跨站请求伪造（CSRF）攻击实验
(Web 应用程序：Elgg)

1 概述

本实验的目的是帮助学生理解跨站请求伪造（CSRF）攻击。CSRF 攻击涉及受害用户、受信任站点和恶意站点。当受害者在访问恶意站点时，恶意站点的网页就有机会向受信任站点发出 HTTP 请求，导致损害。

在这个实验中，学生们将使用 CSRF 攻击来攻击一个社交网络 Web 应用程序 Elgg。我们已经关闭了 Elgg 中的一些防护措施以进行本实验。该实验覆盖了以下主题：

- 跨站请求伪造攻击
- 防御 CSRF 的方法：秘密令牌和同源 cookie
- HTTP GET 和 POST 请求
- JavaScript 和 Ajax

参考资料。 关于 CSRF 攻击的详细内容可以参见《SEED 书籍》第 10 章。

实验环境。 本实验在我们预先构建好的 Ubuntu 20.04 VM（可以从我们的 SEED 网站当中下载）当中测试可行。既然我们使用容器来建立实验环境，本实验不太依赖 SEED VM。您可以在其他 VM、物理机器以及云端 VM 上进行此实验。

2 实验环境搭建

在本实验中，我们将使用三个网站。第一个网站是一个社交网站 www.seed-server.com。第二个网站是攻击者的恶意站点，此网站可通过 www.attacker32.com 访问。第三个网站用于防御任务，其主机名为 www.example32.com。我们使用容器来搭建实验环境。

2.1 容器设置与命令

请从实验的网站下载 Labsetup.zip 文件到你的 VM 中，解压它，进入 Labsetup 文件夹，然后用 docker-compose.yml 文件安装实验环境。对这个文件及其包含的所有 Dockerfile 文件中的内容的详细解释都可以在链接到本实验网站的用户手册 [1] 中找到。如果这是您第一次使用容器设置 SEED 实验环境，那么阅读用户手册非常重要。

[1] 如果你在部署容器的过程中发现从官方源下载容器镜像非常慢，可以参考手册中的说明使用当地的镜像服务器

在下面，我们列出了一些与 Docker 和 Compose 相关的常用命令。由于我们将非常频繁地使用这些命令，因此我们在 .bashrc 文件（在我们提供的 SEED Ubuntu 20.04 虚拟机中）中为它们创建了别名。

```
$ docker-compose build  # 建立容器镜像
$ docker-compose up     # 启动容器
$ docker-compose down   # 关闭容器

// 上述 Compose 命令的别名
$ dcbuild       # docker-compose build 的别名
$ dcup          # docker-compose up 的别名
$ dcdown        # docker-compose down 的别名
```

所有容器都在后台运行。要在容器上运行命令，我们通常需要获得容器里的 Shell 。首先需要使用 docker ps 命令找出容器的 ID ，然后使用 docker exec 在该容器上启动 Shell 。我们已经在 .bashrc 文件中为这两个命令创建了别名。

