# ENPM809W - Introduction to Secure Coding

## Lab - 1 - Attack Lab

*Author: Syed Mohammad Ibrahim*

*UID: iamibi*

*Email:* [*iamibi@umd.edu*](mailto:iamibi@umd.edu)

### Phase - 1, Part - 1: SQL Error Messages

1. 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/SQLInjectionDiscovery.aspx

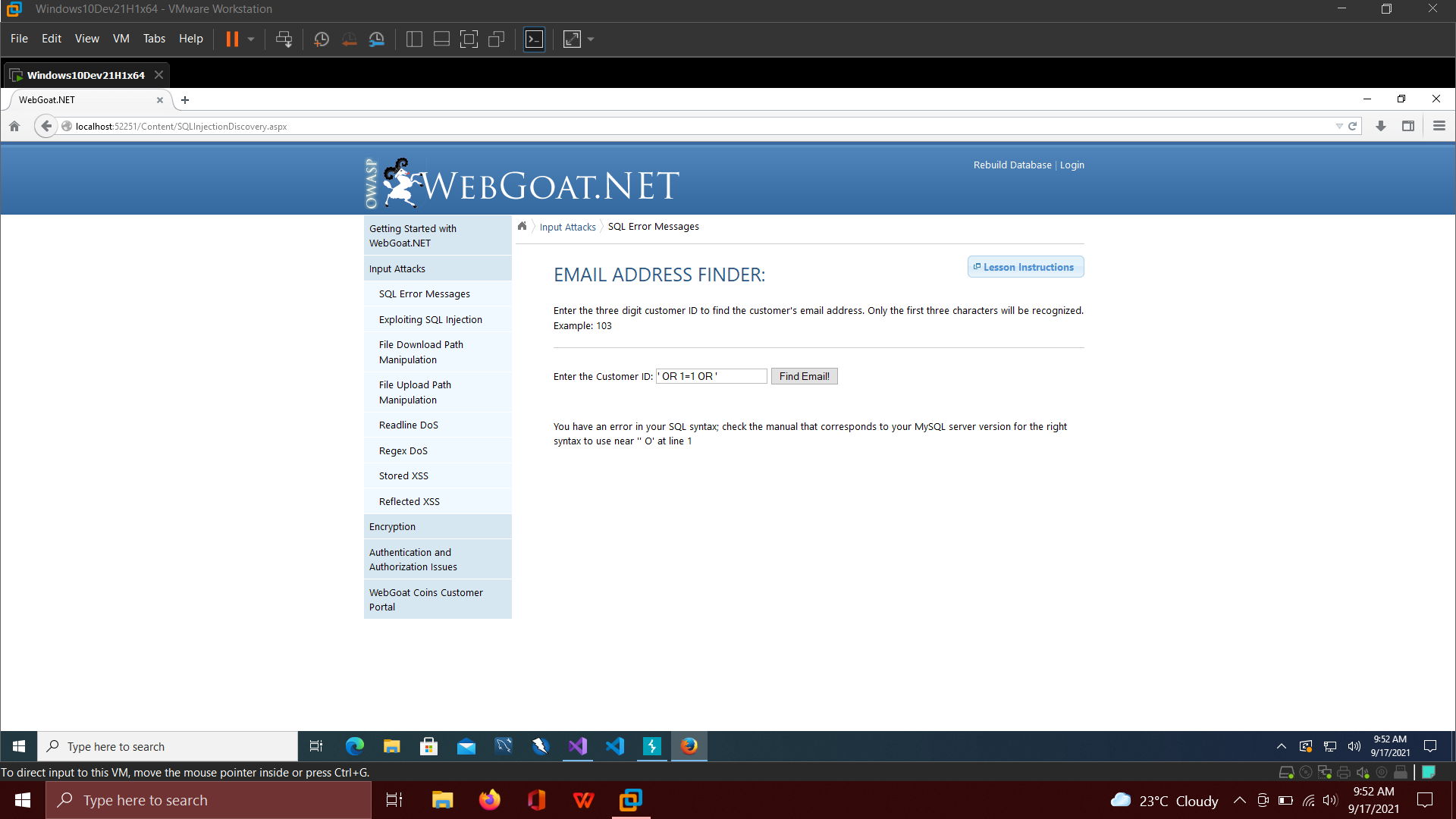
1. Describe and provide the Input given to the application. Provide the input as is with no decorations. If the input is a URL, provide the URL encoded string. If the input is provided as text to multiple fields, list the fields and the input provided for each.If the input is non-printable characters, please provide the bytes provided to the input in Hexadecimal format. For example: 9ABCD01234.

The provided input was ' OR 1=1 OR ' which is a SQL query to always return true when passed in the **where** clause of SQL.

1. Describe the output result.

The output result was "You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' O' at line 1" which is an SQL syntax error message which demonstrates that the application is not sanitizing the inputs from the text field and is directly substituting them to the SQL query to get the result.

1. Provide a screenshot of the output.



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

CWE-89: Improper Neutralization of Special Elements used in SQL Command (‘SQL Injection’)

Filename: MySqlDbProvider.cs

Line number: 539

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

The vulnerability is that the input text is not being tested and sanitized before passing the value to the SQL constructed query. This causes the SQL to substitute the value as is, and the attacker can leverage this vulnerability to gain access to all the credentials stored in the database by crafting a query that will return other user’s information.

