Cross-Site Request Forgery (CSRF) Attack Lab

Due: April 23 2020, 11:00 PM (EST)

1 Overview

The objective of this lab is to help students understand the Cross-Site Request Forgery (CSRF or XSRF) attack. A CSRF attack involves a victim user, a trusted site, and a malicious site. The victim user holds an active session with a trusted site while visiting a malicious site. The malicious site injects an HTTP request for the trusted site into the victim user session, causing damages.

In this lab, students will be attacking a social networking web application using the CSRF attack. The open-source social networking application called `Elgg` has countermeasures against CSRF, but we have turned them off for the purpose of this lab.

1.1 Submission

You need to submit a detailed lab report 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.
Late submissions are accepted with 50% grading penalty within 24 hours of the due. Submissions that are late for more than 24 hours will NOT be accepted.
Please submit your solution in PDF or image on `https://www.gradescope.com/courses/88159`, using Entry Code: **9J2VYB**. And kindly choose the right page for your answer to every question.

Collaboration Policy
You can optionally form study groups consisting of up to 3 person per group (including yourself).
Everybody should write individually by themselves, and submit the lab reports separately. DO NOT copy each other's lab reports, or show your lab reports to anyone other than the course staff, or read other students' lab reports.

1.2 Environment

Please download the customized version of the Seed Labs from Google Drive. **This is the same VM we used for the Set-UID lab**
Use this link for steps to setup the VM.

- User ID: seed
- Password: dees
- Root User ID: root
- Password: seedubuntu

2 Lab Environment

2.1 Environment Configuration

In this lab, we need three things, which are already installed in the provided VM image: (1) the Firefox web browser, (2) the Apache web server, and (3) the `Elgg` web application. For the browser, we need to use the `HTTP Header Live` extension for Firefox to inspect the HTTP requests and responses. The pre-built Ubuntu VM image provided to you has already installed the Firefox web browser with the required extensions.

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
Samy	samy	seedsamy

Configuring DNS. We have configured the following URLs needed for this lab.

URL	Description	Directory
`http://www.csrflabattacker.com`	Attacker web site	/var/www/CSRF/Attacker/
`http://www.csrflabelgg.com`	Elgg web site	/var/www/CSRF/Elgg/

3 Background of CSRF Attacks

A CSRF attack involves three actors: a trusted site (`Elgg`), a victim user of the trusted site, and a malicious site. The victim user simultaneously visits the malicious site while holding an active session with the trusted site. The attack involves the following sequence of steps:

1. The victim user logs into the trusted site using his/her username and password, and thus creates a new session.

2. The trusted site stores the session identifier for the session in a cookie in the victim user's web browser.

3. The victim user visits a malicious site.

4. The malicious site's web page sends a request to the trusted site from the victim user's browser. This request is a cross-site request, because the site from where the request is initiated is different from the site where the request goes to.

5. By design, web browsers automatically attach the session cookie to to the request, even if it is a cross-site request.

6. The trusted site, if vulnerable to CSRF, may process the malicious request forged by the attacker web site, because it does not know whether the request is a forged cross-site request or a legitimate one.

The malicious site can forge both HTTP GET and POST requests for the trusted site. Some HTML tags such as `img`, `iframe`, `frame`, and `form` have no restrictions on the URL that can be used in their attribute. HTML `img`, `iframe`, and `frame` can be used for forging GET requests. The HTML `form` tag can be used for forging POST requests. Forging GET requests is relatively easier, as it does not even need the help of JavaScript; forging POST requests does need JavaScript.

4 Lab Tasks

For the lab tasks, you will use two web sites that are locally setup in the virtual machine. The first web site is the vulnerable `Elgg` site accessible at `www.csrflabelgg.com` inside the virtual machine. The second web site is the attacker's malicious web site that is used for attacking `Elgg`. This web site is accessible via `www.csrflabattacker.com` inside the virtual machine. You can download the codes used in this lab here here.

