Web Application Vulnerabilities

Joseph E. Turner

Metropolitan State University of Denver

Abstract

The purpose of this paper is to examine and understand common vulnerabilities in web applications. The goal is to gain knowledge of common exploits and how to protect against those exploits. The method employed will be to employ common exploits against a vulnerable web application until the desired results are achieved. The result is an understanding of common exploits and how to protect against them.

Web Application Vulnerabilities

Motivation

The purpose of this paper is to explore various exploits that may be used to break into vulnerable web application. The exploits used for research demonstrate potential risks associated with failing to properly secure a web application. Understanding potential vulnerabilities, how vulnerabilities may be exploited, and the consequences of exploiting vulnerabilities is critically important for securing every web application against potential threats.

Introduction

The problem with any web application is the risk of failing to properly understand and protect against potential vulnerabilities. For the purpose of understanding web application vulnerabilities, this paper examines an intentionally vulnerable web application named WebGoat. The exploits used against WebGoat include two structured query language attacks and one cross site scripting attack. In addition to WebGoat, Zed Attack Proxy will be used to intercept and modify communications between a web browser and the WebGoat application.

Method

Setup and configuration used for the experiment involves several steps. The examples given within use Windows 10 operating system and Firefox web browser in addition to WebGoat and Zed Attack Proxy. The version of WebGoat used is 7.1, which requires Java Development Kit 8 in order to run properly. Java Development Kit 8 should be downloaded and installed first, followed by Zed Attack Proxy, and Firefox or Chrome if a web browser is not already installed. Zed Attack Proxy offers manual exploration of web applications through either Firefox or Chrome web browser. After these steps were completed, WebGoat was downloaded.

Performing some exploits requires configuration of Zed Attack Proxy and a web browser. For the examples given within, a manual proxy was configured in Firefox web browser. After configuring the web browser proxy as shown in Figure 1, Zed Attack Proxy was started and configured as the proxy to be used by Firefox so all requests from Firefox go through Zed Attack Proxy as shown in Figure 2. Before launching the web browser from the Zed Attack Proxy, the WebGoat application was started by using the appropriate command in Windows PowerShell as shown in Figure 3. Once WebGoat was running, Firefox web browser was launched from the Zed Attack Proxy application. The final step was to use the web browser to navigate to the address for the WebGoat application as shown in Figure 4. Once WebGoat was up and running the following exploits, detailed below, were attempted: a numeric structured query language attack, a string structured query language attack, and a cross site scripting attack.

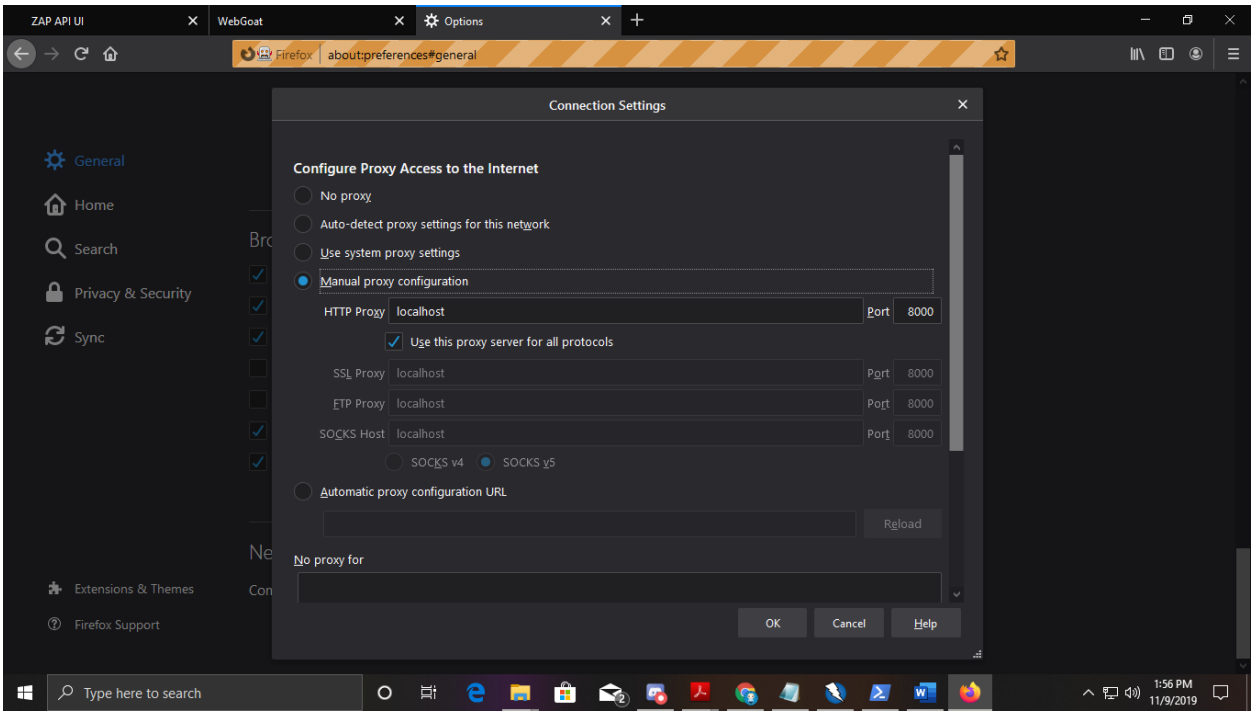

Figure 1. Proxy Configuration for Firefox web browser.

