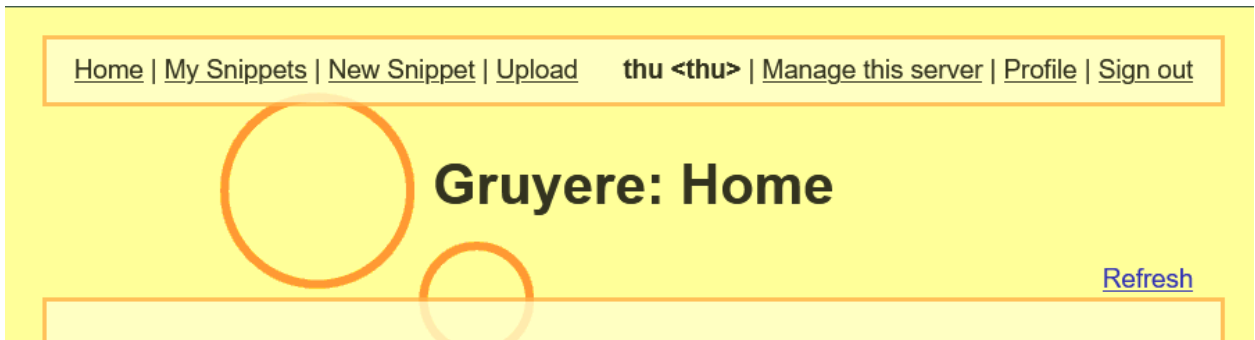
So, it executed the script. There is a bug due to Gruyere returning all GTL files as content type text/html and browsers are quite tolerant of the HTML files they accept.

7. Elevation of Privilege

We can convert our account to an administrator account by issuing request:

https://google-gruyere.appspot.com/348188552841883961228126199322703648536/saveprofile?action=update&is_admin=True

After visiting this URL, your account is now marked as an administrator, but your cookie still says you're not. So, sign out and back in to get a new cookie. After logging in, notice the 'Manage this server' link on the top right.



8. Cookie Manipulation

To conduct cookie manipulation, we got Gruyere to issue a cookie for someone else's account.

Firstly, we created a new account with username "`foo|admin|author`".

When we log into this account, it will issue us the cookie "`hash|foo|admin|author||author`" which logs us into **foo** as an administrator.

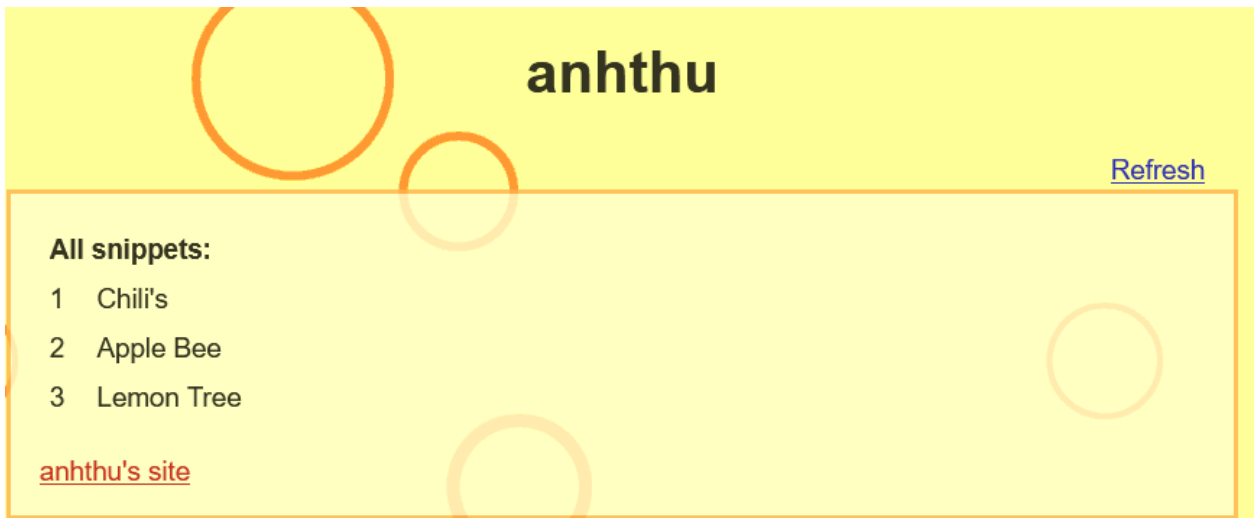


9. Cross-Site Request Forgery (XSRF)

To attack XSRF, we tried a way to get someone to delete one of their Gruyere snippets.

