ECE568 – Computer Security

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 11:59:59pm on Sunday, March 17th. 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 eight exercises. Your files should contain your one- or two-lined answer at the top of the file.

```
submitece568s 3 README part1.txt part2.txt part3.txt part4.txt \
part5.txt part6.txt part7.txt part8.txt
```

README format:

```
#first1 last1, studentnum1, e-mail1
#first2 last2, studentnum2, e-mail2

Part 1 Explanation:
...
```

Note: Please put your explanations in the README, not in the part#.txt files. If your files are not formatted as instructed, you may lose marks.

We have provided a submission checking script to help ensure that the automarker will process your files correctly:

```
/n/share/copy/ece568s/bin/check568_lab3 <your local directory of files>
```

Your solutions will be tested using the Firefox browser on the ECF machines. 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/bin/lab3`

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. The following commands use port 8090 as an example:

```
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.) You can then connect to the service from your local web browser by connecting to:

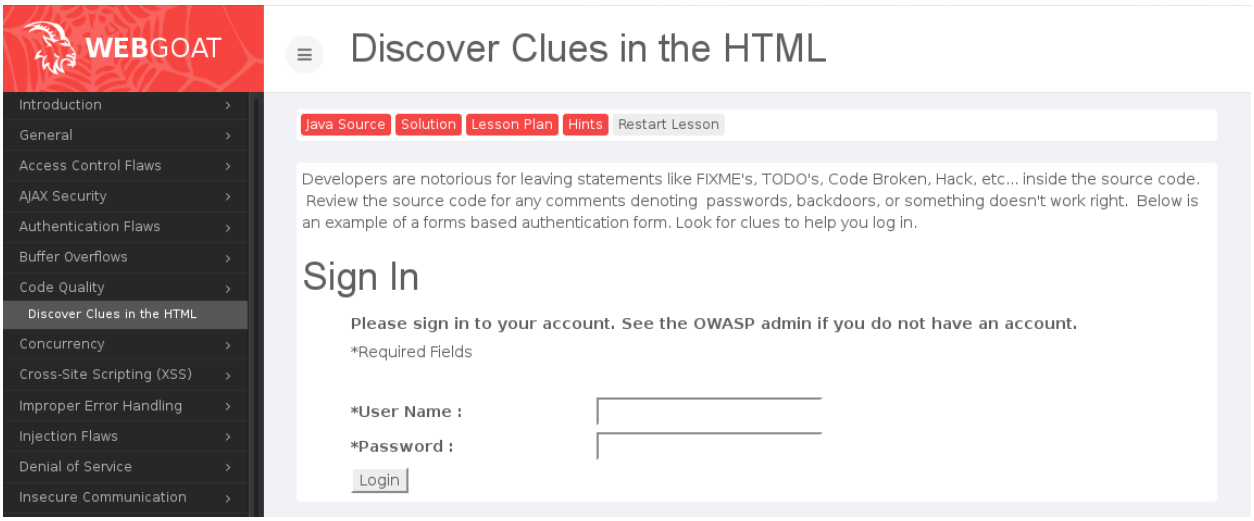
http://localhost:8090/WebGoat/

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 a more-secure alternative, you may opt to install the Java runtime environment on a personal machine and run this application there.)

Getting Started

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 eight exercises to complete this lab:

Part 1: Phishing with XSS

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=___
```

If you are successful, a green checkmark will appear.

Hints:

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_2022s_user`
password text field:	`ece568_2022s_pw`
submit button:	`ece568_2022s_submit`

Part 2: Reflected XSS Attacks

Navigate to the "Cross-Site Scripting (XSS)" lessons and select "Reflected XSS Attacks" (not the "LAB..." lessons).

In this section, you need to determine which form element is vulnerable to script injection. Using this information, you will then craft a URL that injects JavaScript into the page such that when a victim visits the URL, their credit card number is sent to the attacker at the following URL.

```
http://localhost:8090/WebGoat/catcher?PROPERTY=yes&stolenCardNum=___
```

If you are successful, a green checkmark will appear. You may need to reopen the original lesson page to see the green checkmark and congratulations message.

You can assume that the user is logged in before visiting your URL, and that they will enter your URL directly into the address bar. In order to obtain the user's actual credit card number (and not the default number), you should steal the user's information after they have clicked the "Buy" button. You can verify the credit card number sent to the attacker in the WebGoat server logs.

For stealthiness, you should also make sure that the page with the injected script looks as close as possible to the original page for "Reflected XSS Attacks", otherwise you will lose marks.

Hints:

You may find the "Hints" button helpful. Also, you can set form parameters in a URL, as shown here:

```
 http://localhost:8090/WebGoat/start.mvc#attack/SCREEN_ID/900?
input1=123&input2=456
```

If you include special characters in the URL, you may need to encode it. There are a variety of URL encoding tools available online. Please only encode the parts of the URL that are necessary.

What to submit:

You should provide your attack URL in `part2.txt`. Your explanation should indicate the field that's exploitable and a readable, unencoded version of your URL. It should also include an explanation of how/why the attacker was able to obtain the credit card number.

Part 3: Cross Site Request Forgery (CSRF)

Navigate to the "Cross-Site Scripting (XSS)" lessons and select "Cross Site Request Forgery (CSRF)". In this section, you need to successfully complete the "transfer" action to steal money from the victim/user's account.

If you are successful, a green checkmark will appear. You may need to refresh the lesson page to see the green checkmark and congratulations message.

What to submit:

Your explanation should indicate the field that's exploitable and you should provide the exploit you craft in `part3.txt`.

Part 4: CSRF Prompt By-Pass

Navigate to the "Cross-Site Scripting (XSS)" lessons and select "CSRF Prompt By-Pass". In this section, you need to successfully complete the "transfer" action, but it is slightly more complicated than in Part 3. The "transfer" action has two steps, requiring you to start the transfer and then confirm it by fetching a second URL.

If you are successful, a green checkmark will appear. You may need to reopen the original lesson page to see the green checkmark and congratulations message.

You should provide the exploit you craft in `part4.txt`, and your explanation should indicate the field that's exploitable and how/why you were able to defeat the prompt by-pass.

Part 5: CSRF Token By-Pass

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, but it is significantly more complicated than in Parts 3 or 4. 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.

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 will appear. You may need to reopen the original lesson page to see the green checkmark and congratulations message.

You should provide the exploit you craft in `part5.txt`, and your explanation should indicate the field that's exploitable and how/why you were able to defeat the token by-pass.

Part 6: String SQL Injection

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:

http://hsqldb.org/doc/2.0/guide/sqlroutines-chapt.html

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

http://www.sqlinjection.net/

This exploit should be fairly simple. You should include the full contents of the field that you entered in `part6.txt`.

Part 7: Database Backdoors

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 $**999**. 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_24s@utoronto.ca". The WebGoat lesson plan will present you with the proper format for the trigger once you complete the first part.

Your answer should include the full contents of what you entered in both parts. The first line in `part7.txt` should be your answer for the first part, and the second line for the second part.

Part 8: Blind Numeric SQL Injection

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 `1234123412341234`.

You should provide the value of the PIN in `part8.txt`. Your explanation should include the SQL injection you used and a brief description of how you found the PIN.

The remainder of this 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 always download the latest version here:

https://github.com/WebGoat/WebGoat
