ECE 568

Lab #3: Web Application Security

Overview

The purpose of this lab is to familiarize yourself with a number of the most common web application vulnerabilities. You will need to spend some time reading and understanding some material on JavaScript, the HTTP DOM model and SQL.

You may work on this lab individually, or in groups of two. Your code must be entirely your own original work. The assignment should be submitted by **11:59pm on Friday, March 14, 2025**.

Please submit a README file that contains your name(s), student number(s), and email(s) at the top, along with explanations for each exercise.

You should also submit text files that contain your answers for each of the seven exercises. Your files should contain your answer at the top of the file. (Please place your explanations in the *README* file, **not** in in *part#.txt* files.) If your files are not formatted as instructed, you may lose marks.

Please submit the following files:

```
submitece568s 3 README part1.txt part2.txt part3.txt part4.txt part5.txt part6.txt part7.txt
```

README format:

```
#first last, studentnum1, e-mail1
#first last, studentnum2, e-mail2

Part 1 Explanation:

...
```

Your solutions will be tested using the Firefox browser on the ECF machines. While your solutions should be less architecture-dependent than Lab 1, please make sure to test your solutions on these machines prior to submission.

Background

For this lab, we will be using the WebGoat environment; it is an open-source tool that allows you to explore a variety of web vulnerabilities, several of which we will be using in this lab. Begin by creating a local working directory, and copy the **webgoat.jar** file provided from Quercus into the directory. The file webgoat.jar can also be found on the ECF machines in:

```
/n/share/copy/ece568s/lab3/webgoat.jar
```

The WebGoat application creates a private web server and database for you to experiment with. When it runs, the application starts listening on a local port; you will need to pick a unique value for your use. Make sure to check your Java SDK version is 1.7 or 1.8:

```
$ java -version
openjdk version "1.8.0_432"
OpenJDK Runtime Environment (build 1.8.0_432-b06)
OpenJDK 64-Bit Server VM (build 25.432-b06, mixed mode)
$
```

The following examples use **port 8090** as an example, try other port numbers (larger than 1024) if the port you picked is in use by someone else:

```
$ java -jar webgoat.jar -httpPort 8090
```

If you are working on this lab remotely and running this on ECF, then you will need to use SSH to create a tunnel; this will allow you to access the local web service from your own computer:

```
$ ssh username@remote.ecf.utoronto.ca -L 8090:localhost:8090
```

(If you are using Windows, to perform port-forwarding, you can use WSL or a program like Putty, see https://undergrad.engineering.utoronto.ca/undergrad-resources/engineering-computing-facility-ecf/remote-access/). You can then connect to the service from your local web browser by connecting to:

http://localhost:8090/WebGoat/

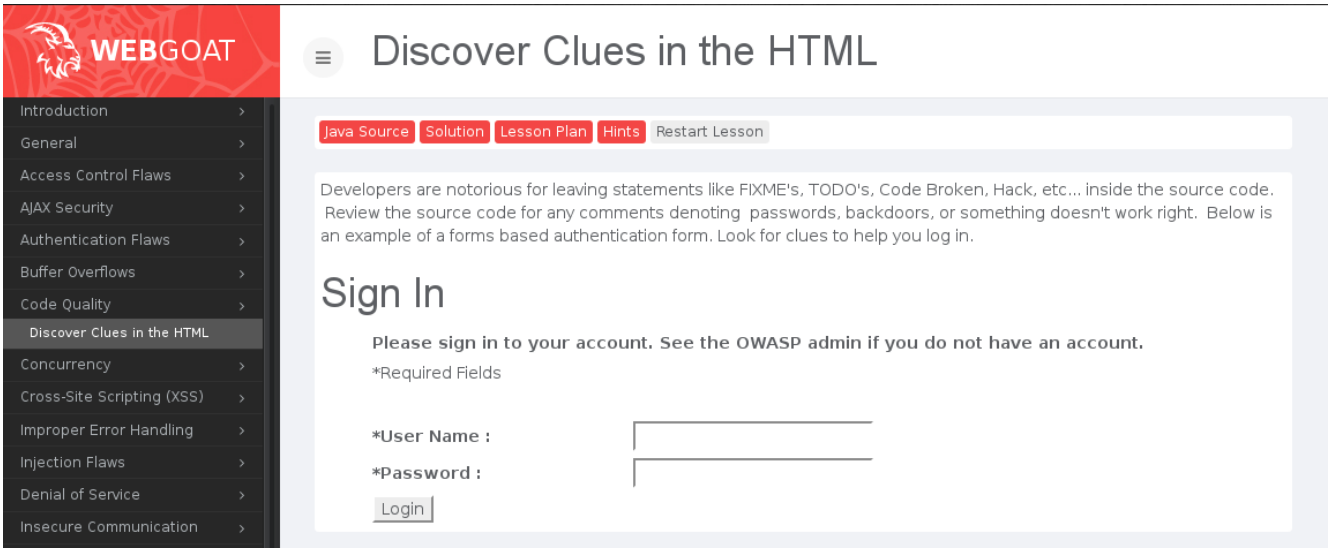
and then logging in as a username of “webgoat” with a password of “webgoat”.