### Phase - 1, Part - 2: SQL Injection

1. 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/SQLInjection.aspx

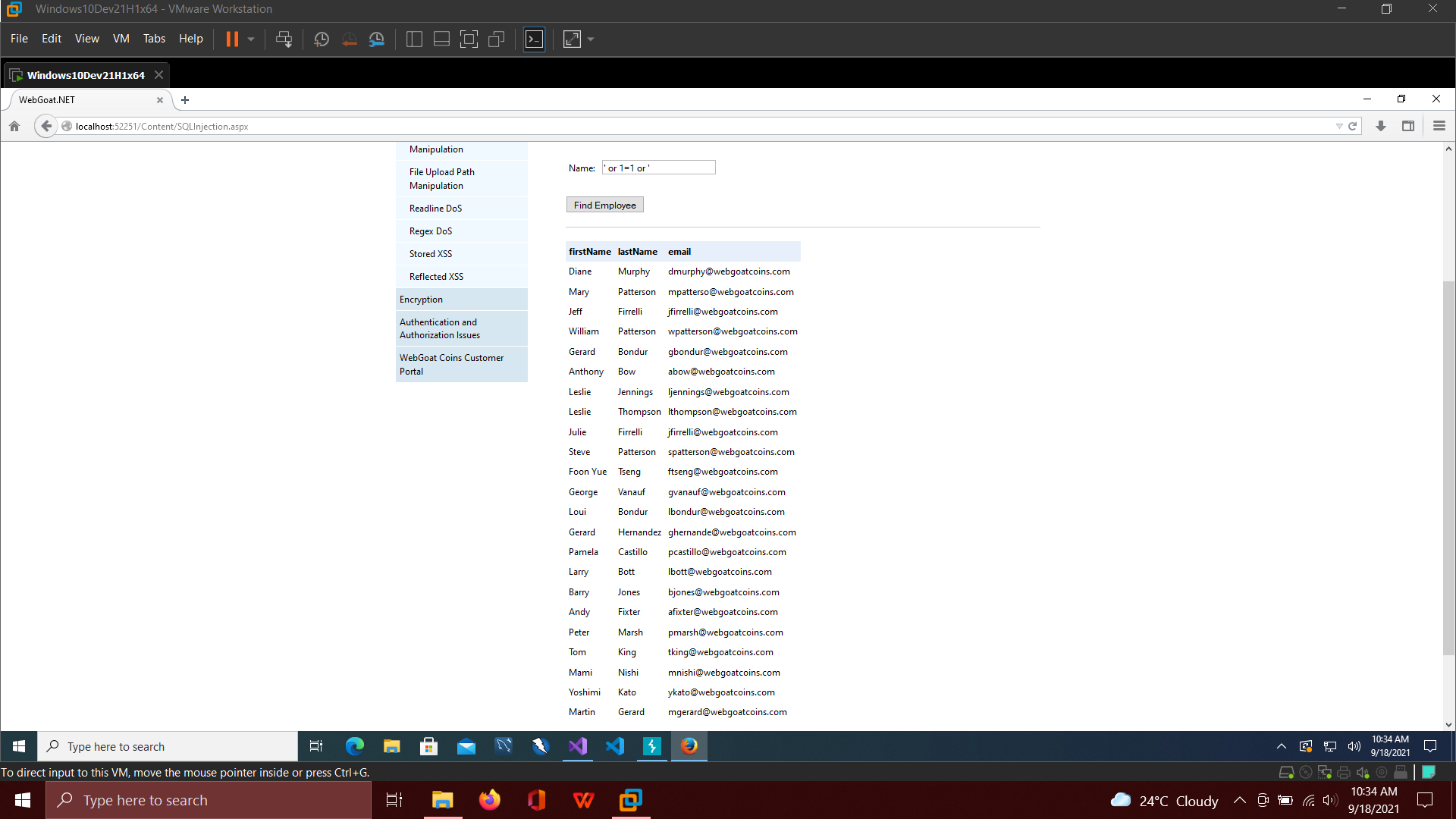
1. Describe and provide the Input given to the application. Provide the input as is with no decorations. If the input is a URL, provide the URL encoded string. If the input is provided as text to multiple fields, list the fields and the input provided for each. If the input is non-printable characters, please provide the bytes provided to the input in Hexadecimal format. For example: 9ABCD01234.

The provided input was ‘ or 1=1 or ’ which makes the regular SQL query to always return true.

1. Describe the output result.

The output result was the list of all the users present in the database with three columns, that is firstName, lastName and email.

1. Provide a screenshot of the output.



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

CWE-89: Improper Neutralization of Special Elements used in SQL Command (‘SQL Injection’)

Filename: MySqlDbProvider.cs

Line number: 517

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

The vulnerability was that the input was not being sanitized while it was being substituted to the SQL query. This led to a random SQL query that can be passed as an input, causing the query to return the information of all the users from the database.