Question 1 (5 points) *Please provide the names and NetIDs of your collaborator (up to 2). If you finished the lab alone, write None.*

4.1 Task 1: CSRF Attack using GET Request

In this task, we need two people in the `Elgg` social network: Alice and Boby. Boby wants to become a friend to Alice, but Alice refuses to add Boby to her `Elgg` friend list. Boby decides to use the CSRF attack to achieve his goal. He sends Alice an URL (via an email or a posting in `Elgg`); Alice, curious about it, clicks on the URL, which leads her to Boby's web site: `www.csrflabattacker.com`. Pretend that you are Boby, describe how you can construct the content of the web page, so as soon as Alice visits the web page, Boby is added to the friend list of Alice (assuming Alice has an active session with `Elgg`).

To add a friend to the victim, we need to identify the Add Friend HTTP request, which is a GET request. In this task, you are not allowed to write JavaScript code to launch the CSRF attack. Your job is to make the attack successful as soon as Alice visits the web page, without even making any click on the page (hint: you can use the `img` tag, which automatically triggers an HTTP GET request).

`Elgg` has implemented a countermeasure to defend against CSRF attacks. In Add-Friend HTTP re-quests, you may notice that each request includes two wired-looking parameters, elgg ts and elgg token. These parameters are used by the countermeasure, so if they do not contain correct values, the request will not be accepted by `Elgg`. We have disabled the countermeasure for this lab, so there is no need to include these two parameters in the forged requests.

Question 2 (25 points) *Please implement Task 1 and answer the following questions:*

Q 2.1 (5 points) *What are the URL and parameters of the GET request for sending friend requests?.*

Q 2.2 (10 points) *Construct a web page such that it automatically adds Boby as a friend when Alice visits it, without even making any click on the page. Please attach the source code for this web page.*

Q 2.3 (10 points) *Please describe your observation with screenshots that show your attack is successful (e.g. using HTTP Live Header).*

4.2 Task 2: CSRF Attack using POST Request

After adding himself to Alice's friend list, Boby wants to do something more. He wants Alice to say "Boby is my Hero" in her profile, so everybody knows about that. Alice does not like Boby, let alone putting that statement in her profile. Boby plans to use a CSRF attack to achieve that goal. That is the purpose of this task.

One way to do the attack is to post a message to Boby's `Elgg` account, hoping that Boby will click the URL inside the message. This URL will lead Boby to your malicious web site `www.csrflabattacker.com`, where you can launch the CSRF attack.

The objective of your attack is to modify the victim's profile. In particular, the attacker needs to forge a request to modify the profile information of the victim user of `Elgg`. Allowing users to modify their profiles is a feature of `Elgg`. If users want to modify their profiles, they go to the profile page of `Elgg`, fill out a form, and then submit the form—sending a POST request—to the server-side script `/profile/edit.php`, which processes the request and does the profile modification.

The server-side script `edit.php` accepts both GET and POST requests, so you can use the same trick as that in Task 1 to achieve the attack. However, in this task, you are required to use the POST request. Namely, attackers (you) need to forge an HTTP POST request from the victim's browser, when the victim is visiting their malicious site. Attackers need to know the structure of such a request. You can observe the structure of the request, i.e the parameters of the request, by making some modifications to the profile and monitoring the request using `HTTP Header Live`. You may see something similar to the following (unlike HTTP `GET` requests, which append parameters to the URL strings, the parameters of HTTP `POST` requests are included in the HTTP message body): (from Line 15 to Line 19)

```
 1  http://www.csrflabelgg.com/action/profile/edit
 2
 3  POST /action/profile/edit HTTP/1.1
 4  Host: www.csrflabelgg.com
 5  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101
 6  Firefox/23.0
 7  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
 8  Accept-Language: en-US,en;q=0.5
 9  Accept-Encoding: gzip, deflate
10  Referer: http://www.csrflabelgg.com/profile/elgguser1/edit
11  Cookie: Elgg=p0dci8baqrl4i2ipv2mio3po05
12  Connection: keep-alive
13  Content-Type: application/x-www-form-urlencoded
14  Content-Length: 642
15  __elgg_token=fc98784a9fbd02b68682bbb0e75b428b&__elgg_ts=1403464813
16  &name=elgguser1&description=%3Cp%3Iamelgguser1%3C%2Fp%3E
17  &accesslevel%5Bdescription%5D=2&briefdescription= Iamelgguser1
18  &accesslevel%5Bbriefdescription%5D=2&location=US
19  .....
```