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

// 下面的例子展示了如何在主机 C 内部得到 Shell
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// 注: 如果一条 docker 命令需要容器 ID, 你不需要
//     输入整个 ID 字符串。只要它们在所有容器当中
//     是独一无二的, 那只输入前几个字符就足够了。
```

如果你在设置实验环境时遇到问题，可以尝试从手册的 “Miscellaneous Problems” 部分中寻找解决方案。

2.2 Elgg Web 应用程序

在本实验中，我们将使用一个开源的 Elgg 社交网络应用。它已经安装到了我们的虚拟机中。

我们在本实验中使用名为 Elgg 的开源 Web 应用程序。Elgg 是一款基于 Web 的社交网络应用程序。它已在提供的容器映像中设置完毕。我们使用两个容器，一个运行 Web 服务器 (10.9.0.5)，另一个运行 MySQL 数据库 (10.9.0.6)。这两个容器的 IP 地址在配置中已经固定，因此不要在 docker-compose.yml 文件中更改它们。

2.3 Elgg Web 应用程序

Elgg 容器。 我们使用 Apache Web 服务器托管 Elgg Web 应用程序。网站设置包含在 Elgg 镜像文件夹内的 apache_elgg.conf 中。配置指定网站的 URL 和存储 Web 应用程序代码的文件夹。

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

攻击者容器。 我们使用另一个容器(10.9.0.105)来运行攻击者的网站,该网站包含一个恶意的 HTML 页面。此页面的 Apache 配置如下:

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

由于我们需要在容器内创建网页文件，为了方便，我们将主机上的 Labsetup/attacker 文件夹挂载到容器的 /var/www/attacker 文件夹中。因此，在 VM 中放置于 attacker 文件夹内的网页将被攻击者的网站托管，我们已经在该文件夹内放置了代码的框架。

DNS 配置。 为了通过不同的 URL 访问 Elgg 网站、攻击者站点和防御站点,我们需要在 /etc/hosts 文件中添加以下条目，以便将这些主机名映射到其相应的 IP 地址。您需要使用 root 权限来更改此文件（使用 sudo)。需要注意的是，由于其他一些实验的原因，这些名称可能已经添加到文件中。如果它们映射到不同的 IP 地址，旧条目必须删除。

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

MySQL 数据库。 容器通常是一次性的，因此一旦销毁，容器内的所有数据都会丢失。对于本实验，我们确实希望将数据保留在 MySQL 数据库中，这样在关闭容器时就不会丢失数据。为了实现这一点，我们将主机上的 mysql_data 文件夹挂载到 MySQL 容器内的 /var/lib/mysql 文件夹（此文件夹是 MySQL 存储数据库的地方)。因此，即使容器被销毁，数据库中的数据仍会保留。这个 mysql_data 文件夹在 Labsetup 内，它将在 MySQL 容器第一次运行后创建。如果您确实想从空的数据库开始，可以删除此文件夹:

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

用户账户。 我们在 Elgg 服务器上创建了几个用户账户，用户名和密码如下所示：

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

3 实验任务: 攻击

3.1 任务 1：观察 HTTP 请求

在跨站请求伪造攻击中，我们需要伪造 HTTP 请求。因此，我们需要了解一个合法的 HTTP 请求长什么样以及它使用了哪些参数等信息。我们可以通过 Firefox 的扩展程序 "HTTP Header Live" 来实现这一目标。本任务的目标是熟悉此工具。有关如何使用此工具的具体说明请参见指导部分 (§ 5.1)。请使用此工具捕获 Elgg 的 HTTP GET 和 POST 请求。在您的报告中，请识别这些请求中使用的参数。

