Laboratorio de Cross-Site Request Forgery (CSRF)

(Web Application: `Elgg`)

1 Descripción General

El objetivo de este laboratorio, es ayudar a que los estudiantes entiendan el ataque de Cross-Site Request Forgery (CSRF). Un ataque de CSRF involucra a un usuario víctima, un sitio de confianza y un sitio malicioso. La víctima tiene una sesión válida en un sitio de confianza mientras que por otro lado visita un sitio malicioso que se encarga de inyectar un request HTTP malicioso para ese sitio de confianza dentro de la sesión del usuario víctima.

En este laboratorio, los estudiantes usarán una aplicación web de red social para realizar ataques CSRF. La aplicación web se llama `Elgg` y ha sido instalada en nuestra Máquina Virtual. `Elgg` contiene contramedidas para prevenir ataques CSRF, pero las hemos desactivado para poder realizar los ataques. Este laboratorio cubre los siguientes tópicos:

- Ataque de Cross-Site Request Forgery
- Contramedidas para CSRF: Secret token y Same-site cookie
- Requests HTTP GET y POST
- JavaScript and Ajax

Lecturas. Para una cobertura más detallada en ataques CSRF puede consultar:

- Capítulo 10 del libro de SEED, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. See details at `https://www.handsonsecurity.net`.

Entorno de Laboratorio. Este laboratorio ha sido testeado en nuestra imagen pre-compilada de una VM con Ubuntu 20.04, que puede ser descargada del sitio oficial de SEED . Dado que utilizamos contenedores para configurar el entorno de laboratorio, este laboratorio no depende estrictamente de la VM de SEED. Puede hacer este laboratorio utilizando otras máquinas virtuales, máquinas físicas o máquinas virtuales en la nube.

2 Configuración del Entorno de Laboratorio

En este laboratorio, usaremos tres sitios web. El primero será nuestro aplicativo web vulnerable Elgg y su URL es `www.seed-server.com`. El segundo será el sitio malicioso que va a ser usado para el ataque y su URL es `www.attacker32.com`. El tercero será el sitio usado para las tareas de defensa y su URL es `www.example32.com`. Usaremos contenedores para configurar el entorno de laboratorio.

2.1 Setup del Contenedor y sus Comandos

Para empezar a preparar el contenedor, deberá descargarse el archivo `Labsetup.zip` ubicado en el laboratorio correspondiente dentro del sitio web oficial y copiarlo dentro de la Máquina Virtual prevista por SEED. Una vez descargado deberá descomprimirlo y entrar dentro del directorio `Labsetup` donde encontrará el archivo `docker-compose.yml` que servirá para setear el entorno de laboratorio. Para una información más detallada sobre el archivo `Dockerfile` y otros archivos relacionados, puede encontrarla dentro del Manual de Usuario del laboratorio en uso, en el sitio web oficial de SEED.

Si esta es su primera experiencia haciendo el setup del laboratorio usando contenedores es recomendable que lea el manual anteriormente mencionado.

A continuación, se muestran los comandos más usados en Docker y Compose. Debido a que estos comandos serán usados con mucha frecuencia, hemos creados un conjunto de alias para los mismos, ubicados en del archivo `.bashrc` dentro de la Máquina Virtual provista por SEED (Ubuntu 20.04)

```
$ docker-compose build  # Build the container image
$ docker-compose up     # Start the container
$ docker-compose down   # Shut down the container

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

Dado que todos los contenedores estarán corriendo en un segundo plano. Necesitamos correr comandos para interactuar con los mismos, una de las operaciones fundamentales es obtener una shell en el contenedor. Para este propósito usaremos "`docker ps`" para encontrar el ID del contenedor deseado y ingresaremos "`docker exec`" para correr una shell en ese contenedor. Hemos creado un alias para ello dentro del archivo `.bashrc`

```
$ dockps        // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

En caso de problemas configurando el entorno, por favor consulte la sección “Common Problems” en el manual ofrecido por SEED.