### Phase - 2, Part - 1: File Download Path Manipulation

1. 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/PathManipulation.aspx

1. 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 provided input was

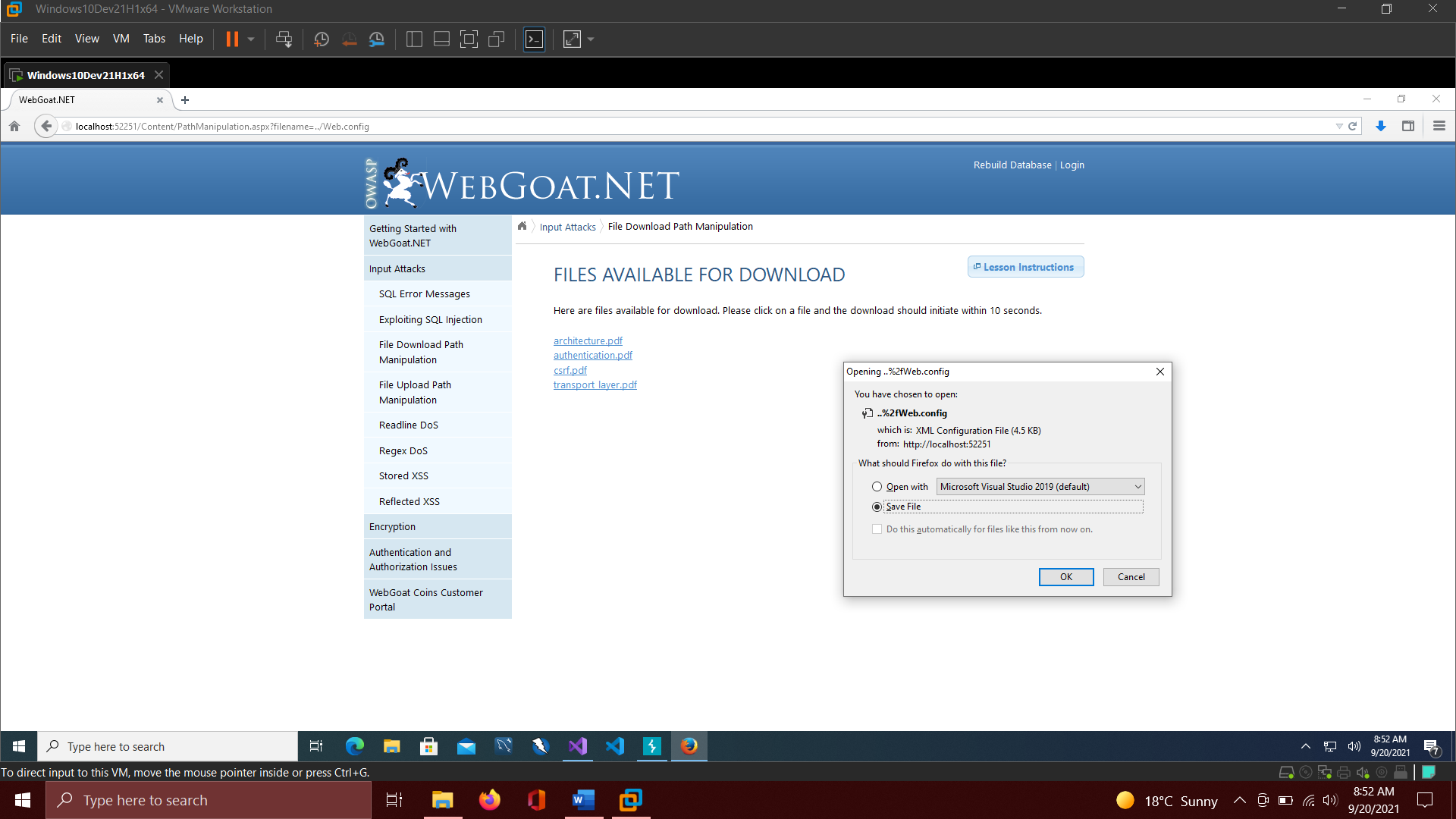
<http://localhost:52251/Content/PathManipulation.aspx?filename=../Web.config>

Where the parameter ‘filename’ was passed in as a path to the web.config file.

1. Describe the output result.

The output was the web.config file which was now accessible and can be downloaded locally.

1. Provide a screenshot of the output.



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

CWE-22: Improper Limitation of a Pathname to a Restricted Directory (‘Path Traversal’)

Filename: PathManipulation.aspx.cs

Line number: 25

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

Since the input from the URL is being substituted to the query parameter “?filename=” directly without any checks being made, an attacker can input a relative path which can get them access to confidential or critical files.

### Phase - 2, Part - 2: File Upload Path Manipulation

1. 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/UploadPathManipulation.aspx

1. 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 **test.asp** file which was uploaded to the webgoat server with the following code:

<html xmlns="www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div>

<%Response. Write( "HeIIo World") %>

</div>

</form>

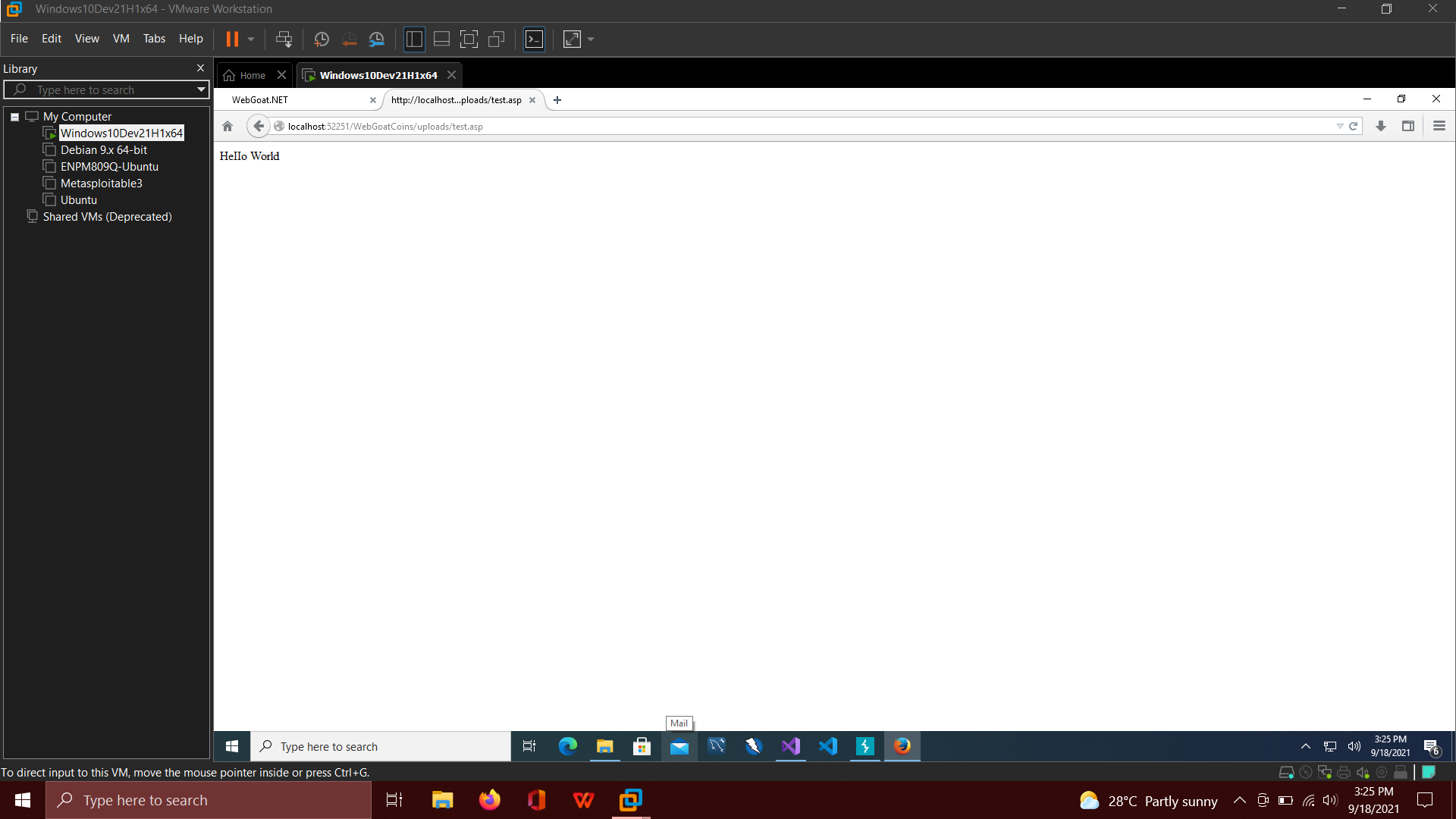
</body>

</html>

1. Describe the output result.

The result of uploading the above content was that when we tried accessing the file with the path **localhost:52251/WebGoatCoins/uploads/test.asp**, it would execute successfully and print the “Hello World” text on the browser.

1. Provide a screenshot of the output.



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

CWE-434: Unrestricted Upload of a File with Dangerous Type

Filename: UploadPathManipulation.aspx.cs

Line number: 25

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

The vulnerability resides in the way the files are being uploaded without any content integrity checks being done either in the front-end or backend systems. The files should always be verified whether or not they are of dangerous types or not.