3.2 任务 2：使用 GET 请求的 CSRF 攻击

在这个任务中，我们需要使用 Elgg 社交网络中有两个用户：Alice 和 Samy。Samy 想要成为 Alice 的好友，但 Alice 并不想将他加到自己的好友列表中。Samy 决定使用 CSRF 攻击来实现这个目标。他会通过邮件或在 Elgg 里给 Alice 发一个 URL。当 Alice 好奇点击该 URL 后，她会被引导至 Samy 的网站：www.attacker32.com。请假装你是 Samy，描述你如何构建网页内容，以便当 Alice 访问此页面后，Samy 就会成为 Alice 好友列表中的成员（假设 Alice 当前的 Elgg 会话还是有效的）。

为了给受害者添加好友，我们需要知道合法的 “添加好友” 的 HTTP 请求（GET 请求）是什么样的。我们可以使用 "HTTP Header Live" 工具进行调查。在这个任务中，不允许编写 JavaScript 代码来发动 CSRF 攻击。您的工作是当 Alice 访问页面后攻击会立即发动，Alice 无需在页面上点击任何内容（提示：您可以使用 img 标签，它会自动触发 HTTP GET 请求）。

Elgg 实现了一种防御 CSRF 攻击的措施。在 “添加好友” 的 HTTP 请求中，您可能会注意到每个请求都包含了两个看起来很奇怪的参数，即 __elgg_ts 和 __elgg_token。这两个参数用于此防护措施，如果它们的值不正确，Elgg 则不会接受该请求。我们在本实验中关闭了这种防御措施，因此在伪造请求时无需包括这两个参数。

重要说明： 如果您使用的是云虚拟机或非 SEED 的其他虚拟机，您的 VM 中的 Firefox 版本可能较新，并且可能会对跨站 cookie 采取一些限制。在较新的版本中，使用 img 标签来伪造 HTTP GET 请求将无法使攻击成功。这是因为浏览器对这类请求会阻止跨站 cookie。如果您遇到这种情况，您可以选

择以下方法之一使攻击生效：

- 不使用 `img` 标签，而是使用其他方式生成 GET 请求。例如，在下一个任务中我们使用 JavaScript 来伪造 POST 请求。我们可以使用相同的方法来伪造 GET 请求。
- 更改 Firefox 设置以取消这些限制。学生需要找出应该取消哪个限制条款。该限制设置在 “隐私与安全” 页面上，可以通过在 URL 场景中输入 `about:preferences#privacy` 来查看。

如果您使用的是我们提供的 SEEDUbuntu 20.04 虚拟机，则此方法仍然有效，除非您已将 Firefox 升级到较新的版本。

3.3 任务 3：使用 POST 请求的 CSRF 攻击

在成功地将自己添加到 Alice 的好友列表后，Samy 想要做更多的事情。他希望 Alice 在她的个人资料中写着：“Samy 是我的英雄”。Alice 并不喜欢 Samy，更不用提把这句话放到她的个人资料中了。Samy 计划使用 CSRF 攻击来实现这个目标，这也是本任务的目的。

一种攻击的方法是给 Alice 的 `Elgg` 帐户发一条消息发，希望她点击该消息中的 URL。这个 URL 会引导 Alice 来到 Samy（即您）的恶意网站 `www.attacker32.com`，在那里您可以发动 CSRF 攻击。

您的攻击目标是修改受害者的个人资料。具体来说，攻击者需要伪造一个请求来修改 Elgg 中受害用户的个人资料信息。允许用户自行修改个人资料是 Elgg 的功能之一。如果用户想要修改自己的个人资料，他们可以访问 Elgg 的个人资料页面，填写表格，并提交表单到服务器脚本中，也就是发送一个 POST 请求到 `/profile/edit.php`，该脚本会处理请求并执行个人资料的修改。

服务器端脚本 `edit.php` 同时接受 GET 和 POST 请求，因此您可以使用与任务 1 中相同的方法来实现攻击。然而，在这个任务中，您需要使用 POST 请求。即，当受害者访问其恶意站点时，攻击者（也就是您）需要伪造一个来自受害者浏览器的 HTTP POST 请求。攻击者需要了解此类请求的结构。通过对个人资料进行一些修改并使用 `"HTTP Header Live"` 工具来监控请求，您可以观察到和下面类似的请求。与 GET 请求将参数附加在 URL 字符串中不同，POST 请求中的参数包含在 HTTP 消息体中（见两个 ✰ 符号之间的内容）：