While WebGoat will not expose ports to outside users, it will allow anyone with access to ECF to connect to your service if they are connected to the same workstation – and, by its nature, this application is created with security holes, including some which may be unintentional: **you should not leave this service running when you are not actively using it**. (As an alternative, you may opt to install the Java runtime environment on a personal machine and run this application there.)

Getting Started

There are two registered users on the WebGoat instance that you will need for part2 and part3: {username: **guest**, password: **guest**} and {username: **victim**, password: **victim**}. You should log in as the guest user when performing the attacker’s action, while log in as the victim user when performing the victim’s action. For other parts, you only need to login as the victim user to complete the task.

To get oriented to WebGoat, start by going to the Discover Clues in the HTML lesson under “Code Quality”:



Select either “Inspect Element” or “Show Page Source” (depending on your web browser) to view the source of the web page. You can find the developer’s comments, with the solution for this first exercise, within the code:

You need to now complete the following seven exercises to complete this lab. The total mark is 100. Total marks for each individual part are indicated below. Good luck!

Part 1: Phishing with XSS [15 marks]

Navigate to the "Cross-Site Scripting (XSS)" lessons and select "Phishing with XSS". Read the description and complete this exercise.

Your goal is to generate a fake "login" form, use XSS to display it, and then submit its contents to the URL provided in the exercise description (Substituting the port number you selected):

http://localhost:**8090**/WebGoat/catcher?PROPERTY=yes&phishingUsername=&phishingPassword=

Hint:

You may find the following references useful:

> https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
> https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

In particular, look at the DOM reference for how to access HTML form data from within JavaScript ("document.forms[0].____.value").

What to Submit:

Submit the full code you create in part1.txt. Your explanation should provide a brief description of the code and the vulnerability you used, and how you phished the victim. For ease of marking, when creating your "login" form, please use the following HTML id values for your form fields:

username text field:	`ece568-25s-username`
password text field:	`ece568-25s-password`
submit button:	`ece568-25s-submit`

Part 2: Stored XSS Attacks [20 marks]

Navigate to the “ECE568 Lab” lessons and select “Stored XSS Attacks”.

In this section, there are two HTML forms on the web page, the “Send Message” form and the “Transfer Money” form.

To complete the attack, you need to first log in as the “guest” user and send a message that contains your attack script to the “victim” user. When the victim user views the message, your script will be executed and carry out the following attacks 1) read the secret of the victim that is hidden inside the “Transfer Money” form, and 2) send the secret back to the attacker (the “guest” user) using the “Send Message” form after the “victim” click on the transfer button.

Hint:

You can set the form parameters in a URL, as shown here:

http://localhost:**8090**/WebGoat/attack?Screen=SCREEN_ID&menu=MENU_ID

(You can append the message form parameters to the URL, and you should modify the **8090** to match your own port number.)

When you are successful you should receive a message of “**Congratulations. You have successfully completed this lesson.**”. You may log in to the guest user account to verify if your attack succeeds. The victim should automatically send the stolen secret back to the guest.

What to Submit:

You should provide the XSS payload that you designed to steal the secret in the **part2.txt**. You should also include the content of the stolen secret. Your explanation for this part in **README** should indicate how the payload works.

Part 3: Client-side CSRF Attacks [20 marks]

Navigate to the “ECE568 Lab” lessons and select “Client-Side CSRF”.