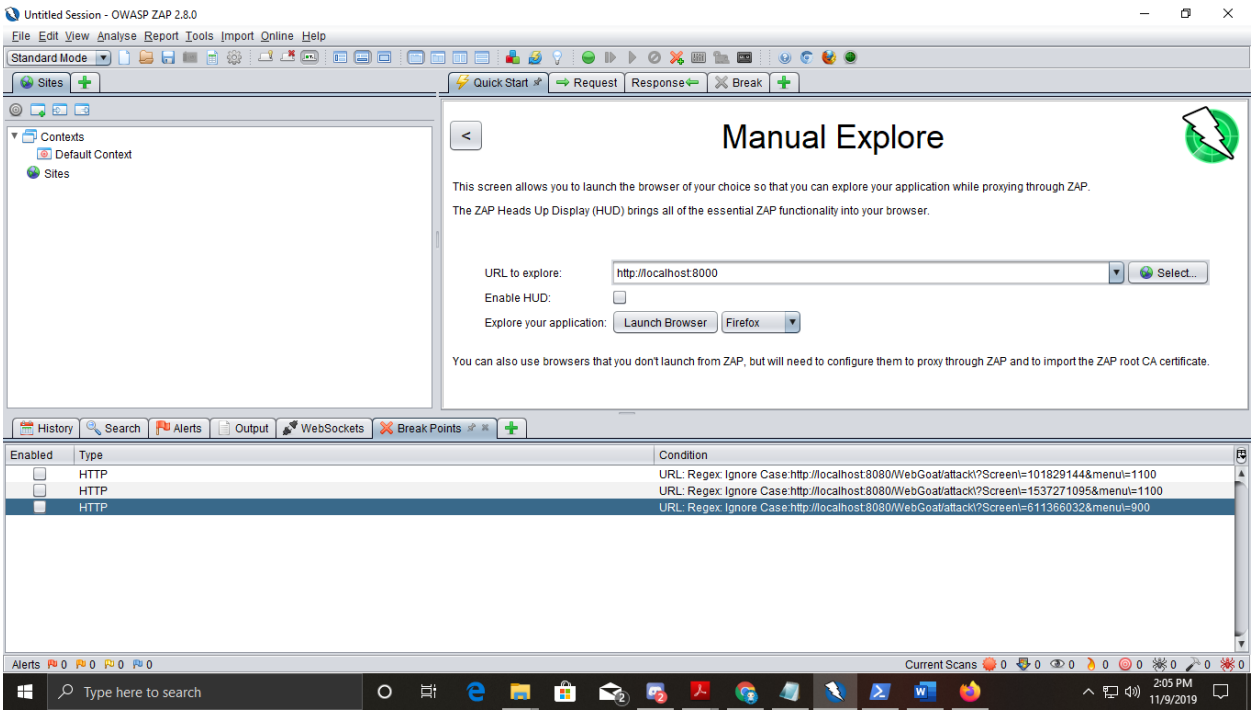

Figure 2. Configuration of Zed Attack Proxy for listening to requests from Firefox web browser.

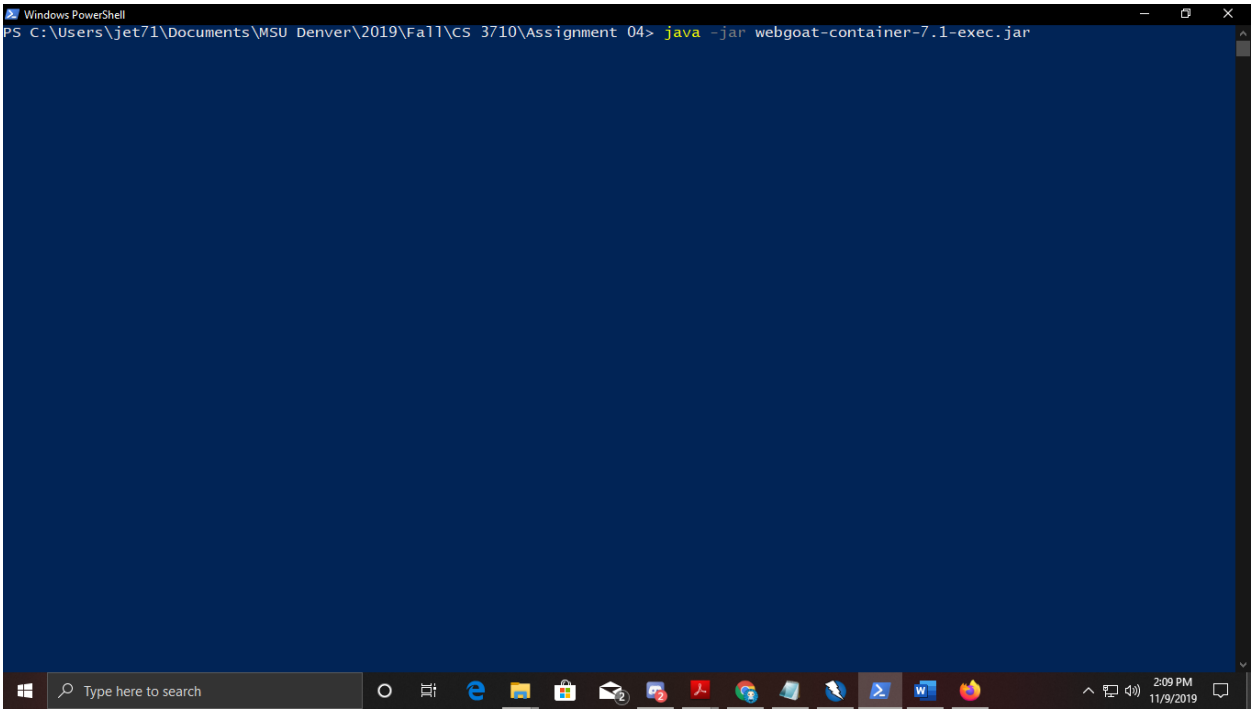

Figure 3. Starting WebGoat application via Powershell.

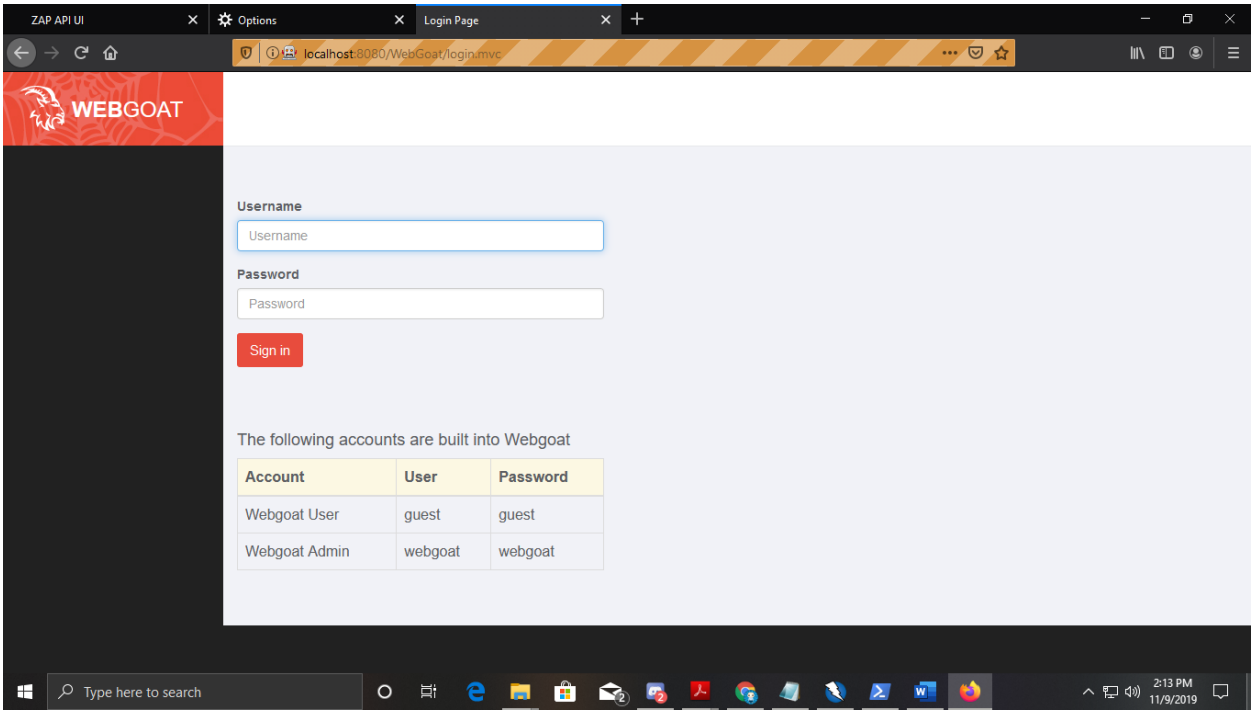

Figure 4. WebGoat running in Firefox web browser.

