cs3235:9: - Laboratory #9 for 30th of October, 2019

I suggest you use any extra time after completing this laboratory to meet up with your project group members, and work on your project.

1 Lab 9, part 1: XSS

In this laboratory, you will be investigating XSS attacks. Your final goal will be to discover passwords for *other* users of a password protected web site which is intended to get feedback comments.

Open the web browser and go to

`https://hugh.comp.nus.edu.sg/cs3235/lab9/commentslab9-1.php`

and use the interface to enter a comment. This is a mindless interface that asks you to enter your ID, a password and a comment.

It is unsafe to use your proper password for this laboratory, so use the secondary password you were sent in the email that you got before the course started. You can also experiment using one of the 9 users called test1 to test9, with passwords test1 to test9. You can use these users to try out your attacks.

Once you have entered a comment you can view comments left by other people in cs3235, but be warned - the web pages are open to XSS injections, and they will be putting in their own injections.

Experiment with the interface to discover if it has controls on the input form data or the output web page, by trying to enter Javascript or HTML in the comments field.

```
<h3> Hi there </h3>
<script> alert('Hello World!'); </script>
```

There are comment locations for each of you, indexed by your matriculation ID. If you forget your password, talk to your tutor to get it again.

I wrote an attack as user `hacker` (who no longer exists), and the result of this attack was in the comment for user `test` (who no longer exists). When the user `hugh` looked at the comment left by `hacker`, nothing much seemed to happen, but actually the `hacker` wrote the URL that `hugh` had used to the comment for user `test`. Have a look at the comment for the user `test`. You will notice that it contains the password for `hugh`.

```
https://hugh.comp.nus.edu.sg/cs3235/lab9/commentslab9-1.php\
        ?matric=hugh&pass=HighlySecret&method=2&user=hacker
```

The attack seen in the test comment was performed when `hugh` looked at the comment. The comment was a small snippet of (in this case) Javascript, which constructed a URL `SAVEURL` which was the same sort of URL used for writing a comment to user `test`. The contents of the comment were the calling document's URL. The `SAVEURL` was then forced to the current `document.location`[1] of the current browser document, which meant that the (constructed) comment was written into the comment for the user `test`. Whew! I got lost writing that!

The possible steps for an attack based on this strategy are:

1. Construct a javascript `<script>...</script>` comment, which, if someone views it, will force the current `document.location` of the current browser document to a crafted URL.
2. The crafted URL should write a comment for (another) specified user - perhaps your lab partner, or one of the extra users like `test14`.
3. The comment in this crafted URL could be the original document's URL (which is found in the Javascript object `document.URL`).
4. As a result of these steps, when someone views this comment, their URL will be written as a comment somewhere else. In our case the URL contains a password, but in other cases you might retrieve document cookies or other information from the document.

1.1 Step 2

Investigate the URLs that are needed to write to a comment in this system, and see if you can write a URL directly into the browser which would write something into a comment for a specified user like test9 (i.e. without using the interface). You can see sample URLs by looking at the browser URL line as you enter a comment. To make all this easier, I have used the GET method, rather than POST, so you can see all the information in the (browser) URL.

Once you know what the URL to write a comment looks like, construct some Javascript which forces the current `document.location` to a crafted URL, perhaps writing the comment "`XXXXX`" to a specific user. Put this Javascript into your comment field, and then view it. If it works OK, when anyone views the comment, it should write "`XXXXX`" to the user.

If you have success with this, modify it so that instead of writing "`XXXXX`", your script writes the original document's URL. You should have succeeded in constructing an XSS attack!

The Javascript constructs you might need include ones like:

```
<script> ... </script>              # Javascript is encased in these tags
document.location = ...             # This will cause the browser to access ...
document.URL                        # This returns the URL of current document
'sss' + 'yyy'                       # results in 'sssyyy'
escape(document.URL)                # protects document.URL, escaping special characters like &
```

[1] See `http://www.w3schools.com/jsref/dom_obj_document.asp`.

Once you have your attack working, both you and your lab partner should access the cs3235 grading website to enter in your attack string, and a brief description of how it works. In your description, you should state what sort of XSS attack (Persistent or non-persistent) is explored in this lab:

`https://hugh.comp.nus.edu.sg/cs3235/lab9/gradeslab9-1.php`

You will only see your own attack, not everyone else's.

2 Lab 9, part 2: SEED CSRF laboratory

Have a look at the SEED CSRF lab, found at

`http://www.cis.syr.edu/~wedu/seed/Labs_16.04/Web/Web_CSRF_Elgg/`

There is a lot in this laboratory, but I would like you to just try task 1 and then task 2, a CSRF attack using GET requests. As you work through it, come up with a series of clear instructions that would let others repeat your attack. As you investigate, think about how you could change the source code of the web application to prevent your attack.

Access the cs3235 grading website, and enter in a detailed description of how you did the CSRF attack, and describe in detail how you would modify the application to prevent the attack succeeding:

`https://hugh.comp.nus.edu.sg/cs3235/lab9/gradeslab9-2.php`