2.2 La Aplicación Web Elgg

En este laboratorio usaremos una aplicación web de red social llamada Elgg. Esta aplicación está instalada y configurada en las imágenes de nuestros contenedores. Usaremos dos contenedores, el primero será el

encargado de correr el servidor web (`10.9.0.5`) y el segundo será el encargado de correr el servidor de base de datos MySQL (`10.9.0.6`). Las direcciones IPs para ambos contenedores están hardcodeadas en múltiples lugares de los archivos de configuración del proyecto, por lo tanto se recomienda no cambiarlos en el archivo `docker-compose.yml`

El Contenedor de Elgg. La aplicación web Elgg está hosteada usando un servidor web Apache. La configuración del servidor está dentro del archivo `apache_elgg.conf` dentro del directorio de la imagen Elgg. Este archivo especifica la URL del sitio web y el directorio donde reside el código fuente de la aplicación web.

```
<VirtualHost *:80>
     DocumentRoot /var/www/elgg
     ServerName   www.seed-server.com
     <Directory /var/www/elgg>
          Options FollowSymlinks
          AllowOverride All
          Require all granted
     </Directory>
</VirtualHost>
```

El Contenedor de Ataque. Usaremos otro contenedor para la máquina que será la atacante (`10.9.0.105`) y hosteará un sitio malicioso. La configuración apache para este sitio web se muestra a continuación:

```
<VirtualHost *:80>
     DocumentRoot /var/www/attacker
     ServerName   www.attacker32.com
</VirtualHost>
```

Debido a que necesitamos crear páginas web dentro de este contenedor, hemos montado un directorio en la Máquina Virtual Host (`Labsetup/attacker`) dentro del directorio del contenedor ubicado en `/var/www/attacker` que es el directorio usado por nuestro archivo configuración de Apache. A su vez las páginas web ubicadas dentro del directorio `attacker` en la Máquina Virtual, serán hosteadas en el sitio web del atacante. Hemos usado código genérico para los archivos que se encuentran dentro de este directorio.

Configuración DNS. Accederemos a nuestro sitio Elgg, nuestro sitio malicioso y el sitio de defensa usando sus URLs. Lo primero que hay que hacer es mapear los nombres de dominio del servidor web con sus IPs. Para ello deberá agregar las siguientes entradas en el archivo `/etc/hosts`. Para poder modificar este archivo ud. debe contar con privilegios de root (usando `sudo`):

```
10.9.0.5        www.seed-server.com
10.9.0.5        www.example32.com
10.9.0.105      www.attacker32.com
```

Base de Datos MySQL. Los contenedores suelen ser desechables, esto quiere decir que una vez que son destruidos, toda la información dentro de ellos se pierde por completo. Para este laboratorio queremos que nuestra información quede persistida en la base de datos MySQL, por lo tanto no perderemos nuestro trabajo al apagar nuestro contenedor. Para lograr esto, hemos montado la carpeta `mysql_data` en nuestra Máquina Host (dentro de la carpeta `Labsetup`, esta carpeta será creada después que el contenedor de

MySQL sea creado y este corriendo) ubicada en el directorio `/var/lib/mysql` dentro del contenedor MySQL, en este directorio MySQL guardará todas las bases de datos. Inclusive si el contenedor es destruido la información de la base de datos es conservada. Si Ud. desea resetear la base de datos puede borrar la carpeta, usando el siguiente comando;

```
$ sudo rm -rf mysql_data
```

Cuentas de Usuario. Hemos creado varias cuentas de usuario en el servidor de la aplicación `Elgg` Los usuarios y sus respectivos passwords son detalllados a continuación:

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

3 Tareas del Laboratorio: Ataques

3.1 Tarea 1: Inspeccionando Request HTTP.

Para realizar ataques CSRF, necesitamos capturar Requests HTTP. En este laboratorio necesitaremos crear Requests HTTP. Para observar como luce 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`". Antes de empezar el laboratorio, el estudiante debería estar familiarizado con esta herramienta. Las instrucciones de como utilizarla están descriptas en esta sección (§ 5.1). En su informe de laboratorio, por favor identifique los parámetros usados en esos requests, si es que existen.

3.2 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`. Haga de cuenta que ud. es Samy, describa como puede 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 esto, necesitamos identificar como luce el Request HTTP original de Add-Friend para agregar un amigo. Podemos usar "`HTTP Header Live`" para llevar a cabo la investigación. En esta tarea, no está permitido escribir código JavaScript para hacer el ataque CSRF. Su objetivo es lanzar un ataque exitoso tan pronto Alice visite la página web, sin que Alice haga un sólo click en la página (pista: puede usar el tag `img`, que automáticamente lanza un Request HTTP GET).

