Exercises for XSS,  OWASP A7

Reflected XSS

Implement your own simple server, or use [this](https://github.com/securitydatspring2019/xssDemoServer.git) (see readme.md for how to set it up), with a proof of concept page, to demonstrate reflected XSS.

If you use the provided server above, feel free to use this [mail](https://docs.google.com/document/d/19Xz6eTfeBxK1js8evmRjzJyGI7Hh25VwxpP-EFkzcCs/edit?usp=sharing) as inspiration for the phishing part.

Remember, we (the hacker) will probably only succeed if the victim is using FireFox or IE.

Stored XSS:

Demonstrate an attack as outlined in [this video](https://www.youtube.com/watch?v=cbmBDiR6WaY) (which you hopefully already have watched). This [snippet file](https://docs.google.com/document/d/1sGegv6K4DeDuTAmXMhepzAaeDfDjIqWNIpj-jLL-kcI/edit) includes setup instructions for the simple “evil hacker server” and also a script you can try to inject.

Feel free to use my [demo-server](https://github.com/Lars-m/xssServer.git) (same as above) for your demo’s.

Alternative ways to Demonstrate XSS

The OWASP Juice Shop Application includes several XSS related challenges. Some can be hard to find but Google for the Juice solutions, and then just do/demonstrate the problems.

## Prevent XSS-Attacks

Getting Started:

Clone this maven project as start code for this exercise: <https://github.com/securitydatspring2019/sanitazion-startcode>

It includes only three things of interest. An index.html page, a Servlet class (Sanitizer.java) and the dependencies referred to below, which you must use to Encode and/or Sanitize user inputs to the servlet.

The dependencies above are for these two OWASP projects, and you will find all you need to know  (which is very little) at these links:

* <https://github.com/OWASP/owasp-java-encoder/wiki/2)-Use-the-OWASP-Java-Encoder>
* <https://github.com/OWASP/java-html-sanitizer>

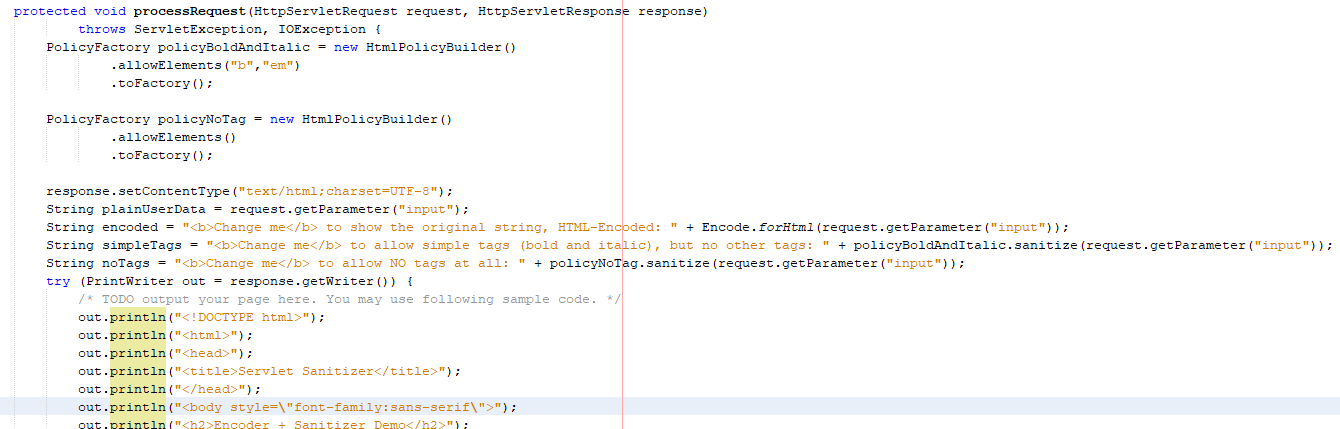
## Use the libraries above to solve the following problems:

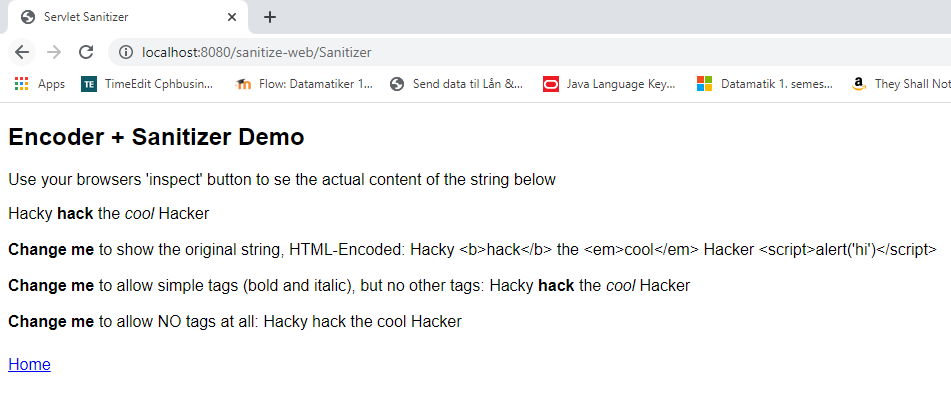
Given this input string which you get from the index-page:

**Hacky <b>hack</b> the <em>cool</em> Hacker <script>alert('hi')</script>**

Change the servlet to return strings as outlined below and in the code:

