ENPM809W - Introduction to Secure Coding

Lab - 2 – Defense Lab

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

Phase 1, Part 1: SQL Error Message

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/SQLInjectionDiscovery.aspx

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

There are multiple safeguards that can be put in place. Considering the input required is only a 3-digit number. I did the following:

1. Verify that the provided input is of size equal to 3 (as per the input suggestion) as an input of size 2 was throwing a server error. If the provided input is less than 3 then we are raising our own custom error. This check is done in the file **SQLInjectionDiscovery.aspx.cs** at line number 29.
2. In **MySqlDbProvider.cs** in the method **GetEmailByCustomerNumber**, we try converting the input string value "num" to an integer using
```
int.TryParse(num, out number);
```
where "**number**" is an integer type and "**isNumber**" holds a boolean value of true or false depending on the input was converted to integer or not.
If the conversion succeeds, we go ahead and run it as part of the SQL query. If it doesn't, we throw a custom error stating the same.

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

My fix for this part will be safeguarding the database-based attacks quite easily as the intended input is a 3-digit integer value and we try converting the given input to an integer. If it succeeds, that means the user entered a valid number. If it doesn't that means it had some or all characters that were not integers and thus failing the conversion. And when the conversion fails, we are throwing our own custom error instead of a SQL based message to avoid the weakness.

d) List all the paths with file names you are changing to implement the fix for the weakness.

- C:\Users\student\Workspace\WebGoat\WebGoat\App_Code\DB\MySqlDbProvider.cs – Method: GetEmailByCustomerNumber
- C:\Users\student\Workspace\webgoat\WebGoat\Content\SQLInjectionDiscovery.aspx.cs – Method: btnFind_Click

e) Commit id: 611914b930afbcb769fa348c538b2cf77a41bff7

Phase 1, Part 2: SQL Injection

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/SQLInjection.aspx

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

Using Regex to parse the input and verifying that only "a-z" and "A-Z" characters are present, and no other characters, spaces or numbers are allowed. Also, the length of the string to be parsed by the regex parser is set to be between 1-10, that is it will only check the string of length 1 to 10.

```
// Remove any whitespace either leading or trailing.
string cleanName = name.ToString().Trim();

// If the Regex for a-z A-Z and length of 10 doesn't match then return null.
if (!Regex.IsMatch(cleanName, @"^[a-zA-Z]{1,10}$")) { return null; }
```

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

With the presence of Trim() method, we are essentially clearing up any leading or trailing spaces that might be present as part of the input. Once the input is cleared with that, I am using regex to verify that the string only contains lower- and upper-case alphabets and is of length between 1 to 10. This essentially gets rid of any special characters, spaces or numbers from the input. And if they are present, we return null to upstream code so that it can display empty string (or anything that we would like it to display).

d) List all the paths with file names you are changing to implement the fix for the weakness.

- C:\Users\student\Workspace\webgoat\WebGoat\App_Code\DB\MySqlDbProvider.cs – Method: GetEmailByName
- C:\Users\student\Workspace\webgoat\WebGoat\Content\SQLInjection.aspx.cs – Method: btnFind_Click

e) Commit id: 611914b930afbcb769fa348c538b2cf77a41bff7

Phase 2, Part 1: File Download Path Manipulation

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/PathManipulation.aspx

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

By implementing a whitelist of files, we can only allow the intended files to be made downloadable.

```
const string fileArchitecture = "architecture.pdf";
const string fileAuthentication = "authentication.pdf";
const string fileCSRF = "csrf.pdf";
const string fileTransportLayer = "transport_layer.pdf";

// Allowed Files whitelist.
public static readonly string[] allowedFiles = { fileArchitecture,
fileAuthentication, fileCSRF, fileTransportLayer };
```

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

As I implemented the whitelist, I am only allowing the input to be of specific type that is acceptable. Any other kind of filename input is skipped through and a label with error message will come up.

d) List all the paths with file names you are changing to implement the fix for the weakness.

  - C:\Users\student\Workspace\webgoat\WebGoat\Content\PathManipulation.aspx.cs – Method: Page_Load

e) Commit id: b5d88cfa858031f462b98e62ea515c1bbc8c5f34

Phase 2, Part 2: File Upload Path Manipulation

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/UploadPathManipulation.aspx

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

   Creating a temporary folder at location WebGoatCoins/uploads folder and after storing the file, I will remove the permission of “student” user to be able to read and execute permission.