After understanding the structure of the request, you need to be able to generate the request from your attacking web page using JavaScript code.

```
1  <html>
2  <body>
3  <h1>This page forges an HTTP POST request.</h1>
4  <script type="text/javascript">
5  function forge_post()
6  {
7      var fields;
8      // The following are form entries need to be filled out by attackers.
9      // The entries are made hidden, so the victim won’t be able to see
10     them.
11     fields += "<input type=’hidden’ name=’name’ value=’****’>";
12     fields += "<input type=’hidden’ name=’briefdescription’ value=’****’>";
13     fields += "<input type=’hidden’ name=’accesslevel[briefdescription]’
14     value=’2’>";
15
16     fields += "<input type=’hidden’ name=’guid’ value=’****’>";
17     // Create a <form> element.
18     var p = document.createElement("form");
19     // Construct the form
20     p.action = "http://www.example.com";
21     p.innerHTML = fields;
22     p.method = "post";
23     // Append the form to the current page.
24     document.body.appendChild(p);
25     // Submit the form
26     p.submit();
27 }
28
29 // Invoke forge_post() after the page is loaded.
30 window.onload = function() { forge_post();}
31 </script>
32 </body>
33 </html>
```

In Line 14, the value 2 sets the access level of a field to public. This is needed, otherwise, the access level will be set by default to private, so others cannot see this field. It should be noted that when copy-and-pasting the above code from a PDF file, the single quote character in the program may become something else (but still looks like a single quote). That will cause syntax errors. Replace all the single quote symbols with the one typed from your keyboard will fix those errors.

Note: Please check the `single quote` characters in the program. When copying and pasting the JavaScript program, single quotes are encoded into a different symbol. Replace the symbol with the correct single quote.

Question 3 (45 points) *Please implement Task 2 and answer the following questions:*

Q 3.1 (5 points) *The forged HTTP request needs Alice's user id (guid) to work properly. If Boby targets Alice specifically, before the attack, he can find ways to get Alice's user id. Boby does not know Alice's* `Elgg` *password, so he cannot log into Alice's account to get the information. Please describe how Boby can find out Alice's user id.*

Q 3.2 (5 points) *What are the parameters of the POST request for editing profiles?.*

Q 3.3 (10 points) *Construct a web-page such that it automatically changes Alice's profile's brief description to "Boby is my Hero" when Alice visits it, without even making any click on the page. Please attach the source code for this web page.*

Q 3.4 (10 points) *Please describe your observation with screenshots that show your attack is successful (e.g. using HTTP Live Header).*

Q 3.5 (5 points) *How can Boby find Alice's user id (guid) without her credentials? How can Boby find his own user id?*

Q 3.6 (10 points) *If Boby would like to launch the attack to anybody who visits his malicious web page. In this case, he does not know who is visiting the web page beforehand. Can he still launch the CSRF attack to modify the victim's Elgg profile? Please explain.*

4.3 Task 3: Implementing a countermeasure for `Elgg`

`Elgg` does have a built-in countermeasures to defend against the CSRF attack. We have commented out the countermeasures to make the attack work. CSRF is not difficult to defend against, and there are several common approaches:

- *Secret-token approach*: Web applications can embed a secret token in their pages, and all requests coming from these pages will carry this token. Because cross-site requests cannot obtain this token, their forged requests will be easily identified by the server.
- *Referrer header approach*: Web applications can also verify the origin page of the request using the *referrer* header. However, due to privacy concerns, this header information may have already been filtered out at the client side.

The web application `Elgg` uses secret-token approach. It embeds two parameters `__elgg_ts` and `__elgg_token` in the request as a countermeasure to CSRF attack. The two parameters are added to the HTTP message body for the POST requests and to the URL string for the HTTP GET requests.

