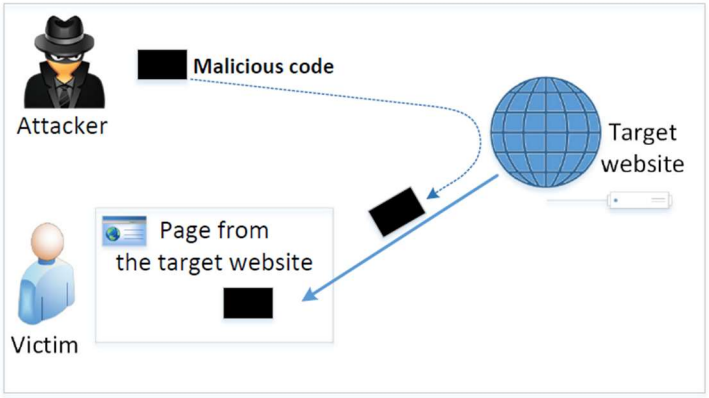
Cross-Site Scripting Attack (CS 915) Post-Lab Assignment Report

Hanzhi Zhang - 5525549

1. Explain how a cross-site scripting attack work with a diagram, the countermeasures and why these countermeasures may address the attack.

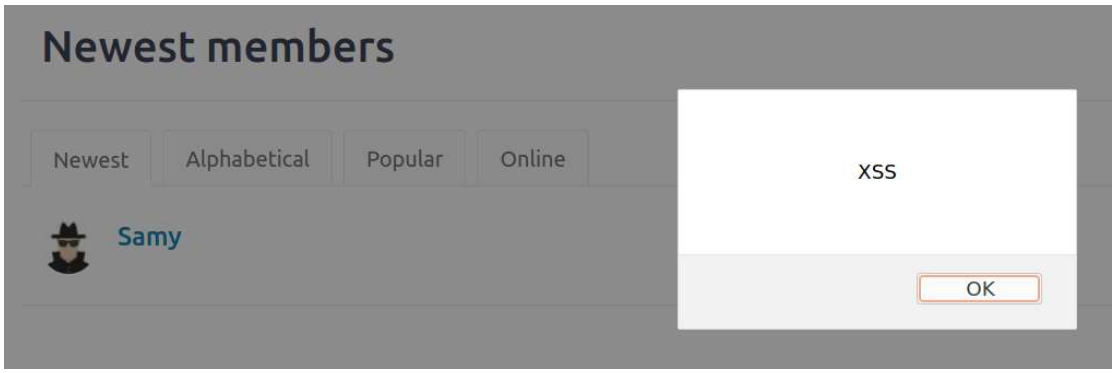


In cross-site scripting attack, attacker injects the malicious code to the victim's browser through a target website. The code is trusted as it becomes part of the website, attacker may then access and change content, read cookies or send requests on behalf of the user through the malicious code.

Countermeasures include encoding (convert everything user provides to html text data before sending them to browsers). This is achieved through replacing HTML markups with alternate representations, so that the embedded code in user data will be interpreted as data and displayed by browsers rather than executed by them.

Another approach is using the Content Security Policy to force the separation between data and code. With the policy we either disallow all inline code and only allow the link approach, or decide whether the code is allowed based on its origin, using nonce. This is to say, only allow code from specified known/trusted sources, so any code that comes from the user (inserted by an attacker) will not be accepted or executed.

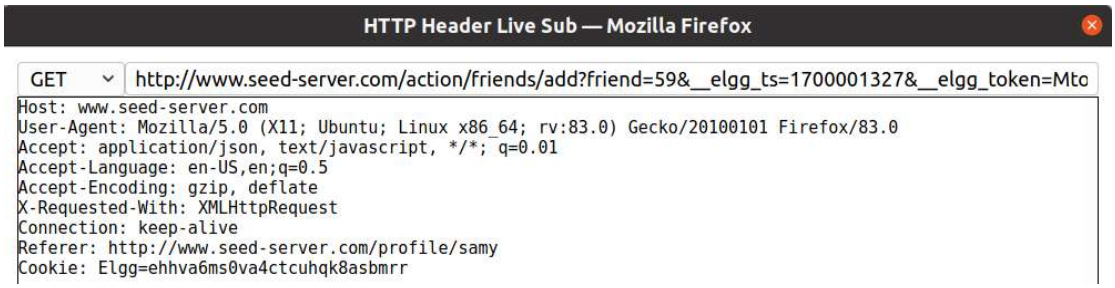
2. Task 1: 1) Explain the result of this task with screenshots.



The alert message pops up when we see Samy's profile.

2) Explain why this attack works.
The JavaScript program embedded in Samy Elgg profile is executed.