Injection Flaws - Numerical SQL Injection

WebGoat offers a several lessons on injection flaws. The Numerical SQL Injection lesson involves a web page which may be used to retrieve weather data for a single geographic location, as shown in Figure 5. The goal of the lesson is to implement an SQL injection which results in weather data for all locations in the database being displayed.

For this lesson the button for requesting weather on WebGoat was pressed, and the generated request was intercepted by Zed Attack Proxy, allowing viewing of the specific SQL query generated by WebGoat when the request was made. As shown in Figure 6, a break point was set in Zed Attack Proxy for the specific request. The breakpoint allowed for interception of the request, and modification of the SQL query, before the request is passed to WebGoat as shown in Figure 7. From this point repeated intercepts and query modifications could be performed until the desired result was produced.

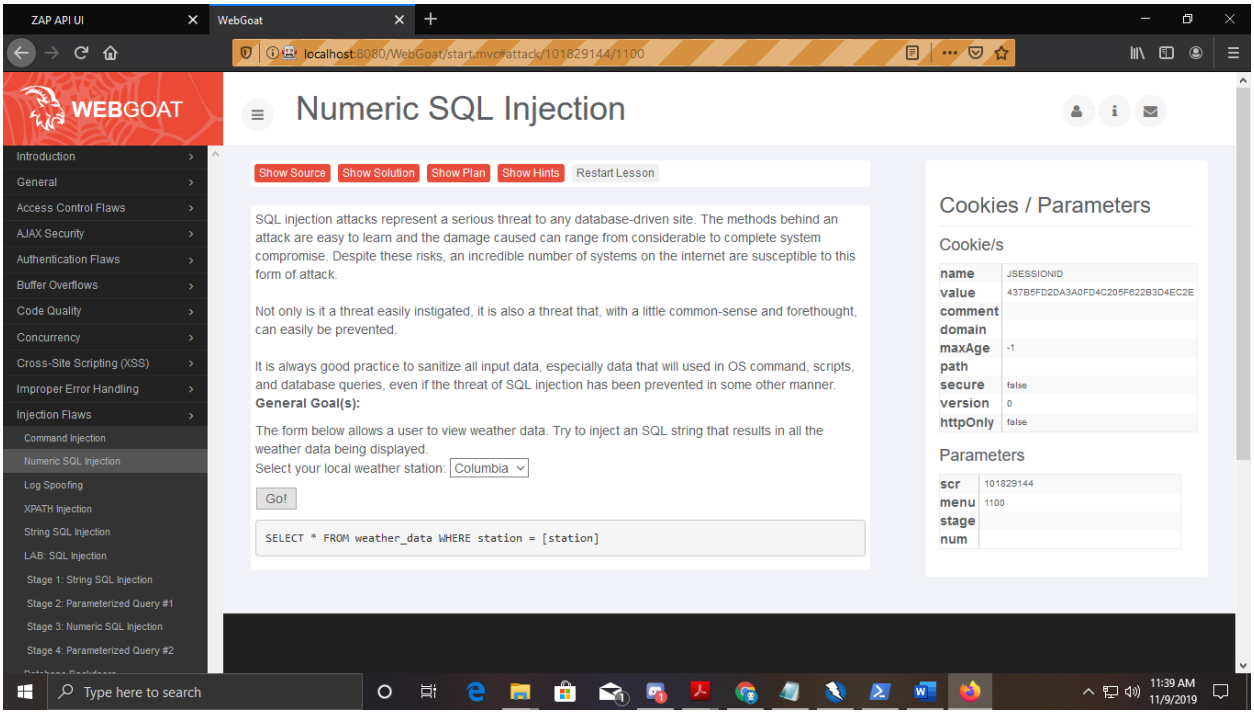

Figure 5. Numerical SQL Injection lesson in WebGoat.

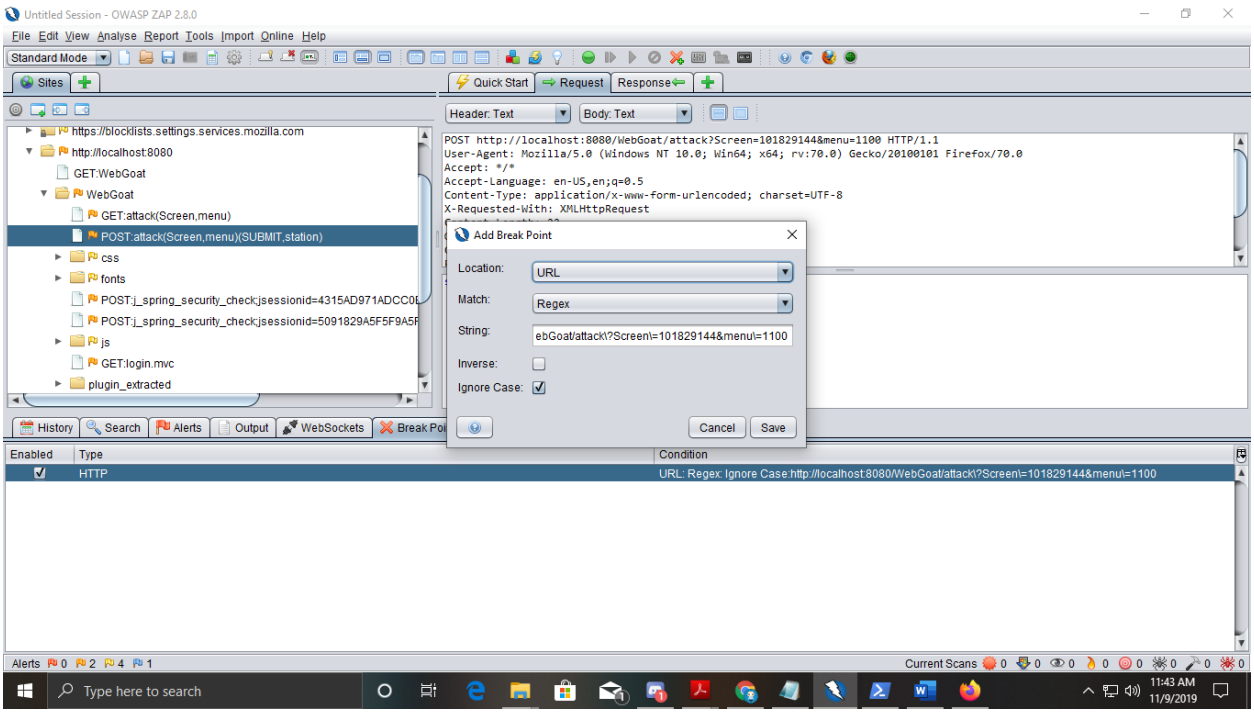

Figure 6. Setting a breakpoint for a POST request in Zed Attack Proxy.