Firstly, we create another account with username "anhthu".

In this username, create 3 new snippets: "Lemon Tree", "Apple Bee" and "Chili's"



Then, lure this user to visit a page that makes the following request:
<https://google-gruyere.appspot.com/348188552841883961228126199322703648536/deletesnippet?index=0>

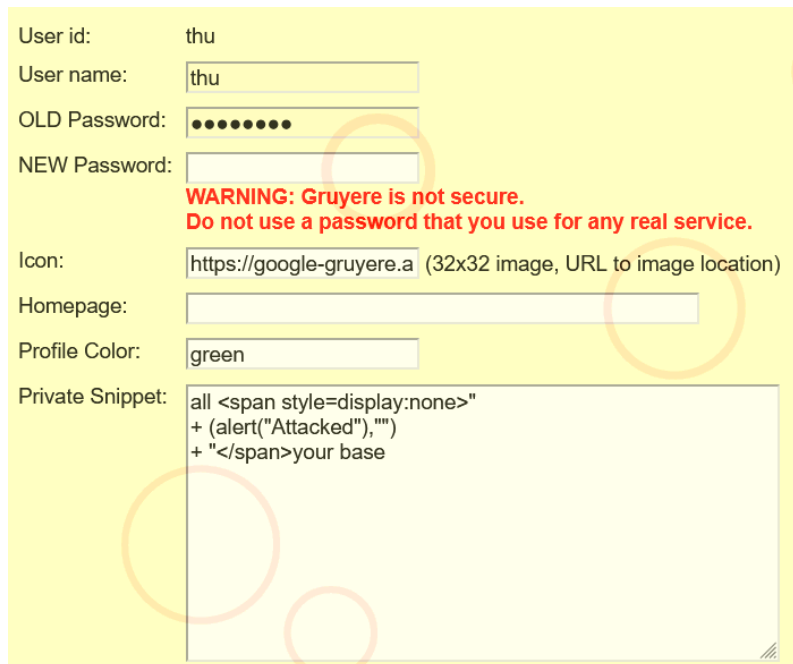
Now, the snippet "Chili's" is deleted.



10. Cross-Site Script Inclusion (XSSI)

To do XSSI attack, we find a way to read someone else's private snippet using XSSI. Basically, we create a page on another web site and put something in that page that can read the other's private snippet.

This is private snippet of user account "thu" and the message is "Attacked". This account will play as a victim.



User id: thu

User name:

OLD Password:

NEW Password:

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

Icon: (32x32 image, URL to image location)

Homepage:

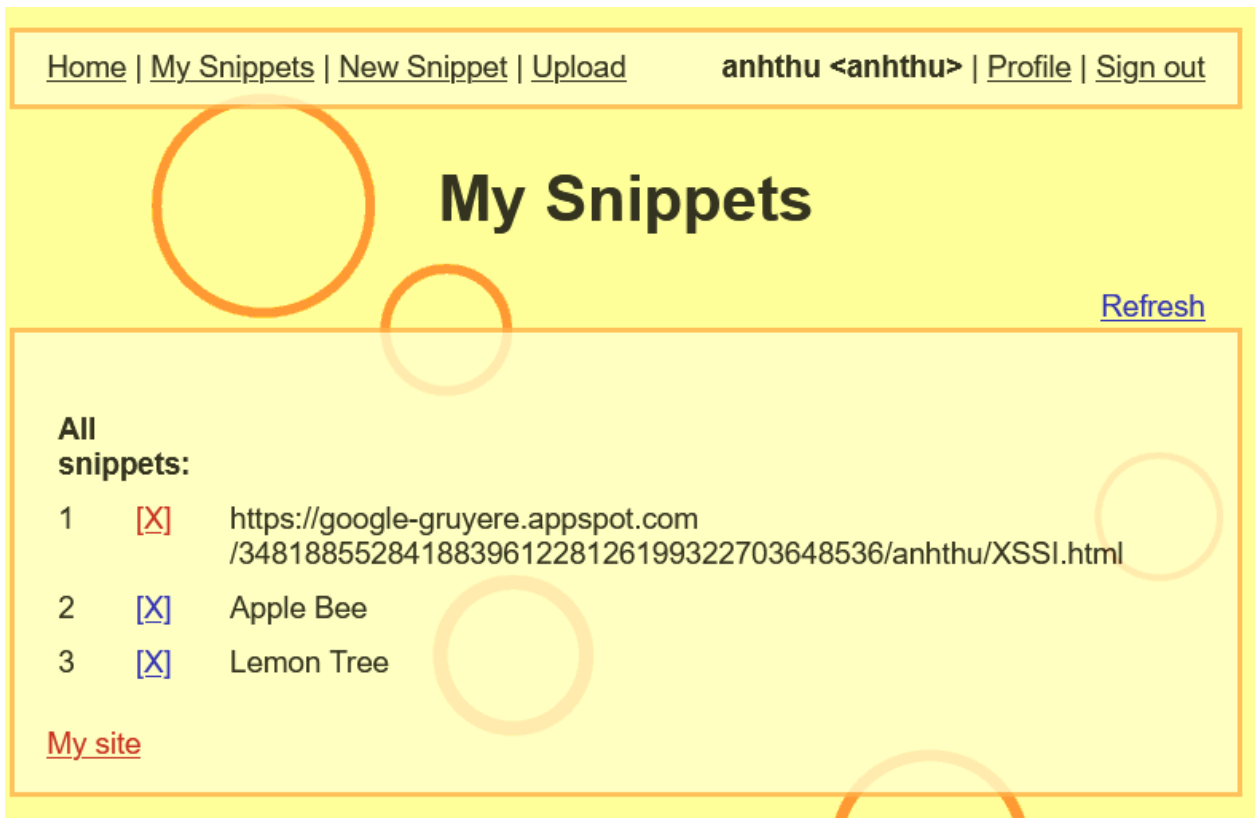
Profile Color:

Private Snippet:

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

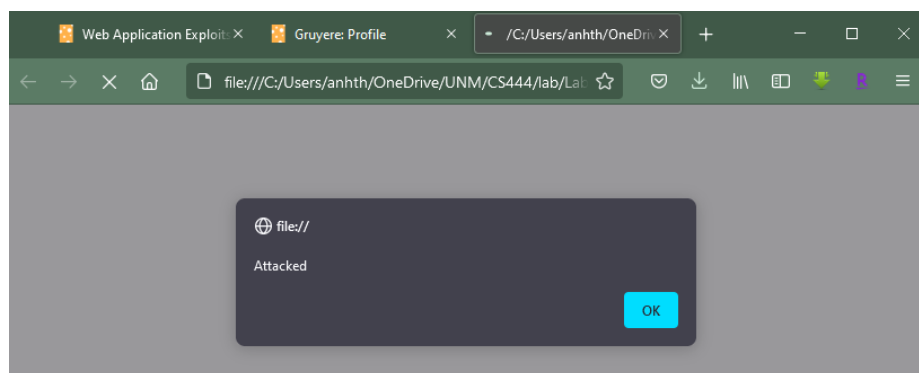
On the user account "anhthu" who plays as attacker, we create a .html in our home directory and double click on it to open in a browser

```
XSSI.html X
1  <script>
2  function _feed(s) {
3      alert("Your private snippet is: " + s['private_snippet']);
4  }
5  </script>
6  <script src="https://google-gruyere.appspot.com/348188552841883961228126199322703648536/feed.gt1"></script>
```



We copy the link to snippet of user “anhthu”, then lure the victim “thu” to click on that.

When the script in feed.gtl is executed, it runs in the context of the attacker's web page and uses the _feed function which can do whatever it wants with the data, including sending it off to another web site.



Now, the attacker can see the private snippet of victim.