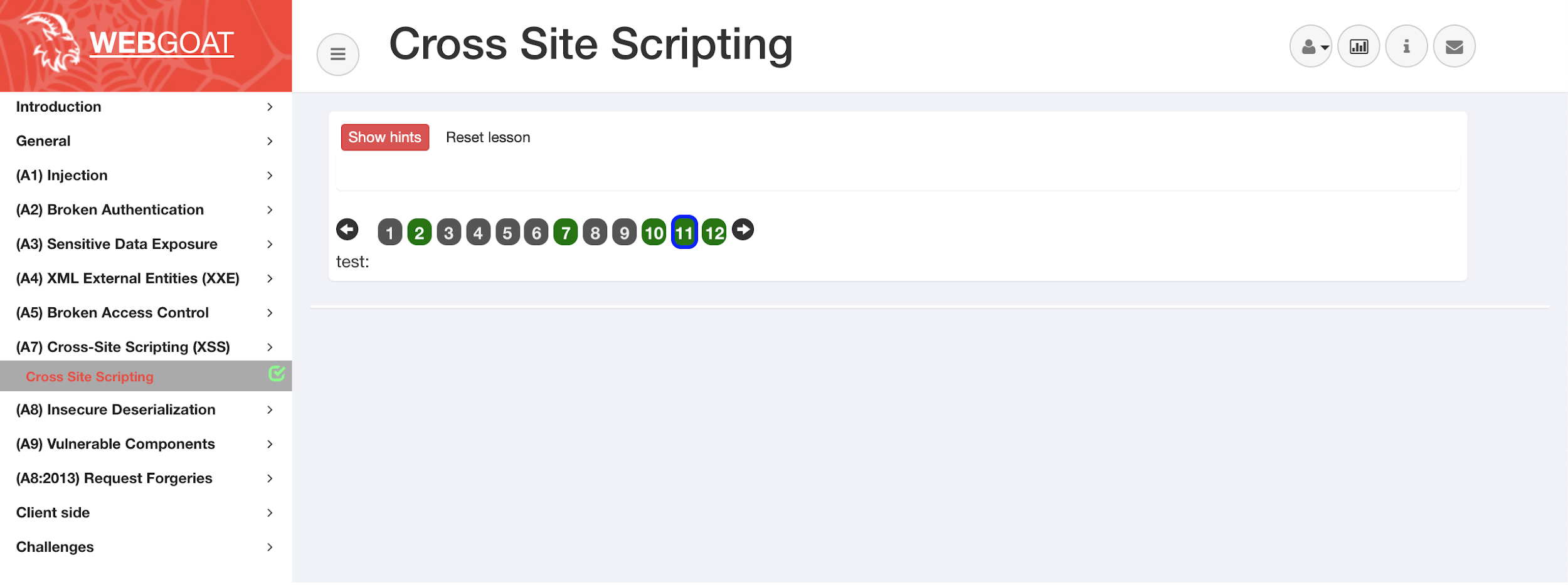
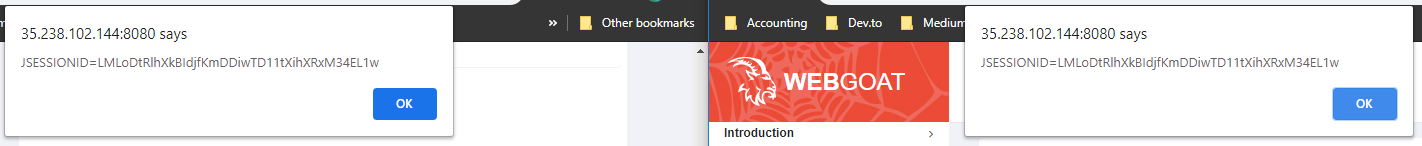
# WebGoat