### Phase - 2, Part - 3: Readline DOS

1. 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/ReadlineDoS.aspx

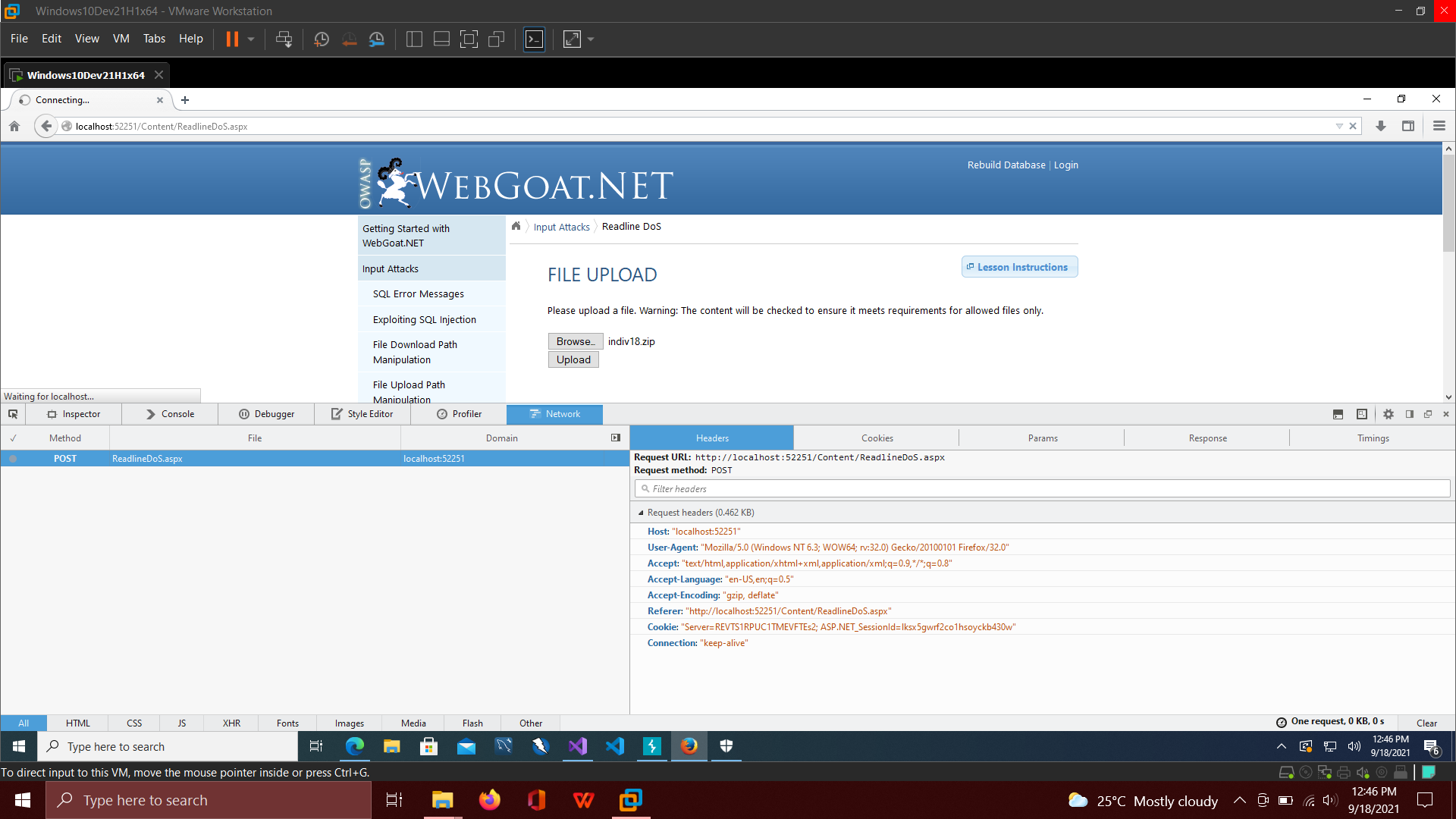
1. Describe and provide the Input given to the application. Provide the input as is with no decorations. If the input is a URL, provide the URL encoded string. If the input is provided as text to multiple fields, list the fields and the input provided for each. If the input is non-printable characters, please provide the bytes provided to the input in Hexadecimal format. For example: 9ABCD01234.

Input was a large text file with the first 10 bytes being C00401224.

1. Describe the output result.

The result was that the webpage crashed while trying to read the large file.

1. Provide a screenshot of the output.



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

CWE-434: Unrestricted Upload of File with Dangerous Type

Filename: ReadlineDoS.aspx.cs

Line number: 23

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

The vulnerability was that the file reading was being done using “Readline()” which tries to halt the program until it has completed reading the line from the buffer memory and reached a return carrier (\r) or new line (\n). Because of this, an attacker can push a file with a very big single line causing the full memory usage.

### Phase - 3: Regex DOS

1. 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/RegexDoS.aspx

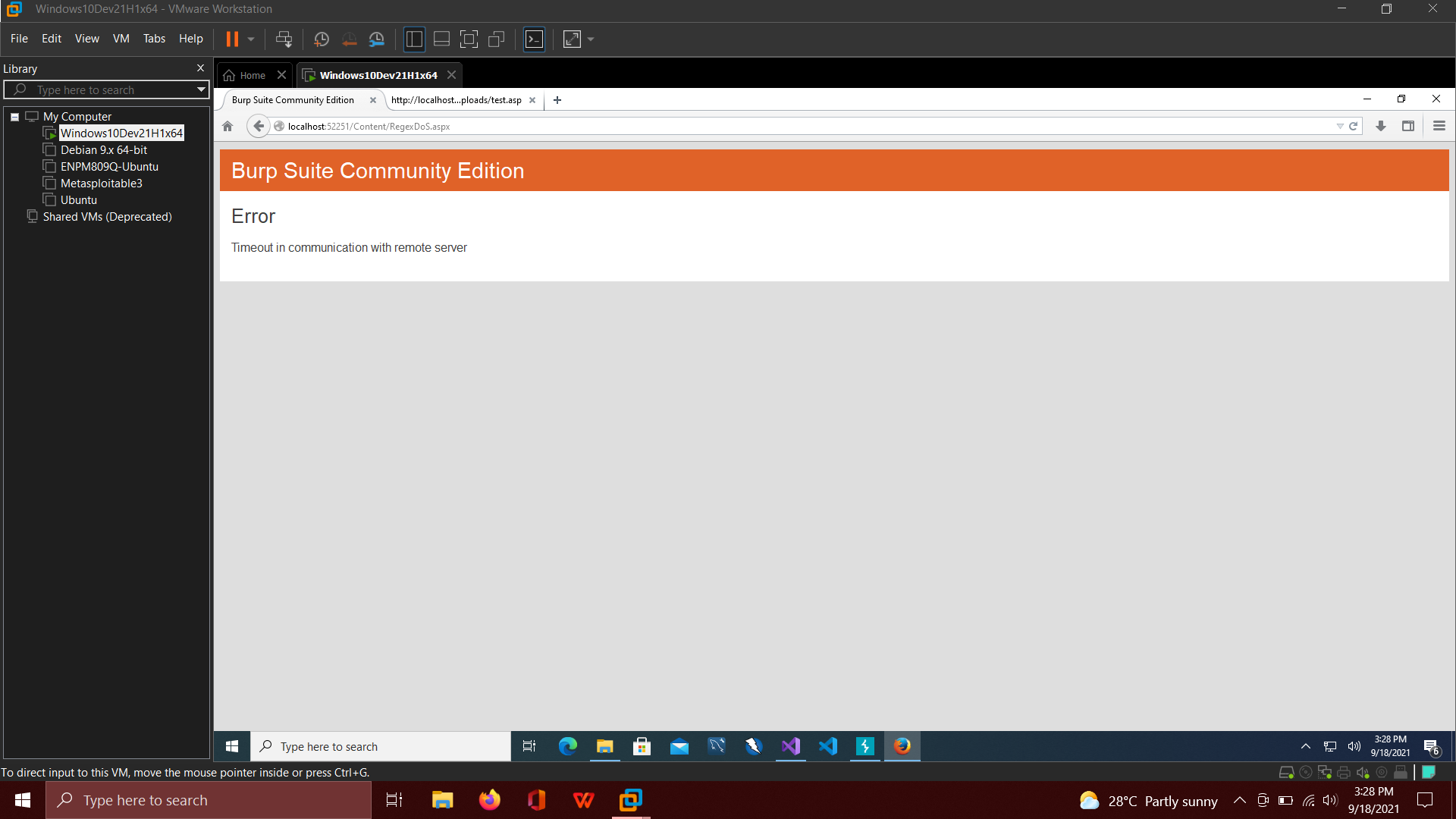
1. Describe and provide the Input given to the application. Provide the input as is with no decorations. If the input is a URL, provide the URL encoded string. If the input is provided as text to multiple fields, list the fields and the input provided for each. If the input is non-printable characters, please provide the bytes provided to the input in Hexadecimal format. For example: 9ABCD01234.