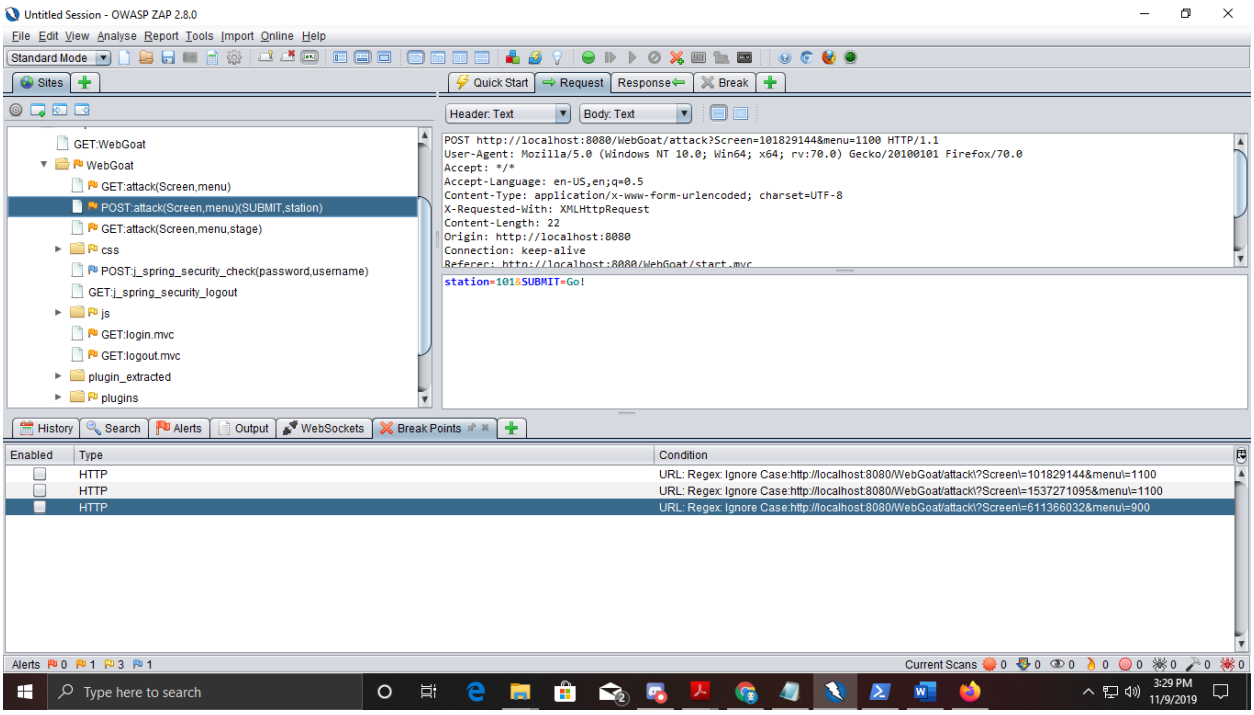

Figure 7. Intercepted POST request and SQL Query in Zed Attack Proxy.

Injection Flaws - String SQL Injection

This attack attempted involves using an SQL Injection to subvert a login page. The goal given for this WebGoat lesson page is to, “Use String SQL Injection to bypass authentication. Use SQL injection to log in as the boss ('Neville') without using the correct password.” (The OWASP Foundation, n.d.)

For this lesson the WebGoat page displays a drop-down menu containing user names, a text field for entering a password, and a button to press for submitting the login credentials, as shown in Figure 8. Pressing the login button allowed for the same capture of the request described in the previous example as shown in Figure 9. From that point the steps of setting a breakpoint in Zed Attack Proxy, capturing the request, and modifying the SQL query from the

previous example could be used.

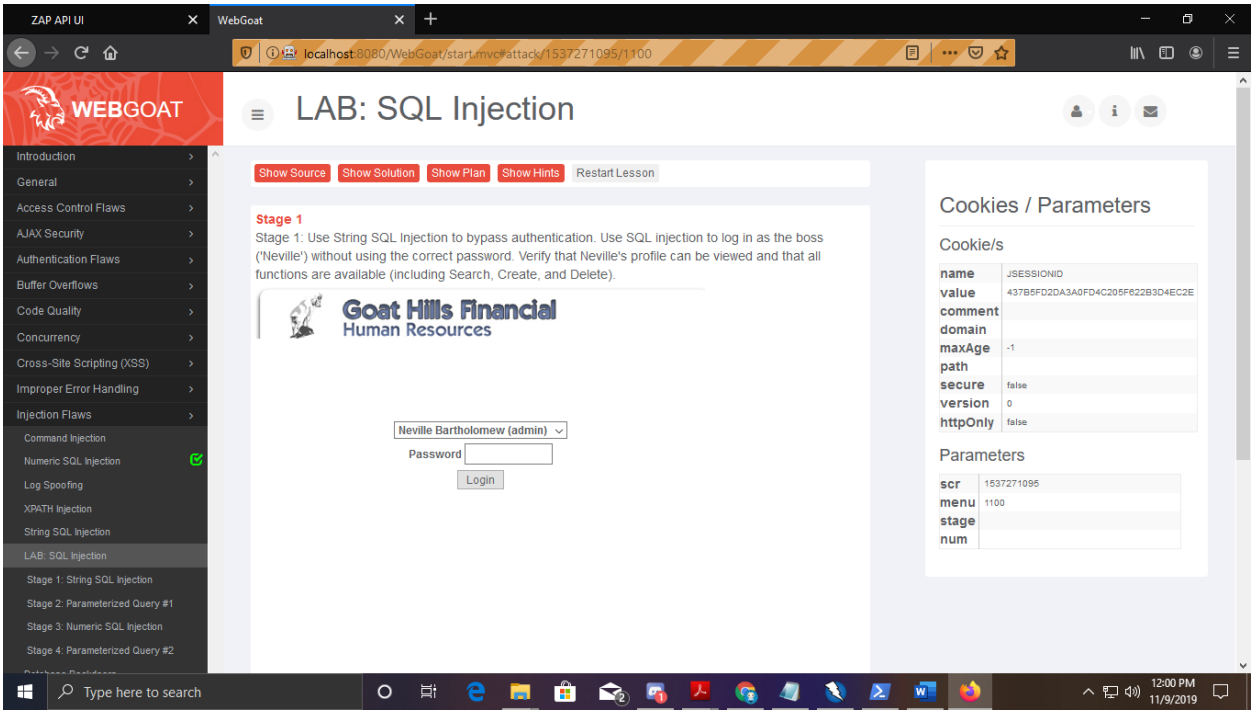

Figure 8. String SQL Injection in WebGoat Injection Flaws Lab.

