Joshua Hoshiko

CS3710

Steve Beaty

November 8, 2019

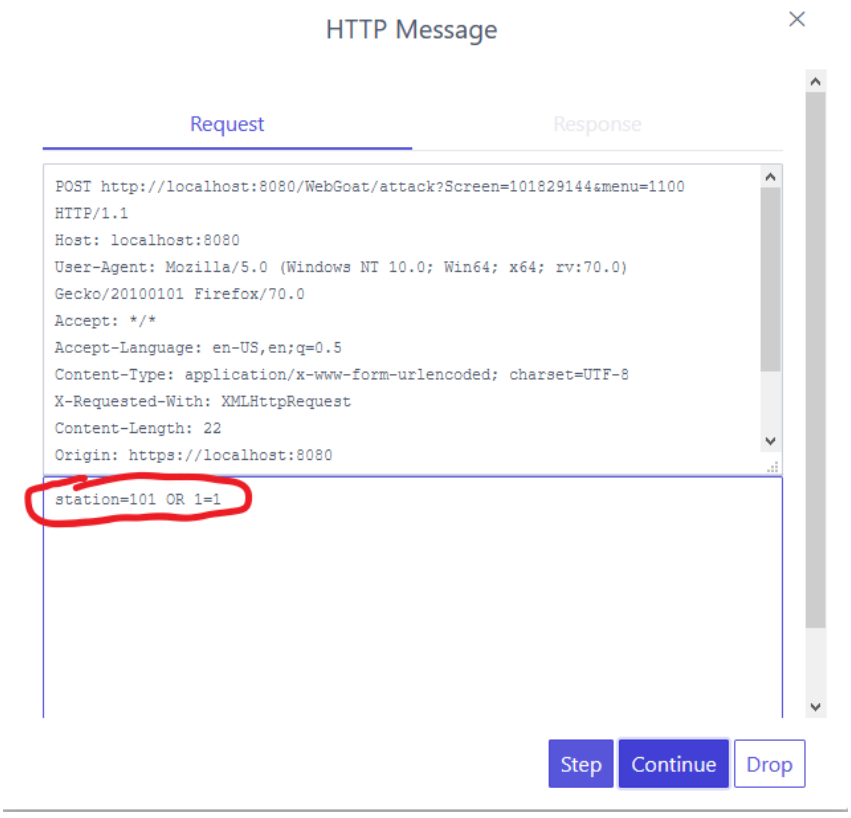
Project 4

Abstract: In this paper, I will discuss several different penetration tests using WebGoat using common methods of attack. These methods are SQL injection and XSS (Cross Site Scripting). WebGoat is a deliberately vulnerable web application that allows users to experiment with attacks that frequently occur to all web servers. Other tools include OWASP ZAP for HTTP request intercepting.

Motivation: This paper will cover commonly used exploits to attack web applications. It is important to know about these attacks when designing a web application because our applications are always being attacked. Knowing about these attacks will allow web application designers to defensively design their applications to prevent against such exploits.

Introduction: I completed three different exploits, two SQL injections and one XSS attack. The two kinds of injection were numeric and string injection. The idea is to intercept the POST and insert code that alters the statement to behave in an unintended way. With XSS attacks, the goal is to take advantage of how web applications store information between pages and use this fact to create malicious code that will persist to other pages or steal information from the stored information. In this case, I performed a malicious attack by inserting HTML code into the form to effect other employees viewing the profile I edited.

Methods and Results: SQL injection is a technique in which a user injects SQL code into their POSTs to the server. Generally, a vulnerable server does not sanitize the data that is being accepted from the user. In this case, the POST can be intercepted and SQL code can be injected. When determining what to inject, I knew that in order to get all of the weather information, the form was just completing a comparison statement and altering the statement would give me more information then intended. So if I injected the right code, then the statement would have always be true and print everything in the database. This is where a tautology comes in handy. Adding an OR statement with a tautology always results in true, and in turn, prints all of the weather information. Because the form uses a dropdown, I cannot inject directly into the form. In this case, using ZAP, I can intercept the POST and inject my exploit.



```
SELECT * FROM weather_data WHERE station = 101 OR 1=1
```

STATION	NAME	STATE	MIN_TEMP	MAX_TEMP
101	Columbia	MD	-10	102
102	Seattle	WA	-15	90
103	New York	NY	-10	110
104	Houston	TX	20	120
10001	Camp David	MD	-10	100
11001	Ice Station Zebra	NA	-60	30

The second SQL injection I did was string injection. In this scenario, there was a text field that accepts a last name and searches a database. The same exploit can be used here, but instead, you have to construct a tautology that utilizes strings including the quotation marks that are present in the statement. In this case, because the form uses a text box, I can just enter my malicious query directly into the form.