## Section 2:

**Were the cookies the same on each tab?**

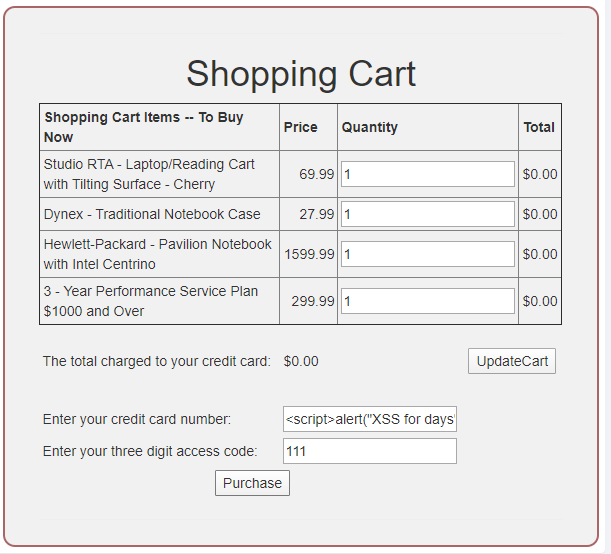
* Yes

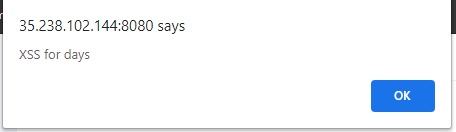


## Section 7:

**Enter your credit card number:**

* <script>alert(“XSS for days”)</script>





## Section 10:

**What is the route for the test code that stayed in the app during production?**

* http://34.68.19.121:8080/WebGoat/start.mvc#test/
* Found in webgoat>goatApp>view>GoatRouter.js => 'test/:param': 'testRoute'

## Section 11:

* .../[WebGoat/start.mvc#](http://34.68.19.121:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9)test/<script>**webgoat.customjs.phoneHome()**<%2fscript>
* Our number was: 193835984

## Section 12:

1. **Are trusted websites immune to XSS attacks?**

* Solution 4: No because the browser trusts the website if it is acknowledged and trusted, then the browser does not know that the script is malicious.

2. **When do XSS attacks occur?**

* Solution 3: The data is included in dynamic content that is sent to a web user without being validated for malicious content.

3. **What are Stored XSS attacks?**

* Solution 1: The script is permanently stored on the server and the victim gets the malicious script when requesting information from the server.

**4. What are Reflected XSS attacks?**

* Solution 2: They reflect the injected script off the web server. That occurs when input sent to the web server is part of the request.

**5. Is JavaScript the only way to perform XSS attacks?**

* Solution 4: No there are many other ways. Like HTML, Flash or any other type of code that the browser executes.

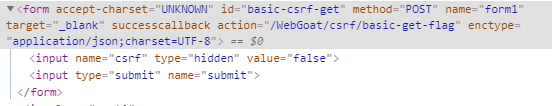
**How can we prevent XSS attacks? Discuss your answer from an end-user point of view, and also a system developer?**

* From a systems developer point of view, there are a few main ways to prevent XSS attacks. The first is “escaping” user input when passing it to end users from your application. This means ensuring that the data a user inputs is secure and disallowing characters that could be malicious before passing it to other users. For example, ensuring any user uploaded data going out does not contain: ‘<’, ‘>’ or ‘/’, and other key characters that could be used to pass malicious code to the end user. Essentially it is good policy to check any data contributed from a user that is now being passed to another user.
* Second is to validate and secure the incoming data that users send. This can be done by whitelisting characters that can be sent to the application / database. This ensures that the characters and strings that are sent from things like forms, requests etc. only contain the valid types. For example ensure that dates, credit cards only accept numeric values, expected lengths for strings, limits on ranges etc.
* From an End User point of view, being aware of clicking or opening unfamiliar links or emails, be aware of emotional messages- meaning to be wary of emails that promise prizes for contests that they don’t remember entering or warnings about banking errors or tax cases etc. Users should check link addresses before clicking them. Users can also disable JavaScript on pages they are not familiar with and clear their browser cookies on a regular schedule.