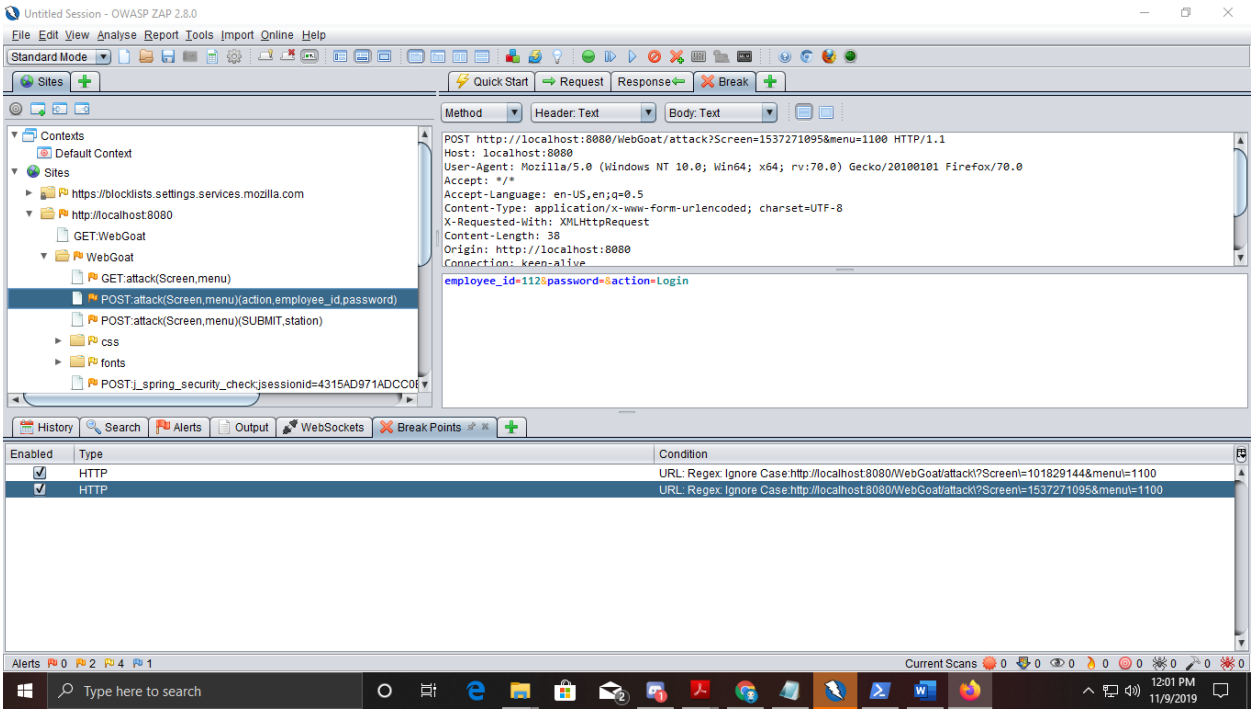

Figure 9. Zed Attack Proxy capturing a POST request and SQL Query from WebGoat.

Cross Site Scripting – Stored XSS Attack

According to the goal for this WebGoat lesson, “The user should be able to add message content that cause another user to load an undesireable page or content.” (The OWASP Foundation, n.d.)

For this lesson, the page displays a from which allows the user to enter a title and message to be stored and a list of stored messages, as shown in Figure 10. Entering some text into both fields and pressing a button would store the message then reload the page to display any stored messages. Use of Zed Attack Proxy was not required for this lesson because there was no authentication involved for submitting a message. All that was required was to submit a message containing some code and then clicking on the link to retrieve the recently added message and see if the submitted code would execute upon retrieval of the message.

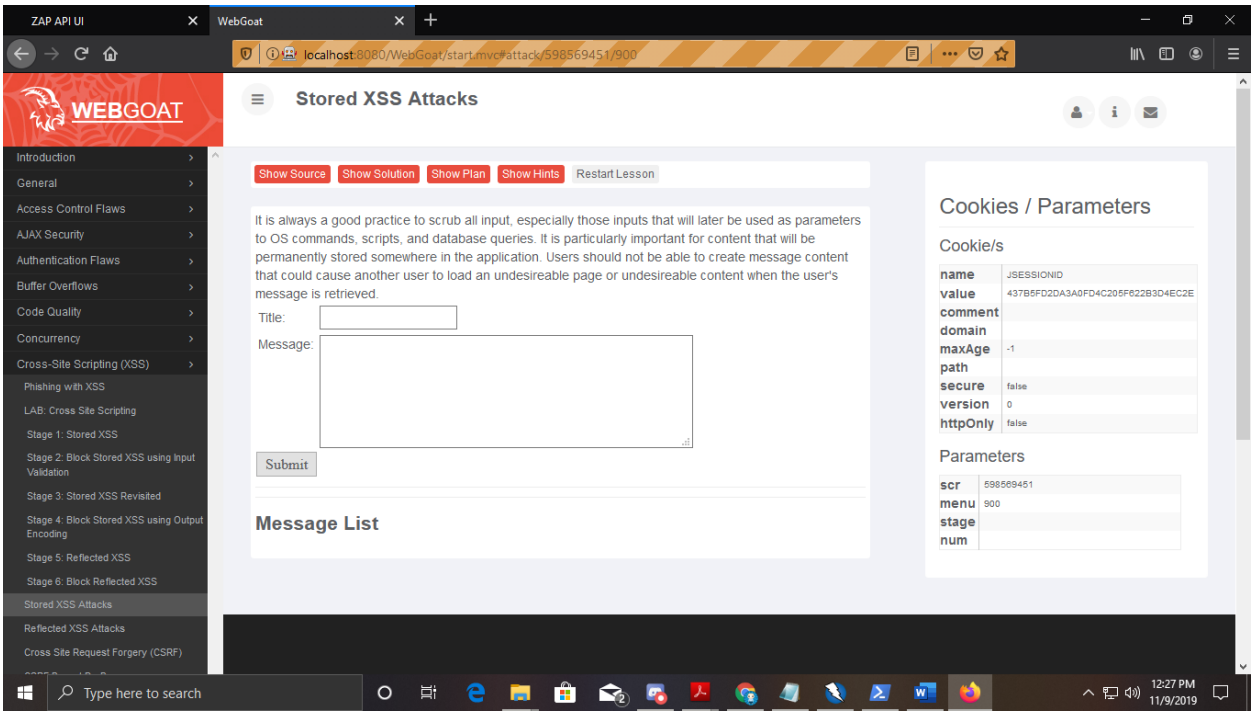

Figure 10. WebGoat XSS Attack lesson.

Results

Injection Flaws - Numerical SQL Injection

The solution to this lesson was to modify the portion of the SQL query specifying the station identifier to include a logical statement which evaluates to true, forcing SQL to accept that the query is valid for any station identifier, as shown in Figure 11. The result is all records in the database table being returned, as shown in Figure 12.

According to the WebGoat application, “SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this form of attack. Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented. It is always good practice to sanitize all input data, especially data that will used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.” (The OWASP Foundation, n.d.)