`Elgg` secret-token and timestamp in the body of the request: `Elgg` adds security token and timestamp to all the user actions to be performed. The following HTML code is present in all the forms where user action is required. This code adds two new hidden parameters `__elgg_ts` and `__elgg_token` to the POST request:

```
<input type = "hidden" name = "__elgg_ts" value = "" />
<input type = "hidden" name = "__elgg_token" value = "" />
```

The `__elgg_ts` and `__elgg_token` are generated by the `/var/www/CSRF/Elgg/vendor /elgg/elgg/views/default/input securitytoken.php` module and added to the web page. The code snippet below shows how it is dynamically added to the web page.

```
$ts = time();
$token = generate_action_token($ts);

echo elgg_view('input/hidden', array('name' => '__elgg_token', 'value' => $token));
echo elgg_view('input/hidden', array('name' => '__elgg_ts', 'value' => $ts));
```

`Elgg` also adds the security tokens and timestamp to the JavaScript which can be accessed by

```
elgg.security.token.__elgg_ts;
elgg.security.token.__elgg_token;
```

`Elgg` security token is a hash value (md5 message digest) of the site secret value (retrieved from database), timestamp, user sessionID and random generated session string. There by defending against the CSRF attack. The code below shows the secret token generation in `Elgg`.

```
function generate_action_token($timestamp)
{
    $site_secret = get_site_secret();
    $session_id = session_id();
    // Session token
    $st = $_SESSION['__elgg_session'];

    if (($site_secret) && ($session_id))
    {
        return md5($site_secret . $timestamp . $session_id . $st);
    }

    return FALSE;
}
```

The PHP function `session_id()` is used to get or set the session id for the current session. The below code snippet shows random generated string for a given session `__elgg_session` apart from public user Session ID.

```
........
// Generate a simple token (private from potentially public session id)
if (!isset($_SESSION['__elgg_session'])) {
$_SESSION['__elgg_session'] = ElggCrypto::getRandomString(32,ElggCrypto::CHARS_HEX);
........
```

`Elgg` **secret-token validation:** The elgg web application validates the generated token and timestamp to defend against the CSRF attack. Every user action calls `validate_action_token` function and this function validates the tokens. If tokens are not present or invalid, the action will be denied and the user will be redirected.

The below code snippet shows the function `validate_action_token`.

```
function validate_action_token($visibleerrors = TRUE, $token = NULL,
$ts = NULL)
{

  if (!$token) {         $token = get_input('__elgg_token');      }
  if (!$ts) {$ts = get_input('__elgg_ts');       }
  $session_id = session_id();
  if (($token) && ($ts) && ($session_id)) {
    // generate token, check with input and forward if invalid
    $required_token = generate_action_token($ts);

    // Validate token
    if ($token == $required_token) {

      if (_elgg_validate_token_timestamp($ts)) {
        // We have already got this far, so unless anything
        // else says something to the contrary we assume we're ok
        $returnval = true;
          ........
        }
      else {
          ........
            register_error(elgg_echo('actiongatekeeper:tokeninvalid'));
          ........
      }
      ........
}
```

Turn on countermeasure: To turn on the countermeasure, please go to the directory /var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg and find the function `gatekeeper()` in the `ActionsService.php` file. In function `gatekeeper()` please comment out the "`return true;`" statement as specified in the code comments.

```
function gatekeeper($action) {
    //SEED:Modified to enable CSRF.
    //Comment the below return true statement to enable countermeasure
    return true;
    ........
}
```

Question 4 (20 points) *Please answer the question on Task 3*

Q 4.1 (10 points) *Please turn on the countermeasures and execute the CSRF attack again. Describe your observations and explain how the countermeasures work in brief.*

Q 4.2 (10 points) *Please explain why the attacker cannot send these secret tokens in the CSRF attack; what prevents them from finding out the secret tokens from the web page??*

References

[1] `Elgg` documentation: http://docs.elgg.org/wiki/Main_Page.

[2] JavaScript String Operations. http://www.hunlock.com/blogs/The_Complete_Javascript_Strings_Reference.

[3] Session Security `Elgg`. http://docs.elgg.org/wiki/Session_security.

[4] Forms + Actions `Elgg` http://learn.elgg.org/en/latest/guides/actions.html.

[5] PHP:Session_id - Manual: http://www.php.net//manual/en/function.session-id.php.

5 Acknowledgements

This document was modified from the SEED lab instruction. The original copyright notice is

This PDF was created on 2020-04-18 19:21:57Z.