Elgg implementa contramedidas para protegerse contra ataques CSRF. En el Request HTTP Add-Friend, puede advertir que cada request incluye dos parámetros extraños llamados `__elgg_ts` y `__elgg_token`. Estos parámetros son usados como contramedida, si ambos no contienen valores válidos, el request no será

validado por Elgg. Para este laboratorio, hemos desactivado esta protección y no hay necesidad de incluirlos en los requests que se van a falsificar.

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

Después de agregar a Alice como amiga, Samy quiere hacer algo más. 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 lo 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.

El server-side script `edit.php` acepta tanto requests HTTP GET como POST, puede usar el mismo truco de la Tarea 1 para hacer este ataque. Sin embargo, en esta tarea ud. debe usar un request POST. Así mismo, ud. como atacante debe falsificar un request HTTP POST desde el navegador de la víctima cuando esta visite el sitio malicioso. Los atacantes necesitan conocer la estructura de dicho request. Puede conocer esta estructura, es decir sus parámetros, haciendo cambios en su perfil y monitoreando el request resultante usando `"HTTP Header Live"`. Lo que se observe puede ser algo similar a lo que se muestra a continuación, a diferencia de los requests HTTP `GET` que ubican sus parámetros en la URL, en los requests HTTP `POST` los parámetros son incluídos en el cuerpo del mensaje (vea el contenido mostrado a continuación ubicado entre los símbolos ☆)

```
http://www.seed-server.com/action/profile/edit

POST /action/profile/edit HTTP/1.1
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) ...
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.seed-server.com/profile/elgguser1/edit
Cookie: Elgg=p0dci8baqrl4i2ipv2mio3po05
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 642
__elgg_token=fc98784a9fbd02b68682bbb0e75b428b&__elgg_ts=1403464813  ☆
&name=elgguser1&description=%3Cp%3Iamelgguser1%3C%2Fp%3E
&accesslevel%5Bdescription%5D=2&briefdescription= Iamelgguser1
&accesslevel%5Bbriefdescription%5D=2&location=US
......                                                              ☆
```

Después de entender la estructura del request, necesita generar un request desde su página web de ataque usando código JavaScript. Para simplificar esta tarea, hemos provisto un código genérico a partir del cual se puede generar el ataque CSRF que se quiere lanzar desde el sitio web malicioso. Este es un código de ejemplo y necesita modificarlo para que funcione en su ataque.

```
<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='****'>";
    fields += "<input type='hidden' name='briefdescription' value='****'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]'
                                    value='2'>";                          ①
    fields += "<input type='hidden' name='guid' value='****'>";

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

    // Construct the form
    p.action = "http://www.example.com";
    p.innerHTML = fields;
    p.method = "post";

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

    // Submit the form
    p.submit();
}


// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
</script>
</body>
</html>
```

En la línea Line ①, el valor `2` establece el nivel de acceso público de un campo en particular. Esto es necesario de otra forma, el nivel de acceso será puesto por defecto como privado lo que hará que otros usuarios no puedan verlo. Cabe aclarar que al hacer copy and paste del código en el archivo PDF, muy a menudo, las comillas dobles y en especial las comillas simples, pueden convertirse en símbolos diferentes que parecen similares. Esta situación causará errores en el código, así que tenga en cuenta este escenario. Cuando esto suceda, elimínelos y escriba manualmente esos símbolos.

Preguntas. Además de describir su ataque de forma detallada, deberá de responder las siguientes preguntas en su informe de laboratorio:

- **Pregunta 1**: El request falsificado necesita el user id de Alice (guid) para funcionar de manera correcta. Si Boby elije a Alice como víctima antes de hacer el ataque, puede encontrar diversas formas para obtener este id. Boby no conoce el password de Alice en Elgg, por lo tanto no puede loguearse

como Alice para obtener esta información. Por favor explique como Boby puede resolver este problema.

- **Pregunta 2:** Si Boby quisiera ejecutar el ataque sobre cualquier persona que visite su página maliciosa, estariamos en un escenario donde Boby no sabe quién está visitando su página de antemano. Dada esta situación ¿Puede lanzar el ataque CSRF y modificar el perfil de la víctima visitante? Por favor describa esta situación.