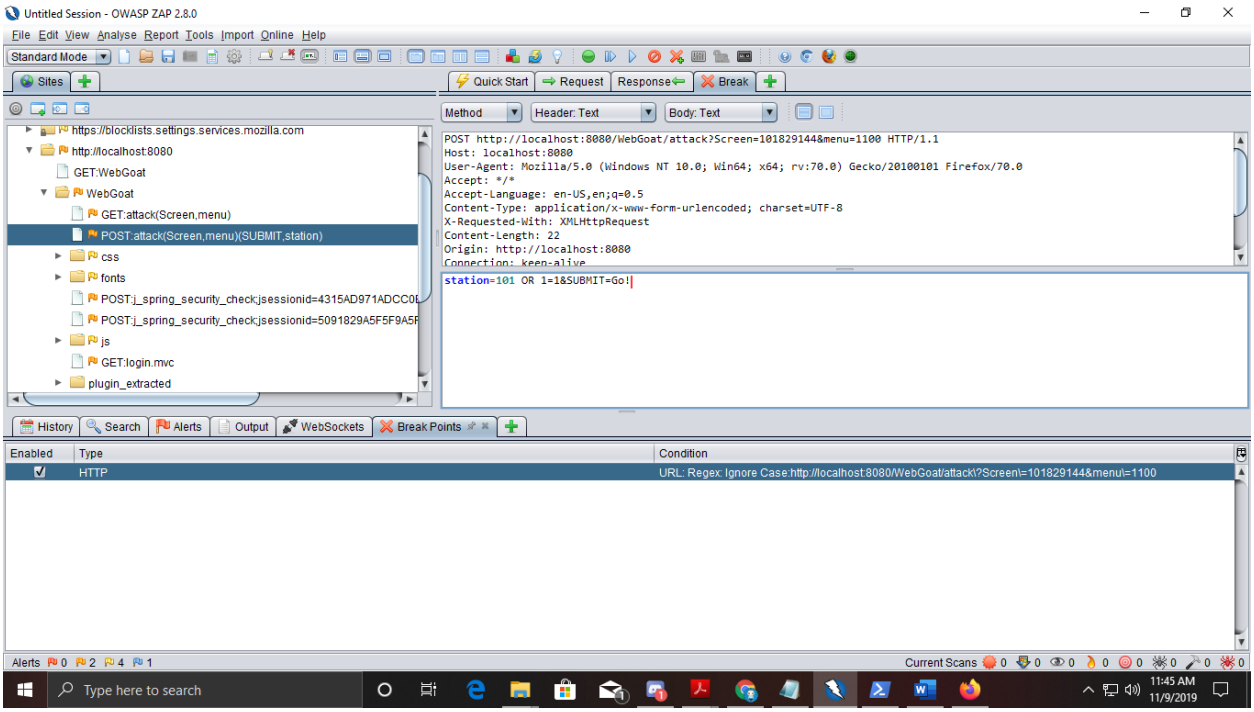

Figure 11. Query sent via Zed Attack Proxy used in Numeric SQL Injection attack.

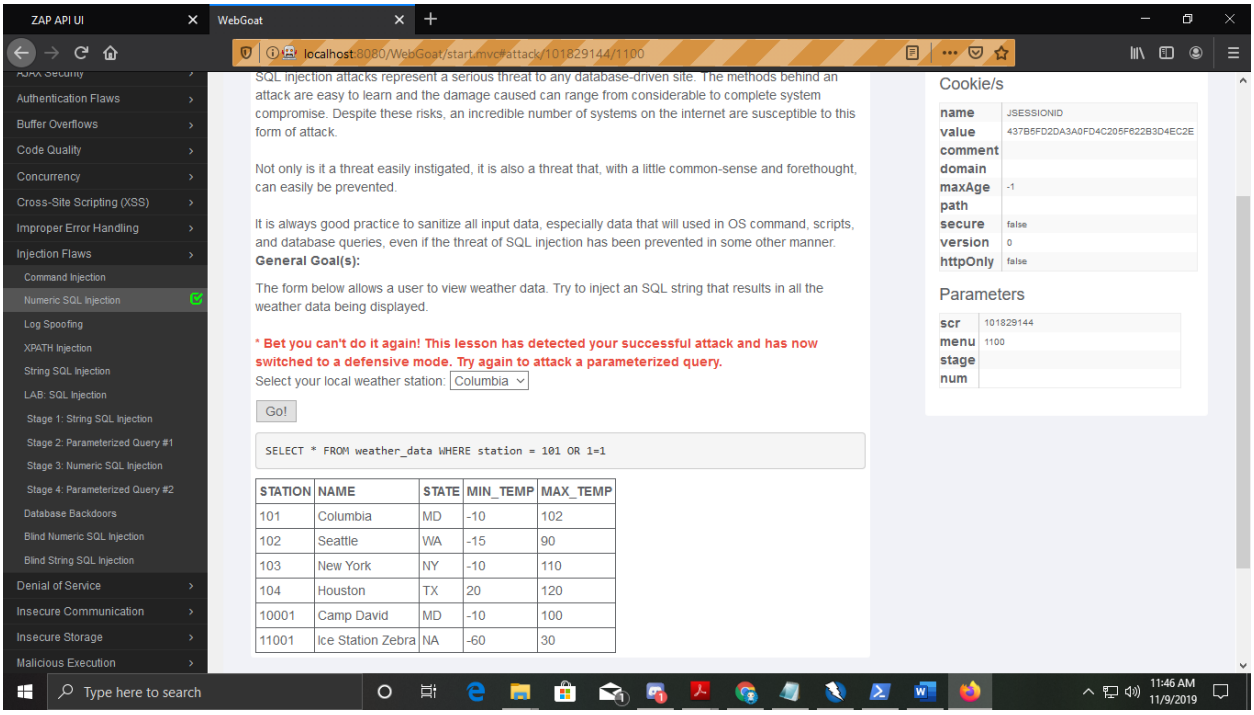

Figure 12. Successful Numeric SQL Injection attack against WebGoat.

Injection Flaws - String SQL Injection

The solution to this lesson was to modify the portion of the SQL query specifying the password to include a logical statement which evaluates to true, forcing SQL to accept that the condition for the password portion of the query is always true, as shown in Figure 13. The result is a successful login and a gain of access to all features for which the user is authorized, as shown in Figure 14.