The input was “aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!” without the quotes.

1. Describe the output result.

The output was the page timed out because of trying to parse a large input regex.

1. Provide a screenshot of the output.



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

CWE-1333: Inefficient Regular Expression Complexity

Filename: RegexDoS.aspx.cs

Line number: 25

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

The vulnerability is that the input is not being sanitized to remove any kind of regular expression which in turn can assist the attacker to input a complex regex that can take a lot of time to compute. The input is directly substituted to the regex matcher.

### Phase - 4, Part - 1: Stored XSS

1. 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

1. 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 following

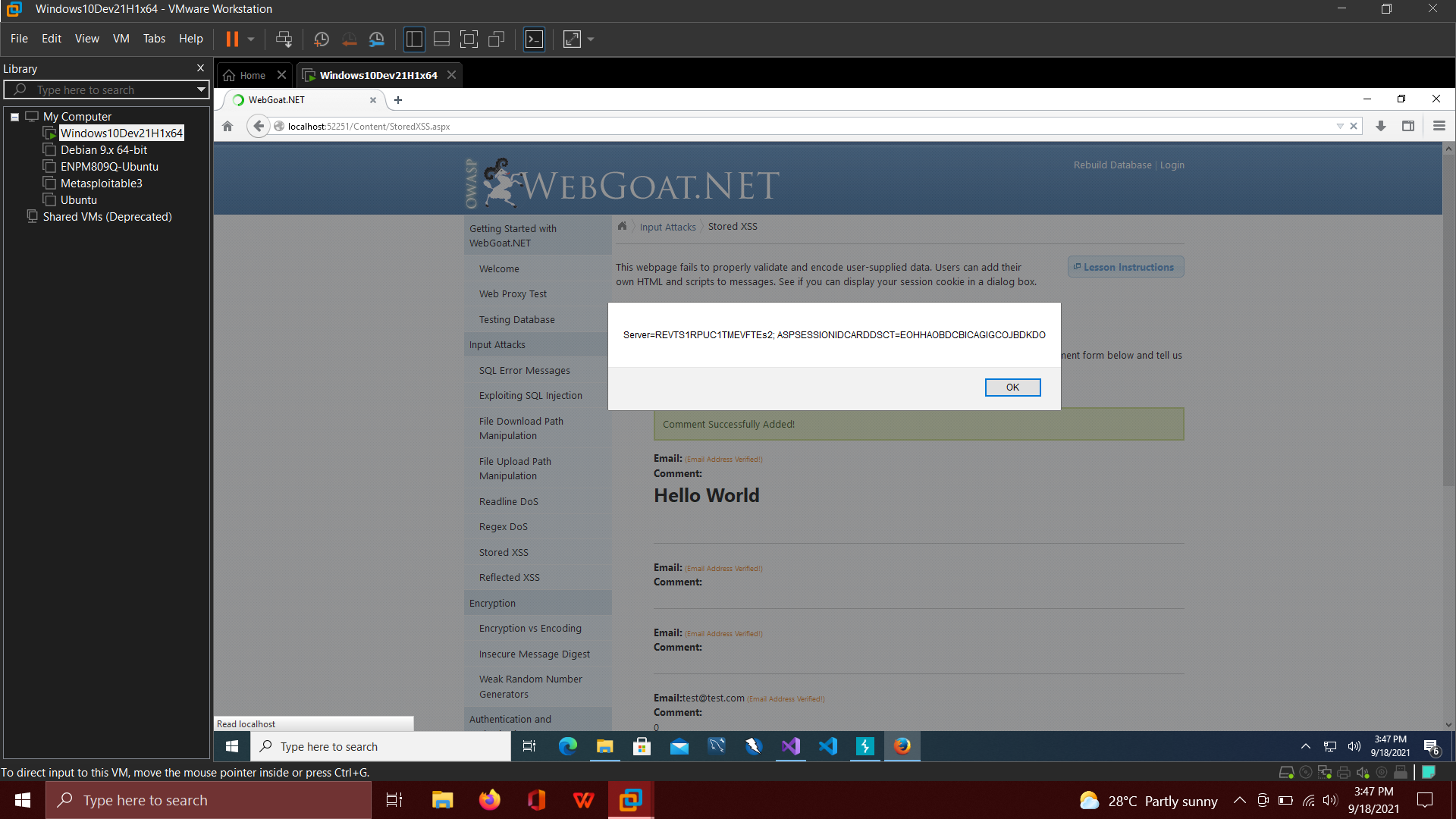
Email: [test@test.com](mailto:test@test.com)