```
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
......                                                          ☆
```

理解了请求的结构之后，您需要在攻击页面里生成这样的请求。为了帮助您编写此类 JavaScript 程序，我们提供了一个示例代码。您可以使用此示例代码来构建 CSRF 攻击，但这只是个示例代码，您需要对其进行修改以使其适用于您的攻击。

```
<html>
<body>
<h1>此页面伪造一个 HTTP POST 请求。</h1>
<script type="text/javascript">

function forge_post()
{
    var fields;

    // 下面是攻击者需要填写的表单输入项。这些条目是隐藏的，受害者看不到。
    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='****'>";

    // 创建一个 <form> 元素。
    var p = document.createElement("form");

    // 构造表单
    p.action = "http://www.example.com";
    p.innerHTML = fields;
    p.method = "post";

    // 将表单添加到当前页面中。
    document.body.appendChild(p);

    // 提交表单
    p.submit();
}


// 页面加载后会调用 forge_post() 函数。
window.onload = function() { forge_post();}
</script>
</body>
</html>
```

在 Line ①，值 2 将字段的访问级别设置为公开，这是必须的，否则默认情况下该字段将被设为私有，其他用户则无法看到此字段。需要注意的是，在从 PDF 文件中复制和粘贴上述代码时，程序中的单引号字符可能变成了其他符号（但仍然看起来像单引号）。这会导致语法错误。用您的键盘输入的单引号替换所有这些符号将解决这些问题。

问题： 除了详细描述您的攻击外，您还需要在报告中回答以下问题：

- **问题 1**：伪造的 HTTP 请求需要 Alice 的用户 ID（guid）才能正常工作。如果 Boby 特别针对 Alice 发动攻击，在发动攻击前他可以想办法来获取 Alice 的用户 ID。Boby 不知道 Alice 在 Elgg 中的密码，因此无法登录到 Alice 的帐户以获取相关信息。请描述 Boby 如何解决这个问题。
- **问题 2：**如果 Boby 想要修改访问他的恶意网页的任何人的 Elgg 个人资料，在事先并不知道谁会访问的情况下还可以发起 CSRF 攻击吗？请解释。

4 防御任务

CSRF 不难防御。最初，大多数应用程序都会在他们的网页中嵌入一个秘密令牌，并通过检查请求中是否存在该令牌来判断请求是同源请求还是跨站请求。这称为 “秘密令牌” 方法。最近，大多数浏览器都实现了一种名为 “同源 cookie” 的机制，旨在简化 CSRF 防御措施的实现。我们将对这两种方法进行实验。

4.1 任务 4：启用 Elgg 的防护措施

为了防御 CSRF 攻击，Web 应用程序可以将秘密令牌嵌入到页面中。所有来自这些页面的请求都必须携带此令牌，否则会被视为跨站请求，不会具有和同源请求相同的权限。攻击者无法获取此秘密令牌，因此他们的请求很容易被识别为跨站请求。

Elgg 使用这种秘密令牌方法作为其内置的防护措施来防御 CSRF 攻击。我们已经关闭了这些防护措施以使攻击生效。Elgg 在请求中嵌入了两个参数 `__elgg_ts` 和 `__elgg_token`。服务器在处理请求前会验证它们。

将秘密令牌和时间戳嵌入到网页中： Elgg 在所有需要用户操作的地方都添加了安全令牌和时间戳。以下 HTML 代码出现在所有的表里。这两个隐藏字段会在表单提交时被附加到请求中：

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

Elgg 还将安全令牌和时间戳的值赋给 JavaScript 变量，以便页面上的 JavaScript 代码可以轻松访问这些变量。

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

在 Elgg 的网页中添加安全令牌和时间戳是通过 `vendor/elgg/elgg/views/default/input/securitytoken.php` 来实现的。下面的代码片段展示了这些内容是如何动态地添加到页面上的。

```
$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]);
```

秘密令牌的生成。 Elgg 的安全令牌是下面信息产生的哈希值：网站提供的秘密值，时间戳、用户会话 ID 和随机生成的会话字符串。下面的代码展示了 Elgg 中的安全令牌的生成过程（在 vendor/elgg/elgg/engine/classes/Elgg/Security/Csrf.php 中）。

```
/**
 * 从会话令牌、时间戳和站点密钥生成一个令牌。
 */
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();
}
```

秘密令牌验证。 Elgg 网站会验证生成的令牌和时间戳以防御 CSRF 攻击。每当用户执行某个操作时，都会调用 Csrf.php 中的 validate() 函数，此函数会对这些令牌进行验证。如果令牌不存在或无效，则将拒绝该操作并将用户重定向到其他页面。我们在该函数开始添加了一个 return 语句，从而禁用了这一功能。

```
public function validate(Request $request) {
   return; // 为 SEED 实验（禁用 CSRF 防护措施）而添加