In this section, a script (js/client_csrf.js) inside the web page is vulnerable to CSRF attacks. You should attack the “victim” as the “guest” user. Your goal is to modify the transfer amount to 9999 and the receiver of the money to “guest” whenever the victim tries to make a transfer.

You can assume that the victim will click the “Transfer Money to a User” button first, check every message in the message list within the opened window, and transfer the money using the transfer form on the opened window when they try to make a transaction. However, you cannot assume the victim will willingly send the money to the guest. Therefore, they won’t directly type the “guest” into the “Send to” field. You shouldn’t directly include the value of the secret inside your message (script) either since it is unknown to the attacker.

For stealthiness, the victim should not notice changes in their transaction receiver and amount before the attack is complete. Therefore, your script should not change the value attributes of these two inputs.

Hint:

Please inspect `js/csrf_transfer.js` within the opened window to see how the two windows exchange messages.

What to submit:

You should provide the message body (script) that the guest user sends to the victim in **part3.txt**. The marker follows the same attack model as the description above. It will send the message to the victim as a “guest”. Then, it will login as the “victim” to check the message and complete the transaction. Your explanation for this part in **README** should include how the script works.

Part 4: CSRF Token By-Pass [20 marks]

Navigate to the “Cross-Site Scripting (XSS)” lessons and select “CSRF Token By-Pass”. In this section, you need to successfully complete the “transfer” action. The “transfer” action has two steps and now requires you to start the transfer, receive a token value and then provide that random value while fetching the second URL.

Hint

There are a number of ways you could approach this problem. As a hint, you may wish to create two iframes, and then write some JavaScript to load and read the contents of those frames.

If you are successful, a green checkmark and the “congratulations” message will appear. It should be triggered by clicking/reading the message submitted. You may need to refresh/reopen the original lesson page to see the green checkmark and congratulations message.

What to Submit

You should provide the exploit you craft in **part4.txt**, and your explanation for this part in **README** should indicate the field that’s exploitable and how/why you were able to defeat the token by-pass.

Part 5: String SQL Injection [5 marks]

Navigate to the “Injection Flaws” lessons and select “String SQL Injection” (not the “LAB...” lessons).

For the next three parts, you may find it useful to review some of the SQL syntax for the database engine that WebGoat uses:

> https://hsqldb.org/doc/guide/sqlroutines-chapt.html

Additionally, you may find some useful SQL injection tips here:

> http://www.sqlinjection.net/

Hint

This exploit should be fairly simple. You are expected to see a green completion mark and the “Congratulations” message as soon as you click the “Go!” button.

What to Submit

You should include the full content of the field that you entered in part5.txt.

Part 6: Database Backdoors [10 marks]

Navigate to the “Injection Flaws” lessons and select “Database Backdoors”. This section has two parts.

For the first part, you must find an exploit that will change the salary of user 101 to $23 (twenty three dollars). As a hint, you should look at the “update” SQL command.

For the second part, you must find an exploit that will insert a database “trigger”. This trigger will stay resident inside the database, and will automatically alter the database for you. Create a trigger that will automatically change the email of any new user entry to “**ece568-25s@utoronto.ca**”. The WebGoat lesson plan will present you with the proper format for the trigger once you complete the first part.

Hint

You are expected to see a green completion mark and a “Congratulations” message once you successfully submit the first part and the second part of the SQL queries.

What to Submit

Your answer should include the full contents of what you entered in both parts. The first line in **part6.txt** should be your answer for the first part, and the second line for the second part.

Part 7: Blind Numeric SQL Injection [10 marks]

Navigate to the “Injection Flaws” lessons and select “Blind Numeric SQL Injection”.

You will need to find a SQL injection vulnerability that allows you to find the PIN for credit card 1111222233334444. (Please ignore the cc_number on webgoat)

What to Submit

You should provide the value of the PIN in **part7.txt**. Your explanation should include the SQL injection you used and a brief description of how you found the PIN. Do not brute force.

Learning More

The remainder of the WebGoat package covers examples of dozens of other frequently-encountered web exploits. While it is beyond the scope of this assignment, my hope is that you will find some of them interesting to explore at some later time. WebGoat has had active development for the past few years, and is constantly expanding to include new tutorials; you can download the latest version here:

https://github.com/WebGoat/WebGoat