Comment: <script>alert(document.cookie);</script>

1. Describe the output result.

An alert was displayed with the session cookie values printed in it.

1. Provide a screenshot of the output.



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

CWE-79: Improper Neutralization of Input During Web Page Generation (‘Cross-Site Scripting’)

Filename: StoredXSS.aspx.cs

Line number: 61

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

The vulnerability resides in the way the input is being treated. There is no sanitization happening on the comments text box which leads to an attacker passing in a javascript code script and getting the session information. This happens because the input is first stored and then loaded by the application as part of the HTML code to render the text.

### Phase - 4, Part - 2: Reflected XSS

1. 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/ReflectedXSS.aspx

1. 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.

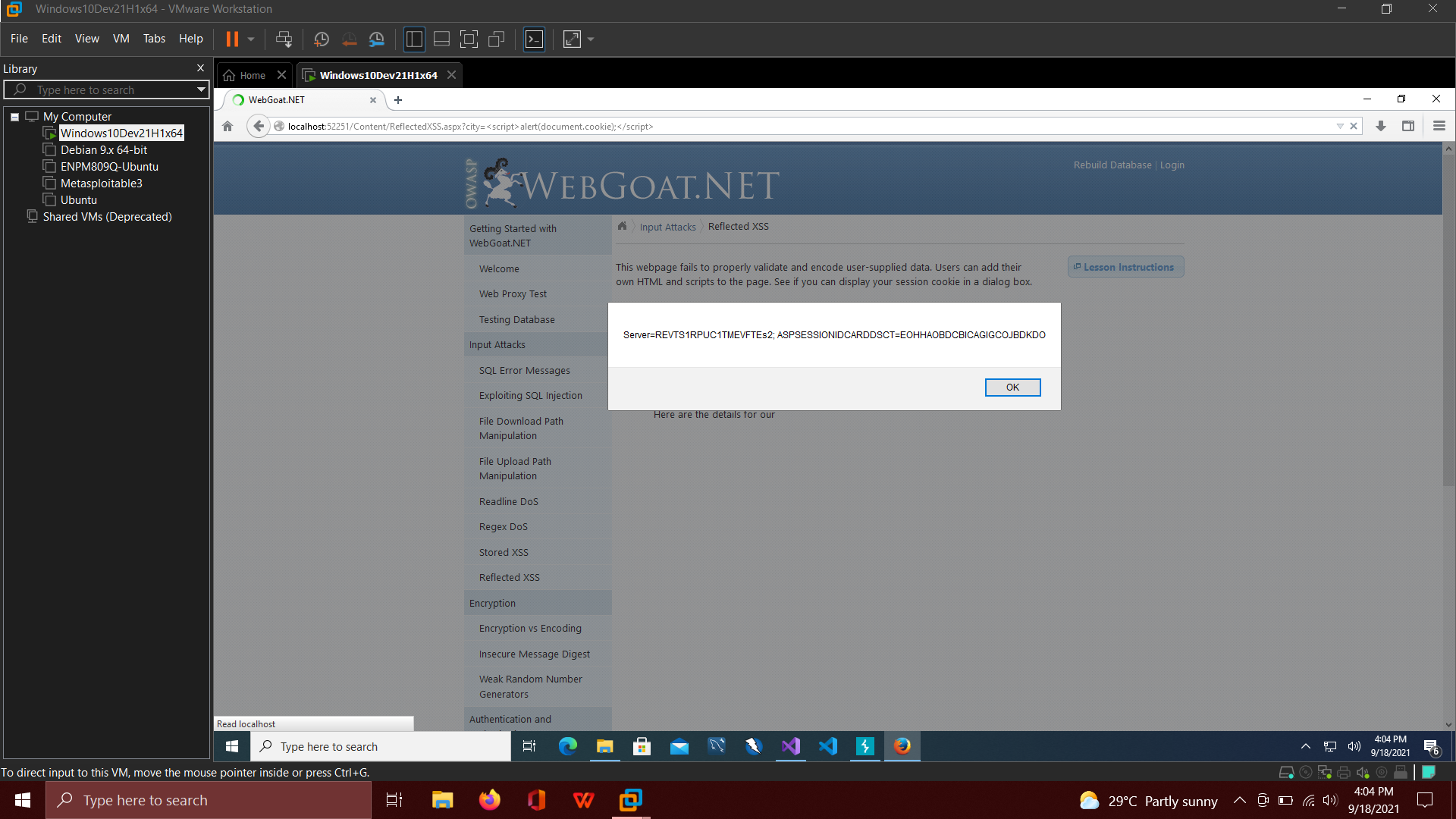
The input was a modified query parameter to the URL where we pass a javascript code snippet instead of an expected city value.

http://localhost:52251/Content/ReflectedXSS.aspx?city=<script>alert(document.cookie);</script>

1. Describe the output result.

The result is an alert pop-up which displays the session cookie values.