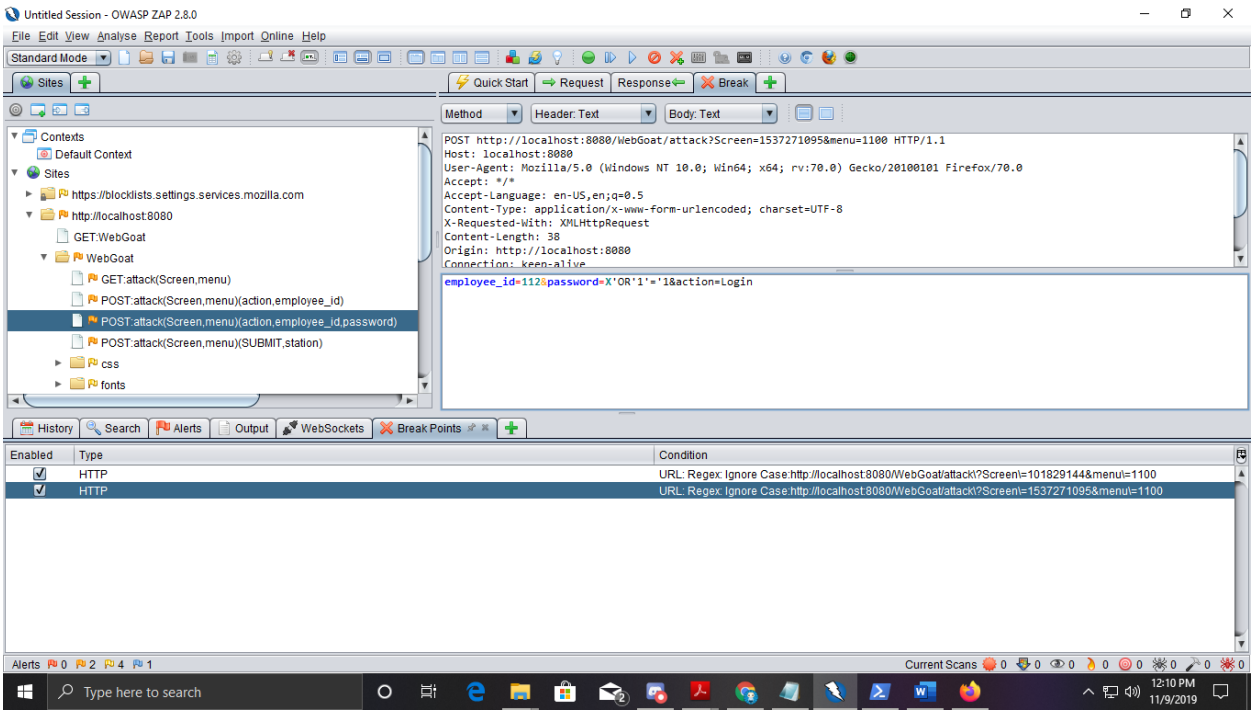

Figure 13. String SQL Injection sent via Zed Attack Proxy.

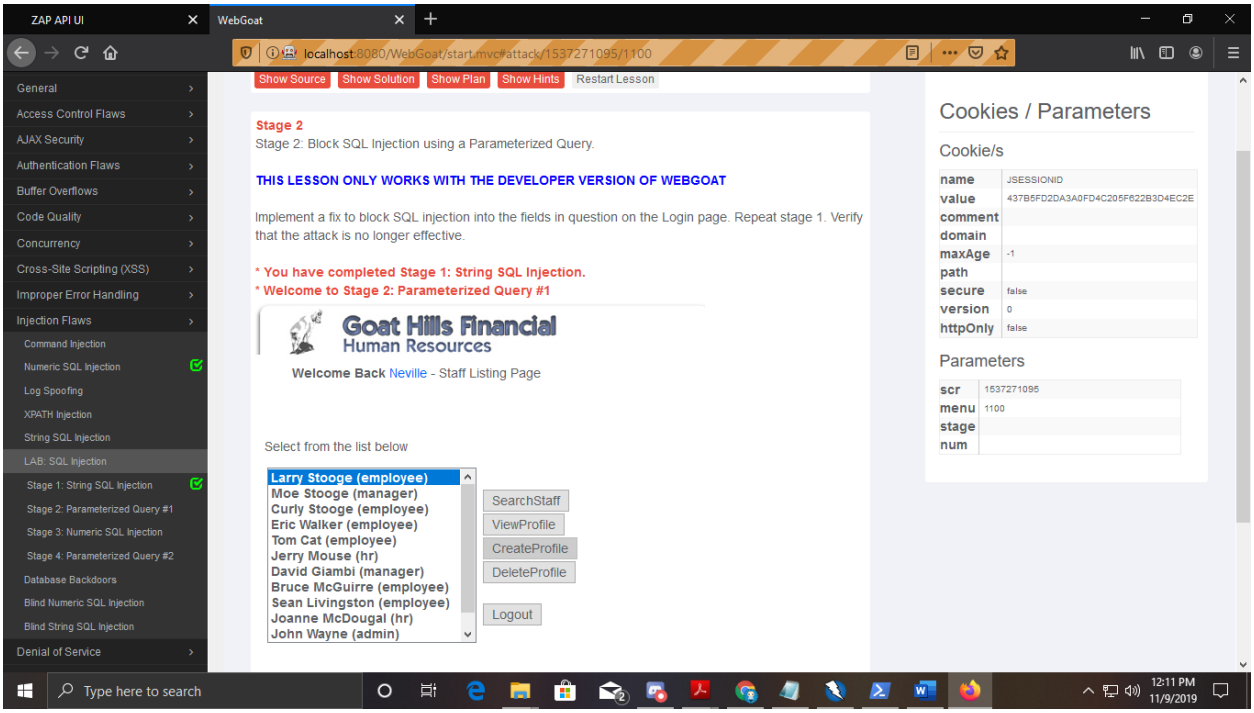

Figure 14. Successful SQL String Injection attack against WebGoat.

Cross Site Scripting – Stored XSS Attack

The solution to this lesson was simple. When submitting a message, the attacker simply needs to include some code which the browser could interpret and execute when the message is retrieved by clicking the link to the message in the list of messages. For the purpose of accomplishing this task, some relatively simple JavaScript for displaying a popup was added to the message field, as shown in figure 15. When the link to the message was clicked, the JavaScript was executed and displayed a popup with a short message, as shown in Figure 16.

According to WebGoat, “It is always a good practice to scrub all input, especially those inputs that will later be used as parameters to OS commands, scripts, and database queries. It is particularly important for content that will be permanently stored somewhere in the application. Users should not be able to create message content that could cause another user to load an undesireable page or undesireable content when the user's message is retrieved.” (The OWASP Foundation, n.d.)