4 Tareas de Laboratorio: Defensa

No es difícil defenderse contra ataques CSSRF. Inicialmente, la mayoría de las aplicaciones usan un token secreto (secret token) en su páginas y por medio del chequeo de la presencia de este token en un request, pueden verificar si se trata de un request same-site o un request cross-site. Esto es llamado protección de *secret token*. Recientemente la mayoría de los navegadores han implementado un mecanismo llamado *Same-Site cookie*, cuya intención es simplificar la implementación de medidas de defensa en contra de ataques CSRF. Haremos unos experimentos usando ambos métodos.

4.1 Tarea 4: Activando las Contramedidas en Elgg

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 serán 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 podrán ser fácilmente identificados como requests cross-site.

Elgg usa este tipo de contramedida en contra de ataques CSRF. Hemos desactivado esta contramedida para hacer posible nuestros ataques. 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 encargará de validarlos antes de procesar el request.

Embebebiendo el token secret y el timestamp en las páginas. Elgg agrega un token y un timestamp de seguridad en todos sus requests HTTP. El siguiente código HTML está presente en todos los formularios con los cuales interactua el usuario. El token y el timestamp son campos ocultos; cuando el formulario es enviado, estos campos ocultos dentro del HTML son agregados al request.

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

Elgg asigna los valores del token y el timestamp a variables de JavaScript, lo cual permite que se acceda fácilmente por medio de código JavaScript en la página en donde están presentes.

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

El código que se encarga de agregar estos parámetros en las páginas de Elgg está ubicado en el archivo `vendor/elgg/elgg/views/default/input/securitytoken.php`. El código a continuación describe su mecánica.

```
$ts = time();
$token = elgg()->csrf->generateActionToken($ts);
```

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

Generación del token secreto. El token de seguridad que utiliza Elgg es un valor hash (md5) que es la resultante de un valor secreto del sitio (obtenido de la base de datos), de un timestamp, del ID de sesión del usuario y una cadena de sesión generada al azar. El código que se muestra abajo es el encargado de generar este token en Elgg y está dentro del archivo `vendor/elgg/elgg/engine/classes/Elgg/Security/Csrf.php`.

```
/**
 * Generate a token from a session token (specifying the user),
 * the timestamp, and the site key.
 */
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();
}
```

Validación del token secreto. 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 y el ussuario será redirigido. En nuestro setup inicial, hemos agregado un `return` al principio de está función, para desactivar esta validación.

```
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) ...
}
```

Tarea: Active la contramedida. Para activar la contramedida, debe entrar al contenedor de Elgg, ir al directorio `/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security` y eliminar el `return` del archivo `Csrf.php`. Existe un editor de texto de consola llamado `nano` que está disponible dentro del contenedor. Después de hacer este cambio, repita el ataque y vea si es exitoso o no. Por favor anote los tokens secretos de los HTTP requests capturados. Por favor explique porque el atacante no puede enviar estos tokens en el ataque CSRF; ¿Qué es lo que evita que un atacante encuentre los tokens secretos de la página web?

Nota (importante) Debe tener en cuenta que al activar la contramedida y al lanzar un ataque que edite

el perfil de un usuario, esta protección hará que su ataque falle y que la página sea recargada, esto provocará que el request HTTP falsificado sea disparado nuevamente. Esto provocará otro intento fallido de ataque y la página se recargará nuevamente y así. Esto lo hará entrar en un loop infinito que enlentencerá su computadora. Para evitar esto, una vez que compruebe que su ataque falló, cierre el tab del navegador en donde está probando su ataque y de esta forma podrá evitar este loop infinito.

4.2 Tarea 5: Experimentando con la Contramedida SameSite Cookie

La mayoría de los navegadores implementan un mecanismo llamado SameSite cookie, que es una propiedad asociada a las cookies. Al envíar 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.

Par darle una idea a los estudiantes del funcionamiento de las cookies SameSite y como pueden ayudarnos a defendernos en contra de ataques CSRF, hemos creado un sitio web en uno de los contenedores llamado `www.example32.com`. Por favor visite la siguiente URL (el hostname está mapeado dentro del archivo`/etc/hosts` a la ip `10.9.0.5`); si ud. no está usando la Máquina Virtual de SEED, debería de agregar esta entrada en su máquina):

```
URL: http://www.example32.com/
```

Una vez que haya entrado al sitio web, serán seteadas tres cookies en su navegador, `cookie-normal`, `cookie-lax`, y `cookie-strict`. Como el nombre lo indica, la primera cookie es una cookie común y corriente, la segunda y tercer cookie son cookies SameSite pero de diferente tipo (`Lax` y `Strict`). Hemos diseñado 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 este no será el caso si la cookie es de tipo SameSite.

