ENPM809W – Introduction Secure Software Engineering

Date Due: Tuesday, November 2, 2021 by 11:59 PM

Attack Lab 7: Poor Session Management

Introduction

This lab will demonstrate some of the common session management related weaknesses in web applications. This assumes you have completed Lab 0 which involves performing the Lab environment setup. For this lab, you will need to run the WebGoat.NET application which was performed as part of Lab 0: Phase 2.

WARNING: Do not attempt to use these techniques elsewhere. Only run them on the WebGoat.NET application either inside the virtual machine or the application running on your own machine. There are legal consequences to you if you use these elsewhere!

For each of the following on the WebGoat.NET application, please submit a single writeup containing all the phases and the answers to each of the questions provided in each phase. For each question, please look at the Lesson Instructions button on the given page unless specified otherwise. The answers should be provided in the following comma separated format:

Question Number.

a) Provide the URL of the WebGoat.NET application page where you are exercising the question. Do not include any query parameters or any other special characters in the answer.
b) Describe and provide the Input given to the application. Provide the input as is with no decorations. If the input is a URL, provide the URL encoded string. If the input is provided as text to multiple fields, list the fields and the input provided for each. If the input is non-printable characters, please provide the bytes provided to the input in Hexadecimal format. For example: 9ABCD01234.
c) Describe the output result.
d) Provide a screenshot of the output.
e) Provide the CWE-ID, Filename, Line Number of the weakness that is at the heart of the vulnerability.
f) Describe the vulnerability and what role the weakness plays in allowing the input (attack vector) to compromise the application.

Note: The instructor may ask you apply some diffs to your codebase to update the Lab code for this Lab.

Phase 1: Capturing Session Identifiers and hijacking sessions

Just like Lupin, you know when the long arm of the law has started encircling you. Your career as an international thief is becoming riskier and you have decided to go into pen testing security for auction sites like the WebGoat .NET Coins portal, since it can also be very lucrative. By now hopefully you have become an expert on how to use BurpSuite Intercept and perhaps the Repeater feature from Attack Lab 5. For this phase of the lab, we will be using BurpSuite Intruder to intercept a live session on the WebGoat Coins Portal. For a tutorial, please see https://www.youtube.com/watch?v=nC9D1ES-nmo.

For all phases of this lab, you will need to download the '**SessionLab**' branch of the WebGoat.NET project from the instructor's git repository on code.umd.edu. This extends the 'AuthLab' branch and uses the same Argon2 passwords you are familiar with. You should be able to run Rebuild database as long as the SCRIPT_DIR variable in DBConstants.cs is updated to the correct scripts directory on your machine. Now do the following:

1. Run the WebGoat .NET coins portal (with BurpSuite running) and login to the portal using:
    a. Username: jerry@goatgoldstore.net
    b. Password: password
2. Now under WebGoat Coins Portal, go to the Customer Orders. You should see three orders displayed. Now click Order No. 10124. You should see Order No. 10124 displayed.
3. Now click Logout at the top.
4. (Optional) If you now run the Stored XSS or Reflected XSS attack on the Input Attacks section to display all the cookie key/value pairs, you will see a new ASP.NET_SessionId displayed. You don't have to do this step.
5. Use the BurpSuite Intruder feature to now view Order No. 10278 and Order No. 10346 for 'jerry@goatgoldstore.net' without logging in on the WebGoat .NET page on Firefox.
6. What is the weakness at the heart of this attack that allowed this to happen? Answer the questions at the top of this lab handout with regards to the attack you just performed. Also note the code in Logout.aspx.cs for the logout button.

Phase 2: Predictable Order Numbers

Your career in penetration testing is short lived as the allure of how easy it is to attack auction sites brings you back to your original career path. Now you happen to know an auction taking place with the vulnerable WebGoat .NET application and you also happen to know a vulnerability that exists in advance. You plan to figure out what coins have already been bid on by looking at the Customer Order history for specific auction bidders since you can figure out what rare coin collections those bidders have. The vulnerability you figured out involves being able to get order numbers for specific auction bidders/users from an insider at the auction. **Note:** For this phase, just displaying the information in a pop up is enough as you can later figure out how to collect all the information in one place. For this you do the following:

1. The user 'julie@golddepotinc.com' has three order numbers you wish to view that you predict to be Order No. 10172, Order No. 10263, and Order No. 10413.
2. Based on what you used in Phase 1, now attempt to view these specific orders without logging in on the WebGoat .NET application at all.
3. What are the weaknesses that allowed this to happen? Please answer the questions at the top of this lab handout in regards to this attack.

Phase 3: Cross-Site Request Forgery (CSRF)

The long arm of the law unfortunately catches up with you and you figure out that the vulnerability in Phase 2 was actually a trap to find you. The insider at the auction gave you up. In order to embarrass the auction site, you decide to perform a Cross-Site Request Forgery just by posting a single comment containing a maliciously crafted link and script, which when clicked by logged in users of the WebGoat

.NET Coins portal posts their order numbers as a new comment to the Stored XSS page. You figure this will be sufficient to embarrass the auctioneers who gave you up to the authorities. Do the following in order to achieve this:

1. First you need JavaScript code that can post a new comment on behalf of the user. For example, in order to post a message from fun@fun.com with a comment 'Fun message' on the StoredXSS exercise page, you can use something like the following:

```
<a onclick="mal()" href="javascript:void(0);">I have a few extra coins. Click here to get free
coins!</a>
<script language=''javascript'' type=''text/javascript''>
    function mal() {
        document.getElementById(''ctl00_BodyContentPlaceholder_txtEmail'').value = "fun@fun.com";
        document.getElementById(''ctl00_BodyContentPlaceholder_txtComment'').value = "Fun
message";
        document.getElementById(''ctl00_BodyContentPlaceholder_btnSave'').click();
    }
</script>
```

2. Now you have been given jQuery 3.3.1 version, which includes the ability to perform AJAX queries as part of WebGoat .NET in client-side scripts that can be included using the following line:

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
```

3. Now you would use AJAX to make a GET request to http://localhost:52251/WebGoatCoins/Orders.aspx and copy the content/value of the span element (with ID of ctl00_BodyContentPlaceholder_lblOutput) on that page as the message for txtComment instead of "Fun message" as shown in #1 above. You can change the email to whatever you like.
4. Now answer the questions at the top of this lab handout with regards to the attack you just performed.

Submission Criteria

Please submit a writeup with:

1. Your name, UMD email ID and Lab Number.
2. The answers provided in the format given at the top of this lab handout as specified in each of the phases.

Please only submit a **Word document** or a **PDF**. There is no code submission for this lab.