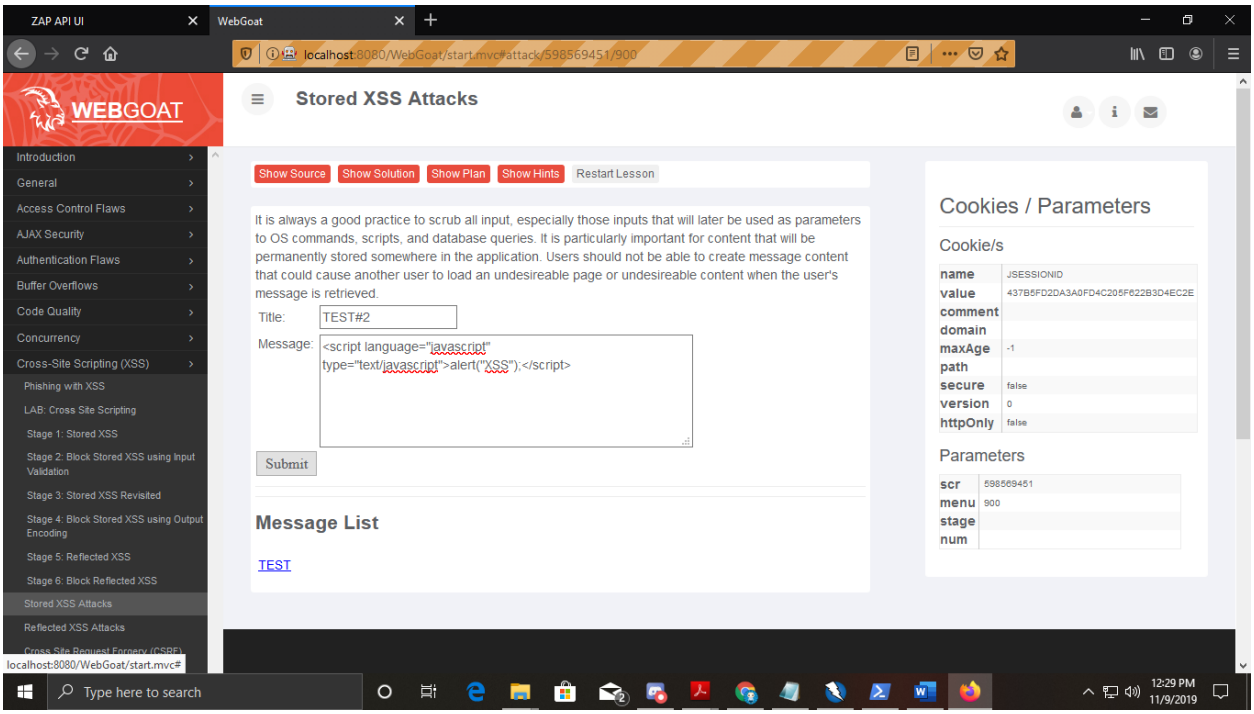

Figure 15. JavaScript used in XSS attack against WebGoat.

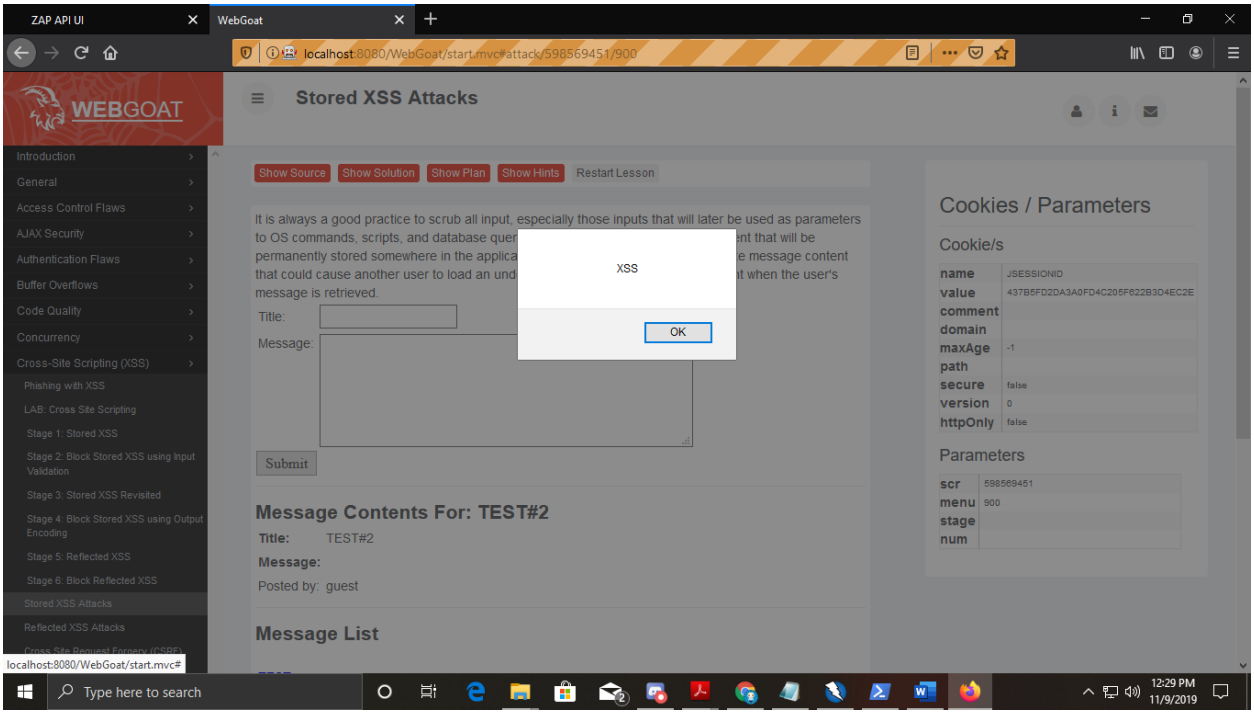

Figure 16. Successful XSS attack against WebGoat.

Conclusion

As demonstrated by the first two lessons, SQL injection attacks are a serious concern. Failing to protect a web application from such attacks may result in a malicious attacker gaining access to important information such as user credentials which could be used in a variety of crimes such as identity theft. As discussed in the lessons, proper sanitization of input data is essential for protecting against injection attacks. As demonstrated by the third lesson, Cross site scripting attacks are also a serious concern. Failing to protect a web application from executing external code may result in a malicious user gaining access to a server, information stored on that server, or even provide a malicious user the opportunity to force malicious software to be downloaded by all who visit a compromised web application. As discussed in all three lessons, proper sanitization of input data is of the utmost importance in preventing attacks against and protecting a web application.

Reflection

Several lessons were learned during the experiments with WebGoat. First, having the proper version of Java installed is imperative for proper functioning of WebGoat. Second, using Zed Attack Proxy for the SQL Injection attacks is something like a man in the middle. Third, sanitization is of critical importance for all input data. All three of the vulnerabilities exploited could have bee prevent with proper sanitization techniques. Both SQL injection attacks may have been prevented, perhaps by checking the query parts against regular expressions. The cross-site scripting attack may have been prevented by checking the input entered in the message field for inappropriate characters or parts that match code. Lastly, writing test that check portions of web applications against potential attacks should always be part of developing a web application.

References

The OWASP Foundation. (n.d.). *Cross-Site Scripting (XSS) > Stored XSS Attacks*. Retrieved from WebGoat: http://localhost:8080/WebGoat/start.mvc#attack/598569451/900

The OWASP Foundation. (n.d.). *Injection Flaws > LAB: SQL Injection - Stage 1: String SQL Injection*. Retrieved from WebGoat: http://localhost:8080/WebGoat/start.mvc#attack/1537271095/1100/1

The OWASP Foundation. (n.d.). *Injection Flaws > Numeric SQL Injection*. Retrieved from WebGoat: http://localhost:8080/WebGoat/start.mvc#attack/101829144/1100