Antonio Ramirez

Bad Tutor Web Application Analysis

SDEV300

Table of Contents

[Plan of Attack 3](#_Toc14034093)

[Initial Functionality and Impressions 3](#_Toc14034094)

[Eyes-On Code Review 4](#_Toc14034095)

[OWASP ZAP Analysis 5](#_Toc14034096)

[Mitigation Plan 5](#_Toc14034097)

[Conclusion 6](#_Toc14034098)

# Plan of Attack

For this week’s project, we were asked to take a pre-made web application, Bad Tutor, and use the OWASP ZAP tool to find, analyze, and mitigate security risks. My initial plan of attack to tackle this assignment was to first go through the site in my web browser and evaluate the functionality. I logged in with all the provided usernames and passwords. I went through each of the features and made note of what the intended function was. After I evaluated how the site was currently functioning, I looked over the code and became familiar with the “how” of the site. While looking through the code I was also looking for any obvious security risks that were apparent to even my untrained eye. After becoming familiar with the code and making note of any obvious security issues, I would run an attack through the ZAP application.

# Initial Functionality and Impressions

As previously stated, I first wanted to get a basic feel for the site and understand what the user would see and how it would operate. For myself, this is always the first step I take when working with a new application. I logged in with all 3 of the provided user accounts and went through all the possible functions. While doing this I made some mental notes of some areas that could possibly be vulnerable. The most obvious is a SQL Injection vulnerability on the login form. Also, of note regarding the login were the obvious and easy-to-guess usernames and passwords with each of the users. Once logged in we can see a list of courses, student names, emails, tutor session details, locations, help details, and an option to delete the entry. When we select “Delete Session?”, we get taken to a confirmation page on which we can either select yes or no. When we select no, we get a message confirming the entry has NOT been deleted and we can return to the list of tutoring sessions. However, upon returning, we no longer get the same table as before. The new table is missing the student names which can make it difficult for the user to discern which session is which. While this is not an apparent security risk, it does impact user experience. On the other hand, when selecting yes on the deletion confirmation page, we get a confirmation message that the entry has been deleted and the option to return to view the sessions. Upon returning we can see that the session has in fact been deleted. This confirms that the database is being properly accessed and the user has the correct permissions to delete entries from the database. Another area for concern is the extent of the permissions the user has. This once again cannot be verified until we look at the code.

Returning to the login page we also have the option to upload a resume. We can select a file from the computer through the browse button and then submit the file using the send file button. I made a test document and uploaded it to the site. The page updates once you select a document and displays the documents name. Once we send the file we get take to an interesting screen that displays some unnecessary data to the user. Instead of just displaying a message informing the user of the successful file upload, we also get some debugging info that was possibly implemented during production of the app and was never removed.

The final option on the login page is the option to “Sign out Guest Book.”. We then have a form that can be filled out with a name and comment. Once again, we are unsure as to whether this form is vulnerable to SQL Injection. Once the form is filled out and submitted, we are once again taken to a confirmation page. The message itself is not very user friendly as it states, “Successfully entered the data 1”. I am unsure as to whether this is a security concern as of now. Along the same user-friendliness vein, there is no option to return to the main login page from the confirmation page. Again, this is not necessarily a security risk, but does affect the user experience.

The last thing that is made note of within the pdf document is the lack of an option to logout. This is a vulnerability that needs to be addressed.

# Eyes-On Code Review

After reviewing things through the browser, it was time to look through the code itself. The main login page comes from the index.html file. The main components of the page were the login form that called on action authcheck.php, as well as 2 others .html links for uploading a resume and signing the guest book. My next step was to look into the authcheck.php file. As previously noted the main concern for the form was SQL Injection. To authenticate the user, findTutor() is used which belongs to the SQLFunctions.php file. Once looking at the actual query, we can see that the user input username and password are not prepared and are just queried. This is an obvious vulnerability for SQL Injection that must be addressed. While we are looking at the SQLFunctions.php file we should also look at the other functions and queries to see if they also have the same vulnerability. Wouldn’t you know it, they do. While I haven’t gone through the rest of the code to see where these functions are accessed, I have a feeling that they are accessed in other parts of the application and fixing the way we run these queries will clear up a lot of the SQL Injection vulnerabilities.

Next to evaluate was the UploadResume.html file. We can now see that there is a hidden field called MAX\_FILE\_SIZE that is commented as having to precede the file input field. We also see that the form on action calls uploadMe.php. Upon evaluating that file, we see that it is the culprit for the unnecessary information being shown on the confirmation screen. It appears that we can simply remove the echo and that should solve that issue.