3. Task 2: 1) Explain the result of this attack with screenshots.



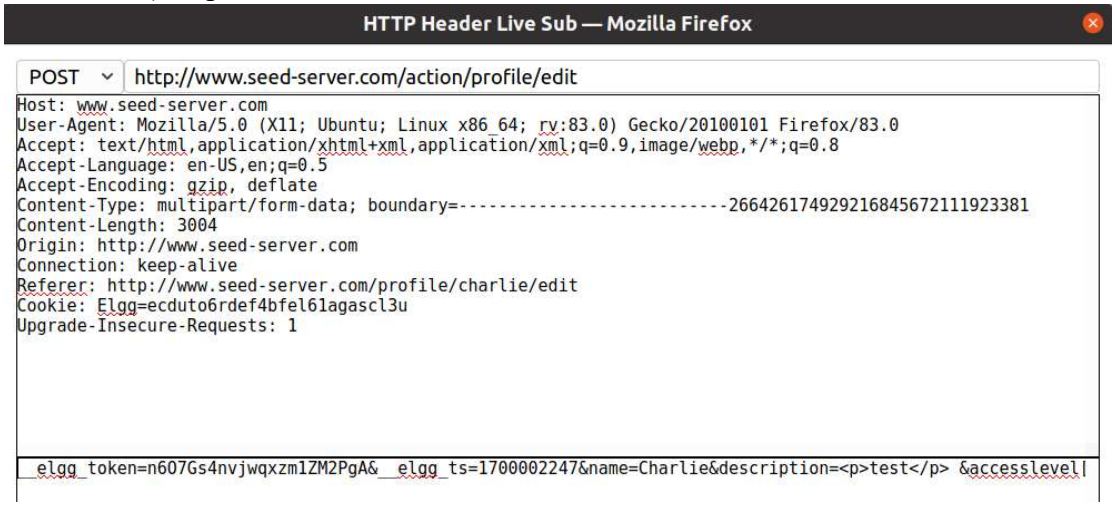
We construct the URL of Elgg's add-friend request with Samy's user ID, 59: var sendurl = "http://www.seed-server.com/action/friends/add" + "?friend=59" + token + ts;

We can see that Samy has been added to Alice's friends list.

2) Answer Question 1 and 2 in the Lab Instruction.
Q1: the "ts" and "token" here are Elgg's CSRF countermeasures parameters, time stamp and secret token, which we get from JavaScript variables inside the page and attach to the URL so that it will be accepted by the server as a same-site request.

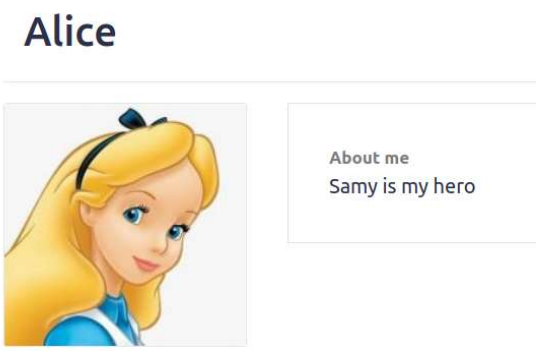
Q2: With the rich text mode, the JavaScript code we enter in the "About Me" field will be encoded and treated as text data, all HTML markups will be replaced, code will no longer be executable. In this case we cannot use "About Me" to launch the attack, but we may still use other fields, for example "Brief Description" for inserting code.

4. Task 3: 1) Explain the result of this task with screenshots.



We already know Samy's Guid is 59 from the previous task. Here we can see the url for modifying one's profile is "http://www.seed-server.com/action/profile/edit", and format of the content is token + ts + name + description(+ accesslevel) + guid, so we have:

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



We have modified Alice's profile to say "Samy is my hero".

2) Answer Question 3 in the Lab Instruction.
Q3: We need line (1) so that our (attacker Samy's) own profile is not modified whenever we view it, otherwise we will have "Samy is my hero" overwriting the JavaScript code in Samy's "About Me" field, and the attack will cease to work since then.

We do not need such check in the add friend attack because according to the design of such social networking sites, a user simply cannot "befriend" oneself. If a friend request from Samy to Samy is sent, nothing will happen, (and even if Samy somehow becomes a friend of himself , this does not affect our attack so we don't care about it).

5. Task 4: 1) Explain the result of this task with screenshots.

After visiting Alice's page, Bob's profile says "Samy is my hero".

2) Explain why this attack works.
DOM API is used to retrieve a copy of the code and attached to the description (inserted right after "Samy is my hero", so that a victim's "About Me" field will be modified to display the visible line "Samy is my hero" as well as to contain the invisible code, and viewing any victim's profile will have the same effect as visiting Samy's profile – code gets executed and we have a new victim. This is how the "propagation" works.

6. Task 5: 1) Explain the result of this task with screenshots.

```
# Purpose: Setting CSP policies in Apache configuration
<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>
```

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

Areas 5 and 6 display OK

```
$cspheader = "Content-Security-Policy:".
             "default-src 'self';".
             "script-src 'self' 'nonce-111-111-111' 'nonce-222-222-222' *.example60.com *.example70.com".
             "";
header($cspheader);
```

CSP Experiment

1. Inline: Nonce (111-111-111): OK
2. Inline: Nonce (222-222-222): OK
3. Inline: No Nonce: Failed
4. From self: OK
5. From www.example60.com: OK
6. From www.example70.com: OK

Areas 1, 2, 4, 5 and 6 display OK

2) Answer Questions 4, 5 and 6 in the Lab Instruction.

Q4: example32a does not have any CSP polices set, thus all areas from 1-7 display OK and button click is responded with the alert. example32b has CSP policies set in Apache configuration, allowing only code from self and example 70, inline code and code from other source are not allowed, so areas 4 and 6 display OK, others all display Failed, and alert does not show upon button click. example32c has CSP set in the php file, allowing only code from self, example70 plus the inline code with nonce 111-111-111, thus only areas 1, 4 and 6 display OK, others display Failed and button click has no response.

Q5: Areas 5 and 6 display OK if we set the Content Security Policy to allow Javascript code script from own site, example60 and example70 (trusted third parties).

Q6: Areas 1, 2, 4, 5 and 6 all display OK if we set the CSP to allow code from own site, inline code with nonce 111-111-111 and 222-222-222, example60 and example70.