DOTA: Laboratory #4

To get information useful for this laboratory, you could look at `https://www.w3schools.com/sql/`. If you finish the laboratory early, I suggest you use the extra time to meet up with your project group members, and work on your project.

1 Lab 4, part 1: SQL injection

In this laboratory, you will be investigating SQL injection attacks. Your final goal will be to change your grade for Lab 4 part 1. It is currently 0.

Open the web browser and go to

`https://hugh.comp.nus.edu.sg/DOTA/lab4/gradeslab4-1.php`

to see an application which allows the office staff at NUS to see your current gradings. There is a mindless interface that asks them to enter their userID and password, and then put in your ID, and then it will return your marks so far. This web page accesses your grade information in a SQL (actually mySQL) database, and unfortunately was written by someone with a brain the size of a pea[1]. As a result, the web page will be open to SQL injections.

I have put in a bit of debugging for you, which actually shows you the SQL command that will be executed when the office staff access your grades, although of course this does not normally happen in practice. I have also set things up so that your usercode and password allows you to pretend you are part of the office staff.

The SQL commands you might need might include ones like:

```
SELECT `field` FROM `database`.`tablename` WHERE field='value';
UPDATE `database`.`tablename` SET `field1` = 'value' WHERE field2 = 'value';
```

Experiment with the mindless interface, until you can see everyone else's marks and also change your own mark. Please do not change anyone else's mark, or do anything malicious to the database[2]. Please give yourself whatever mark you believe that you deserve.

[1] Me, once again.

[2] No DROP TABLES etc.

2 Lab 4, part 2: 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. It might be easier to team up with a laboratory partner to do this lab.

Open the web browser and go to

`https://hugh.comp.nus.edu.sg/DOTA/lab4/commentslab4-2.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. You can see and control all the parameters passed to the web application in the URL in the browser - this web application uses GET instead of POST.

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 19 users called test1 to test19, with passwords test1 to test19. Use these users to try out your attacks. Once you have entered a comment you can view comments left by other people in DOTA, 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`.

```
http://hugh.comp.nus.edu.sg/DOTA/lab4/commentslab4-2.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`[3] 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.

[3] See `https://www.w3schools.com/jsref/dom_obj_document.asp`.

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.

2.1 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.

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 &
```

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

`https://hugh.comp.nus.edu.sg/DOTA/lab4/gradeslab4-2.php`

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

3 Lab 4, part 3: Services, issues

In this part of the laboratory, you will be investigating DoS attacks. Your final goal will be to discover how difficult it is to deny people access to a web server.

The first step is to observe that there is a web server running on each of the lab machines. The web server is commonly called `httpd` (for the HTTP Daemon) or apache (for the apache daemon). Verify that your web server is running by running a browser against the URL `http://localhost/`, or `http://192.168.100.XXX/` (from another machine). If everything is working fine you should see an initial web page. How fast does your web server serve up the page?

You might also want to look at the processes and threads in the computer, and count the webserver threads:

```
a00XXXX@host$ ps -eLf | grep apache
```

Try out the slow loris attack

Download `slowloris.pl`: and change its permissions to be "executable", before running it against the web server on your

```
a00XXXX@host$ wget https://hugh.comp.nus.edu.sg/DOTA/lab4/slowloris.pl
a00XXXX@host$ chmod +x slowloris.pl
a00XXXX@host$ ./slowloris.pl -dns localhost
or (from another host)...
a00XXXX@host$ ./slowloris.pl -dns 192.168.100.XXX
```

While the attack is going on, try to browse/refresh the web page. Look at the processes/threads in the computer running the web server, and count the processes/threads. Run wireshark while you are doing a `slowloris` attack.

Assessment

When you are finished, consider the following questions:

1. What was the web server you attacked. Was it a recent version of the web server? What are the market shares of the top 3 web servers in the world?
2. The webserver was set up and started by the root user on the Ubuntu machines in the lab. Why are you (a normal unix user) not able to start a web server on port 80? What is the "security" issue?
3. Why is the web server running as www-data, and not as the user root? What is the "security" issue?

When you are finished with the laboratory, access the DOTA grading website, and enter your username, password, and the answers to the previous questions (1-3), in order:

`https://hugh.comp.nus.edu.sg/DOTA/lab4/gradeslab4-3.php`

4 Lab 4, part 4: (OPTIONAL - not marked) SEED CSRF laboratory

Have a look at the SEED CSRF lab, found at

`https://seedsecuritylabs.org/Labs_20.04/Web/Web_CSRF_Elgg/`

There is a lot in this laboratory, but I would like you to just try Section 3.2, 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. This attack would have to be done on the SEED VM.

Access the DOTA 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/DOTA/lab4/gradeslab4-4.php`