**Web Application Security Exercise**

***Warning***

**Note: The following techniques are only meant for educational purposes in order to help you understand the importance of input validations and protecting your databases from SQL Injection attacks. DO NOT USE ANY OF THESE TECHNIQUES OUTSIDE OF THIS EXAMPLE IN CLASS. The following techniques are considered hacking and are very illegal if used improperly.**

***We will be hacking into this website using at least 5 different methods.***

***You will need the following applications/plugins installed:***

***--------------------------------------------------------------------------------------------------***

**Common Credentials**

A very easy attack is simply trying common username and password combinations. Google some common username/password and see if any provide you with access. If so, paste a screenshot and the username/password you used to gain access:

**Viewing the Page Source**

1. Open Firefox and go to: <https://lksom.temple.edu/SecurityExample/default.aspx?Authenticated=0>
2. Right click anywhere on the page and select View Page Source.
3. You are now viewing the HTML code that creates the web page. Analyze the code. Do you notice anything of note in the code? I am looking for at least 2 items of interest. Please paste screenshots of both below. Just based on what you found and without utilizing any tools can you exploit this application?

**Editing the Page Source**

1. Based on what you uncovered when viewing the page source can you exploit the site in anyway by editing the page’s html? If so, describe your steps and paste a screenshot below. (Hint: When properly authenticated a button will appear called: “Proceed to Application”).

**Editing the URL Query String**

1. In addition to editing the page source you can also edit anything that is provided in the URL. Is there anything in the URL that looks interesting? Please provide a screenshot and description below:

**Performing a SQL Injection Attack**

1. **Introduction**

The application in this example uses a common login mechanism to provide access to valid users and preventing anonymous users from accessing the application. When a user arrives at the login page for this application:



1. When a user enters their username and password and clicks login the application sends the following to the database:

*Select username, password from Users where username = ‘testuser’ and password = 'testpassword'*

*The above statement queries the Users table in the application’s database with the username and password. If the query returns a record that means the username and password combination have a match and that the credentials entered are valid. The application will then provide the user access to the application. If the query does not return a record that means the username/password combination does not exist in the database and that either the user does not exist in the database or password entered was correct and the user should not be granted access to the database.*

*The Table containing valid users looks like the following:*

**

1. While this is a common authentication method it has an inherent flaw that produces the risk of a SQL Injection attack by manipulating the query sent to the database using the login form of the application. This exercise will show how unauthorized access can be granted to the database using SQL Injection and how to resolve this vulnerability using input validation.
2. A quick overview of SQL Injection:

***SQL injection is a code injection technique, used to attack data driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker) SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.***

\*<http://en.wikipedia.org/wiki/SQL_injection>

1. Below are some common SQL Injection strings that have been used:

' or '1'='1

' or '1'='1' -- '

' or '1'='1' ({ '

' or '1'='1' /\* '

\*<http://en.wikipedia.org/wiki/SQL_injection>

1. Hack the database
2. Go to the DB Security url:

<https://lksom.temple.edu/SecurityExample/default.aspx?Authenticated=0>

1. Attempt to login using username: test1 and password: test1
2. Verify that your login attempt is unsuccessful and take note of the SQL query sent to the server:



1. Now, using some of the SQL injection strings provided above attempt to successfully login
	1. Note that the text you enter into the username and password fields are entered directly into the SQL string sent to the database, so the SQL Injection strings above can be entered into the text fields to manipulate the SQL string sent to the database
2. Once you are able to successfully login please paste your SQL String below:
3. Verify that your authentication was successful with a non-valid username and password
4. Take note of the string that is sent to the server
5. Take note of the list of usernames and passwords that you were able to pull and take note of the SQL query sent to the server
6. Click the Proceed button.
7. Congratulations! You have successfully hacked into this application!
8. Secure the Database
9. Now that that we have seen how much damage can be done simply with a single quote character (‘) and the proper malicious logic how can we remove this vulnerability? SQL Injection can be mitigated in many ways with one of the methods being Input Validation. Please see the input validation definition below:

***White List Input Validation***

It is always recommended to prevent attacks as early as possible in the processing of the user’s (attacker's) request. Input validation can be used to detect unauthorized input before it is processed by the application. Developers frequently perform black list validation in order to try to detect attack characters and patterns like the ' character, the string 1=1, or the <script> tag, but this is a massively flawed approach as it is typically trivial for an attacker to avoid getting caught by such filters. Plus, such filters frequently prevent authorized input, like O'Brian, when the ' character is being filtered out.

White list validation is appropriate for all input fields provided by the user. White list validation involves defining exactly what IS authorized, and by definition, everything else is not authorized. If it's well structured data, like dates, social security numbers, zip codes, e-mail addresses, etc. then the developer should be able to define a very strong validation pattern, usually based on regular expressions, for validating such input. If the input field comes from a fixed set of options, like a drop down list or radio buttons, then the input needs to match exactly one of the values offered to the user in the first place. The most difficult fields to validate are so called 'free text' fields, like blog entries. However, even those types of fields can be validated to some degree, you can at least exclude all non-printable characters, and define a maximum size for the input field. \*<https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet>

1. Using code on the application side we can filter out input as it is supplied to the username and text field:

 

The above code takes an input string (i.e. username and password) and filters out any characters that are not a-z, A-Z, 0-9, period, comma, question mark, parentheses or a space. By sending the input from both username and password through the function above you can control exactly what is sent to the database.

1. Repeat the exact same steps from the hacking example entering in the malicious info, but check the Enable Input Validation checkbox before clicking Login
2. Using the same code you used above to successfully login to the database paste the Server Response and SQL Code sent to the Server below:
3. Take note of the string that is sent to the server
4. You should no longer be able to login to the server since the single quotes used to perform the attack before are being stripped out as malicious input.
5. Congratulations! You have just mitigated a SQL Injection vulnerability!

**Conclusion**

There were 5 critical vulnerabilities in this application which could allow a user to bypass the login form and gain access to the application without really being an authorized user. Please provide all 5 issues below and brief description of how each resulted in a security flaw:

**Extra Credit**

There is another really glaring vulnerability in this application which can be exploited without any tools beyond a web browser (Hint: Google “Direct URL Access”).