Now we will evaluate the guest book function. The main form can be seen in the GuestBook.html file. Just like the previous forms, this form is no exception to the SQL Injection vulnerability. When looking at the guestRecord.php file we see once again that the form relies on functions that are not using prepared statements to mitigate the threat of SQL Injection. To address the weird success message that appears after submission of your comment. We can see that the “1” appears due to the use of a success variable that is outputting 1 to the text. It may just be my personal preference, but I think the user experience would be improved by removing this variable from the echo message.

The last feature I will be looking into is the delete function. When we click the link in the table we are taken to the DeleteSession.php file. We see that the form used on the page calls Deleteit.php. It then calls a cancelSession() and deleteSession() method to remove the entry from the database. While the query doesn’t use user input to query the parameters, the query does not use prepared statements once again. I personally do not see a vulnerability since there is no area for user input here, but I don’t think it would hurt to use prepared statements.

# OWASP ZAP Analysis

The next step I would take is to run an automated attack through the OWASP ZAP tool available on the AMI. Unfortunately, I am unable to do so for several reasons. I had experienced issues with getting the browser and tool to communicate properly to display all the necessary information needed to make an assessment. I have tried changing from localhost, changing ports, as well as certificates. I believe it had been established the problem could possibly be the AMI itself. The prior solution used was to run the ZAP AMI available through the AWS Marketplace. To my surprise this is not longer available. I had exhausted all troubleshooting and fixes that were within my ability and as a result was unable to run the ZAP tool as intended. I was curious as to what other vulnerabilities, big and small, the tool would discover. I am sure it would detect the SQL Injection vulnerability, but possibly other vulnerabilities my untrained eye was unable to see.

# Mitigation Plan

As we can see there are many areas in which the program can be made not only more secure, but more user friendly. The first step would be to modify the SQL queries to use prepared statements to reduce the vulnerability of Cross Site Scripting. Fortunately, this program was written with most all the query methods in a single file, SQL Injection. This is especially helpful having the query logic in a single place which prevents having to scan through each form looking for unsecure query statements.

The next step would be to add a logout button to the application. This is not only a user-experience improvement but also important for application security. Since the application uses session data to store and transfer data, being unable to logout is a security concern. Ideally, we would want this button to be available at any time after the user logs in. This will allow the user to safely leave the application without the risk of another simply sitting down at the same pc and having access to all the secure information.

Another improvement that is more geared towards user experience is the addition of “return home” buttons on most of the pages that are UI “dead-ends”. A prime example of this is the guestRecord.php file confirmation page. After successfully submitting a comment we have no way to return to the login page, besides retyping the address. While this isn’t necessarily a security concern, it would drastically improve the user experience.

Returning to the login page, implementing minimum password requirements would be beneficial. The provided usernames and passwords are remarkably easy to guess, and that is a vulnerability. By implementing password minimum requirements, as we have in prior labs, would make guessing a user’s login a lot more difficult. In my research it seems that password requirements may not make a site any more secure. By broadcasting the password requirements, we are informing potential attackers that a password will always contain certain numbers of values at a minimum. However, for the sake of this application, I think implementing password requirements would be better than not.

Another user-experience fix would be to display the same table as before after accessing the delete forms. As previously stated, the table that appears after the fact does not show student names, which makes it difficult for the user to discern which session is which. This could possibly lead to an erroneous deletion of a session.

The next area for improvement is the file upload process. Selection of a file and submission does not appear to be a concern. The problem arises when the confirmation page loads. The page displays unnecessary information about the user’s PC that doesn’t need to be there. It is my impression this was out in there during the construction of the application for possible troubleshooting. The message does not contribute anything valuable to an honest user but could possibly be exploited by a malicious user. This is once again an easy fix by simply removing the echo statement.

All the above improvements will either impact application security, user-experience, or both. The priority fixes should be those that have an impact on the site’s security. This list of vulnerabilities and issues is by no means complete due to my inexperience and the inability to use the ZAP tool as intended. I am sure if we had been able to properly use the tool we would have other areas that would require attention.

# Conclusion

The biggest threat that was evaluated was SQL Injection. This is unsurprising since the course seemed to have an emphasis on the subject. To my understanding, the primary way to combat this is to use prepared statements in your SQL queries. It is also my understanding, that this kind of attack is prevalent in forms and other areas of user input where a query would be used. I am interested to see if there any other areas that could be vulnerable to SQL Injection.

The secondary area of user-experience can sometimes be just as important as security and experience can go hand-in-hand. An example of this is the implementation of a logout button. It is something that is a staple in almost all applications, and its importance can be easily overlooked.

I hadn’t had any experience with php or SQL prior to this course. It was interesting to see how this all works together and how malicious users can exploit the code. As our reliance on the internet grows, so does the need for application security. I understand we have only scratched the surface of building a secure web application and I look forward to gaining more experience in the field.