1. Provide a screenshot of the output.



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

CWE-79: Improper Neutralization of Input During Web Page Generation (‘Cross-Site Scripting’)

Filename: ReflectedXSS.aspx.cs

Line number: 34

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

The vulnerability is that the input is being loaded in the context that it is a plain text but it is actually never verified and no kind of sanity checks are done. This leads an attacker to load a custom javascript as part of the URL query parameter which gets executed successfully on the browser window.

### Phase - 5: WinSharpFuzz Fuzzing

1. 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.

NA

1. 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.

{

  "menu":{

"menu": {

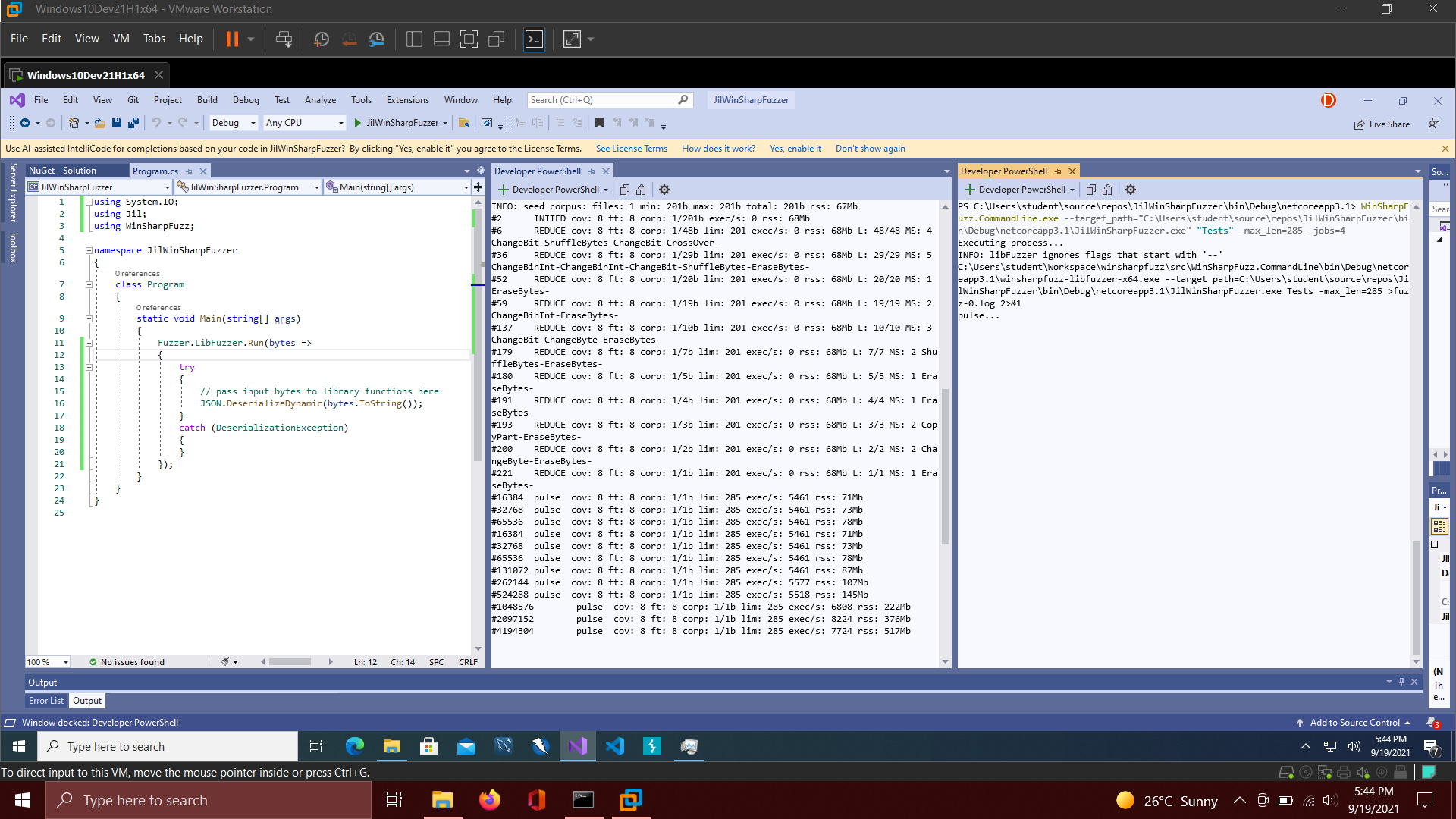
"id": 1,

{

1. Describe the output result.

The fuzzer was trying to find weakness in the JSON library and printed the options it has already tried, in a log file.

1. Provide a screenshot of the output.



From left to right: Code, Log file, Powershell running the fuzzer

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

NA

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

NA