Por favor dirígase a los links de ambos experimentos. El Link A apunta a `example32.com` mientras que el Link B apunta a `attacker32.com`. Ambas páginas son iguales (exepto por el color del fondo), ambas envian tres tipos diferentes de requests a `www.example32.com/showcookies.php`, que mostrará las cookies enviadas por el navegador. Observando los resultados ud. podrá determinar cuales fueron las cookies enviadas por el navegador. A continuación haga lo siguiente:

- Por favor describa lo que observa y explique porque algunas cookies no son enviadas en determinados escenarios.
- Basado en su entendimiento, por favor describa como las cookies SameSite pueden ayudar a detectar si un request es cross-site o same-site.
- Por favor decriba como usaría el mecanismo de SameSite cookie para defender a Elgg contra ataques CSRF. Sólo necesita hacer una descripción general de sus ideas y no hay necesidad de implementarlas.

Puntos Extra. Aunque no es necesario, se alienta a los estudiantes a modificar la aplicación Elgg, para que usen el mecanismo de SameSite Cookie para prevenir ataques CSRF. Recomendamos a los instructores otorgar puntos extra a los usuarios que puedan hacer esto. Los estudiantes deben de consultar con los instructores todo lo que tenga que ver con los puntos extra del bonus.

5 Guías

5.1 Usando "HTTP Header Live" para inspeccionar Headers HTTP

La versión de Firefox 60 de nuestra Máquina Virtual de Ubuntu 16.04 no soporta el plugin `LiveHTTPHeader`, que fue usado en nuestra Máquina Virtual de Ubuntu 12.04. Dada esta situación, se usará "`HTTP Header Live`" como reemplazo. Las instrucciones de como habilitar y usar este plugin se muestran en la figura Figure 1 solamente haga click en el ícono mosotrado en el marcador ①; aparecerá una barra lateral en la izquierda, asegúrese que `HTTP Header Live` este seleccionada en la posición mostrada en el marcador ②. Luego haga click en cualquier link dentro de la página, todas los Requests HTTP serán capturados y mostrados dentro de la barra lateral mostrada en el marcador ③. Si hace click en cualquiera Request HTTP, se abrirá un pop-up que mostrará el Request HTTP seleccionado. Desafortunadamente hay un bug en este plugin (que aún se encuentra en desarrollo); no se mostrará nada dentro de este pop-up al menos que ud. cambie el tamaño del pop-up (Al parecer el evento de re-drawing se ejecuta automáticamente cuando se abre el pop-up, pero cambiando su tamaño ocasiona que este evento sea disparado y en consecuencia se renderize el contenido en pantalla)