    $token = $request->getParam('__elgg_token');
    $ts = $request->getParam('__elgg_ts');
    ... (省略了代码)
}
```

任务：开启防护措施。 为了启用防护措施，请进入 Elgg 容器，前往 /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security 文件夹，从 Csrf.php 中删除 return 语句。容器内有一个简单的编辑器叫做 nano。在完成更改后，请重新运行攻击，并观察您的攻击是否成功。请指出捕获的 HTTP

请求中的秘密令牌。请解释为什么攻击者无法在 CSRF 攻击中发送这些秘密令牌；是什么阻止了他们从网页中找到这些秘密令牌?

需要注意的是（非常重要），当我们启用防护措施后进行 CSRF 攻击去修改个人资料时，攻击失败后攻击者的页面会被重新载入，这将再次触发伪造的 POST 请求。这会导致另一个失败的攻击，然后页面将继续被重新加载并发送出另一个伪造的 POST 请求。这种无限循环可能会导致您的计算机变慢。因此，在确认攻击失败后，请关闭页面以停止无限循环。

4.2 任务 5：实验同源 cookie 方法

大多数浏览器现在都实现了一种称为 “同源 cookie” 的机制，这是与 cookie 相关的一个属性。当发送请求时，浏览器会检查这个属性，并决定是否在跨站请求中附加此 cookie。如果一个 Web 应用程序不希望某个 cookie 用于跨站请求，他们可以标记该 cookie 为同源。例如应用程序可以把 session ID cookie 设为同源 cookie，从而阻止任何跨站请求使用 session ID，因此无法发动 CSRF 攻击。

为了帮助学生了解如何利用同源 cookie 来防御 CSRF 攻击，我们在一个容器中创建了一个名为 `www.example32.com` 的网站。请访问以下 URL。我们已将主机名映射到 `10.9.0.5`。

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

一旦您访问过这个网站,您的浏览器将设置三个 cookie：`cookie-normal`, `cookie-lax` 和 `cookie-strict`。如名称所示，第一个 cookie 是一个普通的 cookie；第二个和第三个是两种类型的同源 cookie（Lax 类型和 Strict 类型）。我们设计了两组实验来测试哪些 cookie 会被发送给服务器。

请点击网页中的链接。链接 A 指向 `example32.com` 的页面，而链接 B 则指向 `attacker32.com` 的页面。这两个页面几乎是相同的（除了背景颜色），并且它们都发送了三种不同类型的请求到 `www.example32.com/showcookies.php`，该页面仅仅显示浏览器发送的 cookie。通过查看显示结果，您可以知道哪些 cookie 被发送。请完成以下任务：

- 请描述您看到的内容并解释在某些场景下为何有些 cookie 不会被发送。
- 根据您的理解，请描述同源 cookie 如何帮助服务器检测请求是跨站还是同源的。
- 请描述如何使用同源 cookie 机制来帮助 Elgg 防御 CSRF 攻击。您只需描述一些基本想法，无需实现它们。

额外加分： 尽管这不是强制要求的，我们鼓励学生修改 `Elgg` 应用程序以利用同源 cookie 机制来防御 CSRF 攻击。建议讲师给予那些成功完成此任务的学生额外加分。学生应该与他们的导师讨论额外得分事宜。

5 指导方针

5.1 使用 HTTP Header Live 插件检查 HTTP 头

在我们的 Ubuntu 16.04 虚拟机中，Firefox（版本 60）不支持之前的 LiveHTTPHeader 插件。取而代之的是，我们使用了一个名为 HTTP Header Live 的新插件。启用和使用此工具的方法如图1所示，

只需点击标记为 的图标，在左侧会弹出侧边栏。确保在位置 选择 HTTP Header Live。然后点击网页中的任何链接，触发的所有 HTTP 请求都将被捕获并在侧边栏区域（标号 ）中显示。如果单击任何 HTTP 请求，将弹出一个窗口以显示所选的 HTTP 请求。不幸的是，在该插件中仍存在一个缺陷；除非改变窗口大小，否则弹出窗口内没有任何内容显示。似乎在窗口弹出时不会自动重绘，但更改其大小会触发重绘。

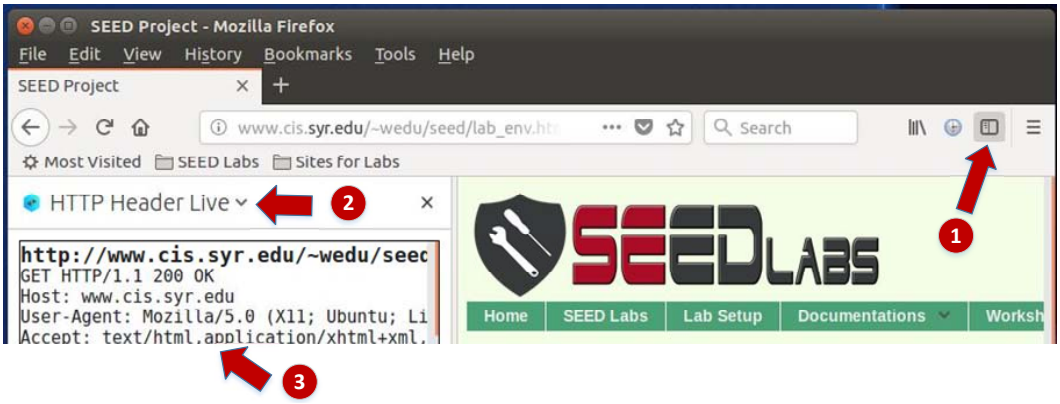


图 1: 启用 HTTP Header Live 插件

5.2 使用 Web 开发者工具检查 HTTP 头

Firefox 还提供了一个非常有用的工具，可以用来检查 HTTP 头。这个工具是 Web Developer Network Tool。在本节中，我们将介绍该工具的一些重要功能。启用此网络工具的方法如下：