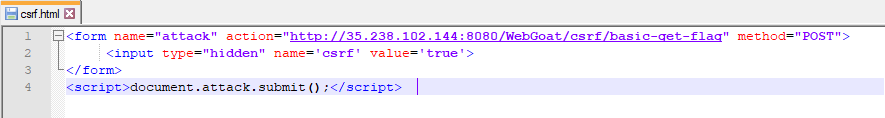
## Cross-Site Request Forgeries

## Section 3

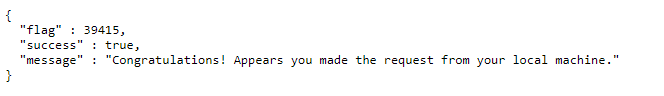
From the hints, we know that there is a hidden form in the HTML body. We can confirm this by inspecting the website



In order to trigger this using CSRF, we can create an external HTML code that when opened will trigger it for us provided that the user is still log in. In a text editor create the following code.

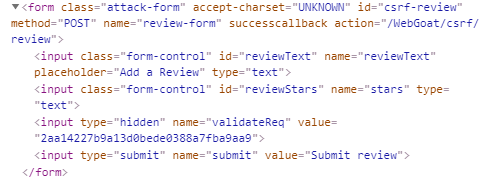


Open the HTML file and we should receive a JSON output similar to below

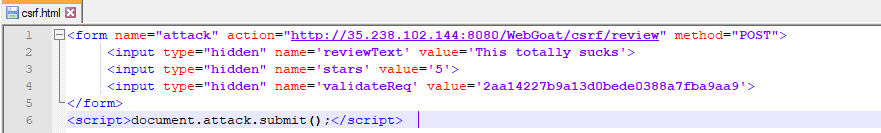


## Section 4

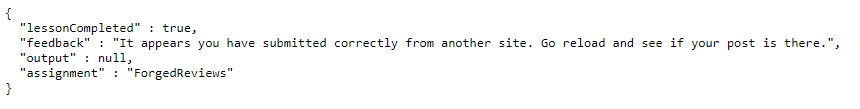
Similar to section 3. We will need to submit the action from an external source. To do this we need to identify the elements of the form.

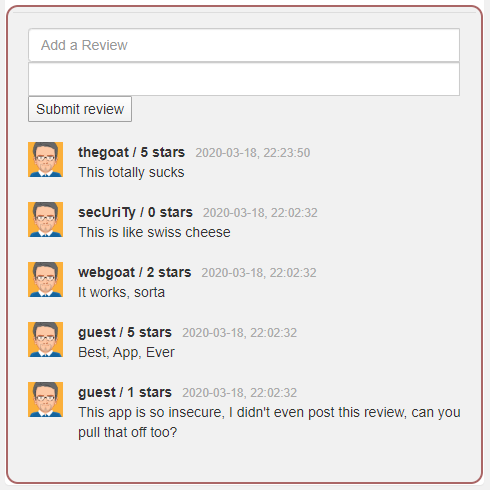


From the HTML snippet above we can create a CSRF attack with the same structure



Opening the HTML page should create results in the following JSON output and create a post in the WebGoat app when it has been refresh.