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

   The intended fix will not allow the file to be executed by the server and when the user if at all is able to access it, will not be able to run it on the server as it will throw a 403 unauthorized exception.

d) List all the paths with file names you are changing to implement the fix for the weakness.

  - C:\Users\student\Workspace\webgoat\WebGoat\Content\UploadPathManipulation.aspx.cs – Method: btnUpload_Click

e) Commit id: b5d88cfa858031f462b98e62ea515c1bbc8c5f34

Phase 2, Part 3: Readline DOS

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/ReadlineDoS.aspx

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

   Increasing the executionTimeout and maxRequestLength to a bigger value in Web.config.

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

   By increasing the execution timeout, we are making sure that the server doesn’t halts after a specific interval. With the increase in maxRequestLength to a value that makes large files acceptable we can upload a large file successfully.

d) List all the paths with file names you are changing to implement the fix for the weakness.

   - WebGoat/Web.config

e) Commit id: b5d88cfa858031f462b98e62ea515c1bbc8c5f34

Phase 3: Regex DOS

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/RegexDoS.aspx

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

I would use timeout of 5 seconds while trying to parse the input string using regex. .NET library 4.7.2 supports an implicit timeout option which will retire the parser after the time specified in the timeout.

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

   The timeout regex will terminate any expression that is taking too long to execute after five seconds. This saves the expression from any backtracking that might be happening.

d) List all the paths with file names you are changing to implement the fix for the weakness.

   - C:\Users\student\Workspace\webgoat\WebGoat\Content\RegexDoS.aspx.cs – Method: btnCheckUsername_Click

e) Commit id: 239ba9305ab7f1f3bed4f062bd8b37eae6b0dc2d

Phase 4, Part 1: Stored XSS

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) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

   By using the AntiXssEncoder library in .NET Framework, I would escape the input text by using AntiXssEncoder.HtmlEncode which will encode any special tags that respond to HTML and then store it in the database.

```
AntiXssEncoder.HtmlEncode(txtEmail.Text, false),
AntiXssEncoder.HtmlEncode(txtComment.Text, false)
```

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

The intended fix will escape any tag that responds to HTML and encode the input string. After the input is encoded, it will be stored in the database so that when we retrieve it again from the database, it doesn't trigger the HTML tag or content.

d) List all the paths with file names you are changing to implement the fix for the weakness.

- C:\users\student\workspace\webgoat\WebGoat\Content\StoredXSS.aspx.cs – Method: btnSave_Click

e) Commit id: 03124b8c0faa876956e116dd6f10d782ade76140

Phase 4, Part 2: Reflected XSS

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/ReflectedXSS.aspx

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

I plan to take two approaches to fix the Reflected XSS issue:

1. Create a whitelist of locations that are passed as part of query parameter in the URL.
```
// Whitelist for the cities.
string[] cities = { "San Francisco", "Boston", "NYC", "Paris", "Tokyo",
"Sydney", "London" };
```

2. Using AntiXssEncoder library to display the output based on whether the query parameter was valid or not. This will always make sure that HTML relevant tags are escaped, and the parameters are encoded properly.
```
lblOutput.Text = $"Here are the details for our
{AntiXssEncoder.HtmlEncode(city, false)} Office";
```

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

The whitelist approach prevents any other kind of entry as part of the query parameter and the AntiXssEncoder library properly encodes the given parameter so that it is escaped properly if there are any HTML based tags present.

d) List all the paths with file names you are changing to implement the fix for the weakness.

- c:\users\student\workspace\webgoat\WebGoat\Content\ReflectedXSS.aspx.cs – Method: LoadCity

e) Commit id: 03124b8c0faa876956e116dd6f10d782ade76140

Phase 5: Fuzzing

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.

   NA

b) For the given CWE-ID, Filename, Line Number of the weakness identified in the previous Attack Lab, describe how you intend to fix the weakness. This can be as detailed as possible to explain how it will address the weakness.

   Sanitize and validate the input and rescue the errors appropriately that may come up while serializing or deserializing. Using appropriate data structures and types so that no underflow or overflow happens.

c) Describe why your intended fix will address the weakness (either directly or indirectly) and whether to your knowledge it will prevent future attacks along the lines of the attack vector used in the previous lab. This can be as detailed as possible to explain why it will address the weakness.

   Doing proper input validation will not allow any kind of random input to cause the application crash. Always handling all kind of possible exceptions will safeguard from leaking any kind of information to the attacker. Having proper data structures and types will help prevent any kind of weakness involving underflow or overflow situation.

d) List all the paths with file names you are changing to implement the fix for the weakness.

   NA

e) Commit id

   NA