* Encode the string so, if rendered in a browser, we will see exactly the text (with all characters) given above
  + To encode the string so the browser renders the entire input string, we can use the Basic HTML Context Encode.forHtml(UNTRUSTED) from the OWASP Java Encoder dependency (org.owasp.encoder) on the string input field request.getParameter("input"):  
      
    
* Sanitize the string to allow for simple formatting, but NOTHING else. That is, if the input string was rendered in a browser, we should see this text (bold and italic): Hacky **hack** the *cool* Hacker
  + To sanitize the input string to allow tags for bold, italic and nothing else, we can use a crafted policy from the OWASP Java HTML Sanitizer (com.googlecode.owasp-java-html-sanitizer). Here we can use the PolicyFactory to create a custom policy and use the methods allowElements("b","em") to allow only bold and italic tags. We can then do .sanitize on this policy and put our string input field request.getParameter("input") as parameter:  
      
      
    
* Sanitize the string to allow no tags at all. That is, if the input string was rendered in a browser, we should see this text: Hacky hack the cool Hacker
  + To sanitize the string to allow on tags at all we can use a new crafted policy from the OWASP Java HTML Sanitizer as before, and his time leave out ant tags in the allowElements() method. After this we can do .sanitize on the policy and put our string input field request.getParameter("input") as parameter like before:  
      
      
    

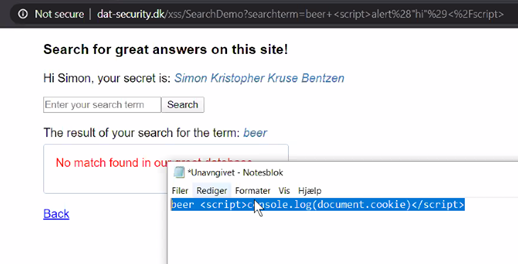
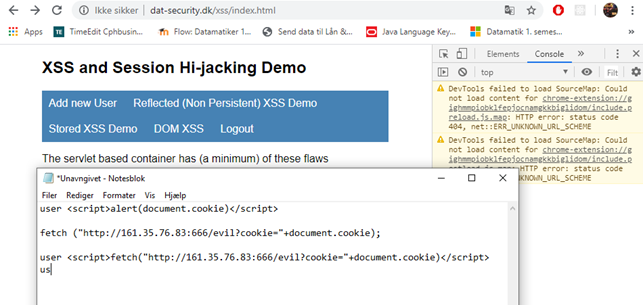
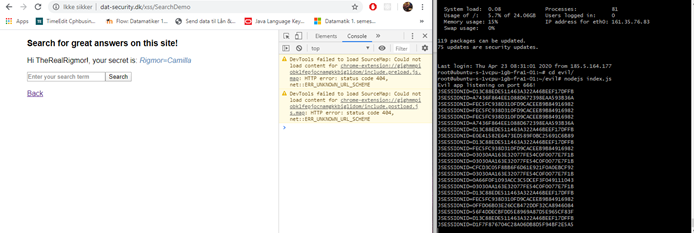




See this deployed project for how it should end: <http://www.dat-security.dk/sanitize/index.html>

### **Hand-in for this lecture**

Write down your own agenda for how you would present the questions given for today, and what you will have to do as exam preparation to set up what is required (for example the XSS vulnerable server) and/or the Encoding/Sanitizing exercise.

* XSS vulnerable server
  + To prepare a setup that demonstrates an XSS vulnerable server, I can use [this](https://github.com/securitydatspring2019/xssDemoServer.git) simple demo server. It is a simple Servlet based server, meant for XSS demoes ONLY.
    - Reflected XSS  
      On this server I can demonstrate how we for example could get hold of the session cookie on an unsecure page by running tags and scripts in the user input fields. Here I will be able to demonstrate why it is important to sanitize one’s data as well as using HTTPOnly.  
      
    - Persisted (stored) XSS  
      I could also demonstrate how unsecure user input that can be stored and exploit other users’ browsers through persistence. This could for example be done by writing dangerous script that sends us the SESSIONID of the current browser, and saving it as our username, so that when another user loads our username, the script will automatically be run in their browser. This way we can get the sessional of multiple users through persistence. This will demonstrate the danger of on sanitizing one’s user input as well as not using HTTPOnly.  
        
        
        
      
* Encoding/Sanitizing
  + Here I could demonstrate what Encoding and Sanitizing is used for and what it can do. This could take emphasis on The OWASP encoding and the OWASP sanitization project dependencies, as well as the provided start code <https://github.com/securitydatspring2019/sanitazion-startcode>.
    - First I could demonstrate how to render the entire input string through encoding, so that one of the tags are run in the browser, with the Basic HTML Context Encode.forHtml(UNTRUSTED) from the OWASP Java Encoder dependency (org.owasp.encoder) on the string input field request.getParameter("input"):  
        
      
    - Then I could demonstrate how to sanitize and only whitelise certain tags. This could be done through a a crafted policy from the OWASP Java HTML Sanitizer (com.googlecode.owasp-java-html-sanitizer). Here we can use the PolicyFactory to create a custom policy and use the methods allowElements("b","em") to allow only bold and italic tags. We can then do .sanitize on this policy and put our string input field request.getParameter("input") as parameter:  
        
        
      
    - Lastly I could demonstrate how to allow NO tags at all through a new crafted policy from the OWASP Java HTML Sanitizer as before, and his time leave out ant tags in the allowElements() method. After this we can do .sanitize on the policy and put our string input field request.getParameter("input") as parameter like before:  
        
        
      

**Last part, how to prevent, is more import than demonstrating the attack.**