```
点击 Firefox 右上角菜单 --> Web Developer --> Network
或
点击 Tools 菜单 --> Web Developer --> Network
```

以 Elgg 的用户登录页面为例，图2显示了网络工具中用于登录的 HTTP POST 请求。

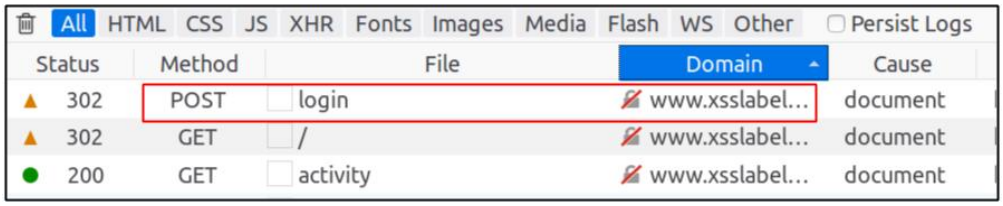


图 2: Web 开发者网络工具中的 HTTP 请求

要查看更多详细信息，可以点击特定的 HTTP 请求，此时该工具会在两个分栏中显示相关信息（见图3）。

所选请求的详细信息将显示在右侧分栏中。图4(a)展示了登录请求在 Headers 标签页中的详细信息（包括 URL、请求方法和 cookie）。可以观察到右分栏中包含的请求头和响应头。要检查 HTTP 请求

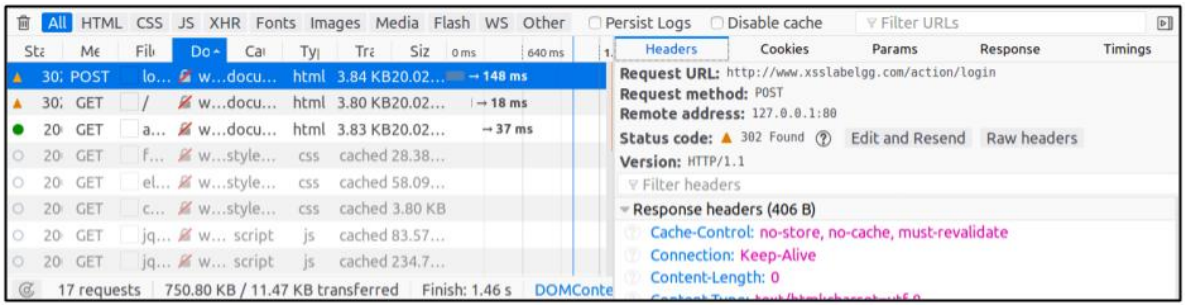
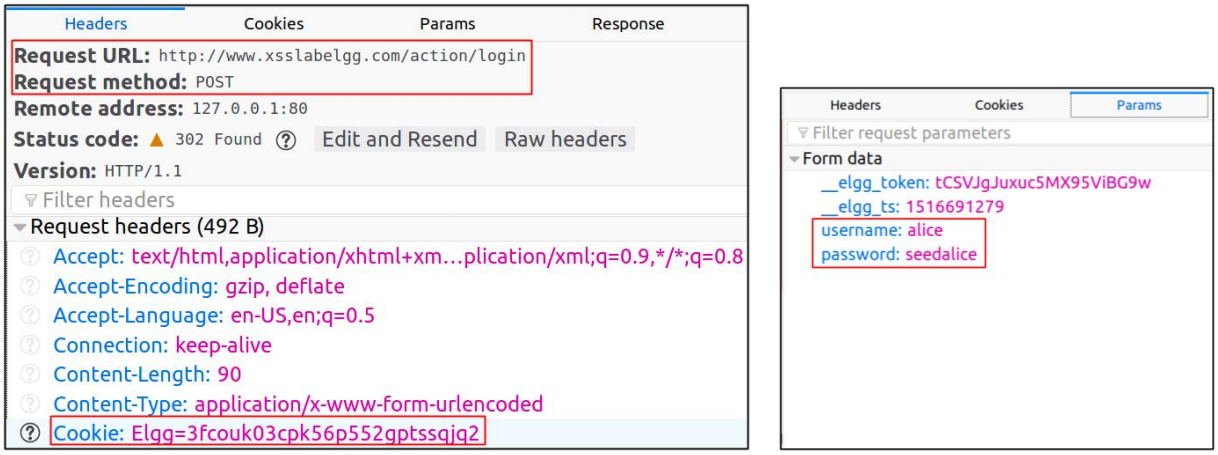


图 3: 分栏中的 HTTP 请求和详细信息

中的参数，可以使用 Params 标签页。图4(b)展示了登录请求发送给 Elgg 的参数，其中包括 username 和 password。该工具也可以像处理 HTTP POST 请求一样来检查 HTTP GET 请求。



(a) HTTP 请求头　　(b) HTTP 请求参数

图 4: HTTP 头和参数

字体大小。 Web 开发者工具窗口的默认字体大小较小，可以通过在网络工具窗口中任意位置点击一次，然后使用 `Ctrl +` 键来增大。

5.3 JavaScript 调试

我们可能还需要调试我们的 JavaScript 代码。Firefox 的开发者工具也可以帮助调试 JavaScript 代码。它可以指出错误发生的具体位置。下面的指令显示了如何启用此调试工具：