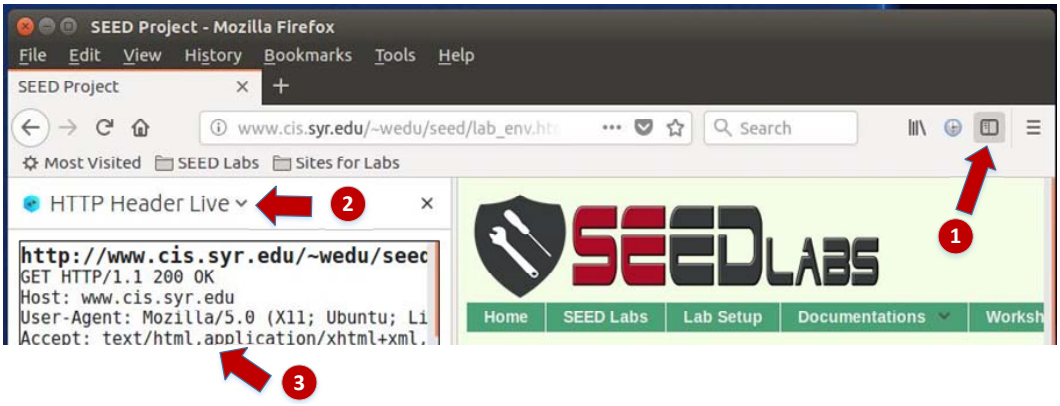


Figure 1: Hablitando el plugin HTTP Header Live

5.2 Usando Web Developer Tool para inspeccionar Headers HTTP

Existe otra herramienta provista por Firefox que puede ser muy útil para inspeccionoar Encabezados HTTP. Esta herramienta es la Web Developer Network Tool. En esta sección, vamos a cubrir algunas de las features más importantes de esta herramienta. La Web Developer Network Tool puede ser habilitada siguiendo estos pasos:

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

Usaremos la página de login de Elgg como ejemplo. La Figure 2 muestra el Request HTTP POST que se envía al momento del login dentro de la Network Tool.

Para más detalles del Request, podemos hacer click en un Request HTTP específico y se abrirán dos paneles con información detallada del mismo. (Ver Figure 3)

Los detalles del Request seleccionado serán mostrados en el panel de la derecha. La Figure 4(a) muestra los detalles del Request de Login en el Tab de `Headers` (Estos detalles incluyen el método del Request, la

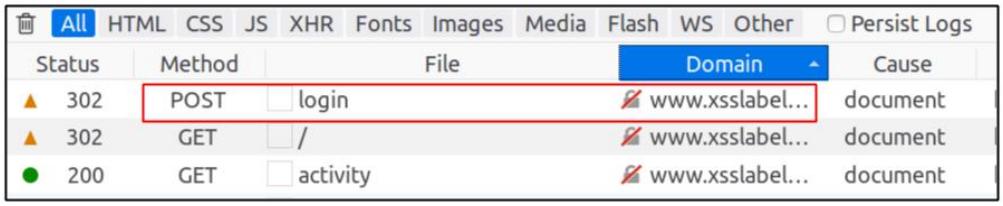


Figure 2: Request HTTP en la Web Developer Network Tool

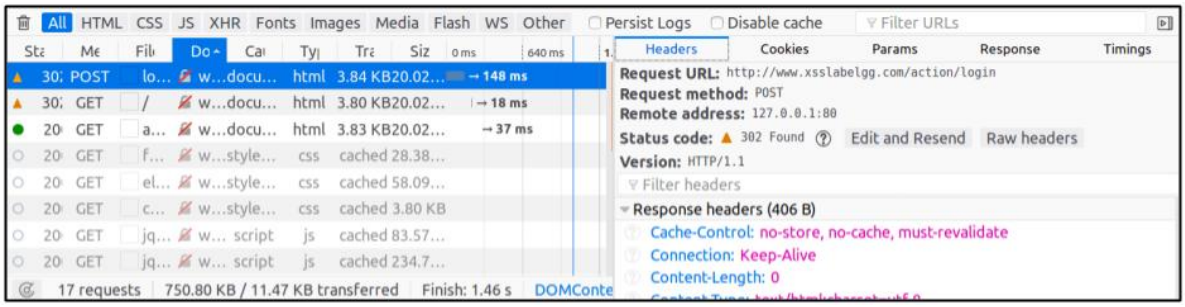


Figure 3: Request HTTP y Detalles del Request

URL y la Cookie). En el panel derecho se pueden observar los Headers de la respuesta como también los del request. Para chequear los parámetros involucrados en un request HTTP, podemos usar el tab `Params`. La Figure 4(b) nos muestra los parámetros enviados en el request del login que se envía a Elgg, estos incluyen el `username` y el `password`. Esta herramienta puede ser usada para inspeccionar tanto Request HTTP GET como POST.

Font Size. El font size usado por defecto en la Web Developer Tool puede ser algo pequeño, para incrementar el tamaño de la fuente se debe hacer click en cualquier lugar de la ventana de la Network Tool y presionar en simultáneo las teclas `Ctrl` y `+`

