ECE568 – Computer Security

Lab #4: 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 Friday, April 8th. Please submit a README file that contains your name(s) and student number(s), along with a text file that contains your answers for each of the eight exercises:

```
submitece568s 4 README part1.txt part2.txt part3.txt part4.txt part5.txt \
      part6.txt part7.txt part8.txt
```

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 archive into it:

http://www.ecf.utoronto.ca/~gibson/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. 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@p50.ecf.utoronto.ca -L 8090:localhost:8090
```

(If you are using Windows, then a program like putty can perform the same port-forwarding.) You can then connect to the service from your local web browser by connecting to:

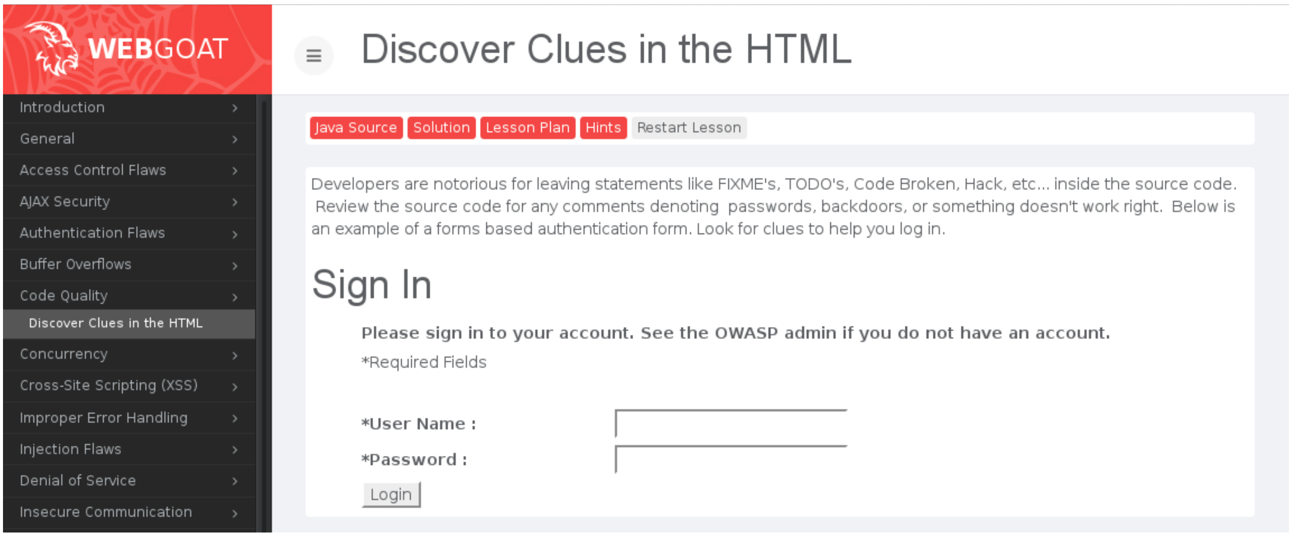
http://localhost:8090/WebGoat/

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

To get oriented to WebGoat, start by going to the *Discover Clues in the HTML* lesson:



Discover Clues in the HTML

Developers are notorious for leaving statements like FIXME's, TODO's, Code Broken, Hack, etc... inside the source code. Review the source code for any comments denoting passwords, backdoors, or something doesn't work right. Below is an example of a forms based authentication form. Look for clues to help you log in.

Sign In

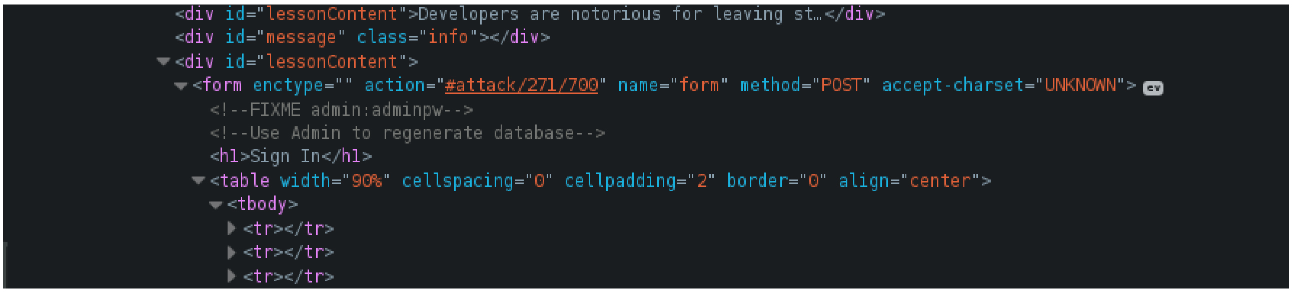
Please sign in to your account. See the OWASP admin if you do not have an account.

*Required Fields

*User Name :

*Password :

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:



```
          <div id="lessonContent">Developers are notorious for leaving st…</div>
          <div id="message" class="info"></div>
        ▼ <div id="lessonContent">
          ▼ <form enctype="" action="#attack/271/700" name="form" method="POST" accept-charset="UNKNOWN">
              <!--FIXME admin:adminpw-->
              <!--Use Admin to regenerate database-->
              <h1>Sign In</h1>
            ▼ <table width="90%" cellspacing="0" cellpadding="2" border="0" align="center">
              ▼ <tbody>
                ▶ <tr></tr>
                ▶ <tr></tr>
                ▶ <tr></tr>
```

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". In this section, you will need to create a solution that includes HTML and JavaScript. Your goal is to generate a fake "login" form 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&user=___&password=___
```

To create your solution, you will need to create an HTML form whose button has an "onclick" action that calls your JavaScript function. 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"). If you are successful, a green checkmark will appear.

Your answer should detail the full code that you create.

Part 2: Reflected XSS Attacks

Navigate to the "Cross-Site Scripting (XSS)" lessons and select "Reflected XSS Attacks". In this section, you need to determine which form element is vulnerable to script injection, and create an injection that causes WebGoat to display "*Congratulations. You have successfully completed this lesson.*". The "Hints" button at the top may be useful.

Your answer should indicate both the field that's exploitable and the exploit you craft.

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 "transferFunds" action.

Your answer should include both the field that's exploitable and the exploit you craft.

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 "transferFunds" action, but it is slightly more complicated than in Part 3. The "transferFunds" action has two steps, requiring you to start the transfer and then confirm it by fetching a second URL.

Your answer should include both the field that's exploitable and the exploit you craft.

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 "transferFunds" action, but it is significantly more complicated than in Parts 3 or 4. The "transferFunds" action has two steps, but 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 those frames.

Your answer should include both the field that's exploitable and the exploit you craft.

Part 6: SQL String Injection

Navigate to the "Injection Flaws" lessons and select "String SQL Injection" (the frist one, 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/guide/ch09.html

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

http://www.sqlinjection.net/

This exploit should be fairly simple. Your answer should include the full contents of the field that you entered.

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 $100,000. 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. 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.

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 1111222233334444.

Your answer should include the SQL injection you used, a *brief* description of how you found the PIN, and the value of 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