```
点击 Tools 菜单 --> Web Developer --> Web Console
或使用 Shift+Ctrl+K 快捷键。
```

进入控制台（Console）后，点击 JS 标签页。点击 JS 旁边的下箭头并确保在 Error 旁边有一个对勾。如果你对警告消息还感兴趣的话，则可以单击 Warning（见图5）。

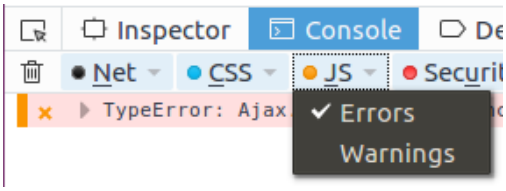


图 5: 调试 JavaScript 代码（1）

如果代码中出现任何错误，控制台将显示一条消息。导致错误的行号出现在错误消息右侧，点击行号即可跳转到具体出错位置（见图6）。

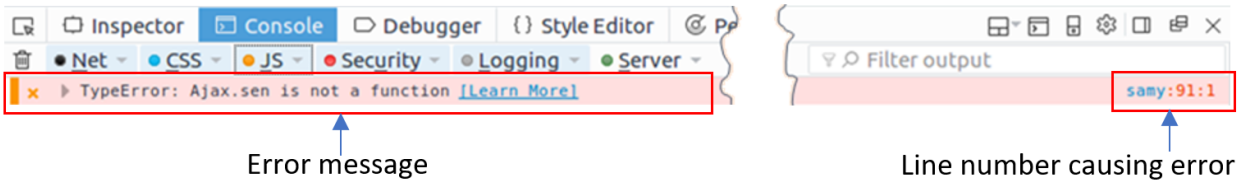


图 6: 调试 JavaScript 代码（2）

6 提交

你需要提交一份带有截图的详细实验报告来描述你所做的工作和你观察到的现象。你还需要对一些有趣或令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。只是简单地附上代码不加以解释不会获得学分。