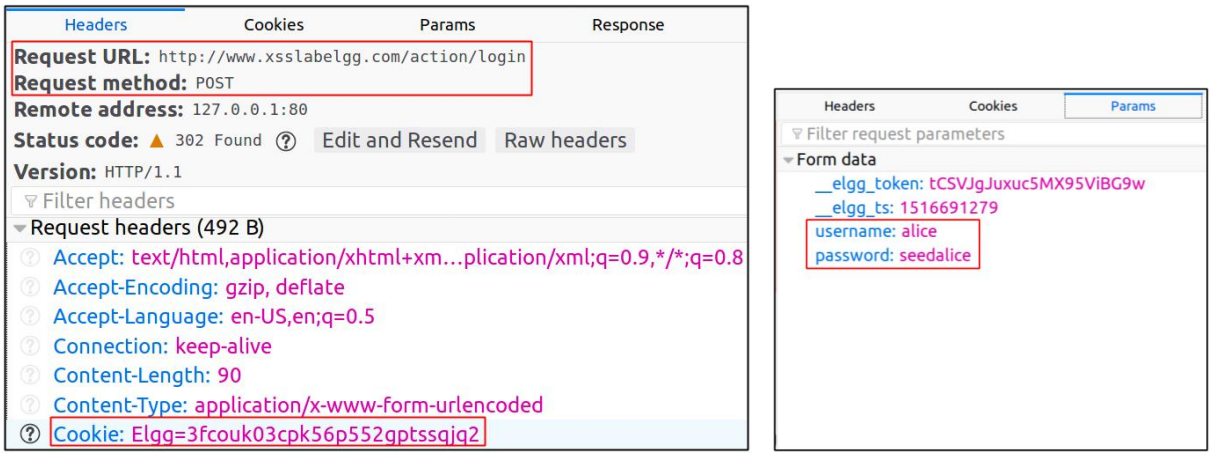
5.3 Debugueando JavaScript

En muchas ocasiones vamos a necesitar debuguear nuestro código JavaScript. La developer tool de Firefox puede ayudarnos en esta tarea. Esta herramienta tiene la posibilidad de indicarnos el punto exacto en el código donde se produjo el error. A continuación se indica como habilitar el debugging en la Web Developer Tool:

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

Una vez situados en la consola, se debe clickear en el tab `JS`. Luego haga click en la flecha que apunta hacia abajo y asegúrese que al costado de `Error` haya una tilde. Si también le interesa activar los mensajes de Warning en la coonsola seleccione `Warnings`. Vea la Figure 5.

Si hay errores en el código, se le mostrará un mensaje de error en la consola. Este mensaje indicará el número des línea que causó este error y estará ubicado en el extremo derecho del mensaje. Para ir al lugar exacto donde el código falló, deberá hacer click en el número de línea que es mostrado en el mensaje de



(a) HTTP Request Headers (b) HTTP Request Parámetros

Figure 4: HTTP Headers y Parámetros

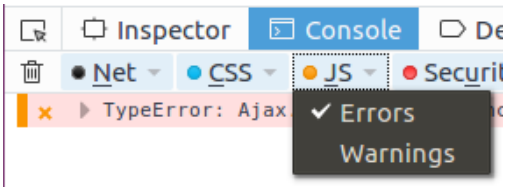


Figure 5: Debugueando Código JavaScript (1)

error. Vea la Figure 6.

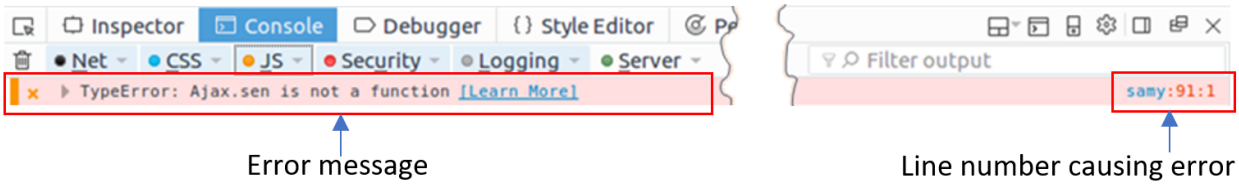


Figure 6: Debugueando Código JavaScript (2)

6 Informe de Laboratorio

Debe enviar un informe de laboratorio detallado, con capturas de pantalla, para describir lo que ha hecho y lo que ha observado. También debe proporcionar una explicación a las observaciones que sean interesantes o sorprendentes. Enumere también los fragmentos de código más importantes seguidos de una explicación. No recibirán créditos aquellos fragmentos de códigos que no sean explicados.

Agradecimientos

Este documento ha sido traducido al Español por Facundo Fontana