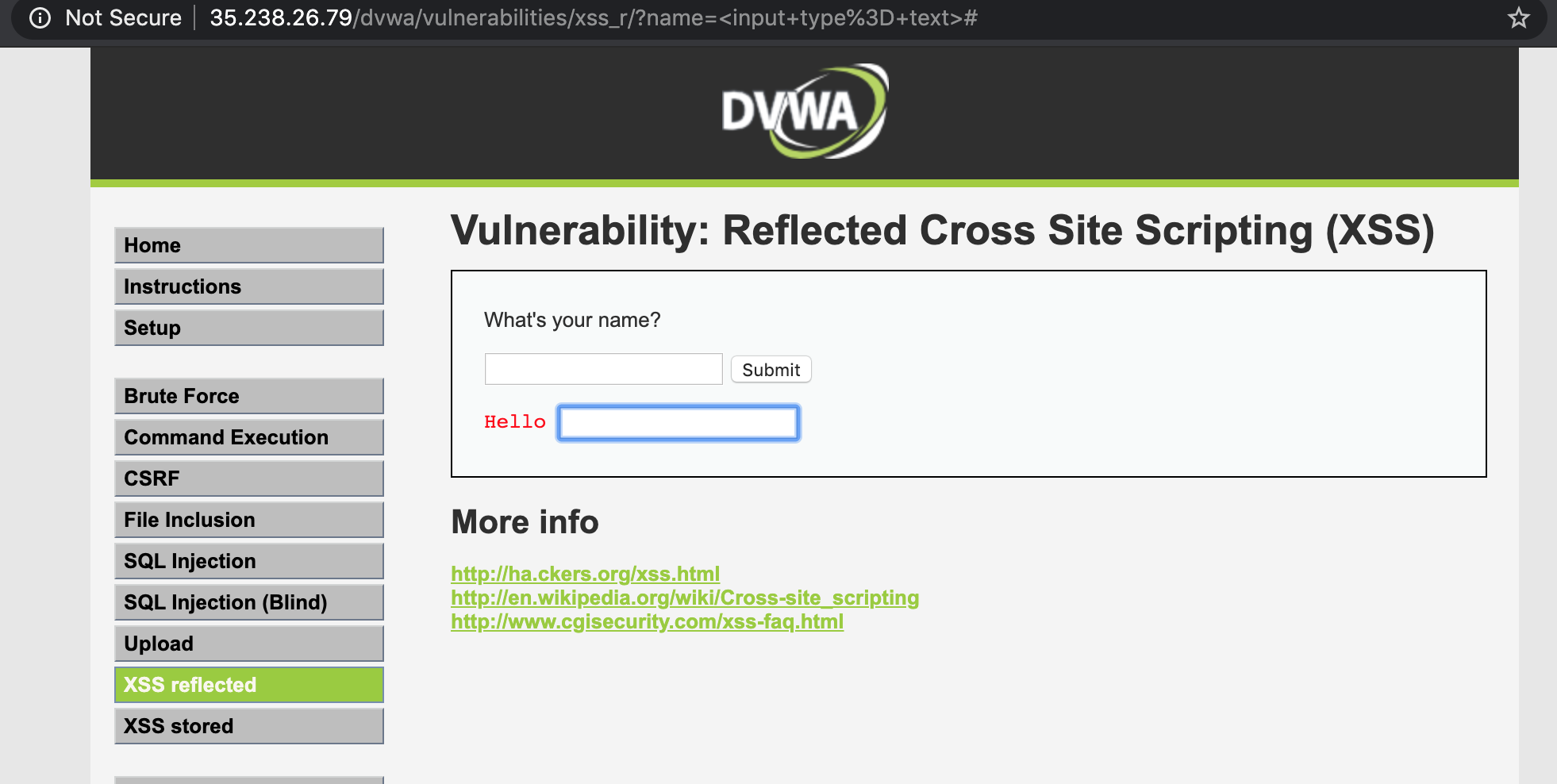




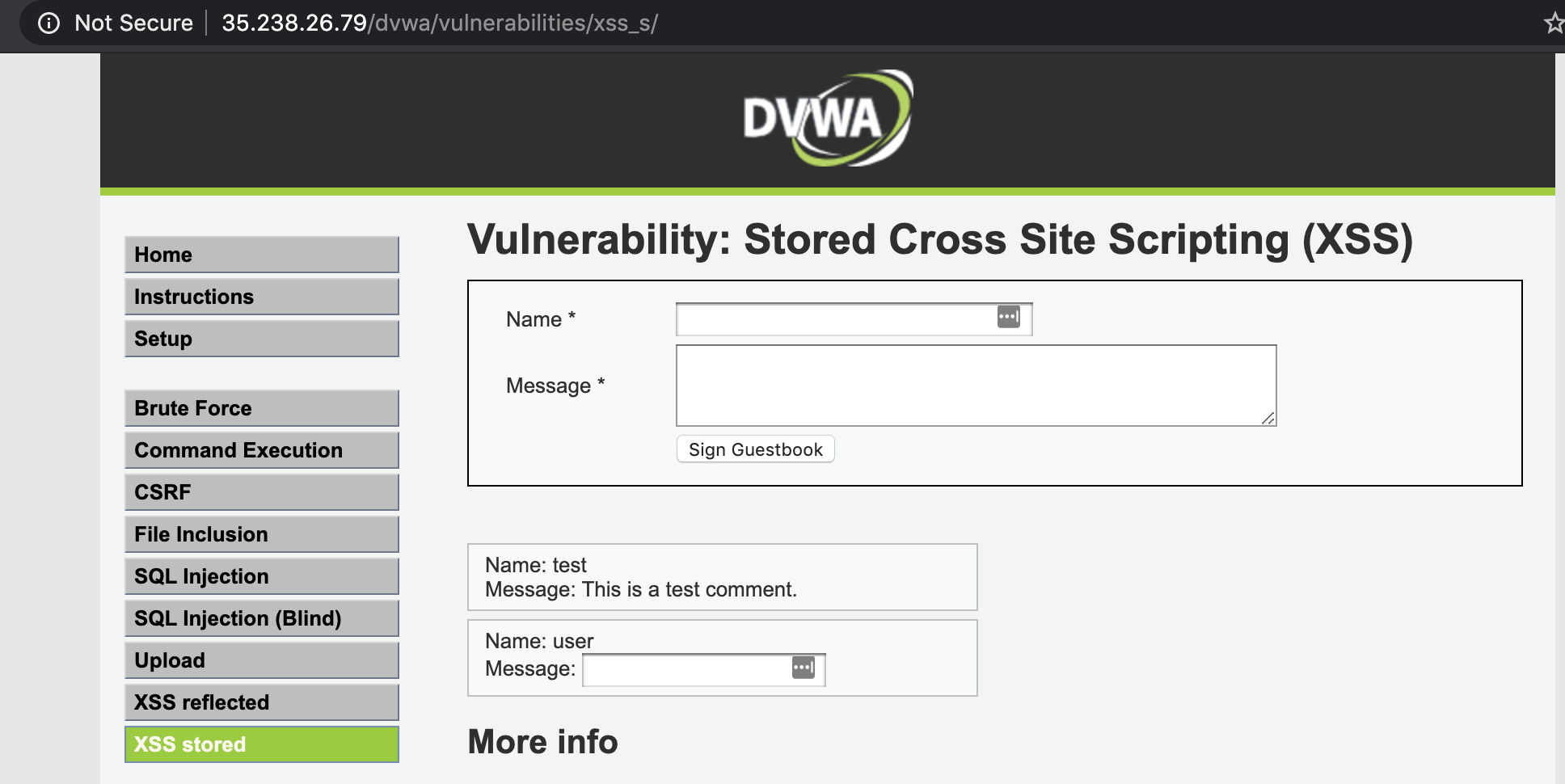
## Complete the following tasks on DVWA

### Cross-site scripting (XSS)

**XSS Reflective**



**XSS Stored**



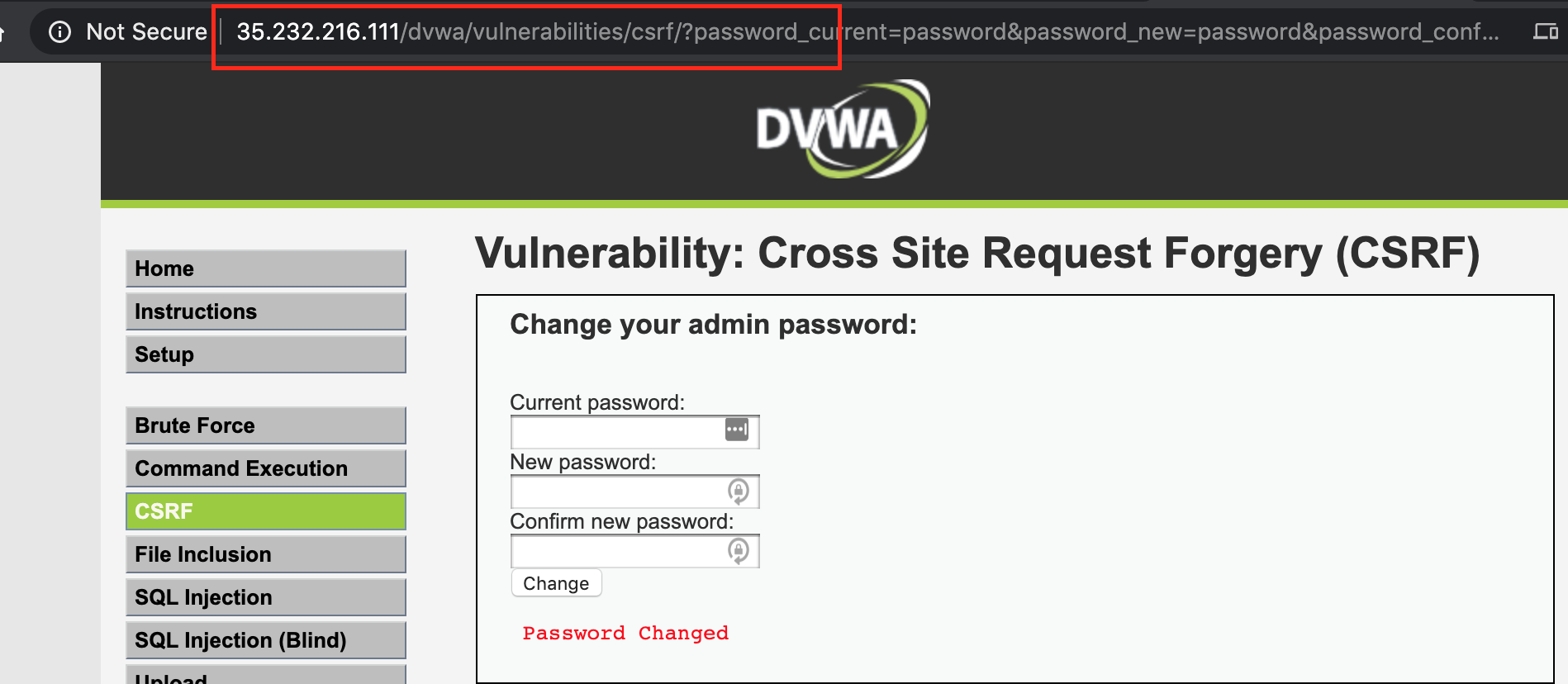
**Question 11: What is the difference between XSS stored and reflected?**

* In XSS reflected attacks, the malicious script (input data sent) is immediately returned to the user application in a response from the server. This means that the attack originates and is sent from the user application to the server and then reflected back to the user from that server. The application or browser will then execute the code because it comes from a “trusted” source. This type of attack affects only the user who enacts it.
* In XSS stored (persistent) attacks, the malicious script is sent to the server and stored there. Then, when the data is requested by a user, it is returned to be executed by that user in their browser. In this way, the stored attack can affect multiple users (anyone who requests the data from the server).

### Request Forgery

**Question 14: Describe how this page (source code with security level low) is vulnerable to CSRF attack. How can an attacker utilize this vulnerability?**

* With the available source code from this page that shows the form details and the Get method details available in the browser, a malicious attacker could create an email or website that would change a users password without them knowing (state change). This would then allow a malicious attacker access to the users account on the site and all the data there in.





**Question 15: How can we prevent CSRF attacks?**

* Instead of cookies, session tokens unique to each session could be adopted. Server side source code should be inspected to ensure any state changing code is protected as best as possible. Users should adopt strong antivirus software, regular clearing and resetting of cookies, use of two-factor authentication and safe browsing mindset (being wary of redirection to strange sites, or opening strange emails).