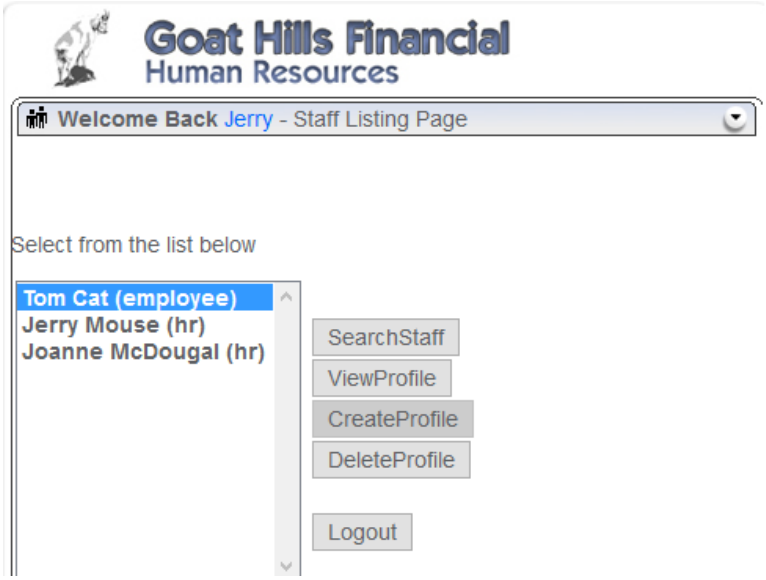
Enter your last name: Smith' OR '1' = '1 Go!

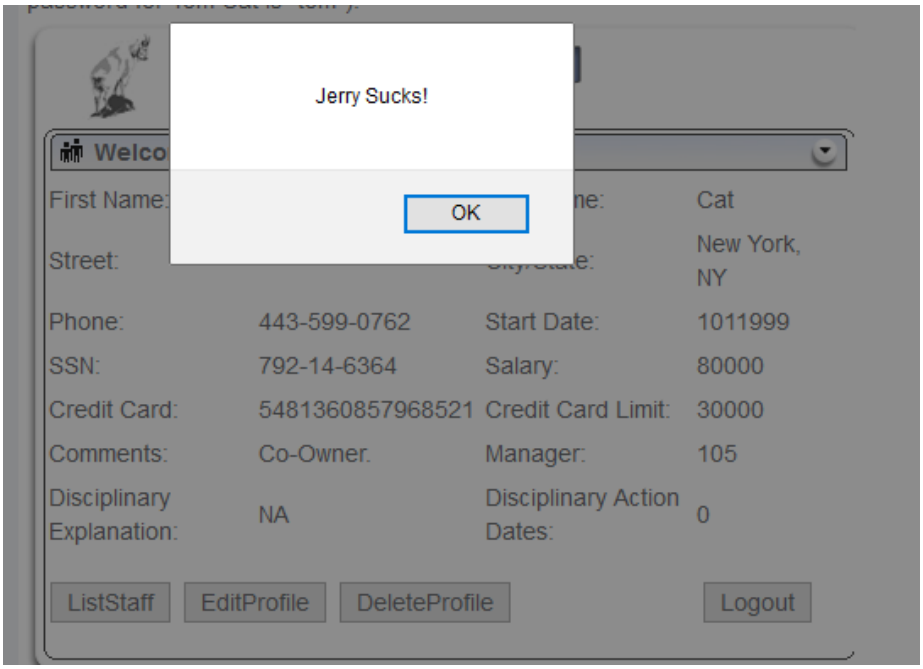
```
SELECT * FROM user_data WHERE last_name = 'Smith' OR '1' = '1'
```

USERID	FIRST_NAME	LAST_NAME	CC_NUMBER	CC_TYPE	COOKIE	LOGIN_COUNT
101	Joe	Snow	987654321	VISA		0
101	Joe	Snow	2234200065411	MC		0
102	John	Smith	2435600002222	MC		0
102	John	Smith	4352209902222	AMEX		0
103	Jane	Plane	123456789	MC		0
103	Jane	Plane	333498703333	AMEX		0
10312	Jolly	Hershey	176896789	MC		0
10312	Jolly	Hershey	333300003333	AMEX		0
10323	Grumpy	youaretheweakestlink	673834489	MC		0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX		0
15603	Peter	Sand	123609789	MC		0
15603	Peter	Sand	338893453333	AMEX		0
15613	Joesph	Something	33843453533	AMEX		0

For the final exploit, I needed to edit a form and add an HTML script in javascript to create a malicious pop up. In order to do this, I log into the “Tom Cat” profile and edit his information. In the street field, I add the code for a javascript alert with the desired message and then save the form. Now, when the supervisor “Jerry” or any other user views the profile, they will get the popup that I have added to the application.

Code: <script language="javascript" type="text/javascript">alert(“Jerry Sucks!");</script>





Conclusion: After performing these exploits, it is easy to see how simple it is to access information that we should not have access to or to add malicious code to applications to change their function to an unintended one. It is always good practice to sanitize information from the form because not only does it stop malicious users, but it also can prevent broken queries being sent from a malfunctioning form. Using frameworks also helps prevent against these attacks since they automatically filter our malicious queries. SQL injection and XSS are among the most

used exploits on the internet because application designers do not take into consideration what kind of attacks their application will face.

Reflection: I think the biggest take away from this assignment was how easy it was to gain access to information that I should not be able to access. I now understand why SQL injection is so common because malicious users can just roam the net attempting these exploits on web applications until they find a vulnerable one. Programming defensibly is even more essential when creating applications that many users will have access to.

Overall, the most difficult part of the assignment was setting up the tools necessary for the attacks (since windows is very uncooperative). However, once the tools were set up, the actual exploits weren't difficult because the source code revealed how the statement was working and using a guess and check strategy was pretty enjoyable.